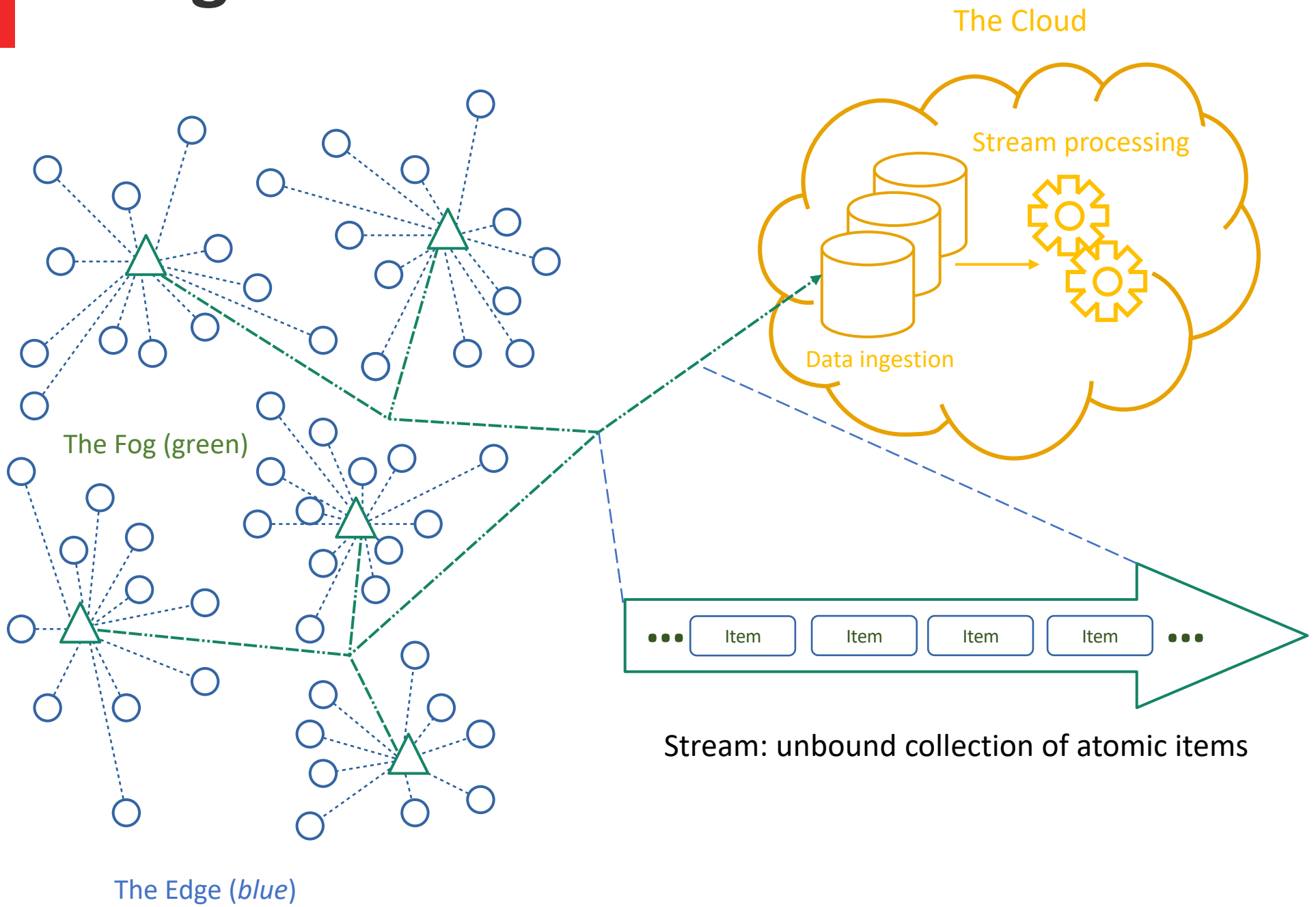
Closing the gap between Edge and Cloud

Smart application deployment on hybrid architectures

Background


The Cloud
Stream processing
Data ingestion
The Fog (green)
Item
Item
Item
Item
Stream: unbound collection of atomic items
The Edge (*blue*)

Background

- How is application performance affected by the difference of bandwidth between Edge and Cloud?
  - *With a sufficiently large bandwidth (e.g. fiber), is it still necessary to execute some computation on the Edge?*

