

SyDEVS Introduction

Theory – Paradigm – Implementation

Autodesk Research

June 2018

SyDEVS is a framework supporting the development and integration of systems analysis and simulation code.

Decades of *theory* on the representation of systems forms a basis for the SyDEVS approach.

A *paradigm* has been developed that combines dataflow programming with discrete event simulation, and allows any simulation to be specified in the form of a node graph.

To *implement* a simulation using this approach, nodes are defined as C++ classes which inherit from classes in the SyDEVS open source library.

Theory

Theory

SyDEVS is based on theory that
dates back to the late 1960s.

1970

1980

1990

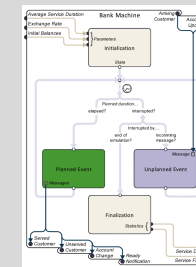
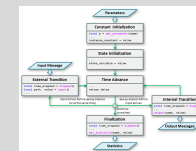
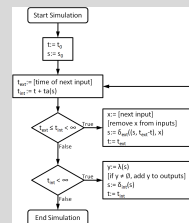
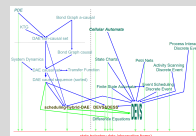
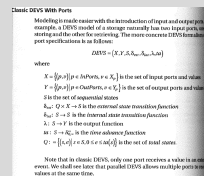
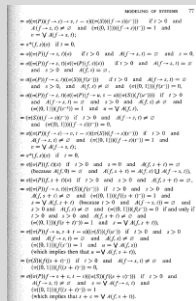
2000

2010

2020

Theory

Here is a small sample of related publications from the last 50 years.



Theory

$$\begin{aligned}
 &= \sigma((\pi(P))(f \rightarrow s) \rightarrow v, t - v)((\pi(S))((f \rightarrow s)(v^-))) \quad \text{if } t > 0 \text{ and} \\
 &\quad A(f \rightarrow s, t) \neq \emptyset \text{ and } (\pi(\{0, 1\}))((f \rightarrow s)(t^-)) = 1 \text{ and} \\
 &\quad v = \bigvee A(f \rightarrow s, t); \\
 &= \sigma^*(f, s)(x) \quad \text{if } t = 0, \\
 &= \sigma((\pi(P))f \rightarrow s, t)(x) \quad \text{if } t > 0 \text{ and } A(f \rightarrow s, t) = \emptyset \text{ and } s = 0, \\
 &= \sigma((\pi(P))f \rightarrow s, t)(\sigma((\pi(P))f, s)(x)) \quad \text{if } t > 0 \text{ and } A(f \rightarrow s, t) = \emptyset \\
 &\quad \text{and } s > 0 \text{ and } A(f, s) = \emptyset, \\
 &= \sigma((\pi(P))f \rightarrow s, t)((\pi(S))(f(s^-))) \quad \text{if } t > 0 \text{ and } A(f \rightarrow s, t) = \emptyset \\
 &\quad \text{and } s > 0, \text{ and } A(f, s) \neq \emptyset \text{ and } (\pi(\{0, 1\}))(f(s^-)) = 0, \\
 &= \sigma((\pi(P))f \rightarrow s, t)(\sigma((\pi(P))f \rightarrow u, s - u)((\pi(S))(f(u^-)))) \quad \text{if } t > 0 \\
 &\quad \text{and } A(f \rightarrow s, t) = \emptyset \text{ and } s > 0 \text{ and } A(f, s) \neq \emptyset \text{ and} \\
 &\quad (\pi(\{0, 1\}))(f(s^-)) = 1 \text{ and } u = \bigvee A(f, s), \\
 &= (\pi(S))((f \rightarrow s)(t^-)) \quad \text{if } t > 0 \text{ and } A(f \rightarrow s, t) \neq \emptyset \\
 &\quad \text{and } (\pi(\{0, 1\}))((f \rightarrow s)(t^-)) = 0, \\
 &= \sigma((\pi(P))(f \rightarrow s) \rightarrow v, t - v)((\pi(S))((f \rightarrow s)(v^-))) \quad \text{if } t > 0 \text{ and} \\
 &\quad A(f \rightarrow s, t) \neq \emptyset \text{ and } (\pi(\{0, 1\}))((f \rightarrow s)(t^-)) = 1 \text{ and} \\
 &\quad v = \bigvee A(f \rightarrow s, t); \\
 &= \sigma^*(f, s)(x) \quad \text{if } t = 0, \\
 &= \sigma((\pi(P))f, t)(x) \quad \text{if } t > 0 \text{ and } s = 0 \text{ and } A(f, s + t) = \emptyset \\
 &\quad (\text{because } A(f, 0) = \emptyset \text{ and } A(f, s + t) = A(f, s) \cup A(f \rightarrow s, t)), \\
 &= \sigma((\pi(P))f, s + t)(x) \quad \text{if } t > 0 \text{ and } s > 0 \text{ and } A(f, s + t) = \emptyset, \\
 &= \sigma((\pi(P))f \rightarrow s, t)((\pi(S))(f(s^-))) \quad \text{if } t > 0 \text{ and } s > 0 \text{ and} \\
 &\quad A(f, s + t) \neq \emptyset \text{ and } (\pi(\{0, 1\}))(f((s + t)^-)) = 1 \text{ and} \\
 &\quad s = \bigvee A(f, s + t) \text{ (because } t > 0 \text{ and } A(f \rightarrow s, t) = \emptyset \text{ and} \\
 &\quad s > 0 \text{ and } A(f, s) \neq \emptyset \text{ and } (\pi(\{0, 1\}))(f(s^-)) = 0 \text{ if and only if} \\
 &\quad t > 0 \text{ and } s > 0 \text{ and } A(f, s + t) \neq \emptyset \text{ and} \\
 &\quad (\pi(\{0, 1\}))(f(s + t)^-) = 1 \text{ and } s = \bigvee A(f, s + t)), \\
 &= \sigma((\pi(P))f \rightarrow u, s + t - u)((\pi(S))(f(u^-))) \quad \text{if } t > 0 \text{ and } s > 0 \\
 &\quad \text{and } A(f \rightarrow s, t) = \emptyset \text{ and } A(f, s) \neq \emptyset \text{ and} \\
 &\quad (\pi(\{0, 1\}))(f(s^-)) = 1 \text{ and } u = \bigvee A(f, s) \\
 &\quad (\text{which implies then that } u = \bigvee A(f, s + t)), \\
 &= (\pi(S))(f((s + t)^-)) \quad \text{if } t > 0 \text{ and } A(f \rightarrow s, t) \neq \emptyset \text{ and} \\
 &\quad (\pi(\{0, 1\}))(f((s + t)^-)) = 0, \\
 &= \sigma((\pi(P))f \rightarrow s + v, t - v)((\pi(S))(f((s + v)^-))) \quad \text{if } t > 0 \text{ and} \\
 &\quad A(f \rightarrow s, t) \neq \emptyset \text{ and } v = \bigvee A(f \rightarrow s, t) \text{ and} \\
 &\quad (\pi(\{0, 1\}))(f((s + t)^-)) = 1 \\
 &\quad (\text{which implies that } s + v = \bigvee A(f, s + t)).
 \end{aligned}$$

General methods to formally represent systems began with the work of Wayne Wymore.

A. Wayne Wymore

A Mathematical Theory of Systems Engineering

1967

Theory

Bernard P. Zeigler
Theory of Modeling and Simulation

1976

Classic DEVS With Ports

Modeling is made easier with the introduction of input and output ports. For example, a DEVS model of a storage naturally has two input ports, one for storing and the other for retrieving. The more concrete DEVS formalism with port specifications is as follows:

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

where

$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values

$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values

S is the set of *sequential* states

$\delta_{ext}: Q \times X \rightarrow S$ is the *external state transition function*

$\delta_{int}: S \rightarrow S$ is the *internal state transition function*

$\lambda: S \rightarrow Y$ is the output function

$ta: S \rightarrow R_{0, \infty}^+$ is the *time advance function*

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the set of *total states*.

Note that in classic DEVS, only one port receives a value in an external event. We shall see later that parallel DEVS allows multiple ports to receive values at the same time.

Bernard Zeigler applied similar ideas to simulation.

Theory

He found that essentially all simulations can be represented in a common form based on the discrete event simulation paradigm. Zeigler named this common form "DEVS".

Bernard P. Zeigler
Theory of Modeling and Simulation

1976

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

Theory

[illegible]

Classic DEVS with Ports

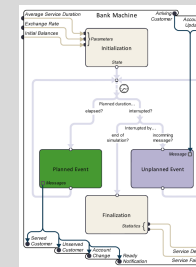
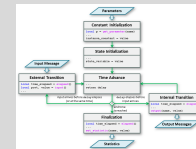
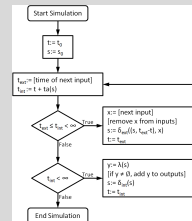
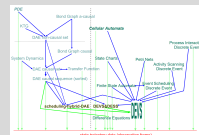
Modeling is made easier with the introduction of *input* and *output* ports. For example, a DEVS model of a memory running on two input ports on which it receives data and on which it outputs data, has the following port specifications in its definition:

$$\text{DEVS} = \langle X, Y, \lambda, \text{out}, \text{in}, \text{out}_0, \text{in}_0 \rangle$$

where

- $X = \{x \in \mathcal{P}\} \mid \exists p \in \text{InPorts}, p \in X_p\}$ is the set of input ports and value.
- $Y = \{y \in \mathcal{P}\} \mid \exists p \in \text{OutPorts}, p \in Y_p\}$ is the set of output ports and value.
- In the set of sequential states.
- $\text{Out} : X \times \mathcal{P} \rightarrow S$ is the internal state transition function.
- $\text{in}_0 : S \rightarrow S$ is the output function.
- $\text{out} : S \rightarrow \mathcal{K}_p$ is the data selection function.
- $\text{Q} = \{[x, y] \in S \times S \mid x \in \text{In}, y \in \text{Out}\}$ is the set of total states.

Note that in classic DEVS, only one port receives a value at one time. We shall see later that parallel DEVS allows multiple ports to receive a value at one time.

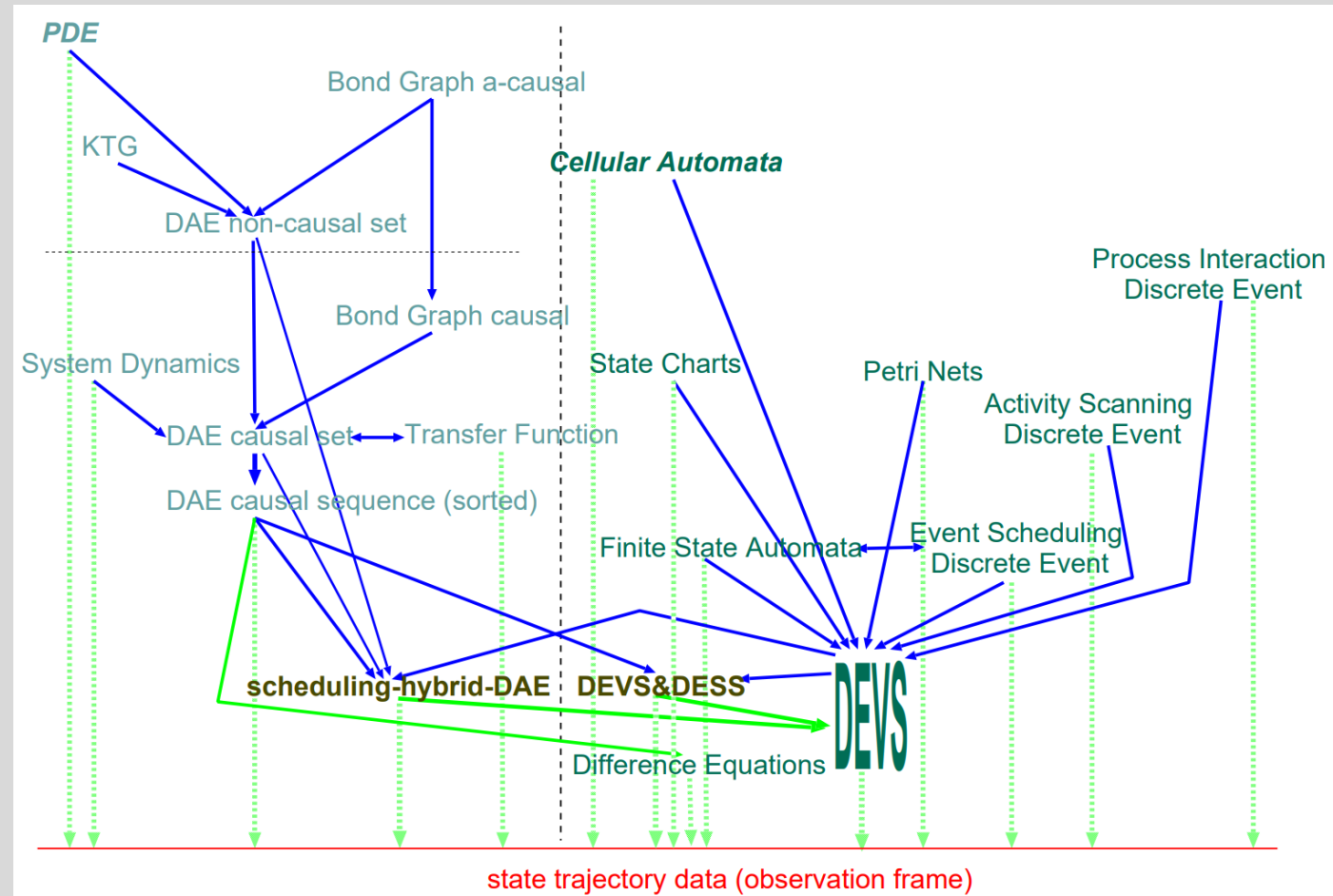


Theory

Over the years, the generality of DEVS was confirmed. It was found that for each of the most common modeling paradigms, any model expressed in that paradigm could also be represented using DEVS.

Hans L. M. Vangheluwe
DEVS as a Common Denominator...

2000



Theory

[illegible]

Clinic DEVS Ports

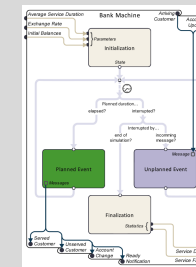
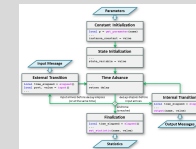
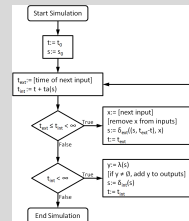
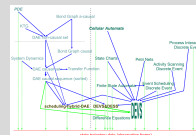
Modeling is made easier with the introduction of *input* and *output* ports, for example, a DEVS model of a storage runway has two input ports, one for the arrival of the train and the other for the arrival of the train. The more concrete DEVS formalism specifications in its full form:

$$DEVS = (X, I, A, S_{init}, A_{out}, S, \lambda)$$

where

- $X = \{x_i | i \in I\}$ is the set of input ports and values
- $I = \{i_j | j \in OutPorts, i_j \in V_{i_j}\}$ is the set of output ports and values
- S is the set of sequential states
- $S_{init} : Q \rightarrow S$ is the external state transition function
- $A_{out} : S \rightarrow S$ is the internal state transition function
- $S \rightarrow S$ is the output function
- $A_{in} : S \rightarrow A_{in}$ is the time advance function
- $Q = \{q_i | q_i \in S, 0.5 \leq q_i \leq 1\}$ is the set of total times

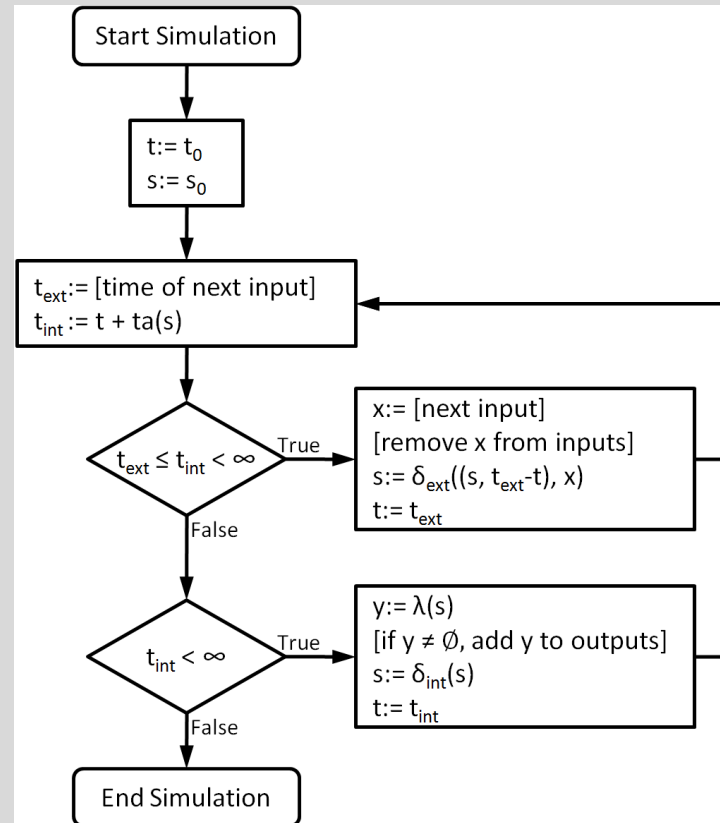
Note that in clinic DEVS, only one port receives a value at a time, even though one that passed DEVS allows multiple ports to receive the same time.



