A Task-Based Programming Model for Scientific Computing

Ben Bergen
Applied Computer Science (CCS-7)
Los Alamos National Laboratory

LA-UR-15-20773

Los Alamos NATIONAL LABORATORY EST. 1943
Operated by Los Alamos National Security, LLC for DOE/NNSA
NNSA

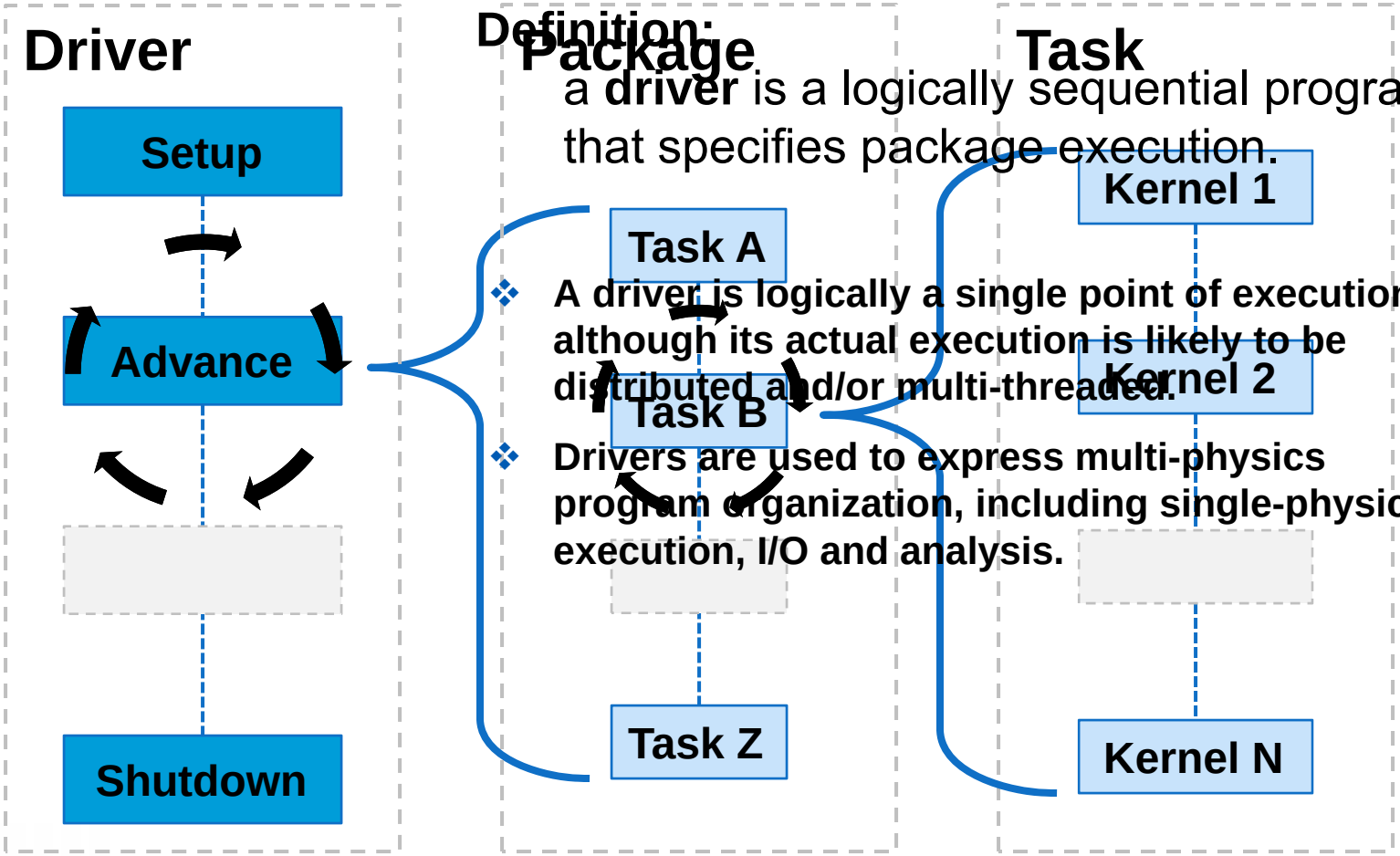
Principles

- **Degree of parallelism is algorithmically bound**
  - Model should expose the maximum level of algorithmic parallelism
- **Data layout artifacts implementation, often exposing hardware specific optimization strategies**
  - Model should defer decisions about data layout
- **Data placement exposes specific details of the memory hierarchy and file system**
  - Model should defer decisions about data placement

