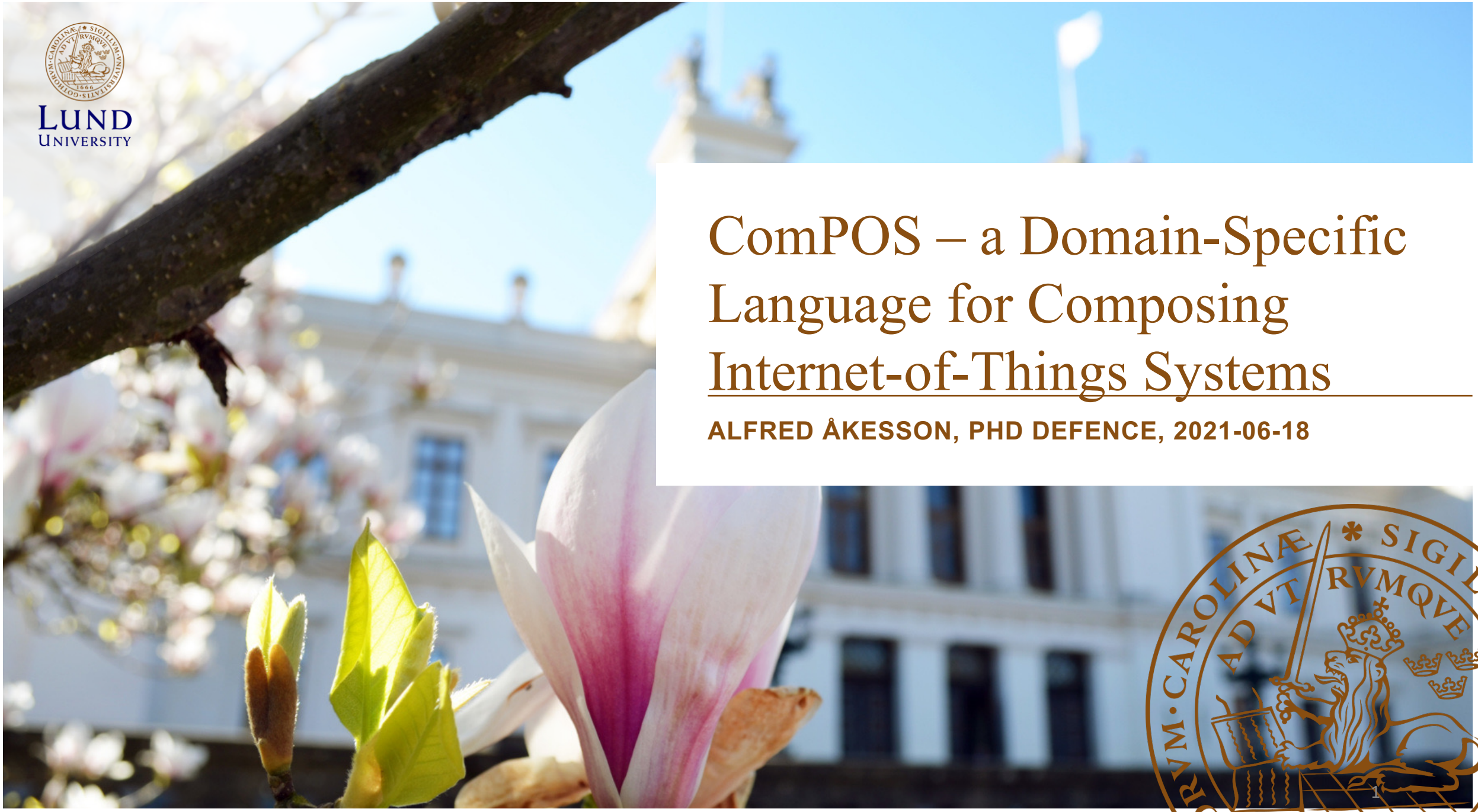


ComPOS – a Domain-Specific Language for Composing Internet-of-Things Systems

ALFRED ÅKESSON, PHD DEFENCE, 2021-06-18

Internet-of-Things (IoT) systems

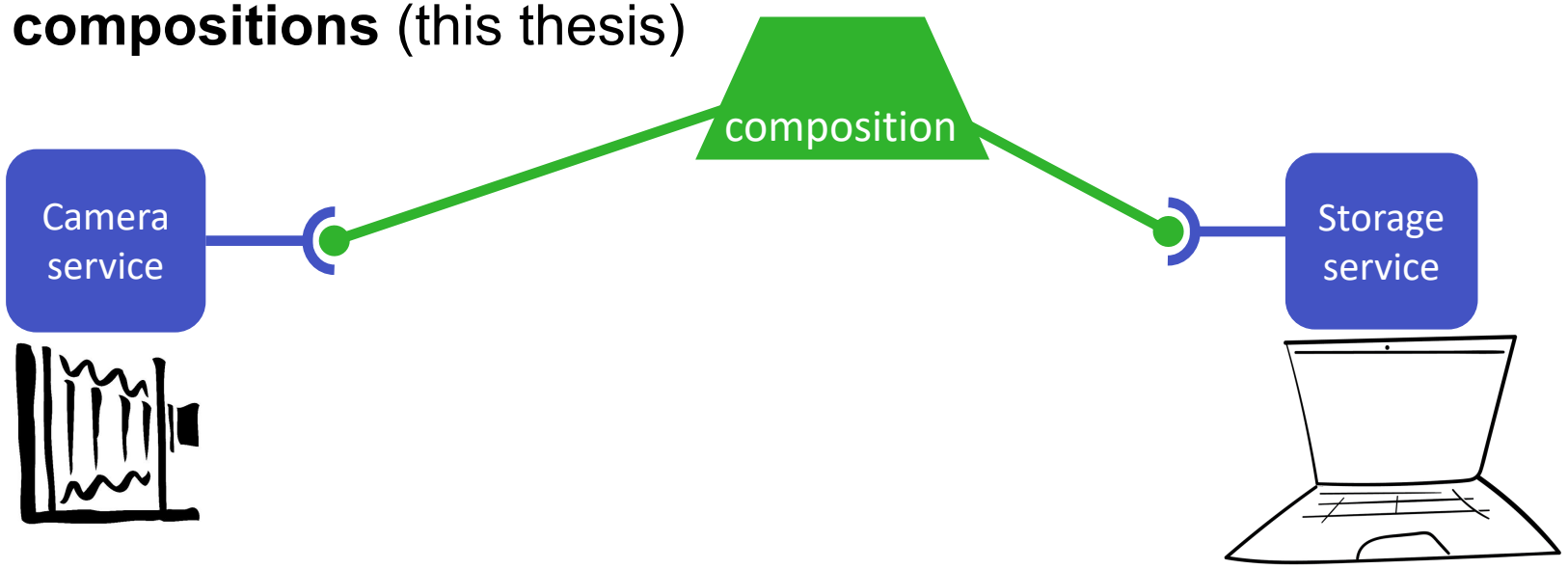
- **IoT**: trend of connected devices (e.g. lamp, dishwasher)
- **IoT system**: system of connected devices
- Two challenges
  - Weak connectivity
  - Always running

How can we simplify development of IoT systems?

- Programming experience
  - **Paper 1:** Live Programming of Internet of Things in PalCom
    - *Combining discovery and composition in connecting IoT system*
  - **Paper 4:** Jatte: A Tunable Tree Editor for Integrated DSLs
    - *Meta-tool for creating custom projectional editors, e.g., IoT editor*
- Programming model
  - **Paper 2:** ComPOS: Composing Systems of Services
    - *Domain specific language (DSL) for connecting devices*
- System understanding
  - **Paper 3:** Runtime modeling and analysis of IoT systems
    - *Runtime model for analysis of IoT systems*
    - *Device Dependency Analysis (DDA)*

PalCom Middleware

- Service-based middleware with message passing
  - A device has a set of services
- Device/service discovery
- **Service compositions** (this thesis)



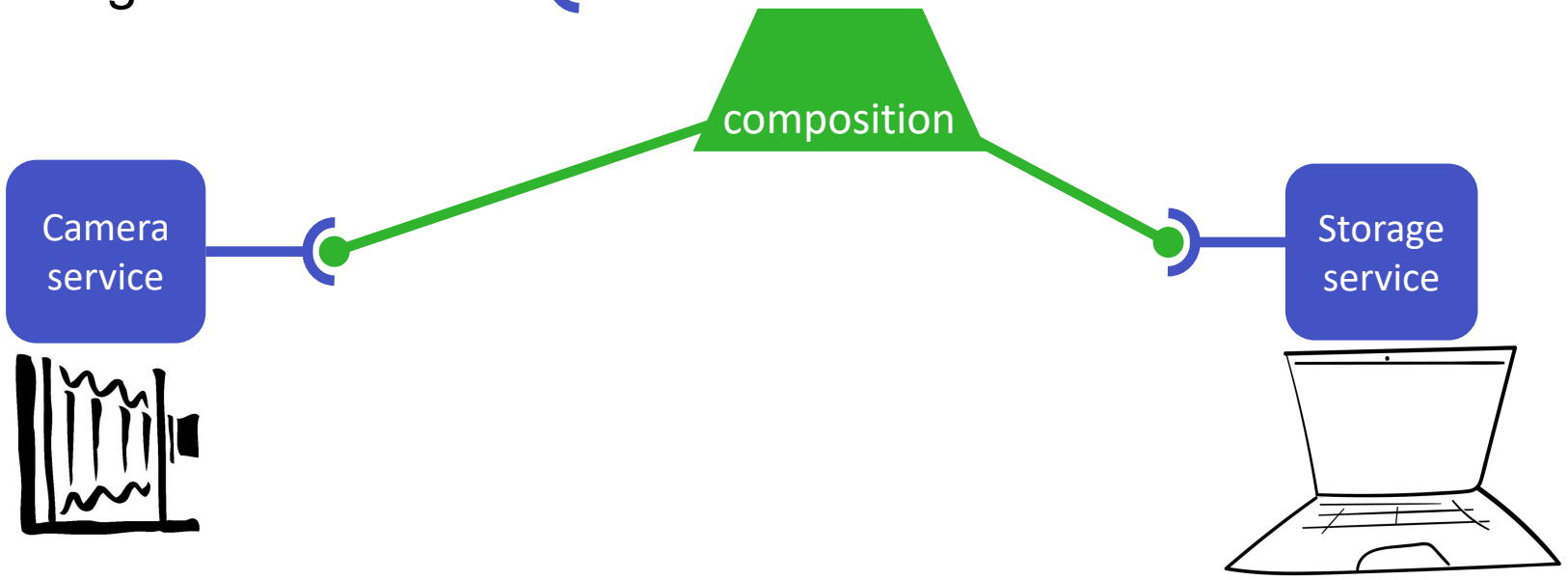
Composing services

- Services

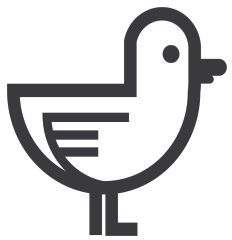
  - Provide functionality
  - Do not know who they talk to
  - Accessible through an interface

