Security Level:

11 / 10 / 2017

Active objects for BSP (Work In Progress)

Parallel And Distributed Algorithms Lab

SCALE team

Pierre Leca

Context

1st year industrial PhD as French CIFRE contract between Huawei and I3S

Company supervisors : Gaetan Hains and Wijnand Suijlen
University supervisors : Ludovic Henrio and Eric Madelaine

Early work on mixing BSP with active objects

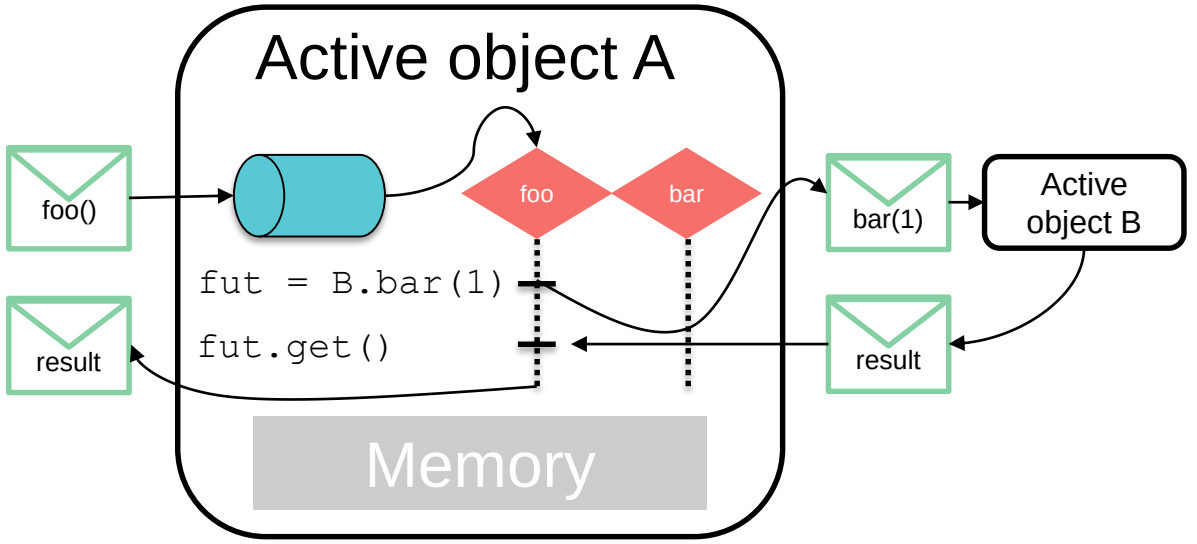
Content

- Active objects
- Bulk Synchronous Parallel
- BSP active objects
- Future work ideas

Content

- Active objects
- Bulk Synchronous Parallel
- BSP active objects
- Future work ideas

Active objects



Represent asynchronous entities living in their own thread

Object function call syntax for sending requests

Result represented as future returned immediately

Blocking access to future only when required (wait by necessity)