Theory

Autodesk Research & Carleton University
Formal Languages for Computer Simulation

2011



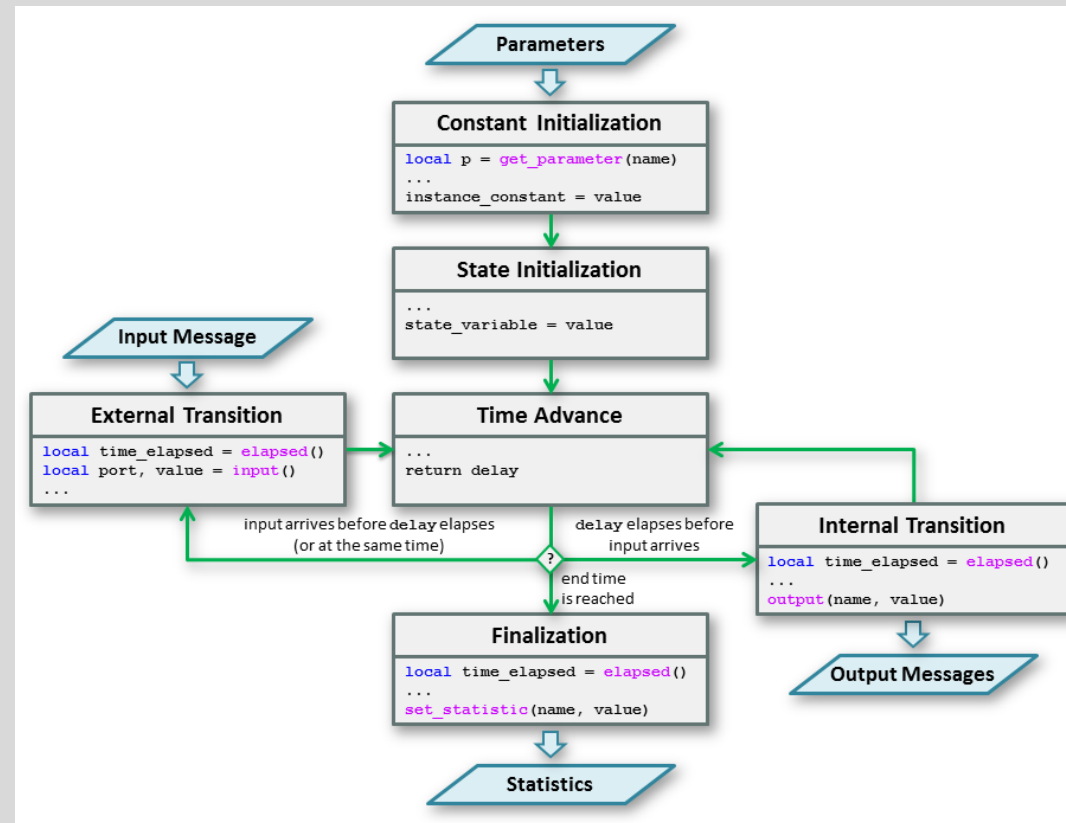
In 2011, researchers at Autodesk began exploring how to make DEVS more approachable to scientific and engineering communities.

Theory

The process involved several iterations.

Autodesk Research
DesignDEVS Help Graphic

2012

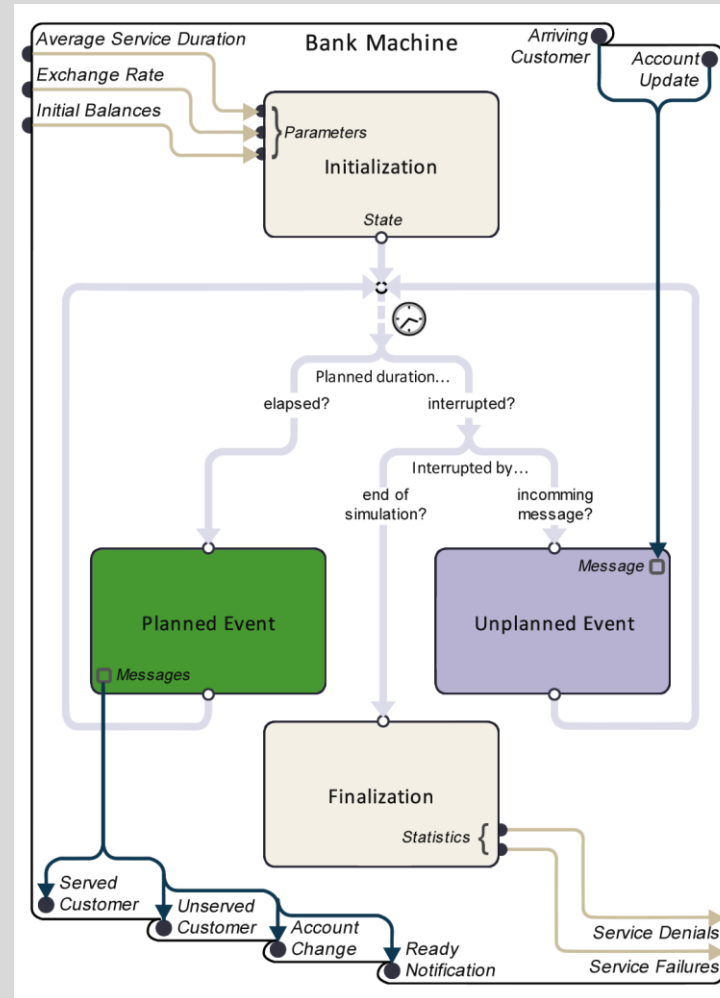


Theory

Eventually, a set of visual interfaces were designed along with a new way of expressing DEVS.

Autodesk Research & Simon Fraser University
Designing DEVS Visual Interfaces...

2015



Theory

[illegible]

Clinic DEVS Ports

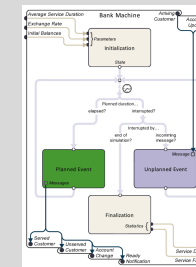
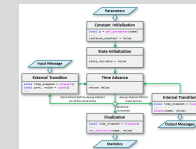
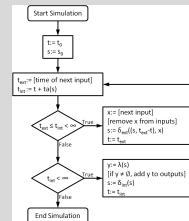
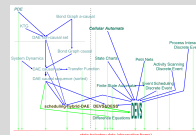
Modeling is made easier with the introduction of *input* and *output* ports, for example, a DEVS model of a storage runway has two input ports, one for the arrival of the train and the other for the arrival of the train. The more concrete DEVS formalism specifications in its full form:

$$DEVS = (X, I, A, S_{init}, A_{out}, S, \lambda)$$

where

- $X = \{x_i | i \in I\}$ is the set of input ports and values
- $I = \{i_j | j \in OutPorts, i_j \in V_{i_j}\}$ is the set of output ports and values
- S is the set of sequential states
- $S_{init} : Q \rightarrow S$ is the external state transition function
- $A_{out} : S \rightarrow S$ is the internal state transition function
- $S \rightarrow S$ is the output function
- $A_{in} : S \rightarrow A_{in}$ is the time advance function
- $Q = \{q_i | q_i \in S, 0.5 \leq q_i \leq 1\}$ is the set of total times

Note that in clinic DEVS, only one port receives a value at a time, even though one that passed DEVS allows multiple ports to receive the same time.



Theory

For more information on the theory underlying the SyDEVS framework, visit the Autodesk Research website and find the "Systems Design & Simulation" project.

www.autodeskresearch.com

Project: Systems Design & Simulation

The screenshot displays the Autodesk Research website for the "Systems Design & Simulation" project. The page features a navigation bar with links to PUBLICATIONS, PEOPLE, GROUPS, PROJECTS, OPPORTUNITIES, BLOG, and NEWS. The main content area includes a title "Systems Design & Simulation" and a descriptive paragraph about the project's goal to create a more scalable and collaborative modeling approach. A diagram on the right illustrates a complex system architecture with various components and data flows. Below the description, there are social media icons for GitHub, Facebook, Twitter, and LinkedIn. A sidebar on the left contains links to OVERVIEW, PUBLICATIONS, PROJECT MEMBERS, and ALUMNI MEMBERS. The main content area also lists 19 publications, with three highlighted: "Multiscale Representation of Simulated Time" (2017), "Simulation-Based Architectural Design" (2017), and "Practical Aspects of the DesignDEVs Simulation Environment" (2017). Each publication entry includes a thumbnail image, the title, authors, and a brief description.

Systems Design & Simulation

While traditional programming practices have produced a wide range of relatively independent simulation methods, predictive models of extremely complex natural and artificial systems will require a more scalable, more collaborative approach to modeling. This project strives for software that will help researchers develop, debug, document, share, and integrate simulation code.

GROUPS
Complex Systems Research

19 publications

Multiscale Representation of Simulated Time
Rhys Goldstein, Azam Khan, Olivier Dalle, Gabriel Wainer (2017)
SIMULATION: Transactions of The Society for Modeling and Simulation International (SAGE)
40 pages.
DOWNLOAD PDF DETAILS >

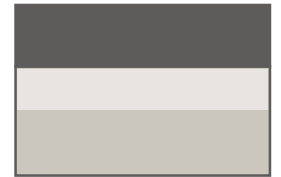
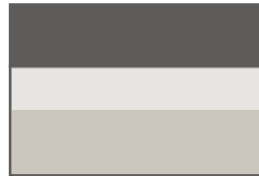
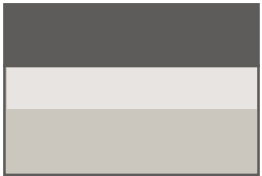
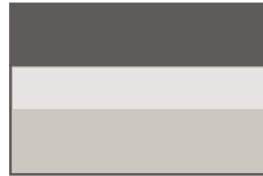
Simulation-Based Architectural Design
Rhys Goldstein, Azam Khan (2017)
Guide to Simulation-Based Disciplines: Advancing Our Computational Future
Springer
July 2017
pp. 167-182
DETAILS >

Practical Aspects of the DesignDEVs Simulation Environment
Rhys Goldstein, Simon Breslav, Azam Khan (2017)
SIMULATION: Transactions of The Society for Modeling and Simulation International (SAGE)
August 2017, Published online.
26 pages
DOWNLOAD PDF DETAILS >

Paradigm

Paradigm

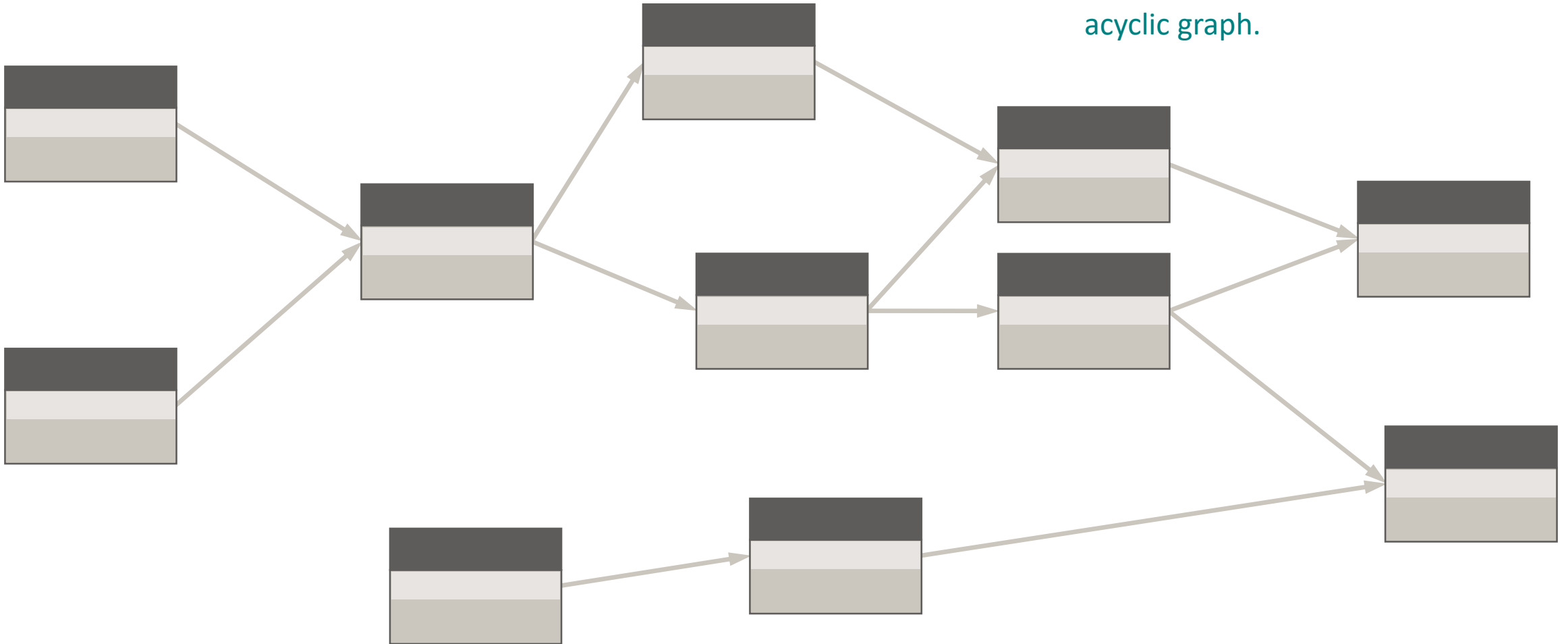
SyDEVS supports a node-based modeling paradigm combining dataflow programming with DEVS.



Paradigm

Dataflow

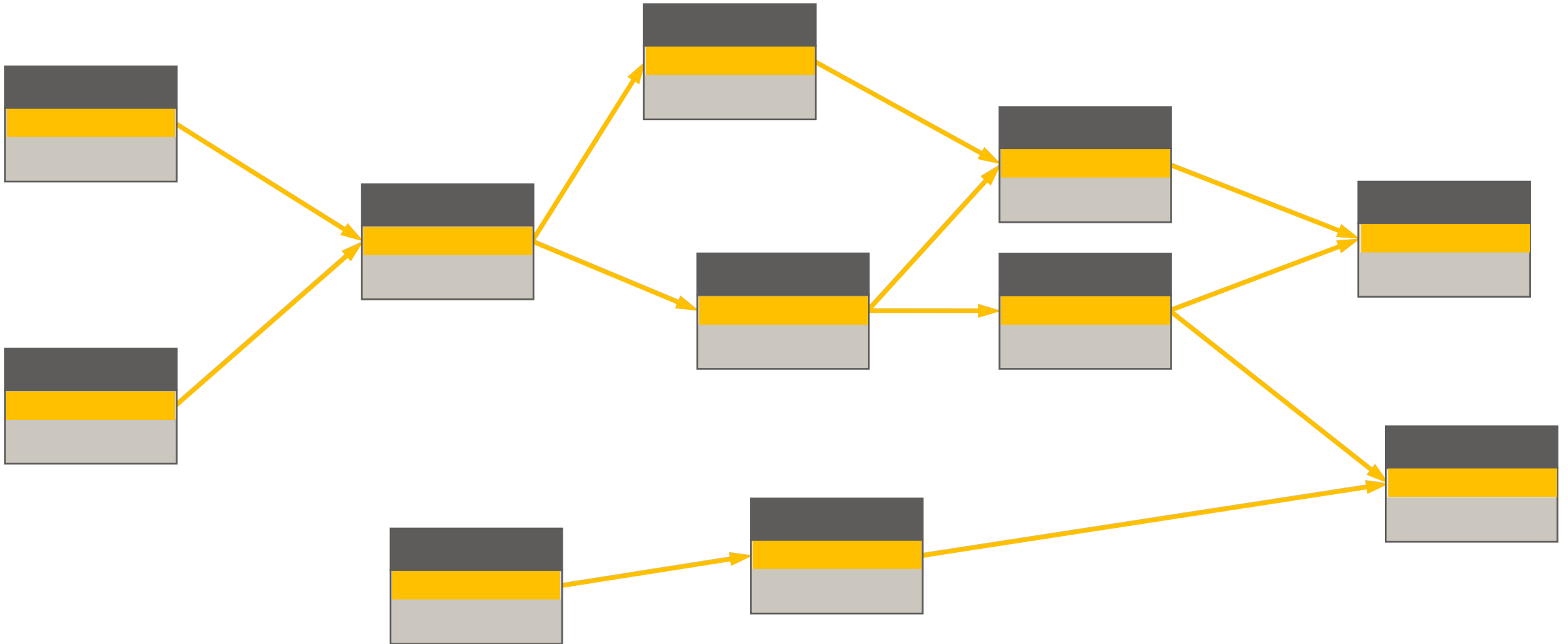
Dataflow programming is a widely used paradigm in which the links between nodes form a directed acyclic graph.



