SyDEVS Introduction

Theory – Paradigm – Implementation

Autodesk Research

December 2024

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 discrete event simulation, dataflow programming, and agent-based modeling, 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.

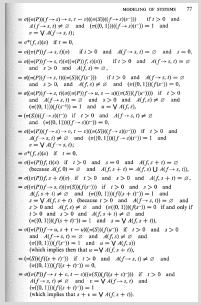
Theory

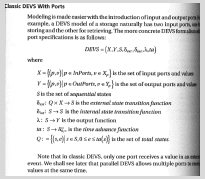
Theory

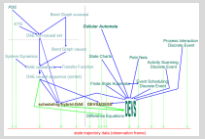
SyDEVS is based on theory that dates back to the late 1960s.

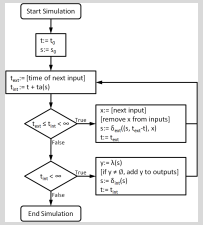
Theory

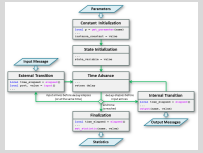
Here is a small sample of related publications from the last 50 years.

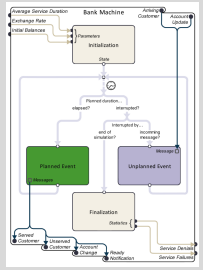






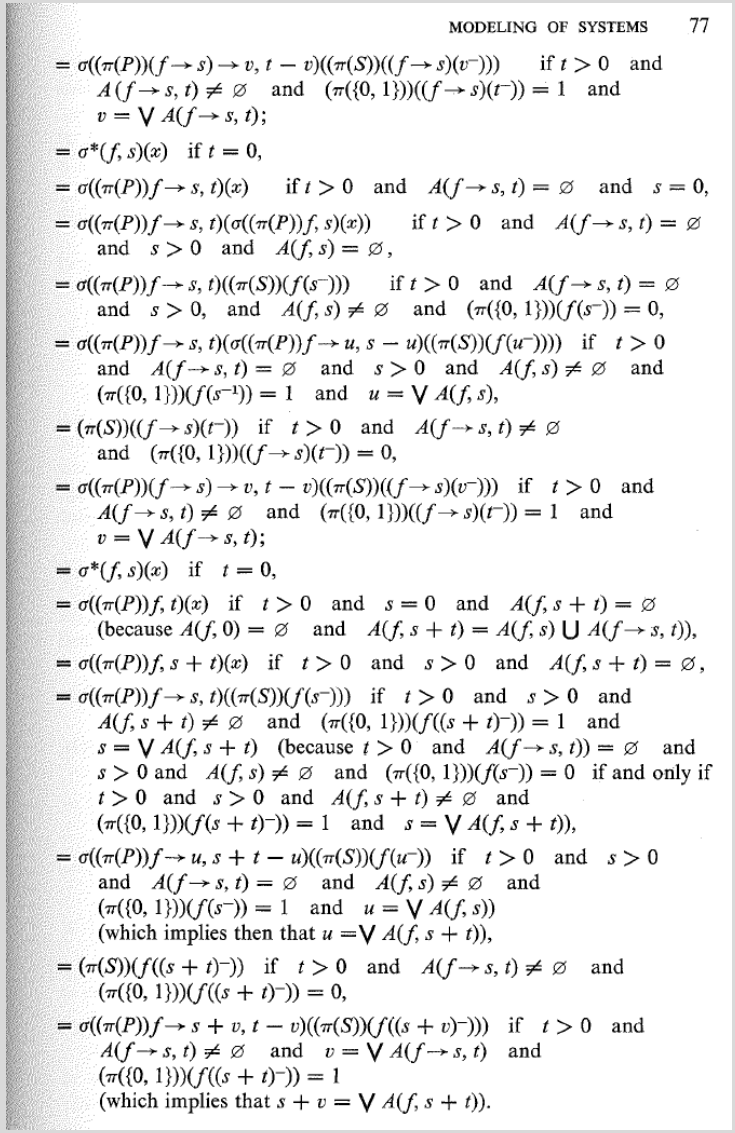






1970 1980 1990 2000 2010 2020

Theory



MODELING OF SYSTEMS 77

$= \sigma((\pi(P))(f \to s) \to v, t - v)((\pi(S))((f \to s)(v^-)))$ if $t > 0$ and $A(f \to s, t) \neq \varnothing$ and $(\pi(\{0, 1\}))((f \to s)(t^-)) = 1$ and $v = \bigvee A(f \to s, t)$;

$= \sigma^*(f, s)(x)$ if $t = 0$,

$= \sigma((\pi(P))f \to s, t)(x)$ if $t > 0$ and $A(f \to s, t) = \varnothing$ and $s = 0$,

$= \sigma((\pi(P))f \to s, t)(\sigma((\pi(P))f, s)(x))$ if $t > 0$ and $A(f \to s, t) = \varnothing$ and $s > 0$ and $A(f, s) = \varnothing$,

$= \sigma((\pi(P))f \to s, t)((\pi(S))(f(s^-)))$ if $t > 0$ and $A(f \to s, t) = \varnothing$ and $s > 0$, and $A(f, s) \neq \varnothing$ and $(\pi(\{0, 1\}))(f(s^-)) = 0$,

$= \sigma((\pi(P))f \to s, t)(\sigma((\pi(P))f \to u, s - u)((\pi(S))(f(u^-))))$ if $t > 0$ and $A(f \to s, t) = \varnothing$ and $s > 0$ and $A(f, s) \neq \varnothing$ and $(\pi(\{0, 1\}))(f(s^{-1})) = 1$ and $u = \bigvee A(f, s)$,

$= (\pi(S))((f \to s)(t^-))$ if $t > 0$ and $A(f \to s, t) \neq \varnothing$ and $(\pi(\{0, 1\}))((f \to s)(t^-)) = 0$,

$= \sigma((\pi(P))(f \to s) \to v, t - v)((\pi(S))((f \to s)(v^-)))$ if $t > 0$ and $A(f \to s, t) \neq \varnothing$ and $(\pi(\{0, 1\}))((f \to s)(t^-)) = 1$ and $v = \bigvee A(f \to s, t)$;

$= \sigma^*(f, s)(x)$ if $t = 0$,

$= \sigma((\pi(P))f, t)(x)$ if $t > 0$ and $s = 0$ and $A(f, s + t) = \varnothing$ (because $A(f, 0) = \varnothing$ and $A(f, s + t) = A(f, s) \cup A(f \to s, t)$),

$= \sigma((\pi(P))f, s + t)(x)$ if $t > 0$ and $s > 0$ and $A(f, s + t) = \varnothing$,

$= \sigma((\pi(P))f \to s, t)((\pi(S))(f(s^-)))$ if $t > 0$ and $s > 0$ and $A(f, s + t) \neq \varnothing$ and $(\pi(\{0, 1\}))(f((s + t)^-)) = 1$ and $s = \bigvee A(f, s + t)$ (because $t > 0$ and $A(f \to s, t)) = \varnothing$ and $s > 0$ and $A(f, s) \neq \varnothing$ and $(\pi(\{0, 1\}))(f(s^-)) = 0$ if and only if $t > 0$ and $s > 0$ and $A(f, s + t) \neq \varnothing$ and $(\pi(\{0, 1\}))(f(s + t)^-)) = 1$ and $s = \bigvee A(f, s + t)$),

$= \sigma((\pi(P))f \to u, s + t - u)((\pi(S))(f(u^-))$ if $t > 0$ and $s > 0$ and $A(f \to s, t) = \varnothing$ and $A(f, s) \neq \varnothing$ and $(\pi(\{0, 1\}))(f(s^-)) = 1$ and $u = \bigvee A(f, s)$ (which implies then that $u = \bigvee A(f, s + t)$),

$= (\pi(S))(f((s + t)^-))$ if $t > 0$ and $A(f \to s, t) \neq \varnothing$ and $(\pi(\{0, 1\}))(f((s + t)^-)) = 0$,

$= \sigma((\pi(P))f \to s + v, t - v)((\pi(S))(f((s + v)^-)))$ if $t > 0$ and $A(f \to s, t) \neq \varnothing$ and $v = \bigvee A(f \to s, t)$ and $(\pi(\{0, 1\}))(f((s + t)^-)) = 1$ (which implies that $s + v = \bigvee A(f, s + t)$).

A. Wayne Wymore

A Mathematical Theory of Systems Engineering

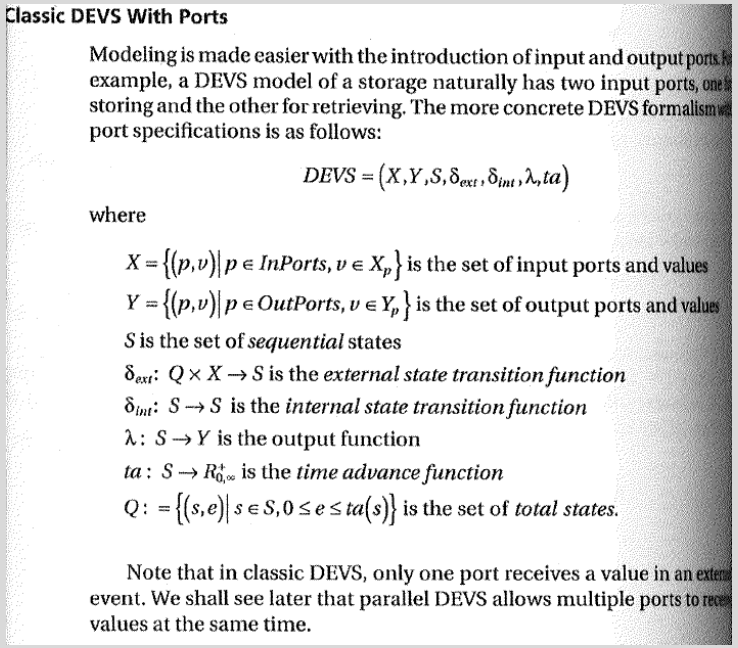
1967

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

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

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

δ_{int}: $S \to S$ is the *internal state transition function*

λ: $S \to Y$ is the output function

ta: $S \to R^{+}_{0,\infty}$ is the *time advance function*

Q: $= \{(s, e) \mid s \in S, 0 \le e \le ta(s)\}$ 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

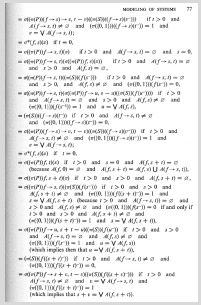
He found that essentially all simulations can be represented in a common form based on the discrete event simulation paradigm. Zeigler named this common form "DEVS".

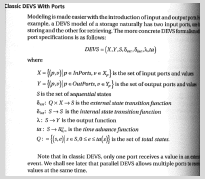
Bernard P. Zeigler
Theory of Modeling and Simulation

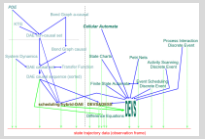
1976

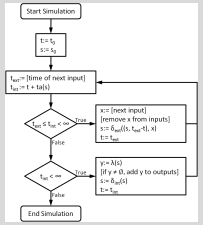
$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

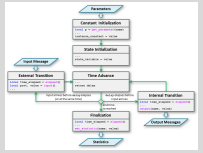
Theory

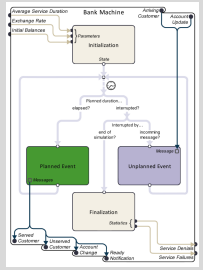












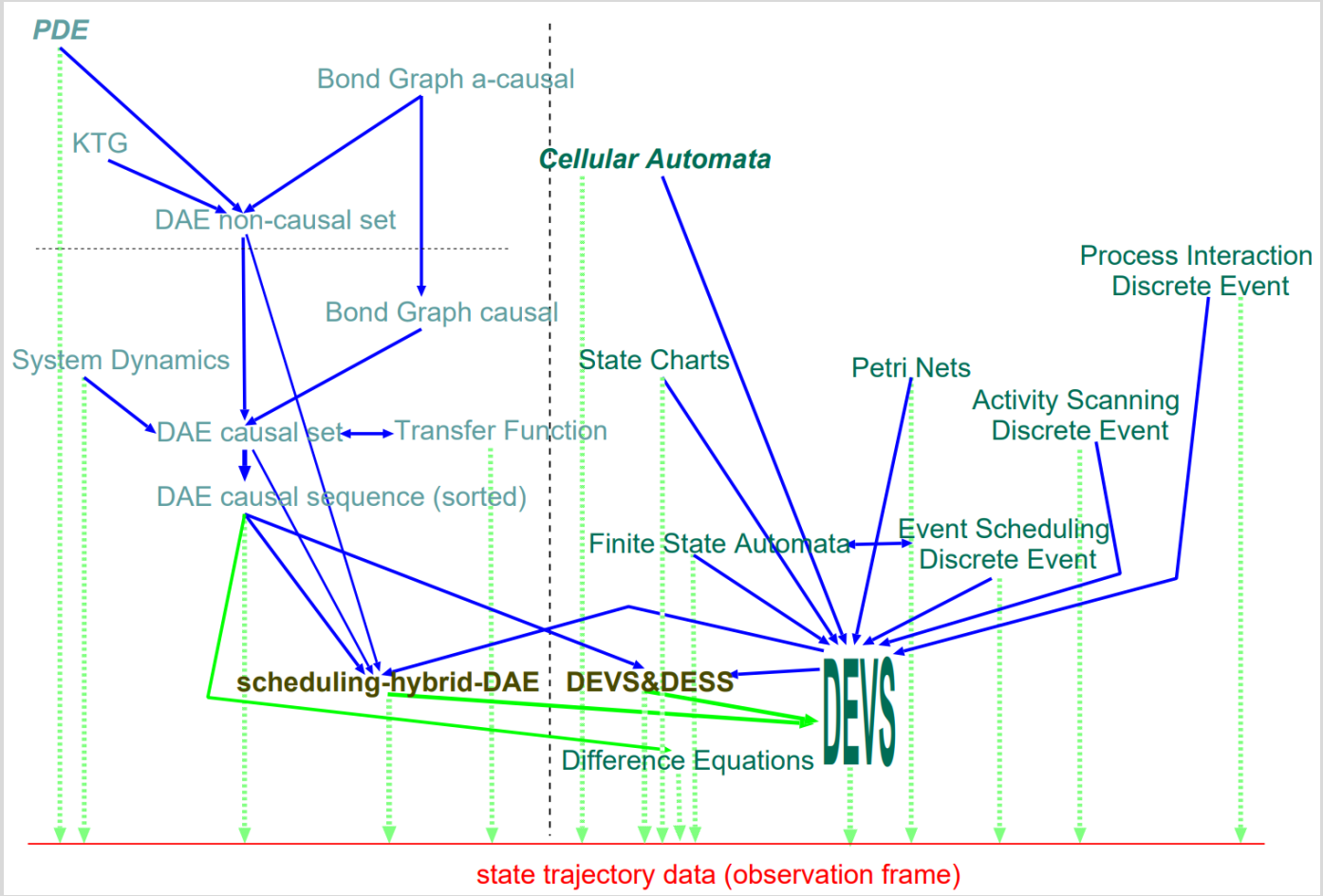
1970 1980 1990 2000 2010 2020

Theory

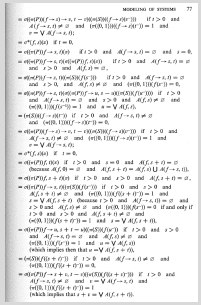
Over the years, the generality of DEVS was confirmed. It was found that for each of the most common modeling paradigms, any model expressed in that paradigm could also be represented using DEVS.