Execution Model

Operated by Los Alamos National Security, LLC for NNSA

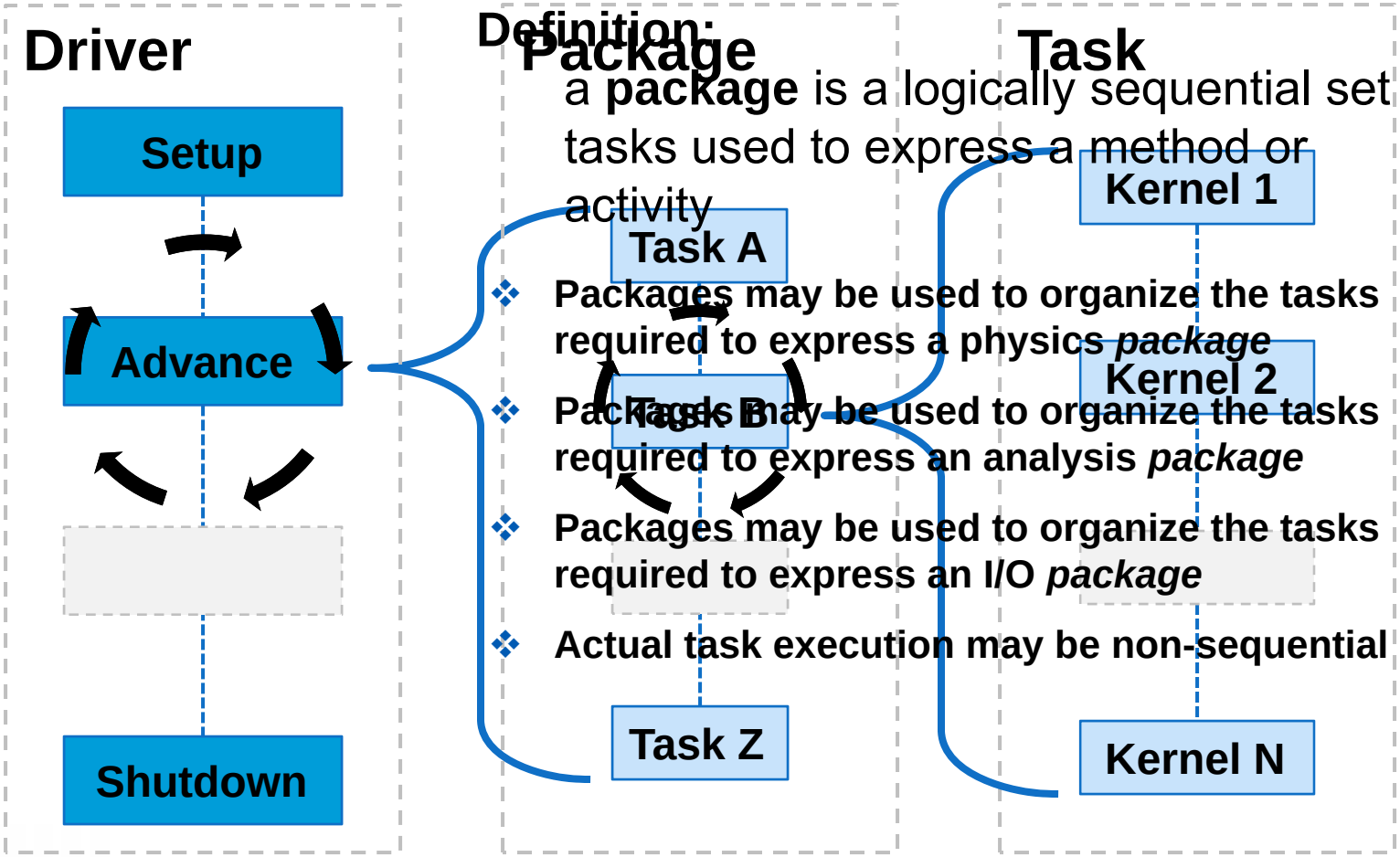
Execution Model: High-Level View



Definition:
a **driver** is a logically sequential program that specifies package execution.

- **A driver is logically a single point of execution, although its actual execution is likely to be distributed and/or multi-threaded.**
- **Drivers are used to express multi-physics program organization, including single-physics execution, I/O and analysis.**

