

SyDEVs Library

Framework Overview

Autodesk Research

April 2018

Classic Theory (1970s)



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

[Interaction](#)

[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

[Tools](#)

[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Not logged in [Talk](#) [Contributions](#) [Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

DEVS

From Wikipedia, the free encyclopedia

DEVS abbreviating **Discrete Event System Specification** is a modular and hierarchical formalism for modeling and analyzing general systems

...

Atomic DEVS [\[edit \]](#)

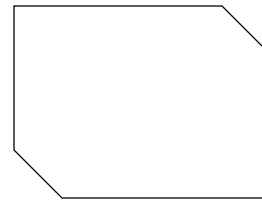
An atomic DEVS model is defined as a 7-tuple

$$M = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

where

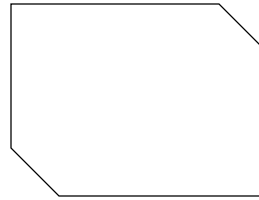
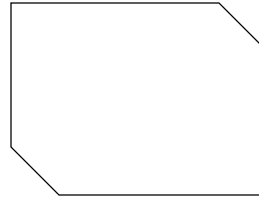
- X is the set of input events;
- Y is the set of output events;
- S is the set of sequential states (or also called the set of partial states);
- $s_0 \in S$ is the initial state;
- $ta : S \rightarrow \mathbb{T}^\infty$ is the time advance function which is used to determine the lifespan of a state;
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function which defines how an input event changes a state of the system, where $Q = \{(s, t_e) | s \in S, t_e \in (\mathbb{T} \cap [0, ta(s)])\}$ is the set of total states, and t_e is the elapsed time since the last event;
- $\delta_{int} : S \rightarrow S$ is the internal transition function which defines how a state of the system changes internally (when the elapsed time reaches to the lifetime of the state);
- $\lambda : S \rightarrow Y^\phi$ is the output function where $Y^\phi = Y \cup \{\phi\}$ and $\phi \notin Y$ is a silent event or an unobserved event. This function defines how a state of the system generates an output event (when the elapsed time reaches to the lifetime of the state);

SyDEVS Approach

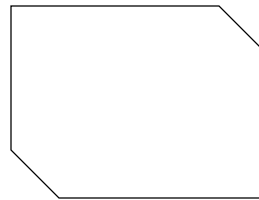
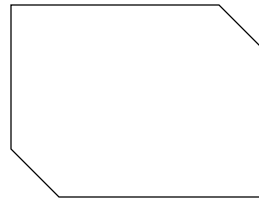


Simulation Node

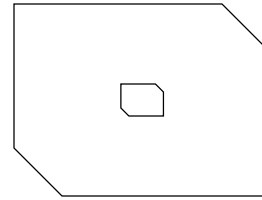
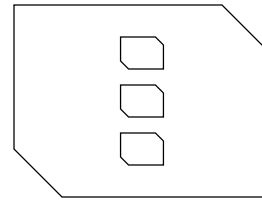
SyDEVS Approach



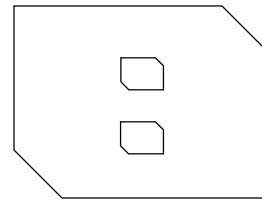
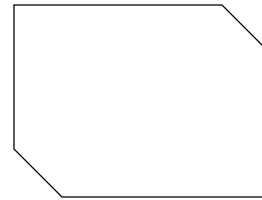
Simulation Node