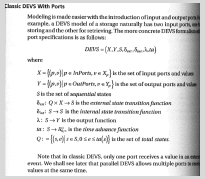
Hans L. M. Vangheluwe
DEVS as a Common Denominator…

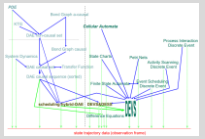
2000

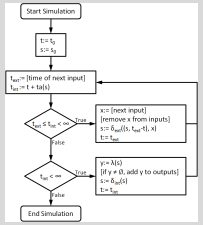


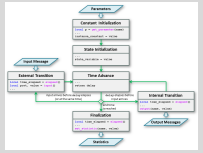
Theory

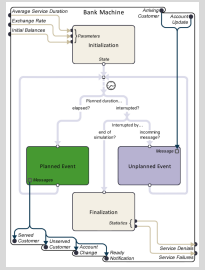


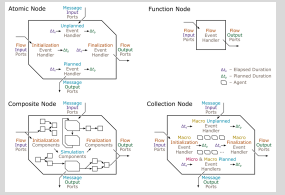












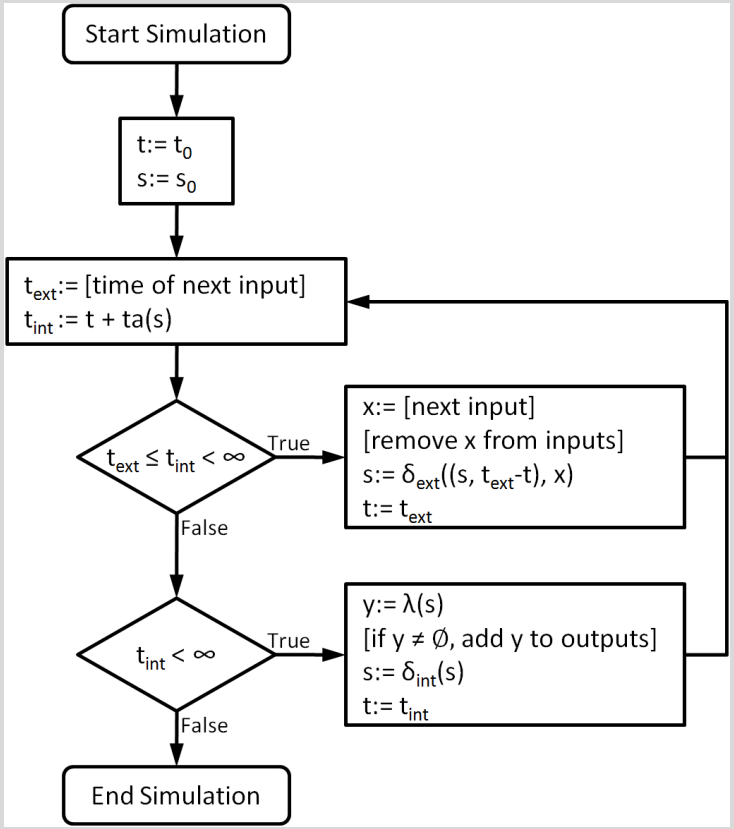
1970 1980 1990 2000 2010 2020

Theory

Autodesk Research & Carleton University

Formal Languages for Computer Simulation

2011

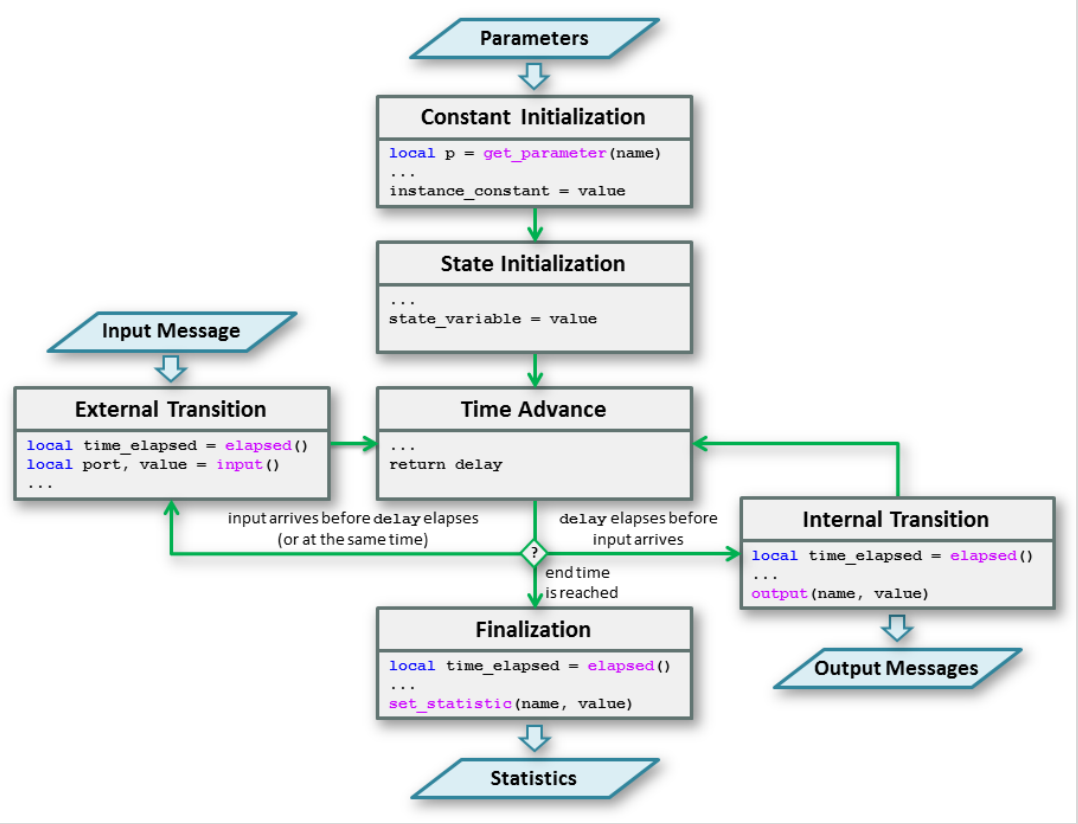


In 2011, researchers at Autodesk began exploring how to make DEVS more approachable to scientific and engineering communities.

Theory

Autodesk Research
DesignDEVS Help Graphic

2012



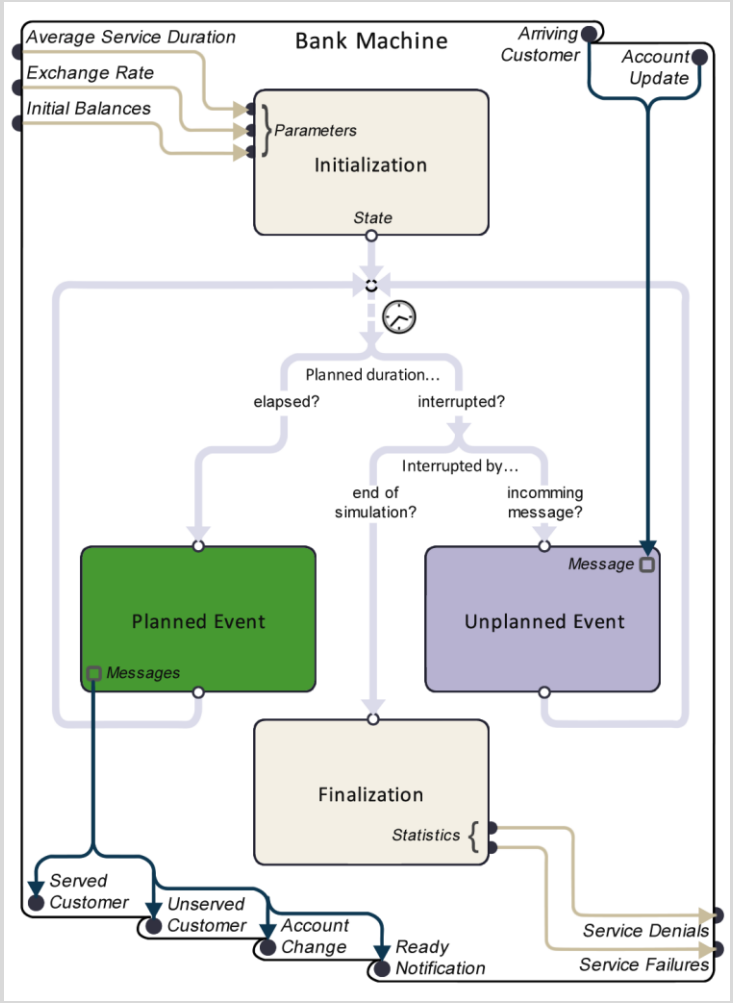
The process involved several iterations.

Theory

Autodesk Research & Simon Fraser University

Designing DEVS Visual Interfaces…

2015



At one point, a set of visual interfaces was designed along with a new way of expressing DEVS.

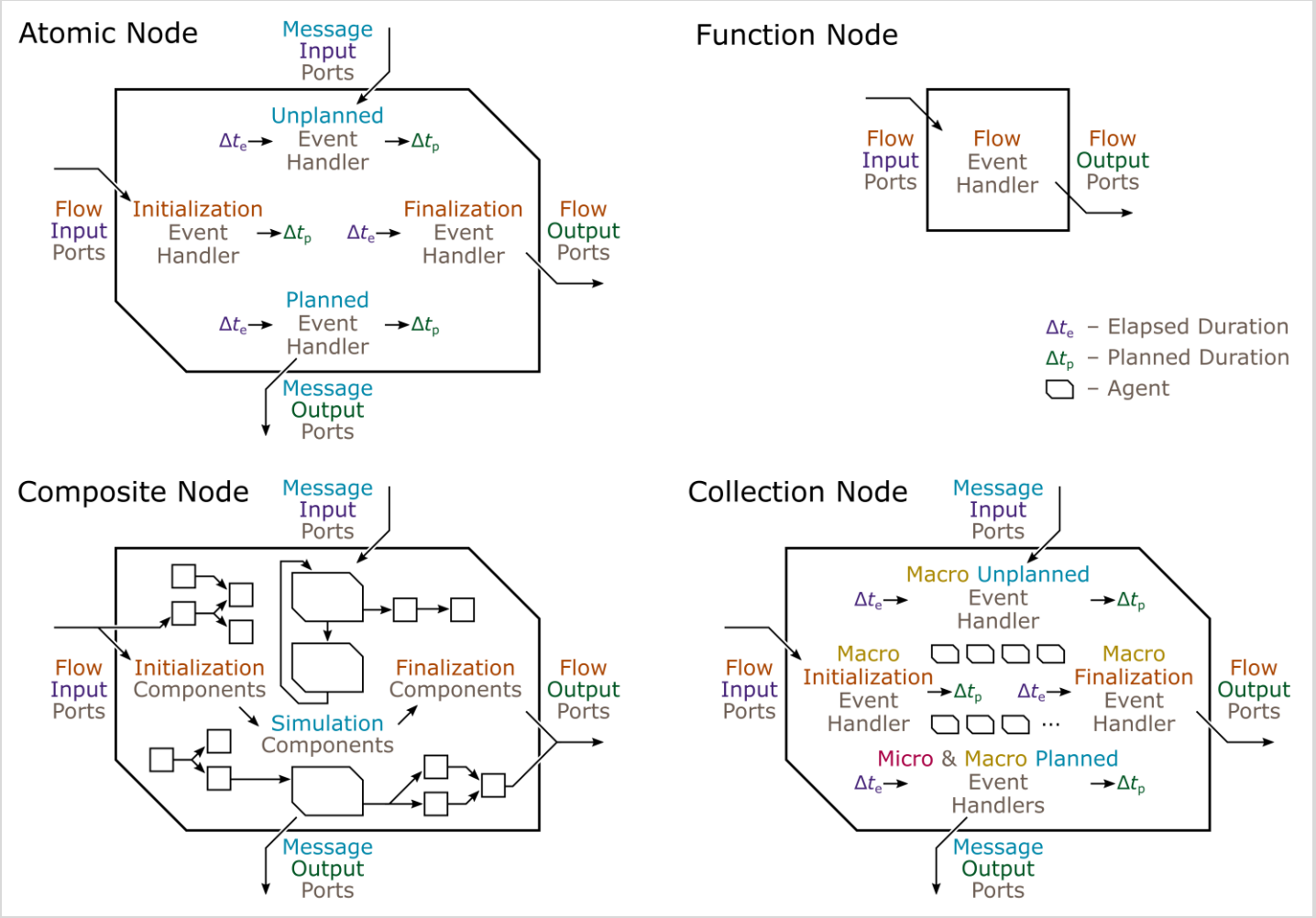
Theory

Eventually, a new variant of DEVS called “Symmetric DEVS” was published, combining discrete event simulation with dataflow programming and agent-based modeling.

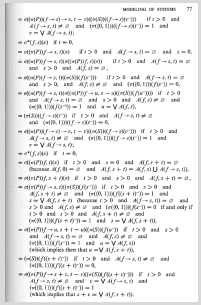
Autodesk Research

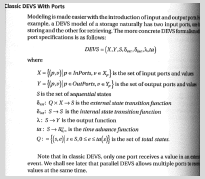
A Symmetric Formalism for Discrete Event Simulation…