Execution Model: High-Level View

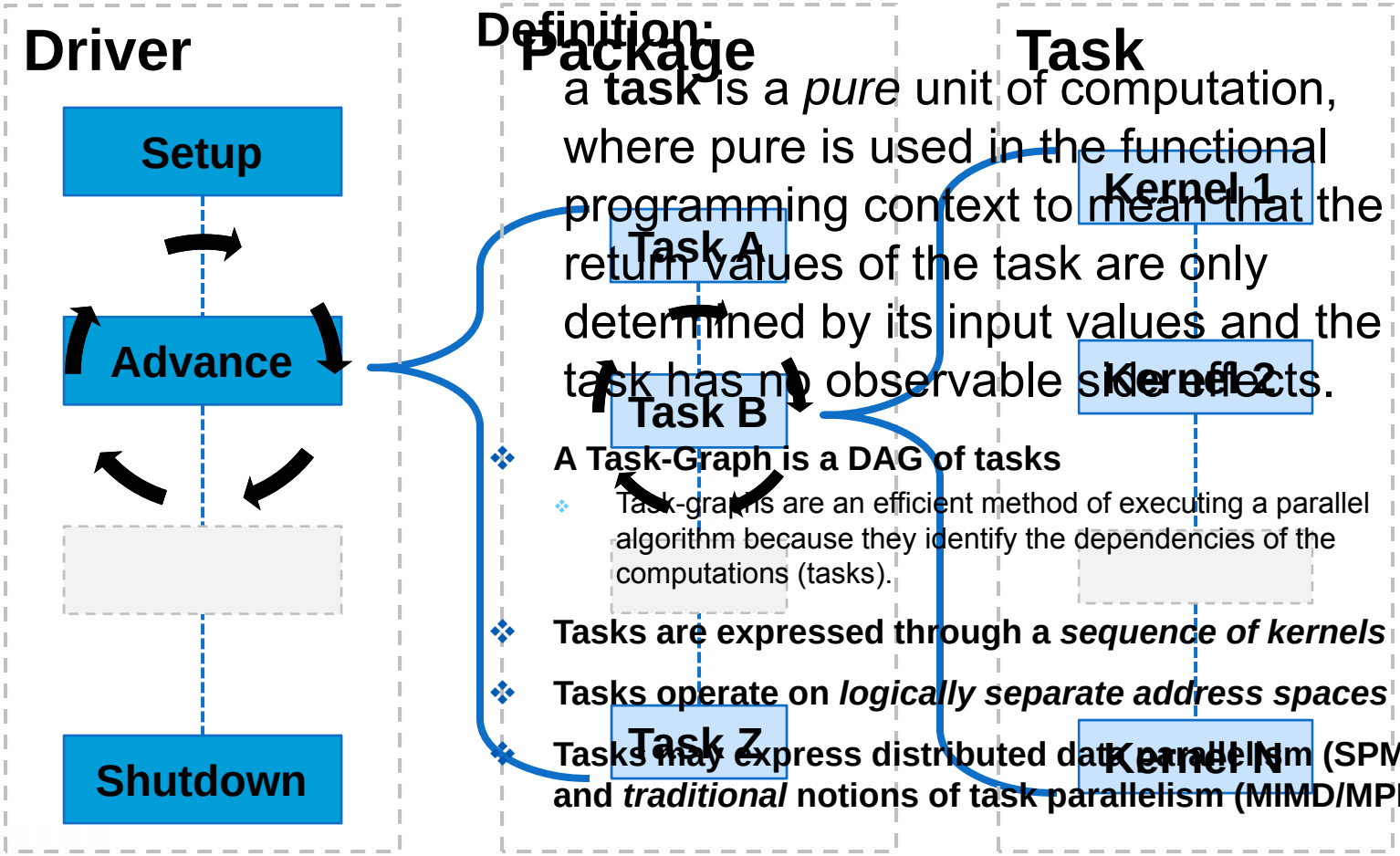


Definition:

a **package** is a logically sequential set of tasks used to express a method or activity

- **Packages may be used to organize the tasks required to express a physics *package***
- **Packages may be used to organize the tasks required to express an analysis *package***
- **Packages may be used to organize the tasks required to express an I/O *package***
- **Actual task execution may be non-sequential**

Execution Model: High-Level View



Definition:

a **task** is a *pure* unit of computation, where pure is used in the functional programming context to mean that the return values of the task are only determined by its input values and the task has no observable side effects.

- **A Task-Graph is a DAG of tasks**
  - Task-graphs are an efficient method of executing a parallel algorithm because they identify the dependencies of the computations (tasks).
- **Tasks are expressed through a *sequence of kernels***
- **Tasks operate on *logically separate address spaces***
- **Tasks may express distributed data parallelism (SPMD) and *traditional* notions of task parallelism (MIMD/MPMD)**