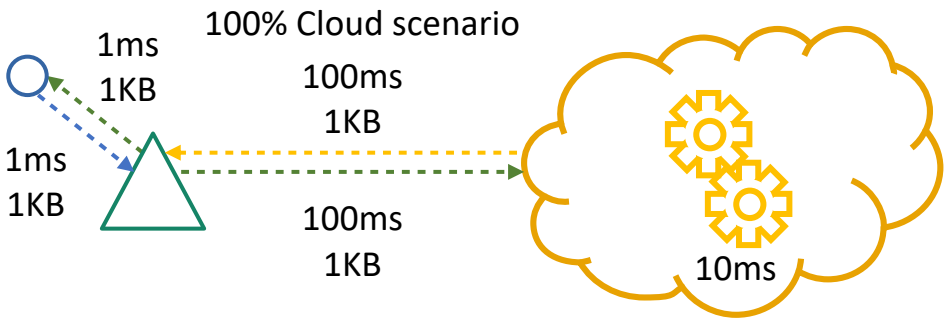
- How does Edge computing impact throughput and machine resources?
  - *The reduction in the amount of data transferred to the Cloud and the latency of the response is not hindered by the processing time and energetic consumption on the Edge?*

- How are Cloud-based frameworks impacted by Edge computing?

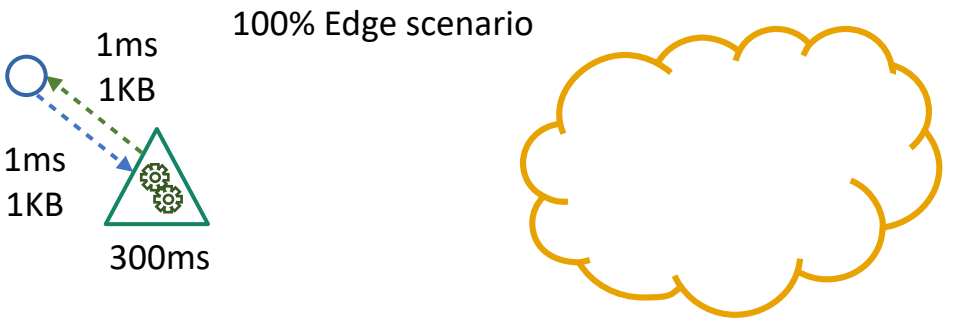
- Is Edge computing really necessary for all scenarios?

Background

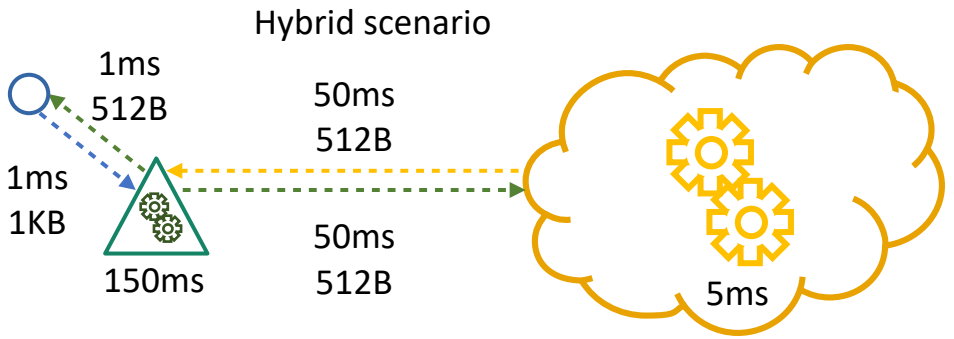
Can Edge computing have a *negative* impact?



Total data transferred: 4Kb
Total time: 212ms



Total data transferred: 2KB
Total time: 302ms



Total data transferred: 2.5KB
Total time: 257ms

Problem statement

The application graph

Applications are modeled as *stream graphs*, which are represented as DAGs

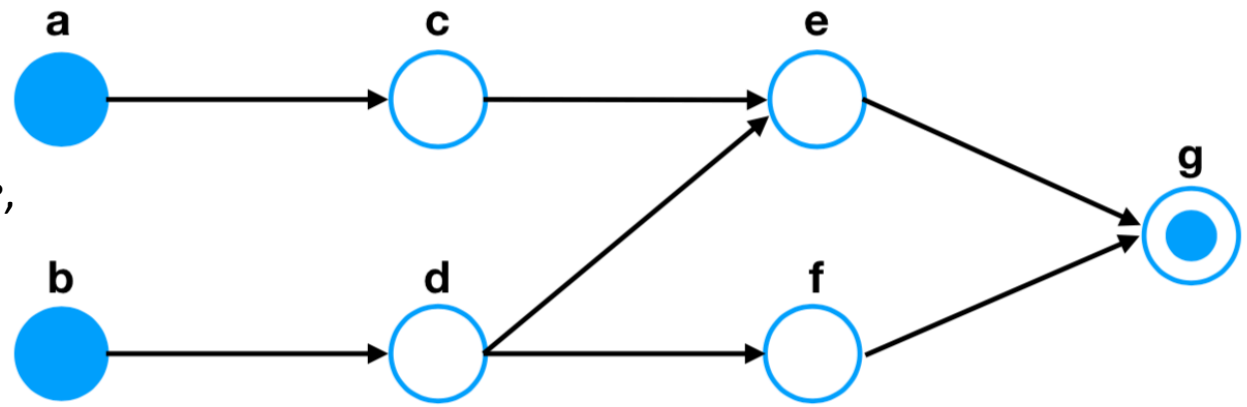
$$A = (P, L)$$

Where:

- P is a set of *operators*
- L is a set of *streams,* i.e. *data dependencies* between operators

3 types of operators are considered:

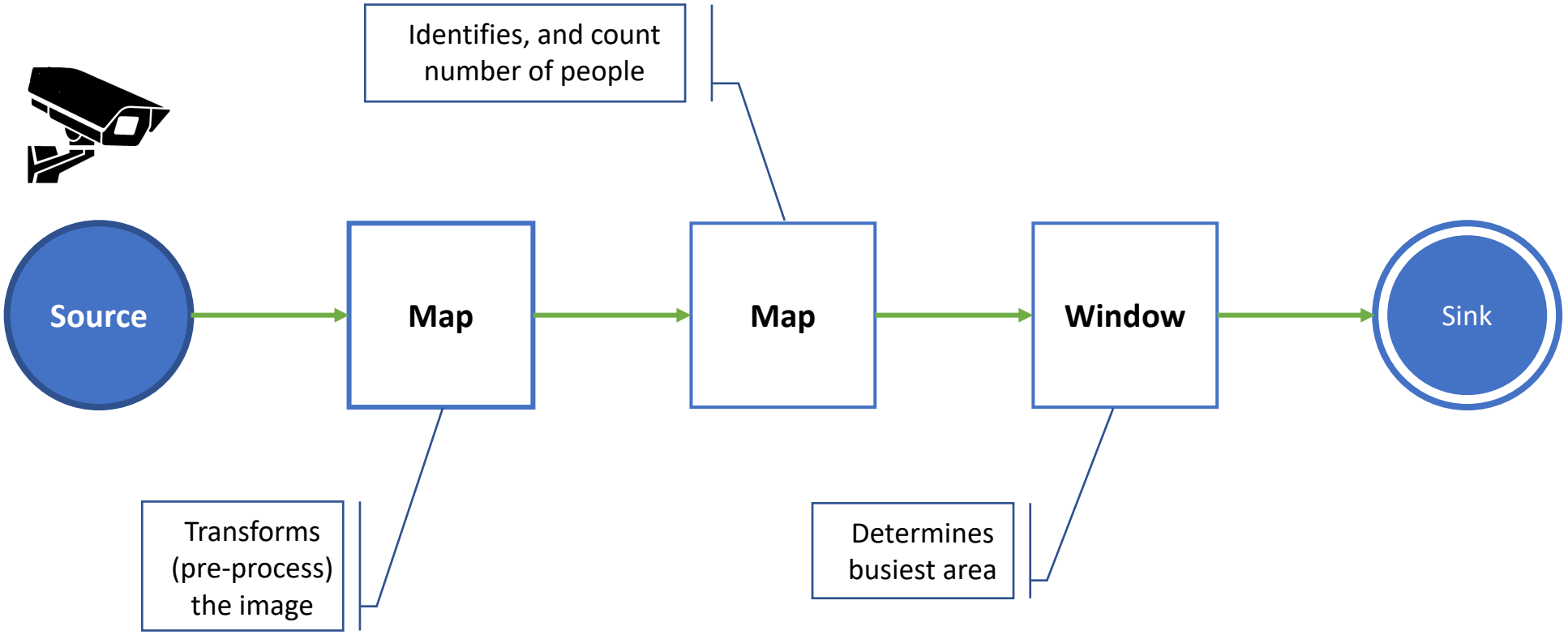
- **Sources** that only *produce* data
  $S = \{a, b\}$
- **Processing** operators that *receive*, *transform* and *send* data
  $O = \{c, d, e, f\}$
- **Sinks** that only *consume* data
  $E = \{g\}$



Problem statement

The application graph

An example of a simple application graph: *CCTV surveillance*



Problem statement

The application graph

Planner takes as input an application graph and select those operators that are best suited to be executed on the Edge (in short, they must be *stateless*, and mustn't have data dependencies on any *stateful* operator.