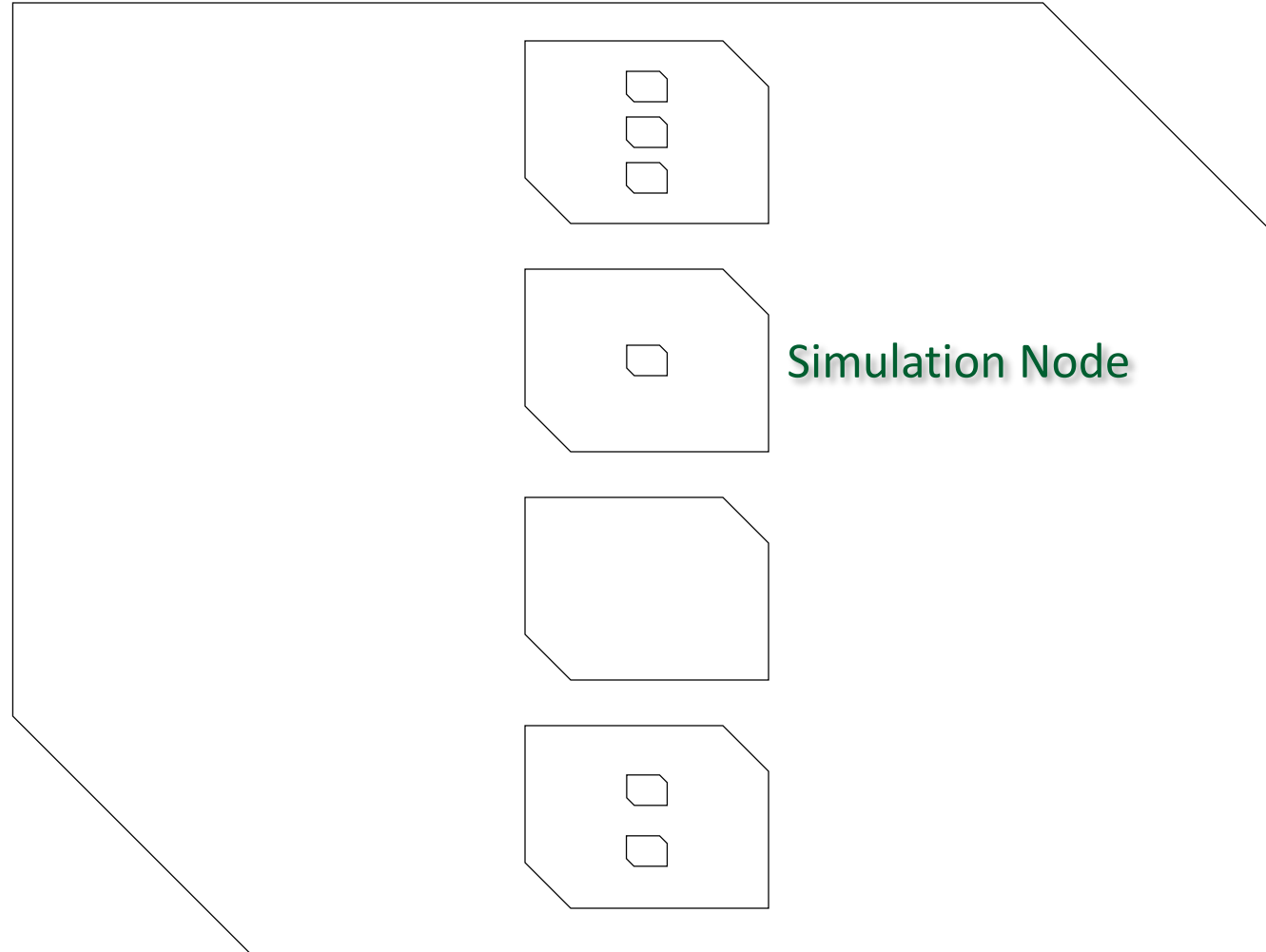
SyDEVS Approach



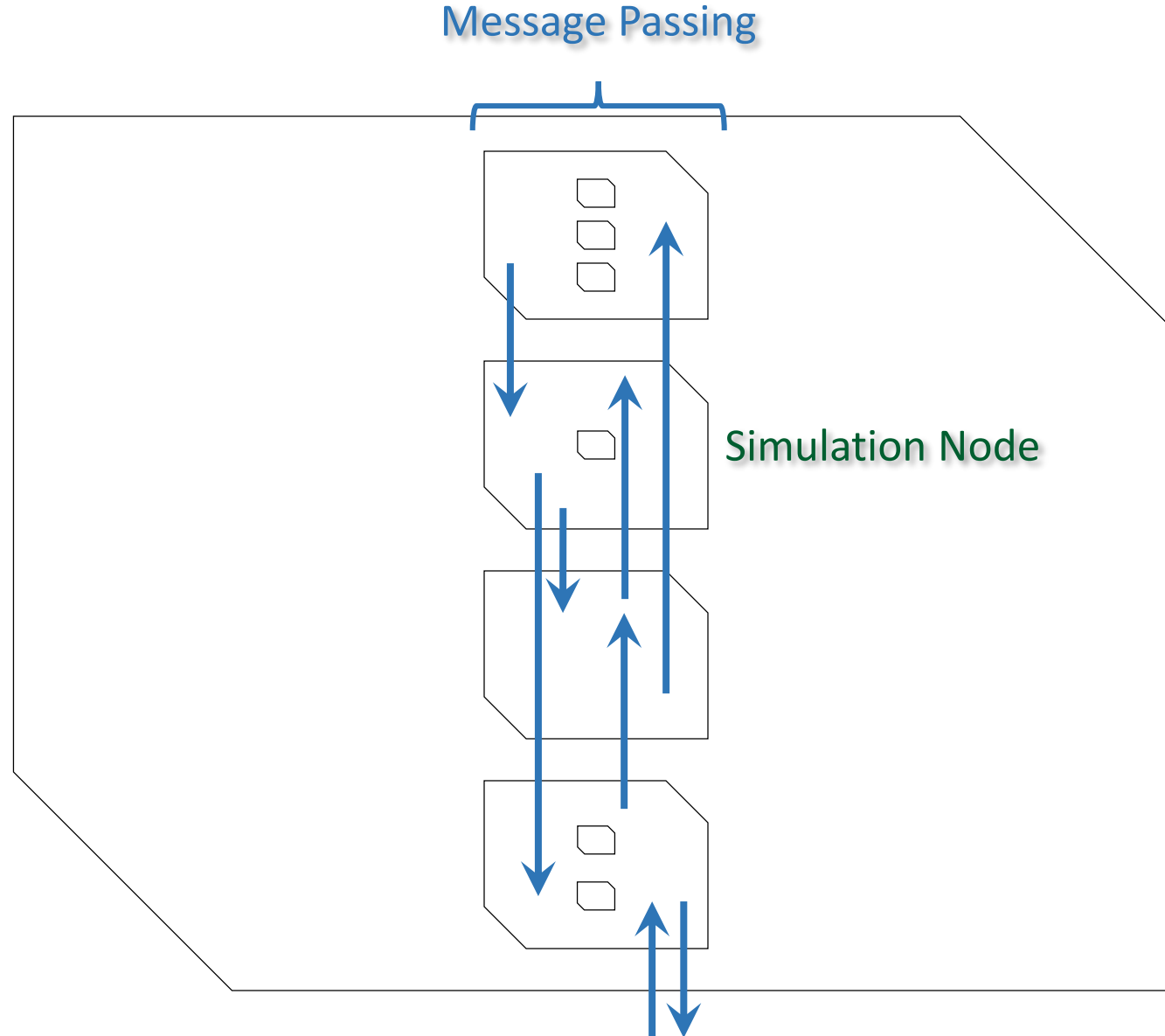
Simulation Node



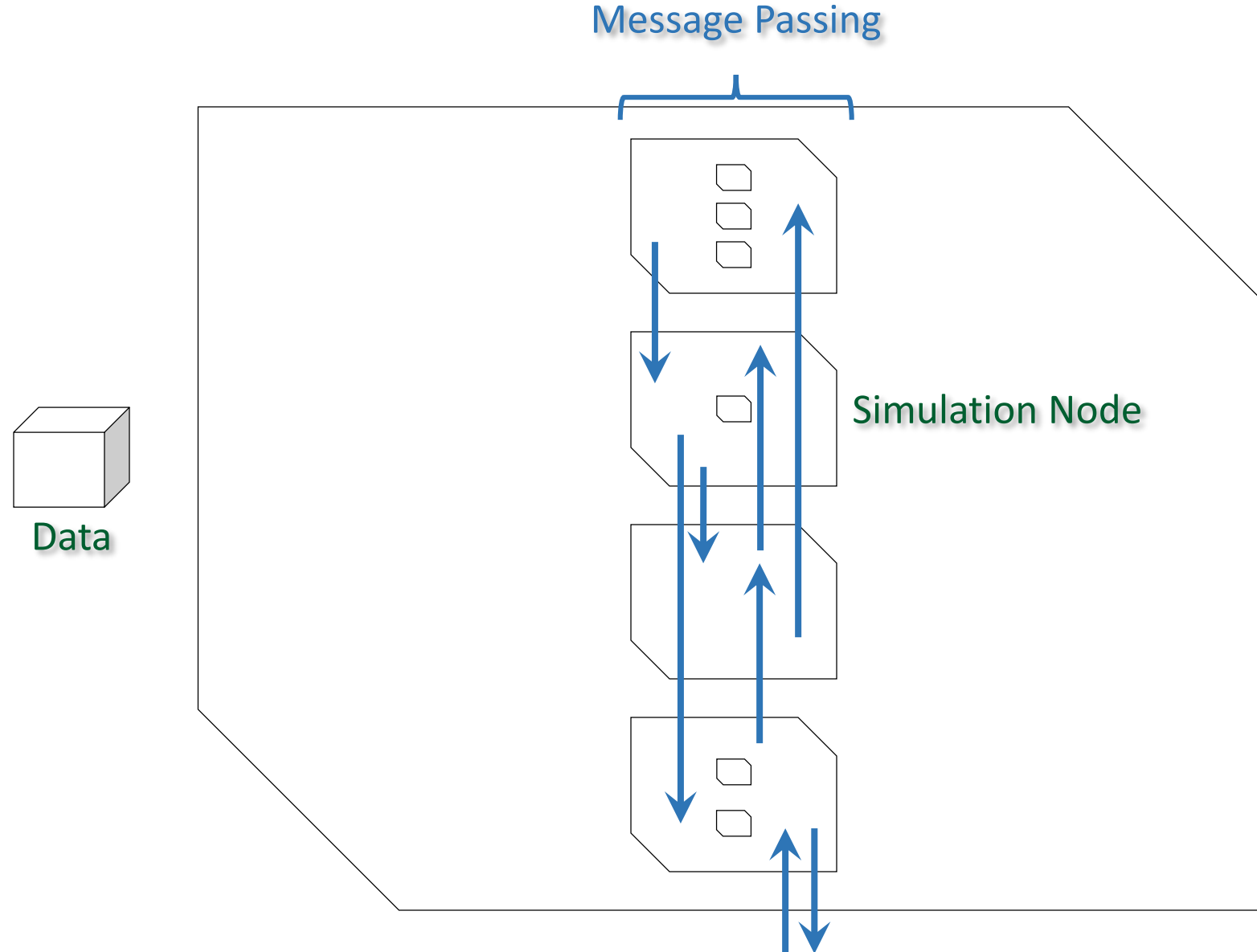
SyDEVS Approach



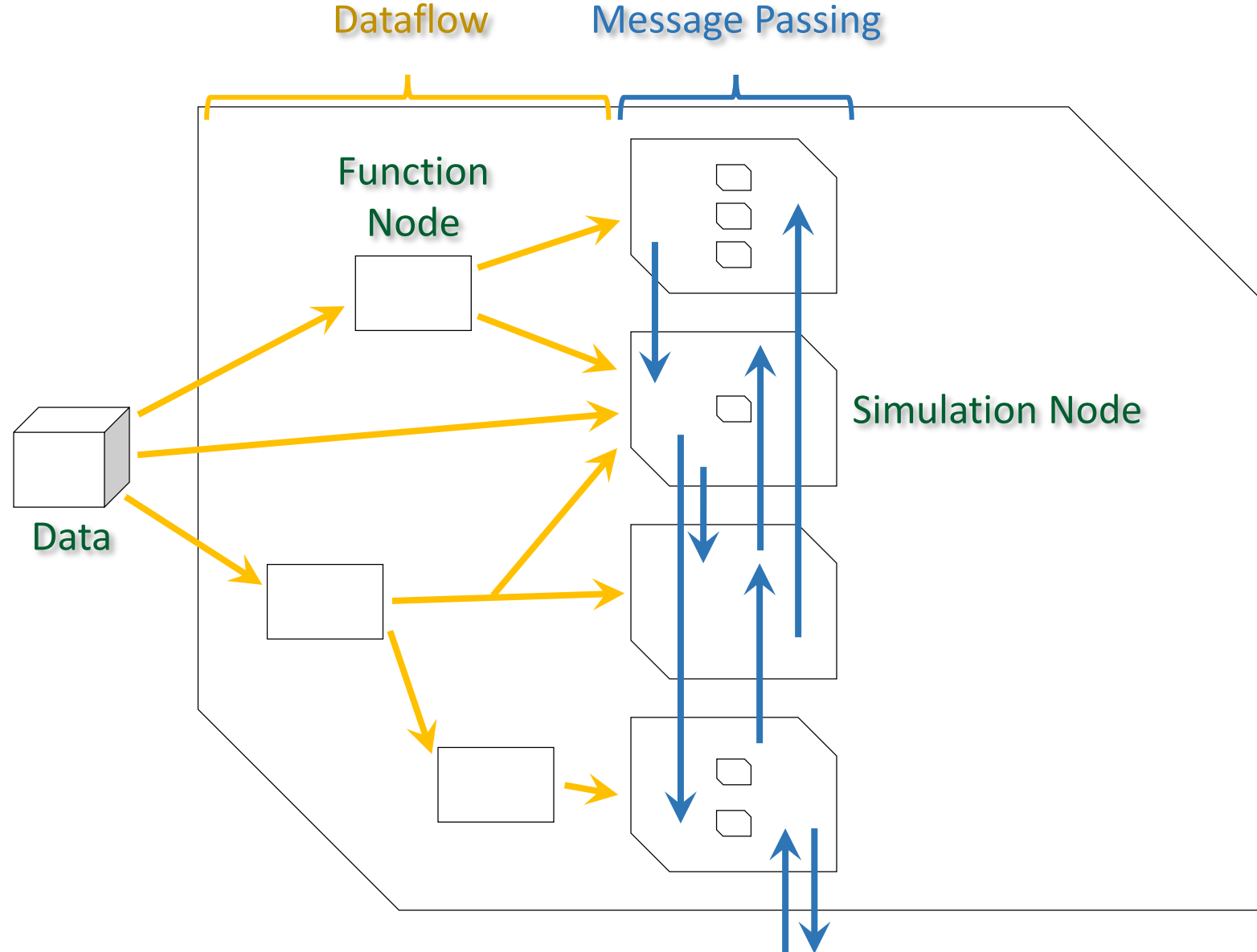
SyDEVS Approach



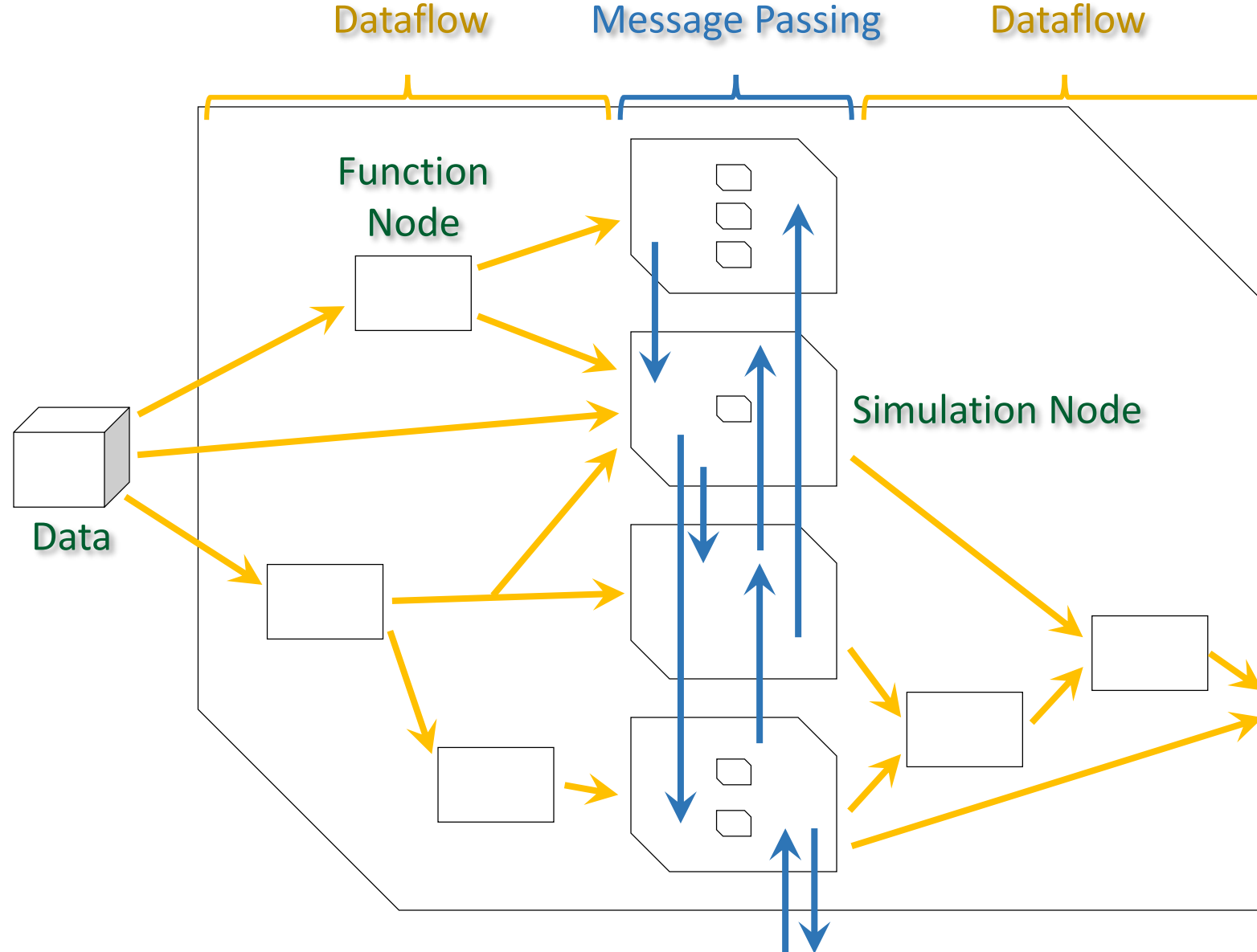
SyDEVS Approach



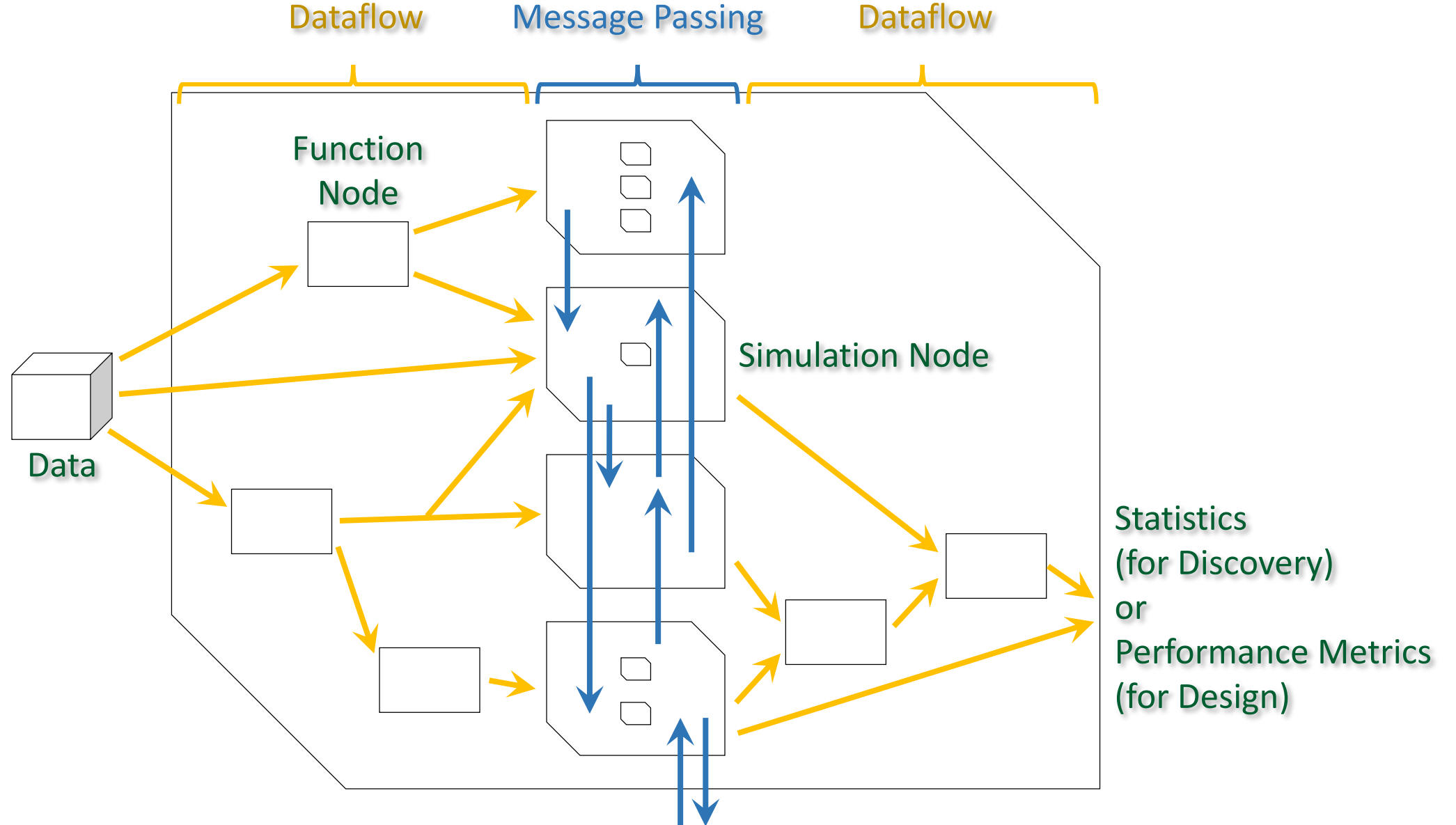
SyDEVS Approach



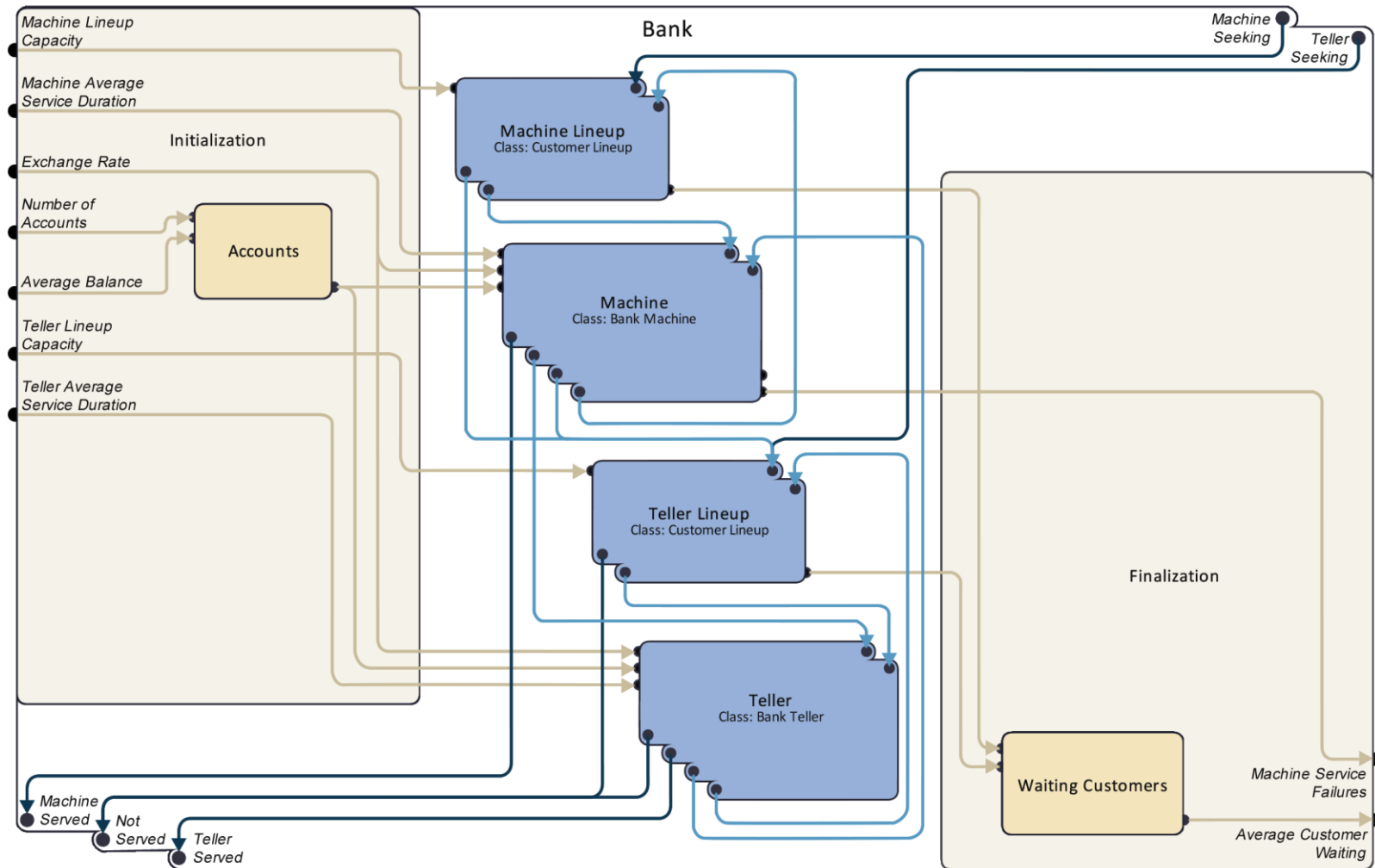
SyDEVS Approach



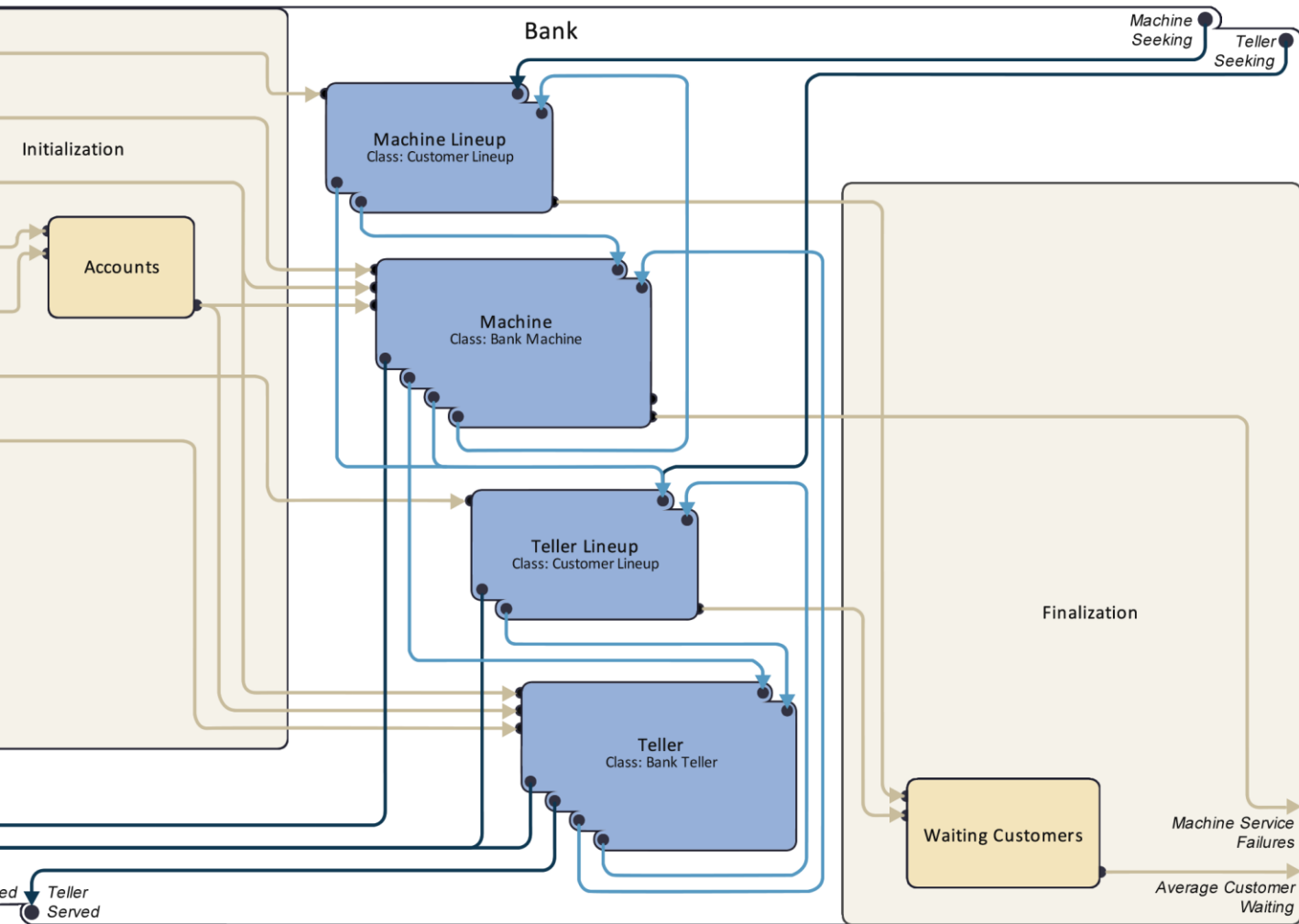
SyDEVS Approach



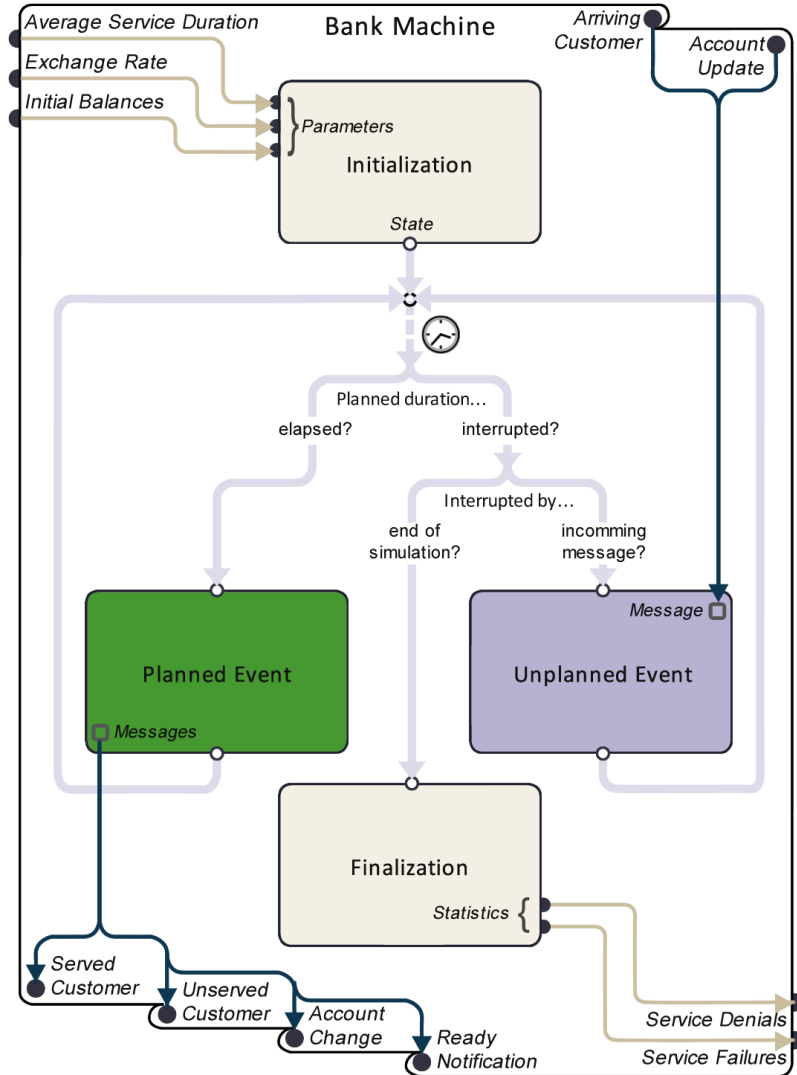
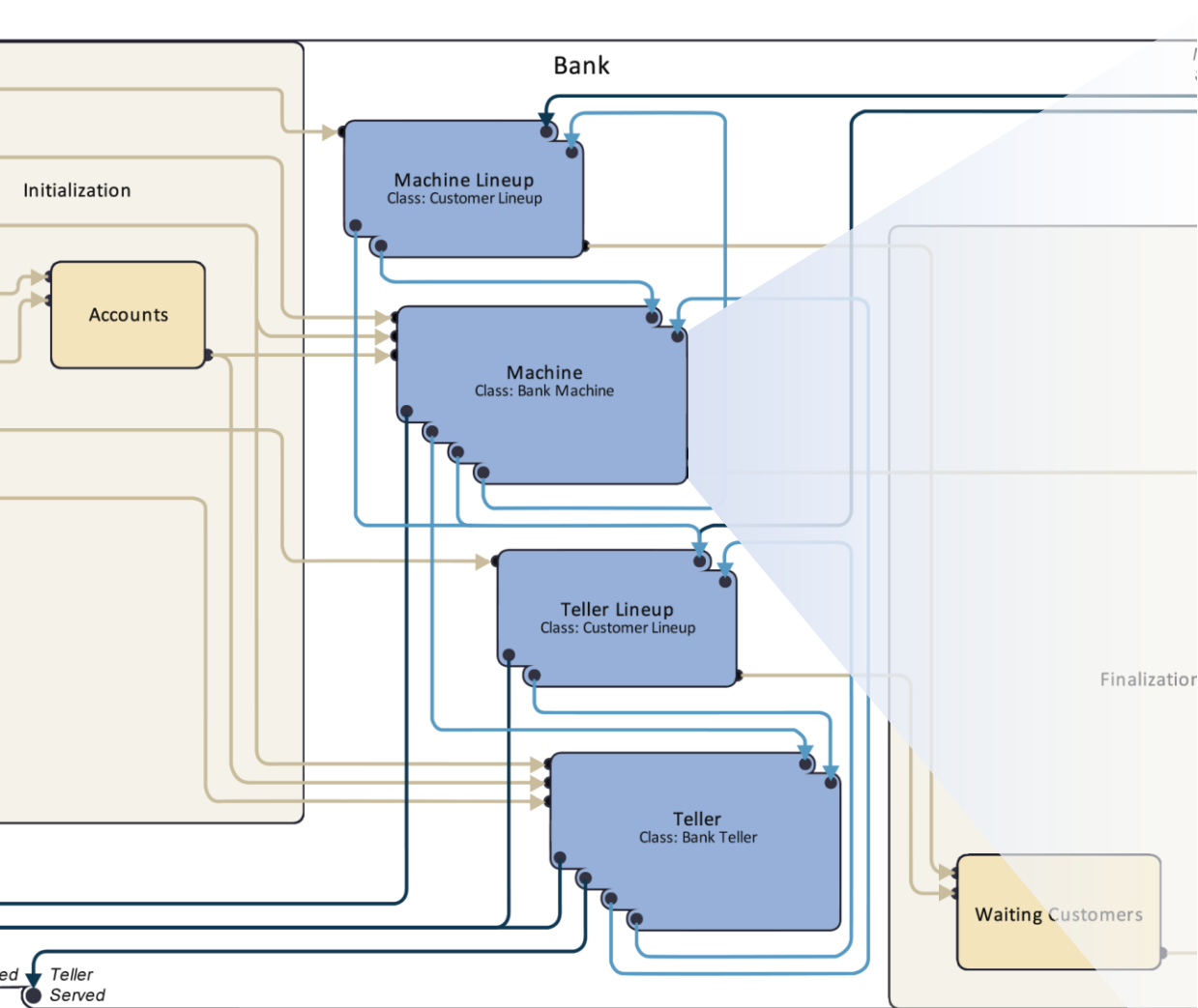
Proposed Visual Interface



Proposed Visual Interface



Proposed Visual Interface



Proposed Visual Interface

DEVS

From Wikipedia, the free encyclopedia

DEVS abbreviating **Discrete Event System Specification** is a modular and hierarchical formalism for m

...

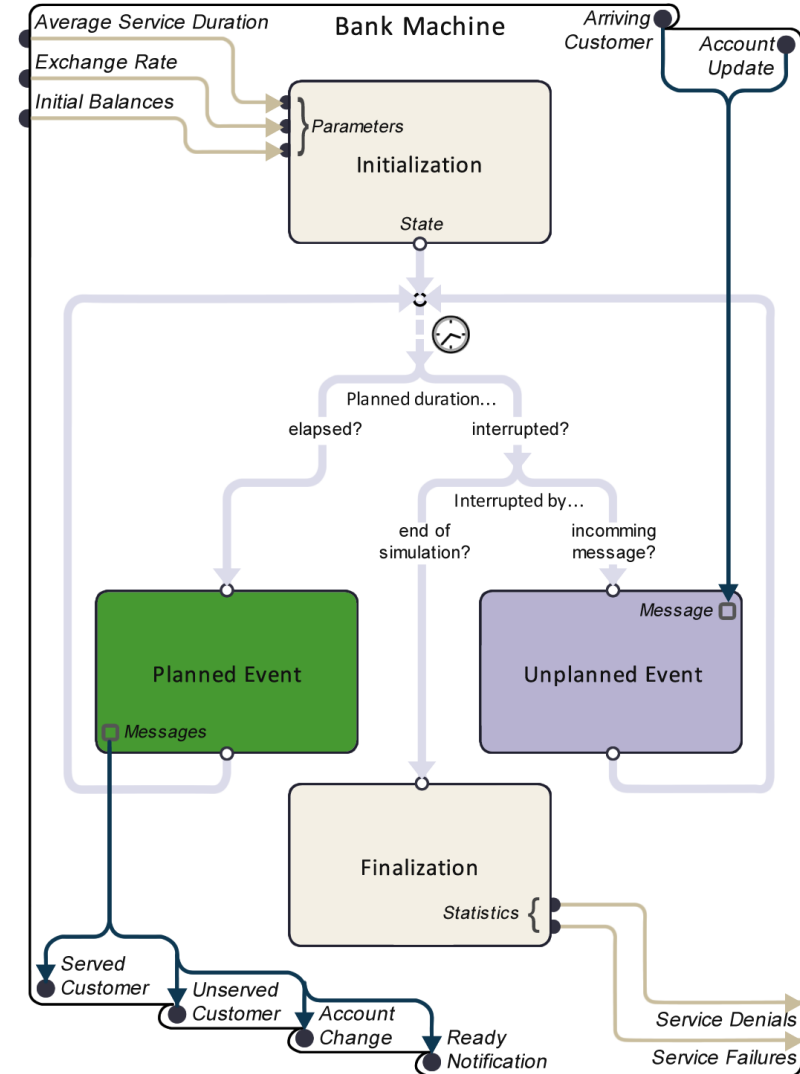
Atomic DEVS [\[edit \]](#)

An atomic DEVS model is defined as a 7-tuple

$$M = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

where

- X is the set of input events;
- Y is the set of output events;
- S is the set of sequential states (or also called the set of partial states);
- $s_0 \in S$ is the initial state;
- $ta : S \rightarrow \mathbb{T}^\infty$ is the time advance function which is used to determine the lifespan of a state;
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function which defines how an input event change $Q = \{(s, t_e) | s \in S, t_e \in (\mathbb{T} \cap [0, ta(s)])\}$ is the set of total states, and t_e is the elapsed time;
- $\delta_{int} : S \rightarrow S$ is the internal transition function which defines how a state of the system change time reaches to the lifetime of the state);
- $\lambda : S \rightarrow Y^\phi$ is the output function where $Y^\phi = Y \cup \{\phi\}$ and $\phi \notin Y$ is a silent event or an internal event; it defines how a state of the system generates an output event (when the elapsed time reaches to the lifetime of the state);



Proposed Visual Interface

DEVS

From Wikipedia, the free encyclopedia

DEVS abbreviating **Discrete Event System Specification** is a modular and hierarchical formalism for m

...