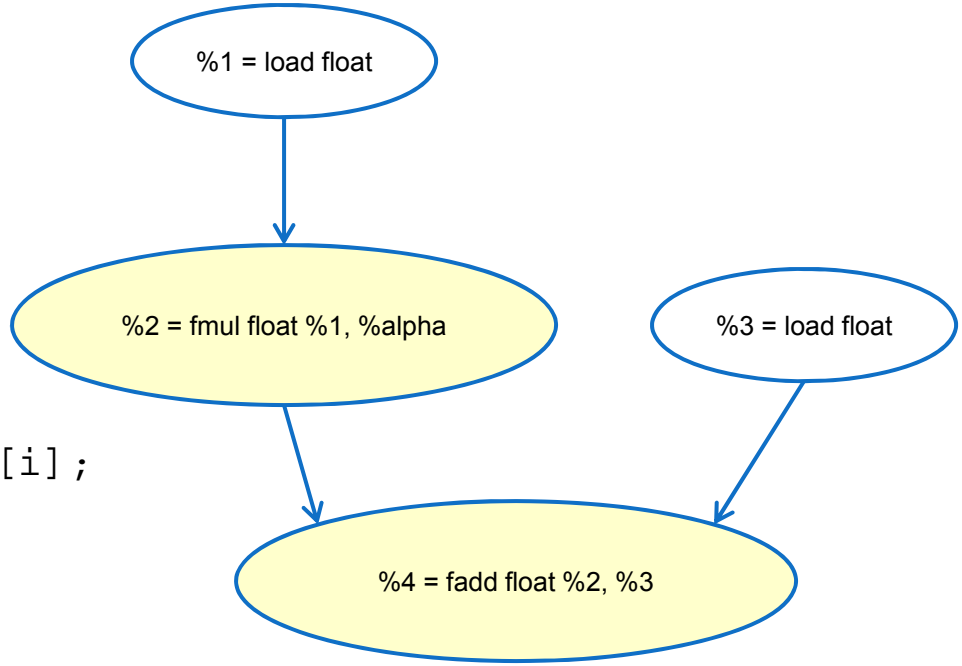
Directed Acyclic Graph (DAG)

Definition:

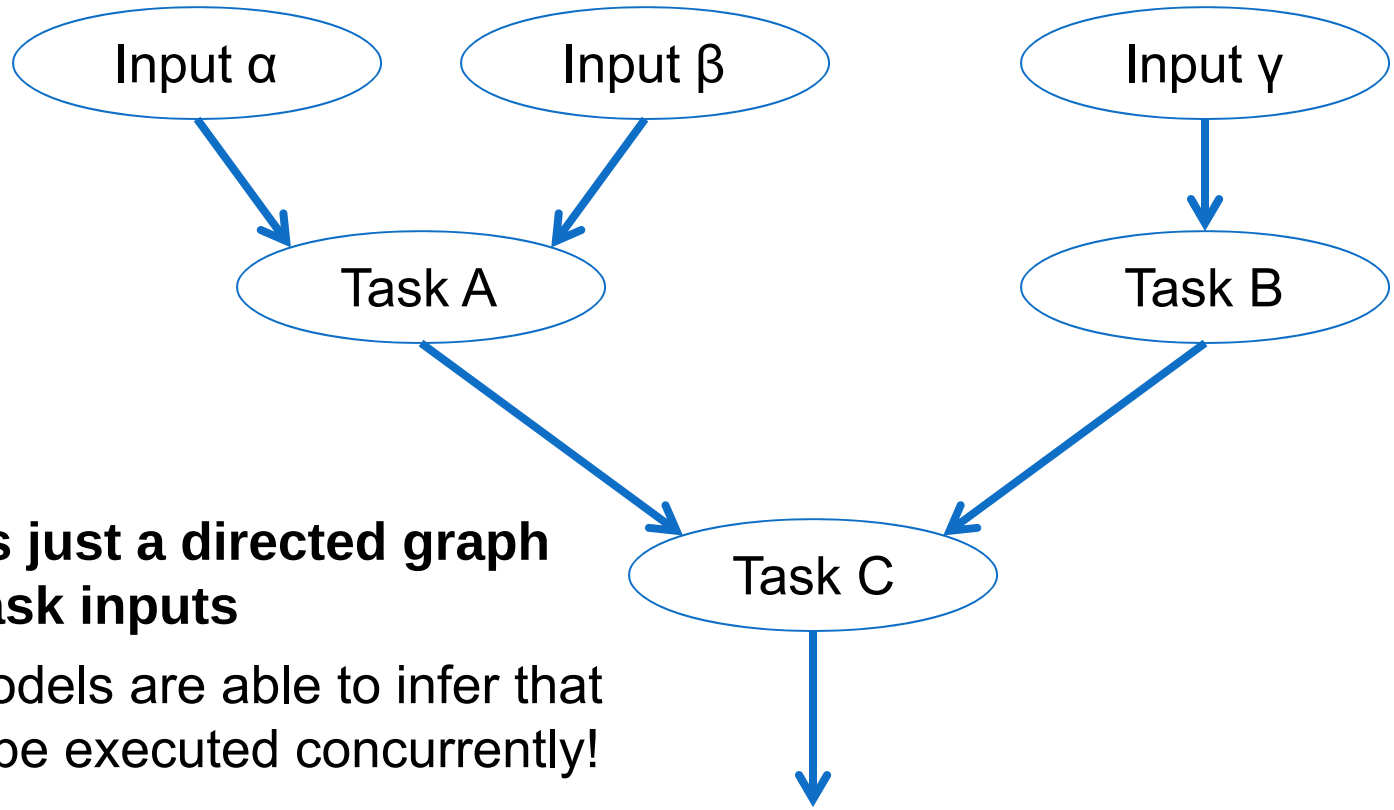
a **DAG** is a directed graph with no directed cycles, i.e., starting from any node in the graph it is not possible to follow a sequence of edges that loops back to the original node.