There are 6 types of stateless operators:

- **Map:** takes one item and produces one item.
- **Flat Map:** takes one item and produces zero, one or more items.
- **Filter:** takes one item and produces zero or one item.
- **Split:** takes one stream and produces two or more streams.
- **Union:** takes two or more streams and produces one stream.
- **Select:**

Each operator encodes the resources constraints that a potential hosting machine must satisfy. In the current work, two resource constraints are defined:

- O^{MOP}: millions of operations per second
- O^{MEM}: RAM memory

which represent the minimal processing capability and minimal amount of RAM memory available on a potential hosting machine O

Problem statement

The network graph

The network graph is represented as a DAG

$$G = (H, X)$$

Where:

- H is the set of *hosting machines*
- X is the set of *connections* between them

Each hosting machine $h \in H$ has a series of parameters that describe its capabilities:

- h^{MOP}: millions of operations per second
- h^{MEM}: available RAM memory
- h^{COST}: renting cost of the machine
- h^{KW}: energetic cost of the machine
- h^{HC}: cost per unit of time

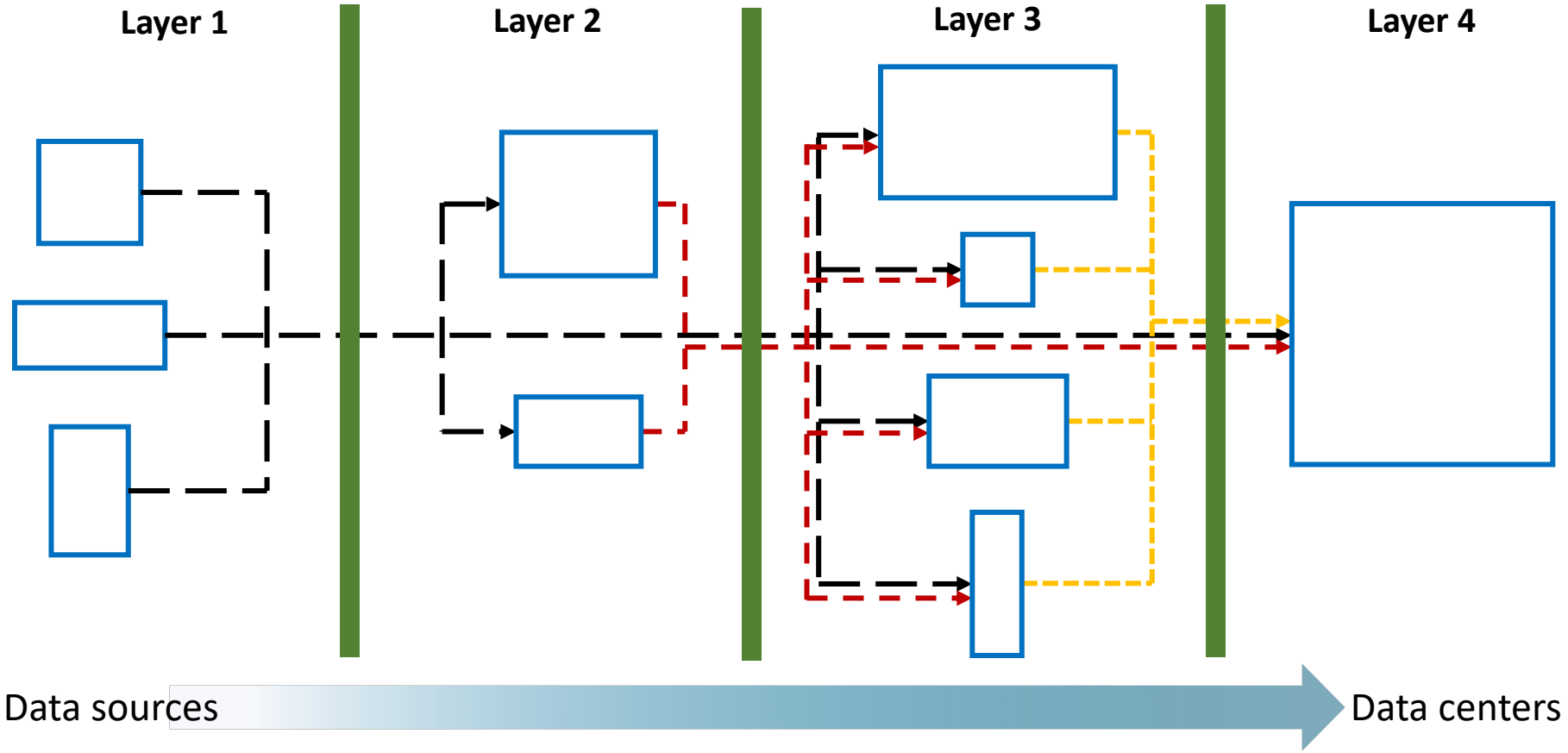
Host machines are organized in *layers*, which are abstract representations of regions in the infrastructure.