Paradigm

Dataflow

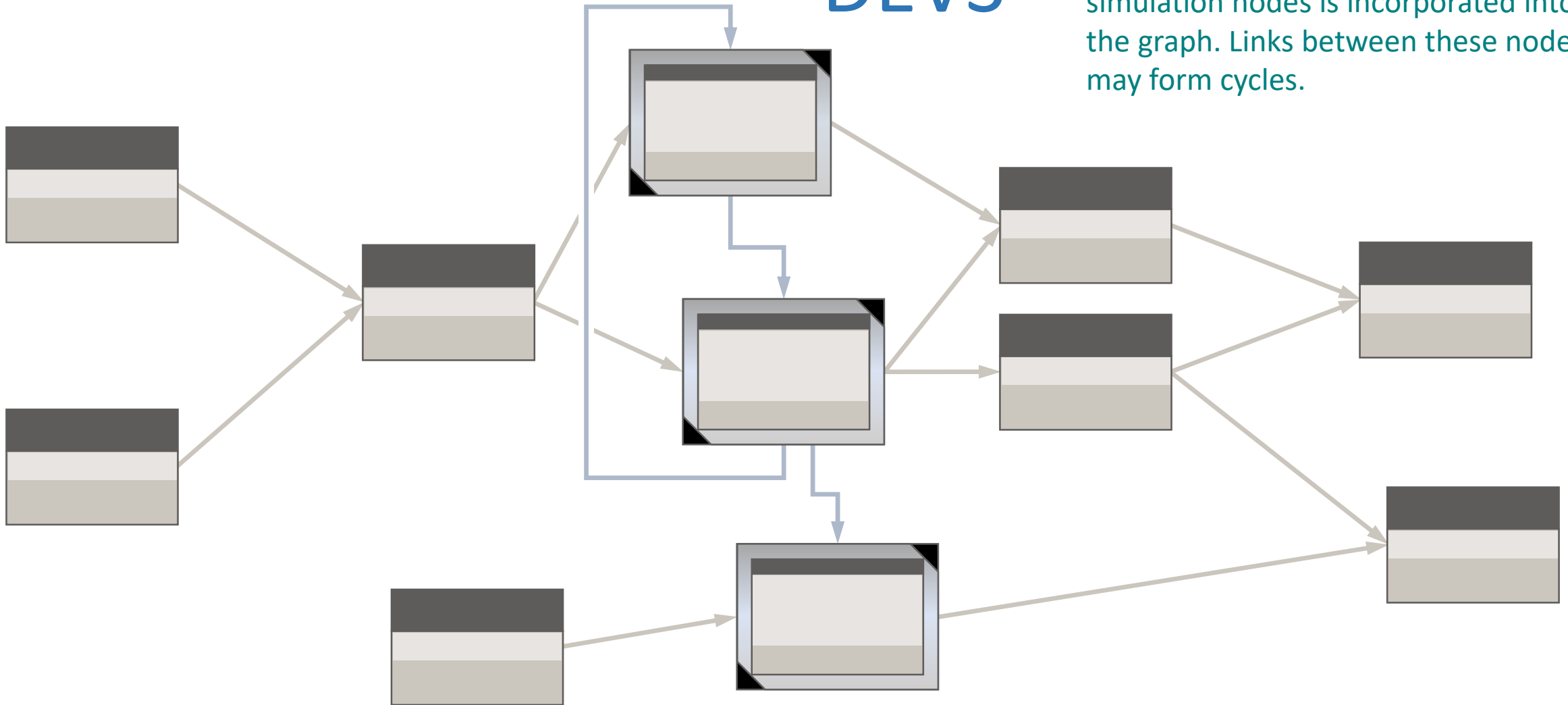
A node is executed once it receives data on all of its input ports.



Paradigm

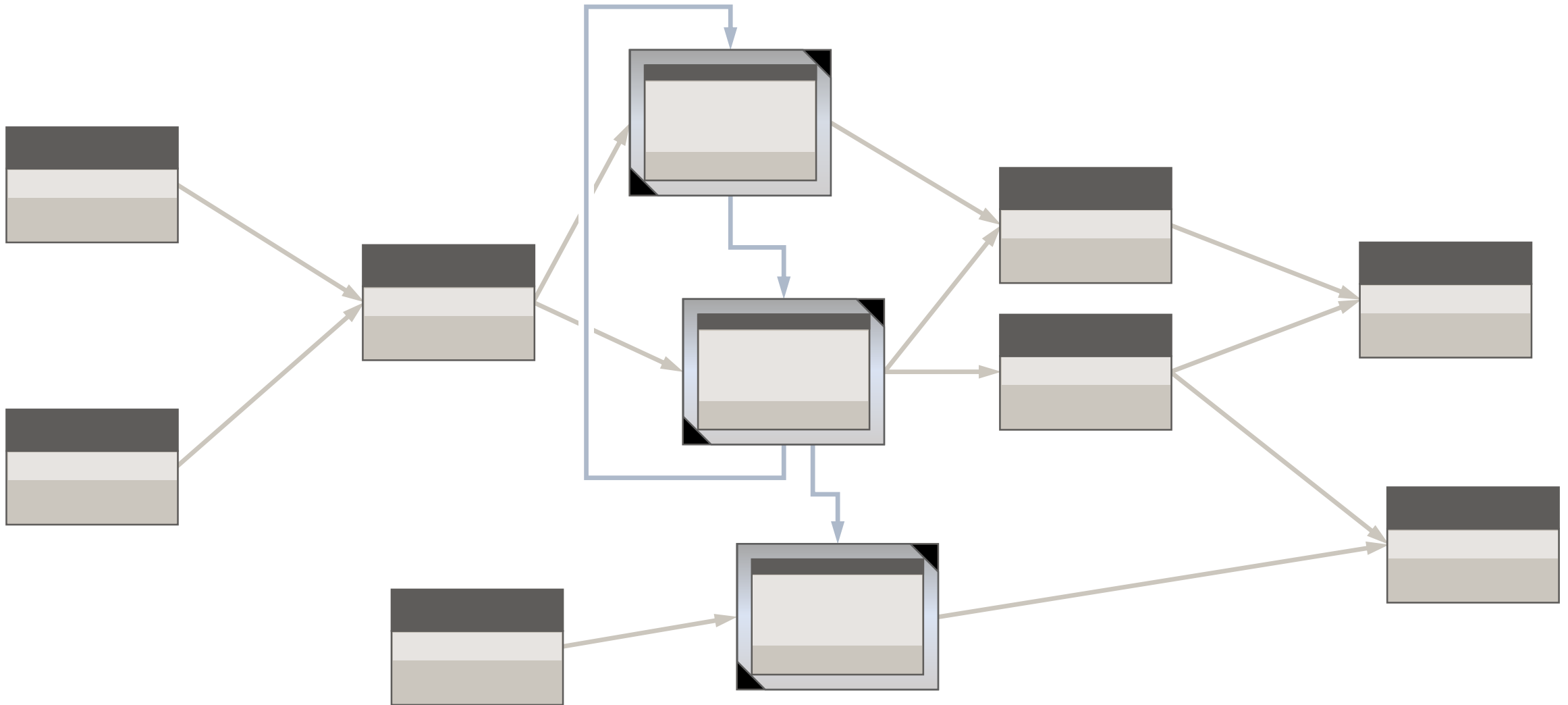
DEVS

To introduce the notion of time into the paradigm, a column of discrete event simulation nodes is incorporated into the graph. Links between these nodes may form cycles.



Paradigm

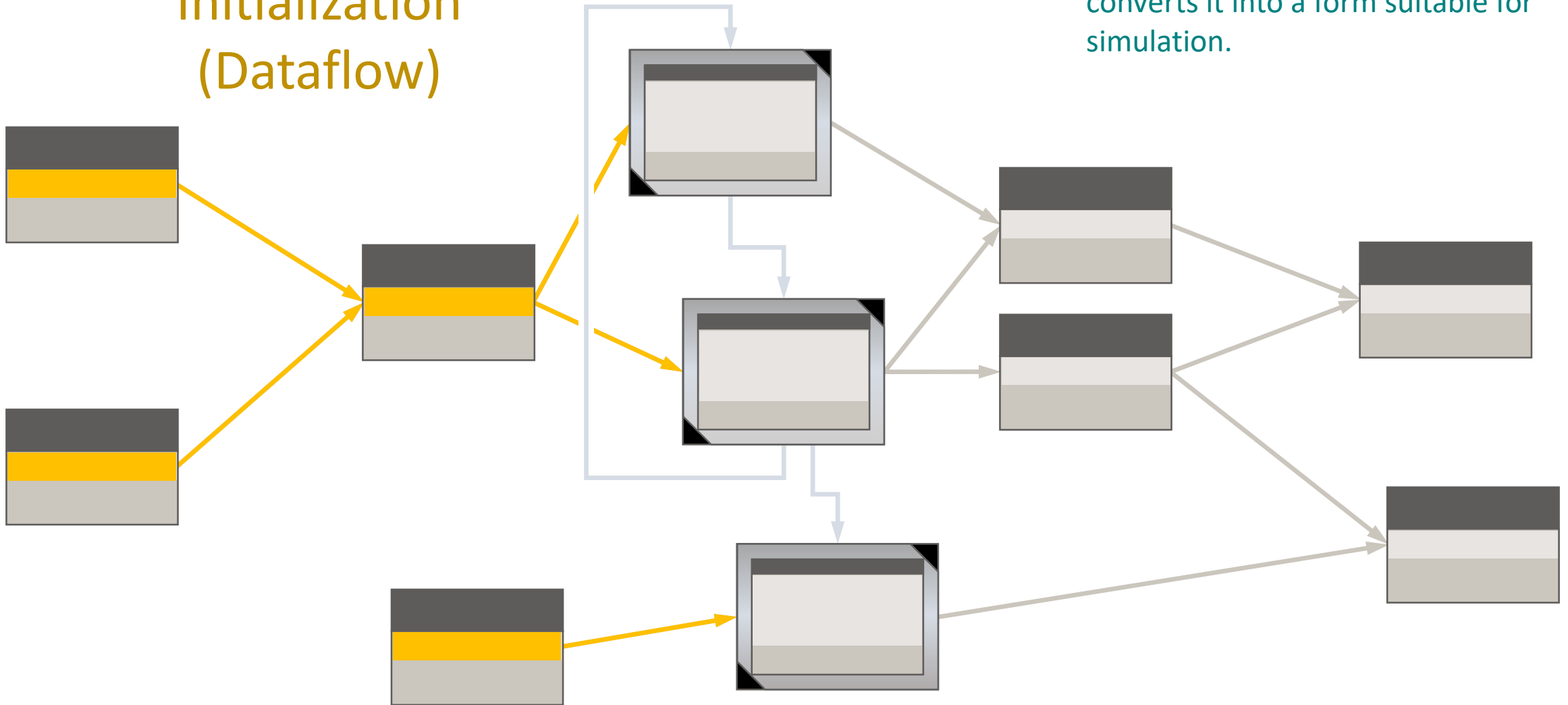
The execution of the entire graph is now partitioned into three phases.



Paradigm

Initialization (Dataflow)

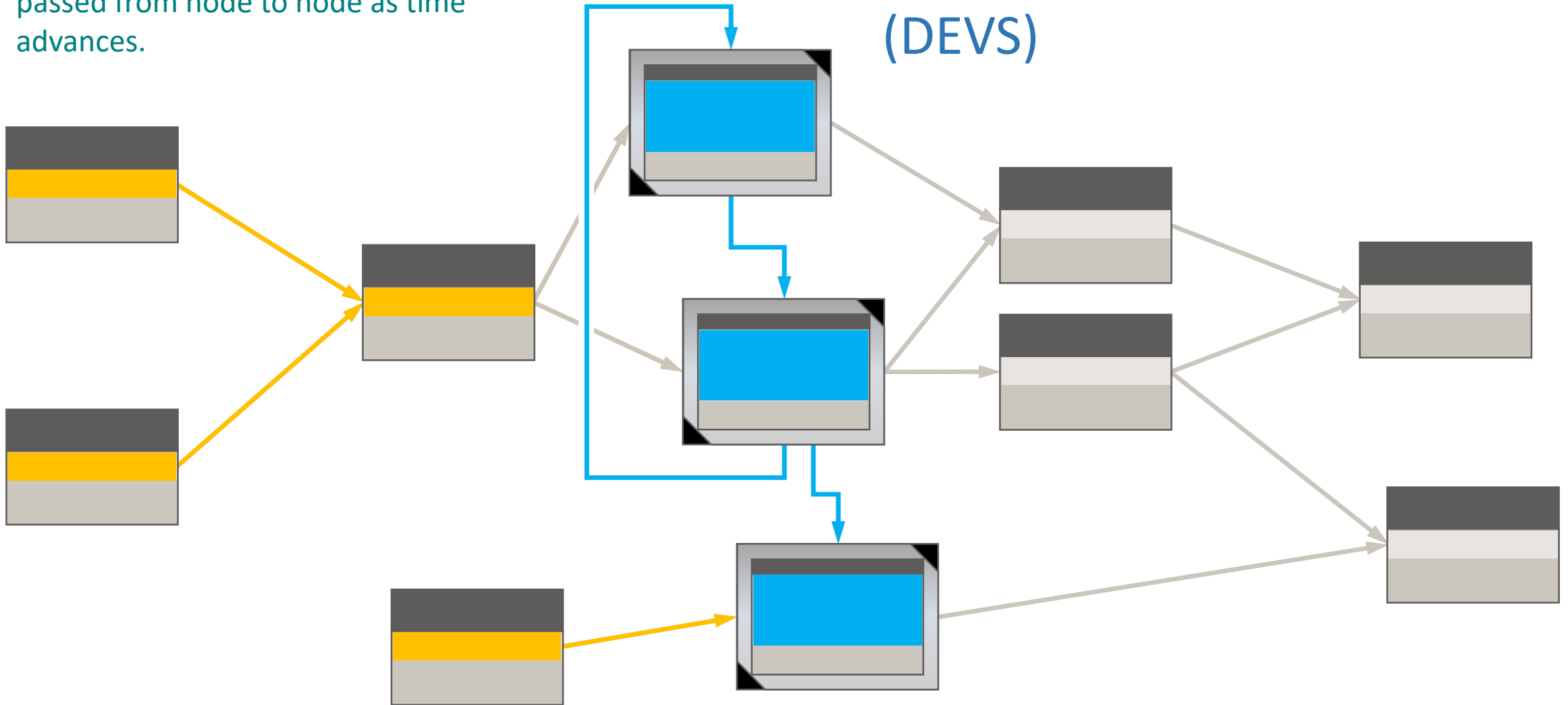
The first phase is called "Initialization".
A dataflow network collects data and
converts it into a form suitable for
simulation.



Paradigm

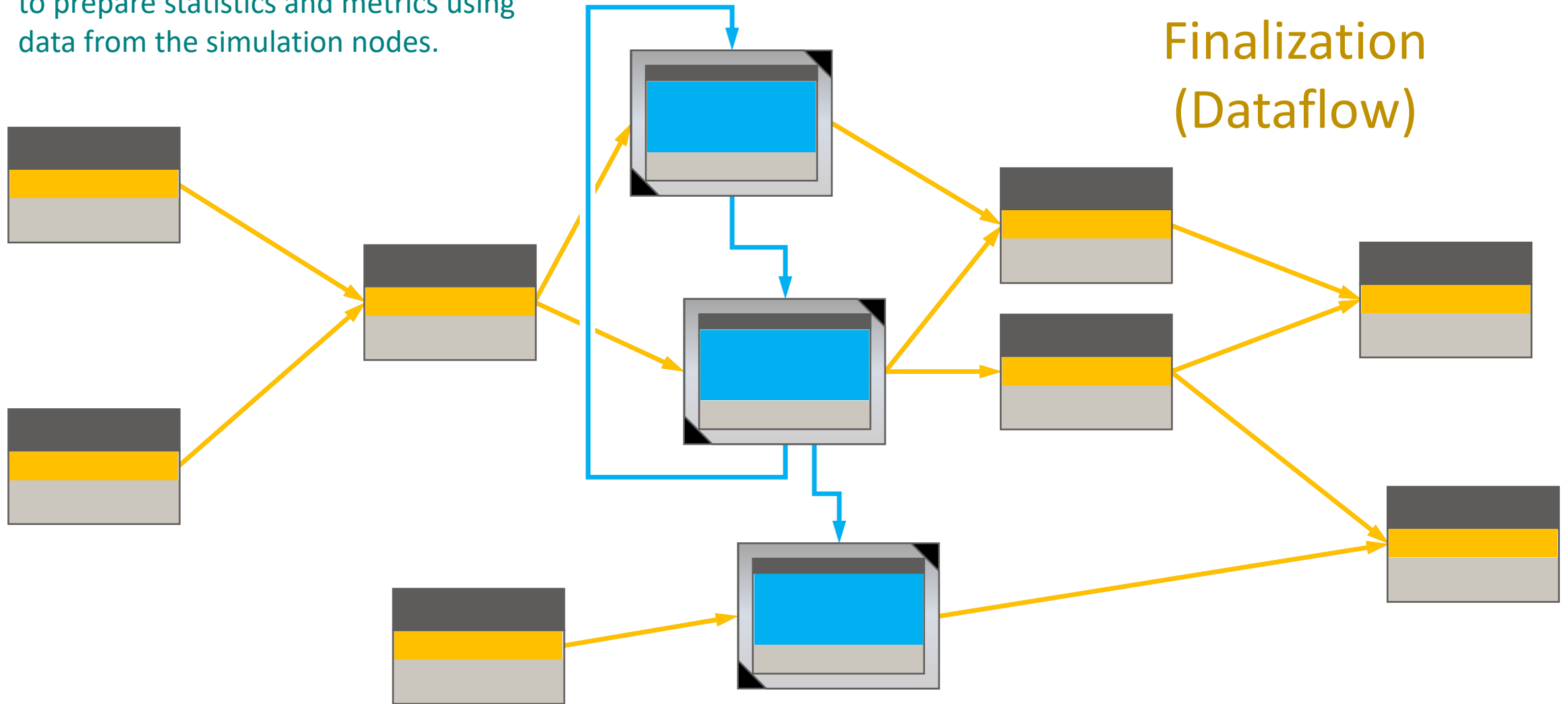
The second phase is the "Simulation" (or "DEVS") phase. Messages are passed from node to node as time advances.

