A Discrete-Event Model of Computation

Joseph T. Buck

Brian L. Evans

Soonhoi Ha

Paul Haskell

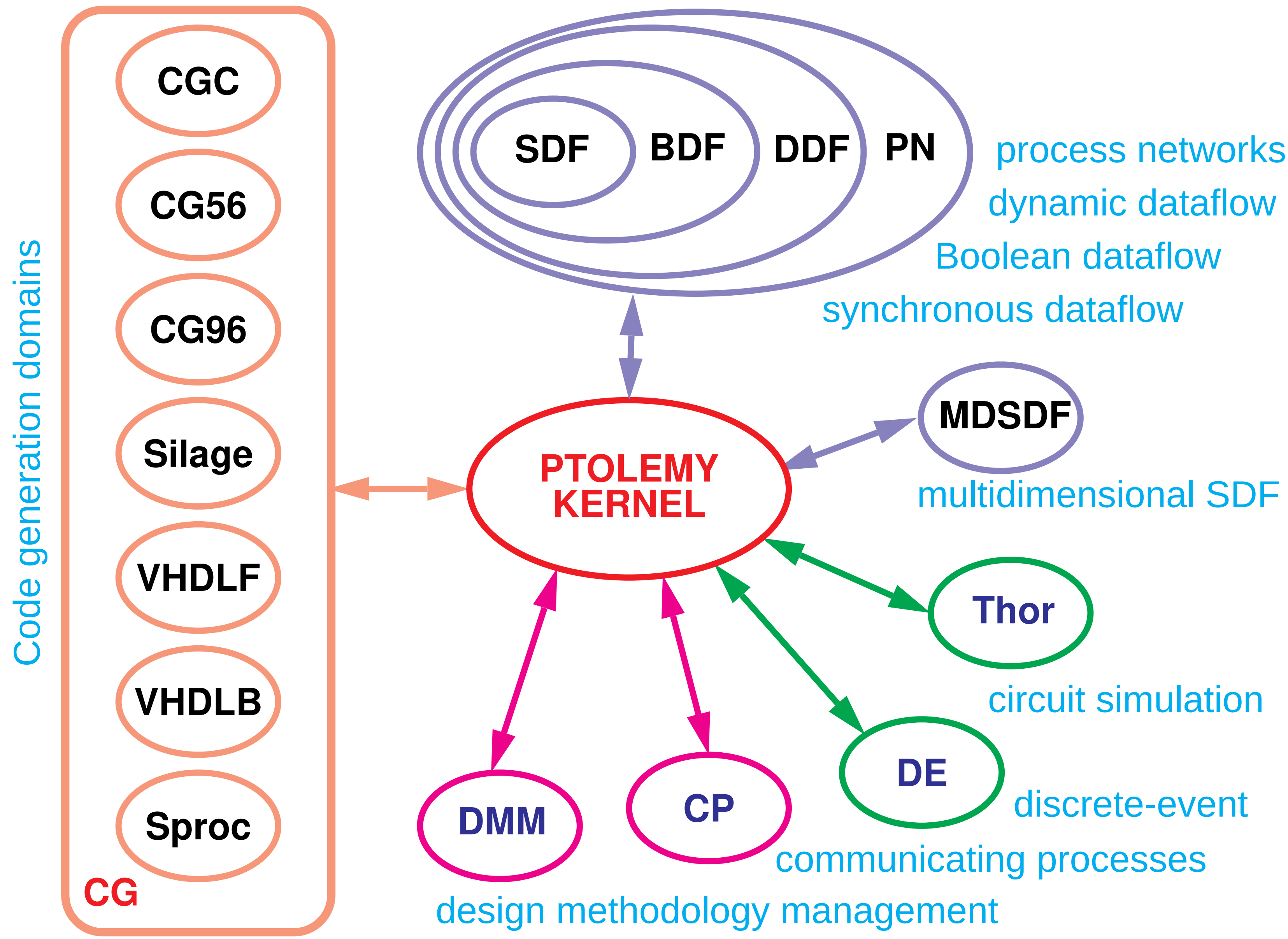
Edward A. Lee

Praveen K. Murthy

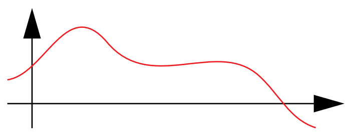
Thomas M. Parks

UC Berkeley

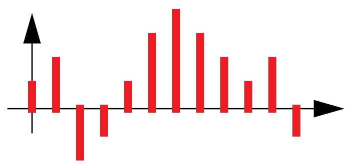
Domains in Ptolemy



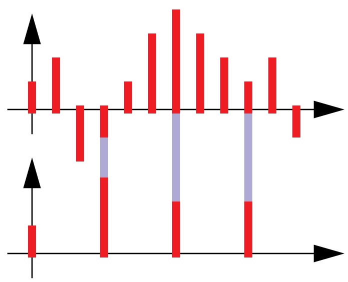
A Taxonomy of Discrete-Event Systems



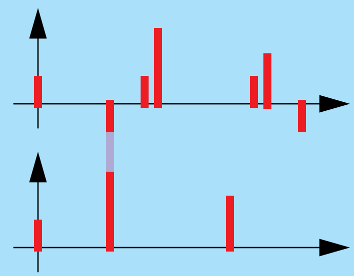
continuous time



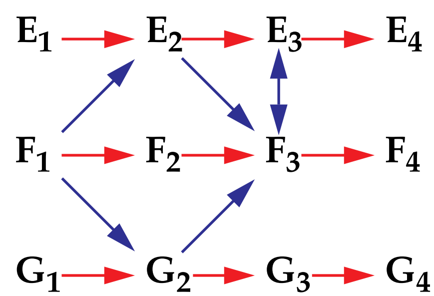
discrete time



multirate discrete time