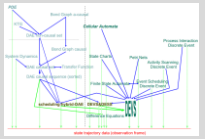
2018

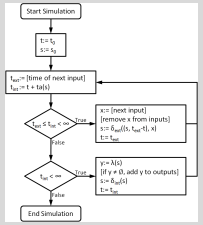


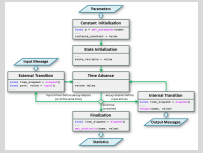
Theory

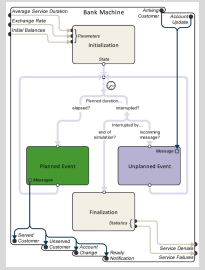


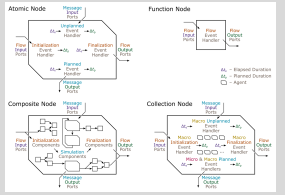










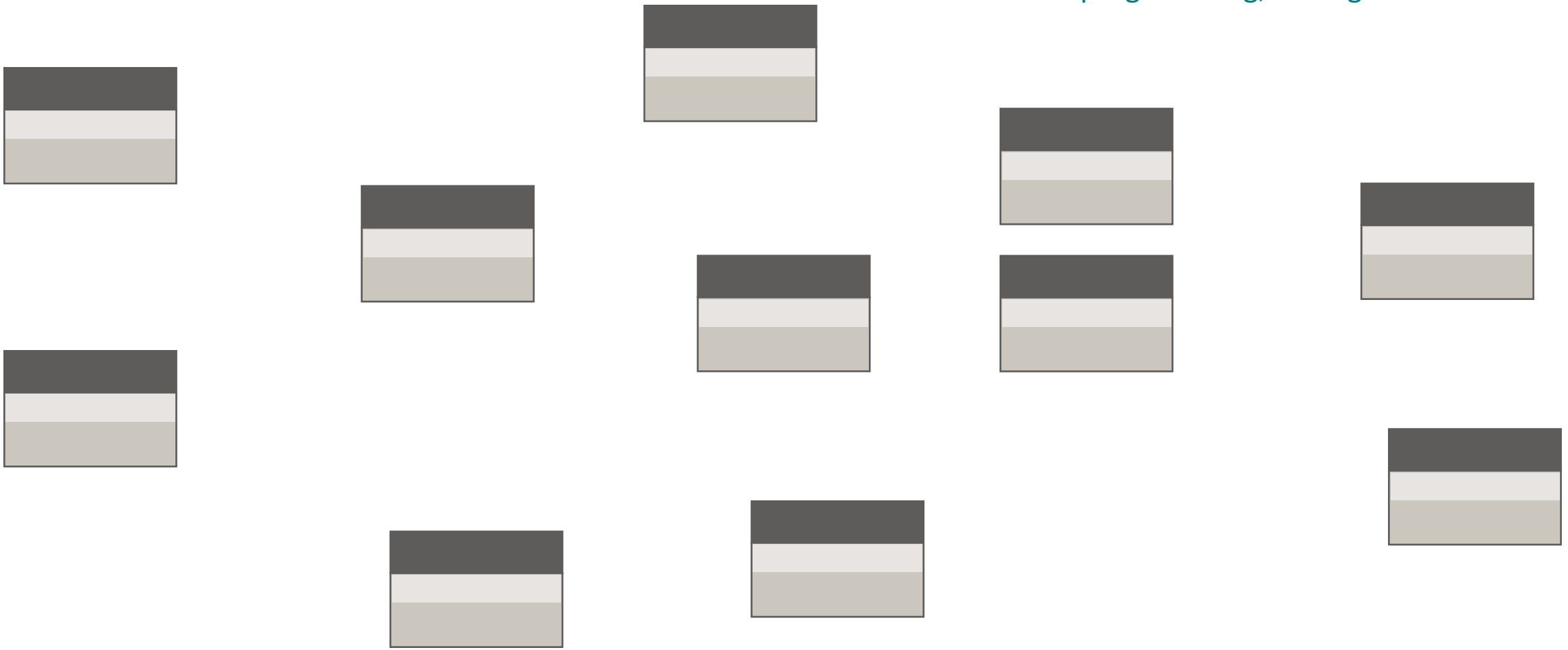


1970 1980 1990 2000 2010 2020

Paradigm

Paradigm

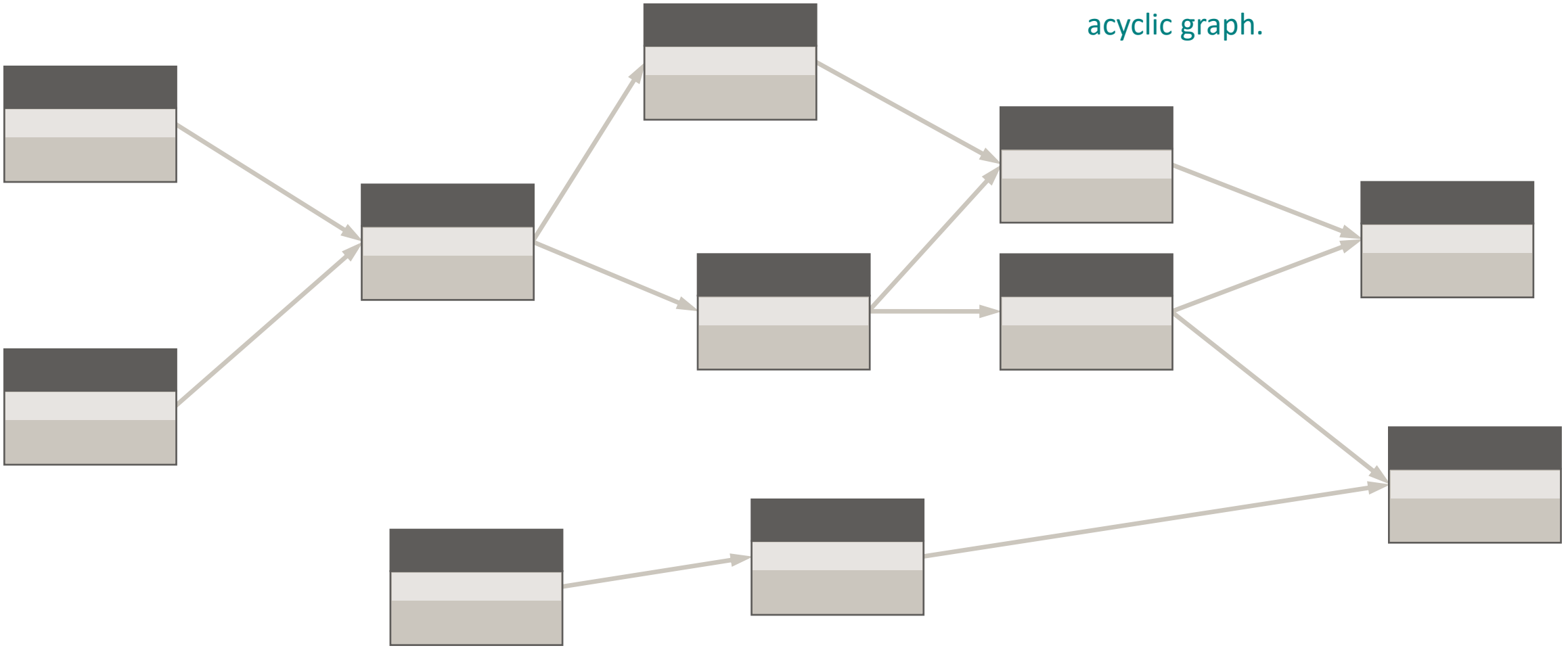
SyDEVS is an implementation of Symmetric DEVS, a paradigm combining discrete event simulation, dataflow programming, and agent-based modeling.



Paradigm

Dataflow

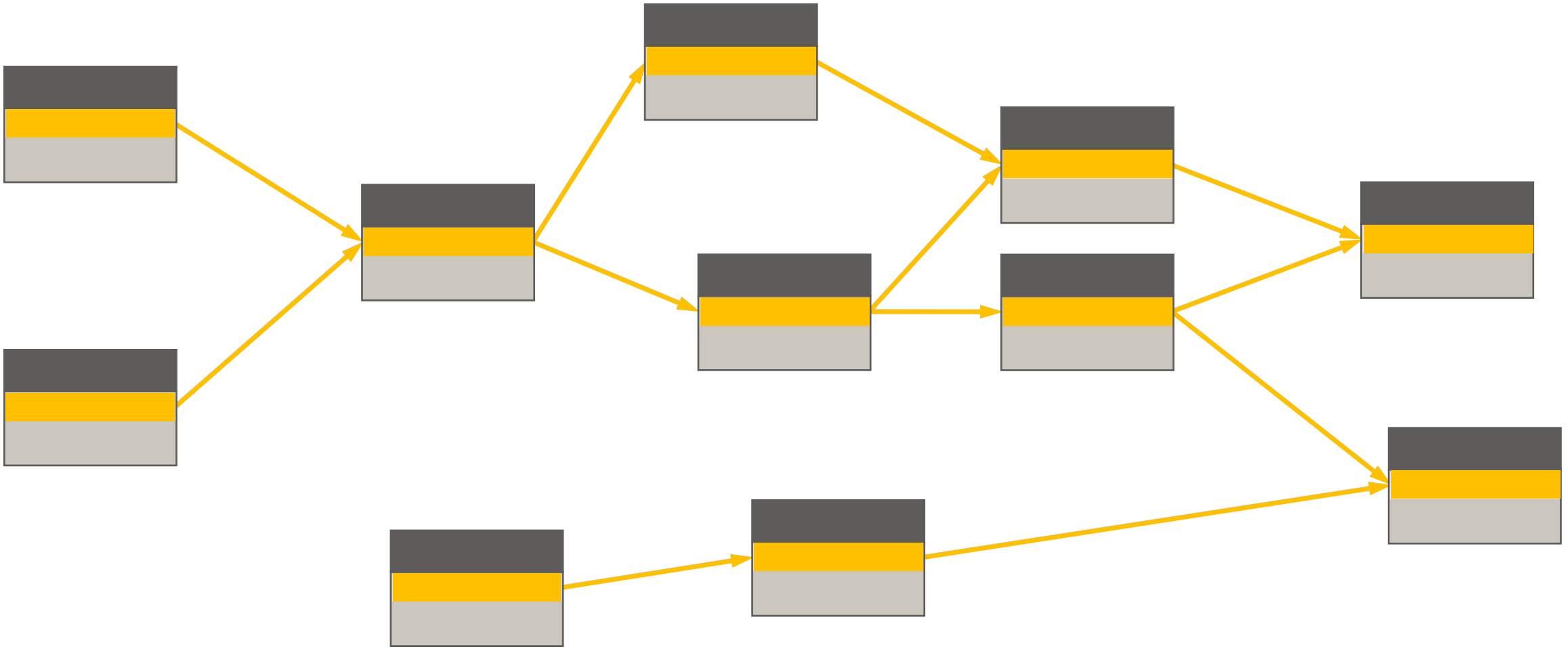
Dataflow programming is a widely used paradigm in which the links between nodes form a directed acyclic graph.



Paradigm

Dataflow

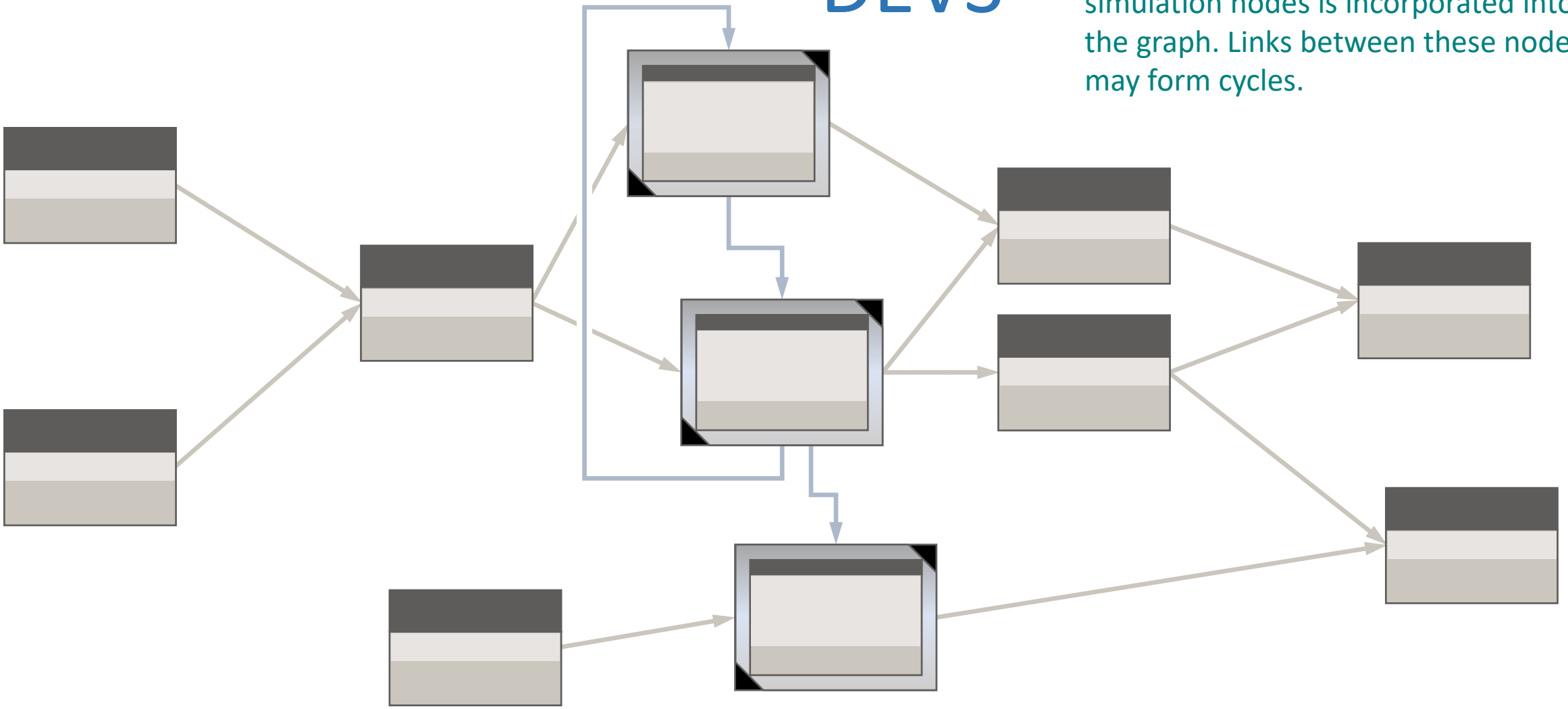
A node is executed once it receives data on all of its input ports.



Paradigm

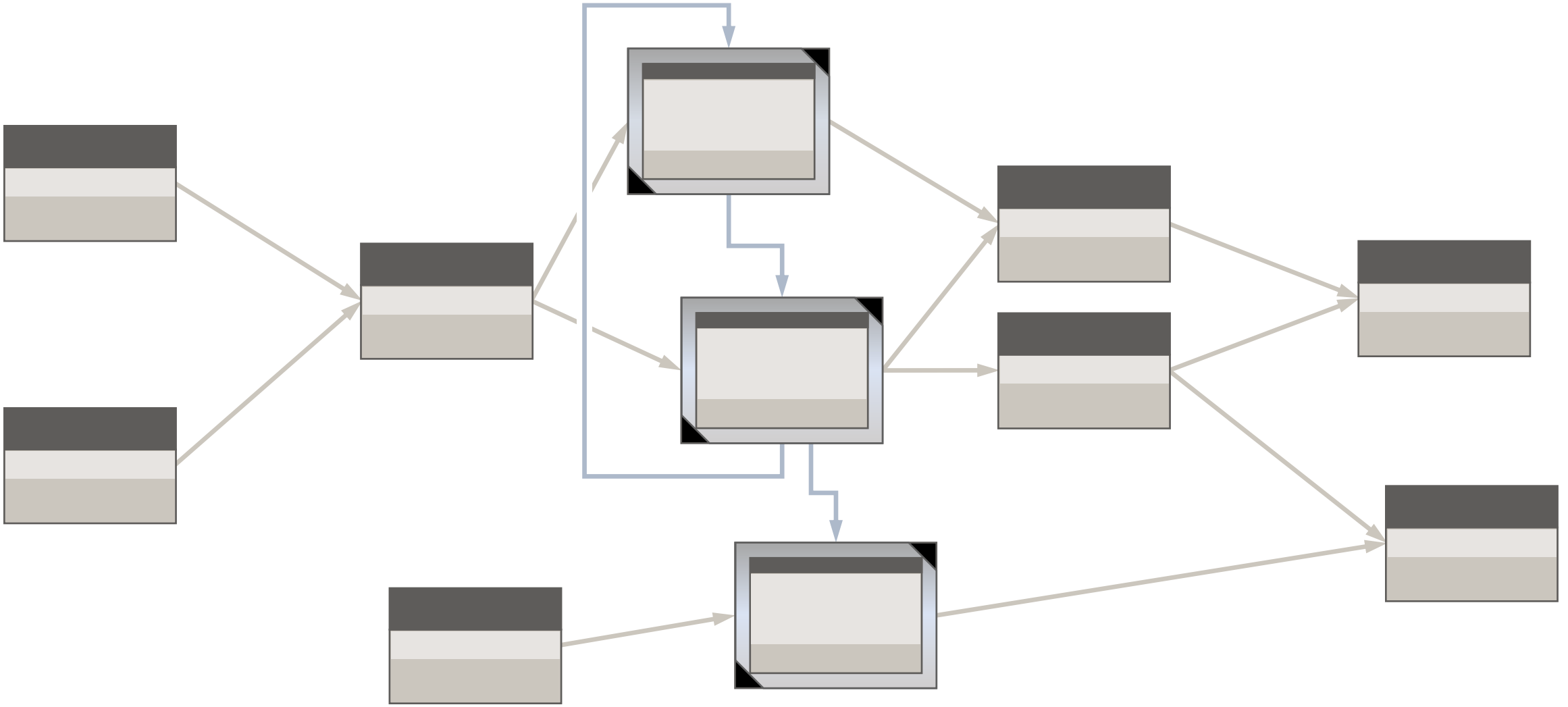
DEVS

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

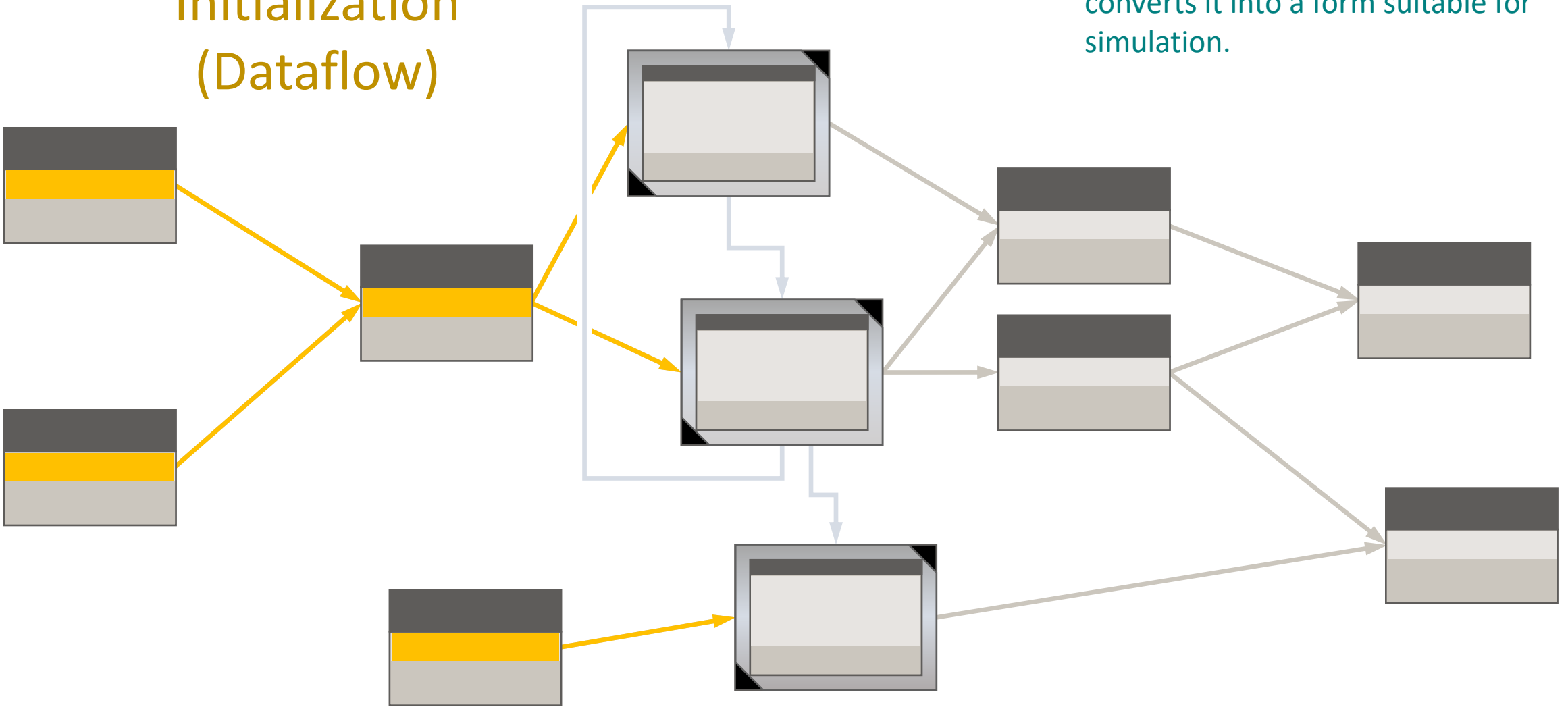
The execution of the entire graph is now partitioned into three phases.