Simulation (DEVS)



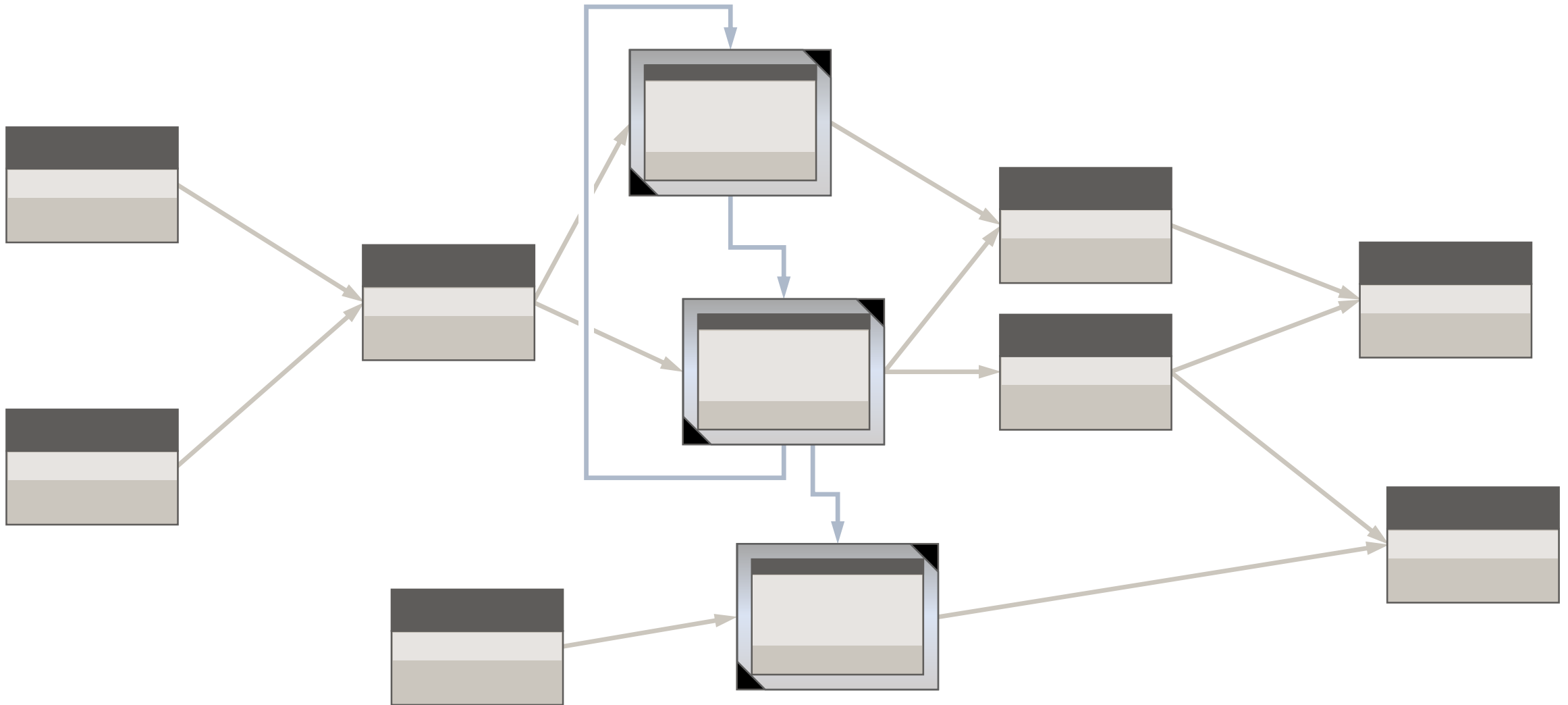
Paradigm

The third phase is called "Finalization".
Another dataflow network is executed
to prepare statistics and metrics using
data from the simulation nodes.

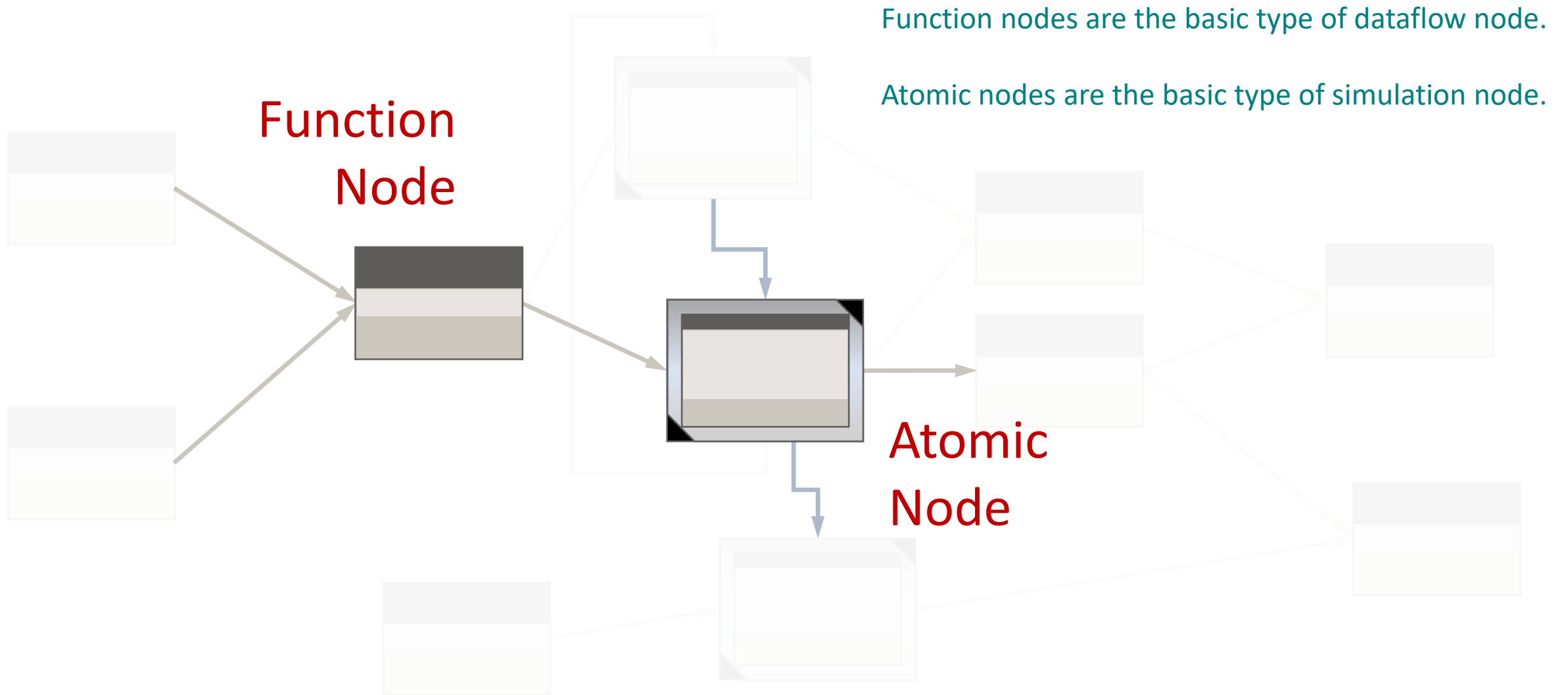


Paradigm

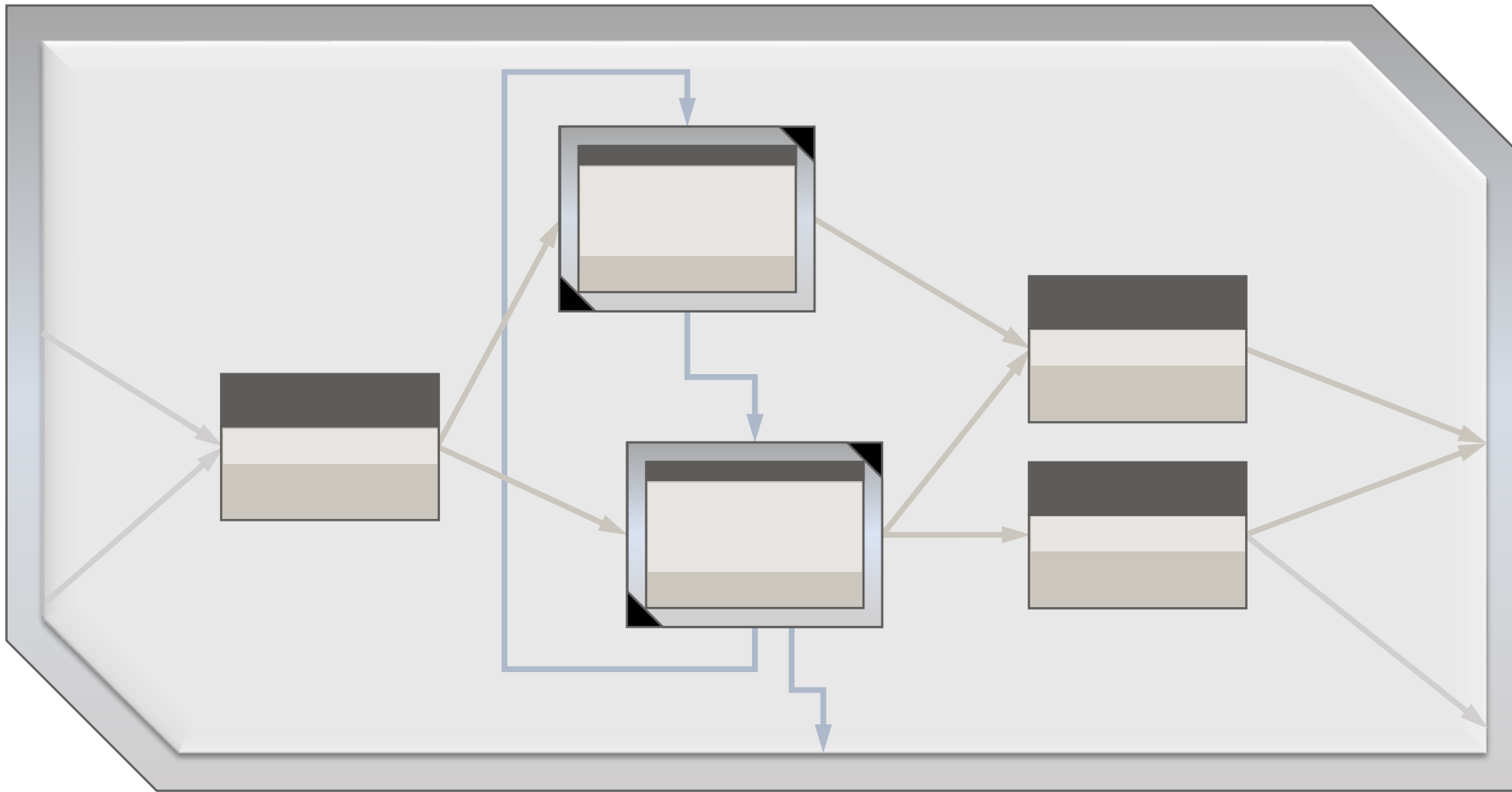
There are four types of nodes.



Paradigm



Paradigm



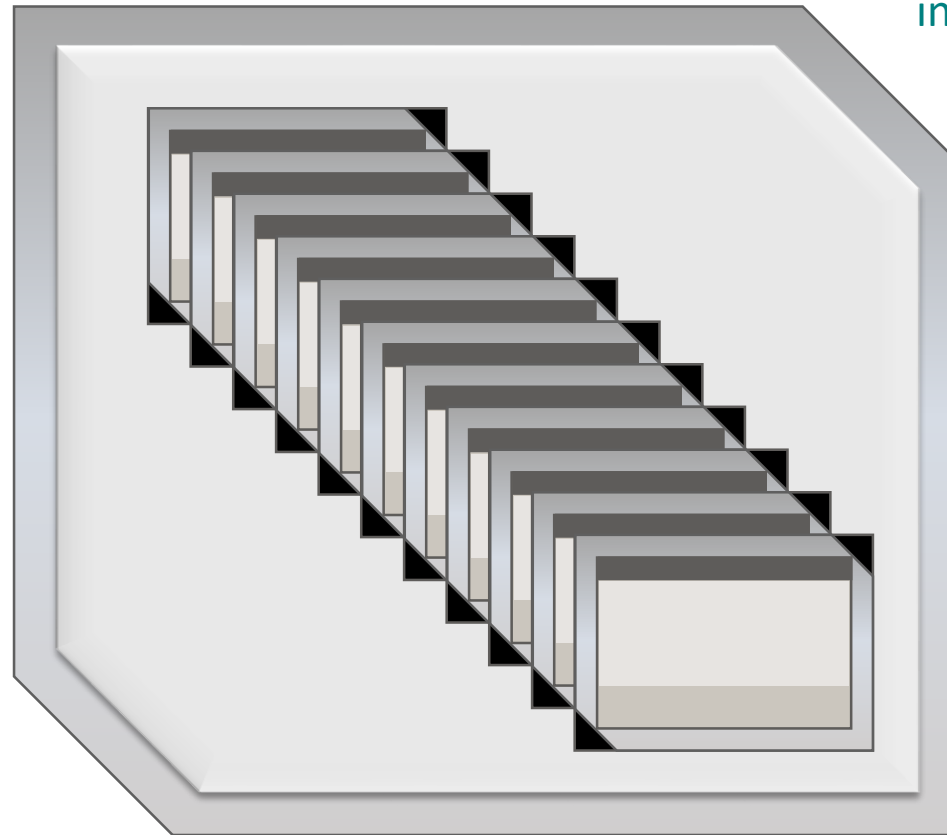
Composite nodes contain networks (dataflow + DEVS + dataflow) of other nodes. The contained nodes can themselves be composite nodes, forming a hierarchy.

Composite
Node

Paradigm

Collection nodes contain any number of instances of a single type of node. The number of instances can change during a simulation. Collection nodes are useful for agent-based modeling, where each instance is an agent.

Collection
Node



Paradigm

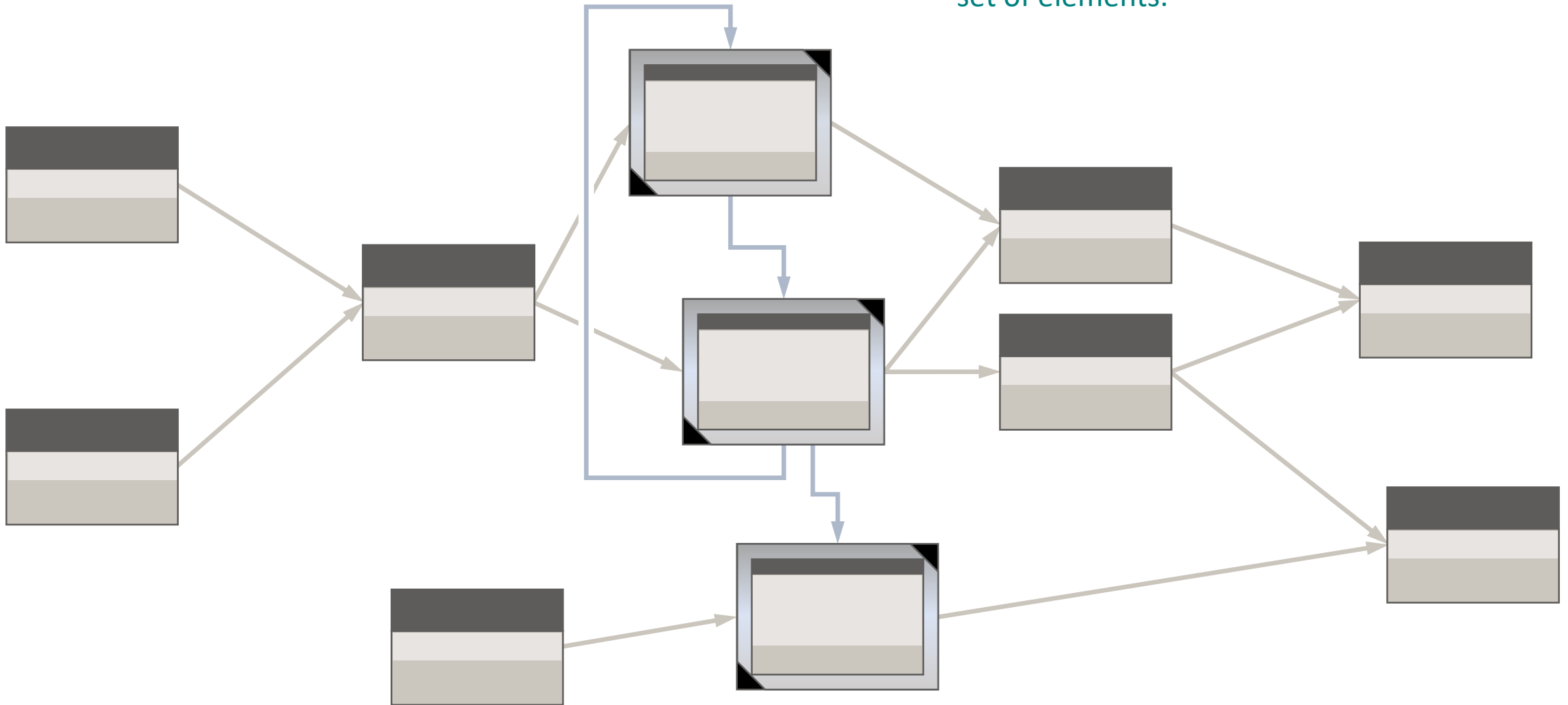
Nodes

1. Function Node
2. Atomic Node
3. Composite Node
4. Collection Node

Here is a list of the four types of nodes.