The set of layers is defined as L, and each $\ell \in L$ has an associated parameter ℓ^{IMP}, which is the cost of enabling ℓ in order to use its machines.

$$\ell_p \; for \; p \in P$$

denotes the layer of operator p.

Problem statement



Connections between machines depends on the layers they belong to

$$x_{\ell_1,\ell_2} \in X$$

represents a connection between machines belonging to layers $\ell_1 \in L$ and $\ell_2 \in L$

Connections also have parameters that denote their capabilities:

- $x^{BW}_{\ell_1,\ell_2}$: average bandwidth
- $x^{BR}_{\ell_1,\ell_2}$: average bitrate

Problem statement

To run an application, the *application graph* needs to be deployed onto the *network graph*, that is, ***operators need to be put on machines.***

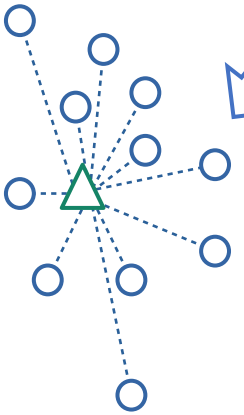
Application

- Minifi
- Edgent

Edge SPE

Cloud SPE

- Kafka
- Spark
- Flink
- Storm
- Sanza
- Pulsar



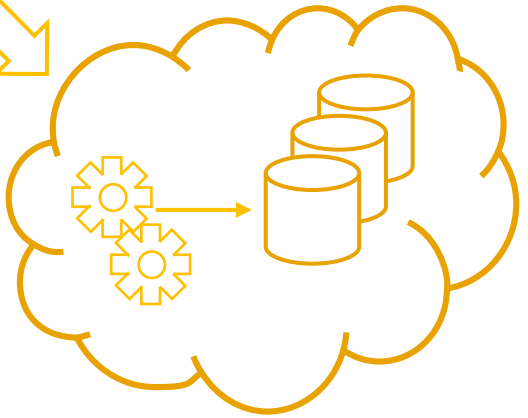
The Edge

Such mapping is called an ***execution plan***.

$$m: O \rightarrow H$$

hence

$$m_o = h \; for \; o \in O \; and \; h \in H, o^{MEM} \leq h^{MEM}$$



The Cloud

For the sake of clarity, m_o is used instead of $m(o)$

Data model

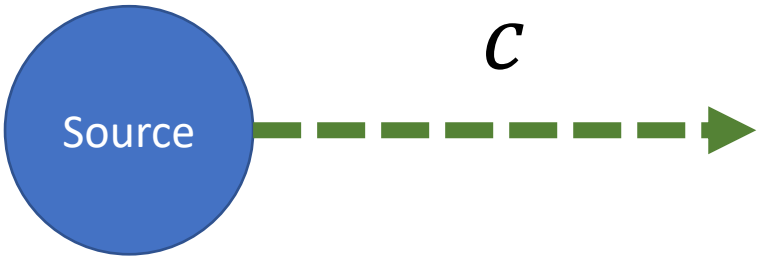
Throughput

The set of all data items is denoted by D, and D_i denotes the set of data items produced by operator $i \in P/E$

As with operators and machines, data items also have parameters:

- d_i^{SIZE}: the size of the item
- d_i^{MOP}: computing power needed to process it