DAG representation of saxby basic block:

```
for(i=0; i<N, ++i) {
   y[i] = alpha*x[i] + y[i];
}
```



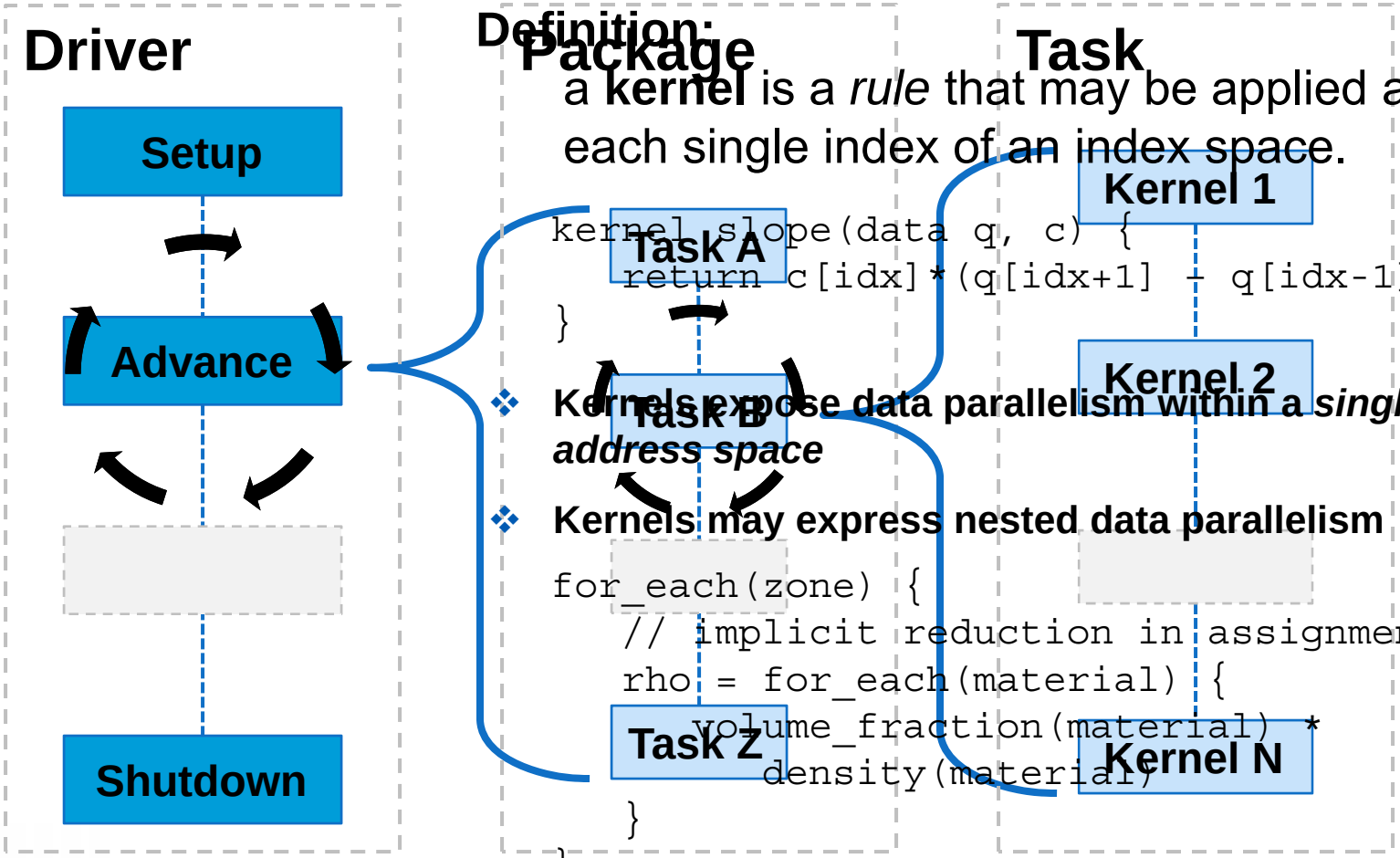
Task-Graph Models



A *task graph* is just a directed graph of tasks and task inputs

Task-graph models are able to infer that **A** and **B** may be executed concurrently!