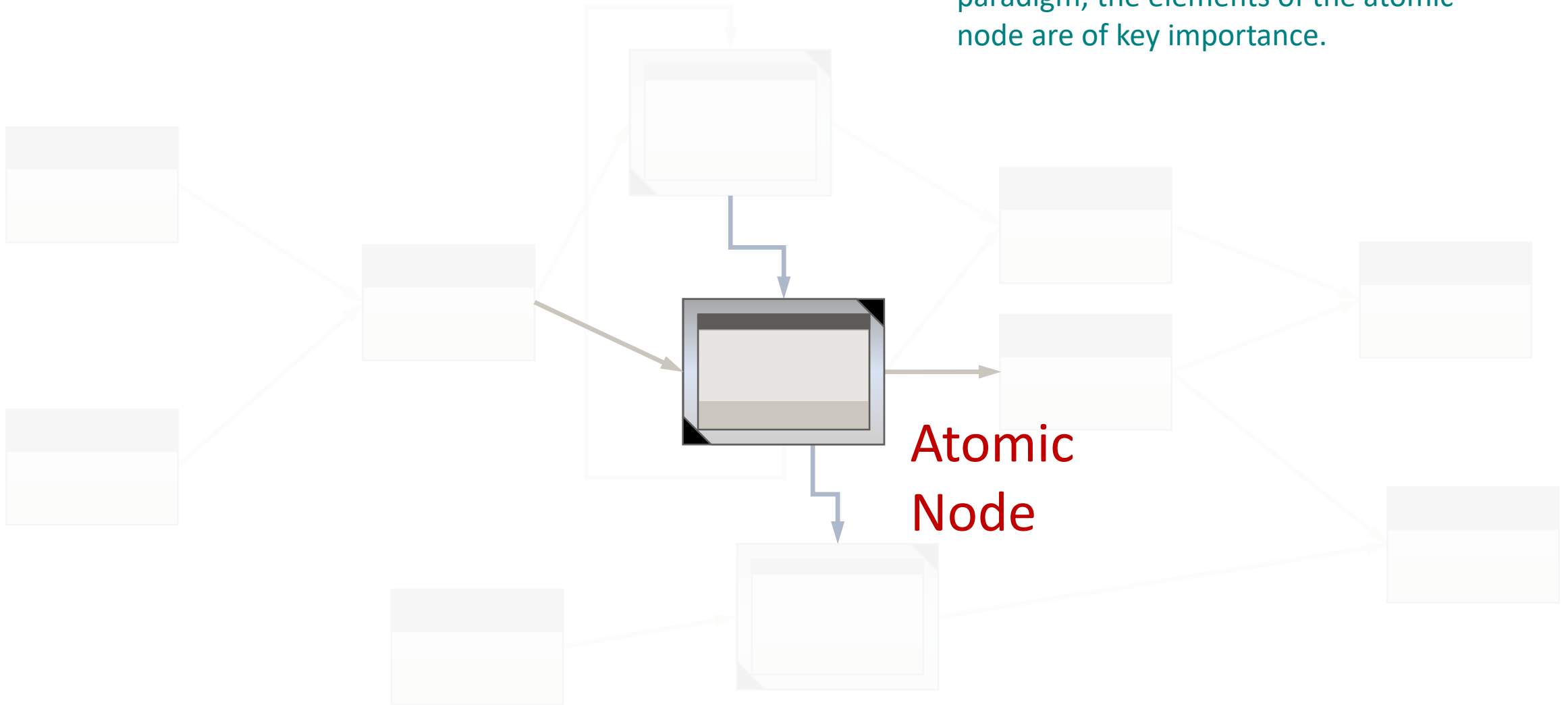
Paradigm

Each type of node contains a particular set of elements.



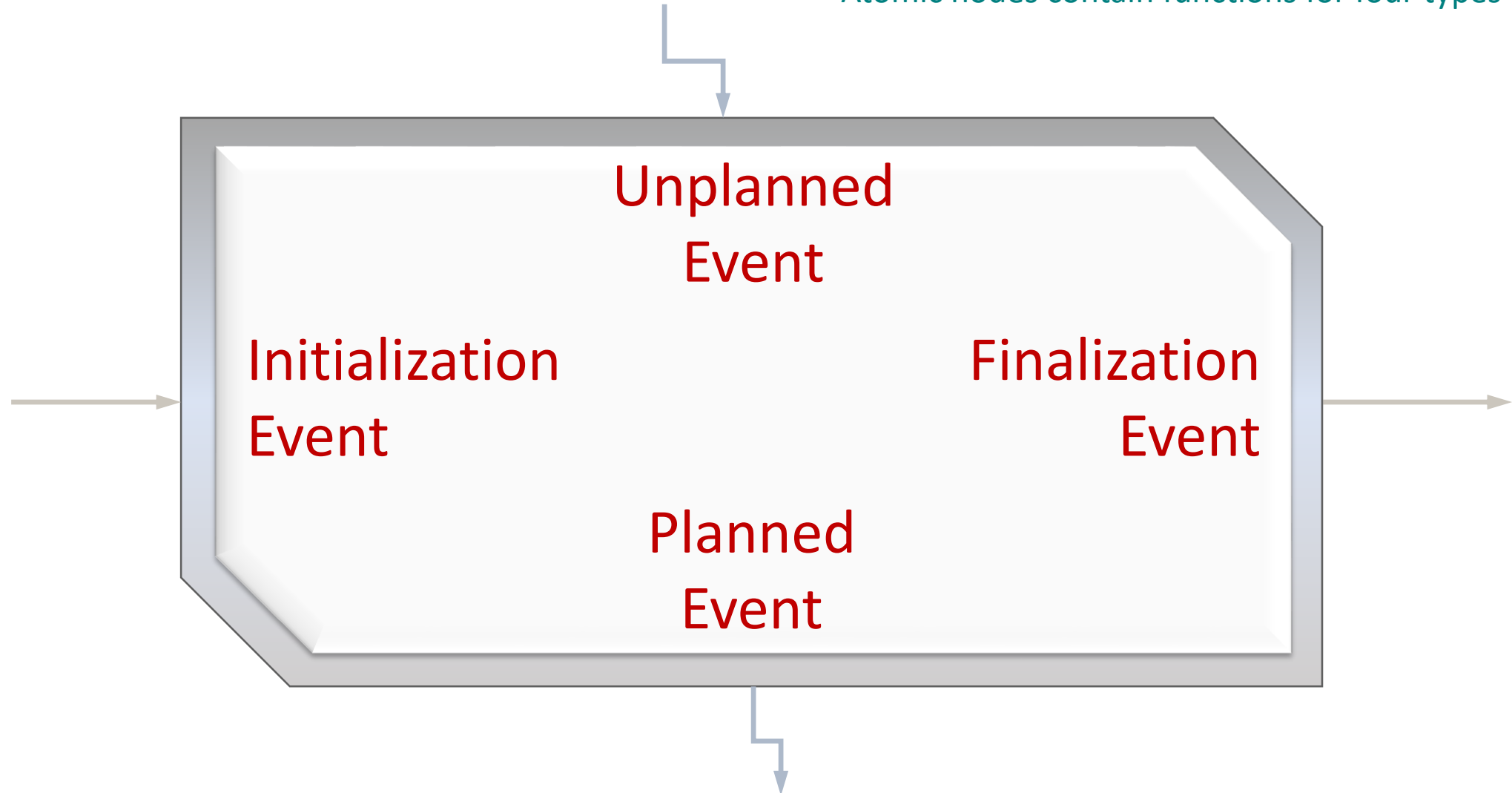
Paradigm

For the purpose of understanding the paradigm, the elements of the atomic node are of key importance.



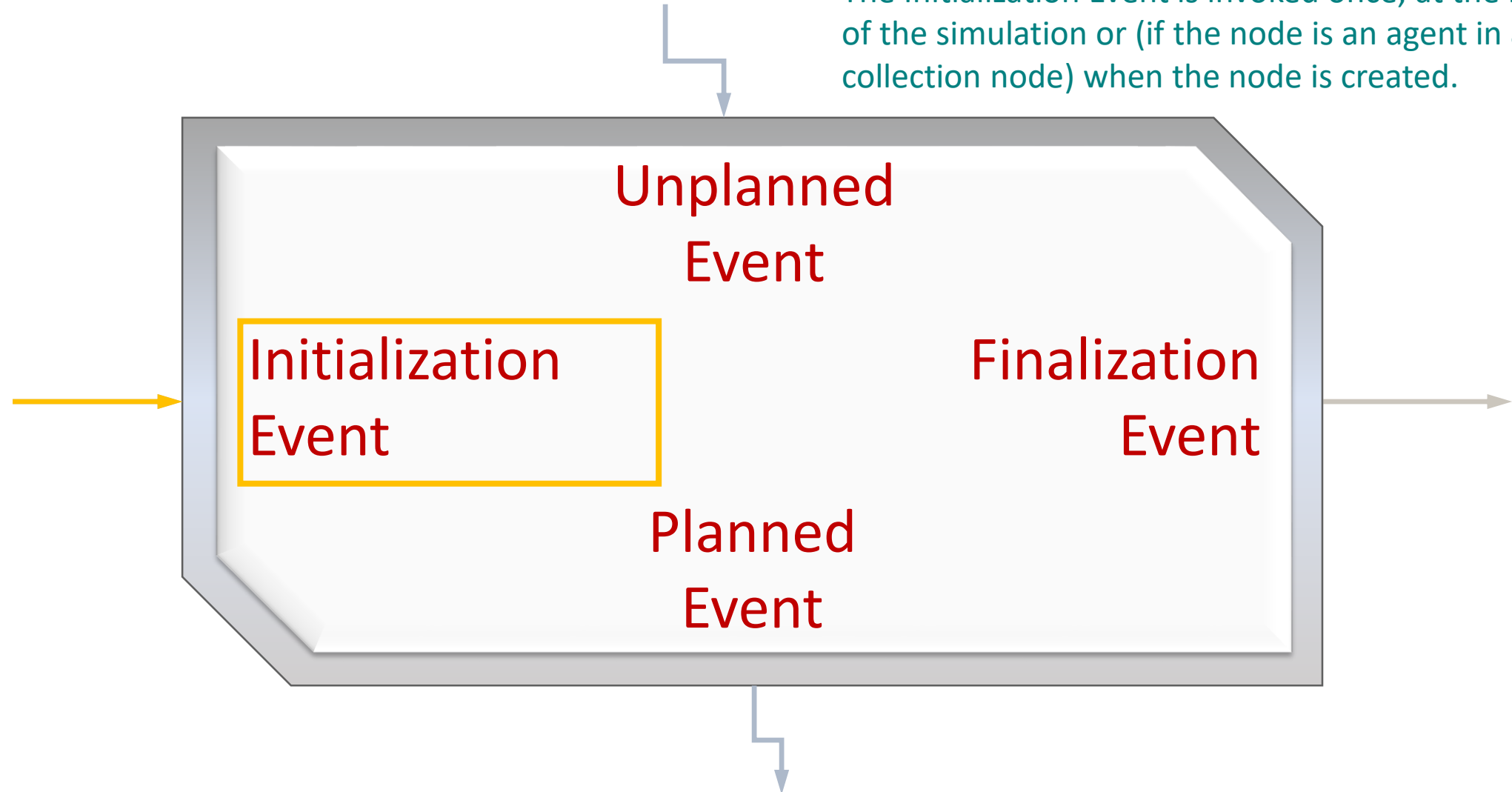
Paradigm

Atomic nodes contain functions for four types of events.



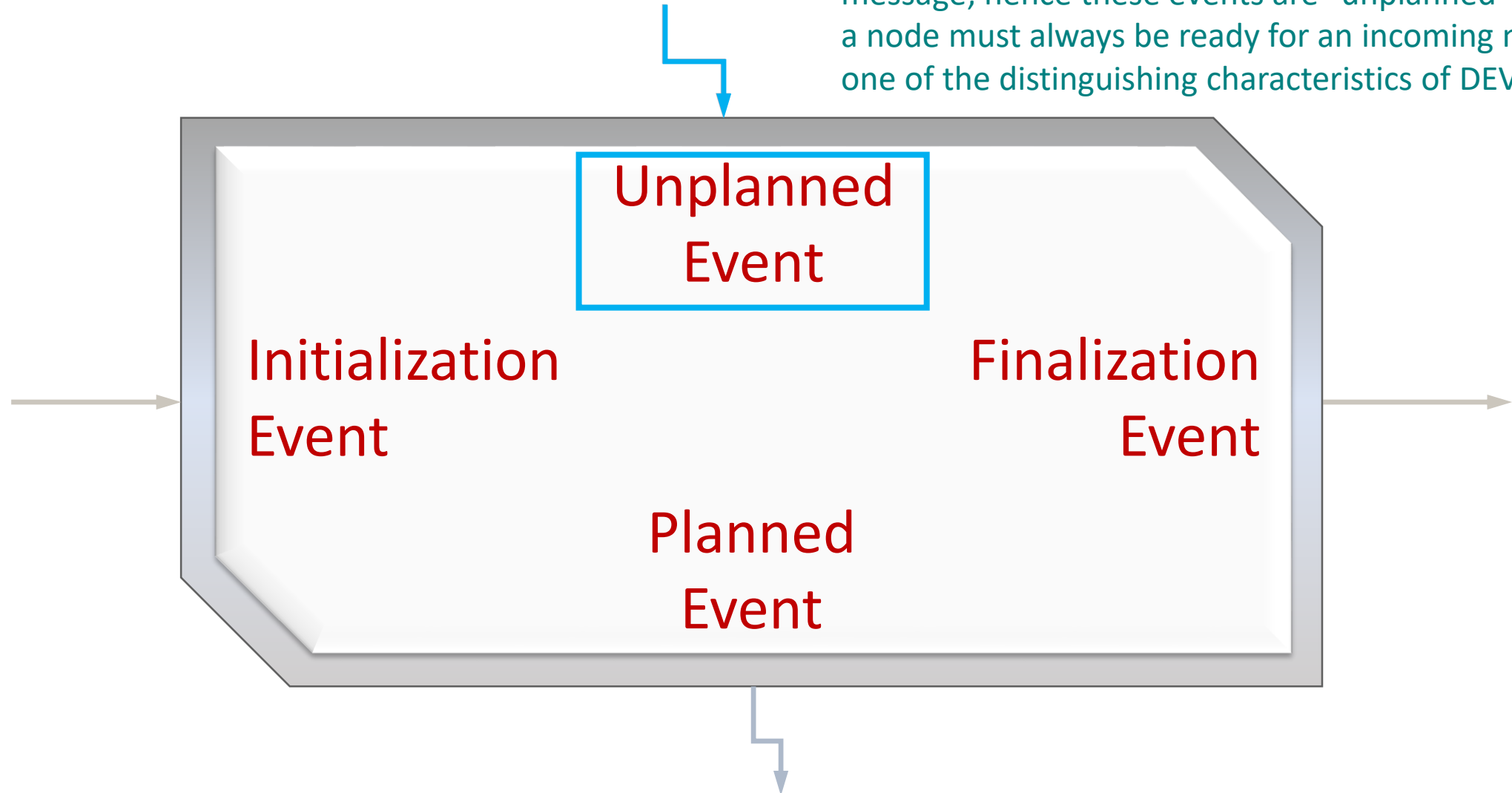
Paradigm

The Initialization Event is invoked once, at the beginning of the simulation or (if the node is an agent in a collection node) when the node is created.