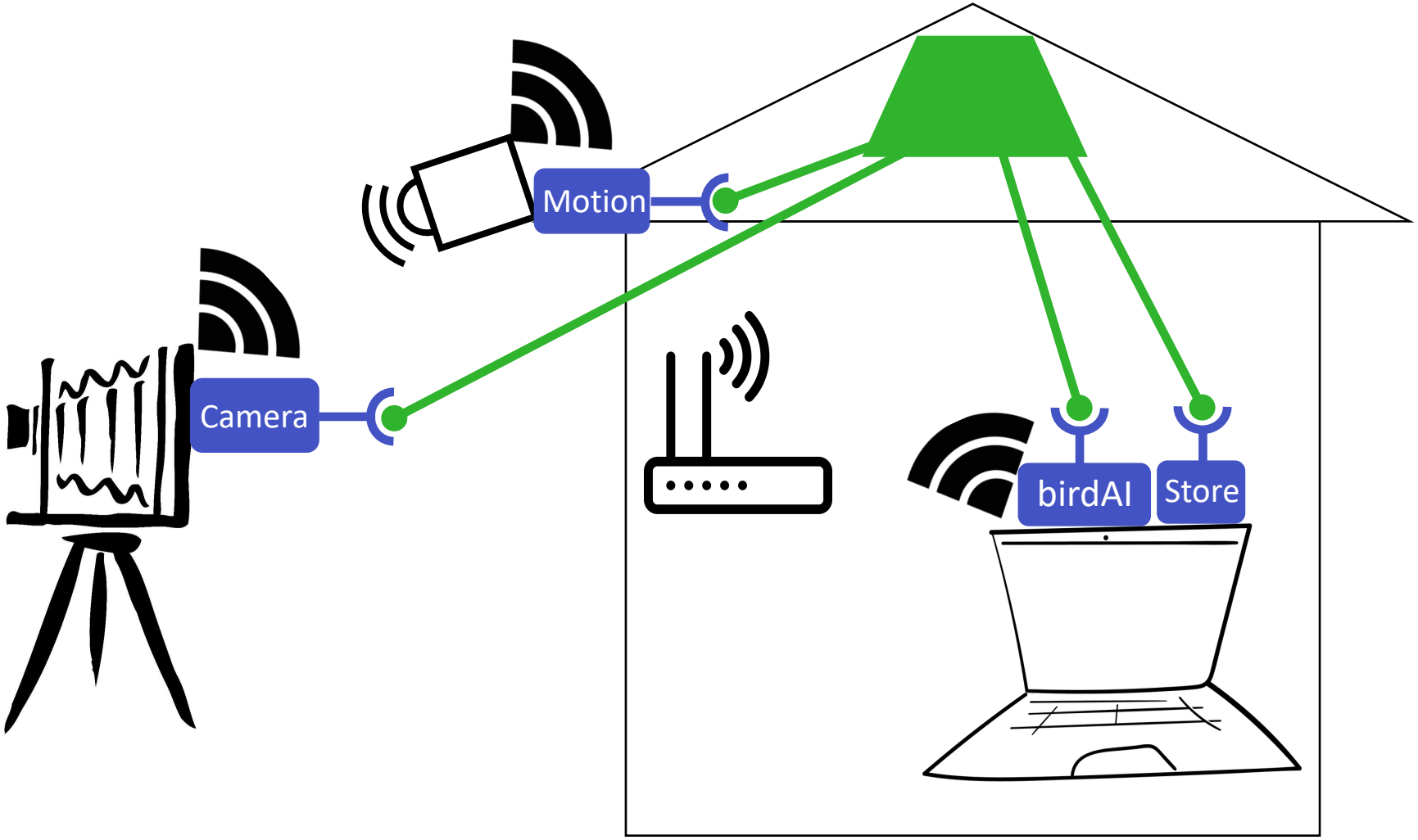
- Compositions

  - Used to compose services
  - Specified in Domain Specific Language

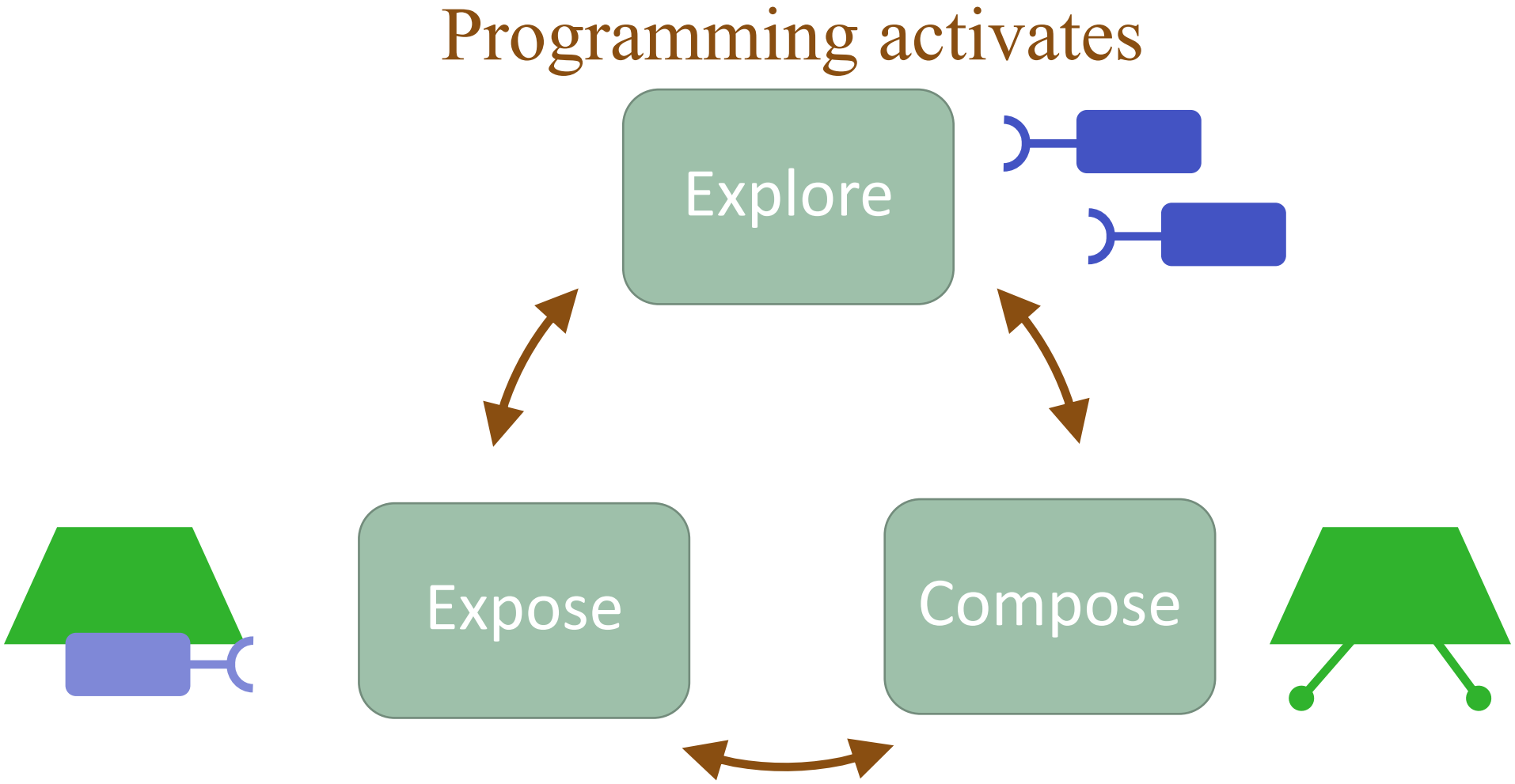


Scenario



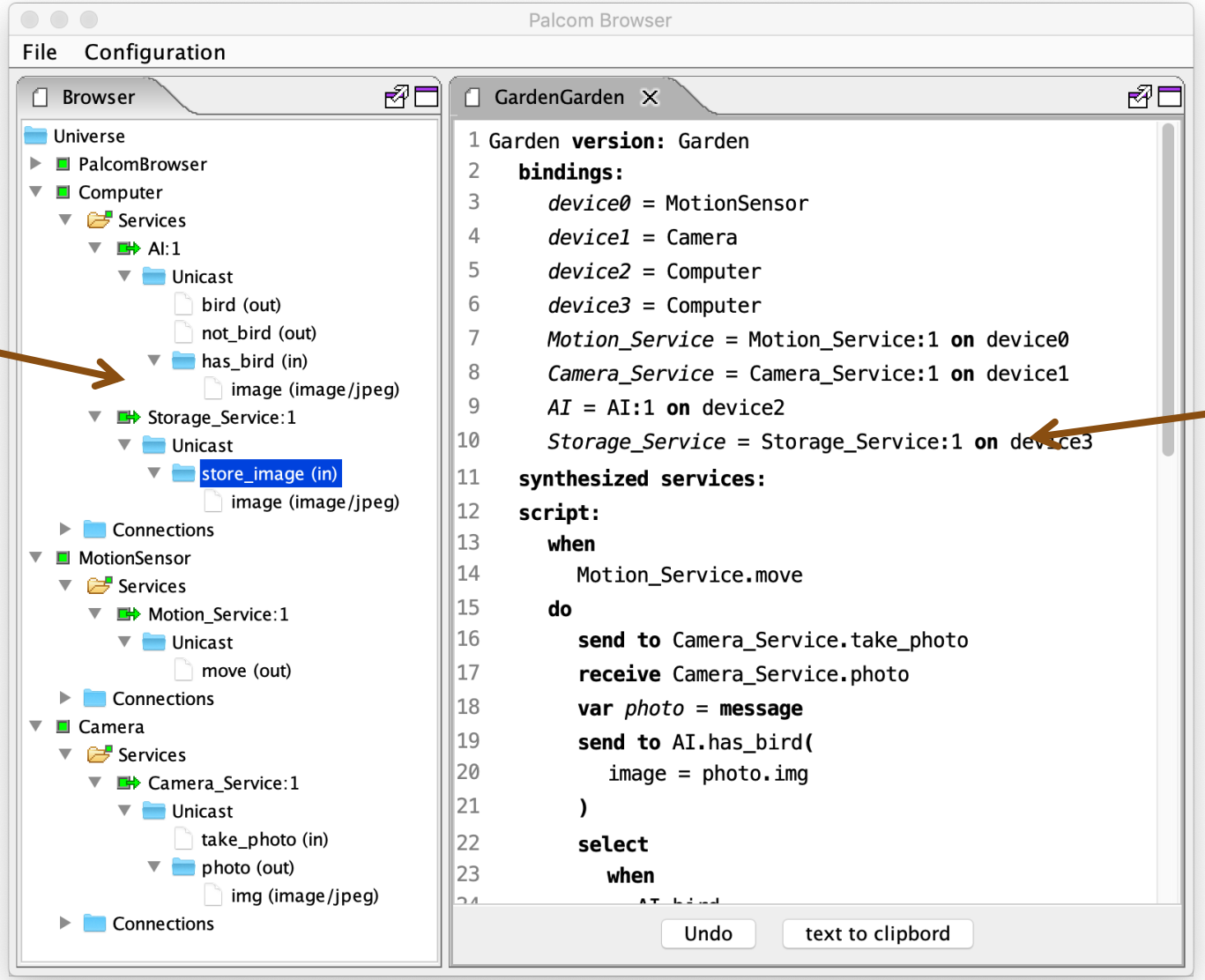


Papers 1: Live Programming of Internet of Things in PalCom

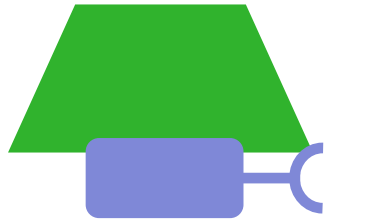


PalCom Browser – Development environment

Device and
Service discovery



Composition
editor built using
Jatte



Paper 2: ComPOS: Composing Systems of Services

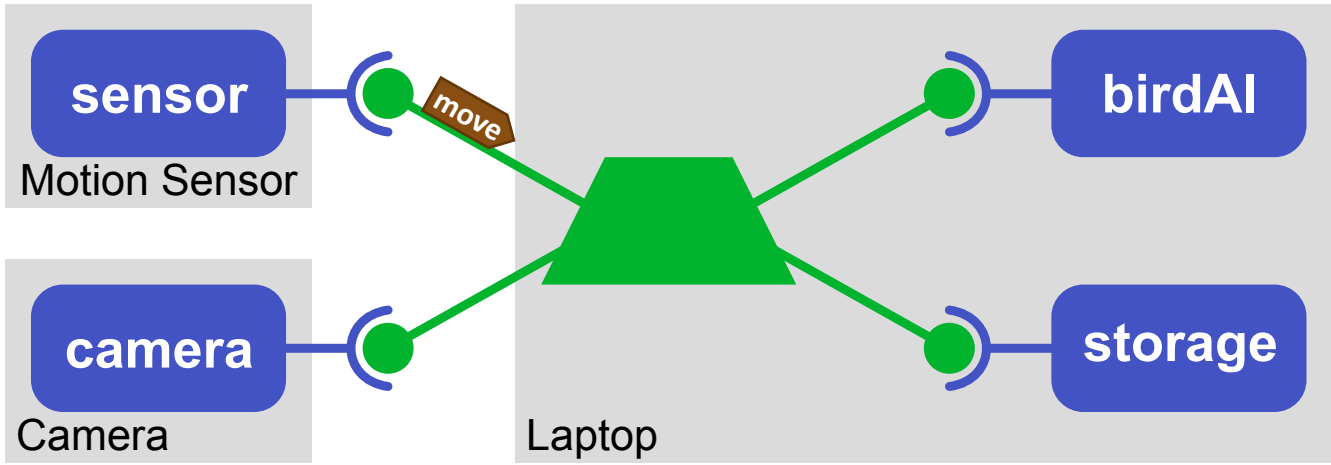
- ComPOS a new DSL for composing services
- Strategies for handling new messages
- Case study

ComPOS DSL

- Coordinates incoming and outgoing messages
- No computations
- Control flow constructs
  - Sequence
  - Select
  - Parallel
  - Finish first

ComPOS example

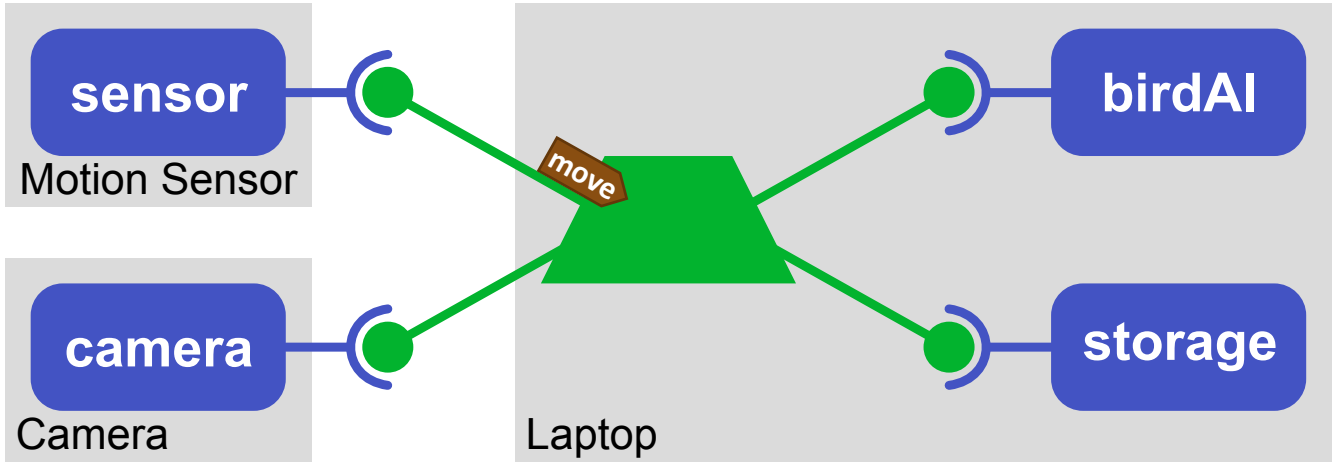
```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



ComPOS example

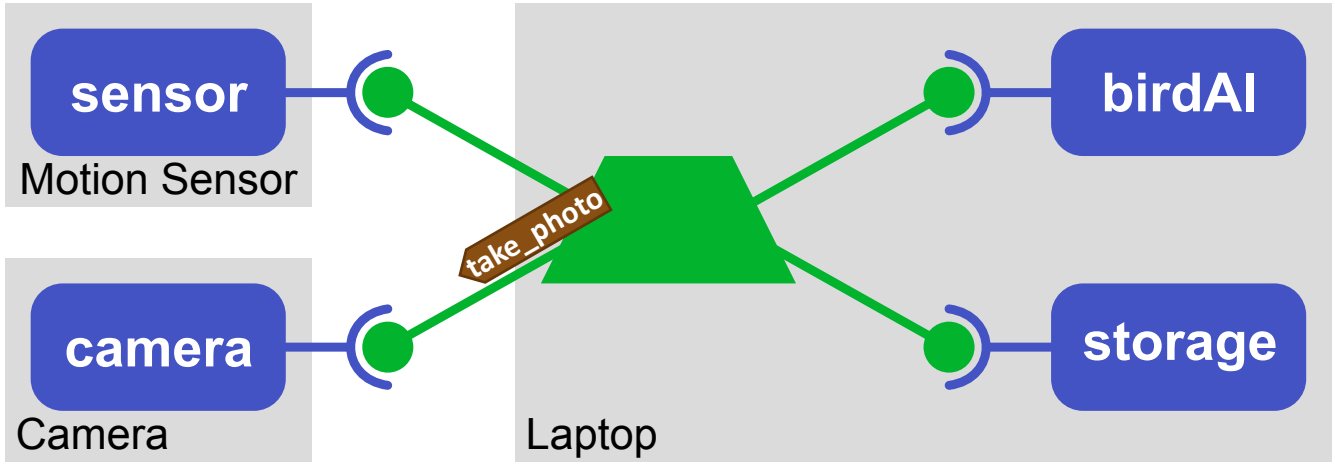
```
composition SimpleBirdwatcher
  service se
  service ca
  servic  bi
  s  vice storage = ...
when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```

Starts new reaction



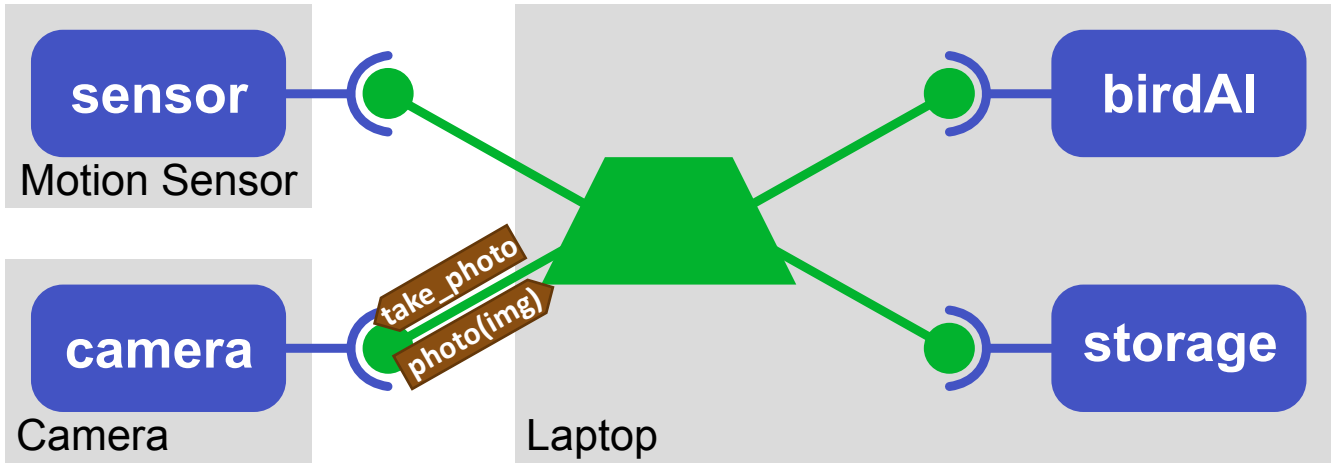
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
  ➡ send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



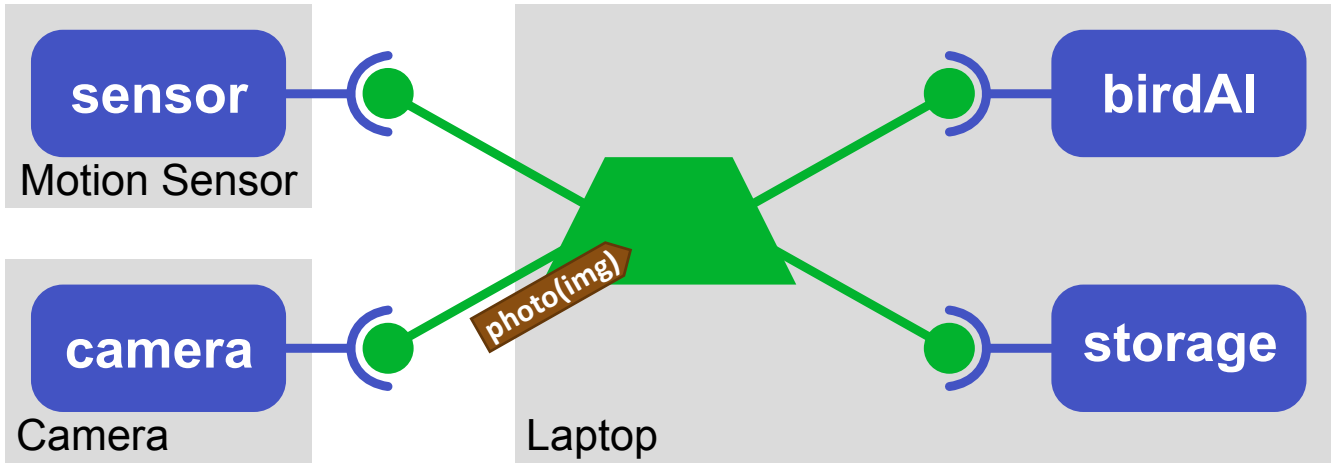
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