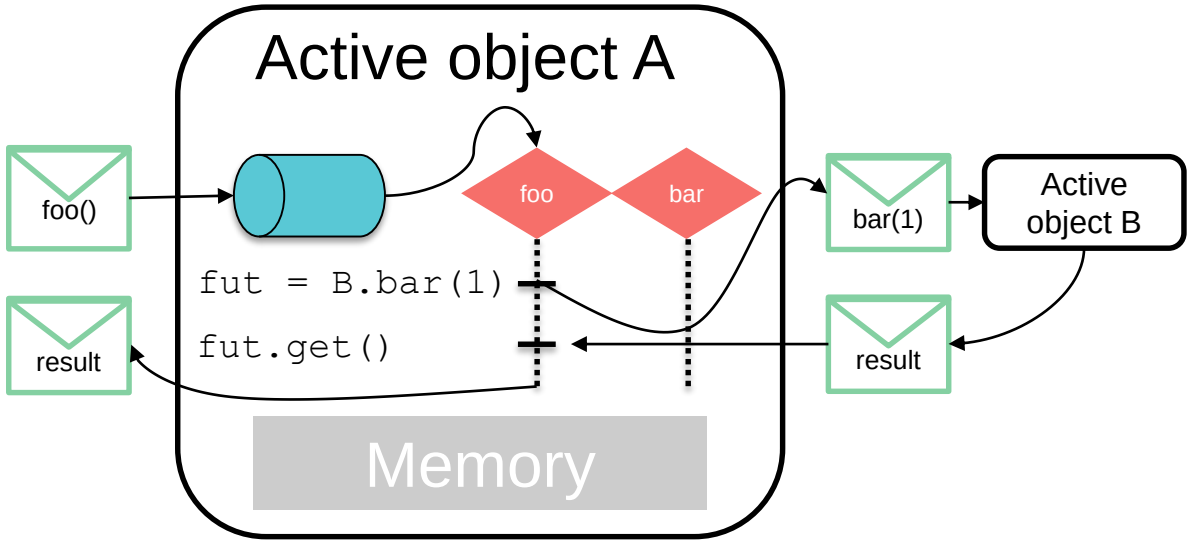
Active objects

SOA vision

Suited for task-parallel algorithms

One request served at a time

Partially deterministic execution



Content

- Active objects
- Bulk Synchronous Parallel
- BSP active objects
- Future work ideas

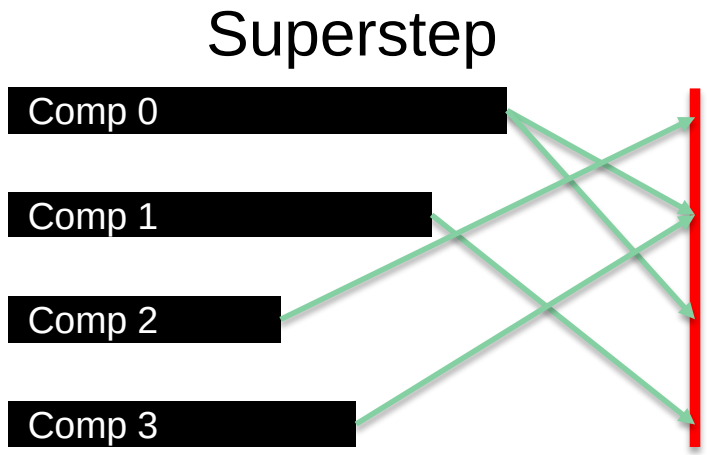
Bulk Synchronous Parallel

Parallel execution model

A program is a sequence of supersteps

Computation → communication → synchronization

Synchronization ensure alignment of supersteps



Bulk Synchronous Parallel

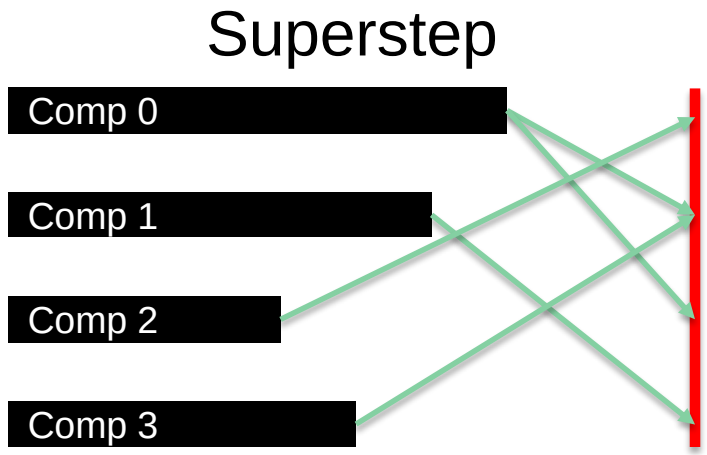
HPC vision

Deterministic execution

Easy to avoid deadlocks

Has a simple cost model

Suited for balanced data-parallel algorithms



Content

- Active objects
- Bulk Synchronous Parallel
- BSP active objects
- Future work ideas