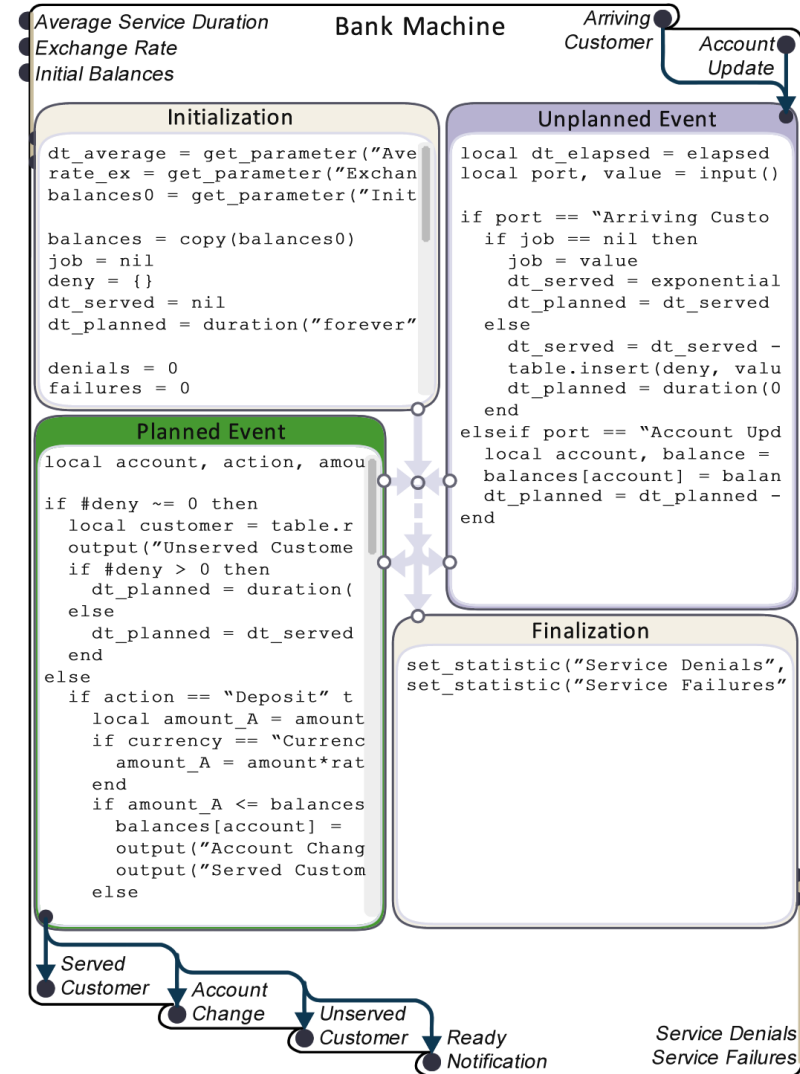
Atomic DEVS [\[edit \]](#)

An atomic DEVS model is defined as a 7-tuple

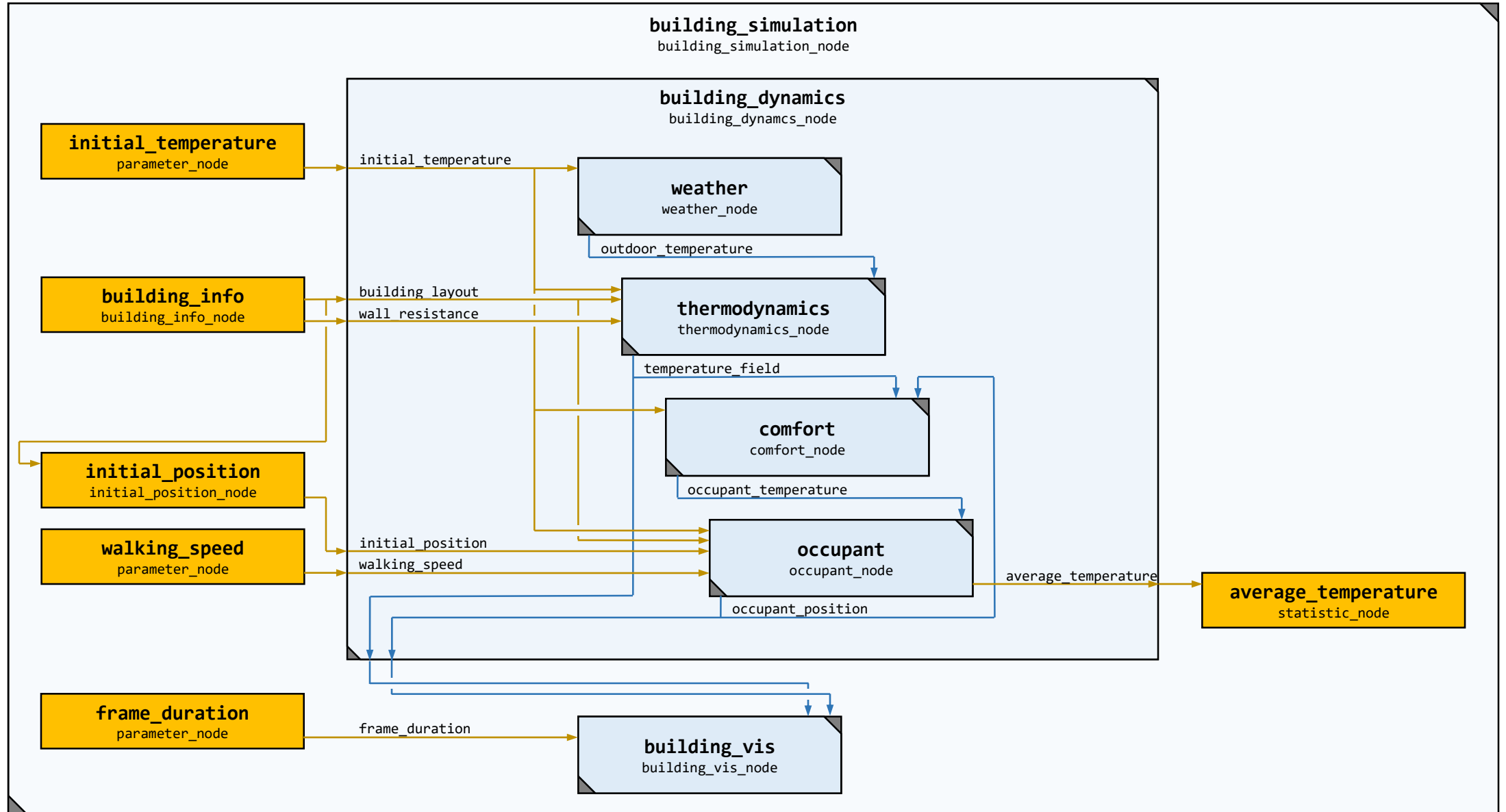
$$M = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

where

- X is the set of input events;
- Y is the set of output events;
- S is the set of sequential states (or also called the set of partial states);
- $s_0 \in S$ is the initial state;
- $ta : S \rightarrow \mathbb{T}^\infty$ is the time advance function which is used to determine the lifespan of a state;
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function which defines how an input event change $Q = \{(s, t_e) | s \in S, t_e \in (\mathbb{T} \cap [0, ta(s)])\}$ is the set of total states, and t_e is the elapsed time;
- $\delta_{int} : S \rightarrow S$ is the internal transition function which defines how a state of the system change time reaches to the lifetime of the state);
- $\lambda : S \rightarrow Y^\phi$ is the output function where $Y^\phi = Y \cup \{\phi\}$ and $\phi \notin Y$ is a silent event or an internal event; it defines how a state of the system generates an output event (when the elapsed time reaches to the lifetime of the state);