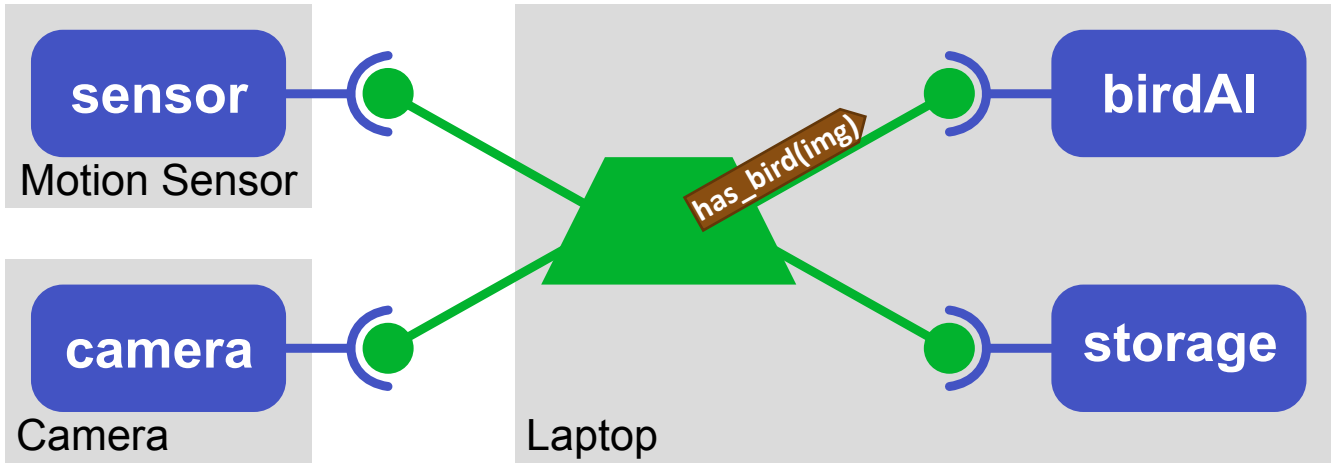
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



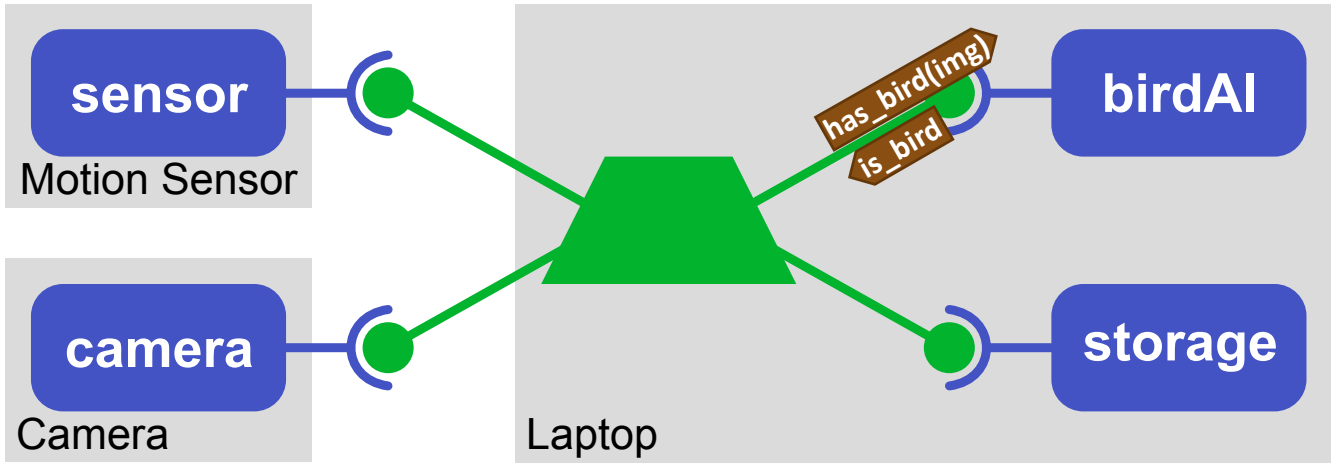
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



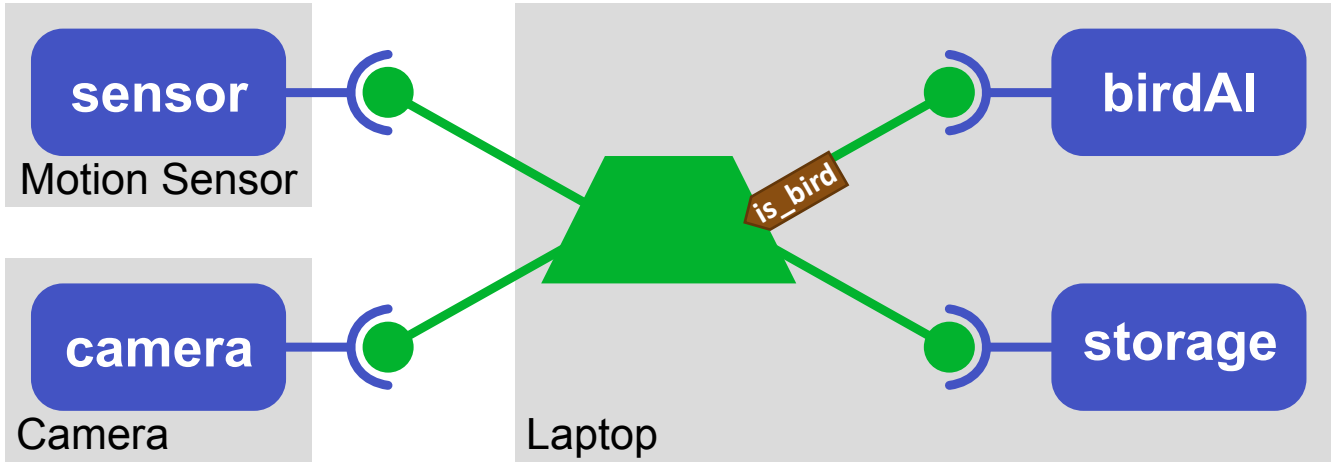
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
  ⇨select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



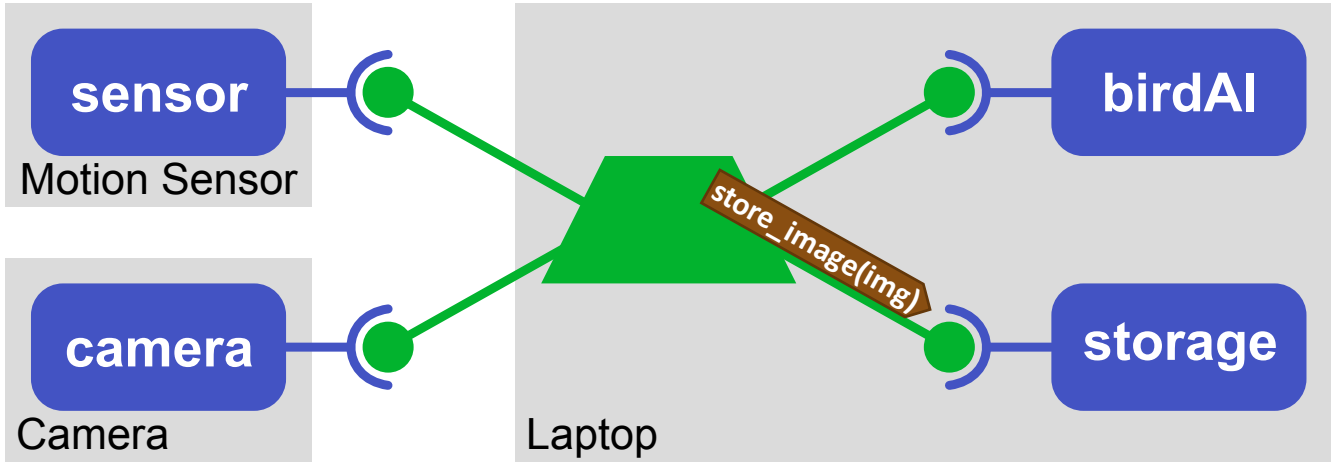
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
    when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



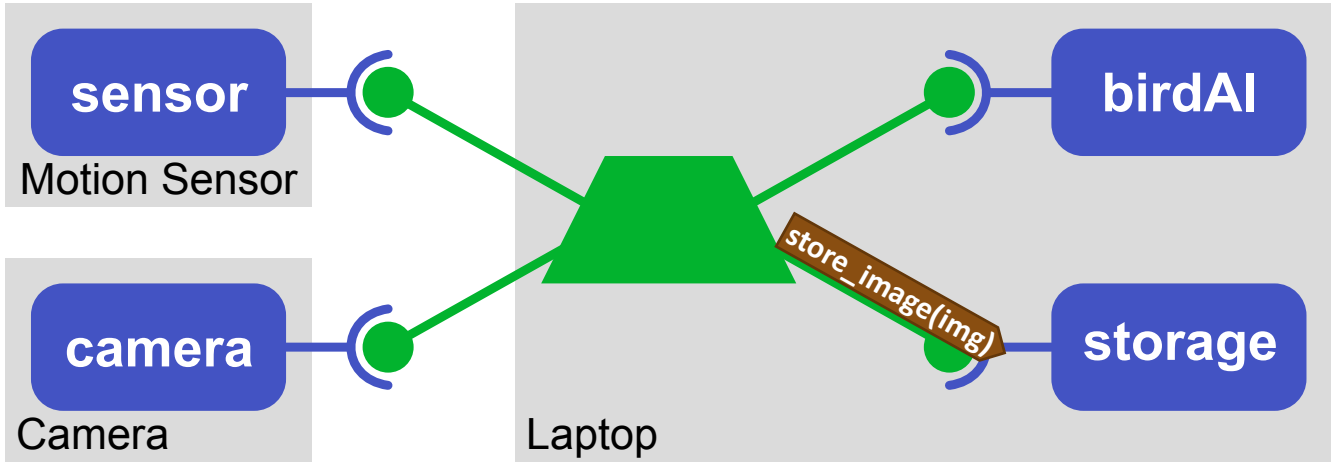
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
      send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



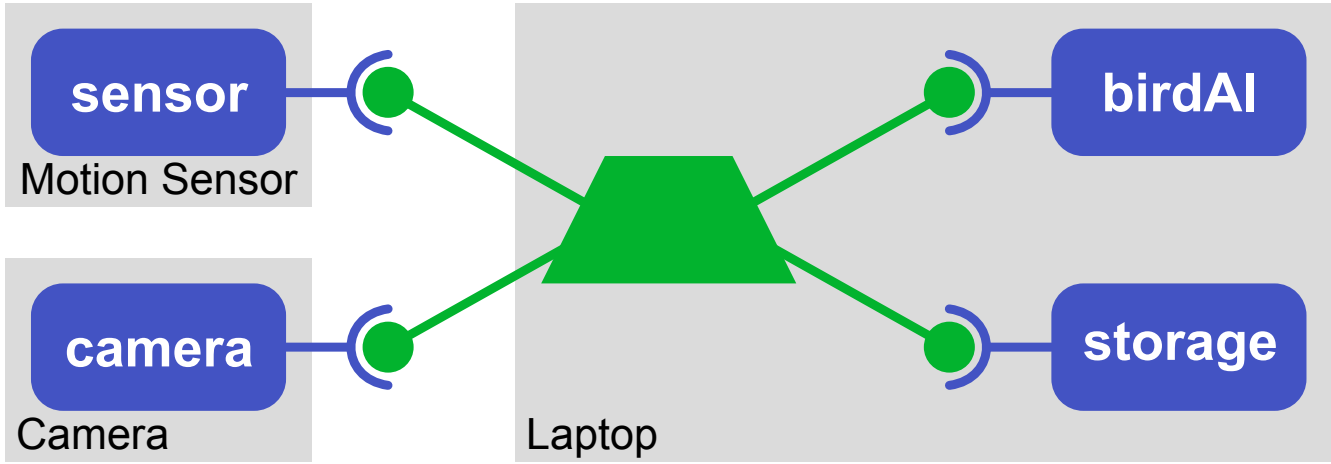
ComPOS example

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
  end
```



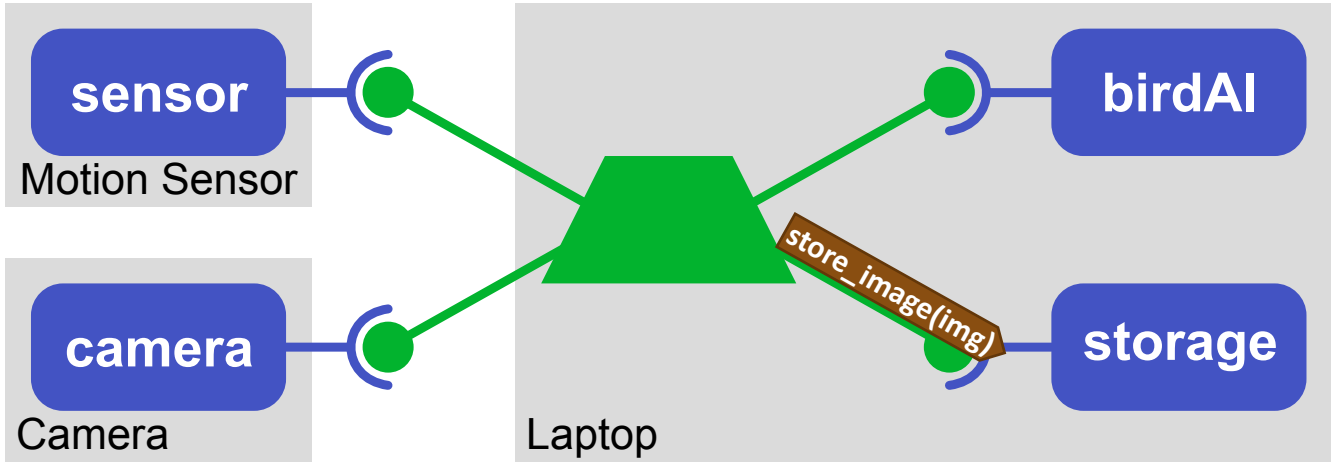
Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



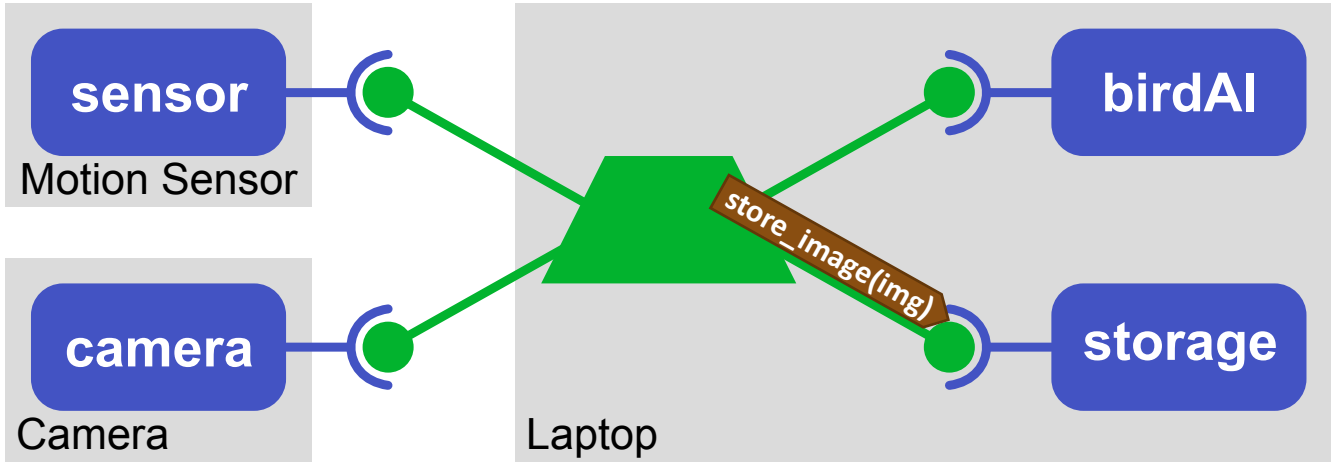
Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
  end
```



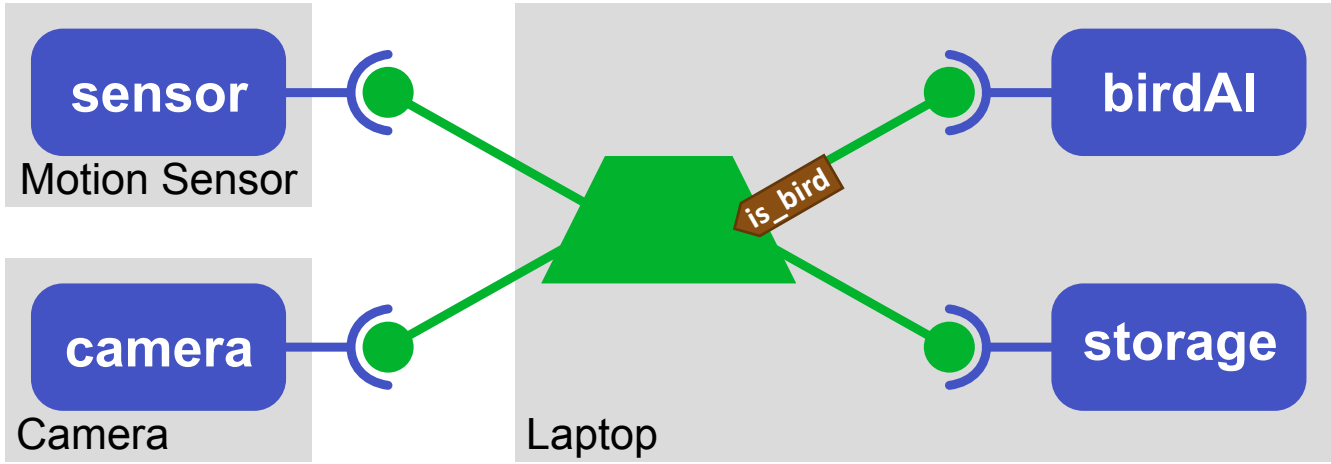
Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
      send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