Throughput: amount of items produced per unit of time by an operator

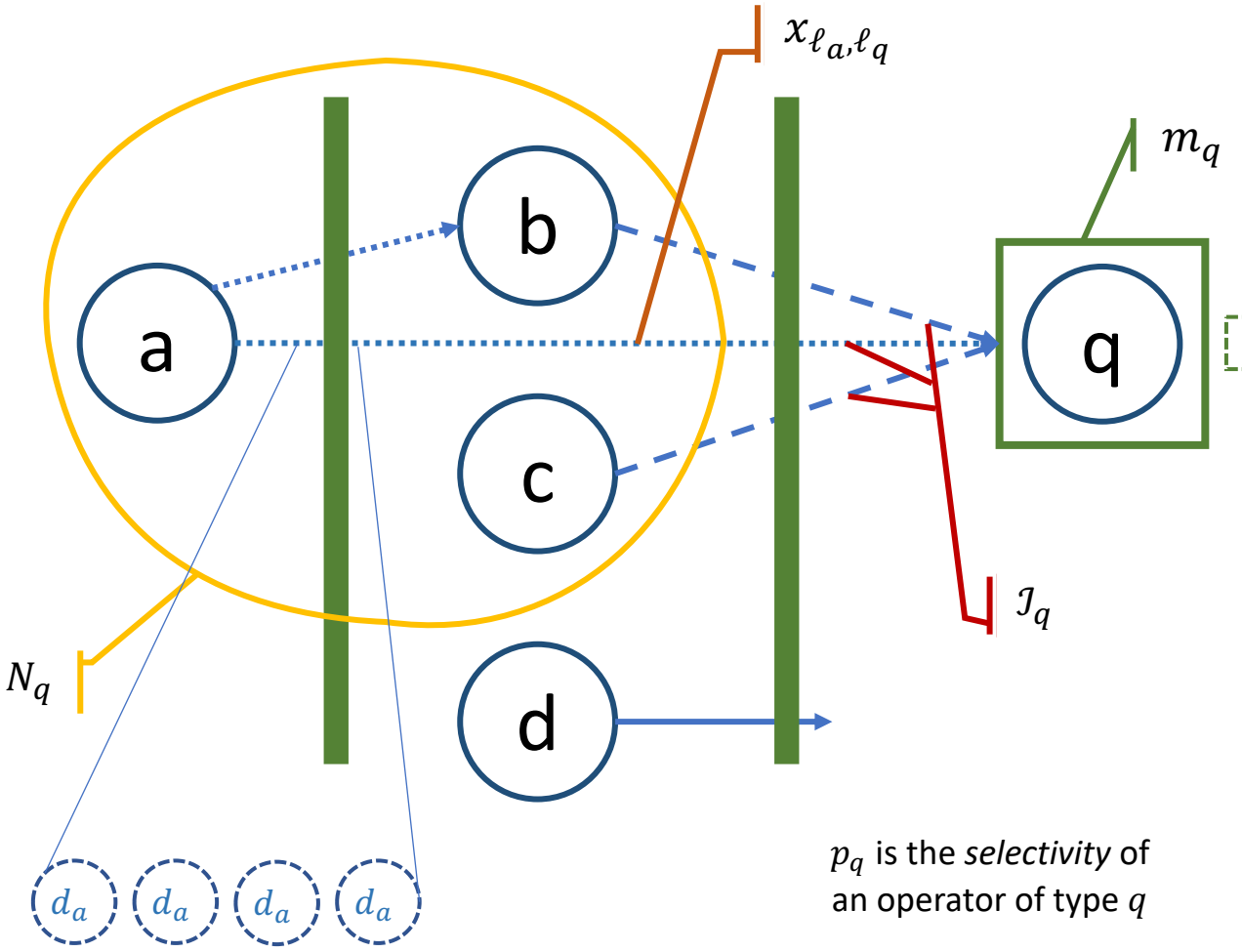


For sources, the throughput is **constant**, and specific for each data source.

Data Model

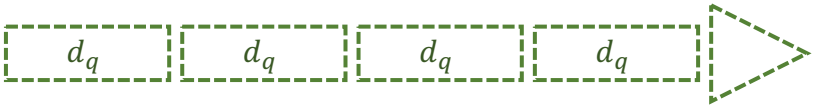
Throughput

For processing operators, throughput depends on the amount of data that *arrives* at the operator, and its processing power *(how much data I receive, and how fast I can manage it).*



N_q represents the set of operators connected to an operator q

$$N_q = \{a, b, c\}$$



$\mathcal{I}_q$ represents the amount of data arriving in operator q

$$\mathcal{I}_q = \sum_{n \in N_q} \left(x^{BR}_{\ell_n,\ell_q} \cdot \frac{1}{T_n \cdot d_n^{SIZ}} \right)$$