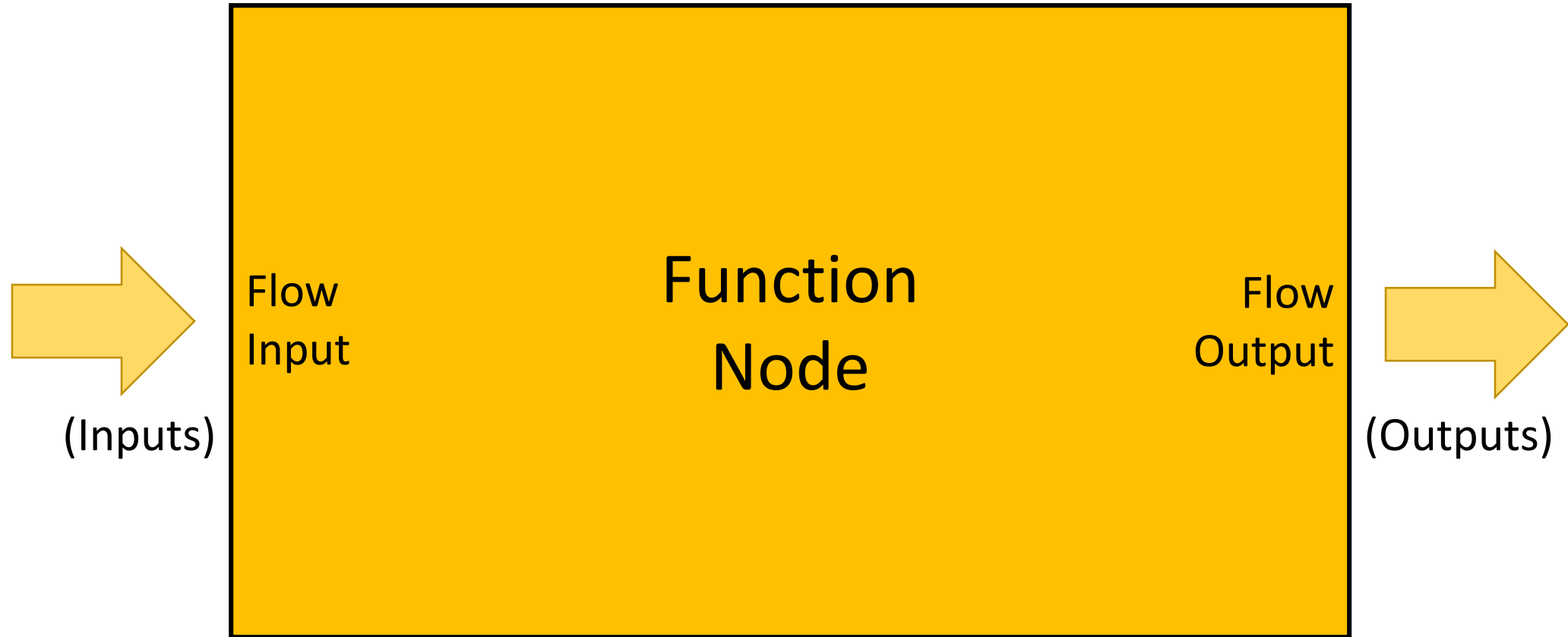
Example



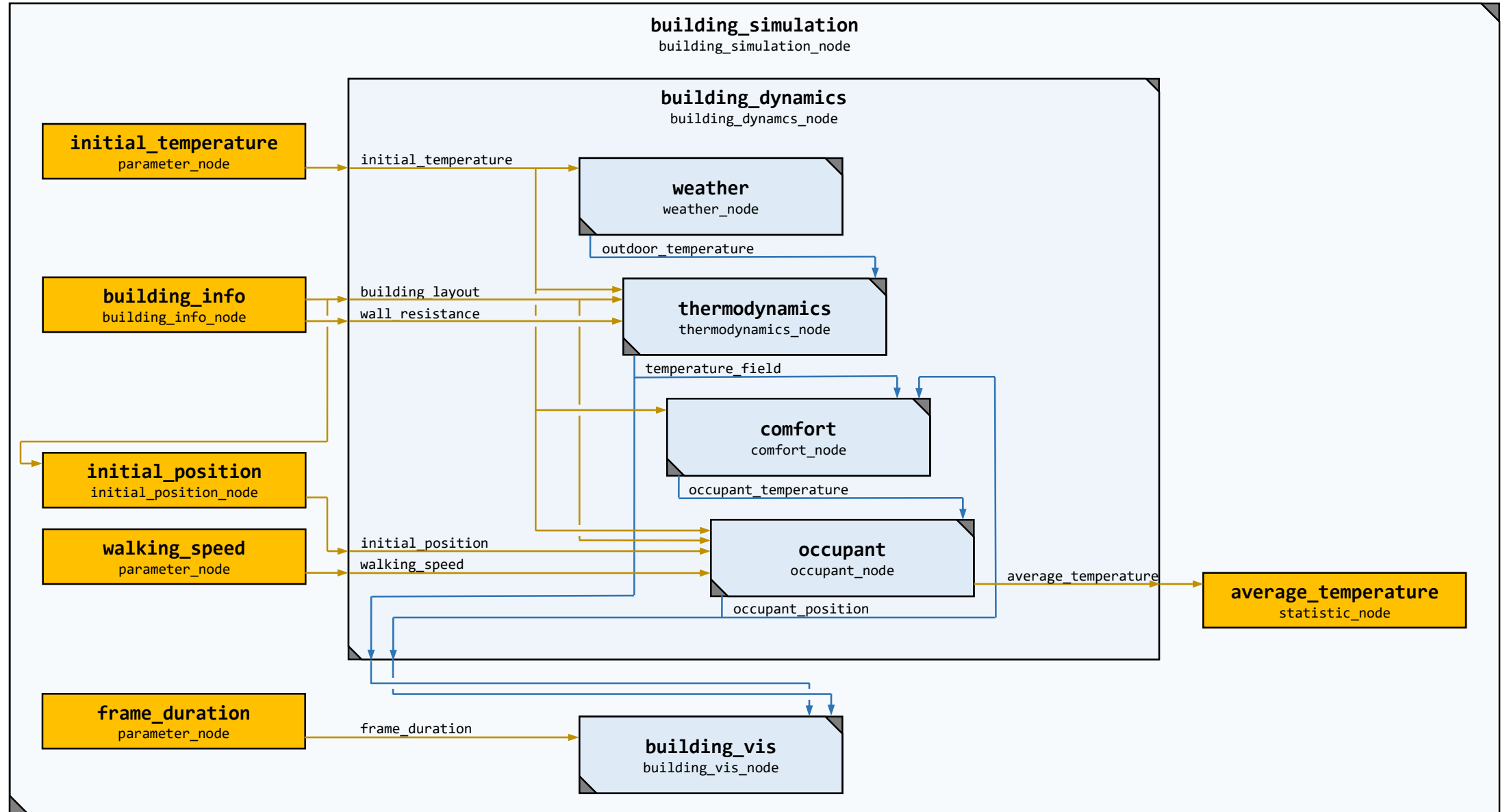
Example – Function Nodes



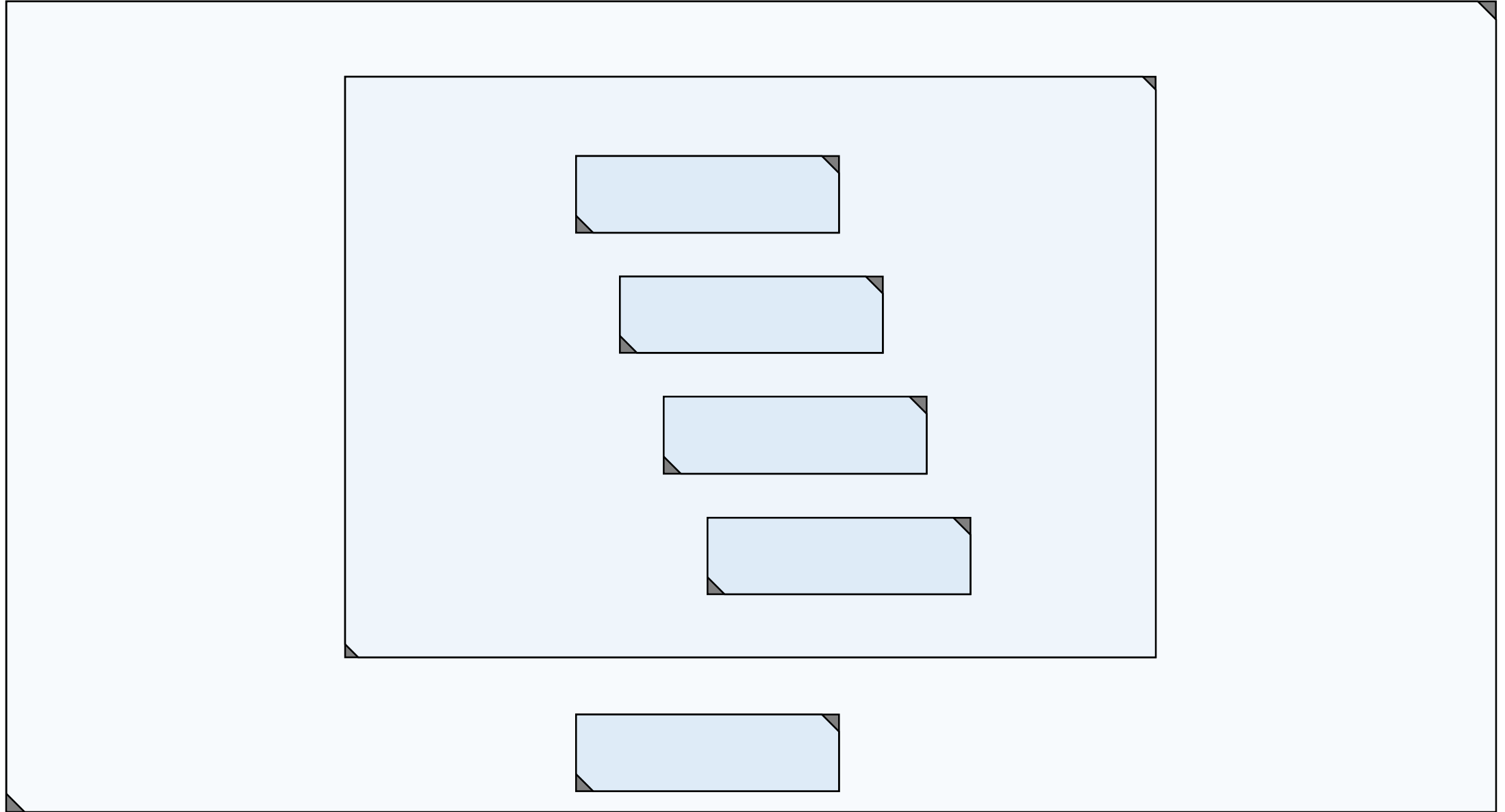
Example – Function Nodes



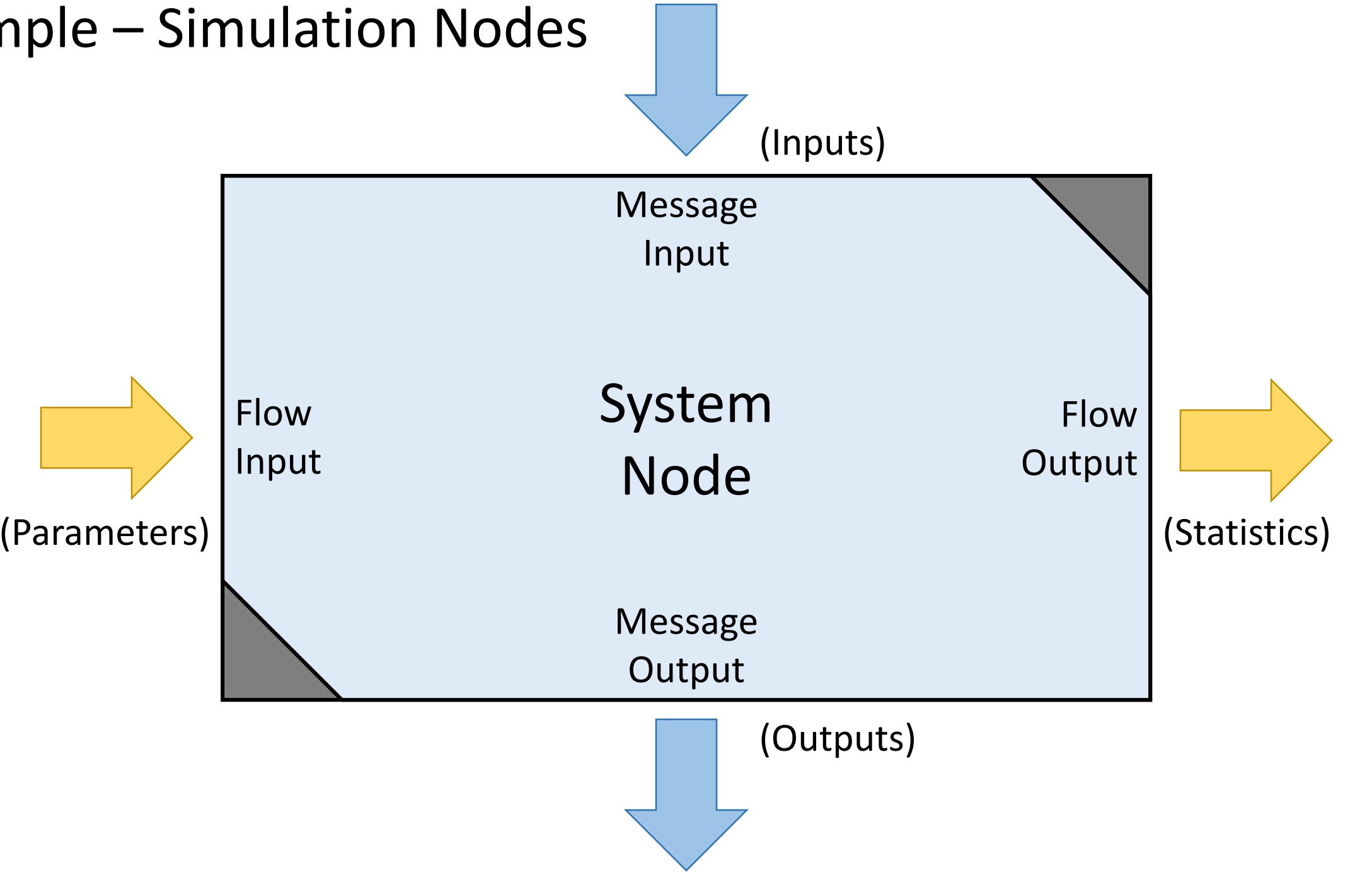
Example



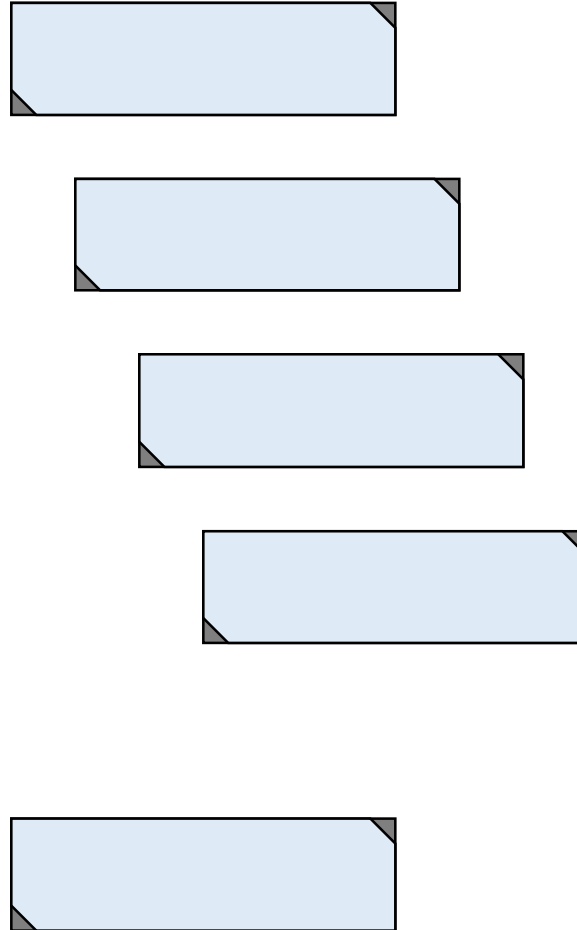
Example – Simulation Nodes



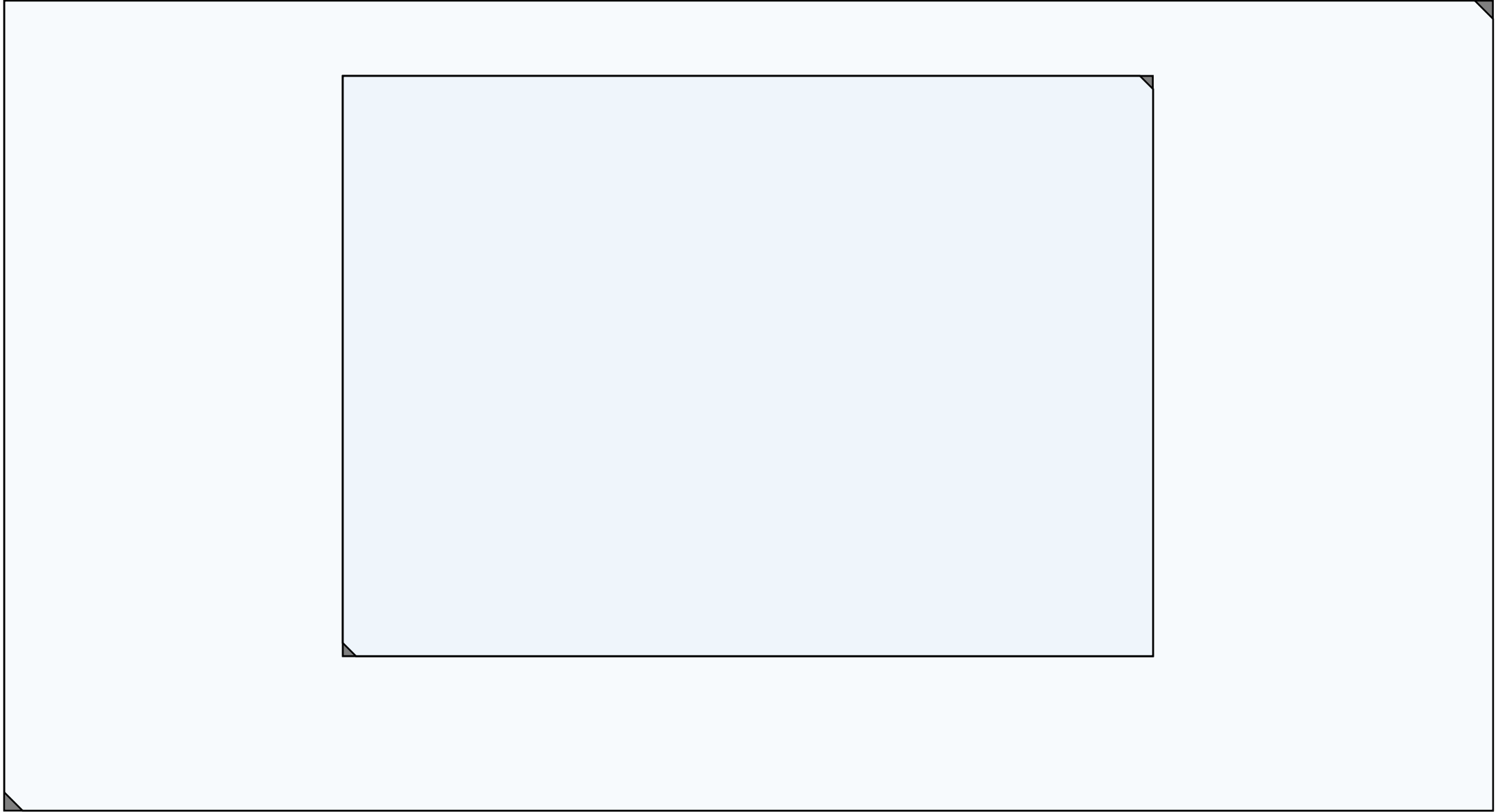
Example – Simulation Nodes



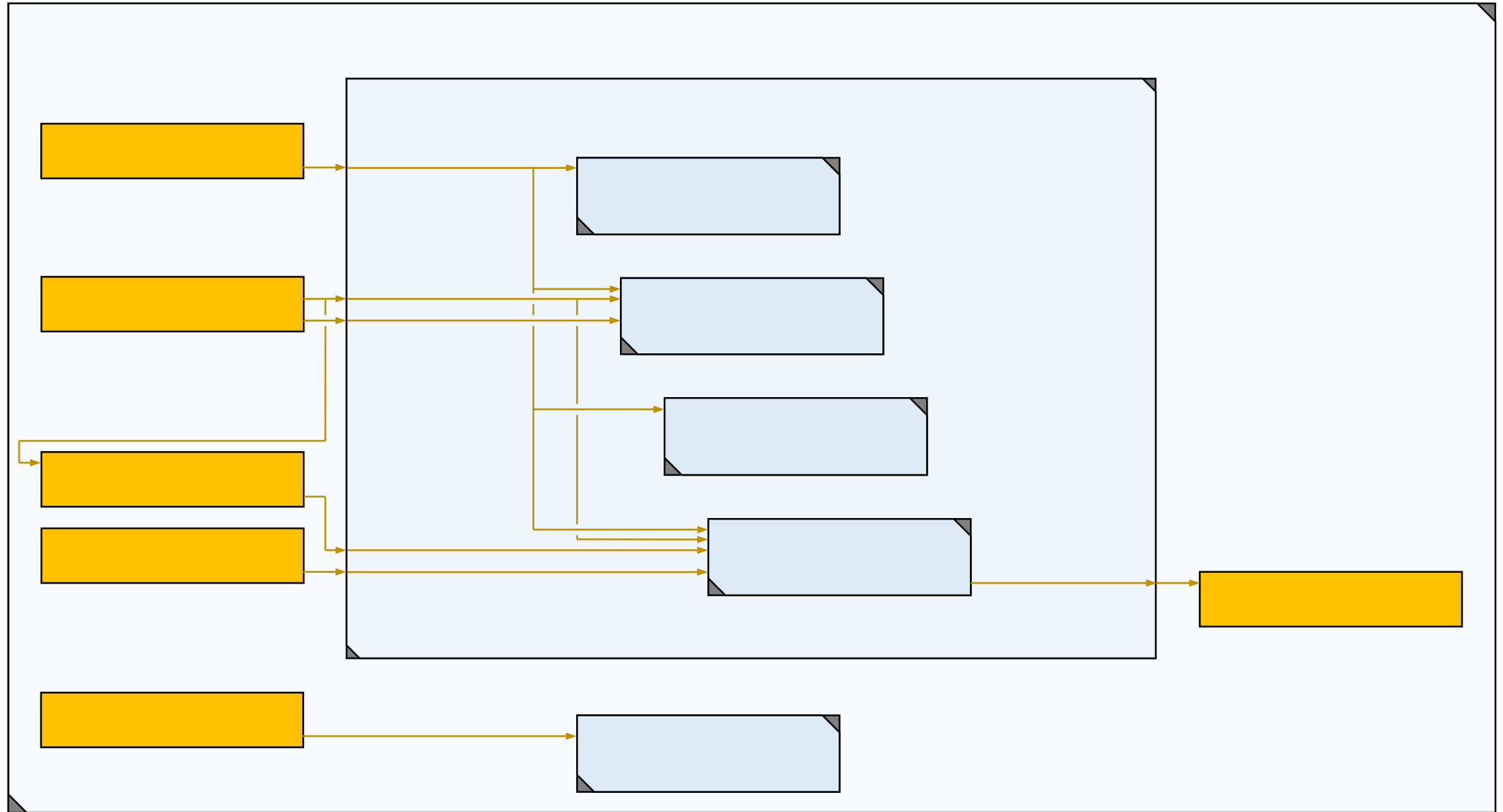
Example – Atomic Nodes



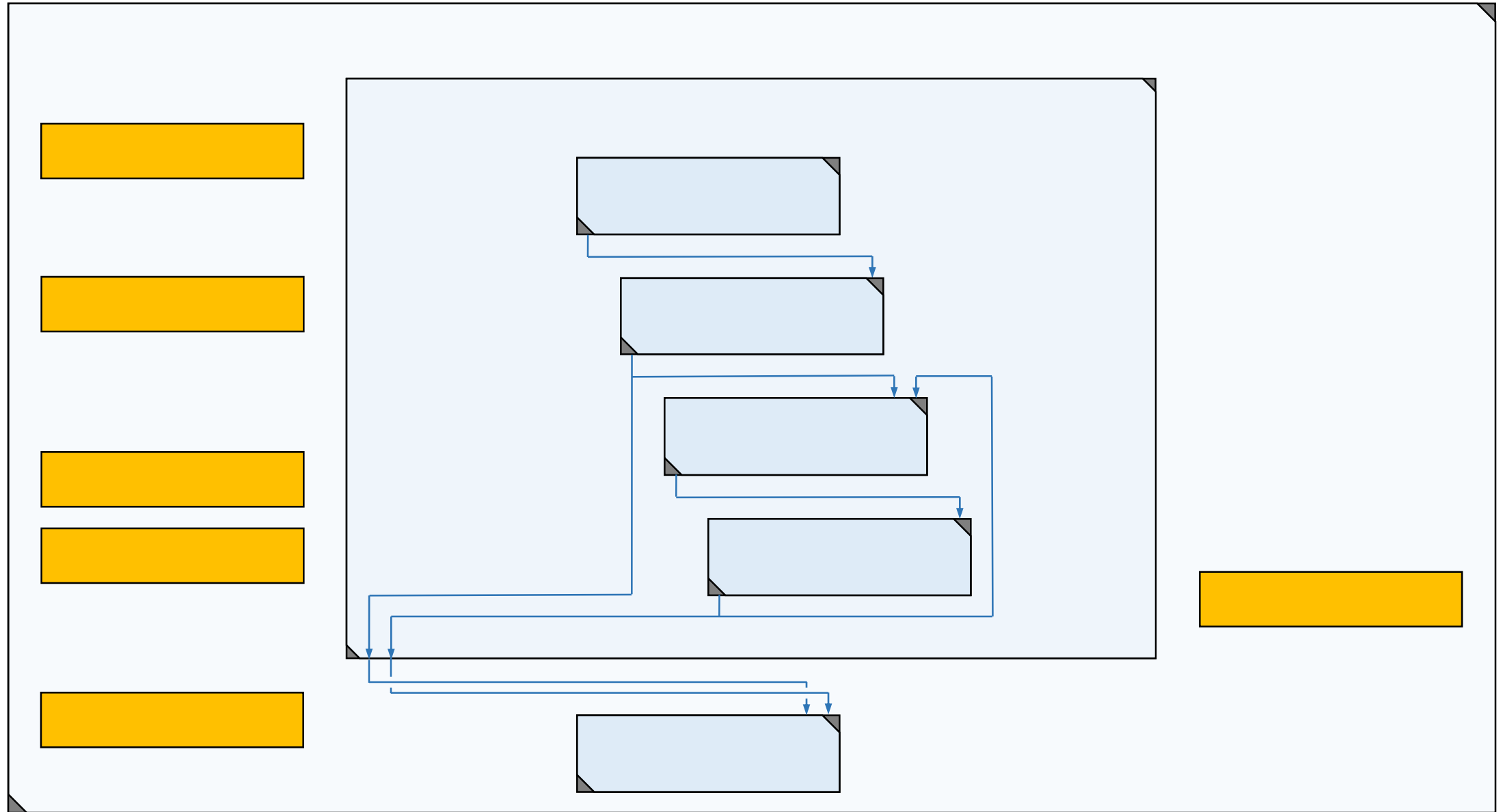
Example – Composite Nodes



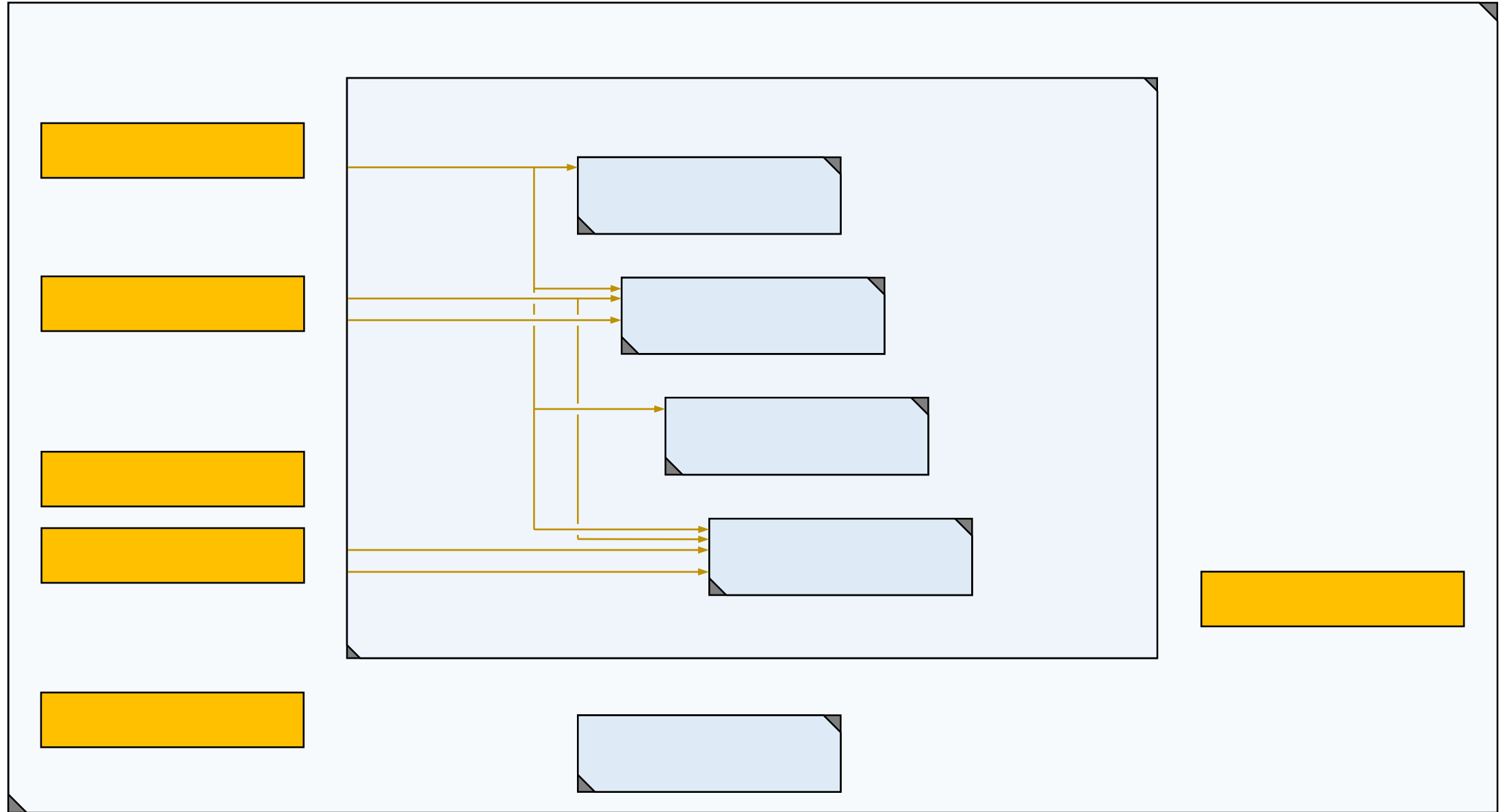
Example – Flow Links



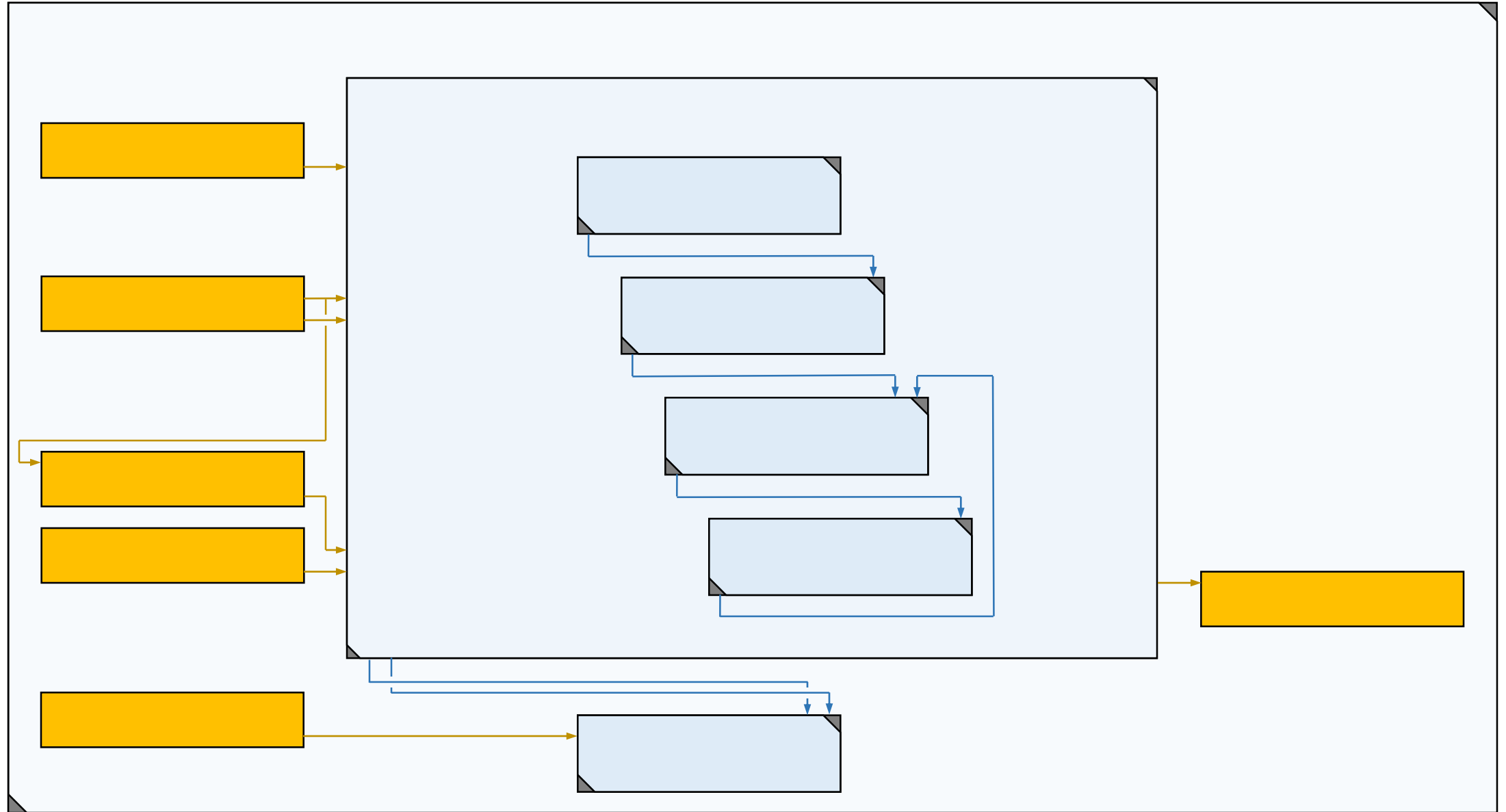
Example – Message Links



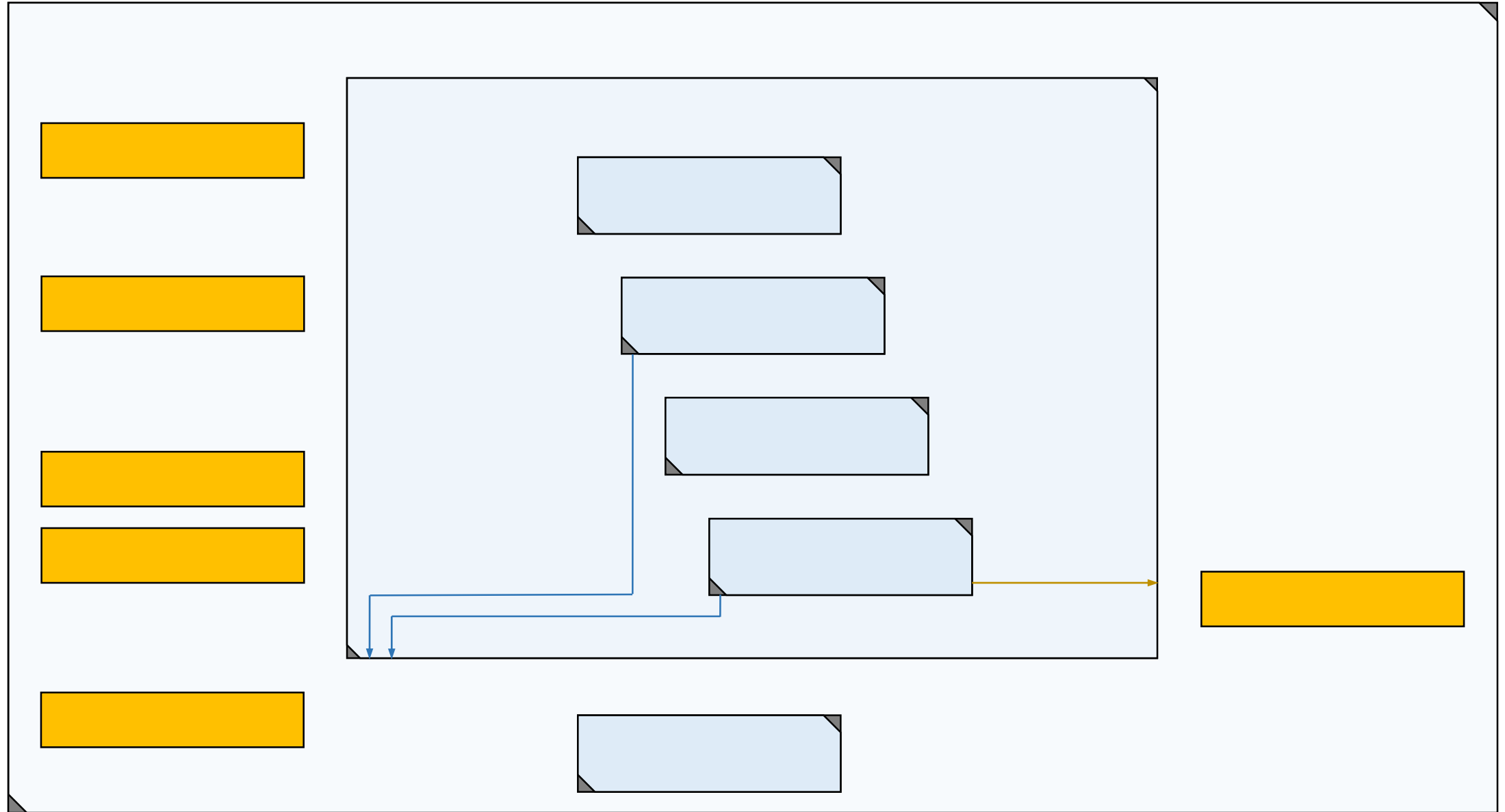
Example – Inward Links



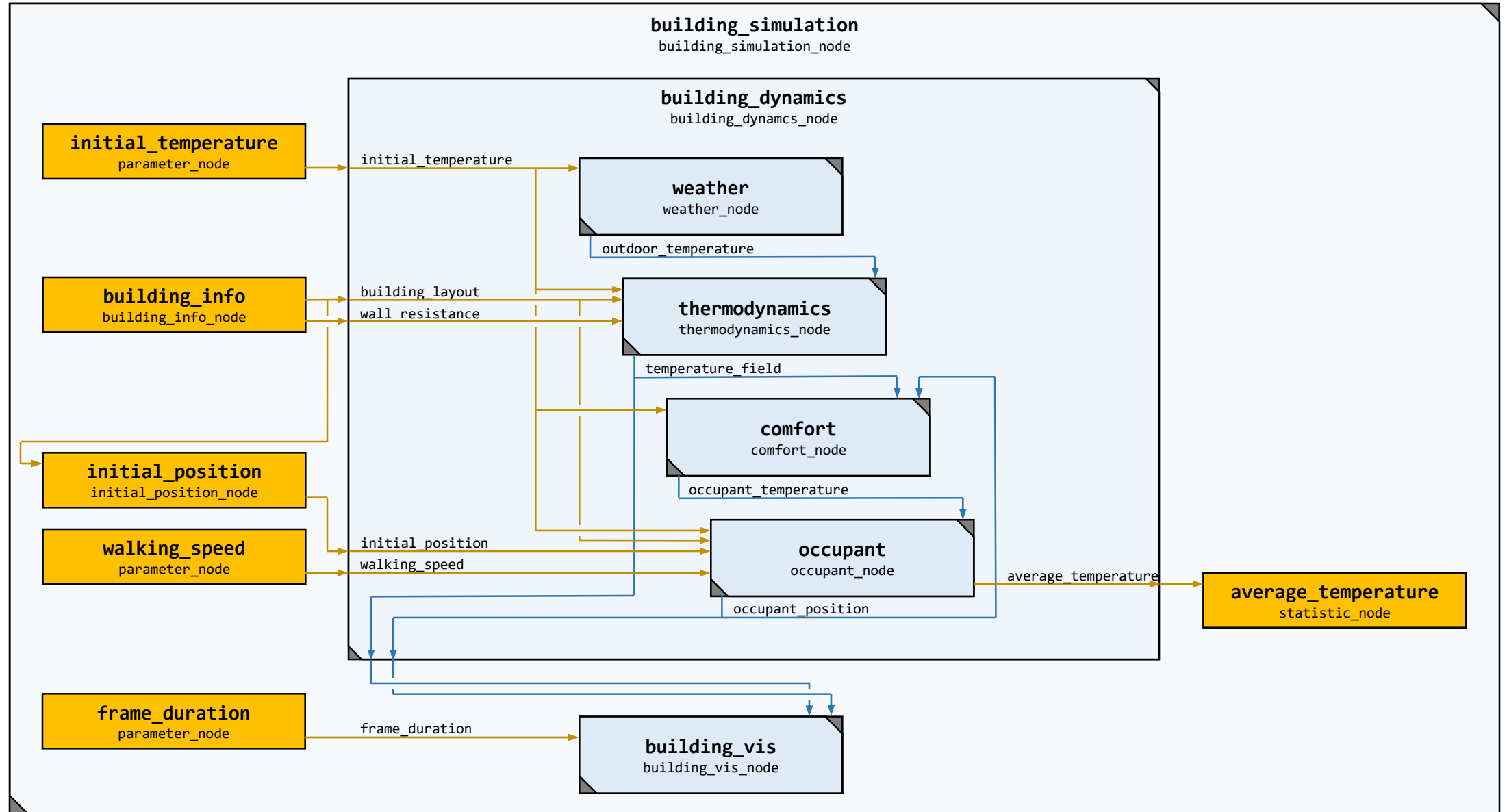
Example – Inner Links



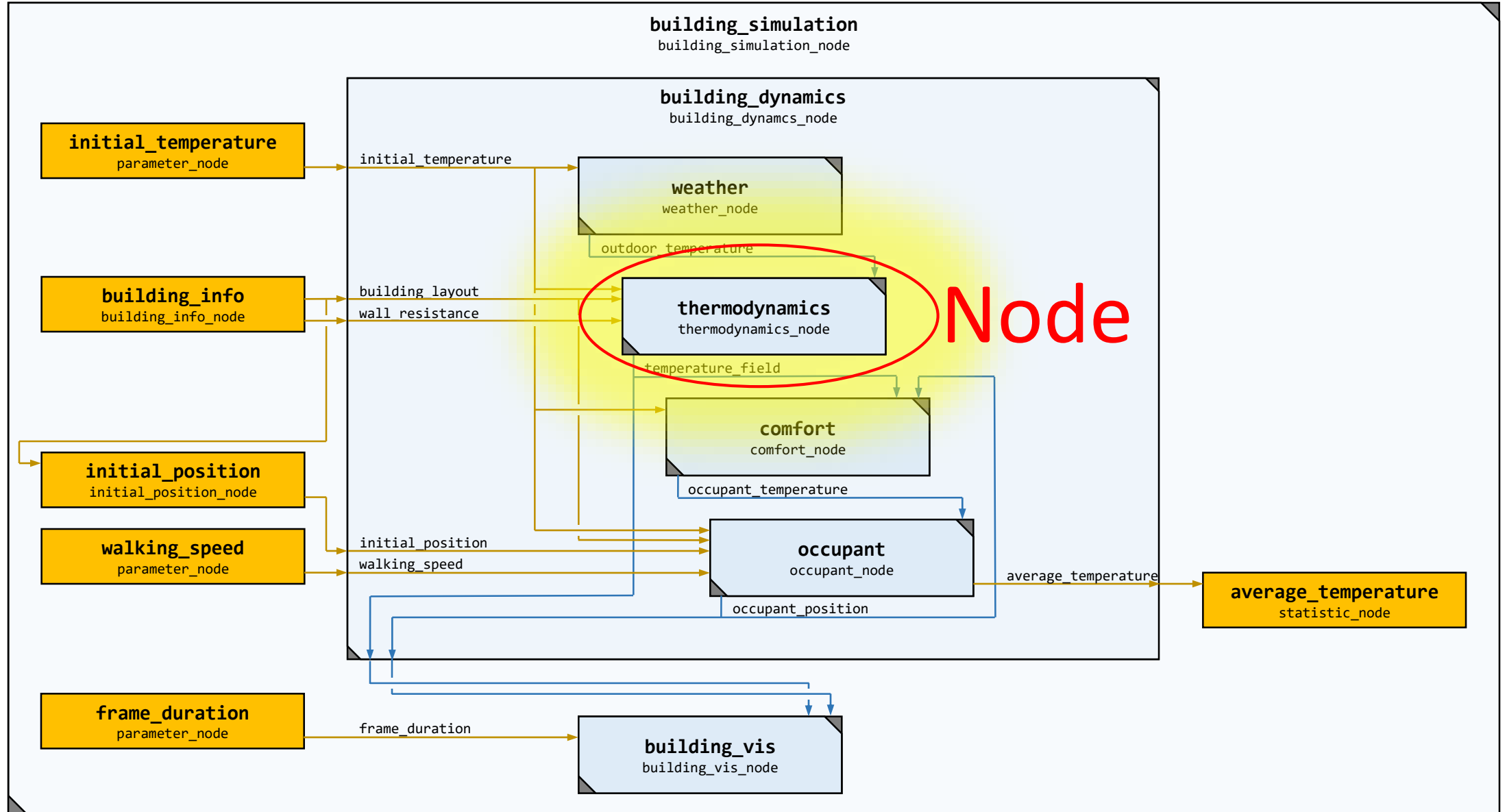
Example – Outward Links



Example



Example



Node

```
class thermodynamics_node: public atomic_node
{
public:
    port<flow, input, thermodynamic_temperature> initial_temperature_input;
    port<flow, input, std::pair<array2d<int64>, distance>> building_layout_input;
    port<flow, input, float64> wall_resistance_input;
    port<message, input, thermodynamic_temperature> outdoor_temperature_input;
    port<message, output, array2d<thermodynamic_temperature>> temperature_field_output;

protected:
    array2d<int64> L; // building layout
    int64 nx; // number of cells in the x dimension
    int64 ny; // number of cells in the y dimension
    float64 wall_R; // wall resistance
    array2d<thermodynamic_temperature> TF; // temperature field
    duration step_dt; // time step
    duration planned_dt; // planned duration

    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);
};
```


Types of Nodes

```
class thermodynamics_node : public atomic_node
{
public:
    port<flow, input, thermodynamic_temperature> initial_temperature_input;
    port<flow, input, std::pair<array2d<int64>, distance>> building_layout_input;
    port<flow, input, float64> wall_resistance_input;
    port<message, input, thermodynamic_temperature> outdoor_temperature_input;
    port<message, output, array2d<thermodynamic_temperature>> temperature_field_output;

protected:
    array2d<int64> L; // building layout
    int64 nx; // number of cells in the x dimension
    int64 ny; // number of cells in the y dimension
    float64 wall_R; // wall resistance
    array2d<thermodynamic_temperature> TF; // temperature field
    duration step_dt; // time step
    duration planned_dt; // planned duration

    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);
};
```

Types of Nodes

```
class system_node
{
};
```

```
class atomic_node: public system_node
{
    virtual duration initialization_event() = 0;
    virtual duration unplanned_event(duration elapsed_dt) = 0;
    virtual duration planned_event(duration elapsed_dt) = 0;
    virtual void finalization_event(duration elapsed_dt) = 0;
};
```

```
class composite_node : public system_node
{
    void inward_link(port<dmode, input, T>& src_port, port<dmode, input, T>& dst_port);
    void inner_link(port<dmode, output, T>& src_port, port<dmode, input, T>& dst_port);
    void outward_link(port<dmode, output, T>& src_port, port<dmode, output, T>& dst_port);
}
```

```
class function_node : public system_node
{
    virtual void flow_event() = 0;
};
```

```
// Also...
class collection_node ...
```

```
class system_node
{
};
```

```
class atomic_node: public system_node
{
    virtual duration initialization_event() = 0;
    virtual duration unplanned_event(duration elapsed_dt) = 0;
    virtual duration planned_event(duration elapsed_dt) = 0;
    virtual void finalization_event(duration elapsed_dt) = 0;
};
```

Event Handlers

```
class composite_node : public system_node
{
    void inward_link(port<dmode, input, T>& src_port, port<dmode, input, T>& dst_port);
    void inner_link(port<dmode, output, T>& src_port, port<dmode, input, T>& dst_port);
    void outward_link(port<dmode, output, T>& src_port, port<dmode, output, T>& dst_port);
}
```

```
class function_node : public system_node
{
    virtual void flow_event() = 0;
};
```

```
// Also...
class collection_node ...
```

Visual Interface

DEVS

From Wikipedia, the free encyclopedia

DEVS abbreviating **Discrete Event System Specification** is a modular and hierarchical formalism for m

...

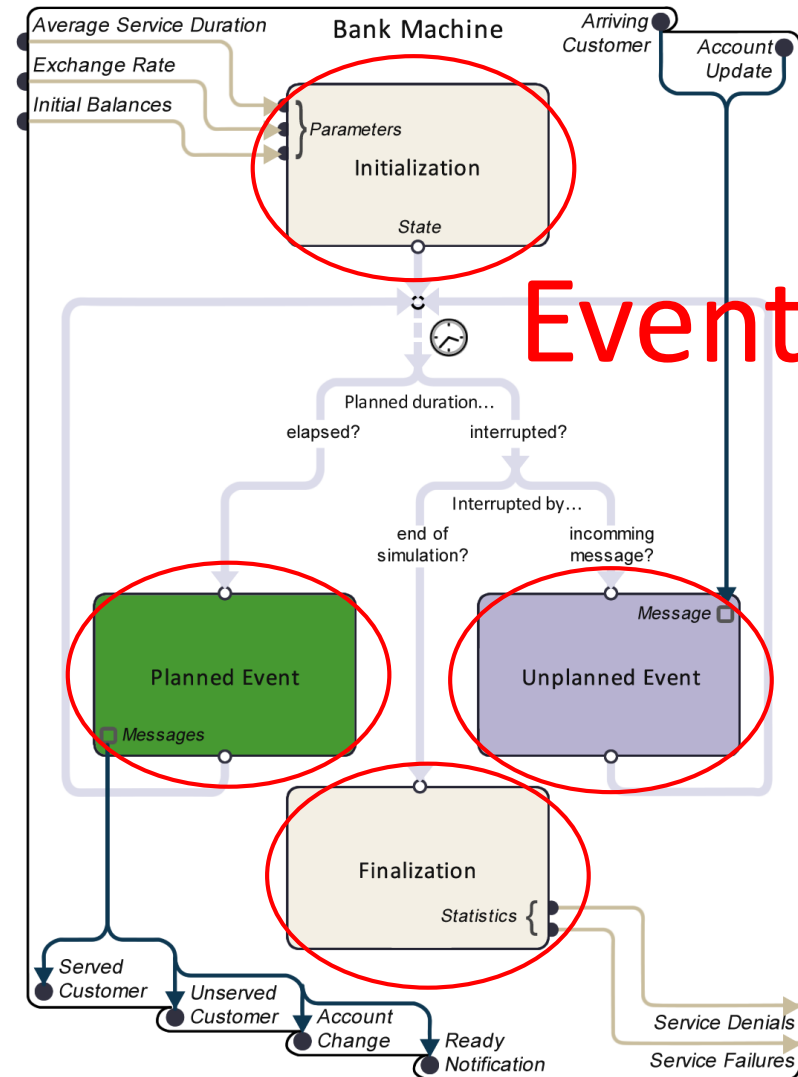
Atomic DEVS [\[edit \]](#)

An atomic DEVS model is defined as a 7-tuple

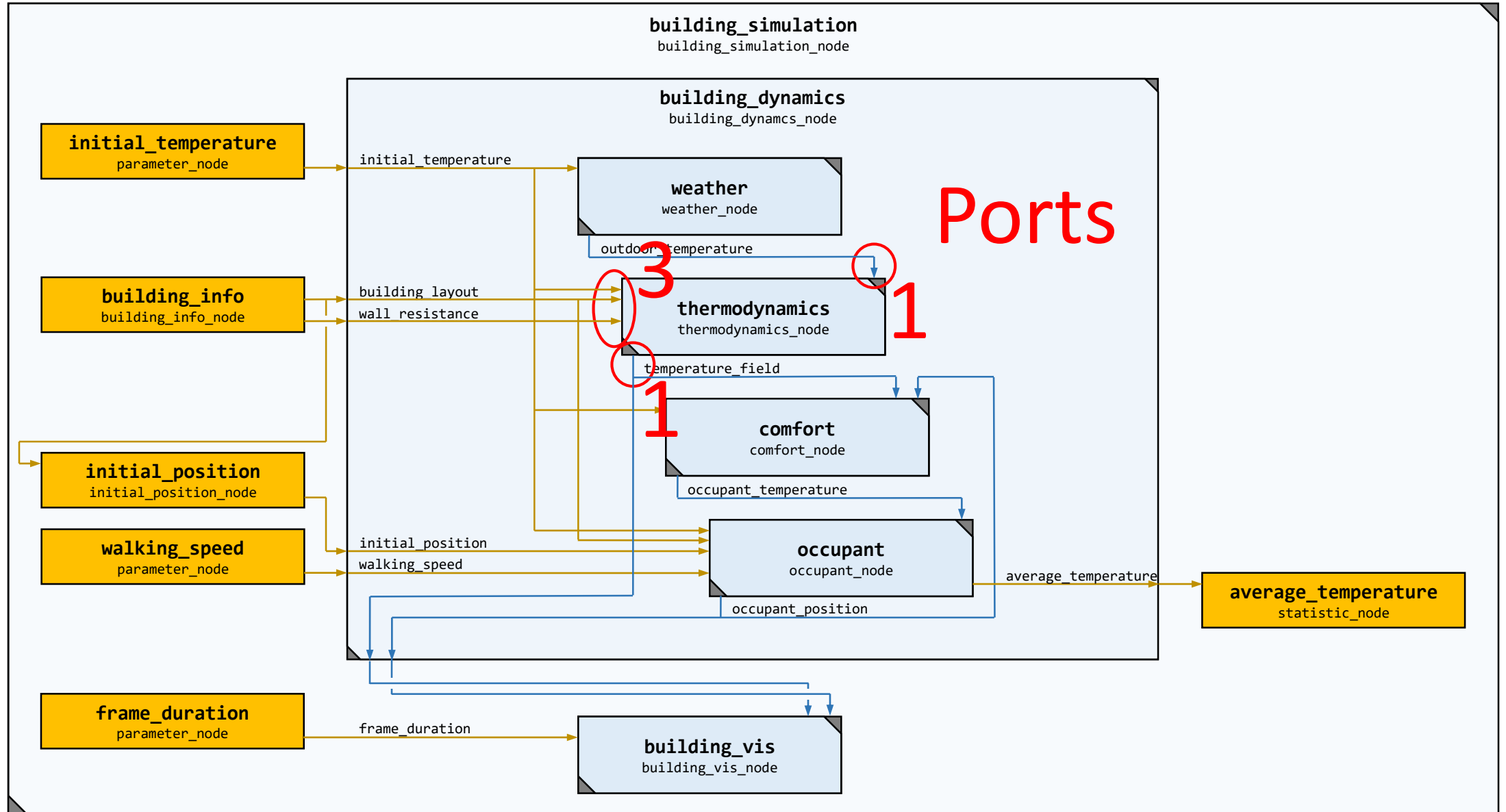
$$M = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

where

- X is the set of input events;
- Y is the set of output events;
- S is the set of sequential states (or also called the set of partial states);
- $s_0 \in S$ is the initial state;
- $ta : S \rightarrow \mathbb{T}^\infty$ is the time advance function which is used to determine the lifespan of a state;
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function which defines how an input event change $Q = \{(s, t_e) | s \in S, t_e \in (\mathbb{T} \cap [0, ta(s)])\}$ is the set of total states, and t_e is the elapsed time;
- $\delta_{int} : S \rightarrow S$ is the internal transition function which defines how a state of the system change time reaches to the lifetime of the state);
