Introduction to FastFlow programming

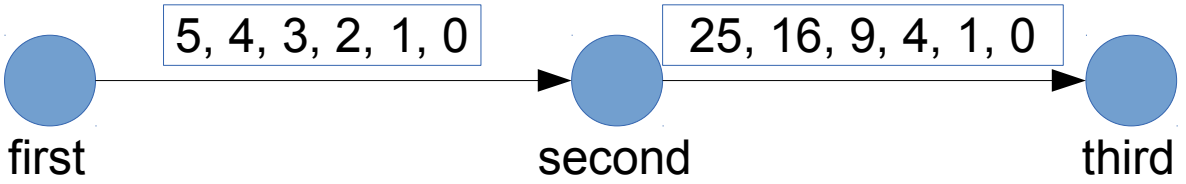
SPM lecture, November 2016

Massimo Torquati <torquati@di.unipi.it>

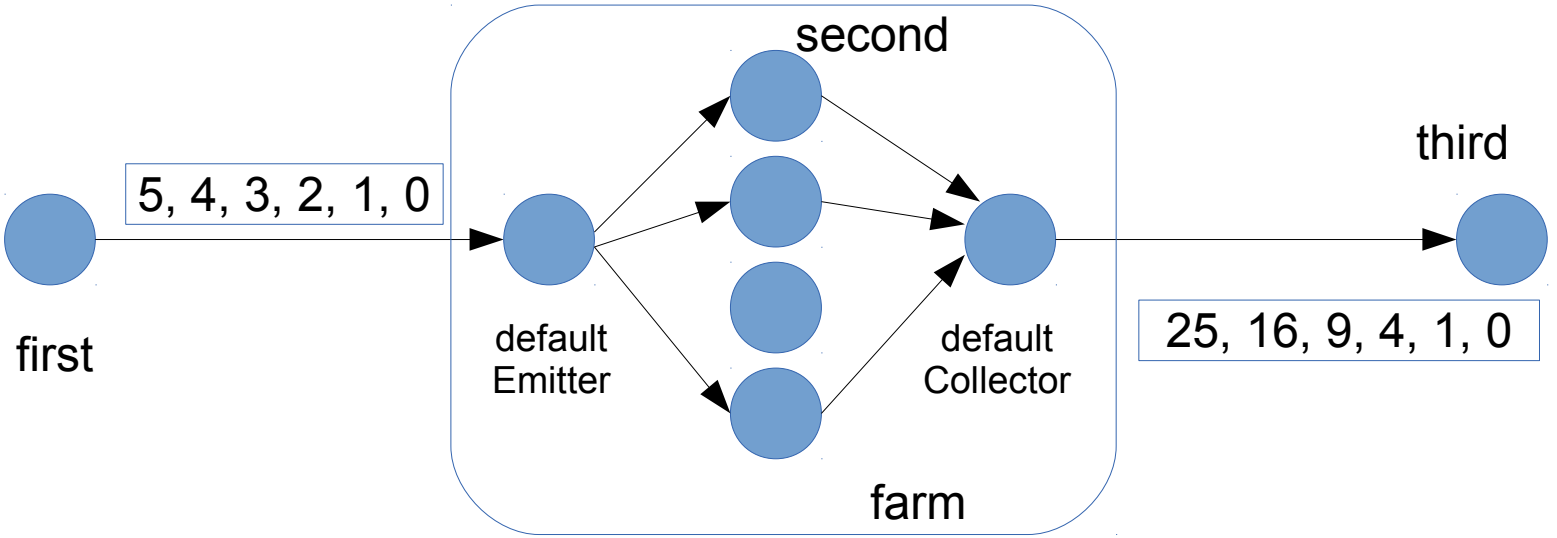
Computer Science Department, University of Pisa - Italy

FastFlow farm

- Let's consider again the ClassWork1: pipe(seq, seq, seq)



- 3-stage pipeline: pipe(seq, farm, seq)