Paradigm

Initialization (Dataflow)

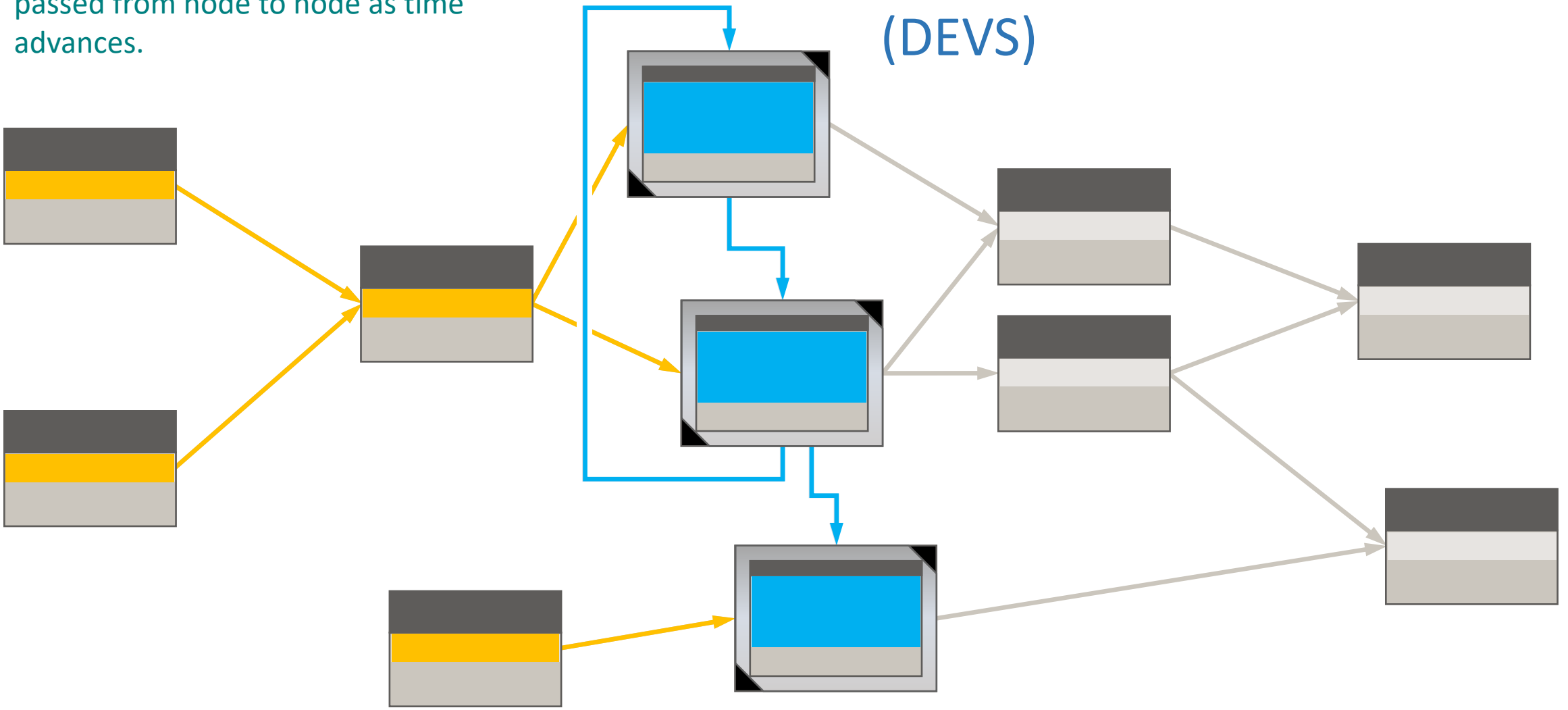
The first phase is called "Initialization". A dataflow network collects data and converts it into a form suitable for simulation.



Paradigm

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

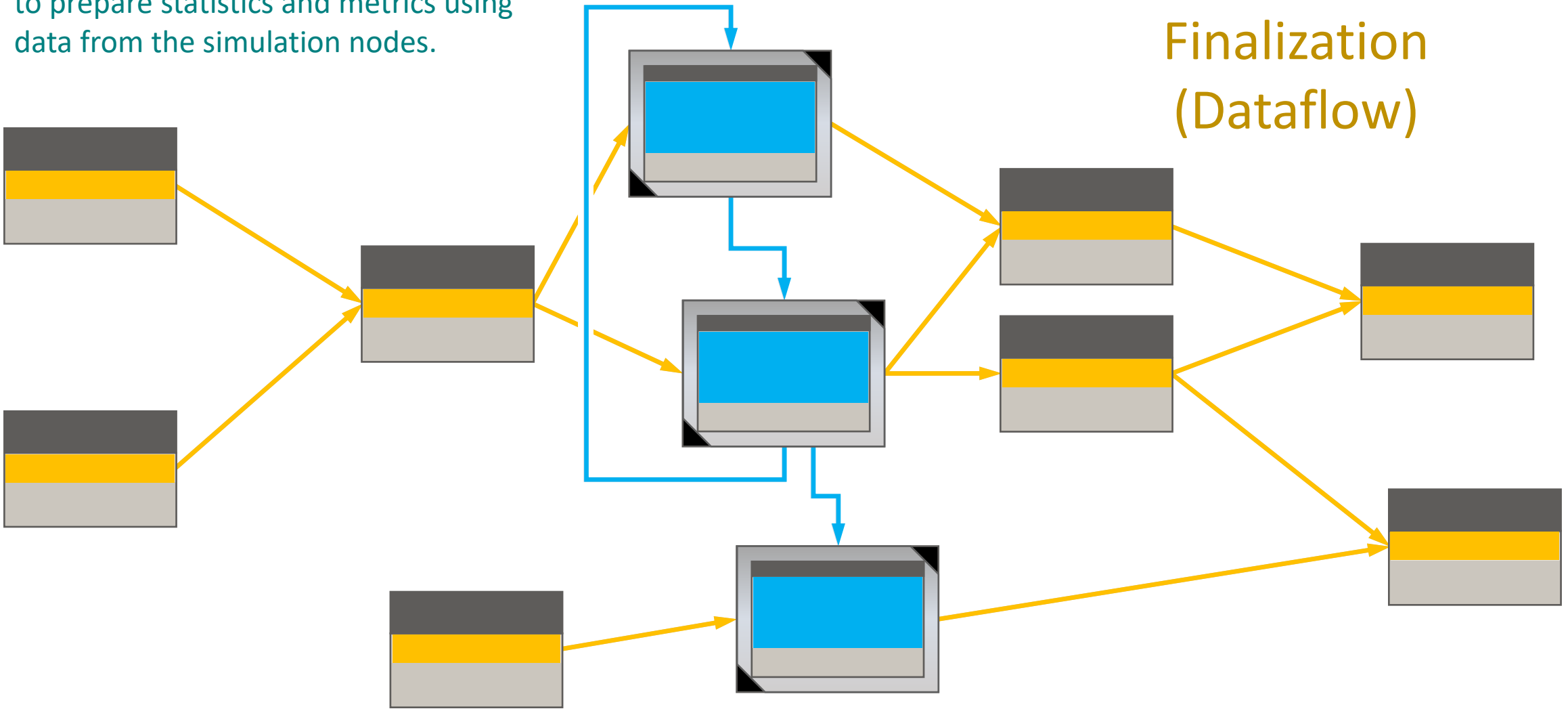
Simulation (DEVS)



Paradigm

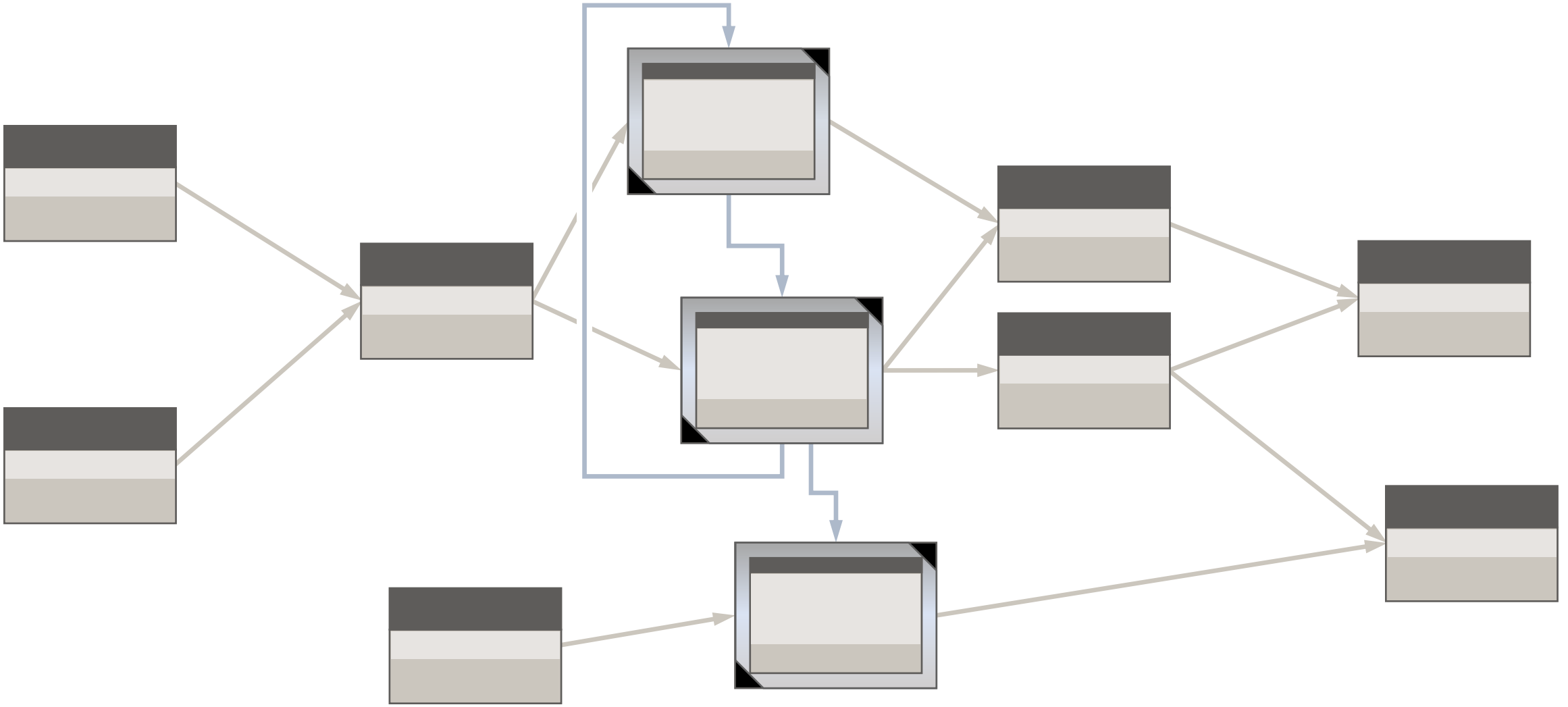
The third phase is called "Finalization". Another dataflow network is executed to prepare statistics and metrics using data from the simulation nodes.

Finalization (Dataflow)



Paradigm

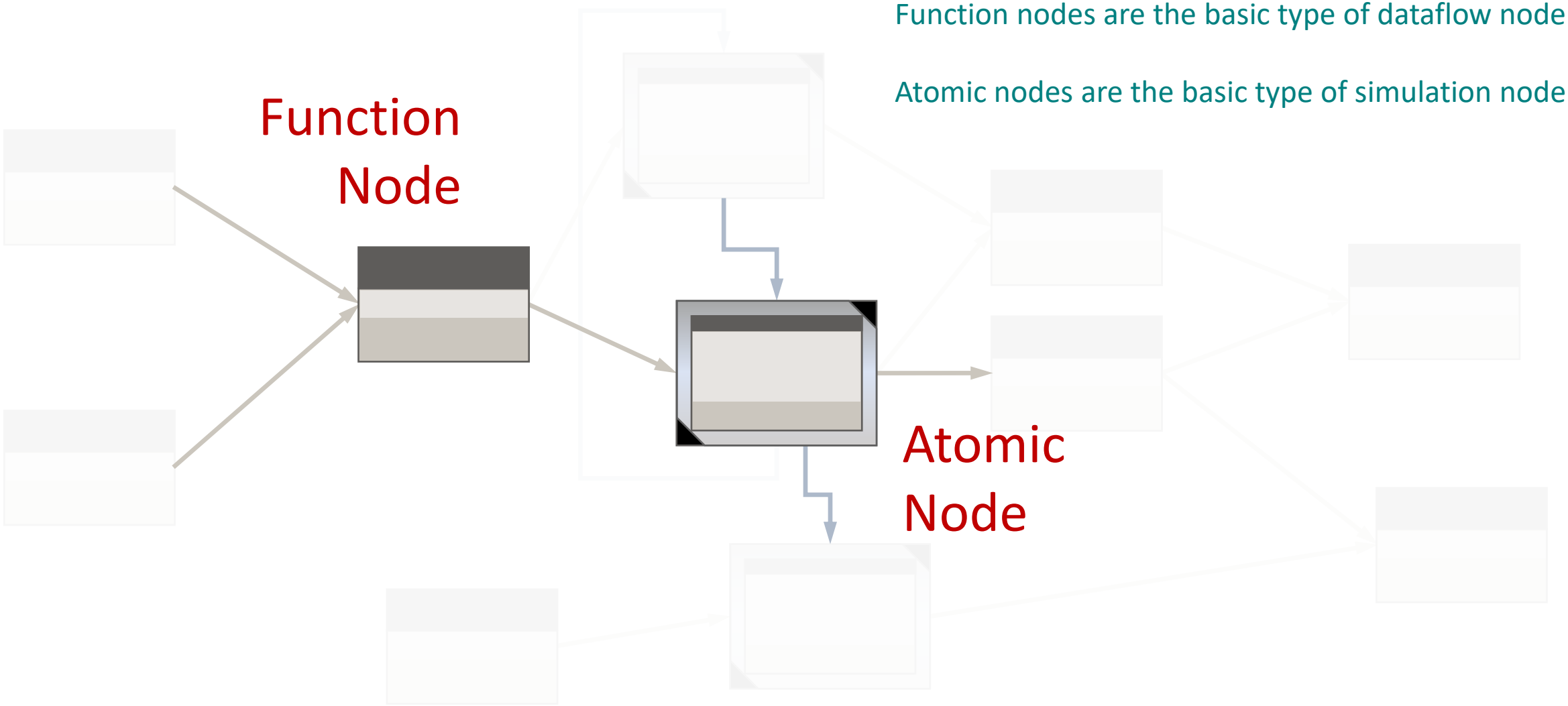
There are four types of nodes.