- $\lambda : S \rightarrow Y^\phi$ is the output function where $Y^\phi = Y \cup \{\phi\}$ and $\phi \notin Y$ is a silent event or an internal event; it defines how a state of the system generates an output event (when the elapsed time reaches to the lifetime of the state);



Example



Ports

```

class thermodynamics_node : public atomic_node
{
public:
    port<flow, input, thermodynamic_temperature> initial_temperature_input;
    port<flow, input, std::pair<array2d<int64>, distance>> building_layout_input;
    port<flow, input, float64> wall_resistance_input;
    port<message, input, thermodynamic_temperature> outdoor_temperature_input;
    port<message, output, array2d<thermodynamic_temperature>> temperature_field_output;

protected:
    array2d<int64> L; // building layout
    int64 nx; // number of cells in the x dimension
    int64 ny; // number of cells in the y dimension
    float64 wall_R; // wall resistance
    array2d<thermodynamic_temperature> TF; // temperature field
    duration step_dt; // time step
    duration planned_dt; // planned duration

    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);
};

```

Data types

```
class thermodynamics_node : public atomic_node
{
public:
    port<flow, input, thermodynamic_temperature> initial_temperature_input;
    port<flow, input, std::pair<array2d<int64>, distance>> building_layout_input;
    port<flow, input, float64> wall_resistance_input;
    port<message, input, thermodynamic_temperature> outdoor_temperature_input;
    port<message, output, array2d<thermodynamic_temperature>> temperature_field_output;

protected:
    array2d<int64> L; // building layout
    int64 nx; // number of cells in the x dimension
    int64 ny; // number of cells in the y dimension
    float64 wall_R; // wall resistance
    array2d<thermodynamic_temperature> TF; // temperature field
    duration step_dt; // time step
    duration planned_dt; // planned duration

    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);
};
```

```
// Core Types
```

```
pointer // not sortable; encapsulates std::shared_ptr<void>
```

```
bool // sortable
```

```
int64 // sortable
```

```
float64 // sortable
```

```
std::string // sortable
```

```
quantity<U> // sortable; includes e.g. distance, duration, quantity<decltype(_kg*_m/_s/_s)>
```

Data types

```
arraynd<T, ndims> // valid if T is valid; not sortable
```

```
std::pair<T1, T2> // valid if T1, T2 are valid; sortable if T1, T2 are sortable
```

```
std::tuple<T> // valid if T is valid; sortable if T is sortable
```

```
std::tuple<T, Ts...> // valid if T, Ts... are valid; sortable if T, Ts... are sortable
```

```
std::vector<T> // valid if T is valid; sortable if T is sortable
```

```
std::set<T> // valid if T is valid; sortable if T is sortable
```

```
std::map<Key, T> // valid if T is valid and Key is valid and sortable;  
// sortable if T is sortable
```

```
std::shared_ptr<T> // not sortable
```

```
T default_value<T>() // create a default value of core type T
```

```
tostring(const T&) // convert the value of core type T to a string
```



```

#include <my_cpp_libraries/my_CFD_solver.h>

class CFD_node : public atomic_node
{
public:
    port<message, output, array3d<float>> velocities;

protected:
    my_CFD_solver_state CFD_state;

    virtual duration initialization_event()
    {
        CFD_state = initialize_my_CFD_solver();
        return 5_s; // planned_dt
    }

    virtual duration planned_event(duration elapsed_dt)
    {
        advance_one_time_step_using_my_CFD_solver(CFD_state);
        velocities.send(CFD_state.get_velocities);
        return 5_s; // planned_dt
    }
};

```

2nd Example

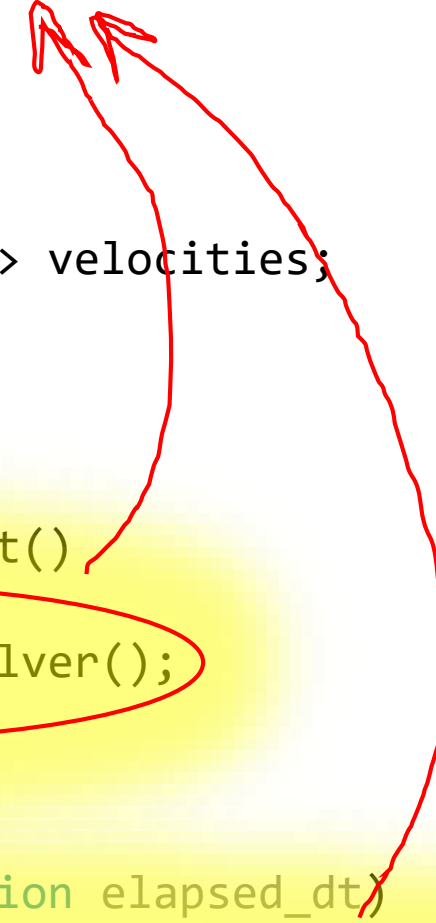
```
#include <my_cpp_libraries/my_CFD_solver.h>

class CFD_node : public atomic_node
{
public:
    port<message, output, array3d<float>> velocities;

protected:
    my_CFD_solver_state CFD_state;

    virtual duration initialization_event()
    {
        CFD_state = initialize_my_CFD_solver();
        return 5_s; // planned_dt
    }

    virtual duration planned_event(duration elapsed_dt)
    {
        advance_one_time_step_using_my_CFD_solver(CFD_state);
        velocities.send(CFD_state.get_velocities);
        return 5_s; // planned_dt
    }
};
```



Possible to wrap
existing simulation
libraries (eg. CFD)

2nd
Example

```

#include <my_cpp_libraries/my_CFD_solver.h>

class CFD_node : public atomic_node
{
public:
    port<message, output, array3d<float>> velocities;

protected:
    my_CFD_solver_state CFD_state;

    virtual duration initialization_event()
    {
        CFD_state = initialize_my_CFD_solver();
        return 5_s; // planned_dt
    }

    virtual duration planned_event(duration elapsed_dt)
    {
        advance_one_time_step_using_my_CFD_solver(CFD_state);
        velocities.send(CFD_state.get_velocities);
        return 5_s; // planned_dt
    }
};

```

The library includes
multidimensional
arrays

2nd
Example

```

#include <my_cpp_libraries/my_CFD_solver.h>

class CFD_node : public atomic_node
{
public:
    port<message, output, array3d<float>> velocities;

protected:
    my_CFD_solver_state CFD_state;

    virtual duration initialization_event()
    {
        CFD_state = initialize_my_CFD_solver();
        return 5_s; // planned_dt
    }

    virtual duration planned_event(duration elapsed_dt)
    {
        advance_one_time_step_using_my_CFD_solver(CFD_state);
        velocities.send(CFD_state.get_velocities);
        return 5_s; // planned_dt
    }
};