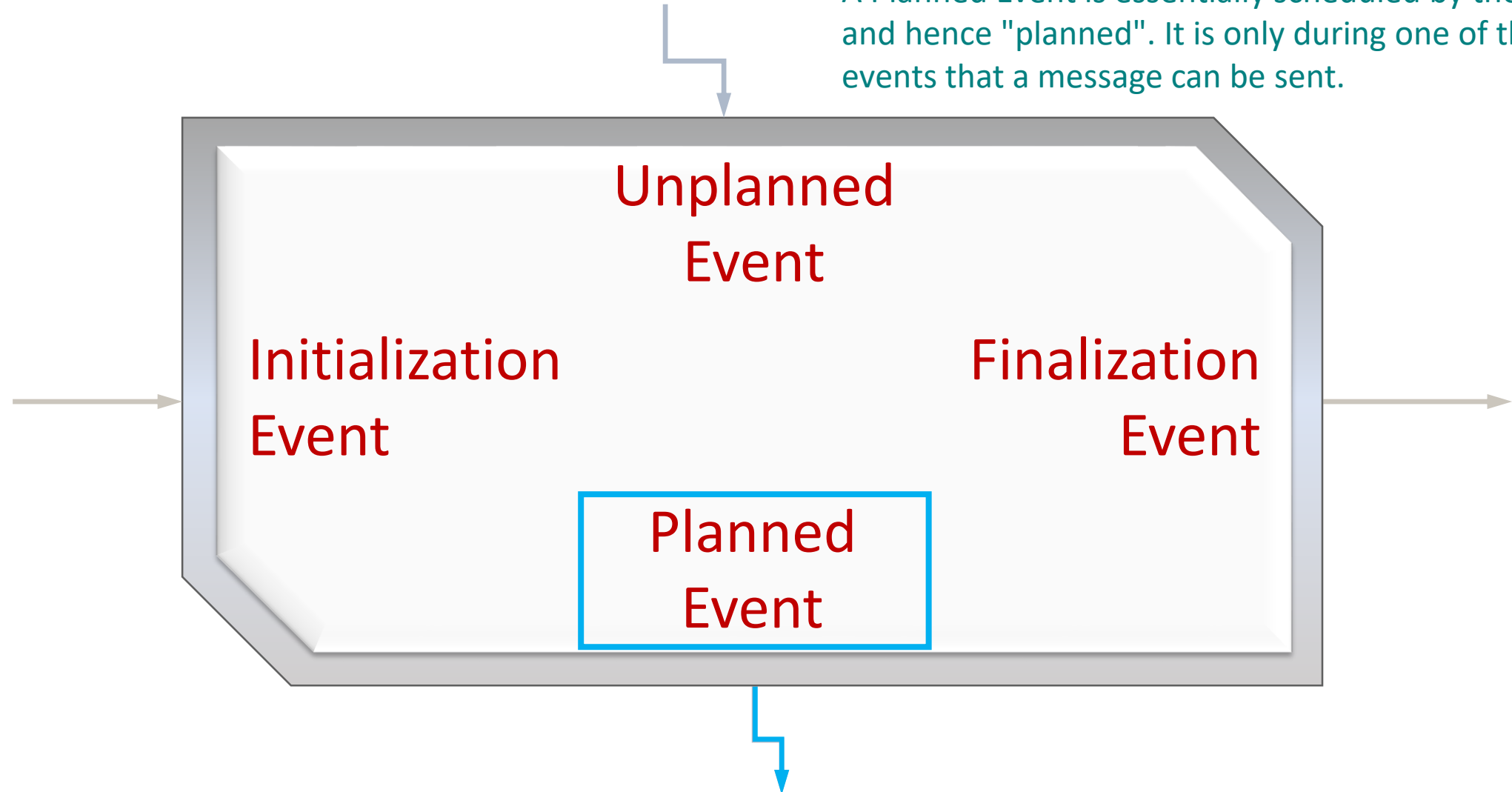
Paradigm

An Unplanned Event is invoked every time a message is received. The node does not know when it will receive a message; hence these events are "unplanned". The fact a node must always be ready for an incoming message is one of the distinguishing characteristics of DEVS.



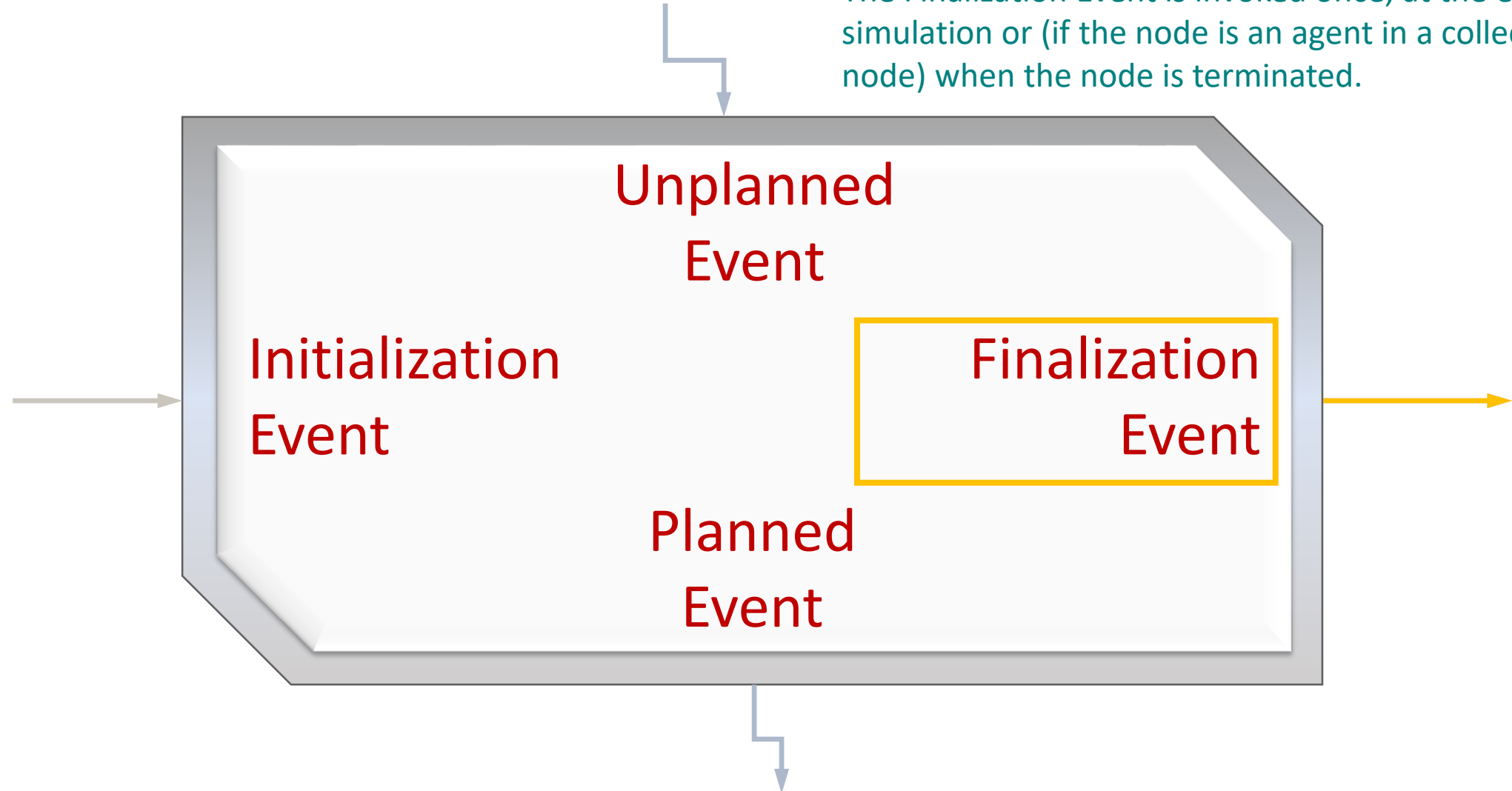
Paradigm

A Planned Event is essentially scheduled by the node, and hence "planned". It is only during one of these events that a message can be sent.



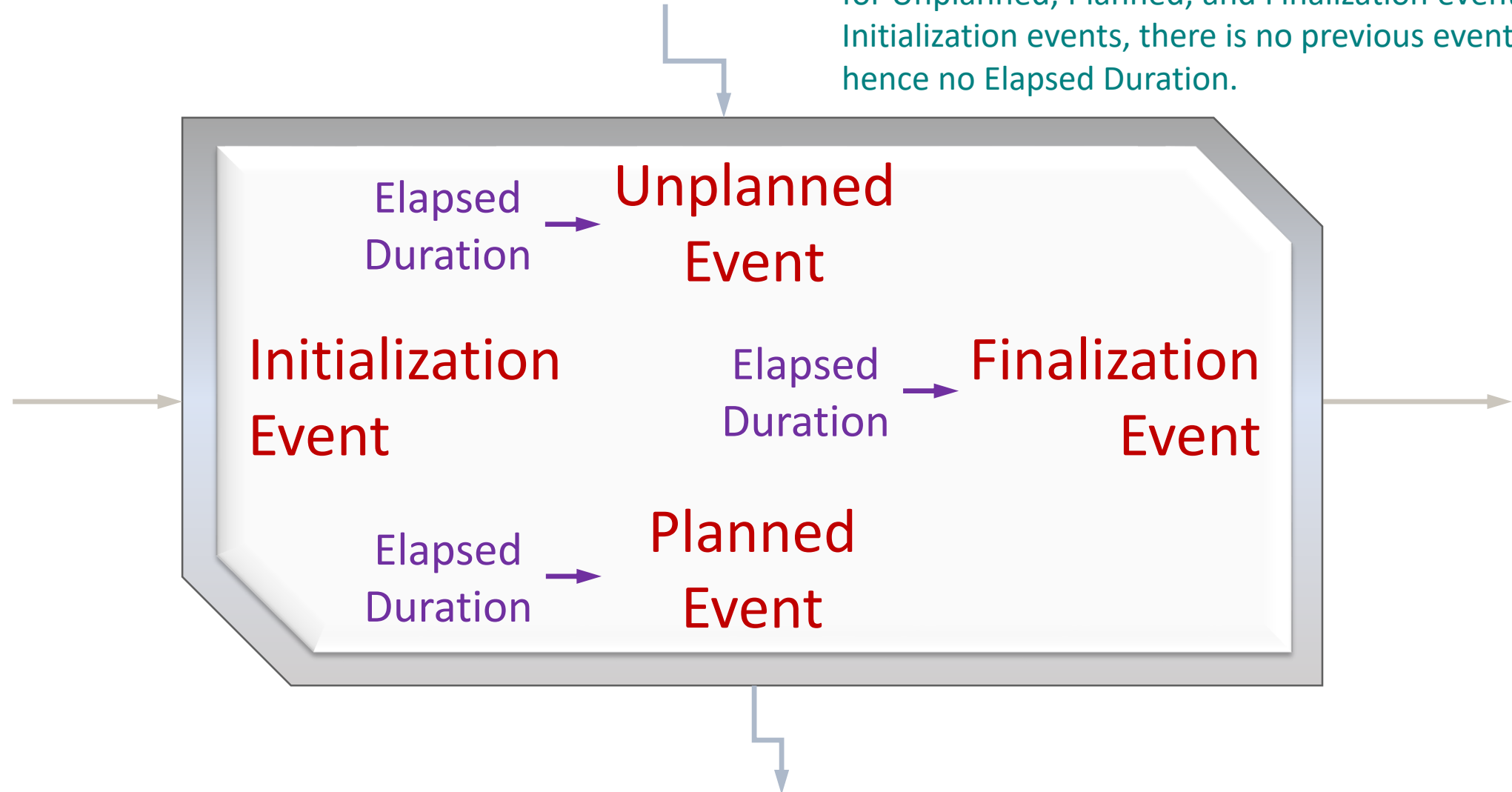
Paradigm

The Finalization Event is invoked once, at the end of the simulation or (if the node is an agent in a collection node) when the node is terminated.



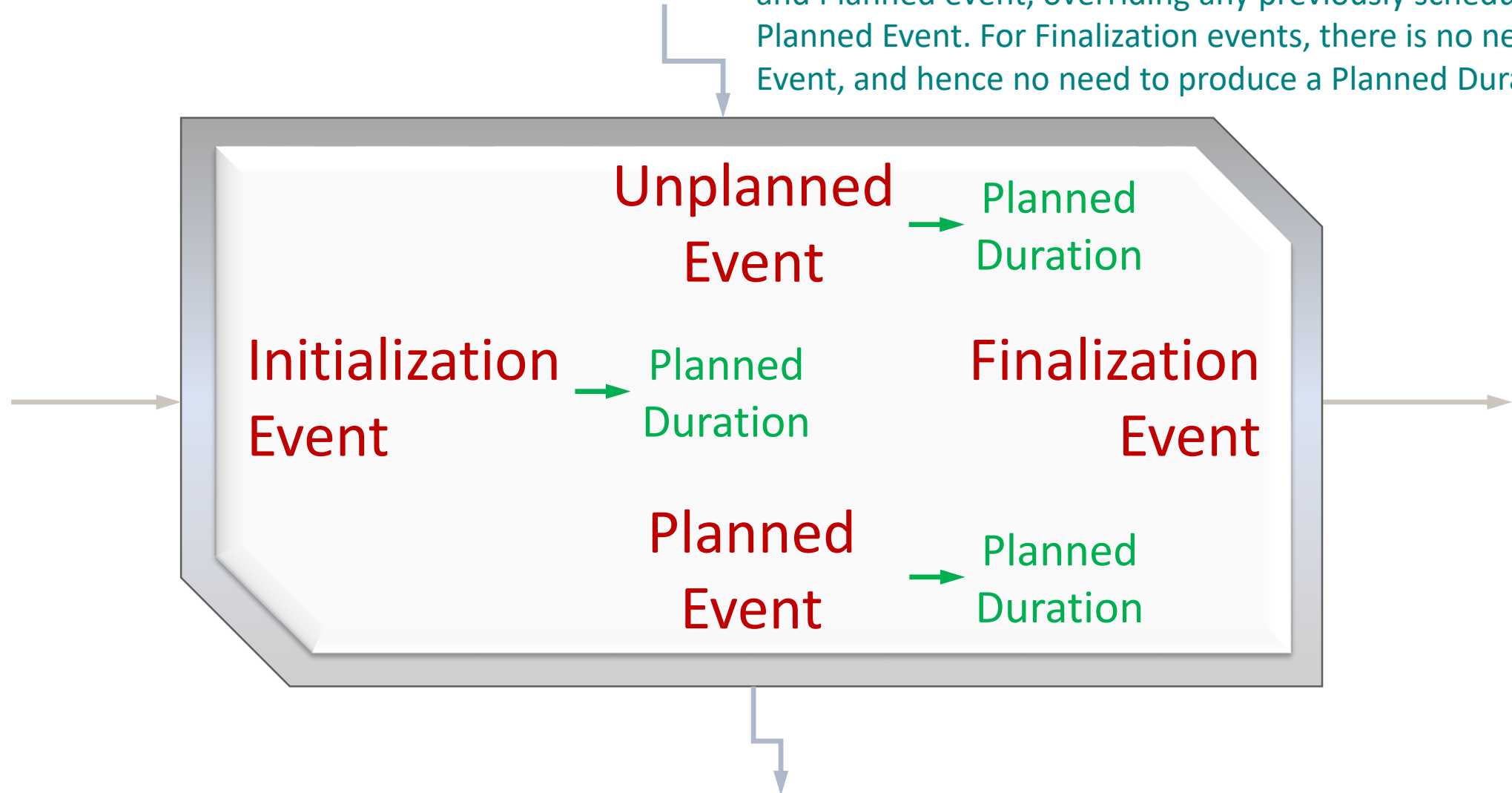
Paradigm

The Elapsed Duration is the time elapsed since the previous event. It is available as a source of information for Unplanned, Planned, and Finalization events. For Initialization events, there is no previous event, and hence no Elapsed Duration.



Paradigm

The Planned Duration is the time before the next scheduled Planned Event. It is produced by every Initialization, Unplanned, and Planned event, overriding any previously scheduled Planned Event. For Finalization events, there is no next Planned Event, and hence no need to produce a Planned Duration.



Paradigm

Nodes

1. Function Node
2. Atomic Node
3. Composite Node
4. Collection Node

Events

1. Initialization Event
2. Unplanned Event
3. Planned Event
4. Finalization Event

Here are lists of the four types of nodes
and four main types of events.

Implementation

Implementation

SyDEVS

Simulation-based analysis of complex systems involving people, devices, physical elements, and dynamic environments.

[View on GitHub](#)

[Home](#)

[Overview](#)

[Getting Started](#)

[API Reference](#)

SyDEVS

Multiscale Simulation and Systems Modeling Library

About

The SyDEVS open source C++ library provides a framework for modeling and simulating complex systems.

In a nutshell, it will help make your simulation code scale.

Using SyDEVS, physics solvers and other simulation models can be implemented as independent nodes, and later integrated. Even nodes that use different time steps (or variable time steps) can be linked together and allowed to interact.