totally-ordered discrete events
(the DE domain)



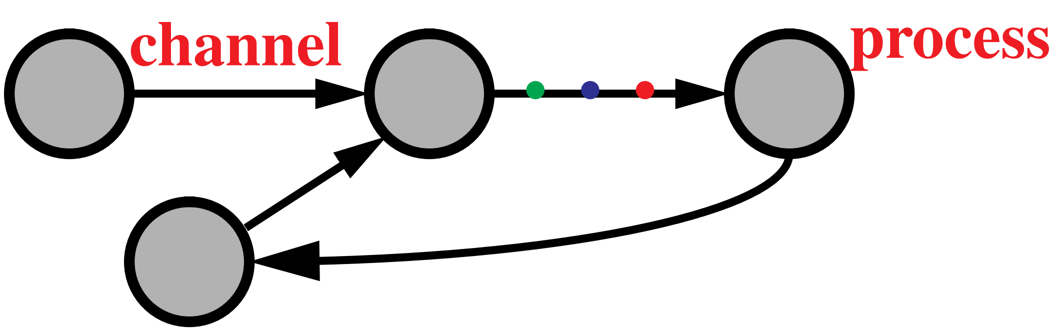
partially-ordered discrete events
(the dataflow domains)

Discrete-Event Model

Network of Processes Communicating Over Channels:

- processes are executed chronologically (ordered in time)
- processes communicate through one-way channels
- channels have one producer and one consumer
- writes to the channels are non-blocking, time stamped
- reads from channels are non-blocking
- data in channels are ordered by a global event queue