```

Elapsed durations and planned durations are important concepts

2nd
Example

```
#include <my_cpp_libraries/my_CFD_solver.h>

class CFD_node : public atomic_node
{
public:
    port<message, output, array3d<float>> velocities;

protected:
    my_CFD_solver_state CFD_state;

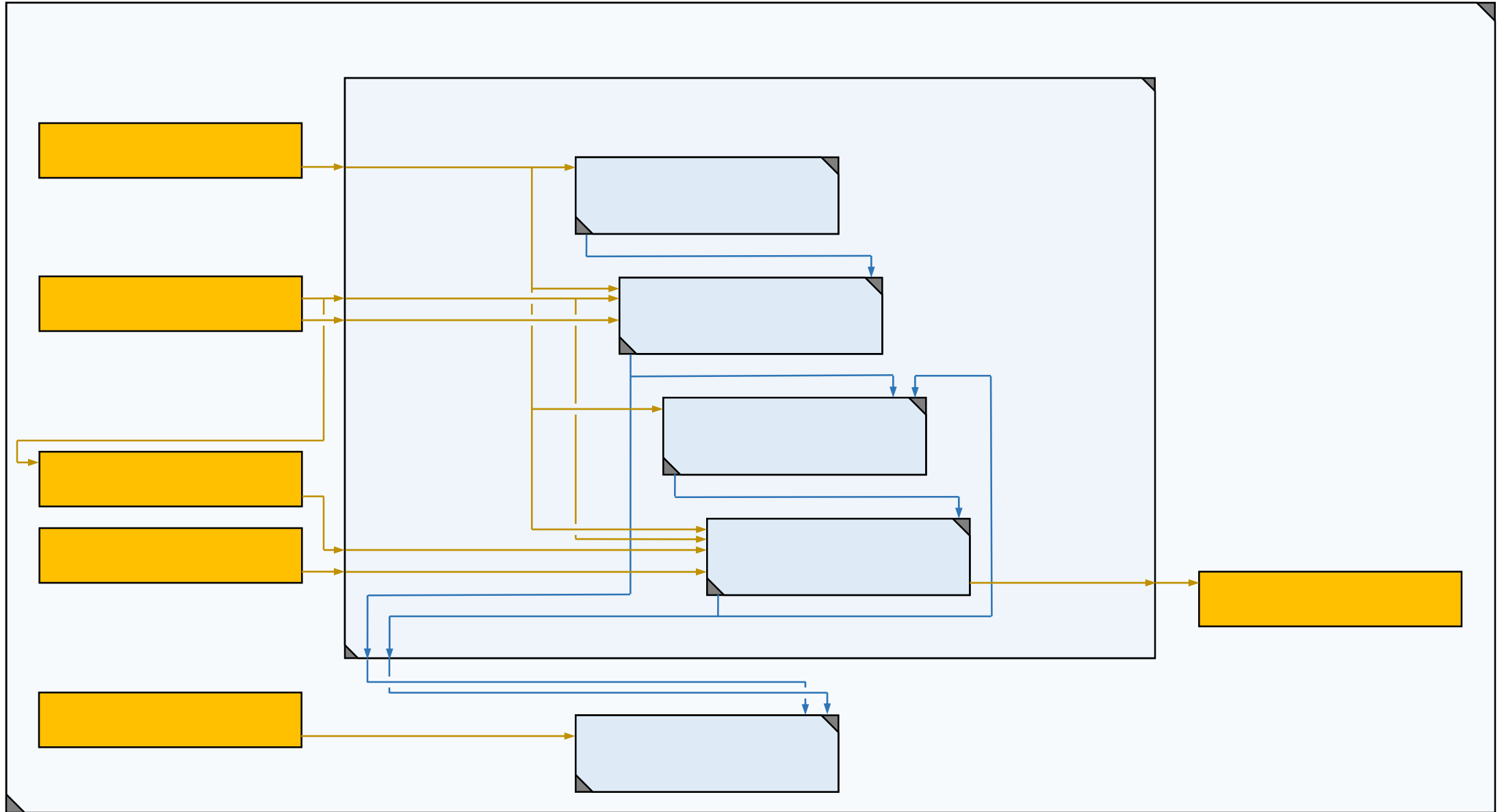
    virtual duration initialization_event()
    {
        CFD_state = initialize_my_CFD_solver();
        return 5_s; // planned_dt
    }

    virtual duration planned_event(duration elapsed_dt)
    {
        advance_one_time_step_using_my_CFD_solver(CFD_state);
        velocities.send(CFD_state.get_velocities);
        return 5_s; // planned_dt
    }
};
```

SI units are represented
explicitly and checked
at compile-time

2nd
Example

Review



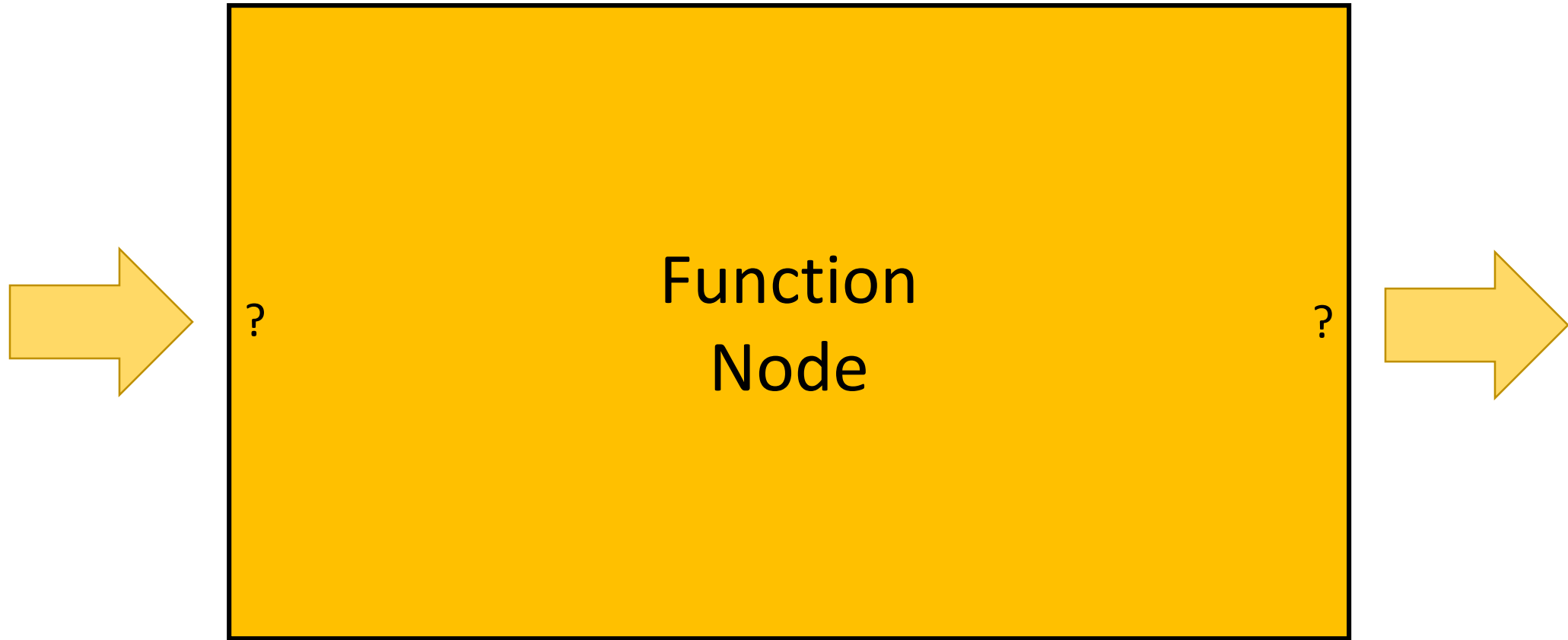
Review – ?



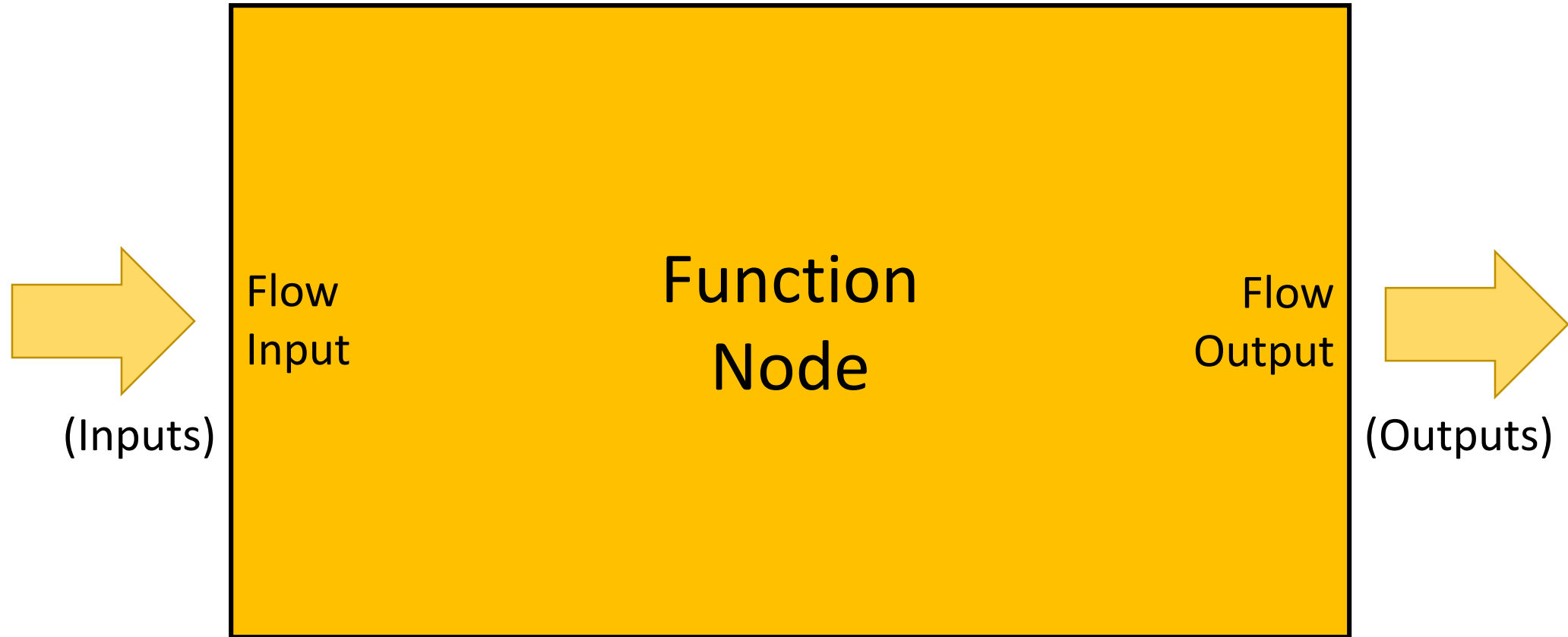
Review – Function Nodes



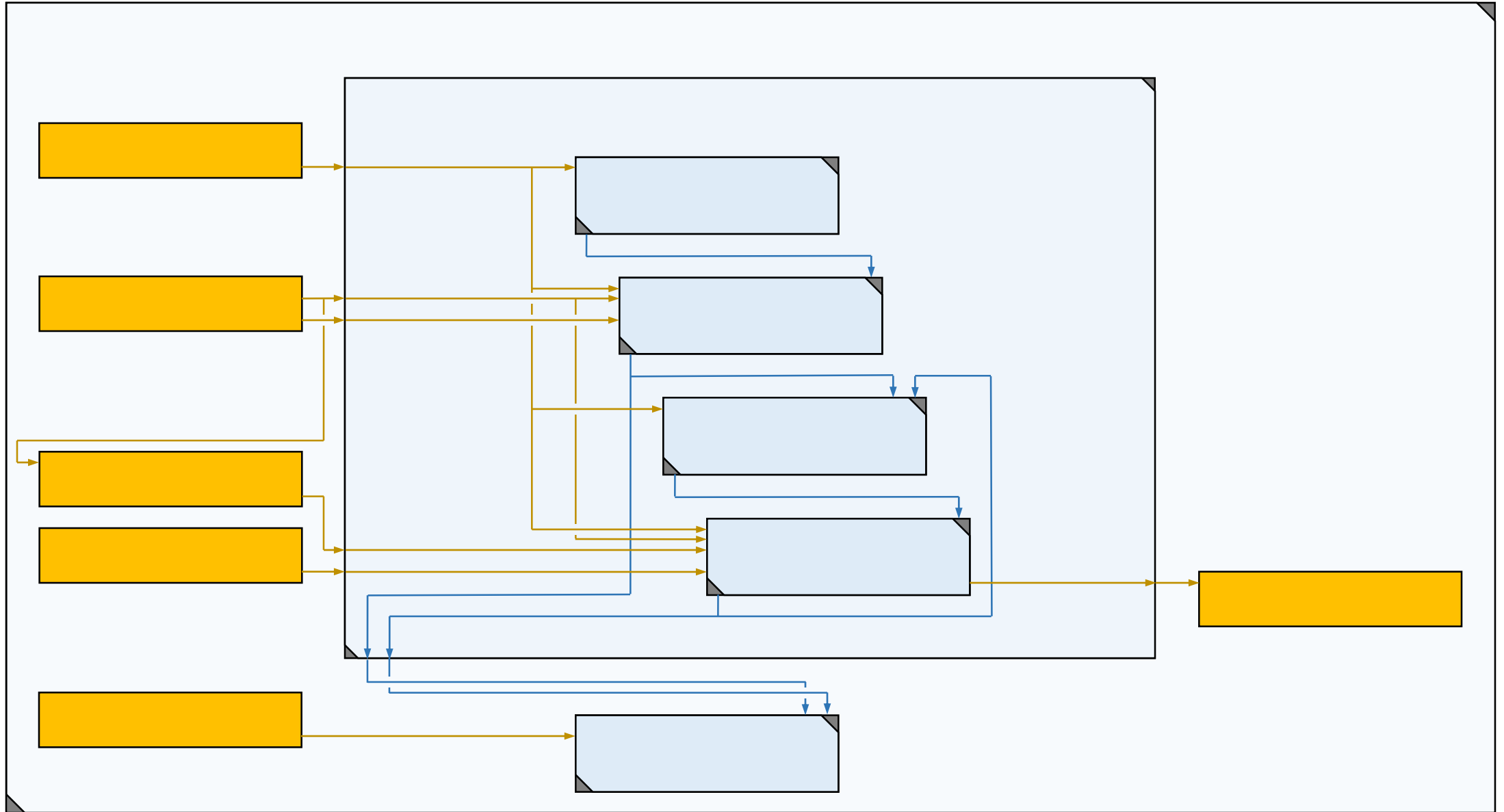
Review – Function Nodes



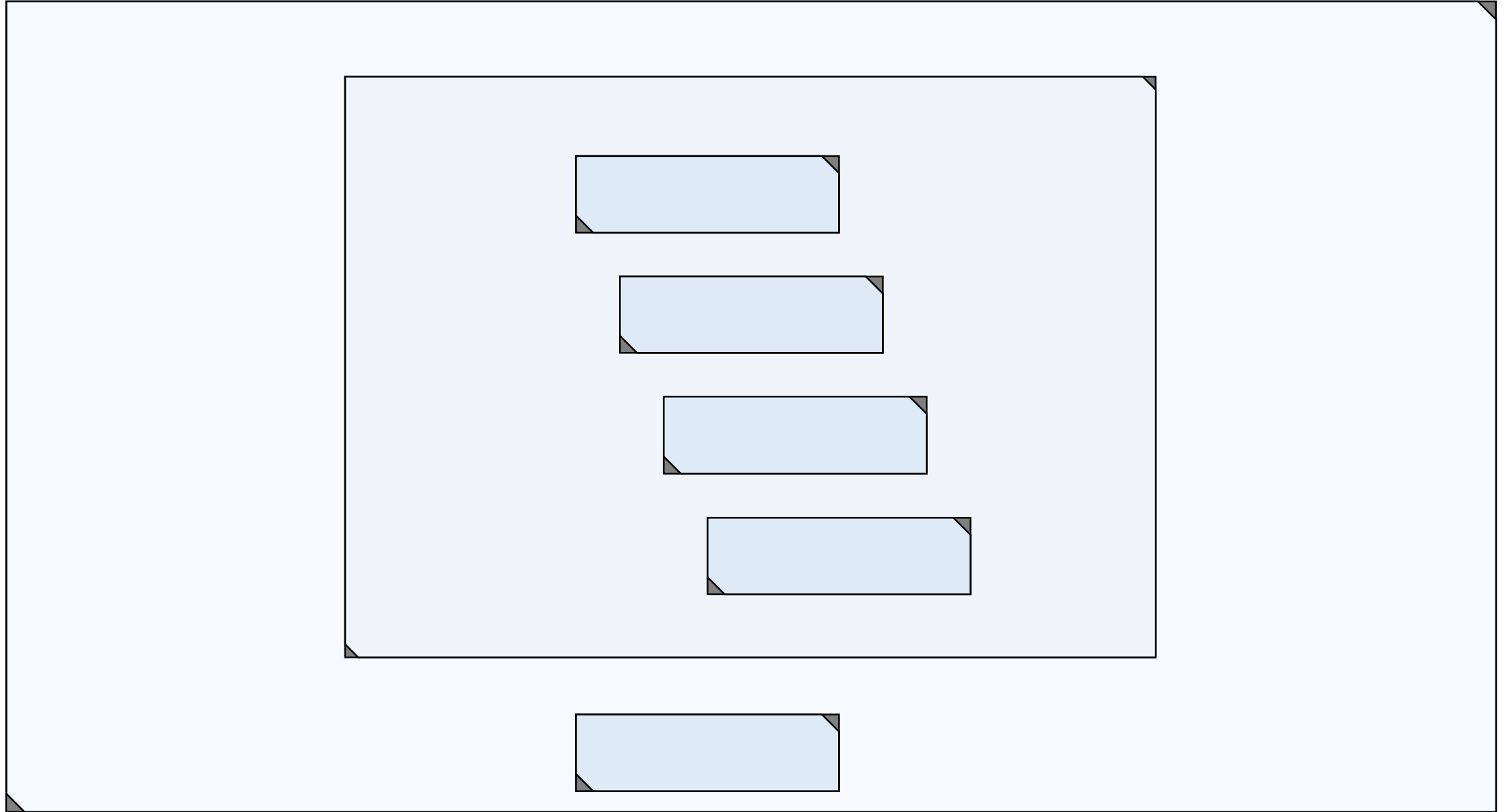
Review – Function Nodes



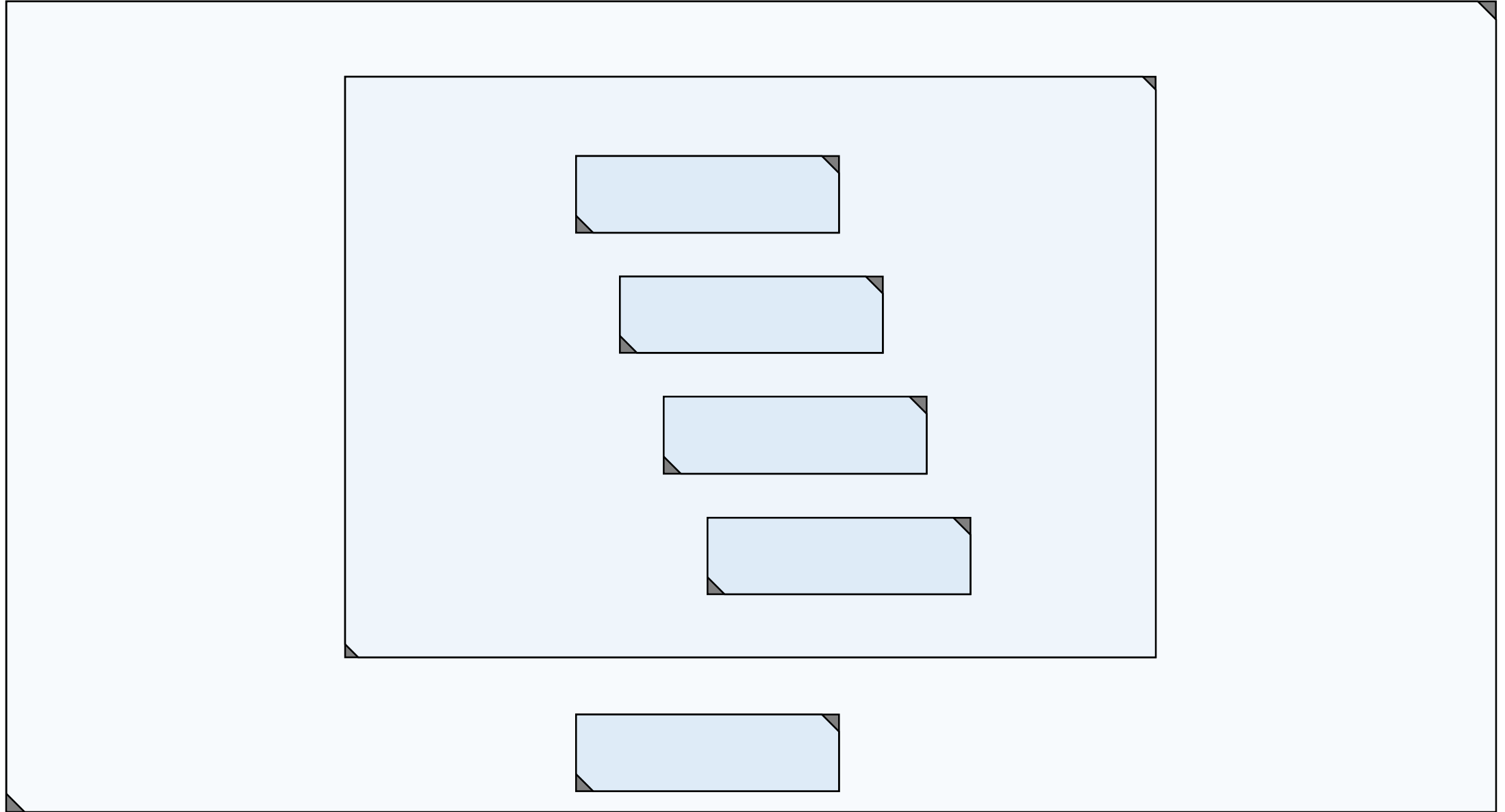
Review



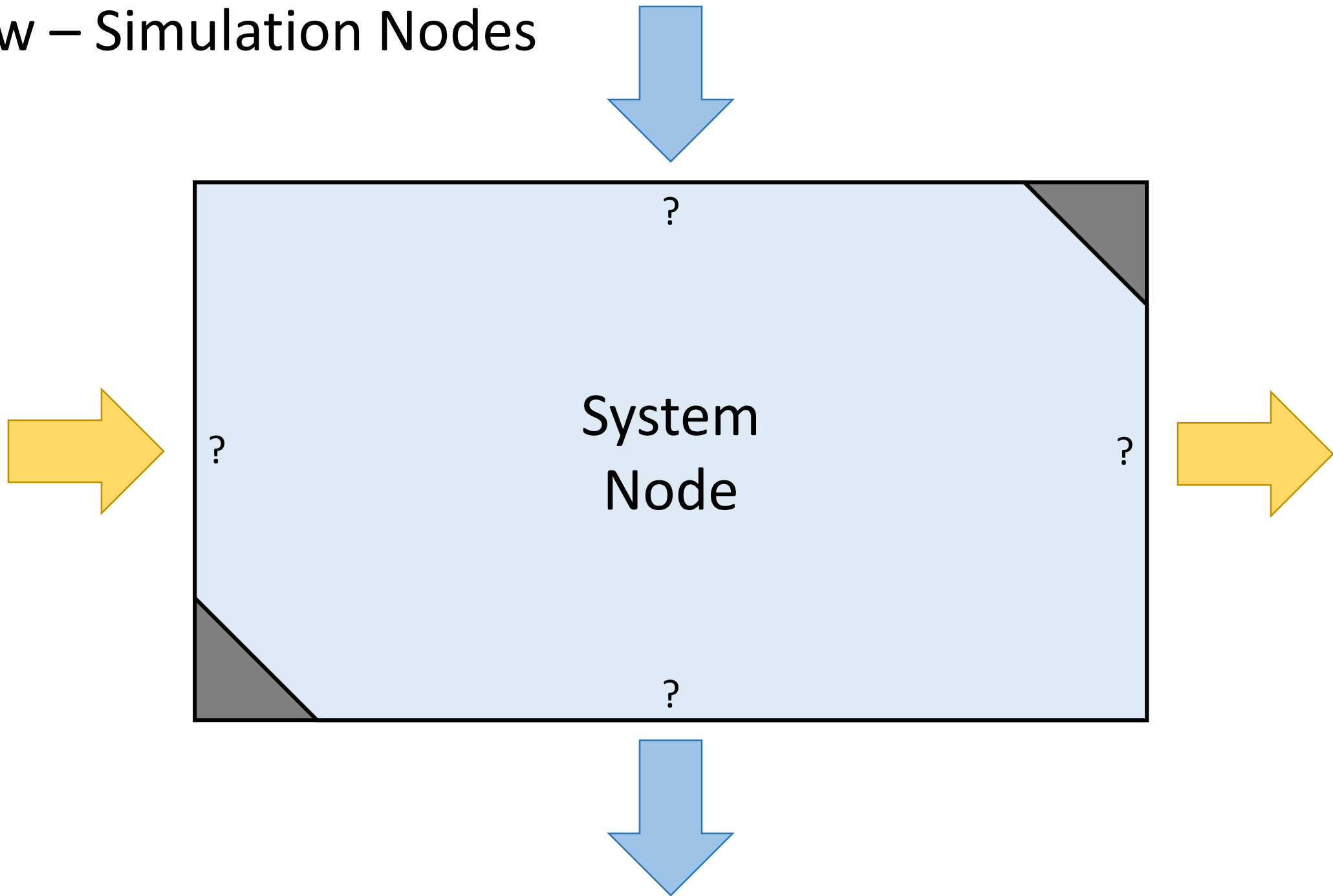
Review – ?