Execution Model: High-Level View



Definition:
a **kernel** is a *rule* that may be applied at each single index of an index space.

```
kernel slope(data q, c) {
    return c[idx]*(q[idx+1] - q[idx-1])
}
```

- **Kernels expose data parallelism within a *single address space***
- **Kernels may express nested data parallelism**

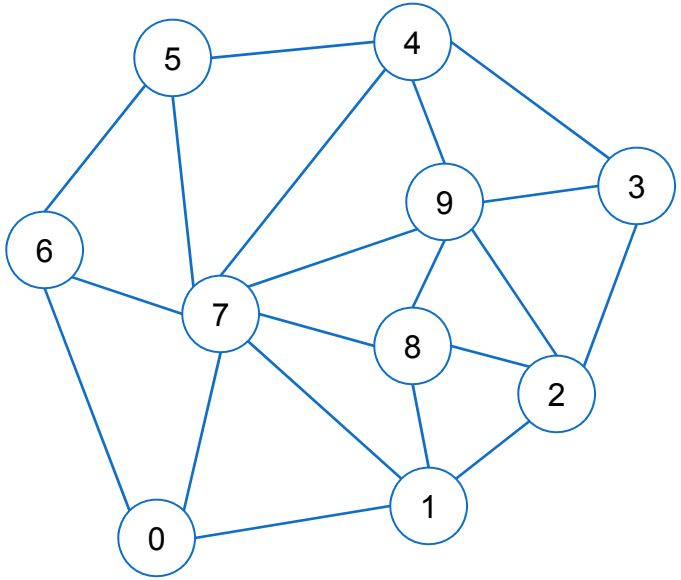
```
for_each(zone) {
    // implicit reduction in assignment
    rho = for_each(material) {
        volume_fraction(material) *
            density(material)
    }
}
```

Index Space

Definition:

an **index space** is an unordered set of unique identifiers that enumerate the items of an execution topology

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4



{0,1, 2, 3, …, 22, 23, 24}

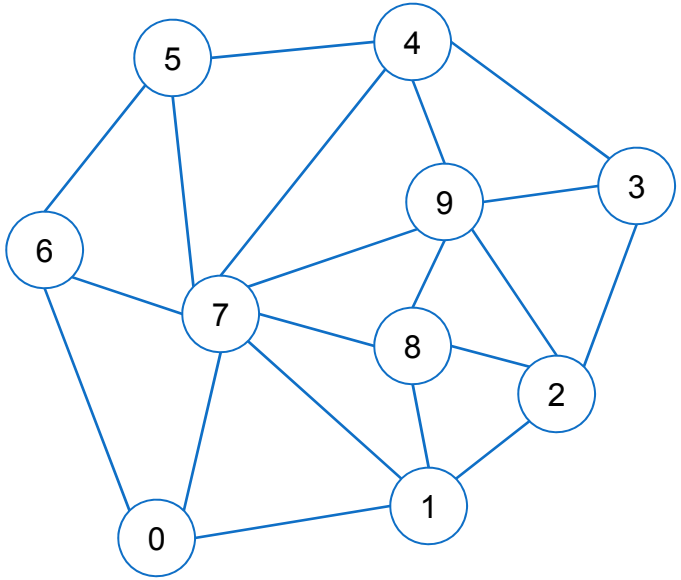
{0,1, 2, 3, 4, 5, 6, 7, 8, 9}

Index Space

Definition:

an **index space** is an unordered set of unique identifiers that enumerate the items of an execution topology