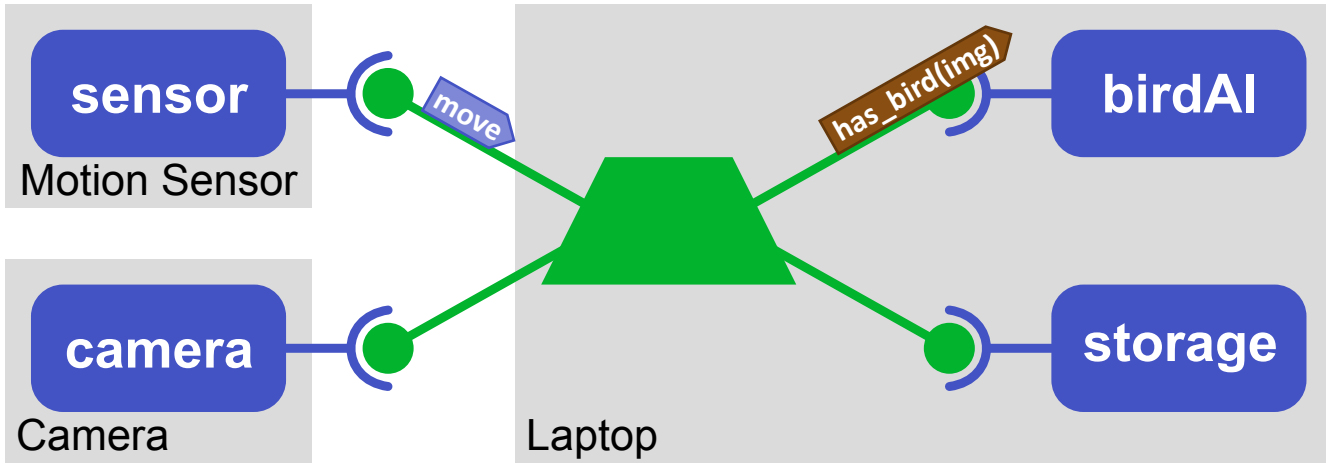
Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
    when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



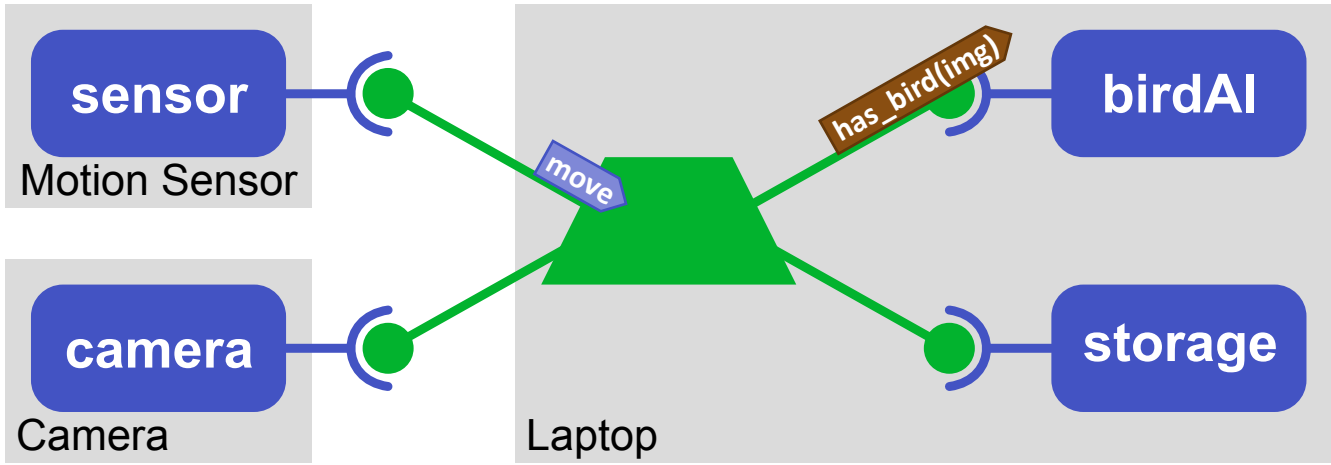
Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



Strategies for handling new messages

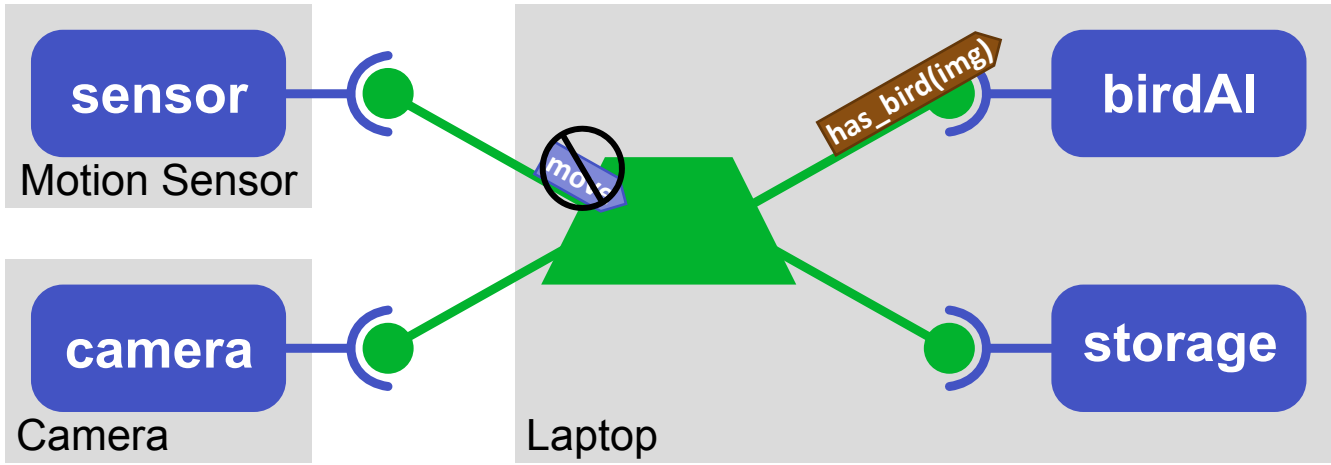
```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



- **Ignore**
- **Queue**
- **Parallel**
- **Abort**

Strategies for handling new messages

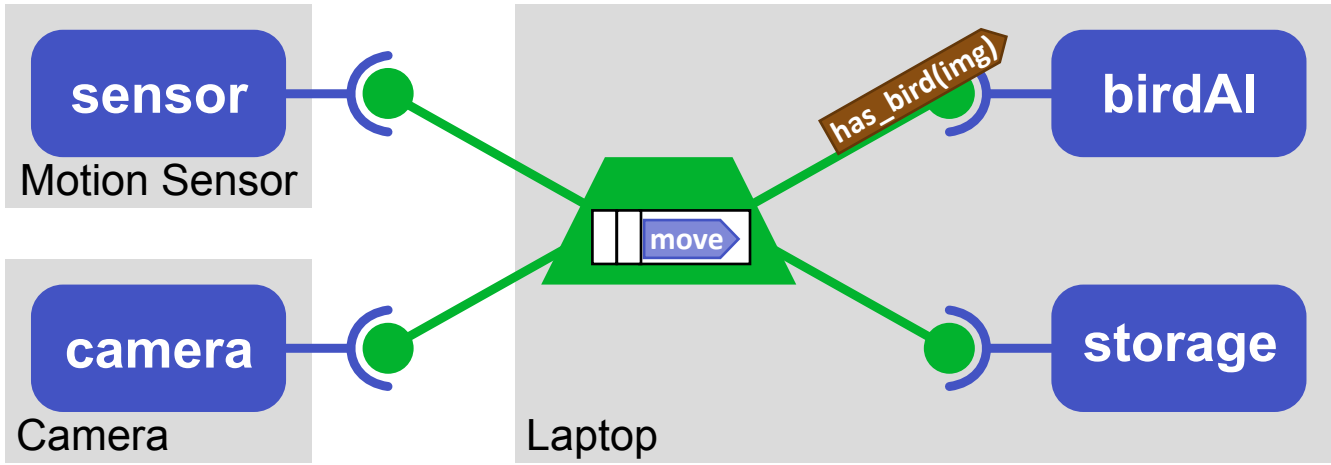
```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



- **Ignore** *the new message*
- **Queue**
- **Parallel**
- **Abort**

Strategies for handling new messages

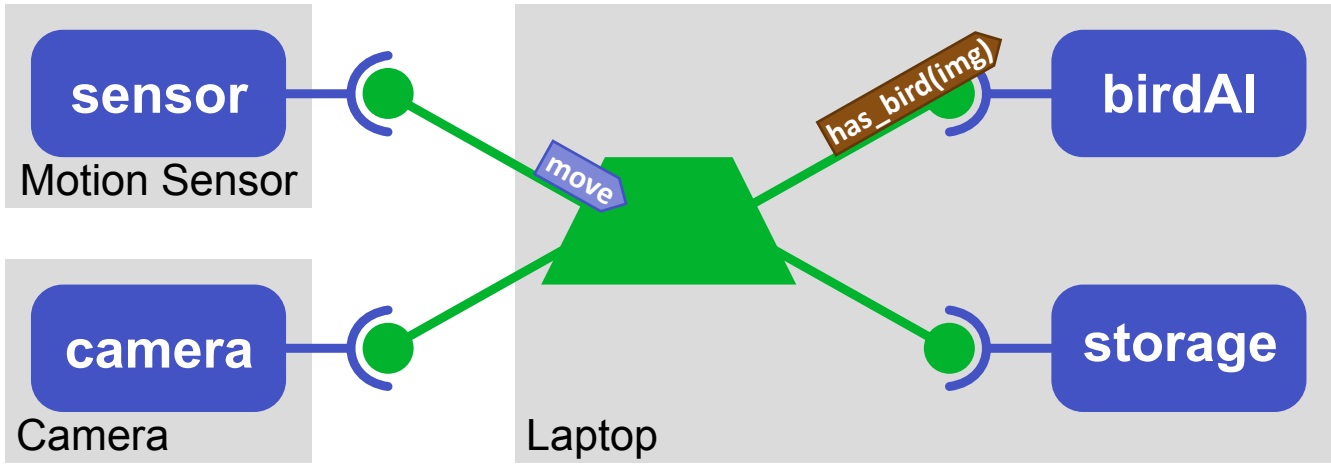
```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
  select
     when resp is_bird from birdAI do
       send cmd store_image(img) to storage
     when resp is_not_bird from birdAI do
    end
```



- **Ignore**
- **Queue** *the new message*
- **Parallel**
- **Abort**

Strategies for handling new messages

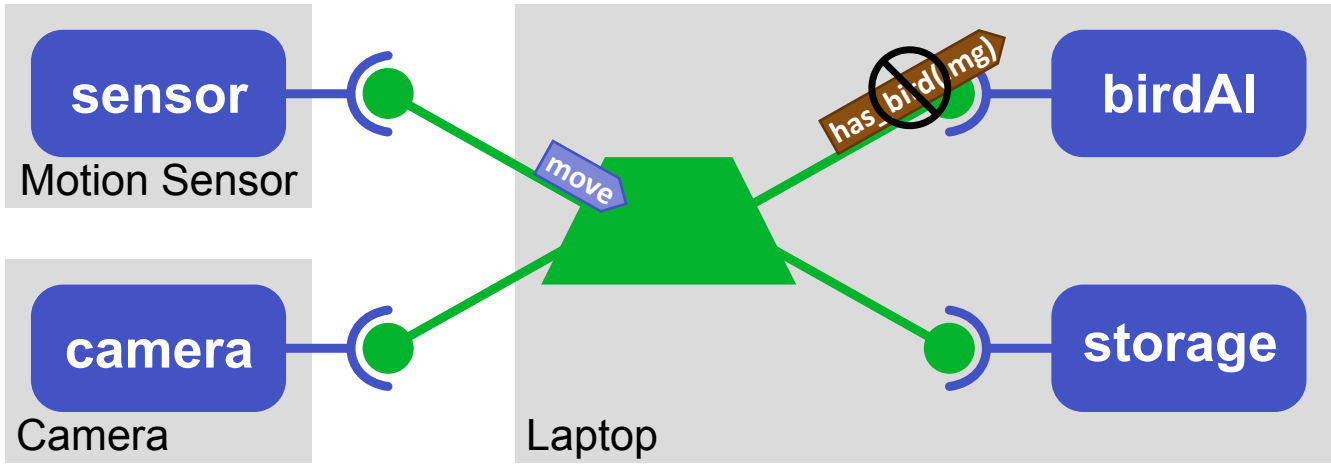
```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



- **Ignore**
- **Queue**
- **Parallel** – *process also the new message*
- **Abort**

Strategies for handling new messages