The framework combines 3 modeling paradigms: discrete event simulation, dataflow programming and agent-based modeling. These foundations give SyDEVS the generality needed to support essentially any type of simulation, regardless of domain, time scale, or time advancement scheme.

Documentation

- [Overview](#): Briefly introduces SyDEVS.
- [Getting Started](#): Explains how to use SyDEVS, step by step.
- [API Reference](#): Documents the C++ classes that make up the library.

SyDEVS is an open source C++ library that supports modeling and simulation using the previously described dataflow + DEVS + dataflow paradigm. The main SyDEVS website is at the following URL: <https://autodesk.github.io/sydevs>

Implementation

[Home](#)[Overview](#)[Getting Started](#)[API Reference](#)

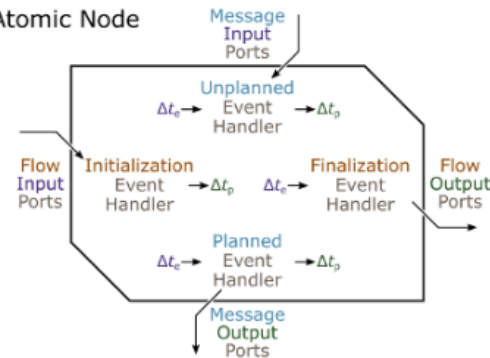
Overview

Brief Introduction to SyDEVS

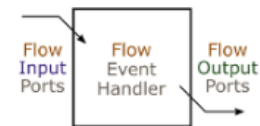
Concept

Using SyDEVS, simulation code is organized into nodes, which can be linked together to form dataflow and simulation networks. The four main types of nodes are illustrated below.

Atomic Node

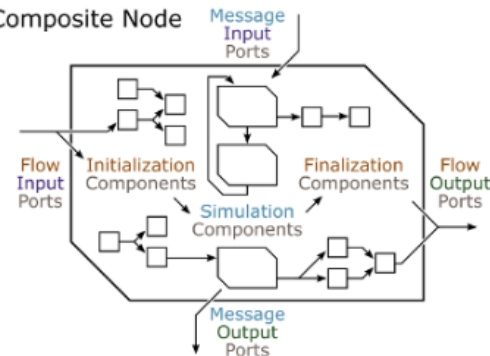


Function Node

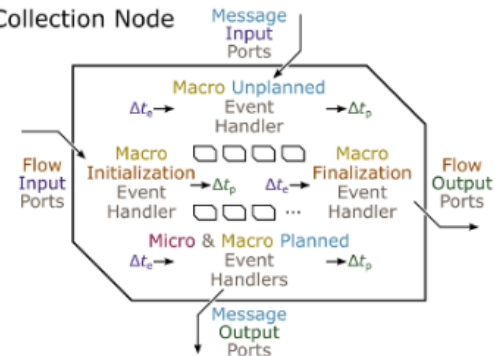


Δt_e - Elapsed Duration
 Δt_p - Planned Duration
□ - Agent

Composite Node



Collection Node



The Overview page on the website illustrates the four types of nodes.

Implementation

[Home](#)[Overview](#)[Getting Started](#)[API Reference](#)[| Prev](#) | [Getting Started – Table of Contents](#) | [Next](#) |

Part 2: Creating your First Simulation

Let's start by adding a few new folders to your `sydevs-examples` project.

1. In `sydevs-examples/src/examples`, create a folder named `getting_started`.
2. In the new `getting_started` folder, create a folder named `waveform`. This is where your first SyDEVS node will be located.
3. In `sydevs-examples/src/simulations` make a folder named `first_simulation`. The code here will invoke the simulation code in `examples/getting_started/waveform`.

The overall directory structure should now be as follows.

```
sydevs-examples/  
  bin/  
  ...  
  external/  
  ...  
  src/  
    examples/  
      getting_started/  
        waveform/  
    simulations/  
      first_simulation/  
      setting_up/
```

The `CMakeLists.txt` file will have to be updated, so let's get that out of the way. Add the following instructions to the `Examples` section. These instructions prepare a list of the header (.h) files you will later create in the `waveform` folder.

```
set(WAVEFORM_DIR ${EXAMPLES_DIR}/getting_started/waveform)  
file(GLOB WAVEFORM_HDRS "${WAVEFORM_DIR}/*.h")
```

The Getting Started tutorial guides users through the process of setting up a SyDEVS project and running simulations.

Implementation

SyDEVS v0.4.2

Multiscale Simulation and Systems Modeling Library

Main Page

Namespaces

Classes

Files

SyDEVS

API Reference Overview

Namespaces

Classes

Files

API Reference Overview

About SyDEVS

This library provides a framework for implementing complex systems analysis and simulation code in a modular/hierarchical fashion. It was originally developed to serve as a backend for the visual programming interfaces described by Maleki et al. (2015), but the same functionality can be achieved without a GUI by defining C++ classes that derive from one of the system node base classes (`atomic_node`, `composite_node`, `collection_node`, `function_node`). The framework combines two programming paradigms: dataflow programming as exemplified by Autodesk's *Dynamo* tool, and the *DEVS* message-passing paradigm implemented in tools such as *DesignDEVS* (see software, conference paper, journal paper). These foundations give the framework the generality needed to support essentially any type of simulation, regardless of domain, time scale, or time advancement scheme.

Main Classes

Namespace: `sydevs`

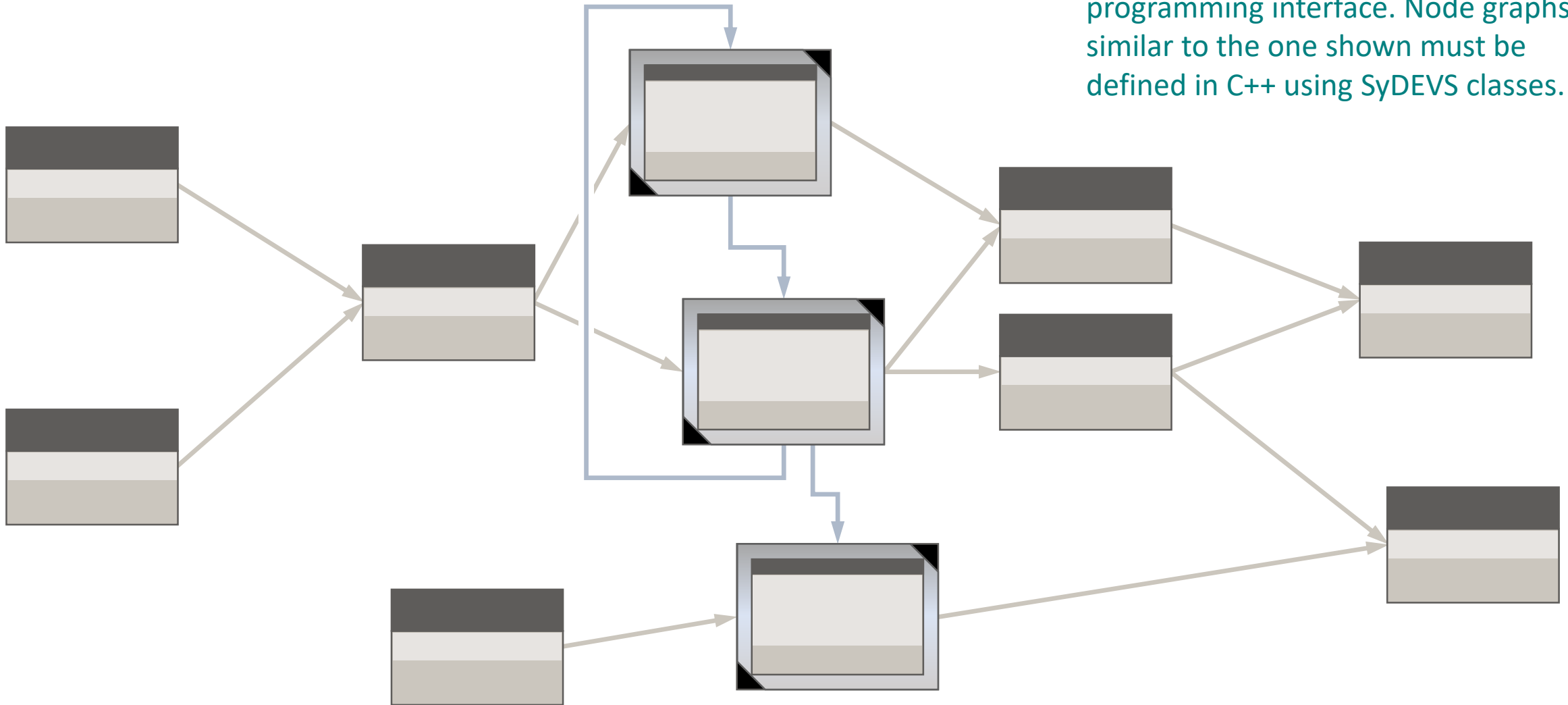
- Core Classes (`sydevs/core`) - generic classes for a variety of applications
 - `scale` (`scale.h`) - dimensionless power of 1000
 - `number_types.h` - related header file with number type aliases, pi constant
 - `quantity` (`quantity.h`) - Standard International (SI) quantity (e.g. mass, acceleration)
 - `units` (`units.h`) - related template for SI units (e.g. grams, meters/second²)
 - `arraynd` (`arraynd.h`) - multidimensional array
 - `range` (`range.h`) - related class representing range of array indices
 - `core_type` (`core_types.h`) - traits for types exchanged between system nodes
 - `pointer` (`pointer.h`) - related class for pointers to any type of data
 - `string_builder` (`string_builder.h`) - related class for value-to-string conversion
- Time Classes (`sydevs/time`) - multiscale time representation
 - `time_point` (`time_point.h`) - arbitrary-precision point in time
 - `time_sequence` (`time_sequence.h`) - sequence of increasing time points
 - `time_queue` (`time_queue.h`) - data structure tracking future event times
 - `time_cache` (`time_cache.h`) - data structure tracking past event times
- Systems Classes (`sydevs/systems`) - dataflow + message-passing networks
 - `system_node` (`system_node.h`) - base class for all nodes
 - `atomic_node` (`atomic_node.h`) - derived from `system_node`, supports event handlers
 - `composite_node` (`composite_node.h`) - derived from `system_node`, supports fixed-structure compositions
 - `collection_node` (`collection_node.h`) - derived from `system_node`, supports variable-length collections
 - `function_node` (`function_node.h`) - derived from `system_node`, supports functions
 - `parameter_node` (`parameter_node.h`) - derived from `function_node`, handles parameter values
 - `statistic_node` (`statistic_node.h`) - derived from `function_node`, handles statistic values
 - `port<data_mode, data_goal>` (`port.h`) - related classes for node ports
 - `simulation` (`simulation.h`) - template for simulations based on a port-free `system_node`
 - `discrete_event_time` (`discrete_event_time.h`) - related class tracking progress through a simulation

Generated on Wed May 2 2018 20:30:44 for SyDEVS by **doxygen** 1.8.6

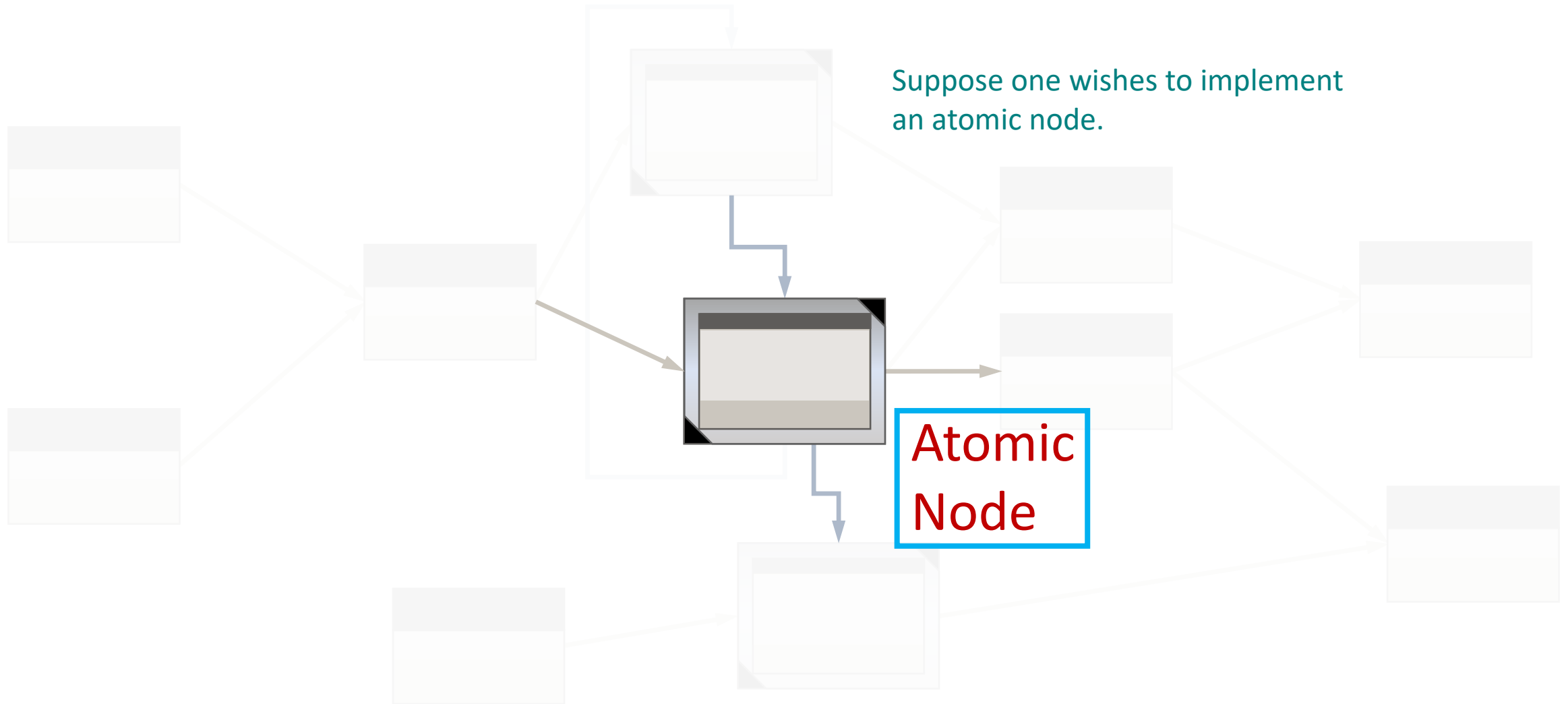
There is also an API Reference.

Implementation

At present, there is no visual programming interface. Node graphs similar to the one shown must be defined in C++ using SyDEVS classes.