BSP active objects : coordination

Active object model is a good match for task-parallelism

BSP is a good match for data-parallelism

Both are not adequate for the other kind of parallelism

Both have interesting properties

Our idea is to combine them into a single execution model

BSP active objects : execution model

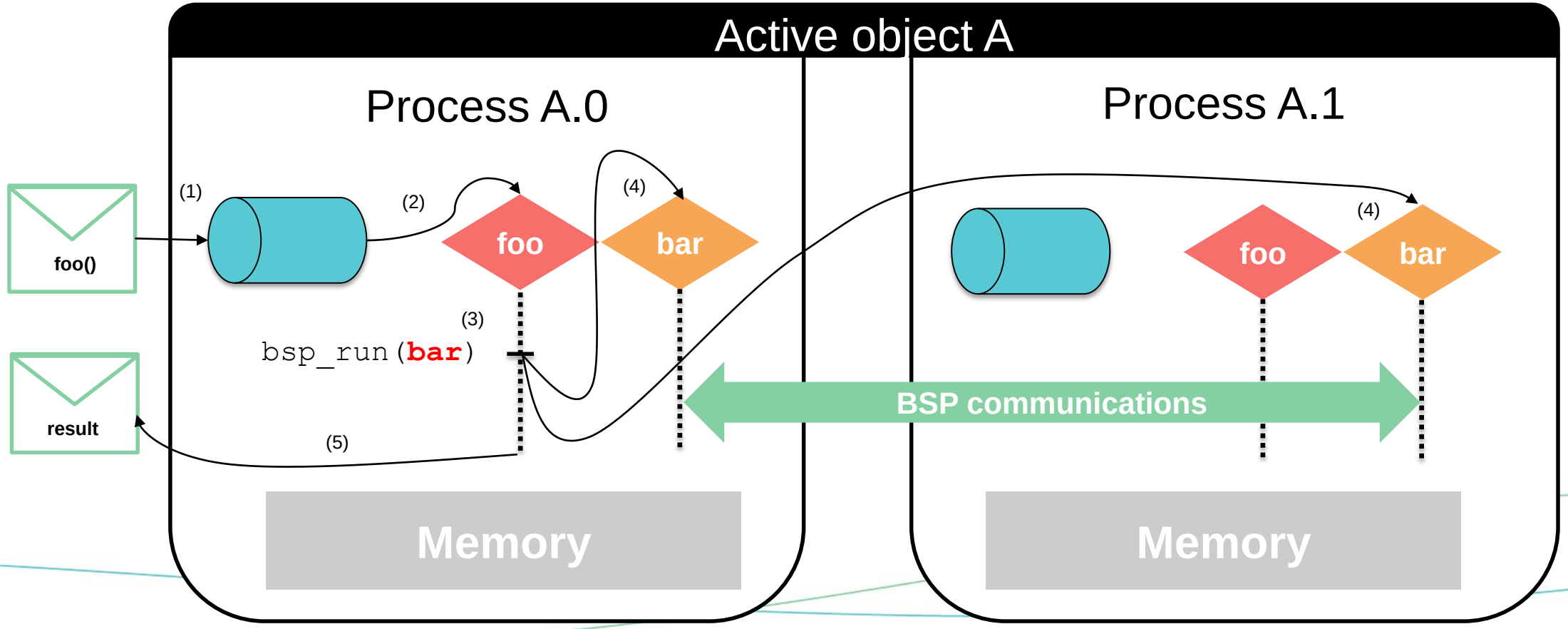
Multiple processes per active object

Requests handled sequentially by one head process

The head process can use other processes in BSP mode

BSP mode can be used for processing data-parallel parts

BSP active objects : execution model



BSP active objects : example

```
vector <int> foo (const vector <int> & v, int d)
{
    int blocksize = v.size() / bsp_nprocs();

    for (int i = 0; i < bsp_nprocs(); ++i)
    {
        bsp_send(i, &blocksize, sizeof(int));
        bsp_send(i, &v[i * blocksize], blocksize * sizeof(int));
        bsp_send(i, &d, sizeof(int));
    }

    bsp_run(bar);

    vector <int> res(v.size());

    for (int i = 0; i < bsp_nprocs(); ++i)
    {
        bsp_recv(i, &res[i * blocksize], blocksize * sizeof(int));
    }

    return res;
}
```

```
void bar()
{
    bsp_sync();

    int blocksize, d;
    bsp_recv(0, &blocksize, sizeof(int));

    int v[blocksize];

    bsp_recv(0, v, blocksize * sizeof(int));
    bsp_recv(0, &d, sizeof(int));

    for (int i = 0; i < blocksize; ++i)
    {
        v[i] += d;
    }

    bsp_send(0, v, blocksize * sizeof(int));

    bsp_sync();
}
```