```
composition SimpleBirdwatcher
  service sensor = ...
  service camera = ...
  service birdAI = ...
  service storage = ...
  when notif move from sensor do
    send req take_photo to camera
    receive resp photo(var img) from camera
    send req has_bird(img) to birdAI
    select
      when resp is_bird from birdAI do
        send cmd store_image(img) to storage
      when resp is_not_bird from birdAI do
    end
```



- **Ignore**
- **Queue**
- **Parallel**
- **Abort** *the old message chain* ***(our choice)***

Strategy comparison

	abort	*ignore*	*queue*	*parallel*
Responsive	Yes	No	No	Yes
Bounded	Yes	Yes	No	No
Eager Abort	Yes	No	No	No
Need Timer	No	Yes	Yes	Yes

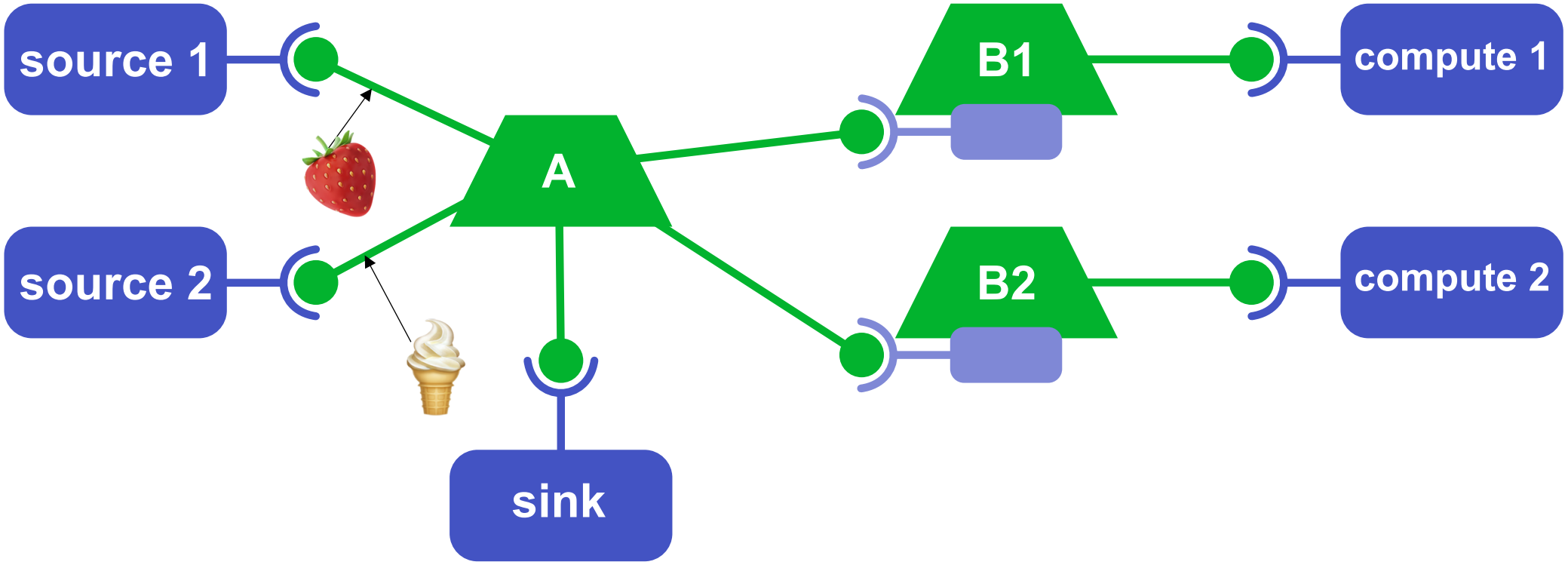
Using abort we can implement other strategies with additional services

Implementing abort with Epoch

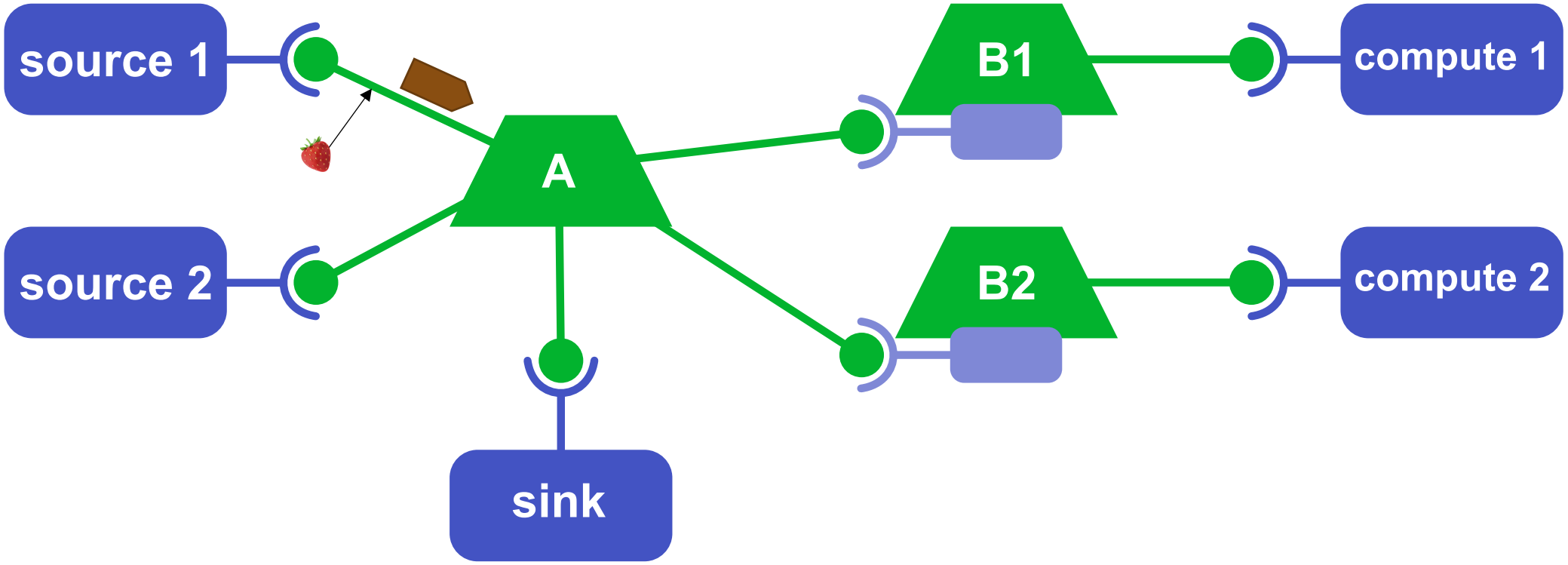
- Only newer “related” messages can abort a reaction
- An epoch has a place (connection) and time (logical clock)
  - P_t
- An epoch aborts another epoch when they are from the same place and one is newer then the other
  - $P'_{t'}$ aborts P_t **IFF** $P' = P$ and $t' > t$

Epoch example 1

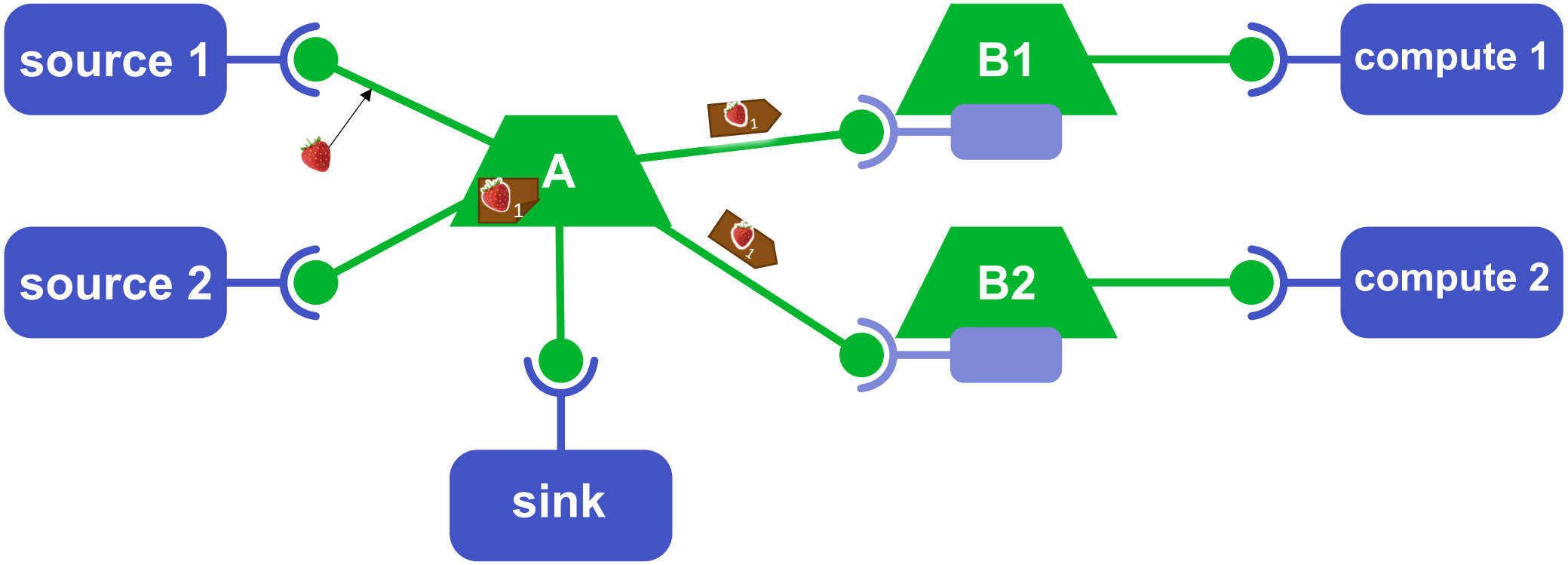