Implementation



Implementation

They would then write a C++ class that inherits from the `atomic_node` base class provided by the SyDEVS library.

```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // Attributes:
    virtual scale time_precision() const { return micro; }

    // Ports:
    port<flow, input, duration> serv_dt_input;    // service duration
    port<message, input, int64> job_id_input;      // job ID (input)
    port<message, output, int64> job_id_output;    // job ID (output)
    port<flow, output, duration> idle_dt_output;  // idle duration

protected:
    // State Variables:
    duration serv_dt;    // service duration (constant)
    std::vector<int64> Q; // queue of IDs of jobs waiting to be processed
    duration idle_dt;    // idle duration (accumulating)
    duration planned_dt; // planned duration

    // Event Handlers:
    virtual duration initialization_event();
    virtual duration unplanned_event(duration elapsed_dt);
    virtual duration planned_event(duration elapsed_dt);
    virtual void finalization_event(duration elapsed_dt);
};
```


Implementation

Observe there are four types of ports.

```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

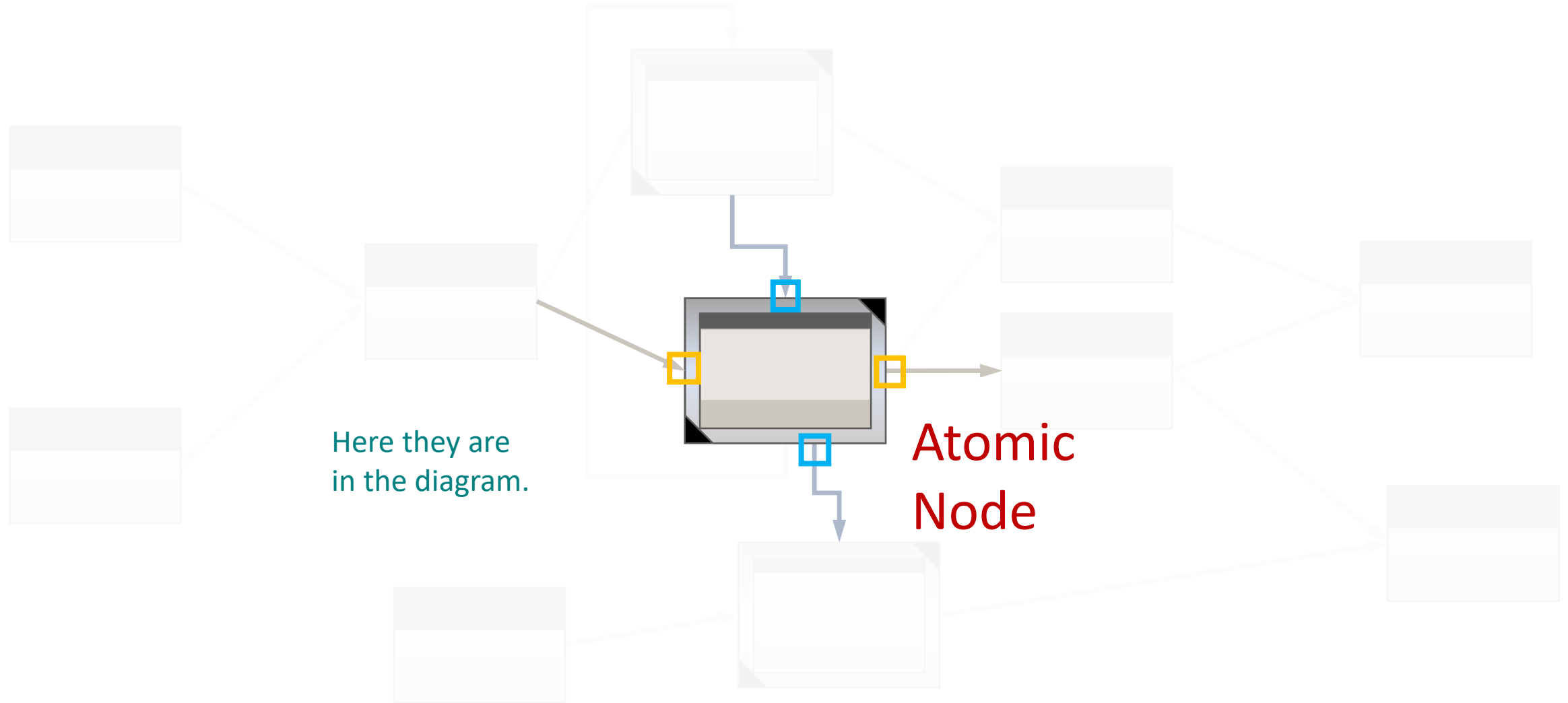
    // Attributes:
    virtual scale time_precision() const { return micro; }

    // Ports:
    port<flow, input, duration> serv_dt_input;    // service duration
    port<message, input, int64> job_id_input;    // job ID (input)
    port<message, output, int64> job_id_output;    // job ID (output)
    port<flow, output, duration> idle_dt_output;    // idle duration

protected:
    // State Variables:
    duration serv_dt;    // service duration (constant)
    std::vector<int64> Q;    // queue of IDs of jobs waiting to be processed
    duration idle_dt;    // idle duration (accumulating)
    duration planned_dt;    // planned duration

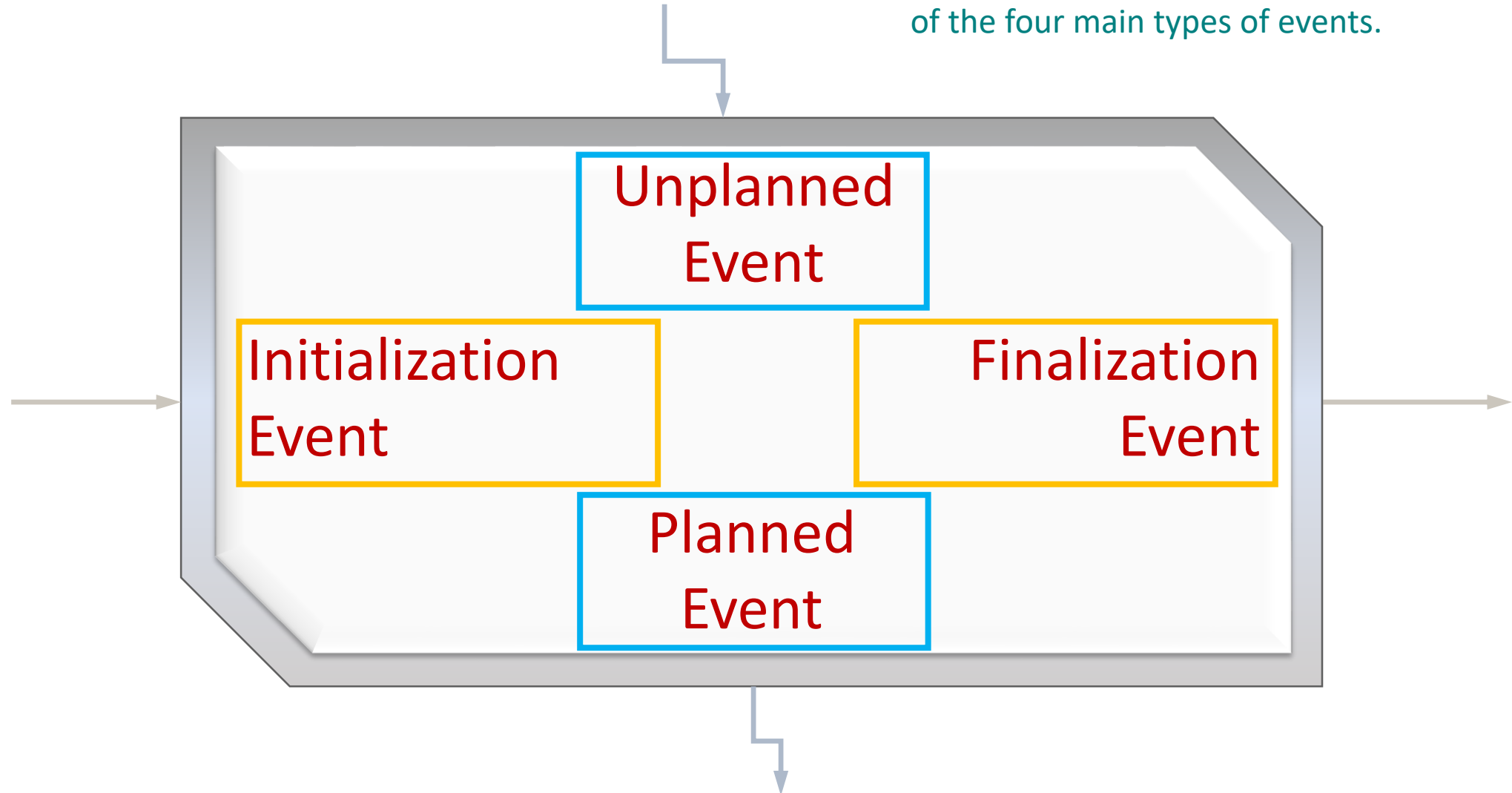
    // Event Handlers:
    virtual duration initialization_event();
    virtual duration unplanned_event(duration elapsed_dt);
    virtual duration planned_event(duration elapsed_dt);
    virtual void finalization_event(duration elapsed_dt);
};
```

Implementation



Implementation

Each type of port is associated with one of the four main types of events.



Implementation

```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // Attributes:
    virtual scale time_precision() const { return micro; }

    // Ports:
    port<flow, input, duration> serv_dt_input;    // service duration
    port<message, input, int64> job_id_input;      // job ID (input)
    port<message, output, int64> job_id_output;    // job ID (output)
    port<flow, output, duration> idle_dt_output;  // idle duration

protected:
    // State Variables:
    duration serv_dt;    // service duration (constant)
    std::vector<int64> Q; // queue of IDs of jobs waiting to be processed
    duration idle_dt;    // idle duration (accumulating)
    duration planned_dt; // planned duration

    // Event Handlers:
    virtual duration initialization_event();
    virtual duration unplanned_event(duration elapsed_dt);
    virtual duration planned_event(duration elapsed_dt);
    virtual void finalization_event(duration elapsed_dt);
};
```

The code to be executed for each type of event is placed in these four member functions.

Implementation

```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // Attributes:
    virtual scale time_precision() const { return micro; }

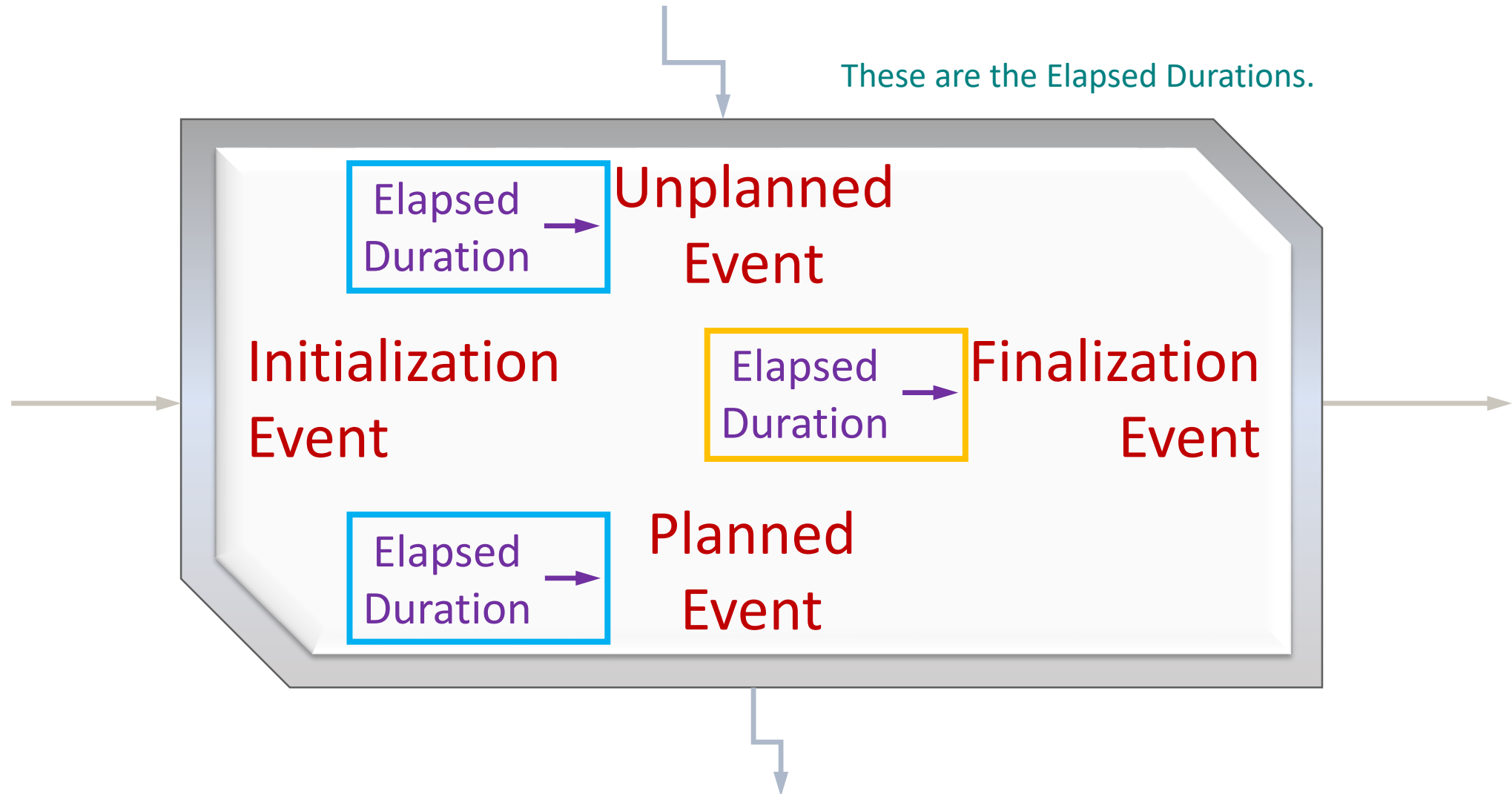
    // Ports:
    port<flow, input, duration> serv_dt_input;    // service duration
    port<message, input, int64> job_id_input;      // job ID (input)
    port<message, output, int64> job_id_output;    // job ID (output)
    port<flow, output, duration> idle_dt_output;  // idle duration

protected:
    // State Variables:
    duration serv_dt;    // service duration (constant)
    std::vector<int64> Q; // queue of IDs of jobs waiting to be processed
    duration idle_dt;    // idle duration (accumulating)
    duration planned_dt; // planned duration

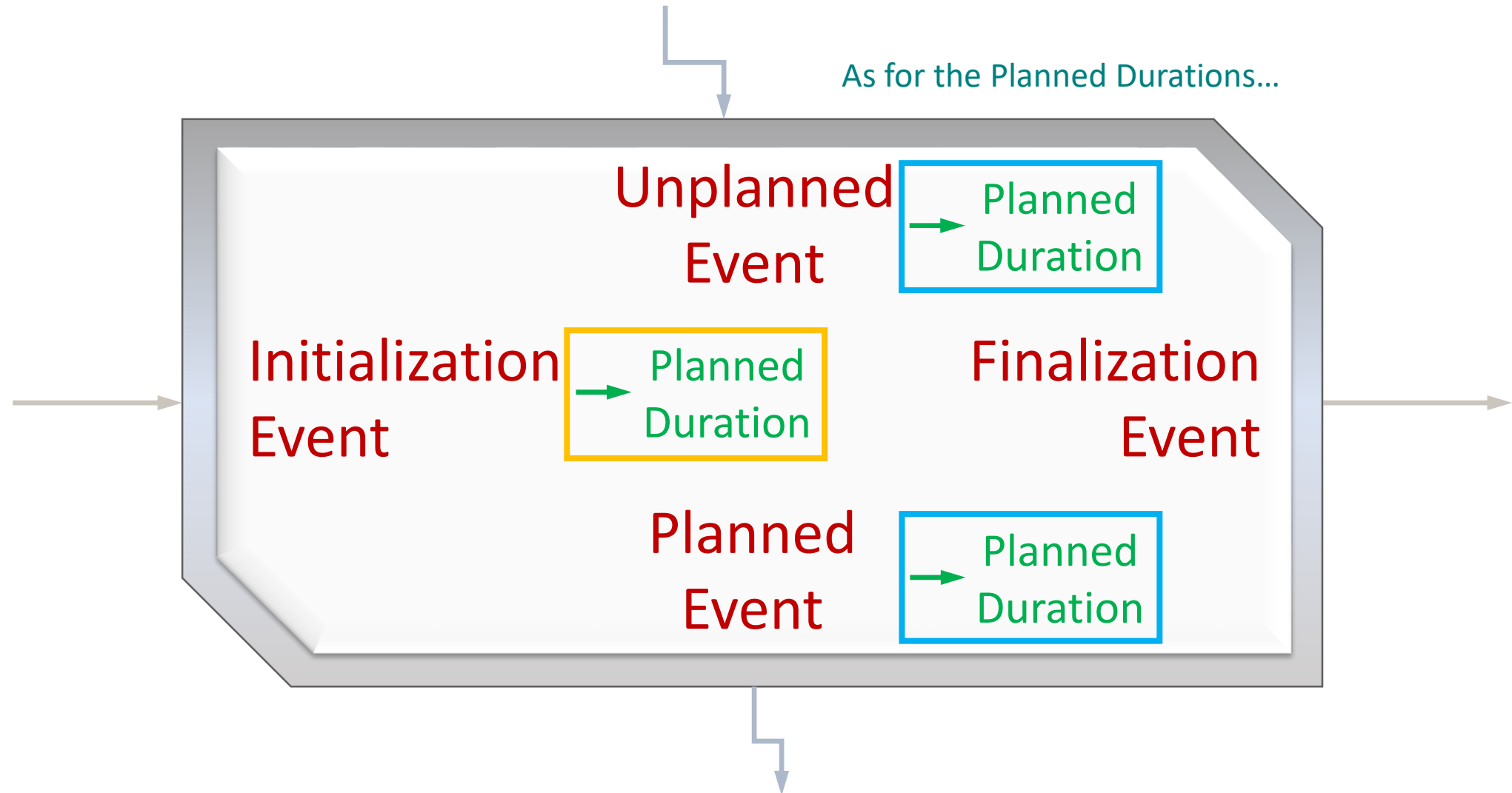
    // Event Handlers:
    virtual duration initialization_event();
    virtual duration unplanned_event(duration elapsed_dt);
    virtual duration planned_event(duration elapsed_dt);
    virtual void finalization_event(duration elapsed_dt);
};
```

Observe that three of the functions have time duration arguments.

Implementation



Implementation



Implementation

```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // Attributes:
    virtual scale time_precision() const { return micro; }

    // Ports:
    port<flow, input, duration> serv_dt_input;    // service duration
    port<message, input, int64> job_id_input;      // job ID (input)
    port<message, output, int64> job_id_output;    // job ID (output)
    port<flow, output, duration> idle_dt_output;  // idle duration

protected:
    // State Variables:
    duration serv_dt;    // service duration (constant)
    std::vector<int64> Q; // queue of IDs of jobs waiting to be processed
    duration idle_dt;    // idle duration (accumulating)
    duration planned_dt; // planned duration

    // Event Handlers:
    virtual duration initialization_event();
    virtual duration unplanned_event(duration elapsed_dt);
    virtual duration planned_event(duration elapsed_dt);
    virtual void finalization_event(duration elapsed_dt);
};
```

Planned Durations are produced by three of the functions as return values.

For more information, visit the SyDEVs website:

<https://autodesk.github.io/sydevs>