BSP active objects : pros

Programming sequentially by default is easy

Possible to express multi-parallelism algorithms

Task and Data parallel with deterministic properties

Implemented algorithms are easy to reuse

BSP active objects : cons

Active objects always communicate results

Distributed data are always aggregated to a single process

May be too complex for straightforward data parallelism

BSP active objects : implementation choices

Combine into a single language : C++

Build on top of MPI

Active objects use existing processes (vs dynamic creation)

Content

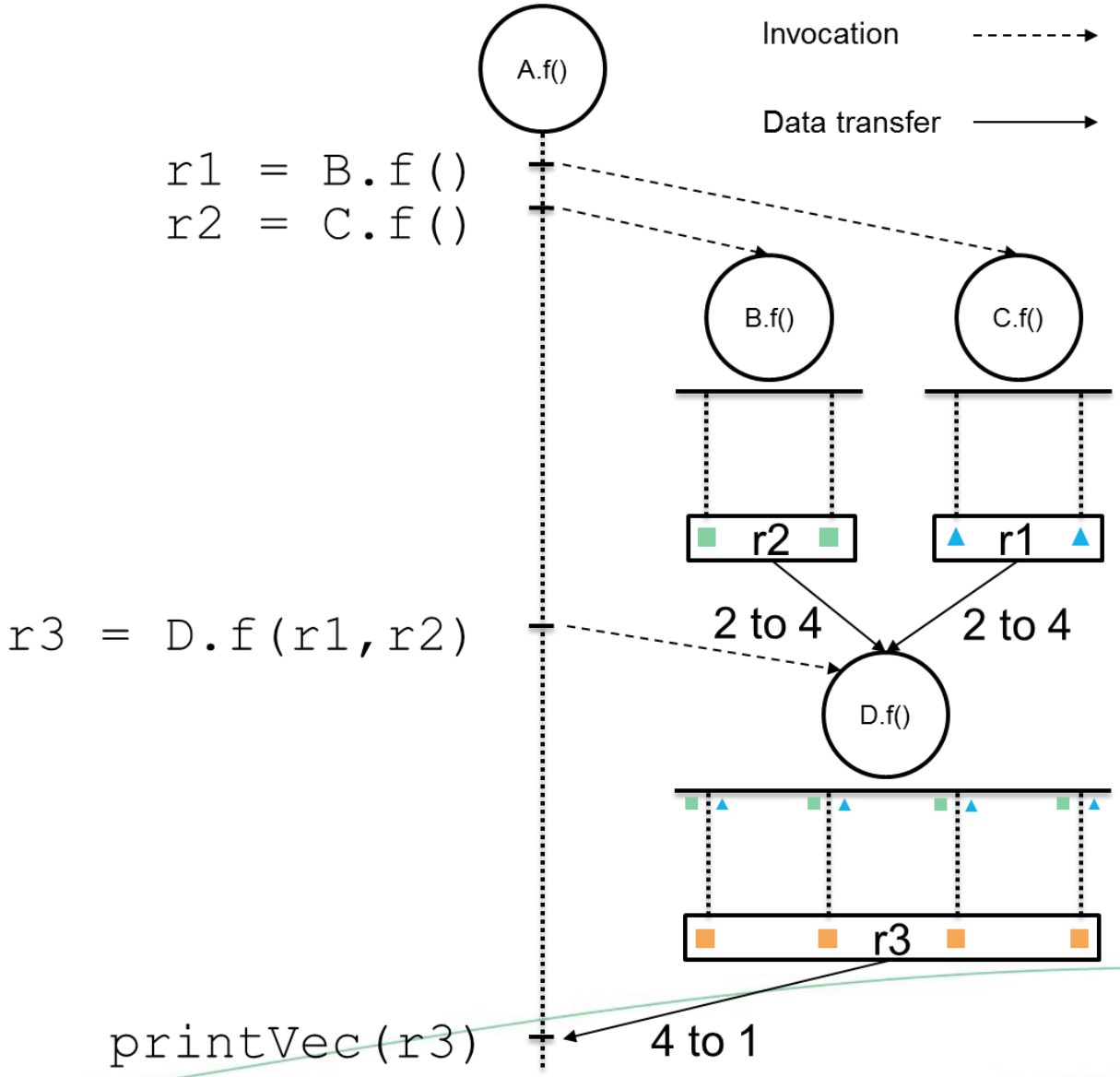
- Active objects
- Bulk Synchronous Parallel
- BSP active objects
- Future work ideas