p_q is the *selectivity* of an operator of type q

The throughput of an operator q is defined as

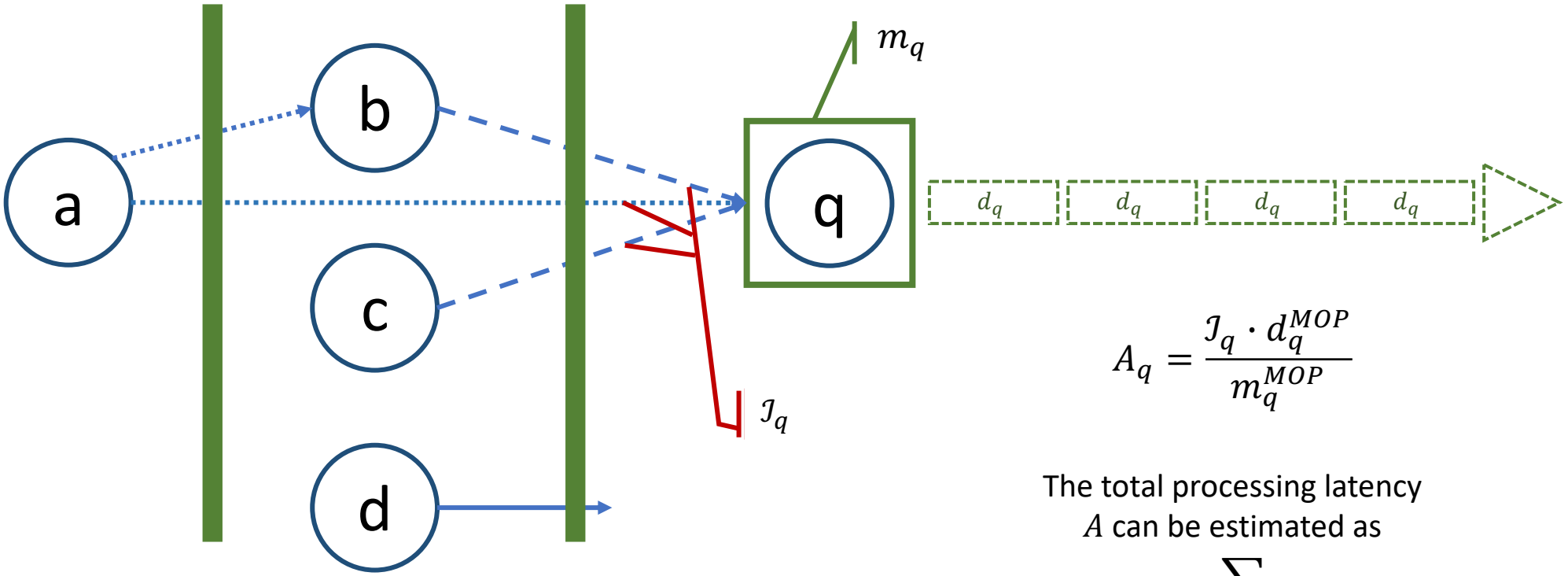
$$T_q = \frac{m_q^{MOP}}{\mathcal{I}_q \cdot d_q^{MOP}} \cdot \frac{1}{|N_q|} \cdot p_q$$

Data Model

Latency

Processing latency: time that a processing operator takes to process a data item.

The processing latency A_q of a processing operator $q \in O$ depends on the amount of data ítems d_q arriving at q, the amount of processing necessary to transform d_q, and the processing capacity of the machine m_q hosting q.



$$A_q = \frac{\mathcal{I}_q \cdot d_q^{MOP}}{m_q^{MOP}}$$

The total processing latency A can be estimated as

$$A = \sum_{q \in O} A_q$$

Cost Model

The proposed algorithm takes into account up to **four** different types of costs: *network* (C^{NET}), *implantation* (C^{IMP}), energetic (C^{KW}) and *on-demand* (C^{OD}).

The **network cost** depends on the amount of data being sent on the network

$$C^{NET} = \sum_{i \in P \setminus \mathrm{E}} T_i \cdot d_i^{SIZE} \cdot x_{\ell_i,\ell_q}^{COST} \; \forall \, j \in P$$

The **implantation cost** is associated to the number of layers having machines hosting operators

$$C^{IMP} = \sum_{\ell \in L} \begin{cases} \ell^{IMP} \; if \; \exists \ell_o = \ell \; for \; o \in O^1 \\ 0 \qquad\qquad otherwise \end{cases}$$

The **energetic cost** depends on the processing load of hosting machines (i.e. the rate between available and needed processing capacity)

$$C^{KW} = \sum_{o \in O} \left(\frac{m_o^{MOP}}{\mathcal{I}_o \cdot d_o^{MOP}} \cdot m_o^{KW} \right)$$

The **on demand cost** is relative to hosting machine renting cost, or any periodic cost in the case where the infrastructure is private

$$C^{OD} = \sum_{o \in O} m_o^{COST}$$

The total cost C is simply the sum of previous costs

$$C = C^{NET} + C^{IMP} + C^{KW} + C^{OD}$$

[1]: **Only processing operators are used, since sources and sinks are considered to be already deployed and thus, there is no implantation cost deriving from them.**

Proposed solution

CIPA: Complex Infrastructure Placing Algorithm

Input: An application graph $A = (P, L)$ and an infrastructure graph $G = (H, X)$.
Output: An execution plan, represented as a function $m: O \subset P \rightarrow H$

Simulated annealing(initial_solution, temp_max, temp_min, step_max): solution

```
temp ← temp_max
solution ← initial_solution
while temp > temp_min do:
        step ← 0
        while step < step_max
                step ← step + 1
                new_solution ← generate_solution()
                delta_energy ← energy(new_solution)-energy(solution)
                if delta_energy <= 0 then:
                        solution ← new_solution
                end if
        end while
        temp ← update(temp)
end while
return solution
```