Epoch example 1



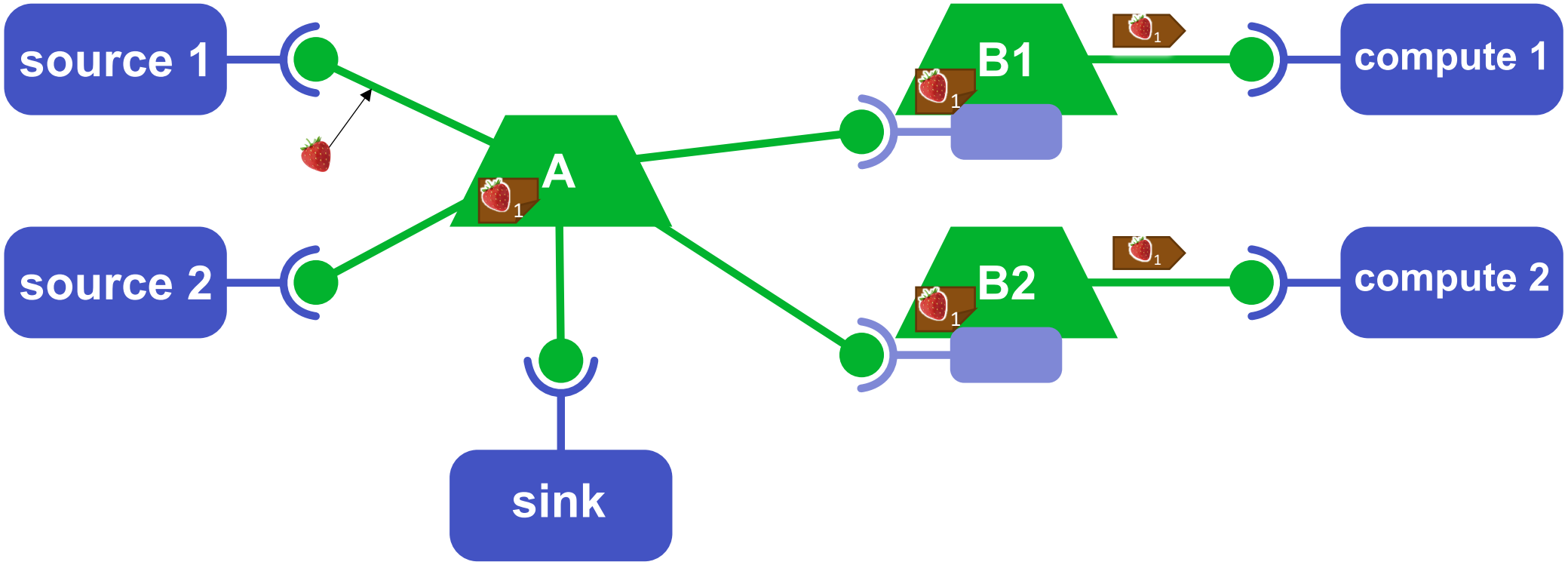
Epoch example 1



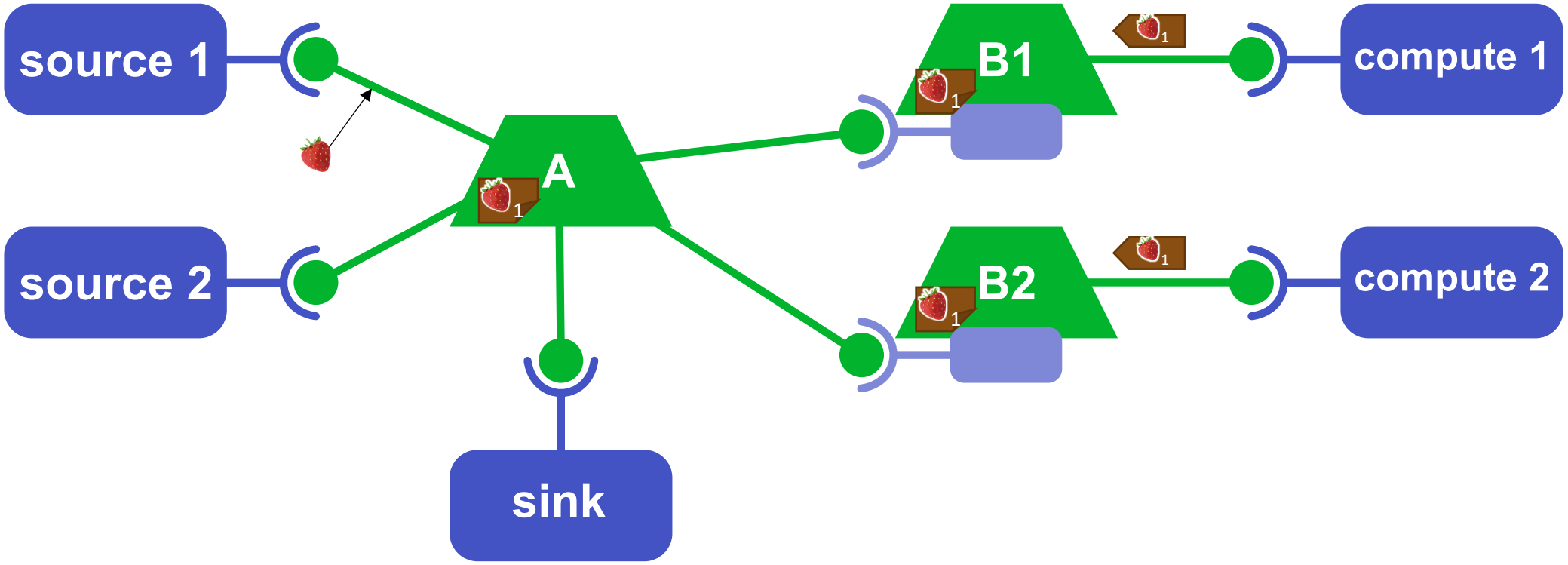
Epoch example 1



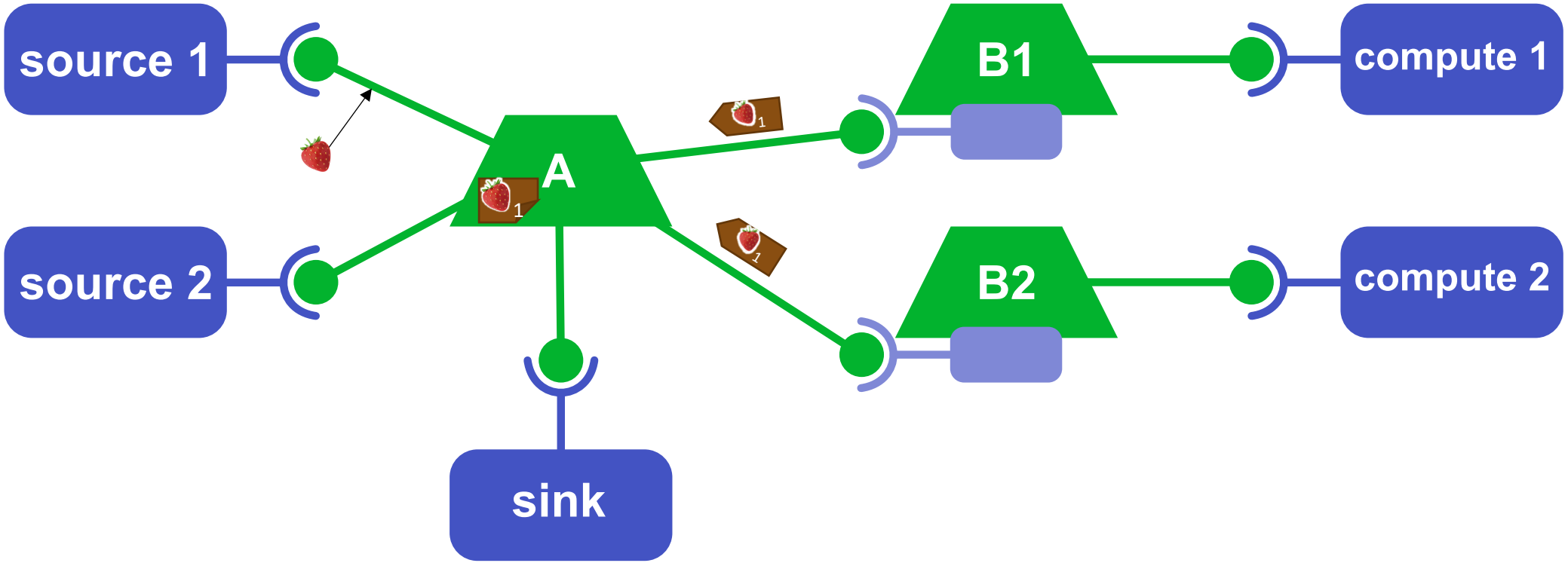
Epoch example 1



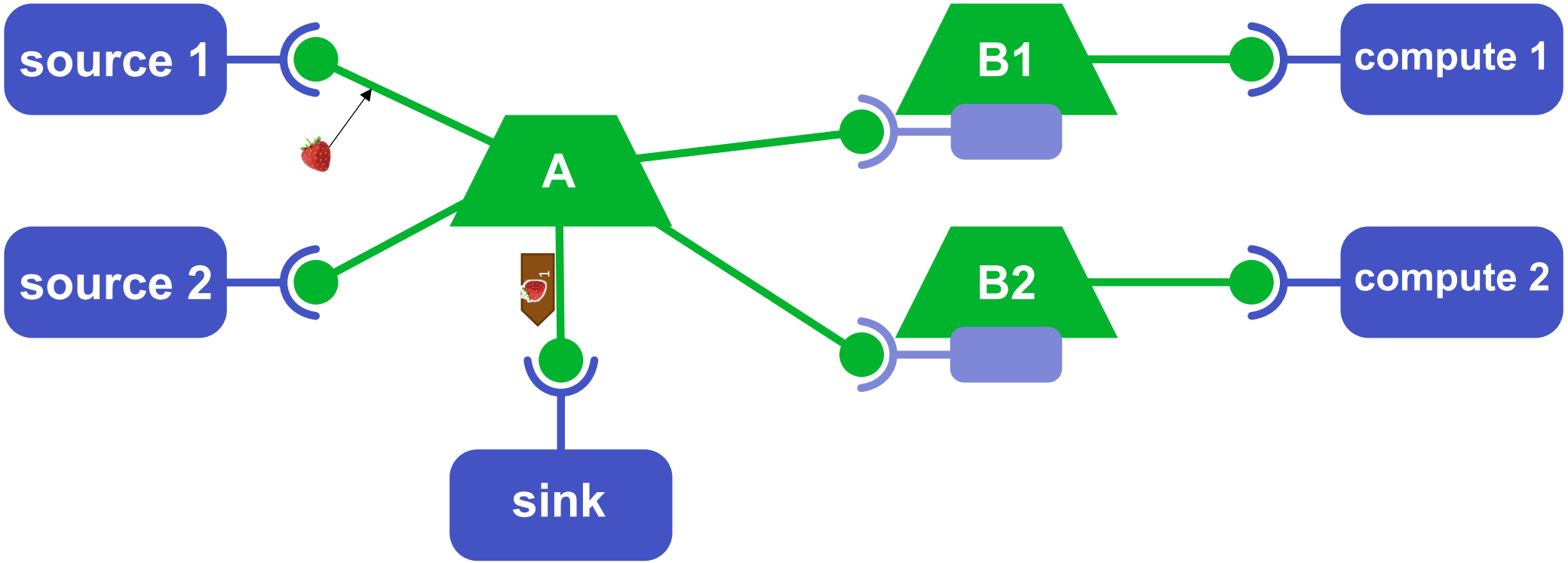
Epoch example 1



Epoch example 1

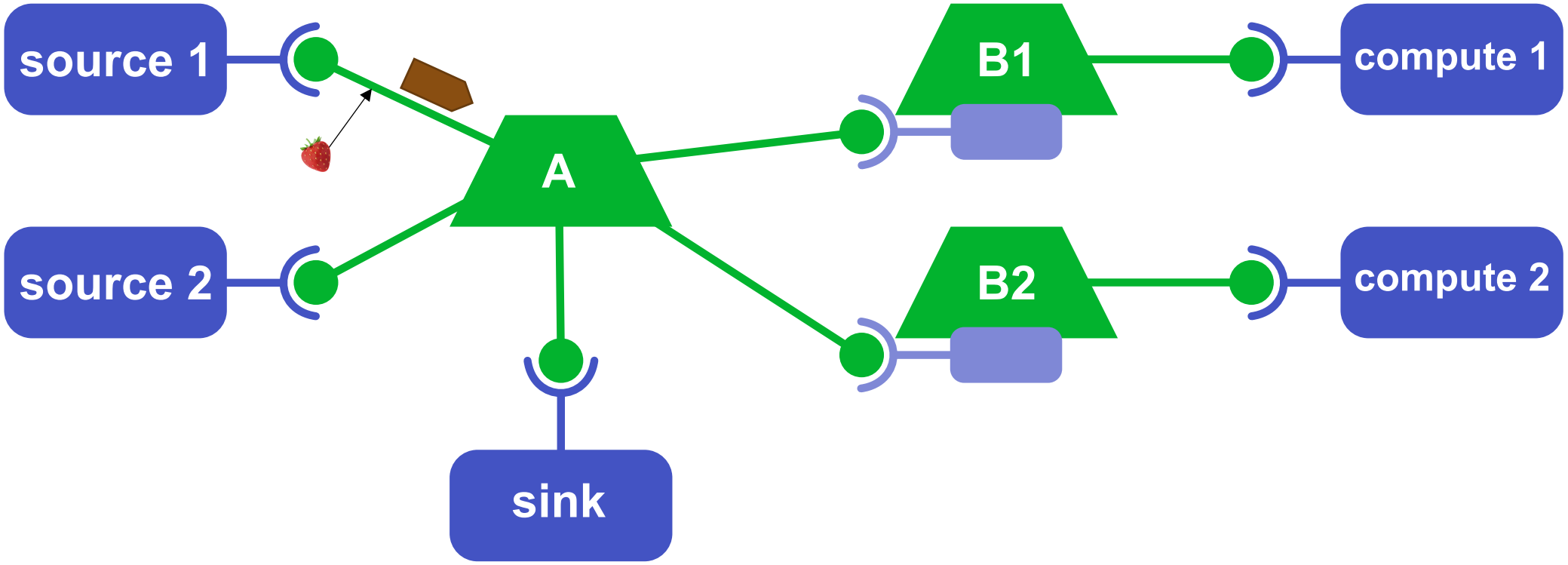


Epoch example 1

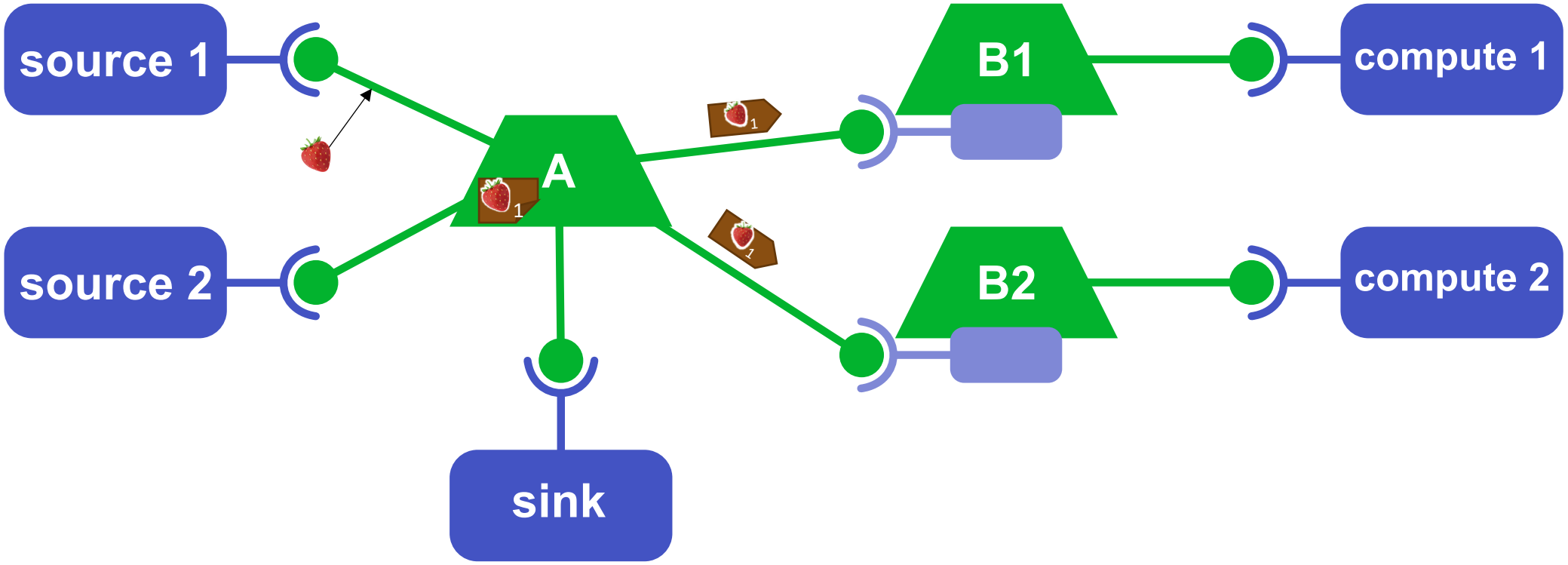


Epoch example 2

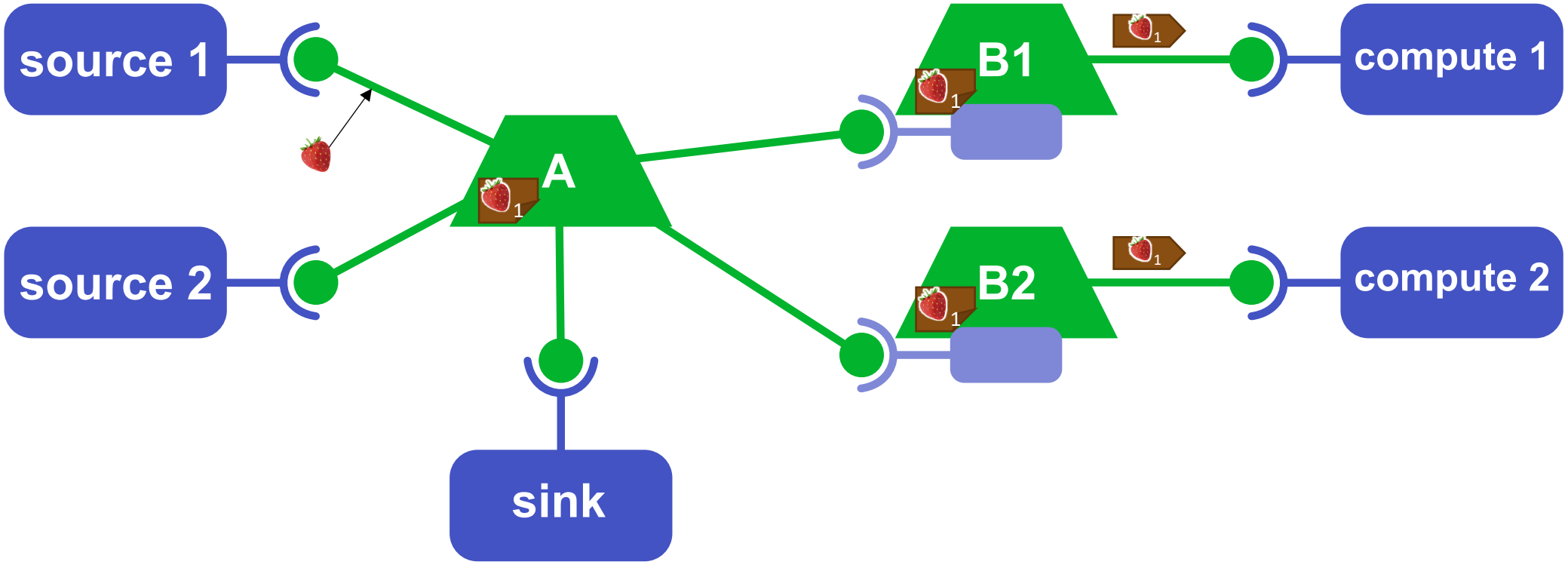
Epoch example 2



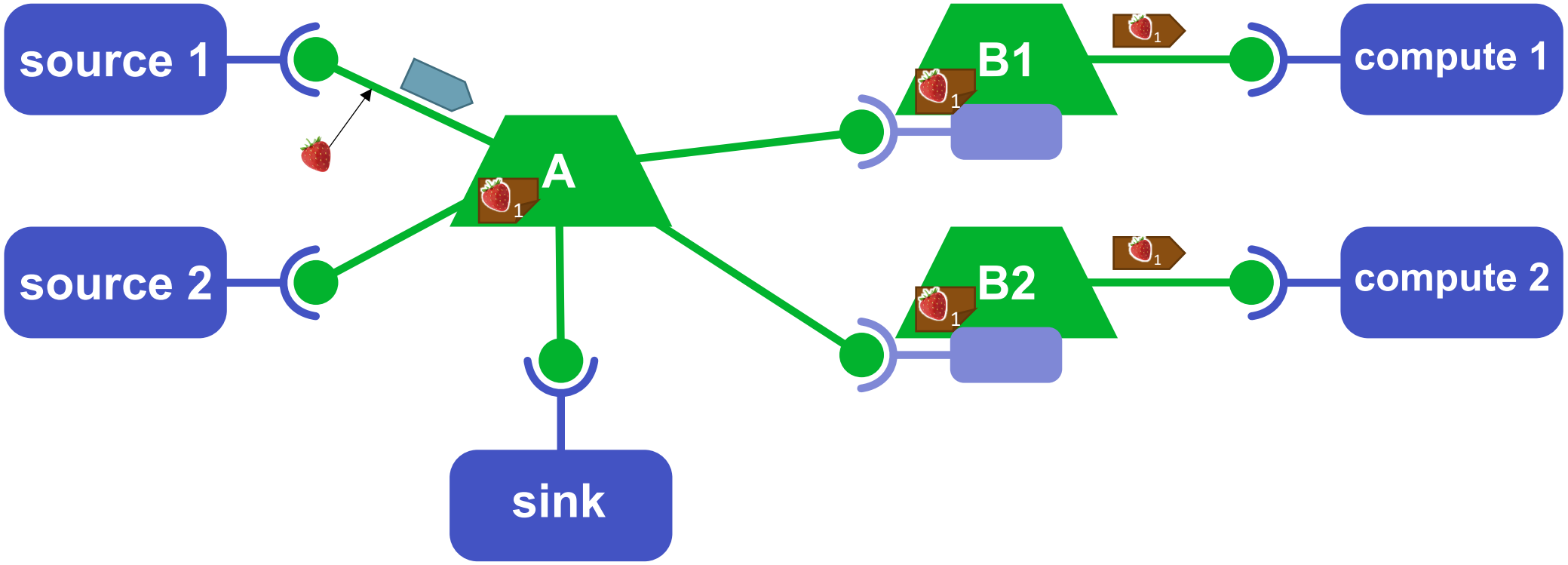
Epoch example 2



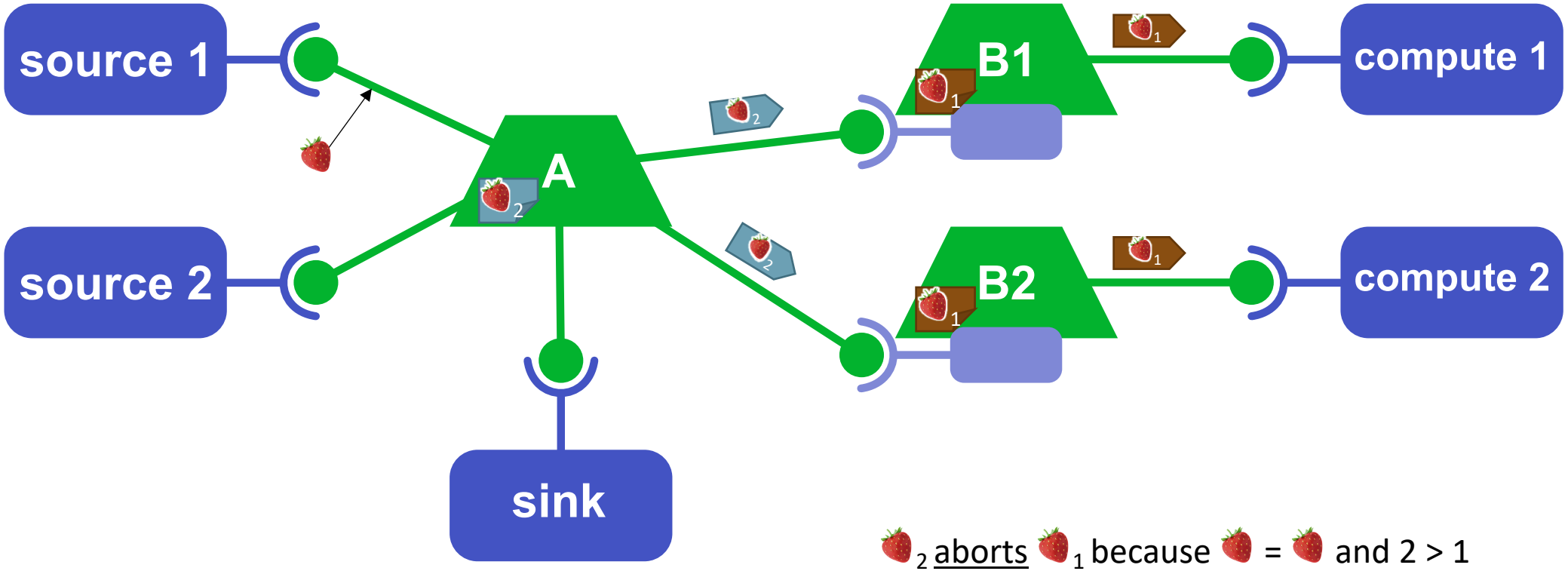
Epoch example 2



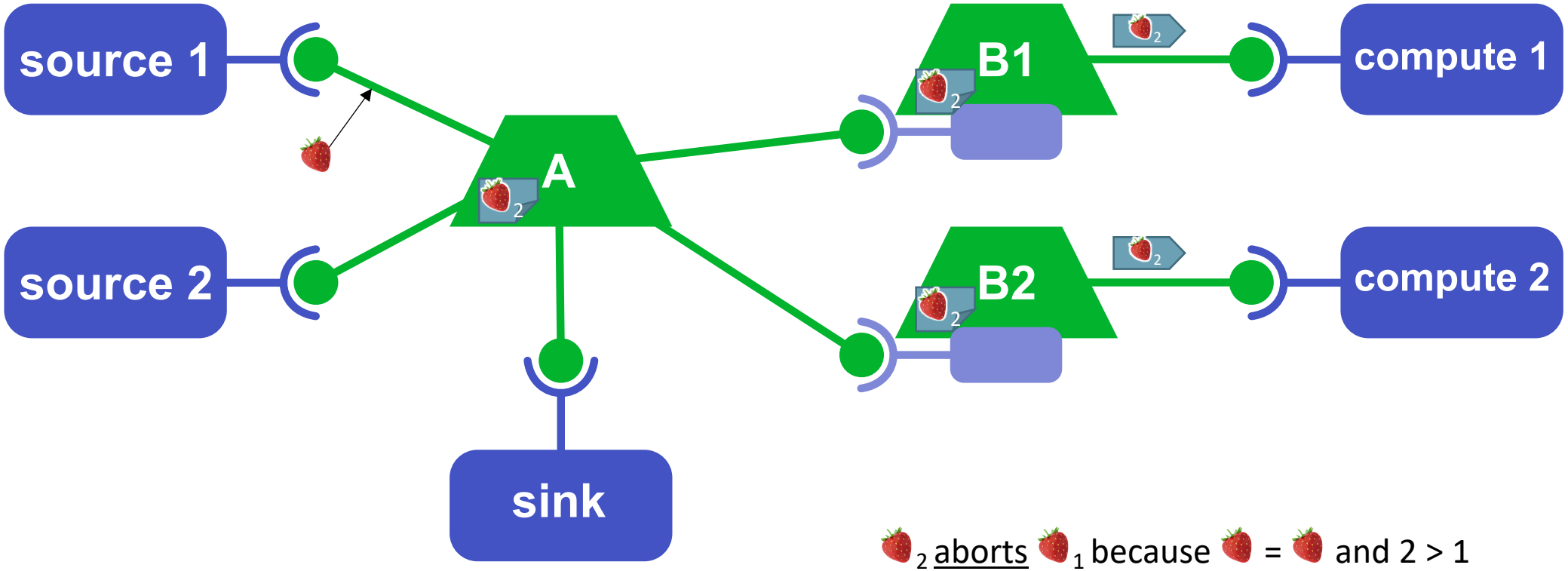
Epoch example 2



Epoch example 2

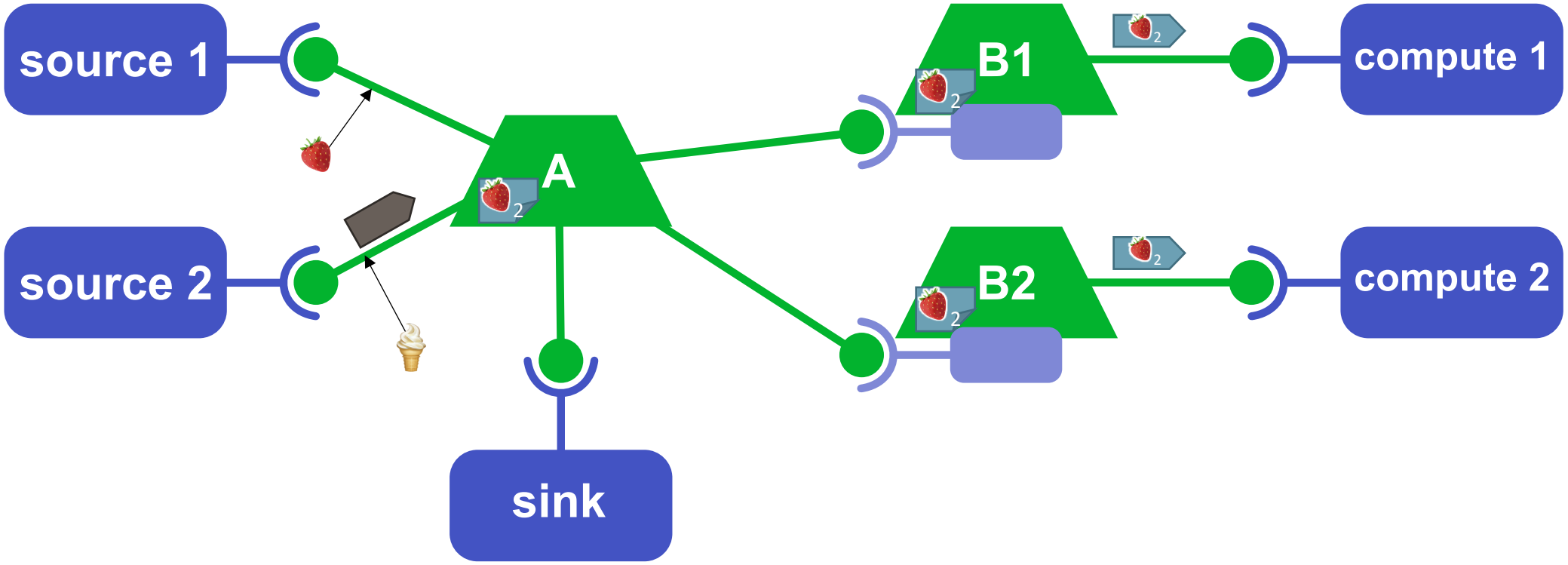


Epoch example 2

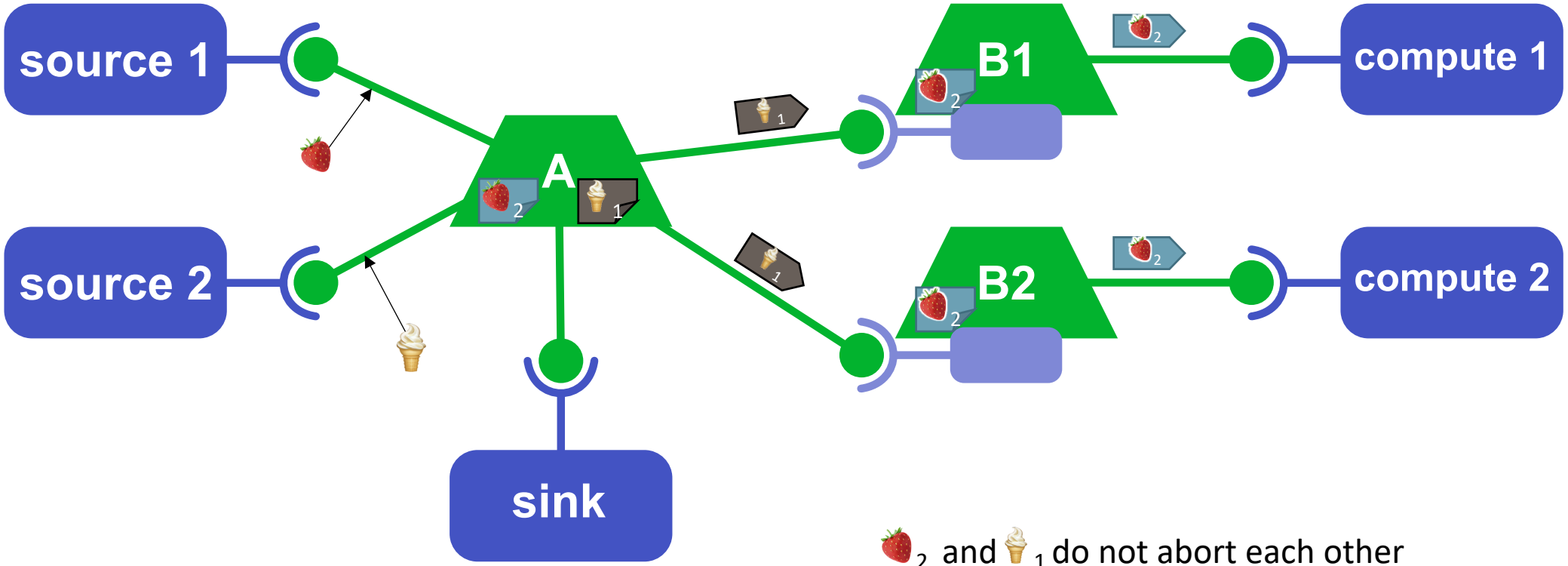


2 aborts 1 because = and 2 > 1

Epoch example 2

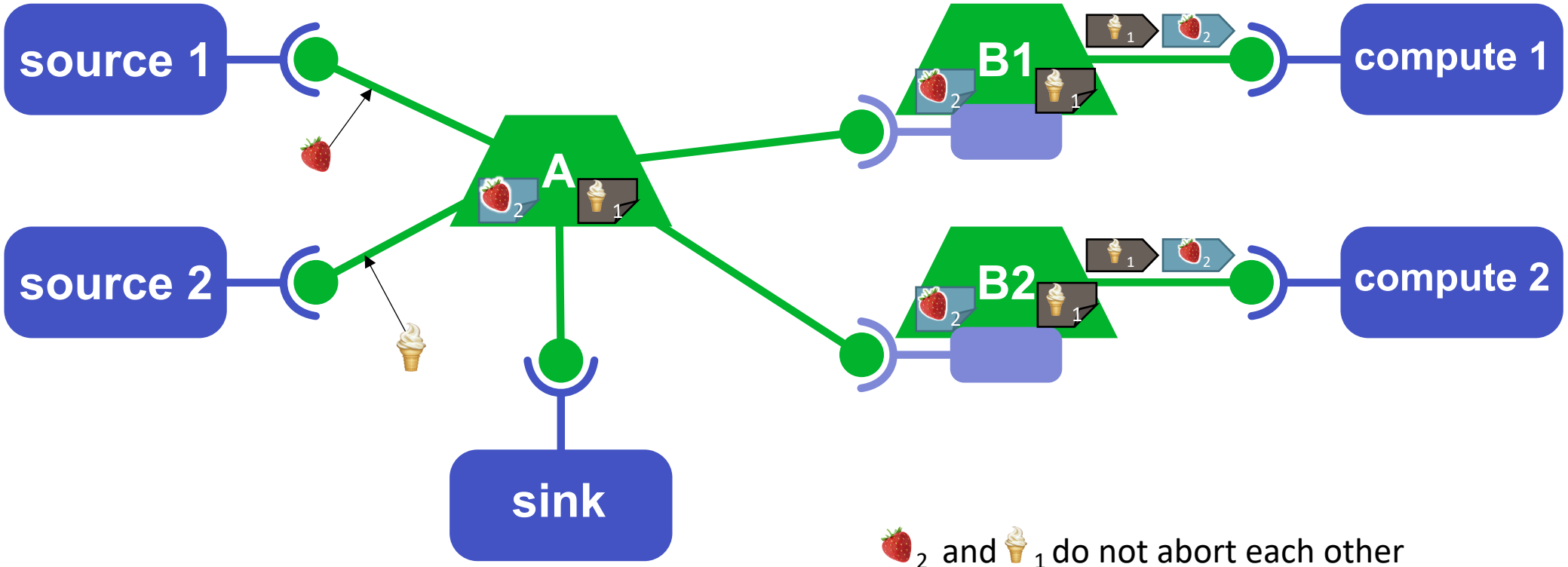


Epoch example 2



🍓$_2$ and 🍦$_1$ do not abort each other because 🍓 ≠ 🍦

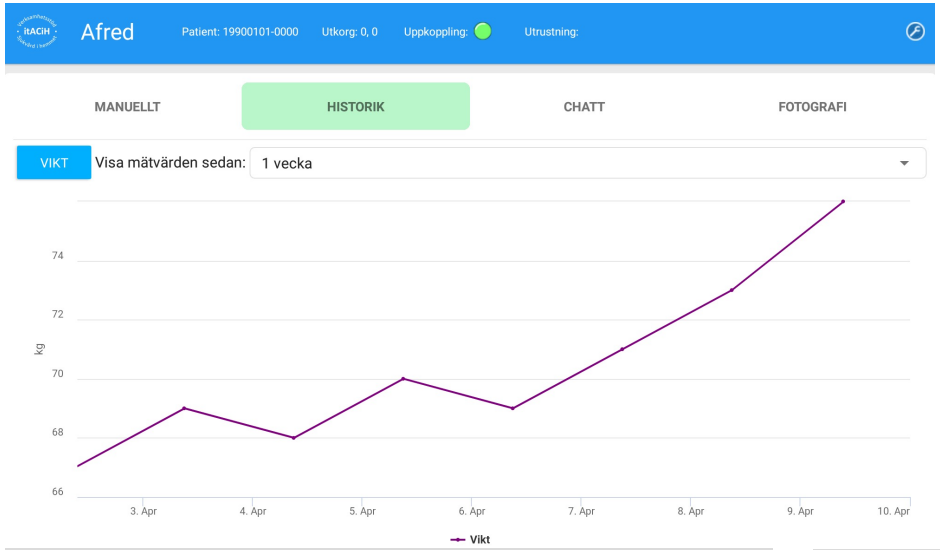
Epoch example 2

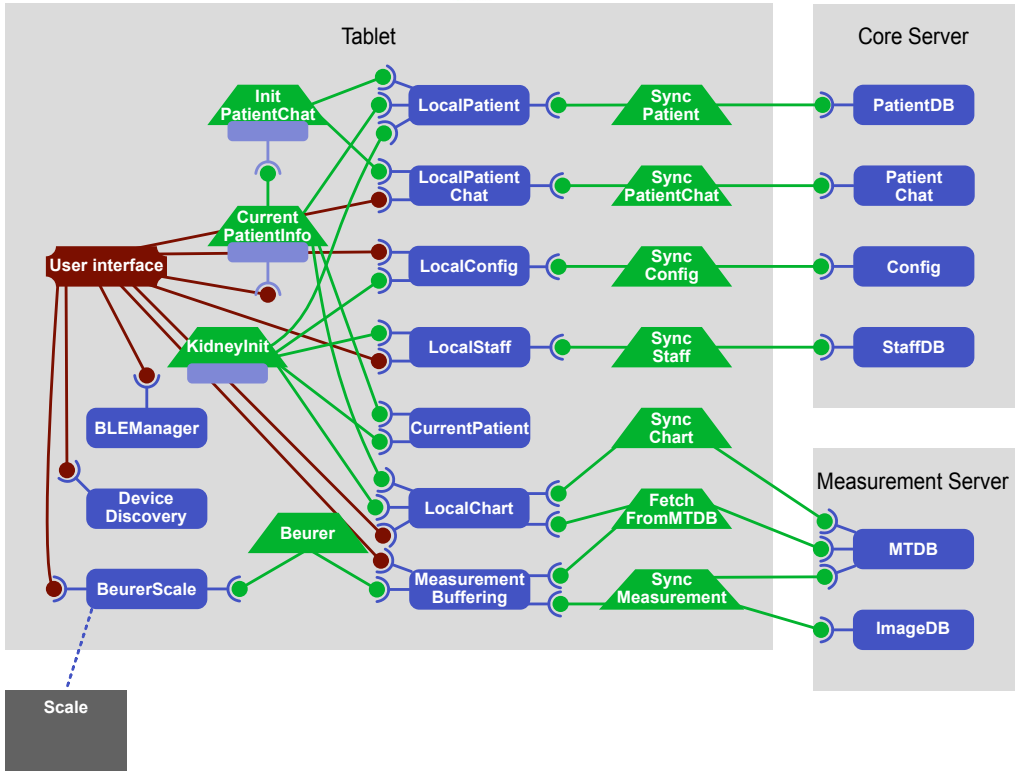


🍓$_2$ and 🍦$_1$ do not abort each other because 🍓 ≠ 🍦

Case study

- Home care of kidney failure
- Reimplementation of 11 compositions using ComPOS