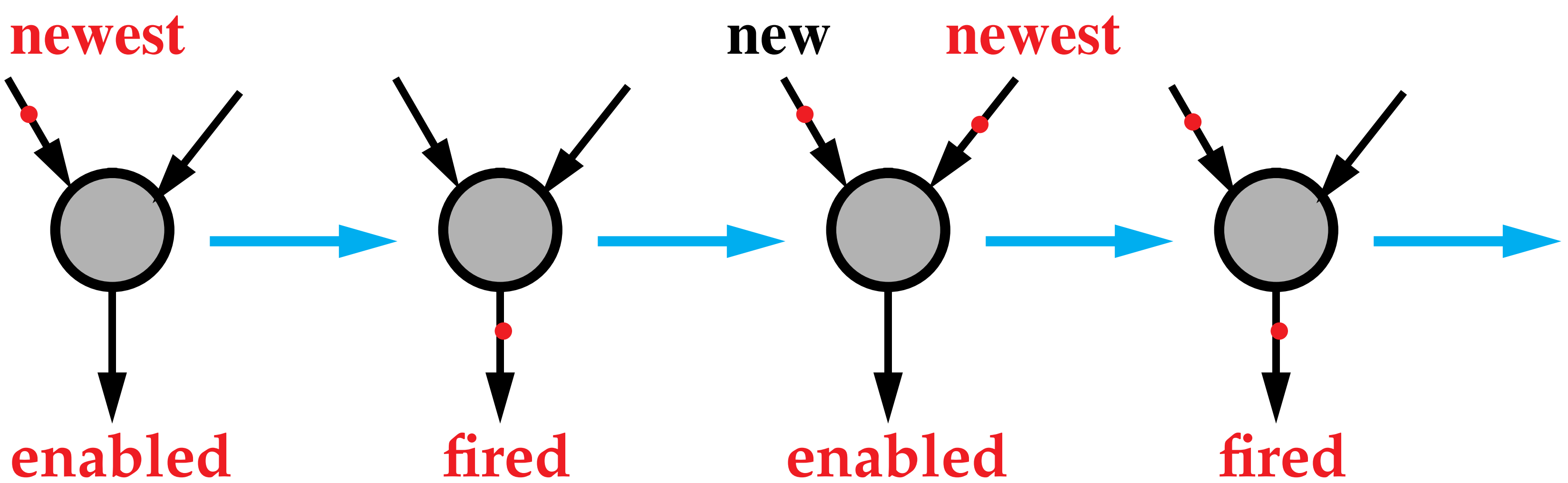
The chronological ordering is based on the timestamps of the data particles (events) in the channels.



Discrete-Event Domain

Discrete-Event Domain:

- A *signal* is a sequence of *discrete events* (data + timestamp).
- Discrete events are ordered in a global *queue* by timestamp.
- The global queue schedules which *actor* to execute.
- An actor *consumes* input tokens, *produces* output tokens.



Execution Model for the Discrete-Event Domain

Run-Time Scheduling

- Global event queue sorts events by their timestamps
- Scheduler fetches the event at the head of the queue and sends it to the input ports of its destination block

Special Cases

- Initialization: insert triggering particles in the event queue.
- Simultaneous events at all input ports to an actor are provided when the actor is executed
- Equal timestamps in the event queue are sorted according to a topological sort:
    1. partial ordering of the actors in the flow graph,
    2. triggering relationships between the input and output ports, and
    3. ordering of the input ports.

Discrete-Event Actors and Channels

Actors

- **Functional actors**
  timestamp of data on output ports =
  timestamp of data on triggering input ports
- **Delay actors**
  timestamp of data on output ports =
  timestamp of data on triggering input ports + *delay*

Properties of Channels

- **Delay attribute**
  represents an infinitesimal delay (sort of)
  introduces *no* actual delay in simulated time
  breaks topological ordering between connected actors
  prevents race conditions
  suppresses delay-free loop warnings

Delay Star Definition

```
defstar {
    name { Delay }
    domain {DE}
    desc { Delays its input by a fixed amount }
    input {
        name {input}
        type {anytype}
    }
    output {
        name {output}
        type {=input}
    }
    defstate {
        name {delay}
        type {float}
        default {"1.0"}
        desc { Amount of time delay. }
    }
```

inherits from DEStar class

Single-input, single-output block

in simulated time units

File is "DEDelay.pl" in $PTOLEMY/src/domains/de/stars

Continued on the next page

Delay Star Definition

```
constructor {
    delayType = TRUE;
}
```

Member of the DEStar class

The star is a delay star

arrivalTime (member of the DEStar class) is set by the scheduler
to the timestamp of the event(s) the triggered the firing of the star