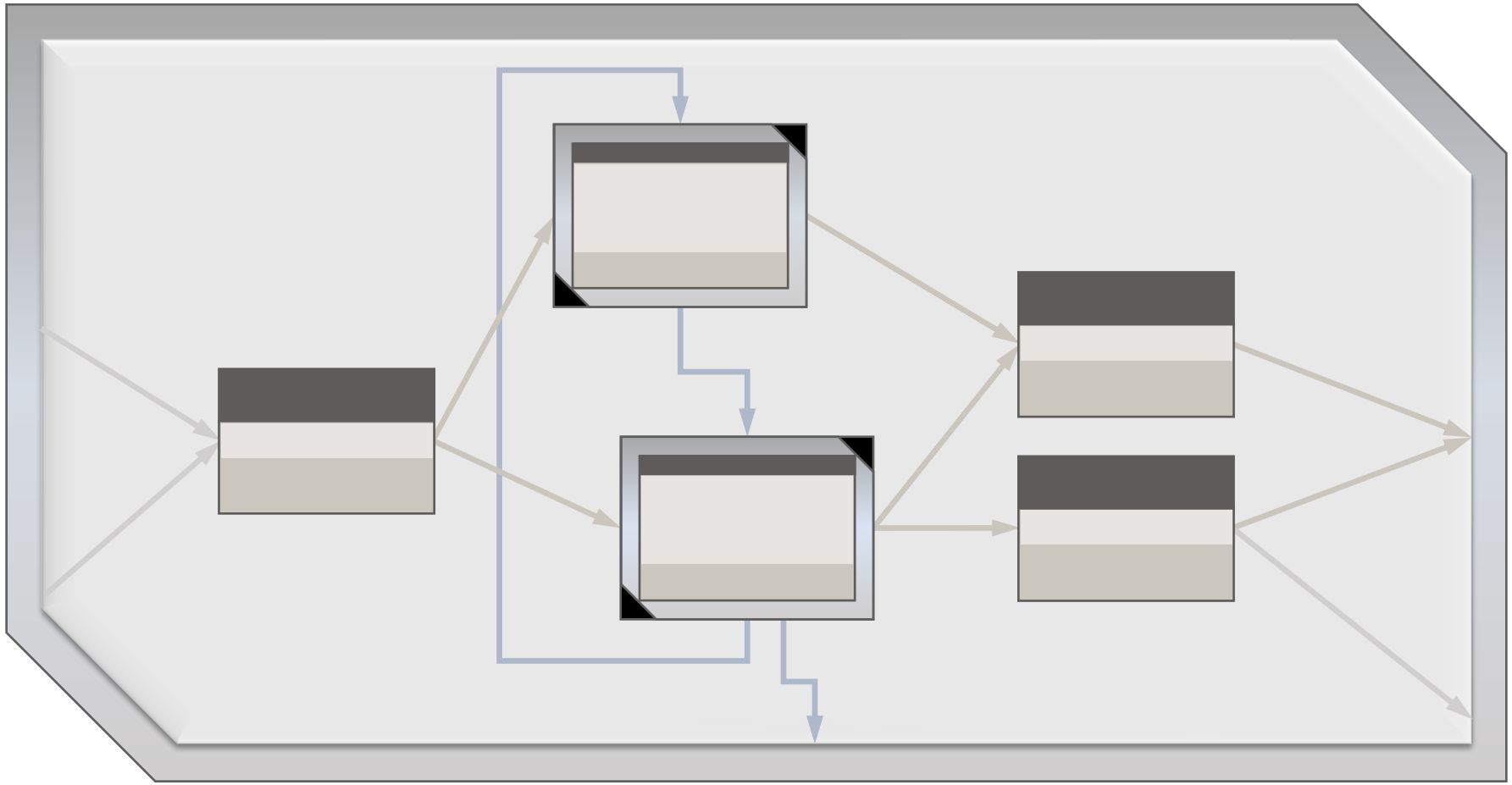
Paradigm

Function nodes are the basic type of dataflow node.

Atomic nodes are the basic type of simulation node.



Paradigm

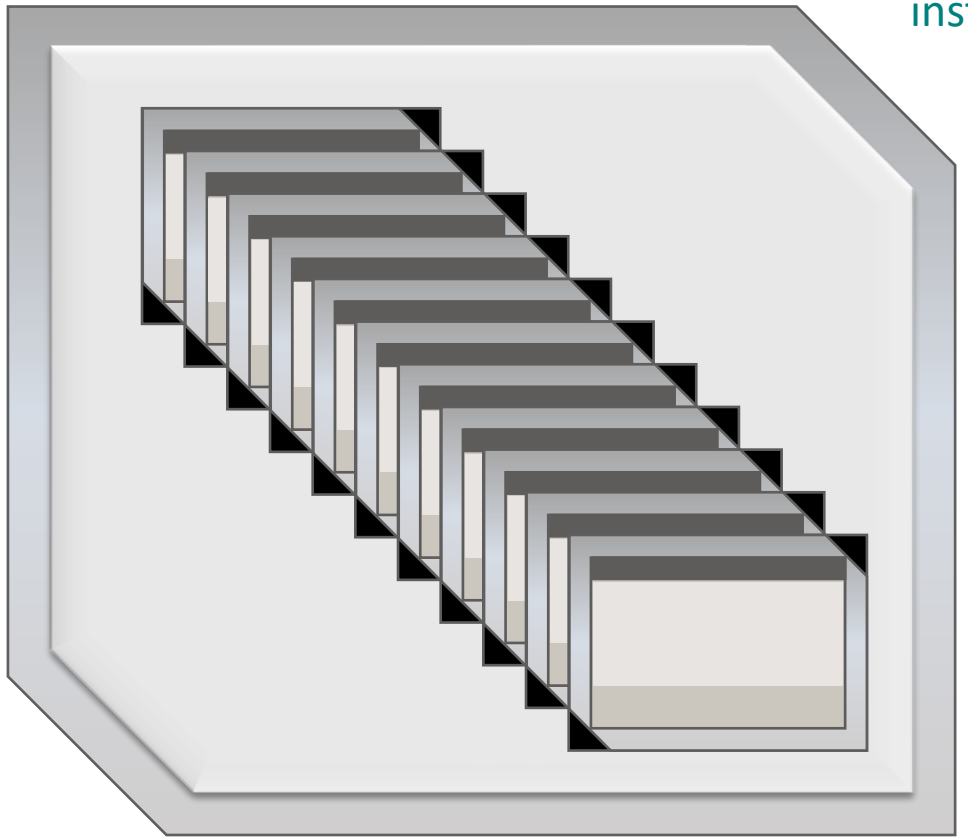


Composite Node

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

Paradigm

Collection Node



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.

Paradigm

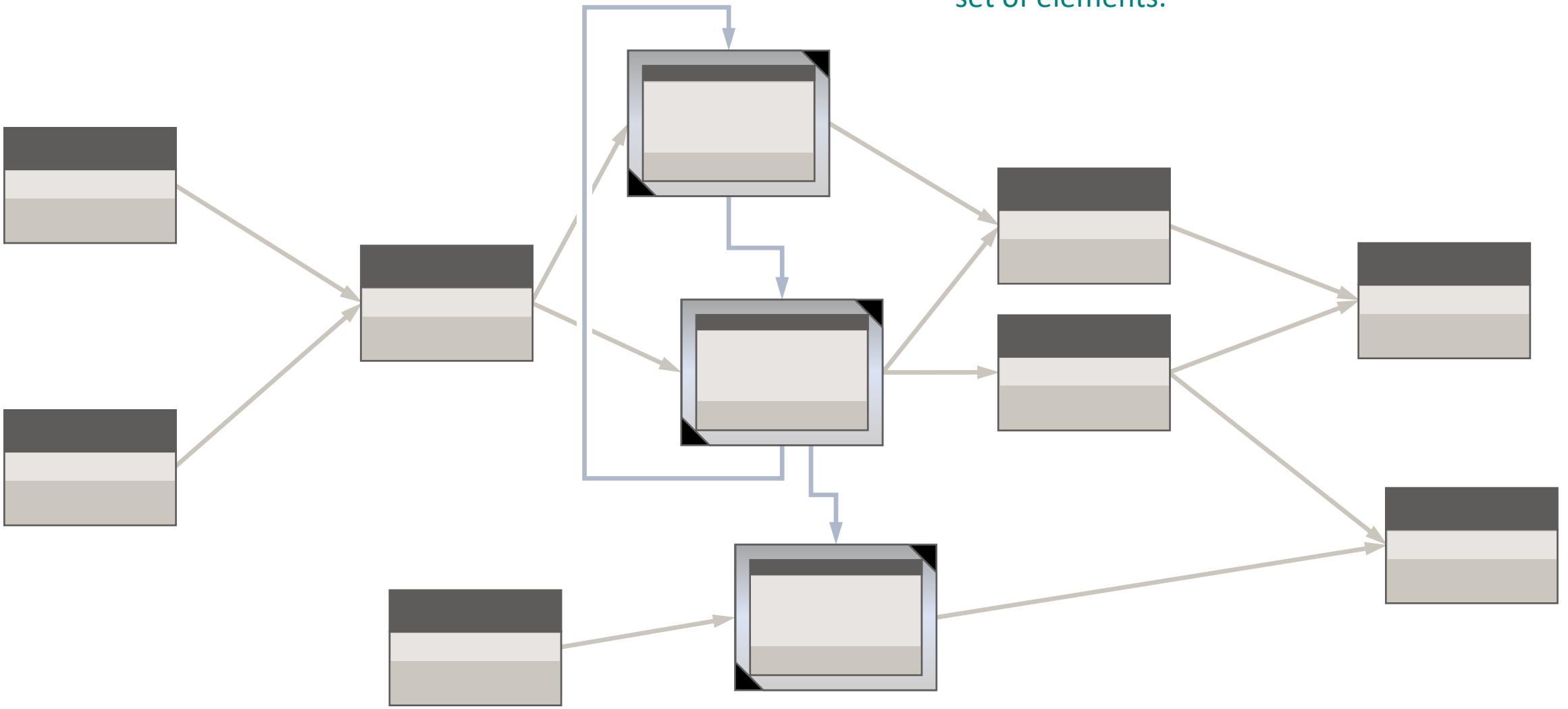
Nodes

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

Here is a list of the four types of nodes.

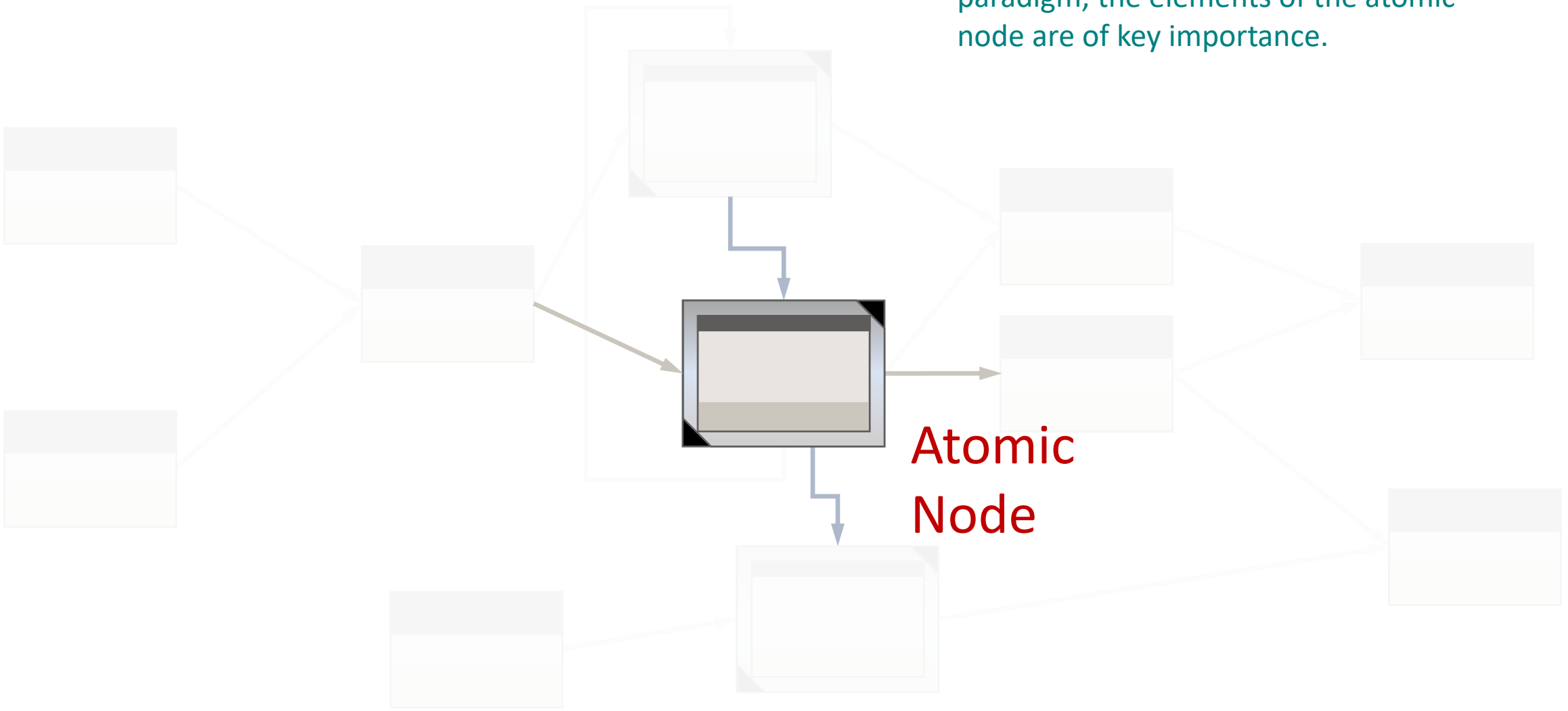
Paradigm

Each type of node contains a particular set of elements.