Proposed solution

n.next() returns the nodes depending on data from n

$$a.next = \{b, q\}$$

$$c.next = \{q\}$$

n.previous() returns the nodes to which n is dependant

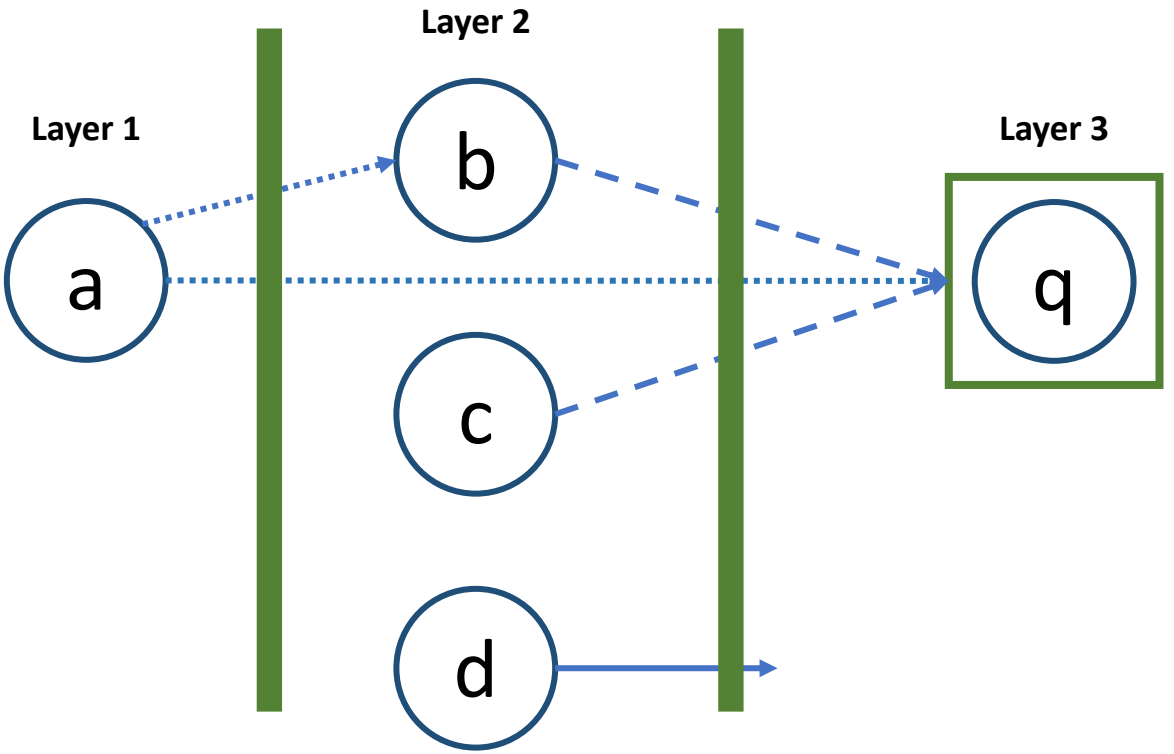
$$q.previous = \{a, b, c\}$$

$$b.next = \{a\}$$

n.level() returns the current layer level onto which n's hosting machine is deployed (null if undeployed)

$$q.level = 3$$

$$d.level = null$$



Proposed solution

CIPA: Complex Infrastructure Placing Algorithm

Input: An application graph $A = (P, L)$ and an infrastructure graph $G = (H, X)$.
Output: An execution plan, represented as a function m: $O \subset P \rightarrow H$

generate_solution(sinks $\subset P$): solution

```
node_queue ← [ e.previous() for e in E ]
while node_queue is not empty do:
        n ← node_queue.pop()
        if n ∈ S ∪ E then continue
        max_level ← ∞, min_level ← 0
        for m in n.next() do:
                max_level ← min(m.level, max_level)
        for p in n.previous() do:
                min_level ← max(p.level, min_level)
                node_queue.push(p)
        n.level ← random(min_level, max_level)
        bpp(p, machines_in_level(n.level))
```

Proposed solution

CIPA: Complex Infrastructure Placing Algorithm

Input: An application graph $A = (P, L)$ and an infrastructure graph $G = (H, X)$.
Output: An execution plan, represented as a function $m: O \subset P \rightarrow H$

bpp(node $\in O$, machines $\subset H$, tolerance): hosting_machine

```
filter_too_small_and_too_large(machines, tolerance)
sorted_machines ← sort_by_cost()

return sorted_machines.first()
```

Next steps

- Hardcode sample stream processing applications to use as tests
  - CCTV
  - Earthquake Early Warning System

- Setup the benchmark ideal environment and the placing found by the algorithm to run the experiments.
  - Automatically convert algorithm output to config files.
  - Metrics to measure: **throughput** and **latency**

- Run the experiments on Grid'5000 using a management tool like EnOSlib

- Analyze, discuss and write the paper with the results.