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4



{17,5, 22, 3, …, 21, 2, 18}

{6,1, 0, 9, 8, 5, 2, 7, 3, 4}

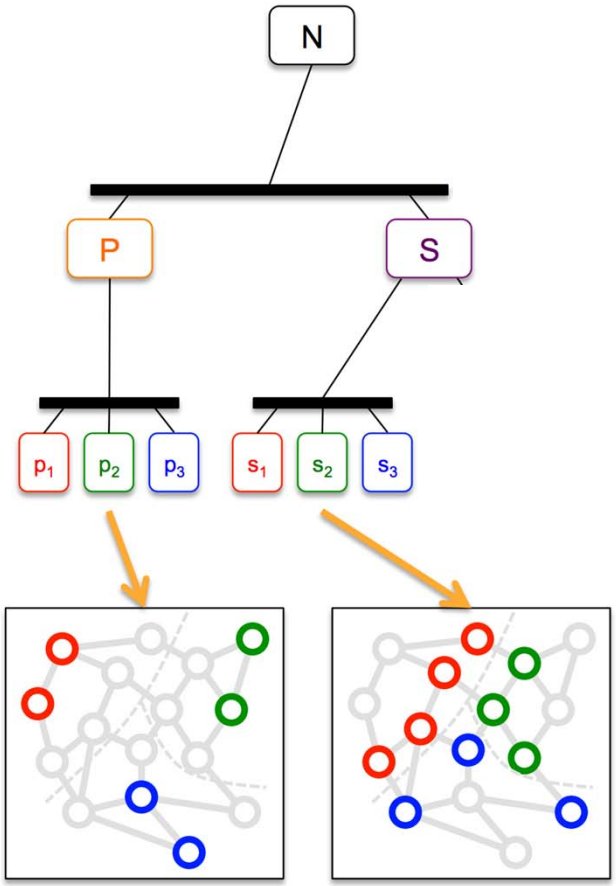
Data Model

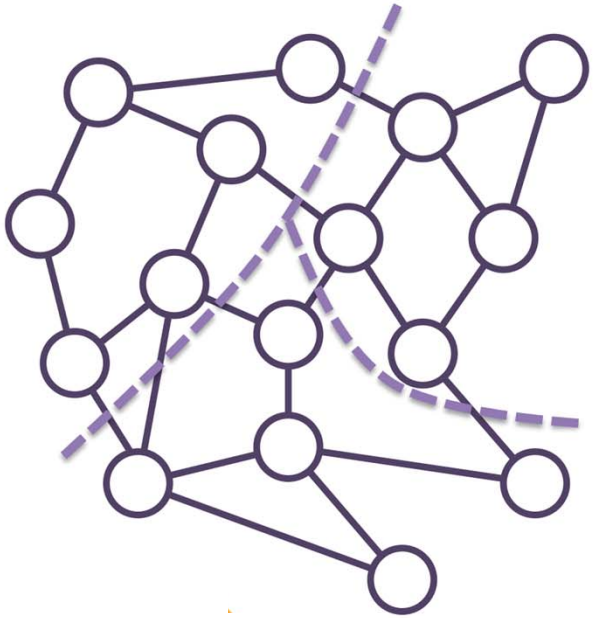
Key/Value Database

- **Data registration and partitioning**
- **Derived from Legion Data Model ➔ Regions**
- **Logical Regions**
  - A logical region is a collection of elements, each of which has stored data in one or more fields
  - Logical Regions are *logical* because:
    - They are not fixed to a particular memory – they can migrate, or be shared or replicated
    - They do not have a fixed memory layout
- **Logical Regions can nested to created multiple data views (subregions)**
  - Region coloring allows data decomposition for distributed-memory type partitioning

Logical Regions cannot be accessed directly

Region Partitioning
(borrowed from Sean Treichler, Legion Boot Camp)