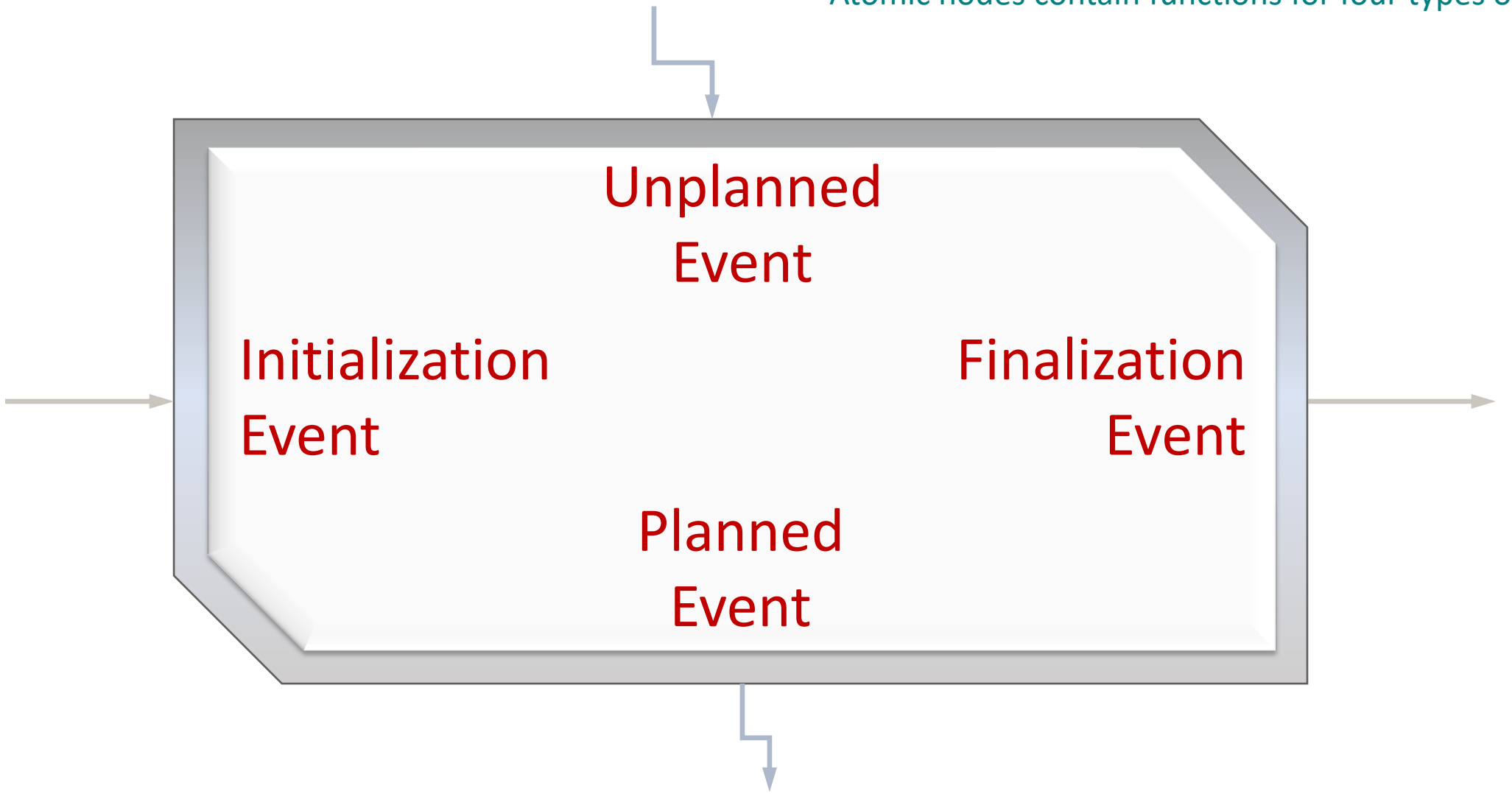
Paradigm

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



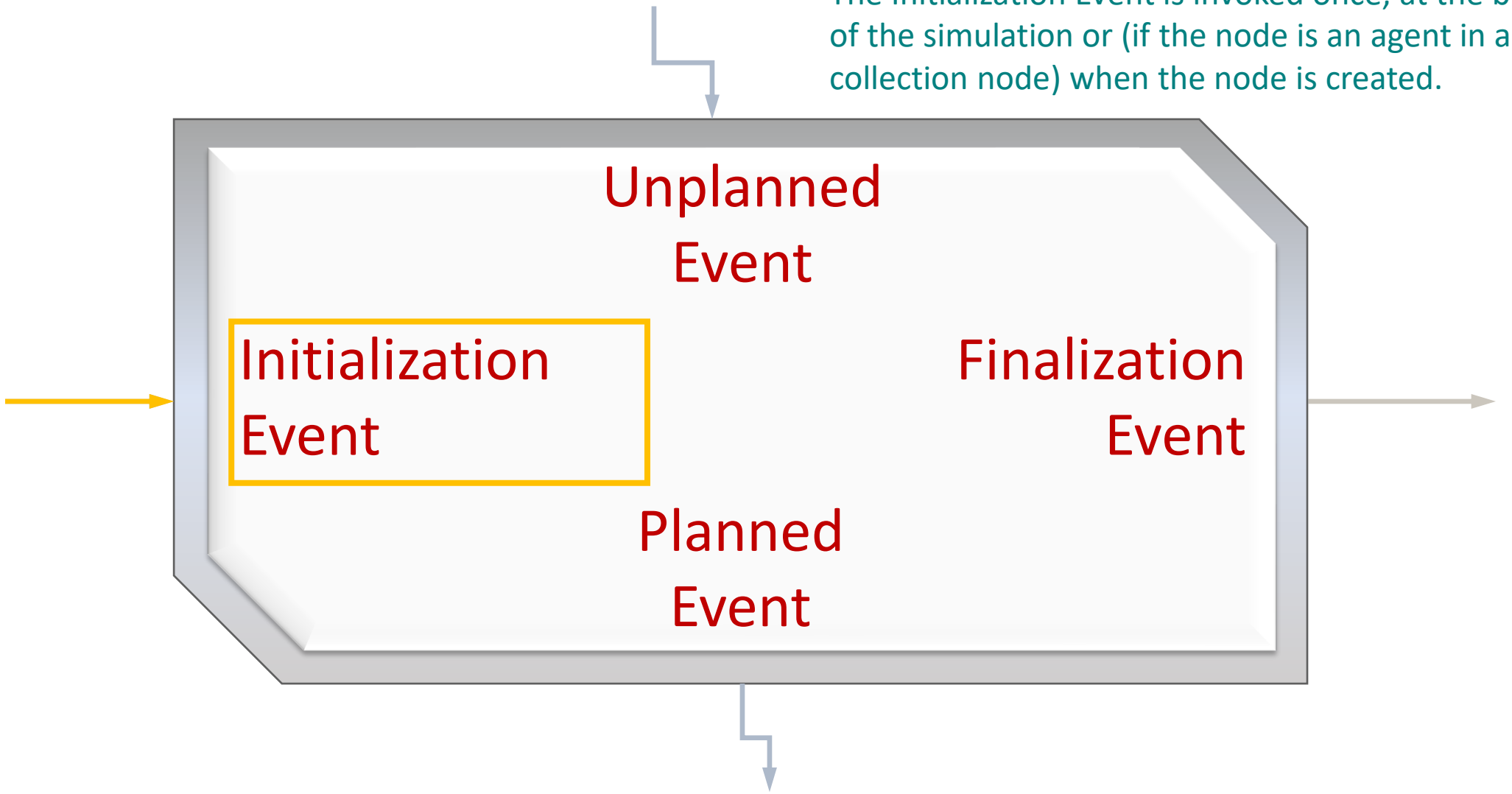
Paradigm

Atomic nodes contain functions for four types of events.



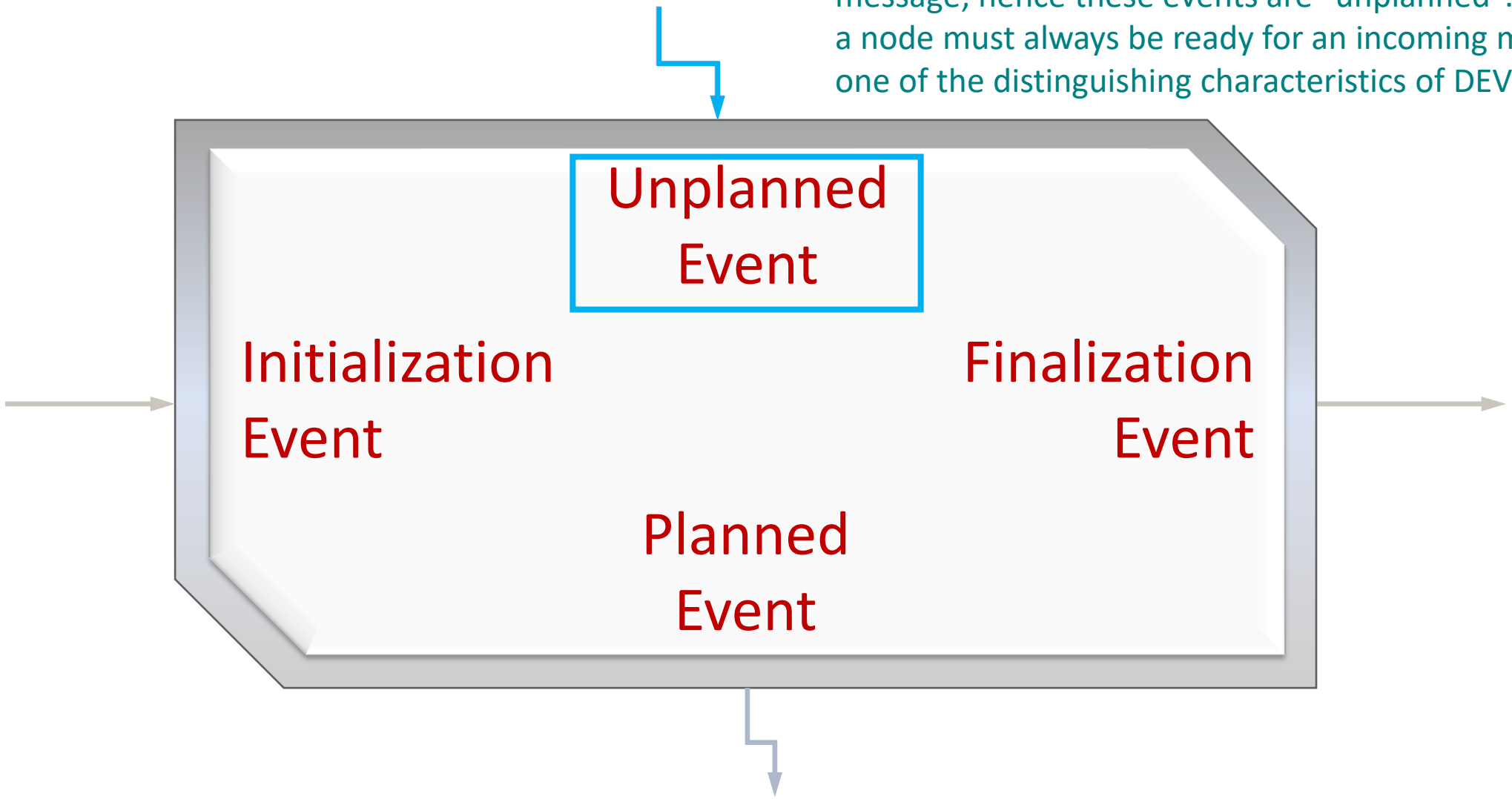
Paradigm

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.



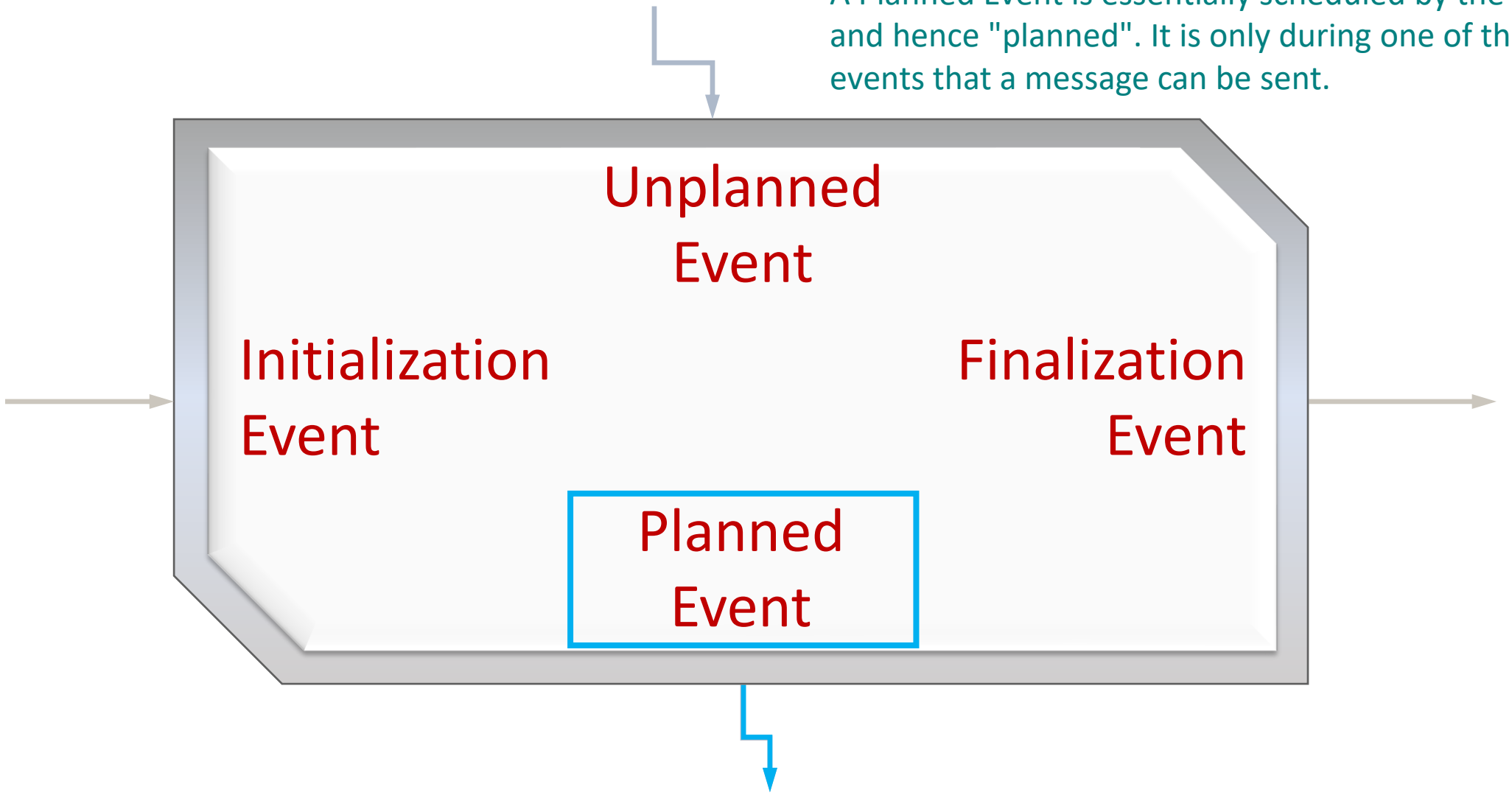
Paradigm

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.



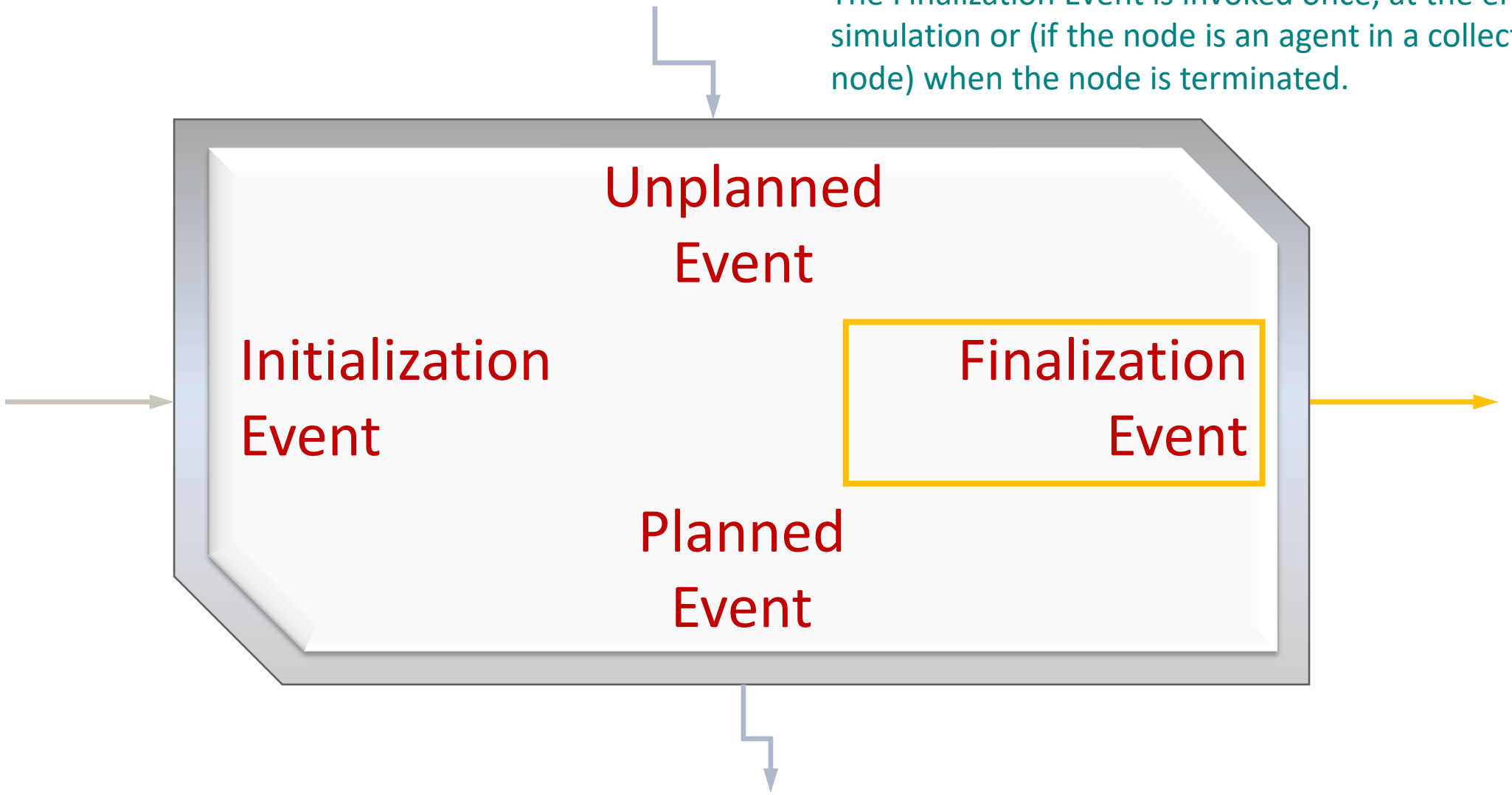
Paradigm

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.