Future work ideas

Combine distributed vectors and lazy future update strategy

Generic communication primitives between subsets

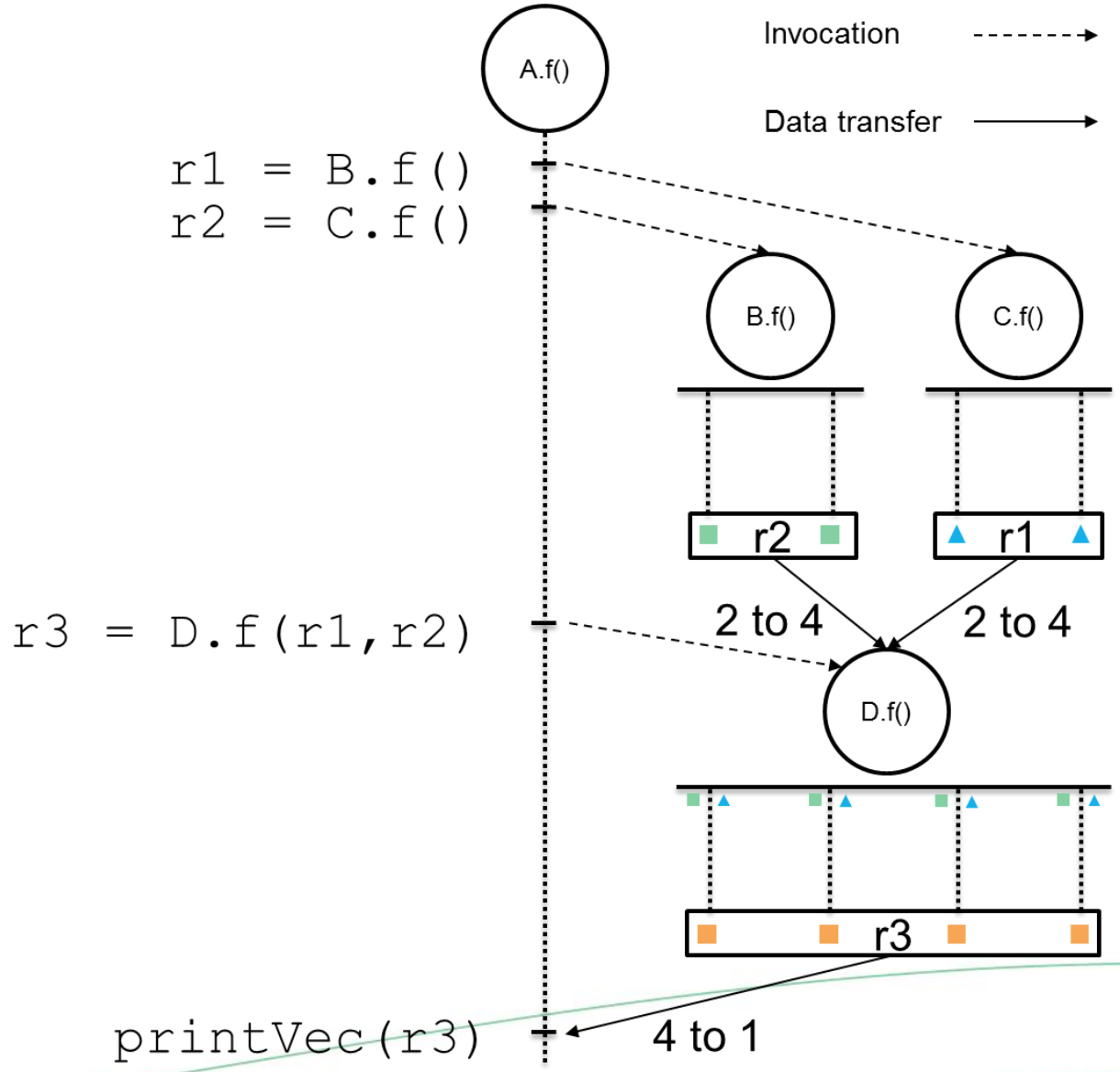
Process sharing



Future work ideas

Formalization

Study of cost model



Thank You.