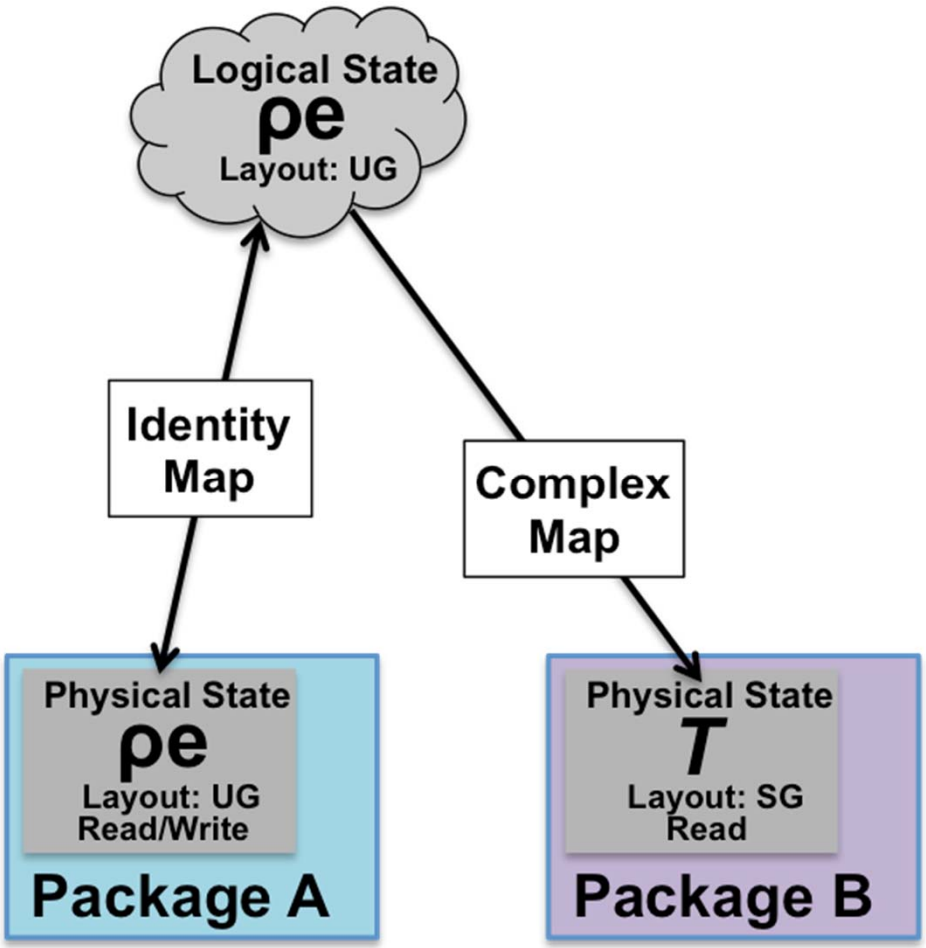
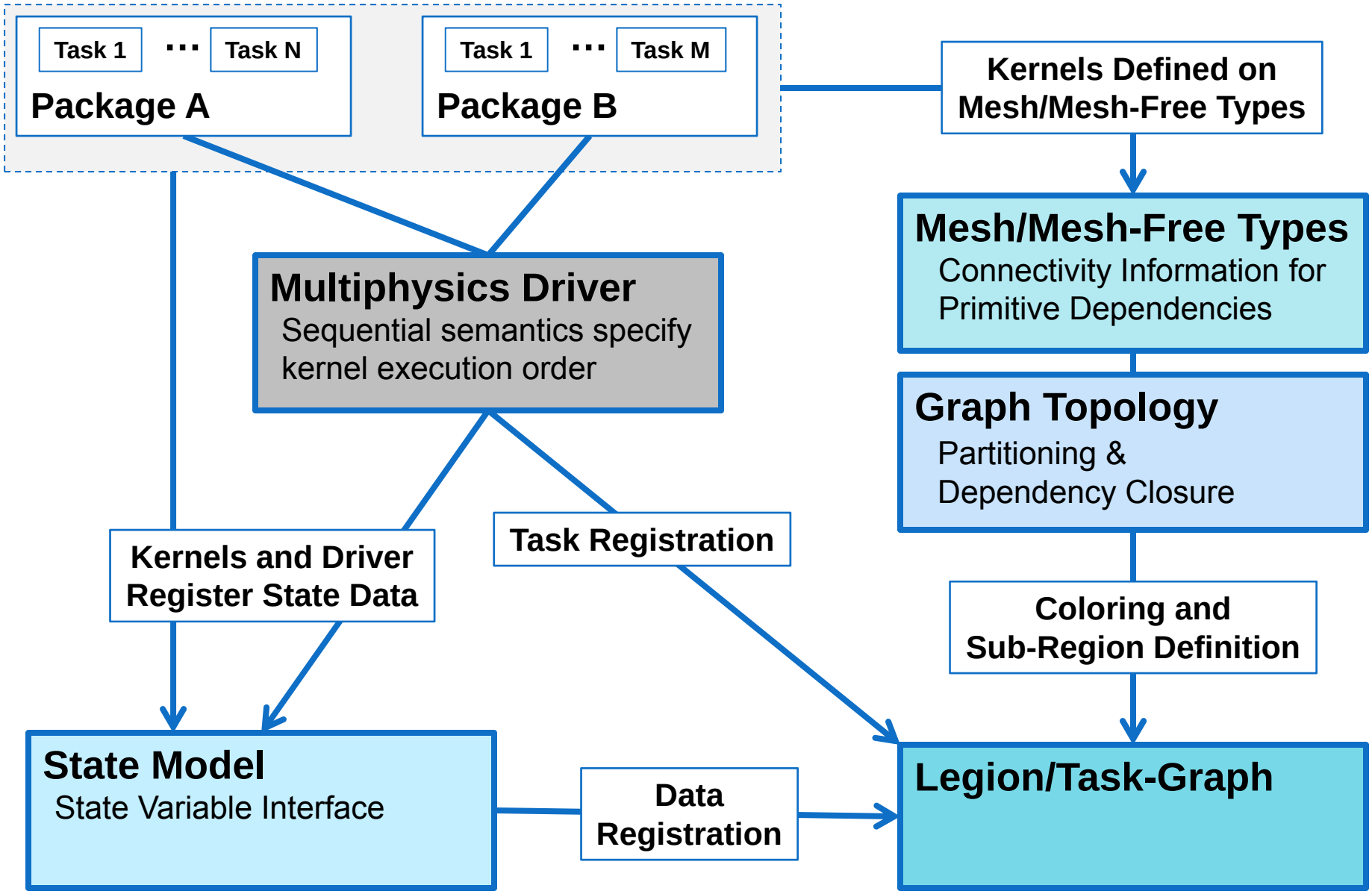
Physical Regions

- **A *physical region* allows access to data through a specific mapping**
  - Use a *region accessor* to access a specific field
- ***Region requirements* specify task-level data dependencies ➔ Allow DAG construction and execution**

State Model

- **Kernels and driver can register state data**
  - Publish/Subscribe model
  - Single database for all packages
- **Graph/Mesh topology layer**
  - Partitioning
  - Dependency closure
- **Logical vs. Physical State**
  - Registered state data are logical
  - Tasks access physical state through a mapping step

State Model


Logical State
ρe
Layout: UG
Identity Map
Complex Map
Physical State
ρe
Layout: UG
Read/Write
Package A
Physical State
T
Layout: SG
Read
Package B

How this might look for a multiphysics application…