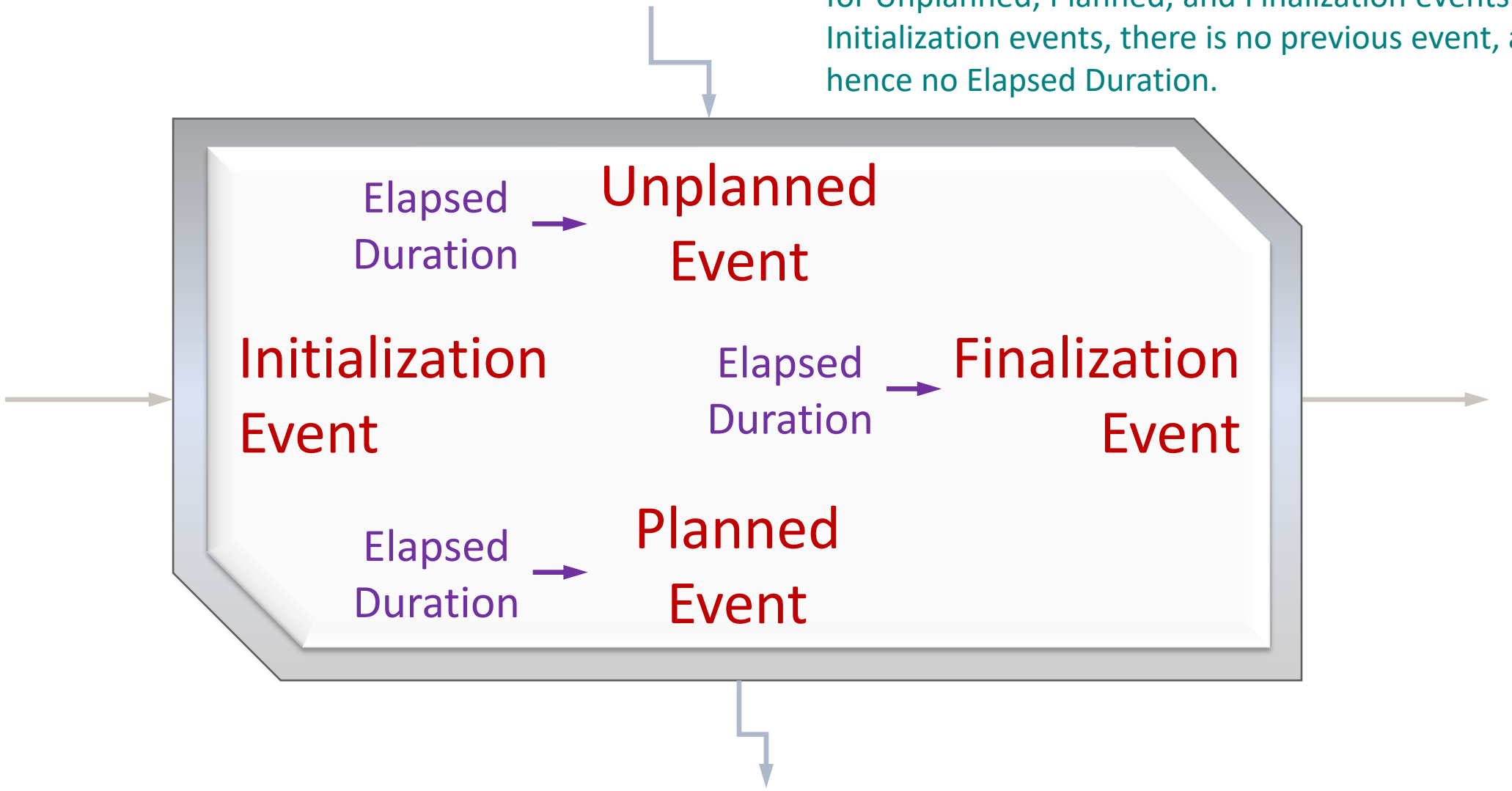
Paradigm

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.



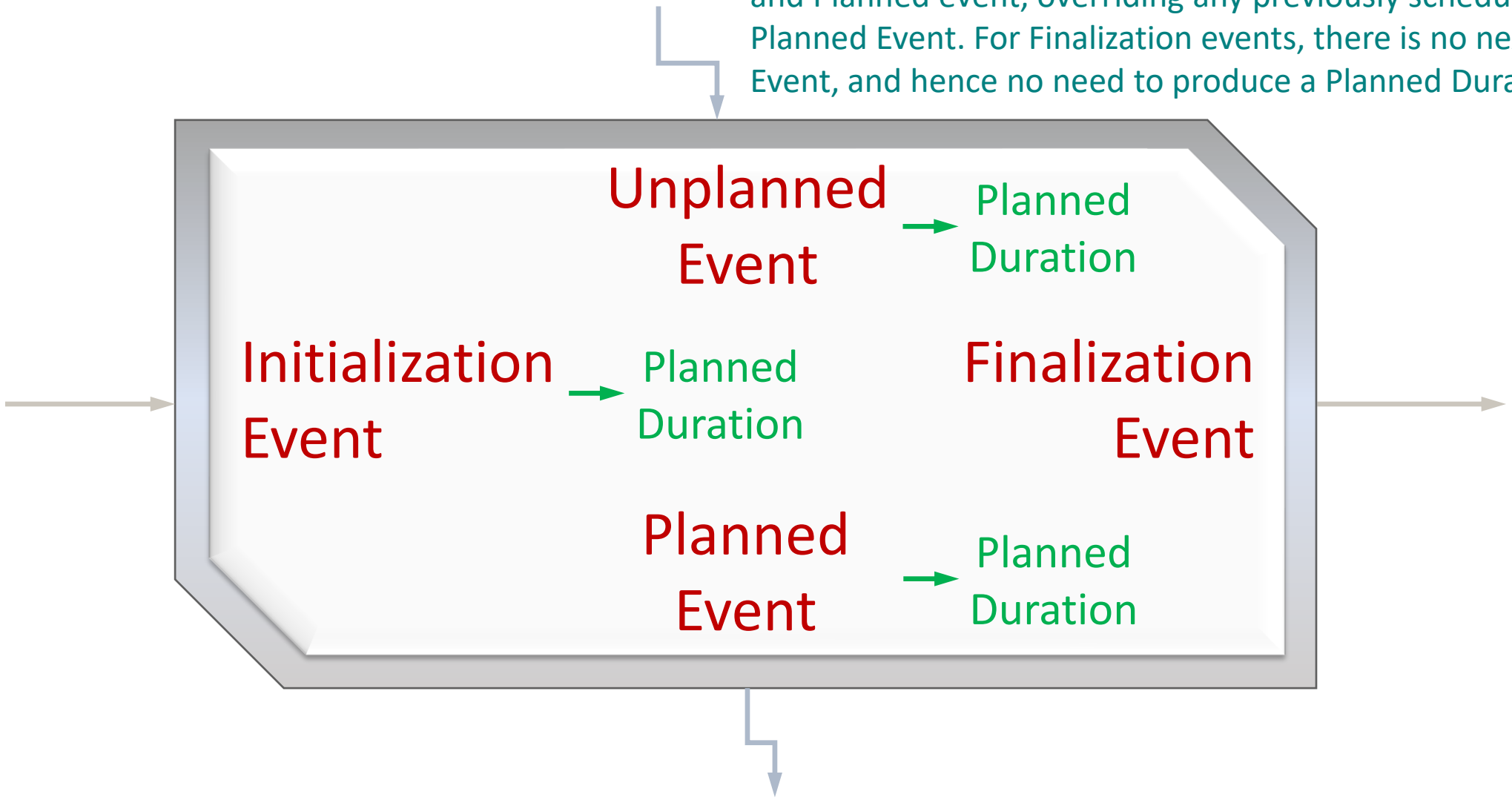
Paradigm

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

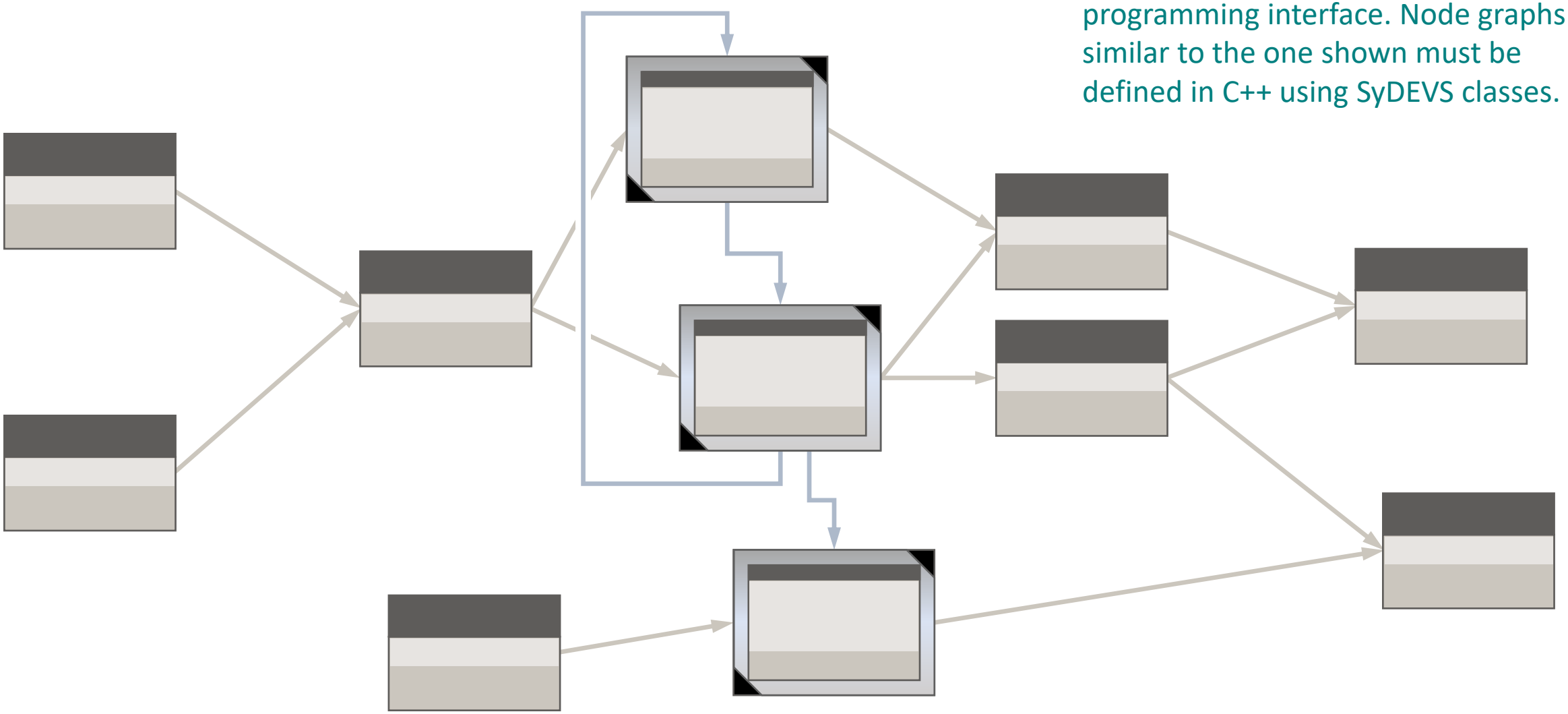
Nodes	Events
1. Function Node	1. Initialization Event
2. Atomic Node	2. Unplanned Event
3. Composite Node	3. Planned Event
4. Collection Node	4. Finalization Event

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

Implementation

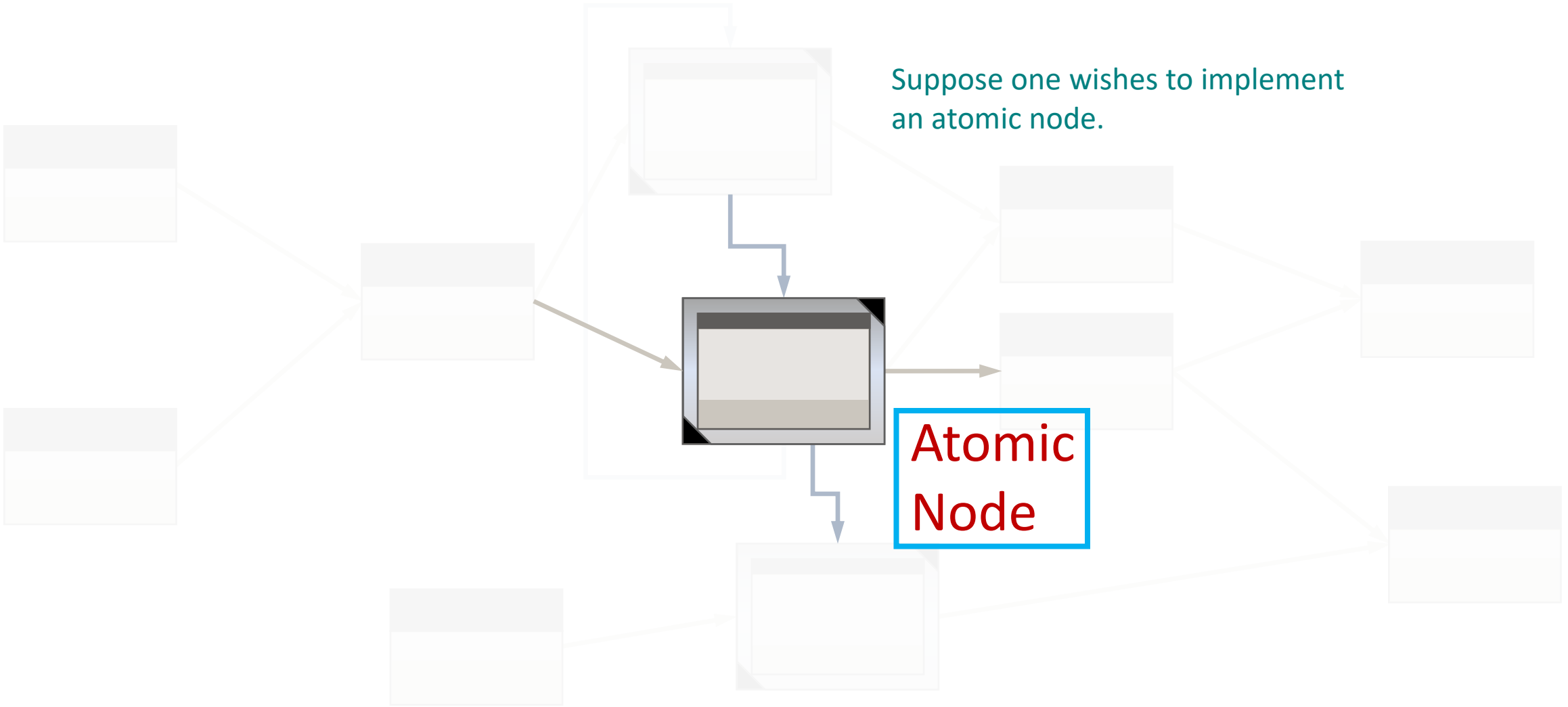
Implementation

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



Implementation

Suppose one wishes to implement an atomic node.