- ComPOS more explicit control flow than original composition language

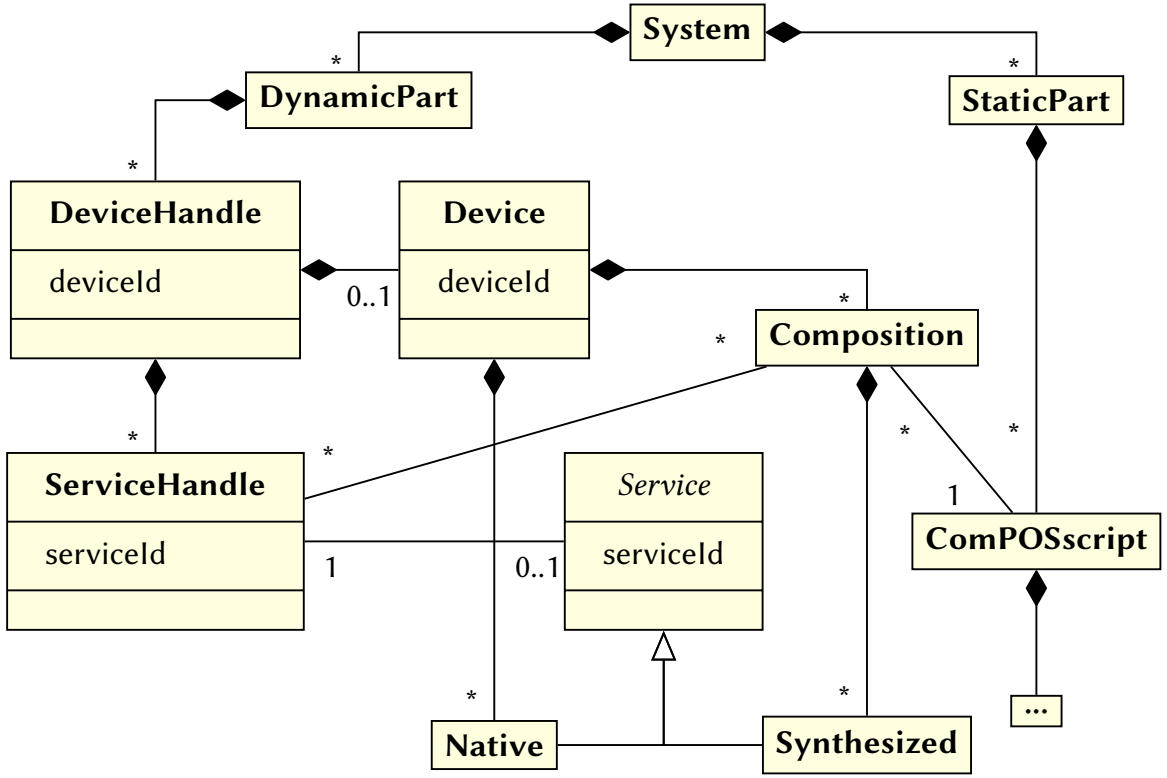




Paper 3: Runtime modeling and analysis of IoT systems

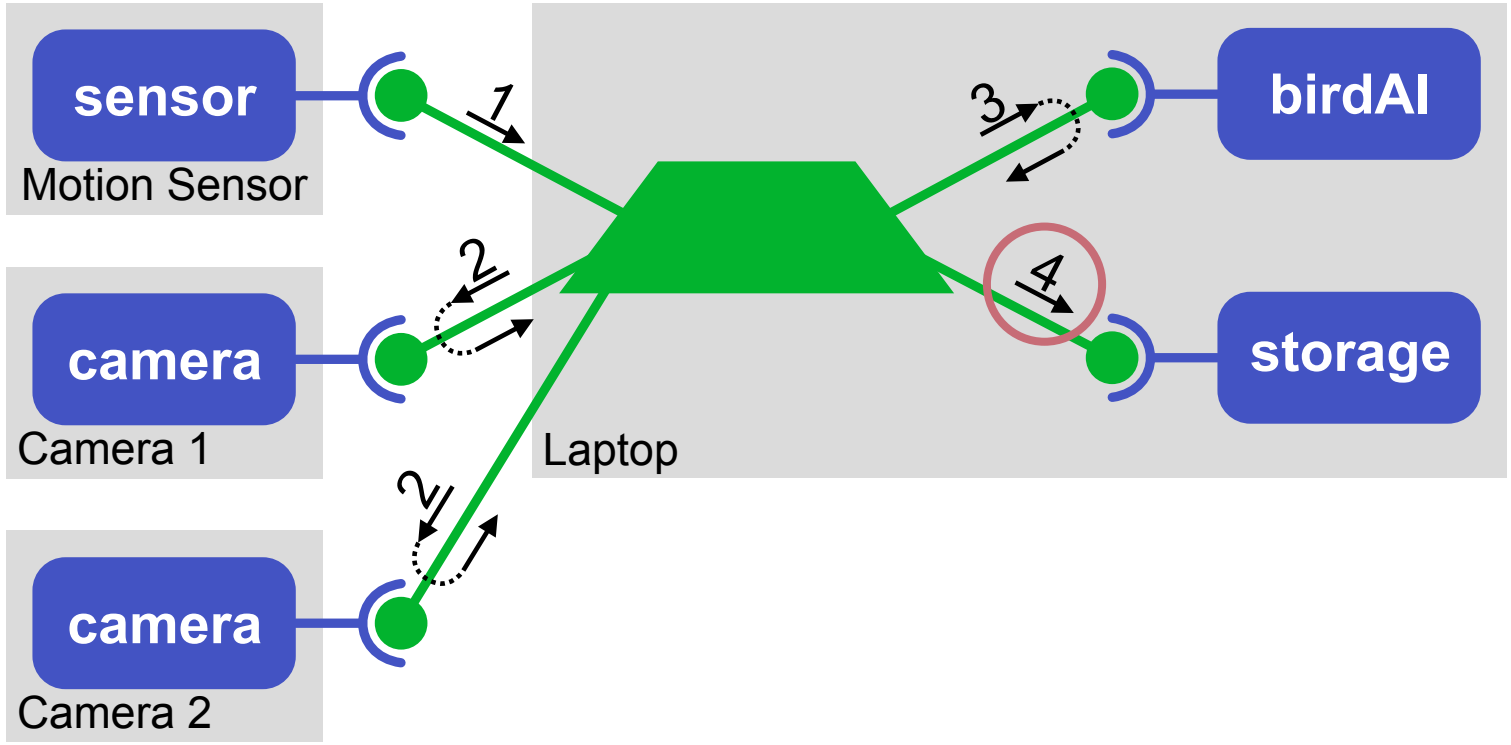
- Runtime model of the system specified using relational reference attribute grammars
- Analysis specified using reference attribute grammars
- Device Dependency Analysis finds: *What devices are needed for a specific event to happen?*

Meta-model



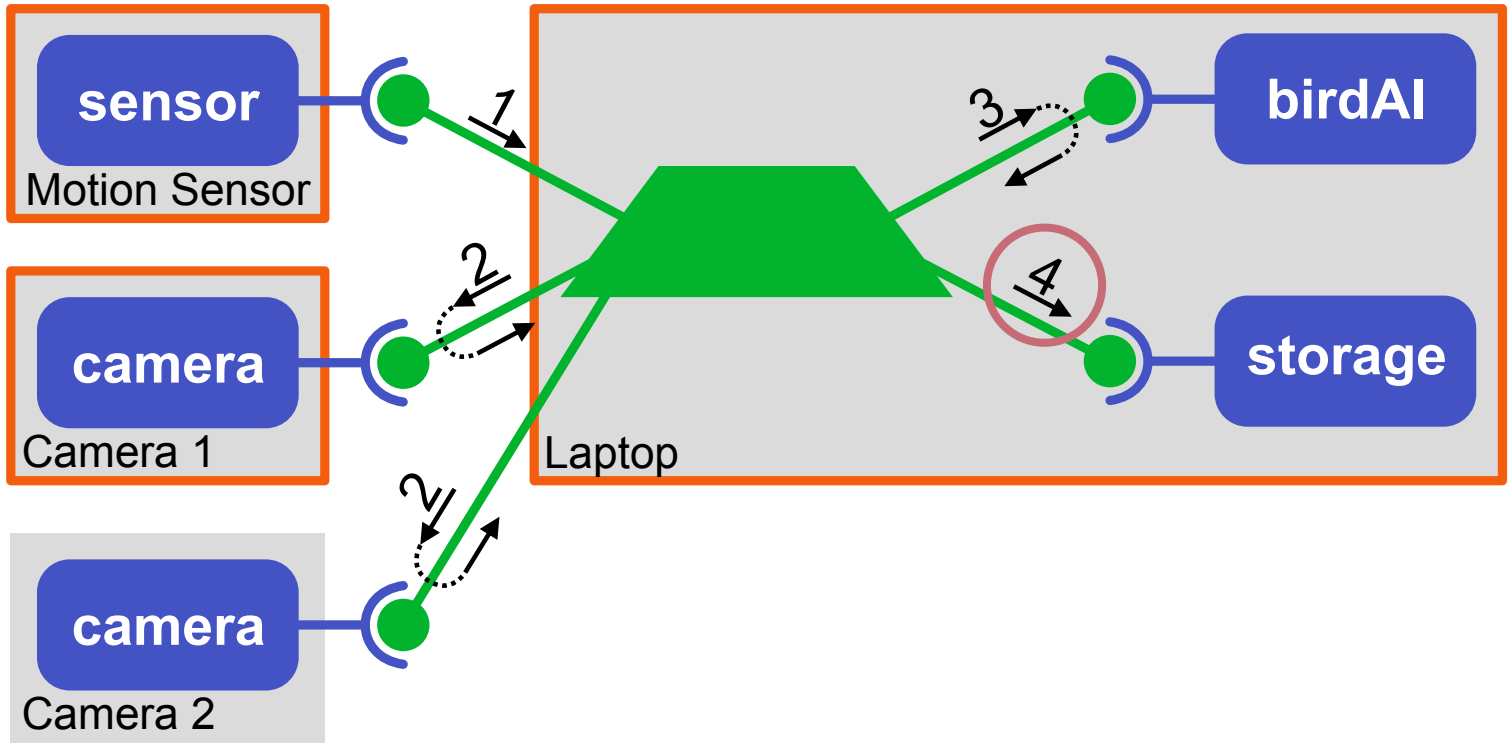
Device Dependency Analysis

- *What devices are needed for 4 to happen?*



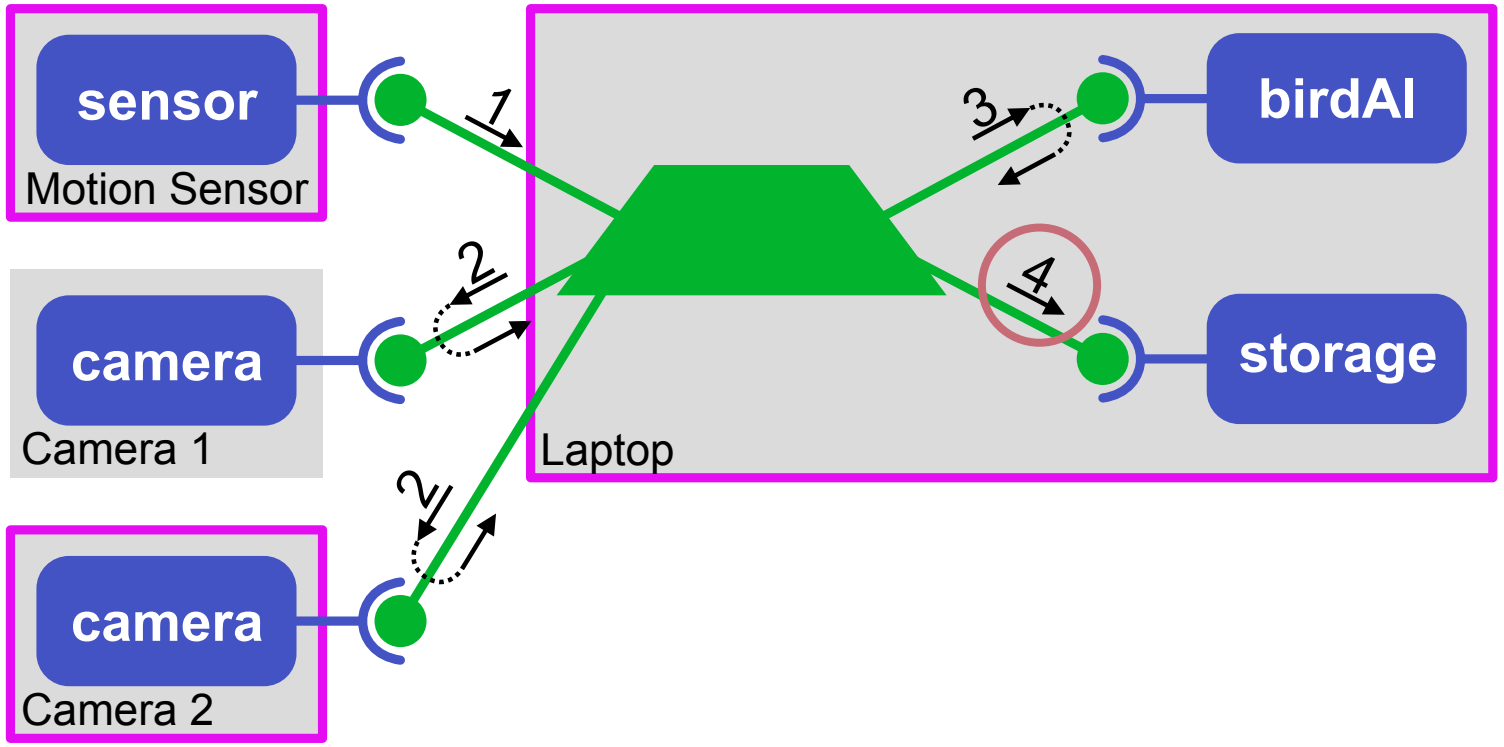
Device Dependency Analysis

- *What devices are needed for 4 to happen?*



Device Dependency Analysis

- *What devices are needed for 4 to happen?*

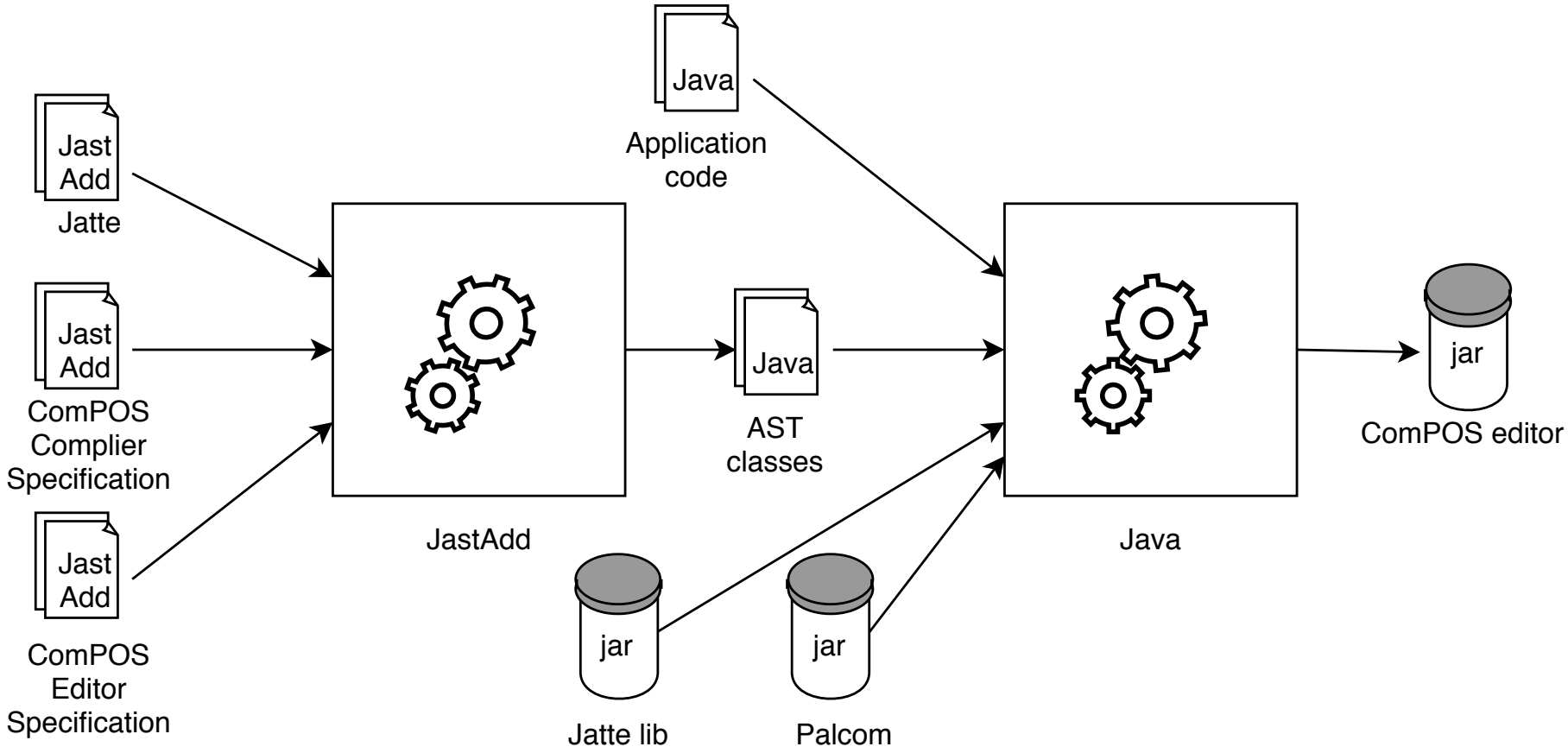


Paper 4: Jatte: A Tunable Tree Editor for Integrated DSLs

- Framework for integrated DSL editors
- Customizable using Attribute Grammars (JastAdd)

Requirement	Solution
Fast prototyping	Editor generated from abstract grammar
Customizable	Tunable using attribute grammars
Hide information	Projectional editing
Interact with Palcom Browser	Drag and Drop

Architecture



Results

Paper 1

- We propose three activities for programming Palcom IoT systems: Explore, Compose, and Expose.
- We classify the Palcom programming environment as between levels 3 and 4 using Tanimoto's levels of liveness.

Paper 2

- We propose ComPOS, a DSL for composing services.
- We describe four different strategies for handling new messages: Ignore, Queue, Parallel, Abort.
- We implement Abort using epochs.
- We show how to get the other strategies using an additional service.
- We evaluate ComPOS in a case study reimplementing compositions in a home care system.
- The case study shows that ComPOS has more explicit control flow than the original composition language.
- We propose implementations for seven common home automation scenarios proposed by Rodeíguez Avila et.al.

Paper 3

- We propose a conceptual model for Palcom systems specified using Relational Reference Attribute Grammars.
- We formulate and implement the Device Dependency Analysis (DDA) on top of our conceptual model.

Paper 4

- We propose a new technique for developing integrated projectional editors using reference attribute grammars.
- We implement this technique in the Jatte tool and assess it by implementing editors for a toy language and ComPOS.

THE END