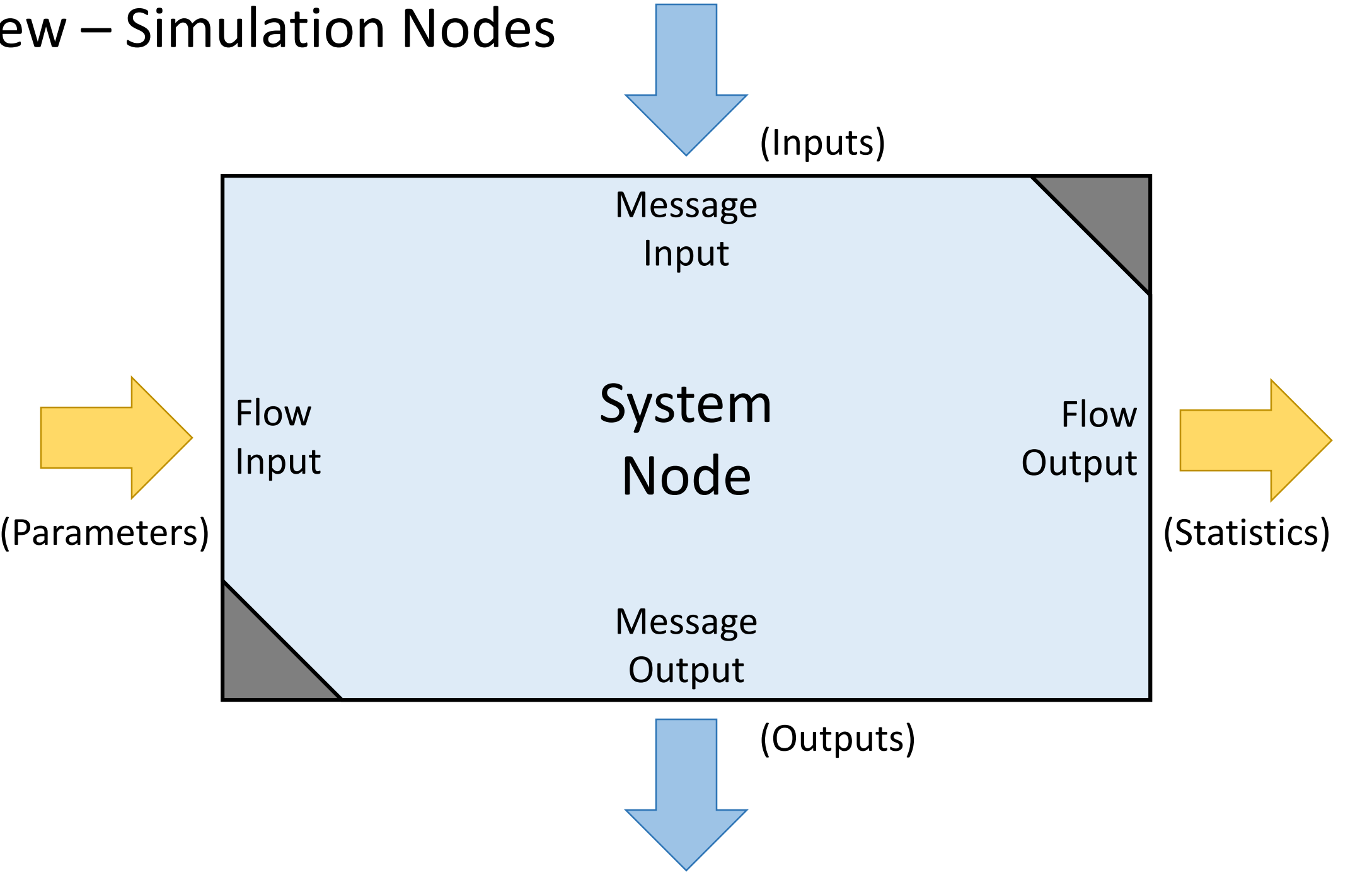
Review – Simulation Nodes



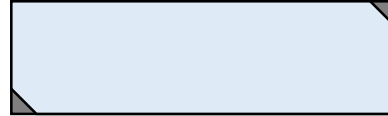
Review – Simulation Nodes



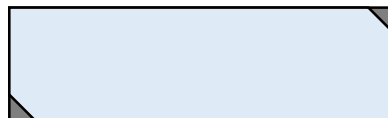
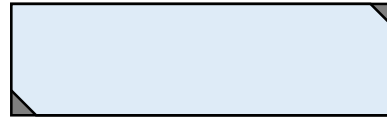
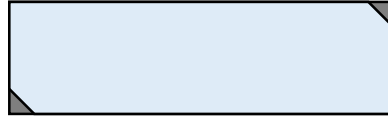
Review – Simulation Nodes



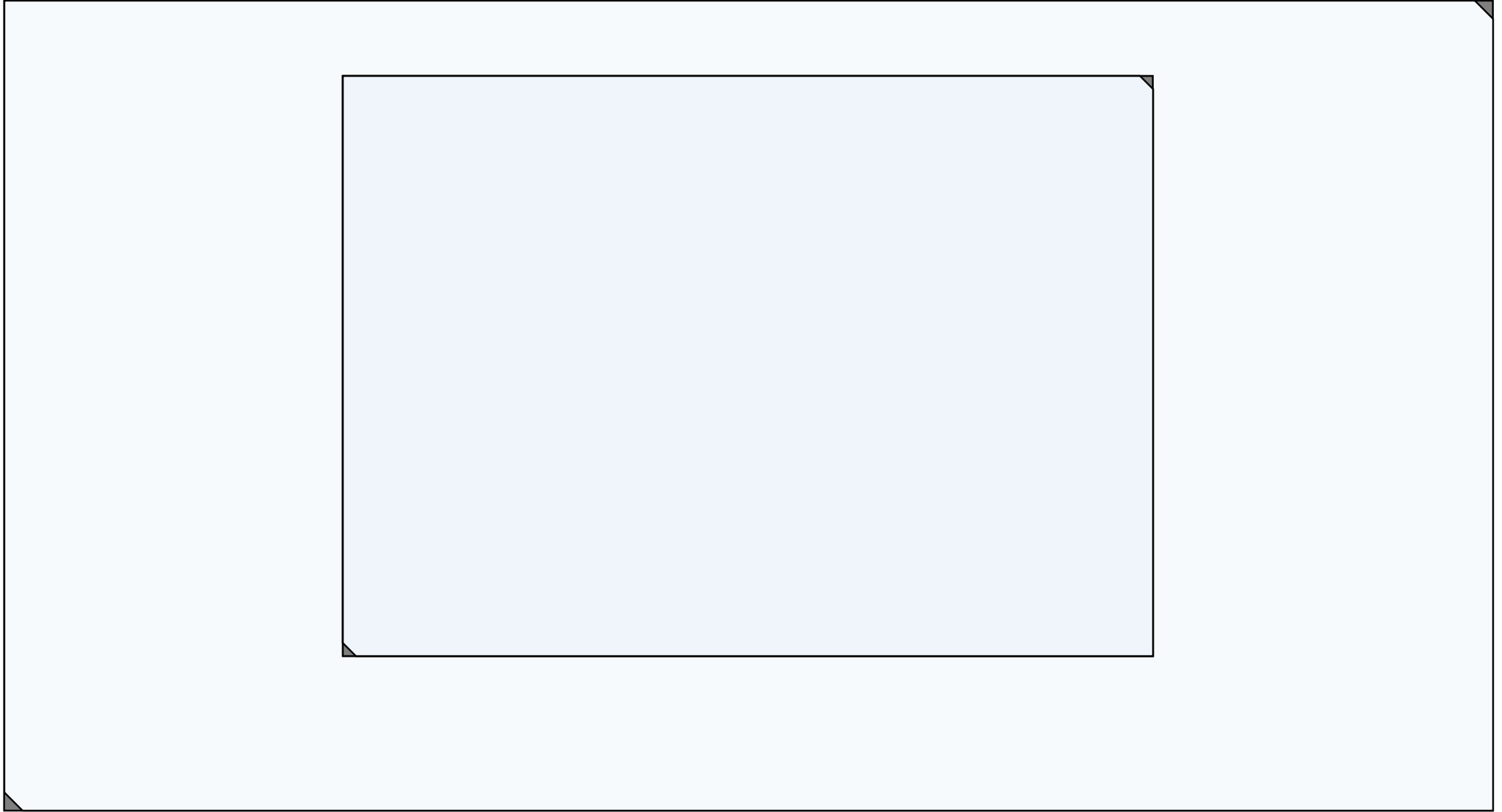
Review – ?



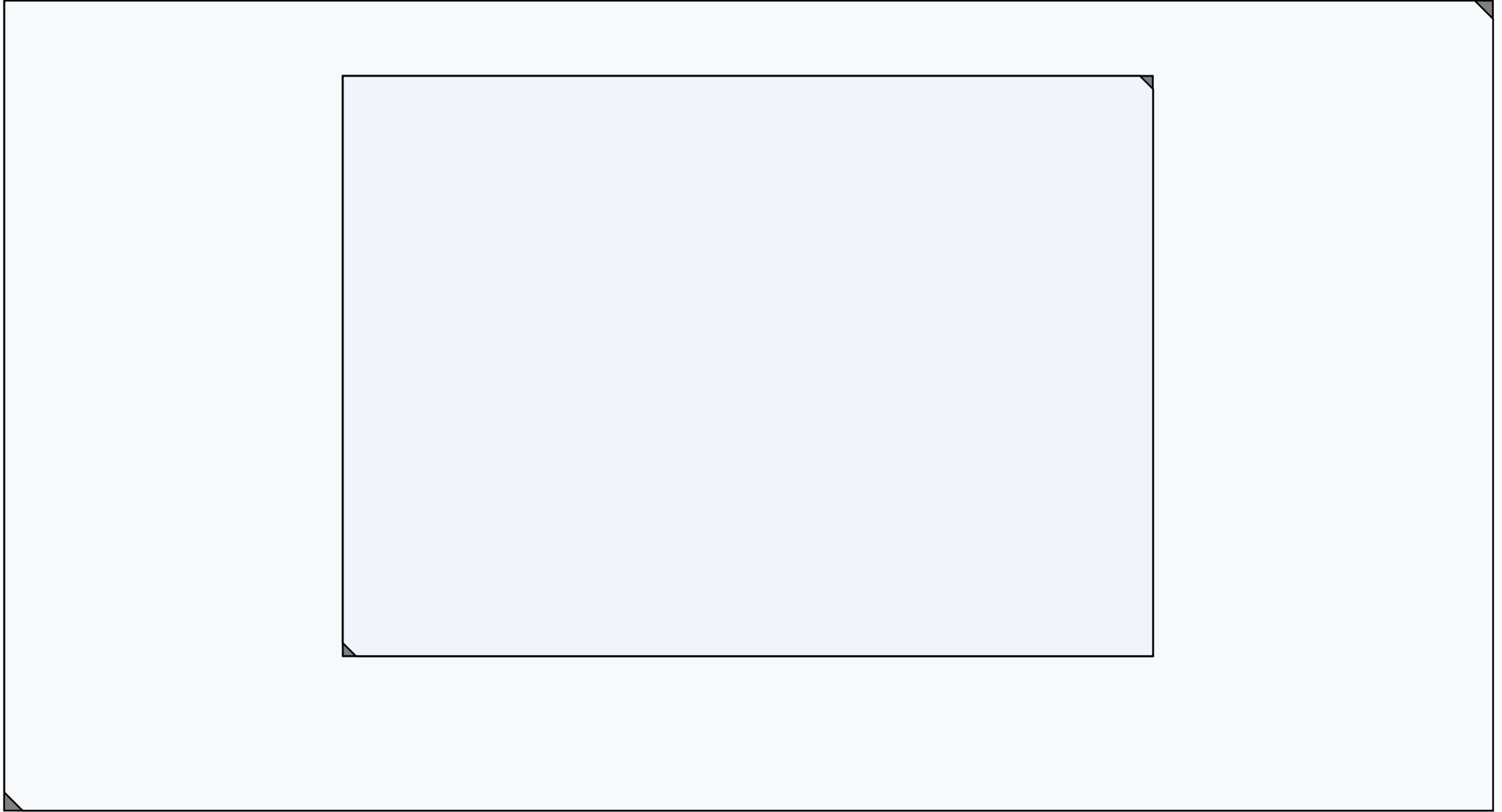
Review – Atomic Nodes



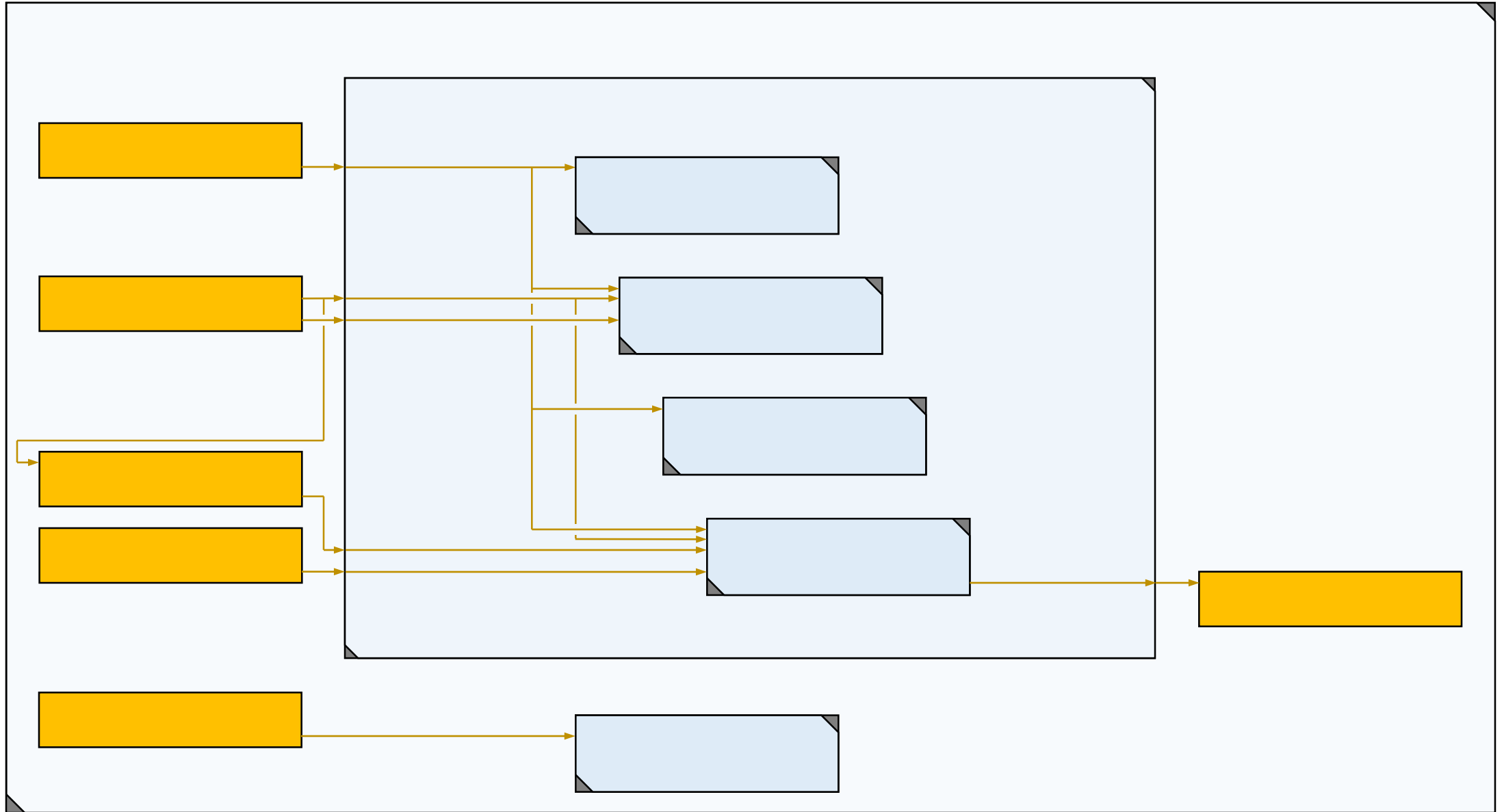
Review – ?



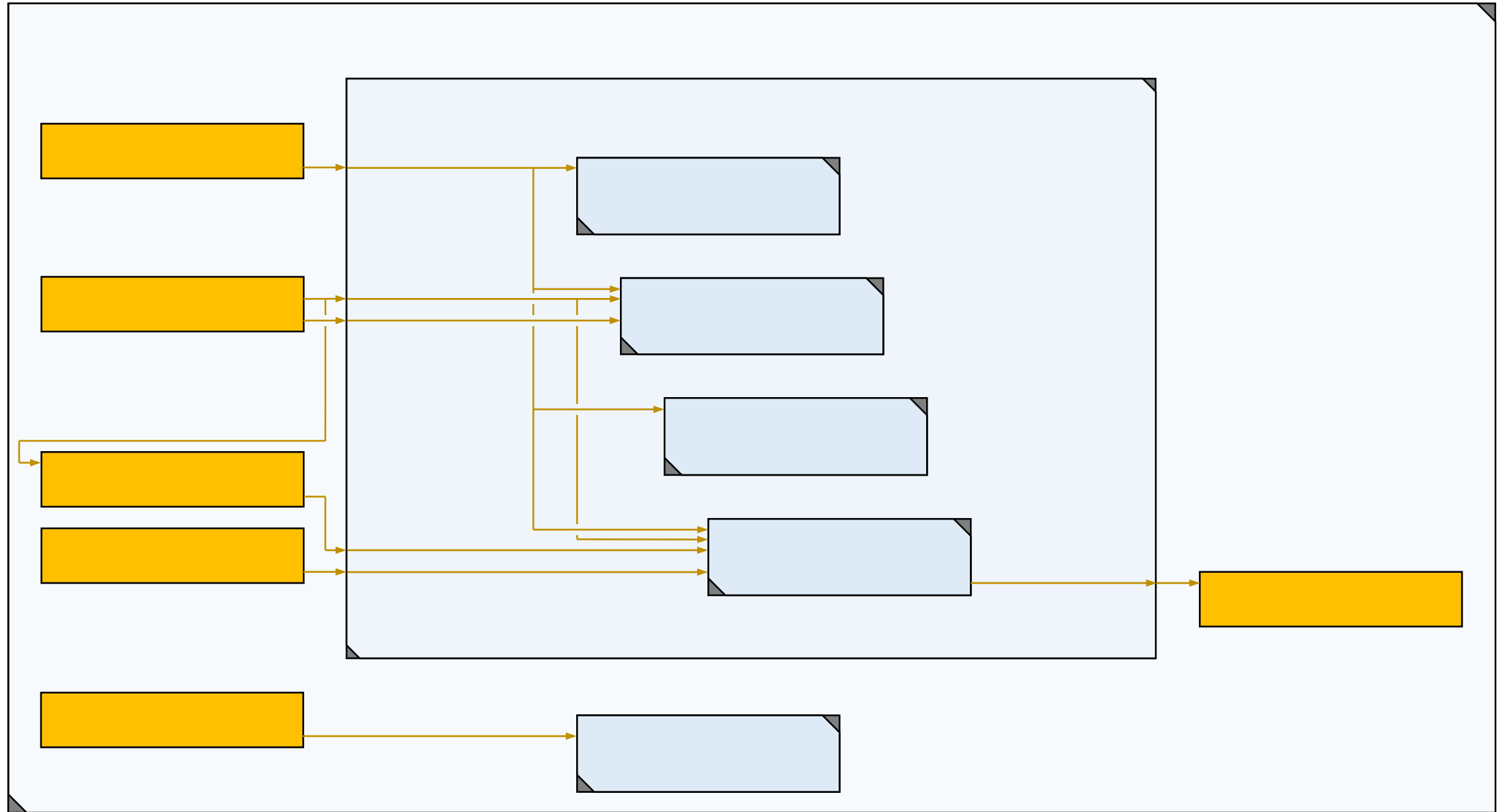
Review – Composite Nodes



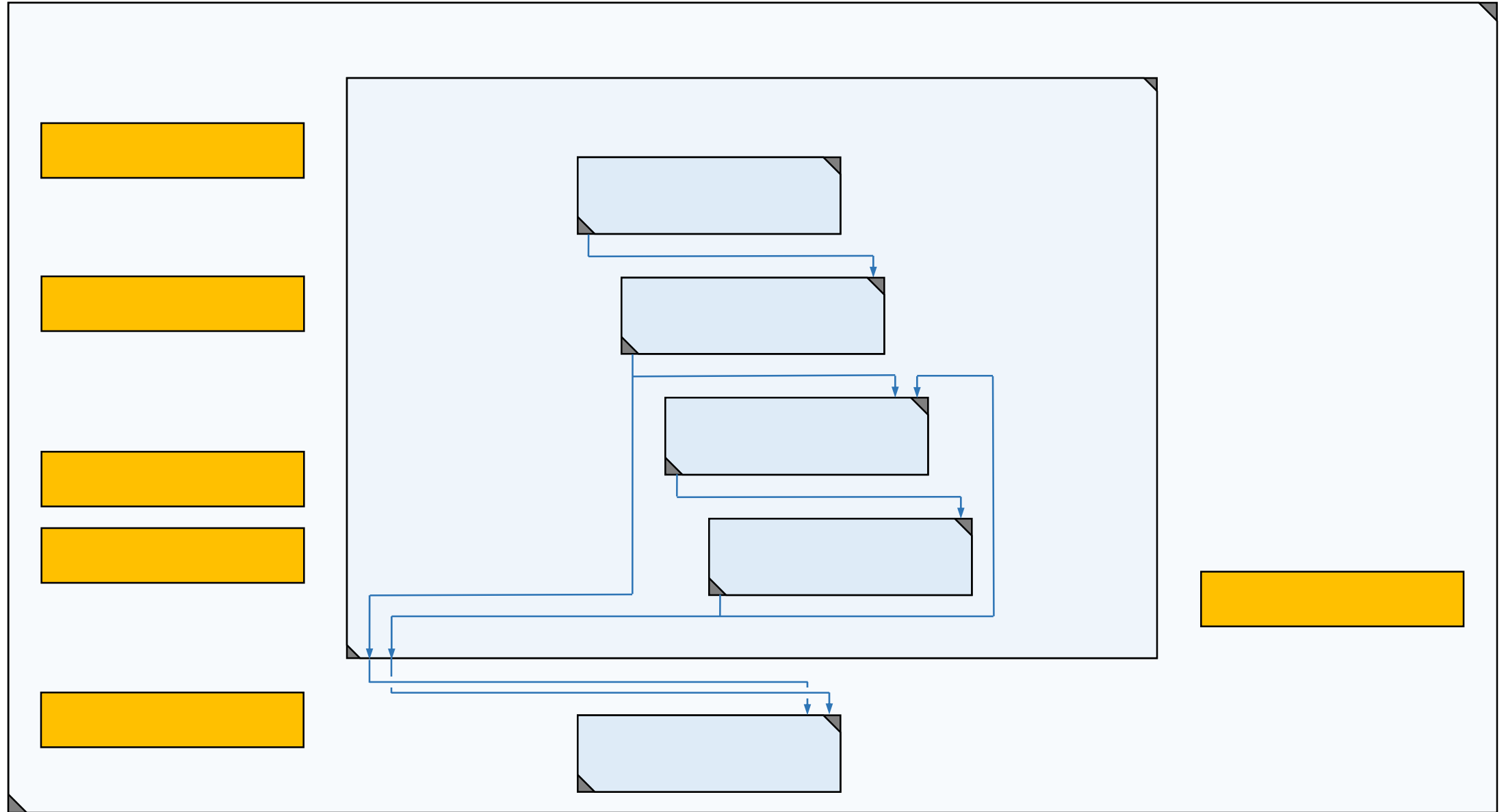
Review – ?