Implementation

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

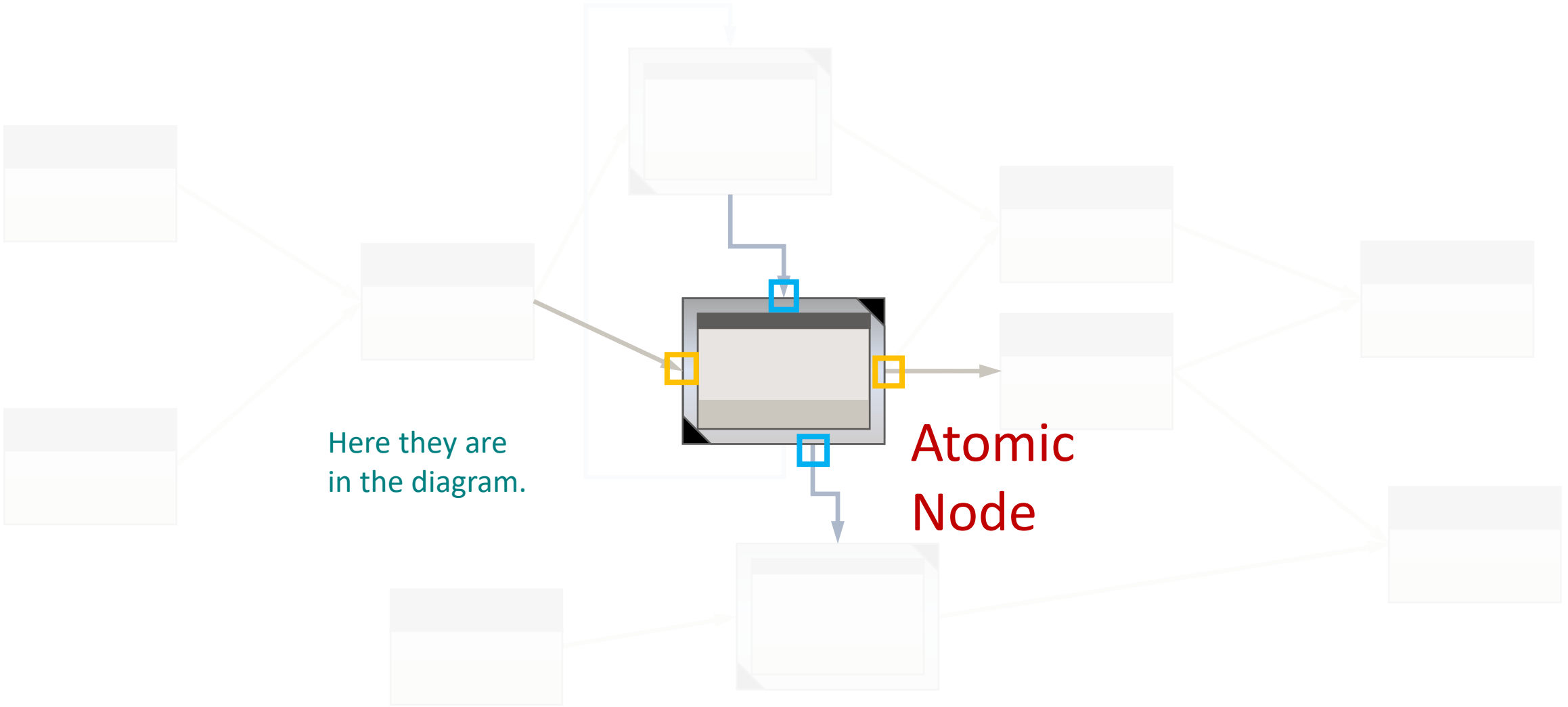
```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // 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

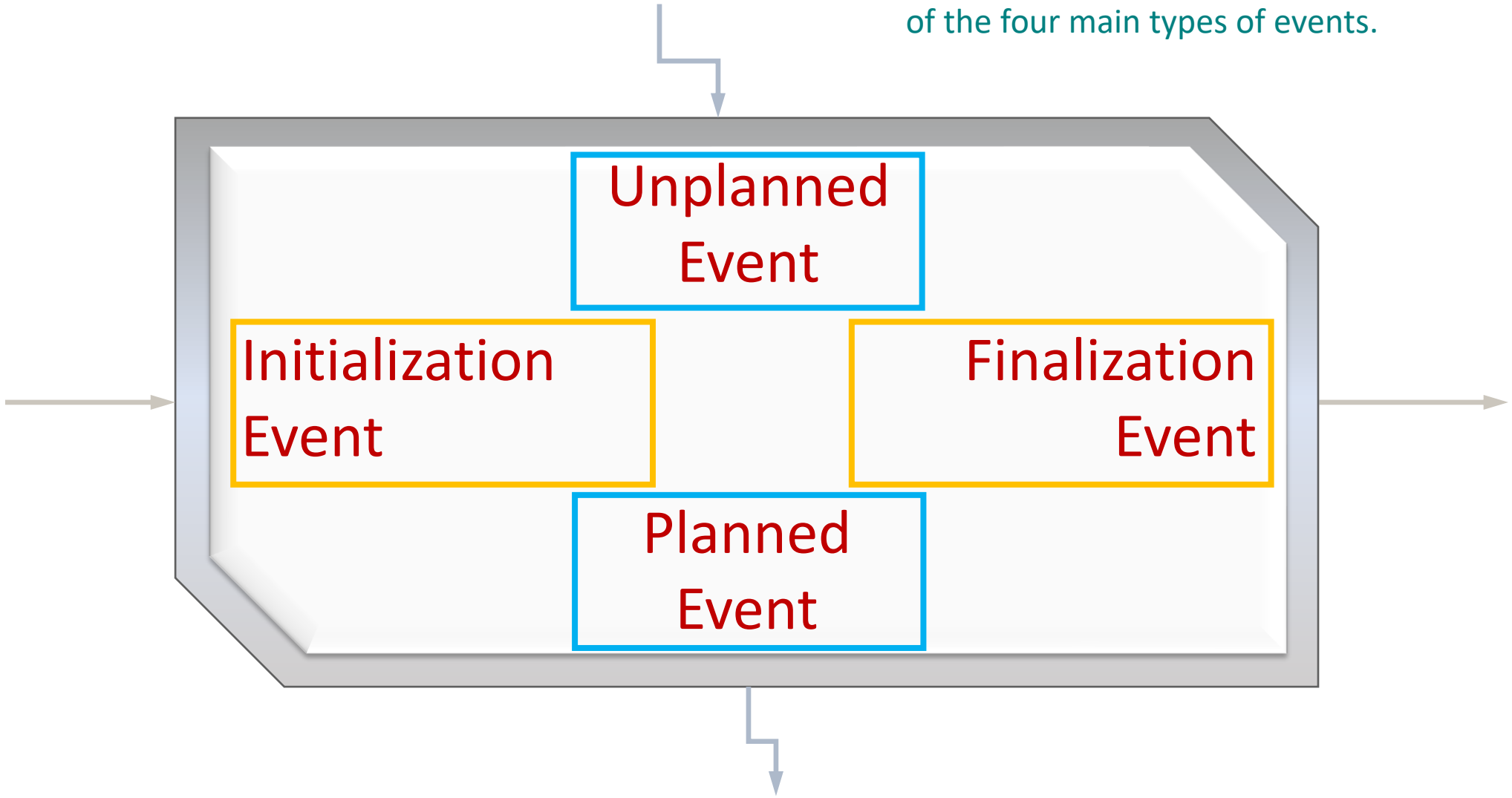


Here they are
in the diagram.

Atomic
Node

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);
};
```

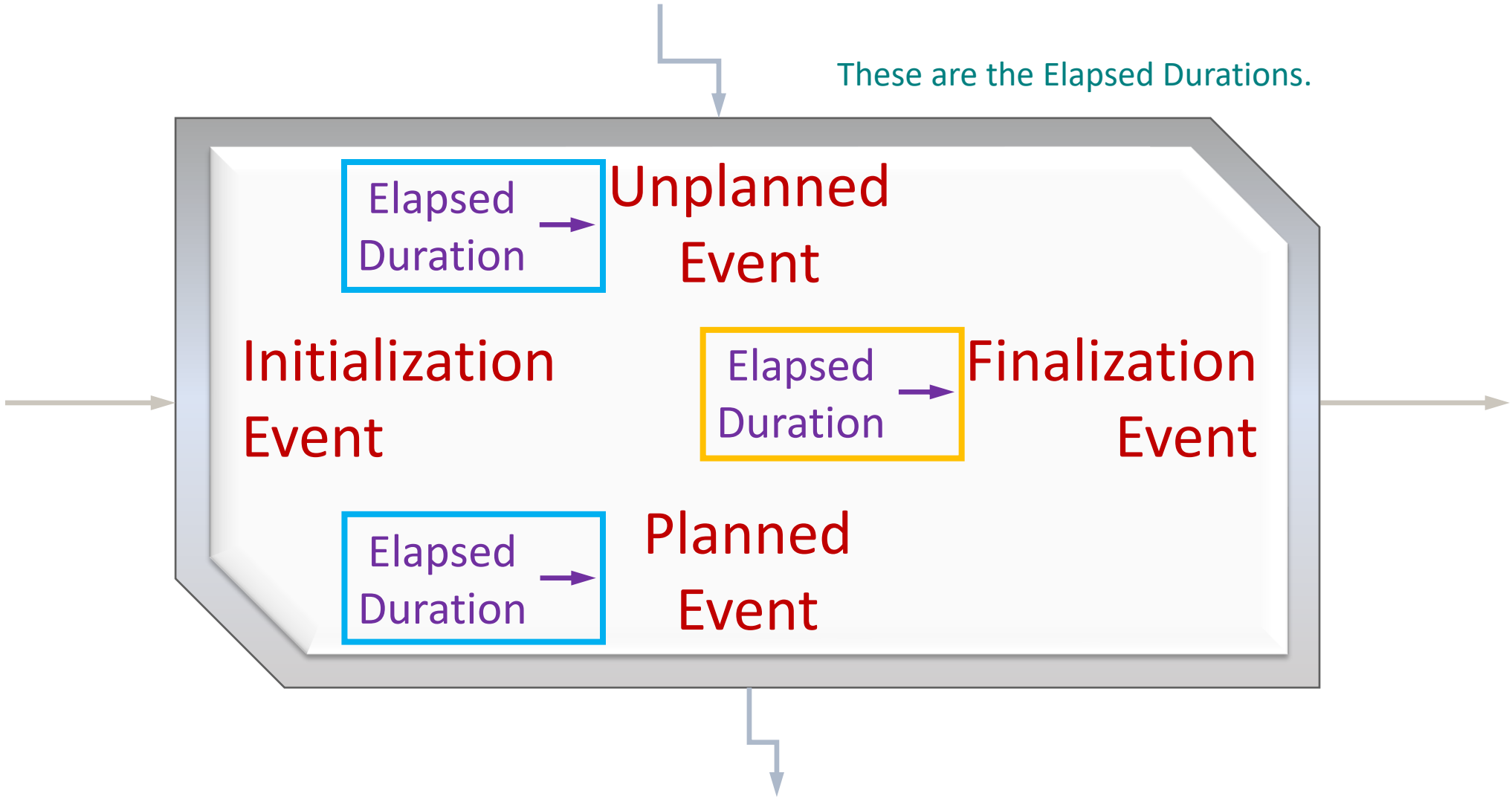
The code to be executed for each type of event is placed in these four member functions.

Implementation

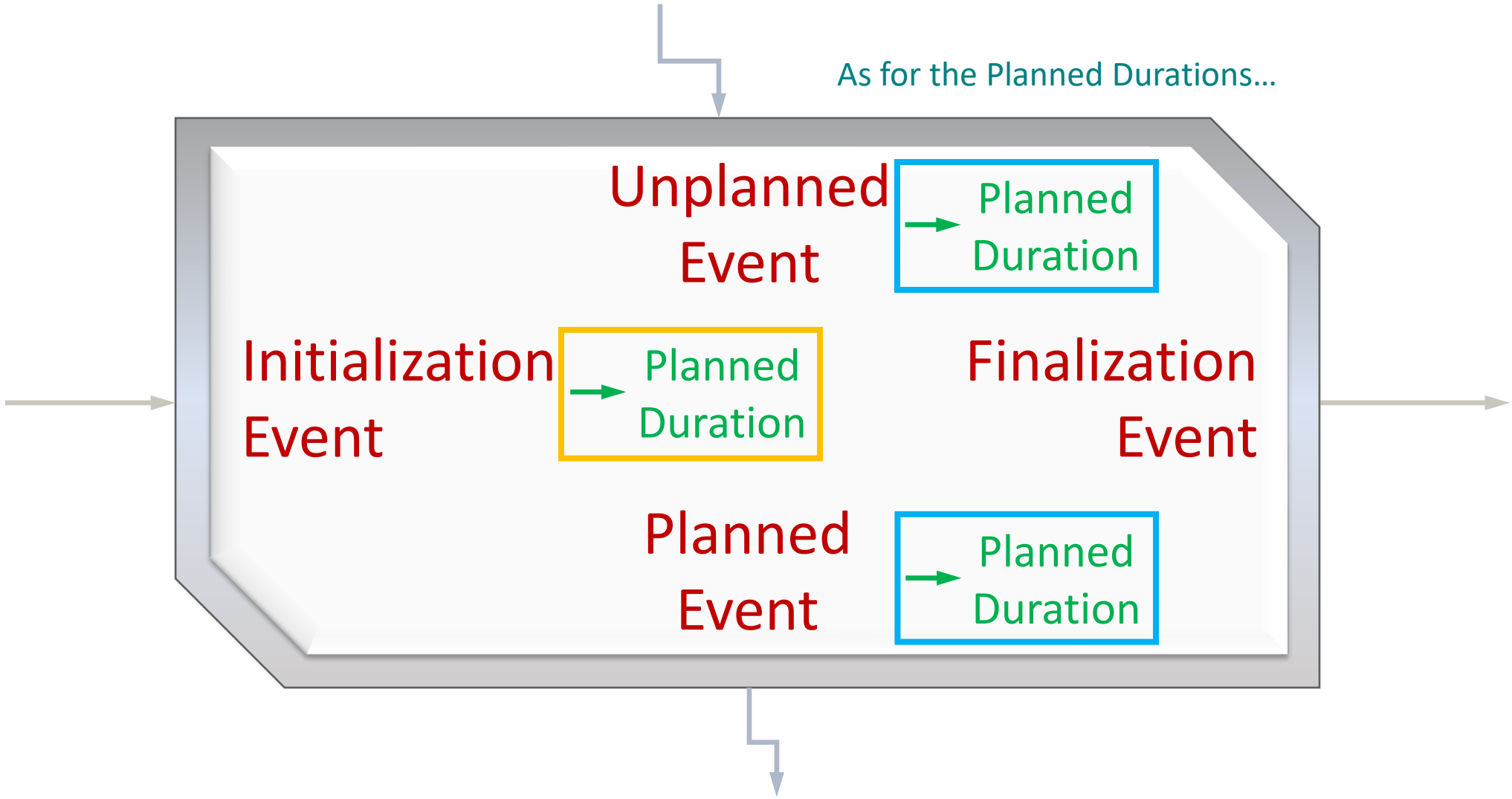
```
class queueing_node : public atomic_node
{
public:
    // Constructor/Destructor:
    queueing_node(const std::string& node_name, const node_context& external_context);
    virtual ~queueing_node() = default;

    // 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