# Report Back of Session III

# Semantics, Abstractions, Attributes and Meanings

- Reoccurring theme all week…
- Up and down the stack (app down, hardware up)

# Memory models, name space, and address space

- Very much an execution time decision.
  - Could be both static and dynamic decisions/determination.
- Namespace and "**useful**" abstractions to be controlled at application level
- **Composition** between namespace and persistent data – same for abstractions, priorities differ depending upon use -- mapping between different application pieces (**multi-physics, multi-scale, in situ vis + analysis, libraries, etc.**)
- Who virtualizes resources (memory/storage) & at what level
  - **Cost models**, etc. Challenges in terms of cost determination and performance/platform portability.
- Implied set of operations per application-level model (transformations/layout for different stages differ)
  - Explicit data layout awareness/selection
  - Need an effective abstraction from data layout to iteration space
- **Workflow** – intent needs to be communicated to runtime (semantic awareness – e.g. understand the use of data)

# Introspection interfaces, policies, and control

- "**Understandability** is the path to believability"
- Information must be available at all levels – **policy must be consistent**
  - Common set of abstractions important for communications/understandability
  - Research in terms of where optimization can be applied (understand feedback data)
  - Have to consider impact of multiple runtimes and interoperability/coordination (composition – portability impacts)
  - Finding building blocks and understand costs/benefits of use (what needs to be integrated vs. what doesn't)
- **Feedback** throughout toolchain and runtime(s) and OS and hardware.
  - Including facility data (power, temperature, etc.)
  - Low/no-impact way to gather data from hardware resources

# Contribution to tools

- Understand why/when/where adaptive choices are made – non-determinism makes debugging hard (need control at a cost)
  - **Integration with data analysis and visualization infrastructure**
  - **Composition** issues for effective/use utilization of system
  - Infrastructure to support **isolation and sharing**
- Tools must meet set of requirements that allow better interoperability with the application
  - Attribution is a difficult problem
  - What abstraction do you use present to the developer? **Want a choice…**
  - **How do you deal with/understand the complexity and size of the resulting information**
  - **Steer performance analysis (in situ problem)**

# Parallelism forms, granularity, and synchronization

- **“Yes…”**
- **Want full range of implicit to explicit parallelism support**
- Need to express dependencies – more freedom from lower layers to expose additional parallelism
  - This comes at a cost of additional overhead in terms of dependency management (aggregation), memory, extra overheads --- hints from application (higher) layers helpful

# Contribution and responsibility to reliability

- Runtime OS provides lower-level capabilities to enable resilience mechanisms
  - **Where to store persistent/recovery data**. Attributes about what is suitable and what should be saved.
  - Must be **recoverable/guaranteed**!
  - Reliability attributes (costs for different approaches – e.g. price, overheads, etc. **how do we understand and reason about costs**?)
- DAGs give finer grained unit of recovery
  - With more complicated state recovery (vs. bulk sync) – details implementation dependent
- **Subjected to the same faults** as the applications (how do we recover?)
- **Attribution & abstraction** about events up the stack
- Networking hardware state is unavailable/inaccessible (shouldn't be)
- **Expression** of what techniques should be used to mitigate error conditions
- In some cases the correct thing to do is just fail – don't over engineer (balance function/capability vs. impact on code base)

# Contribution and responsibility to energy/power management

- Optimizing under power and energy constraints – power is the first constraint
- Runtime needs to know power budget is and what it is using
  - Per system, node, etc… How do we reason about the **critical path**?
- Attribution & layers for communication
  - Two way street. Likely not controlled at the app level but could be hinted at