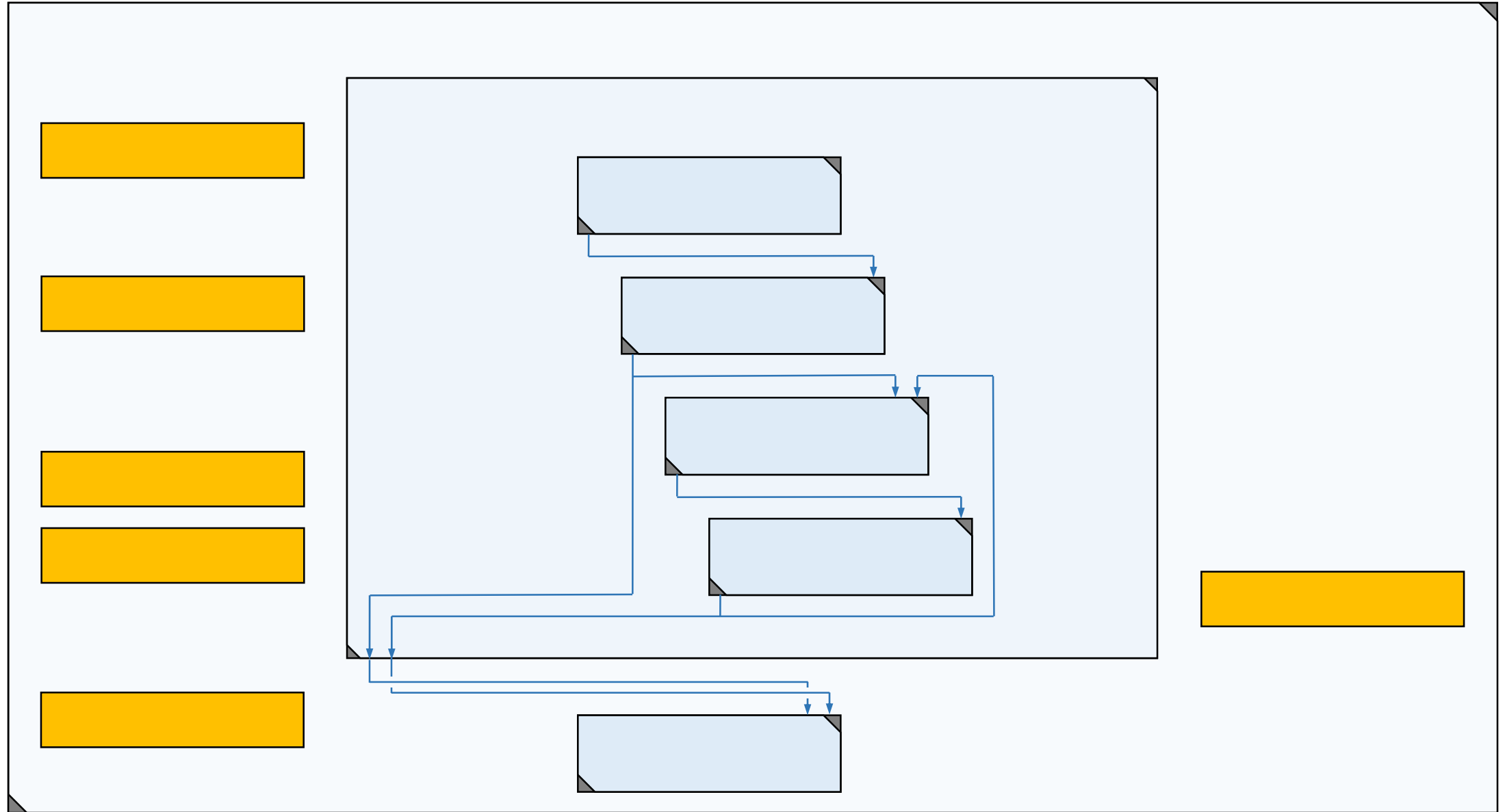
Review – Flow Links



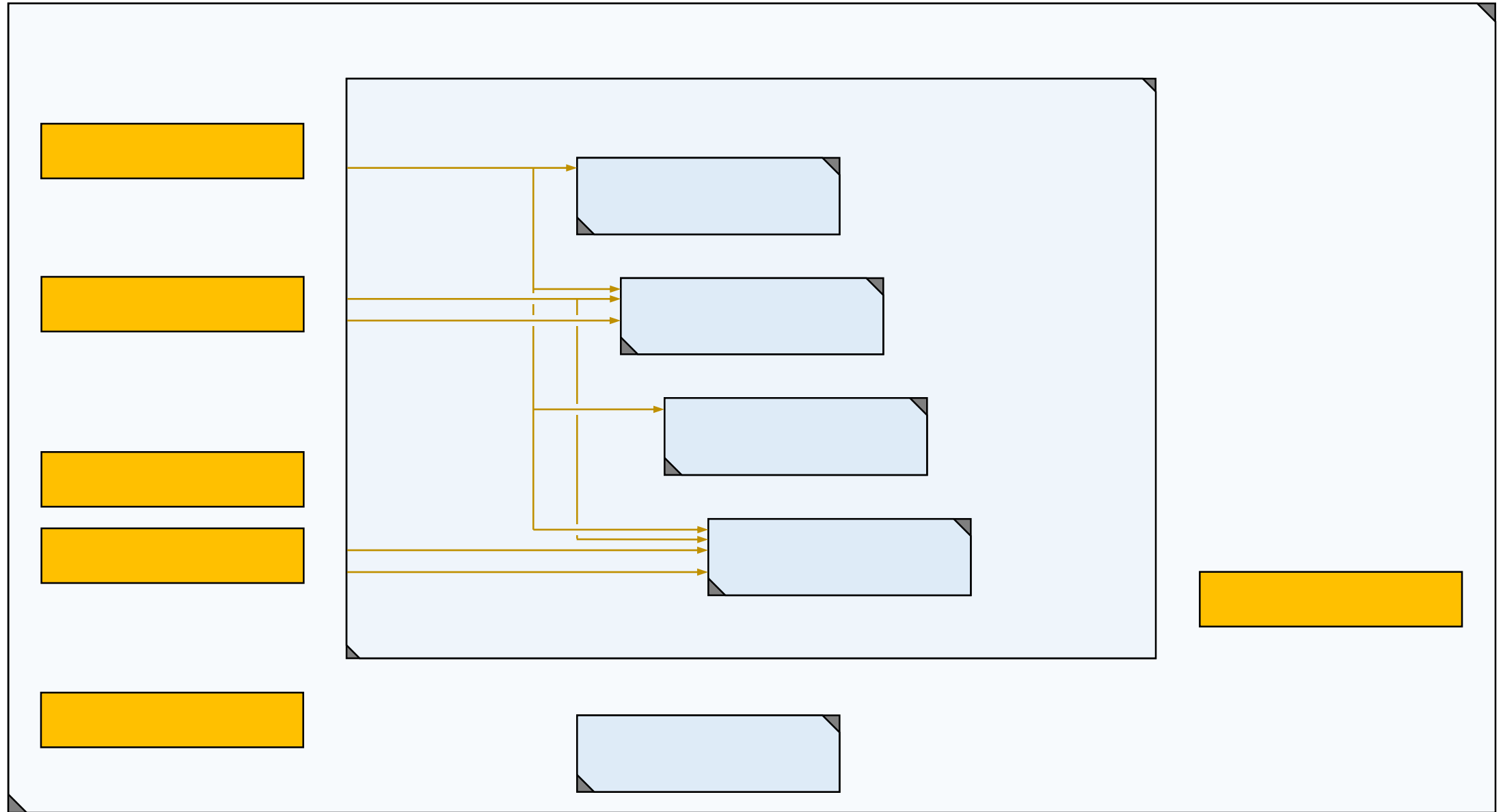
Review – ?



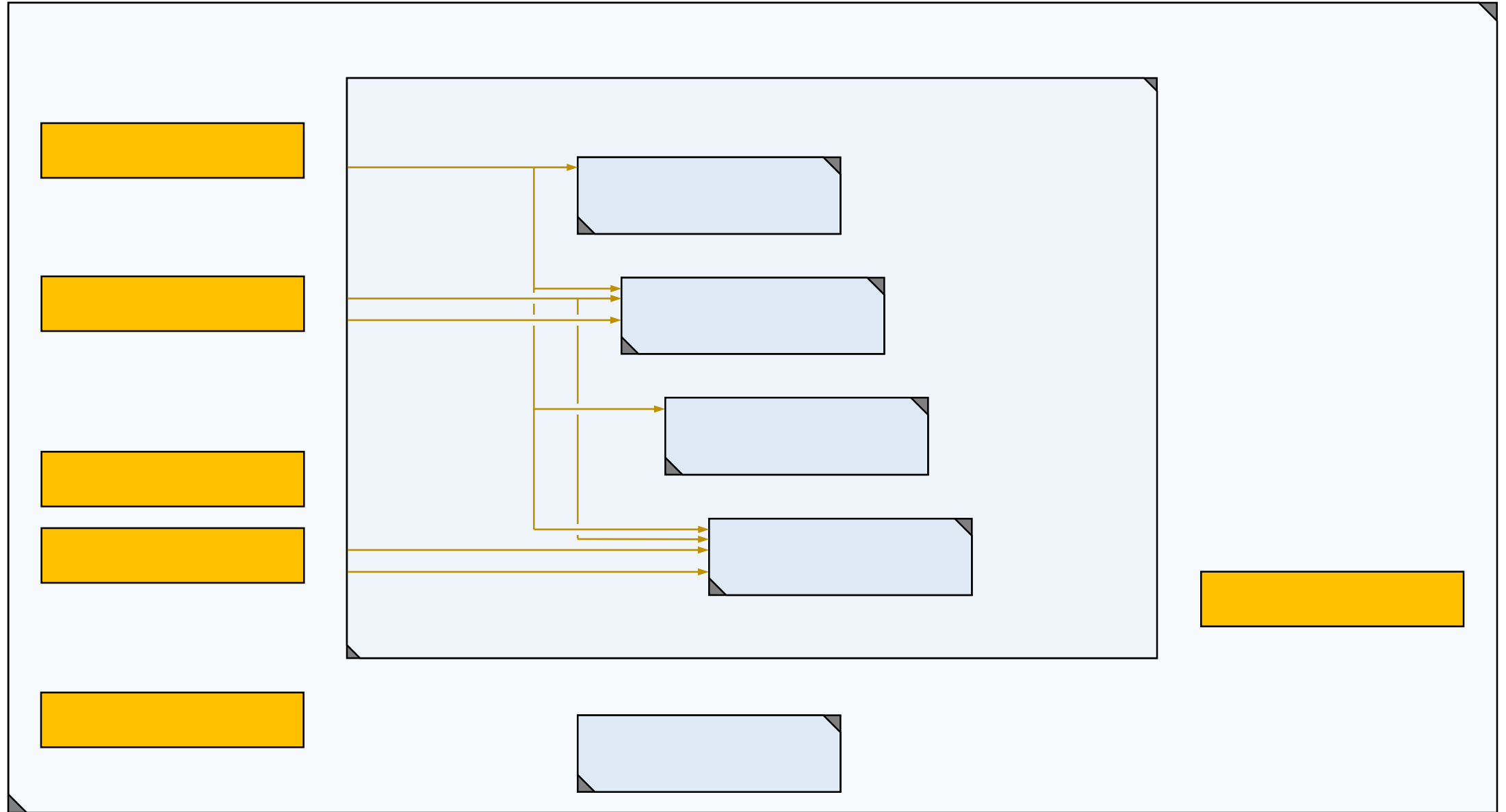
Review – Message Links



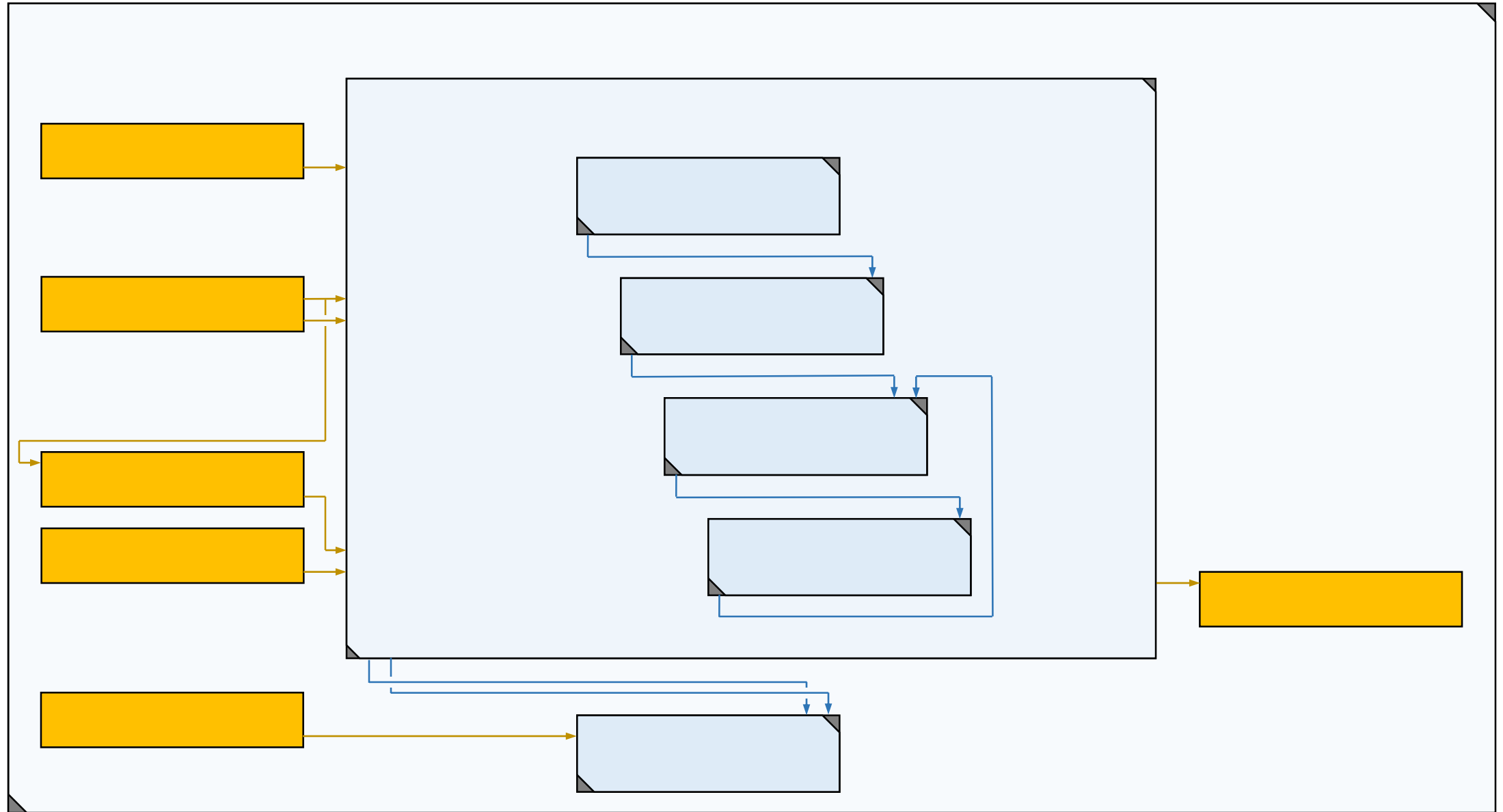
Review – ?



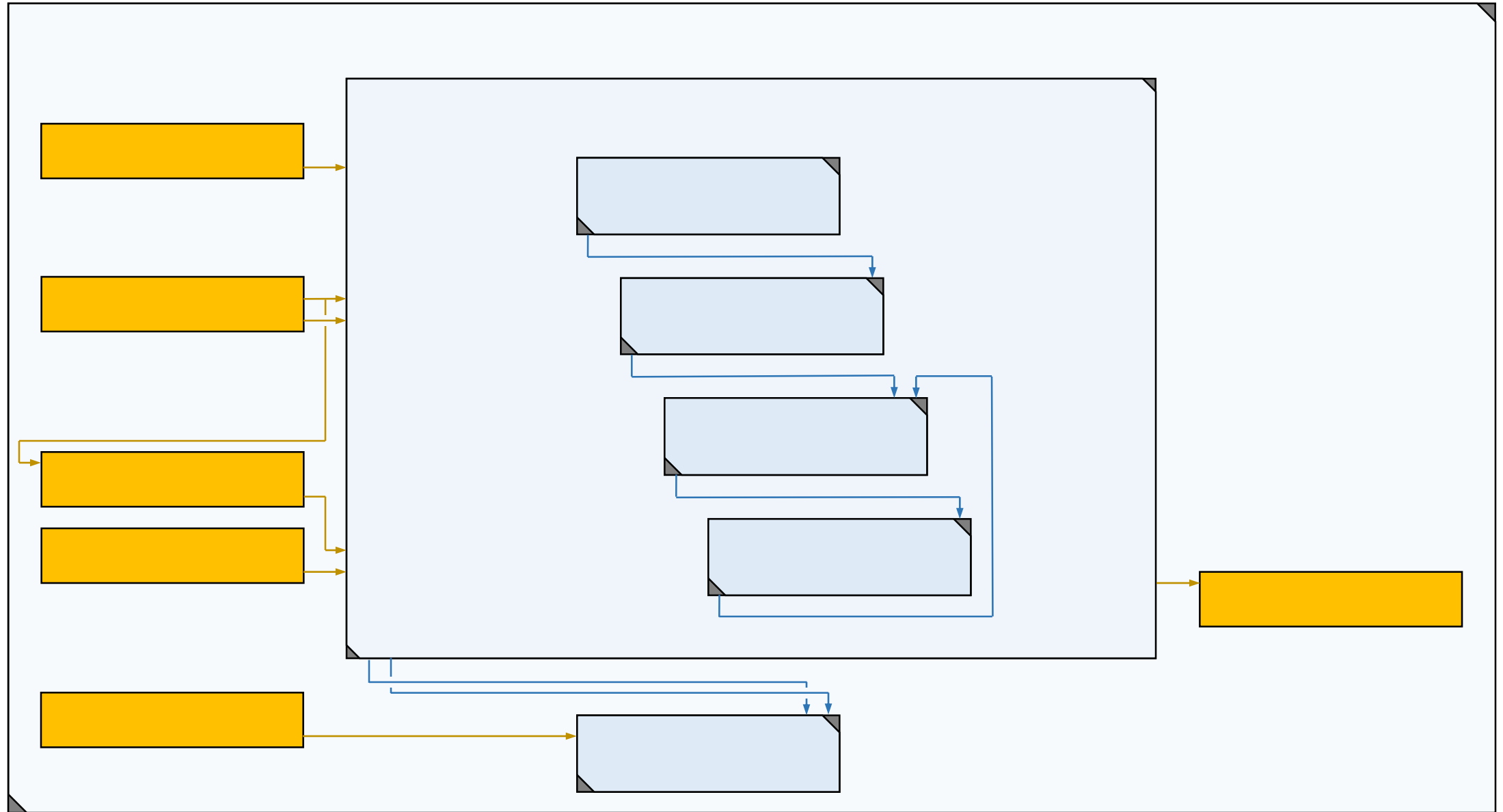
Review – Inward Links



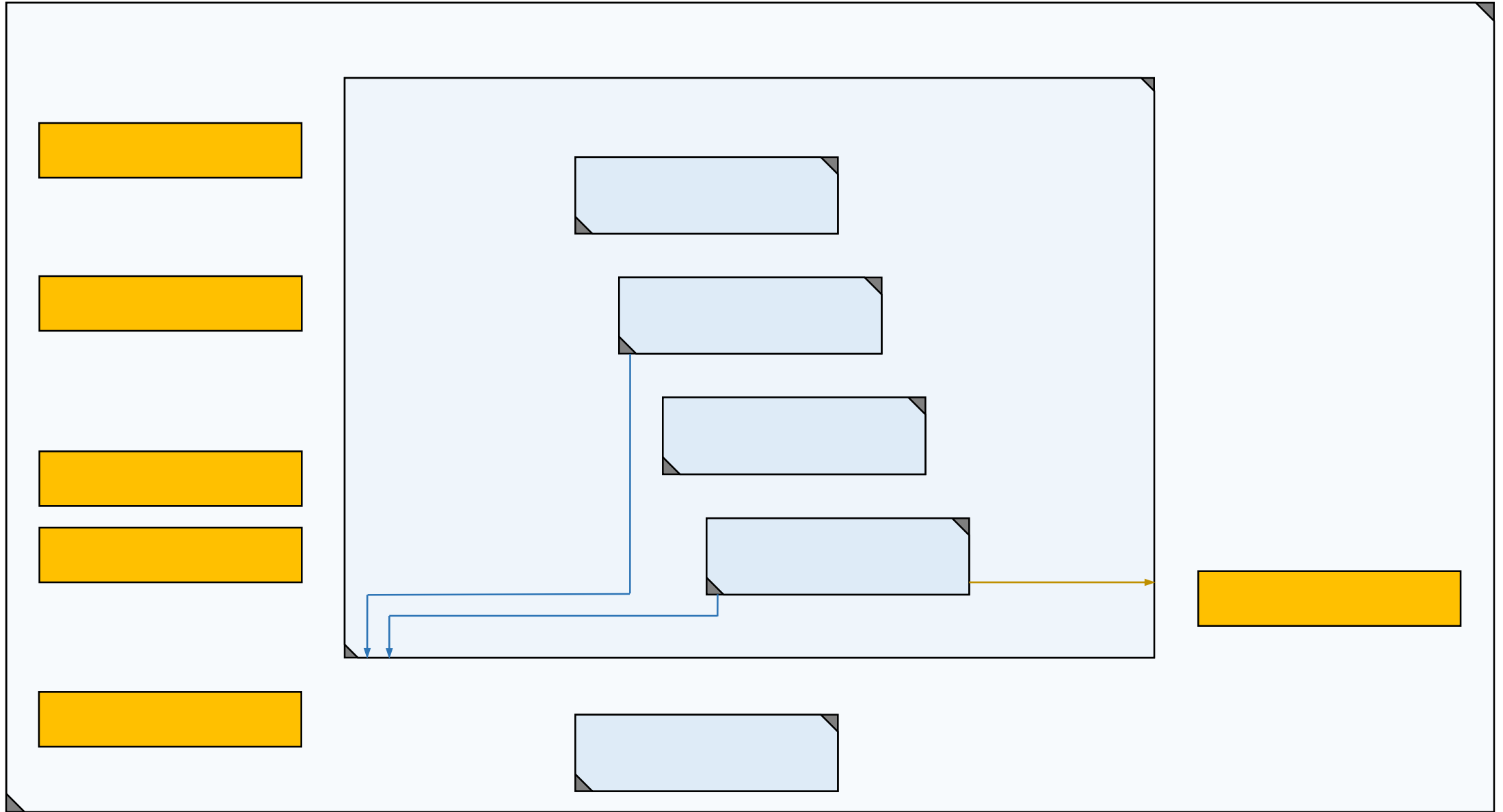
Review – ?



Review – Inner Links



Review – ?



Review – Outward Links