Manipulating Timestamps

```
go {
    completionTime = arrivalTime + double(delay);
    Particle& pp = input.get();
    output.put(completionTime) = pp;
}
```

completionTime (member of DEStar class) records latest output time

More Sophisticated Delay Star (a Server)

Input event waits until a simulated resource becomes free to attend to it

Resource is free

```
go {
    if ( arrivalTime > completionTime )
        completionTime = arrivalTime + double(delay);
    else
        completionTime += double(delay);
    Particle& pp = input.get();
    output.put(completionTime) = pp;
}
```

delay parameter indicates the service time

Server Star

Functional Star

```
defstar {
    name { Switch }
    domain {DE}
    desc { Switches input events to one of two outputs,
depending on the last received control input. }
    input {
        name { input }
        type { anytype }
    }
    input {
        name { control }
        type { int }
    }
    output {
        name { true }
        type { =input }
    }
    output {
        name { false }
        type { =input }
    }
```

Functional Stars

```
constructor {
    control.triggers();
    control.before(input);
}
```

Control input does not trigger outputs

For equal timestamps, read control input first

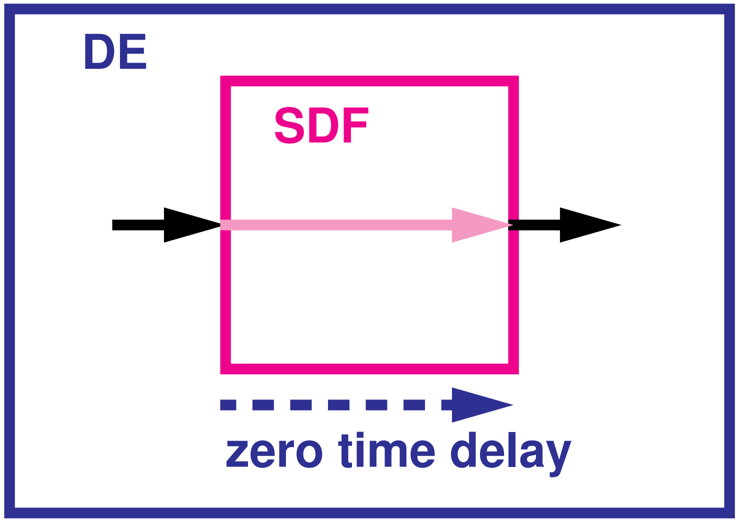
Checking for New Data on Data Input

```
go {
    if ( input.dataNew ) {
        completionTime = arrivalTime;
        Particle& pp = input.get();
        int c = int(control%0);
        if ( c )
            true.put(completionTime) = pp;
        else
            false.put(completionTime) = pp;
    }
}
```

Wormholes

SDF Domain Inside a DE Domain

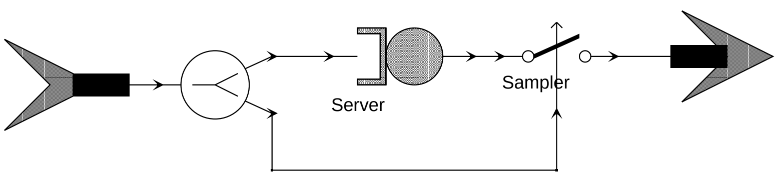
- SDF domain is treated as a functional DE star (zero delay)
- Multirate SDF subsystems may not always produce output for a single input



Wormholes

DE Subsystem Inside an SDF Subsystem

- SDF subsystem treats the DE subsystem as an SDF star: it must consume and produce a fixed number of tokens
- SDF subsystem will expect the DE subsystem to produce an output when given input



The DE sampler star ensures that the appropriate number of output events are produced in response to input events.