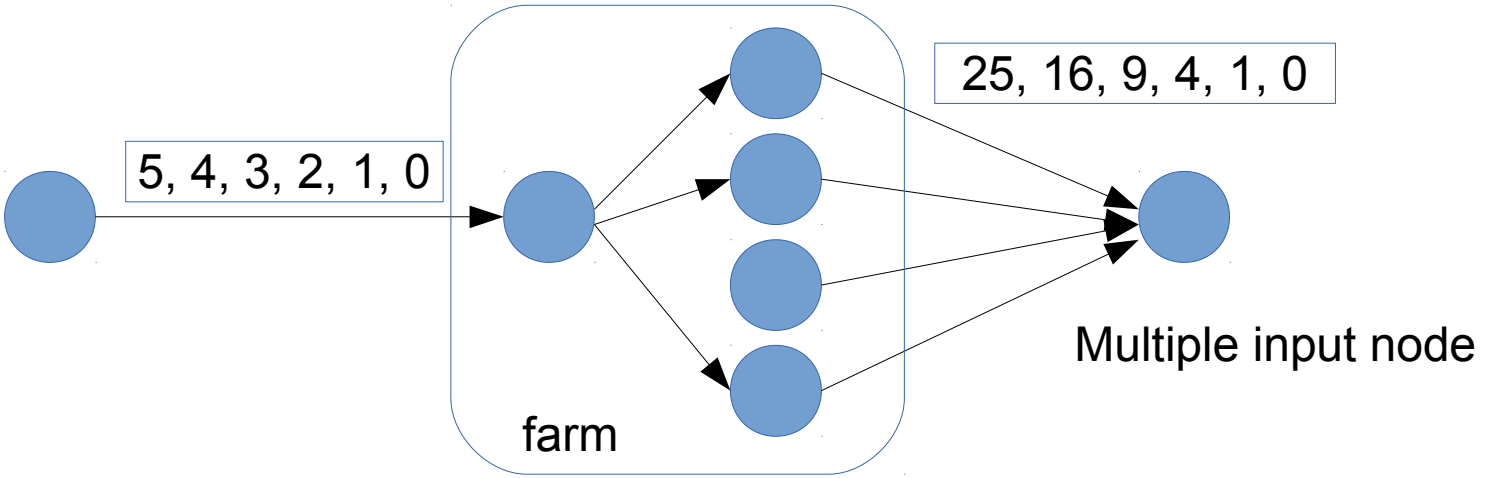


- Default task scheduling is (*pseudo) round-robin*
- The task collection in the Collector thread is "from any" (input non-determinism)

- See the farm_square1.cpp file in the ClassWork2 folder

ClassWork2: comments

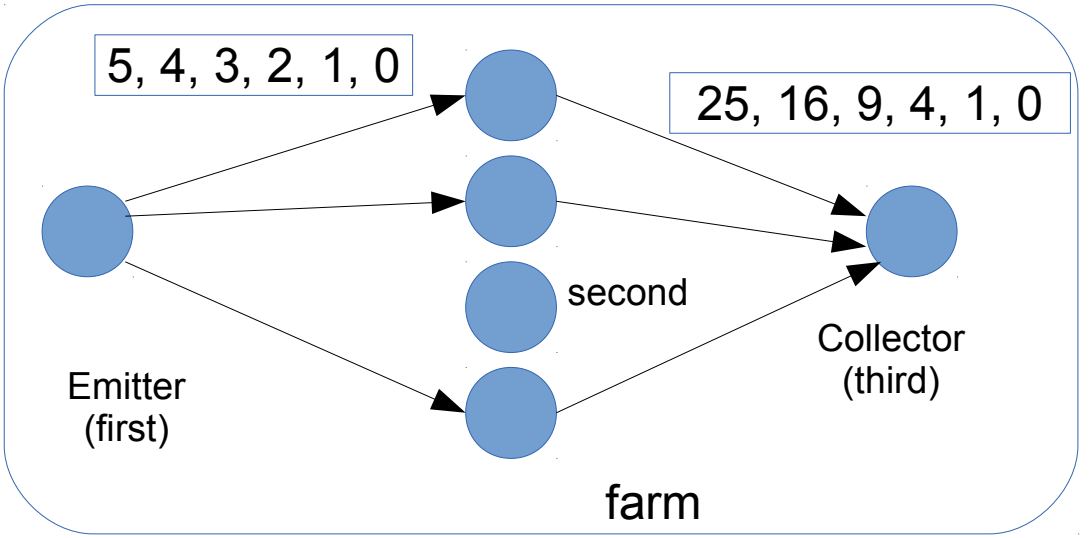
- 3-stage pipeline: pipe(seq, farm, seq)
- The farm does not have the collector node
- The third stage of the pipeline is a multi-input node (ff_minode_t)



- The Collector can be removed using:
  - myFarm.remove_collector();
  - If the next stage after the farm is a sequential node, it must be defined as
  - *ff_minode_t* (multi-input node)
- See the farm_square2.cpp file in the ClassWork2 folder

FastFlow farm (classWork2 comments)

- single farm with specialized Emitter and Collector: farm(seq, nw)



- The farm collector by default acts as a multi input node
- The farm emitter by default acts as a multi output node

- See the farm_square3.cpp file in the ClassWork2 folder

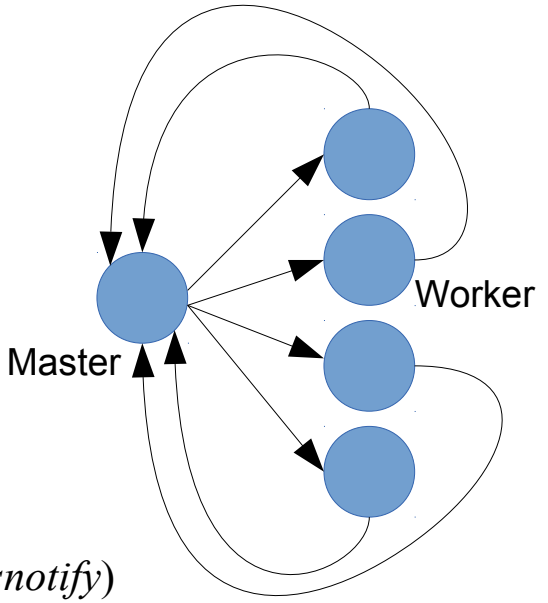
Ordered farm *ff_ofarm*

- Provides a total ordering between input and output
  - use case example: video streaming
- Limitations:
  - The number of tasks produced in output by the workers must be exactly the same of the number of tasks received in input
  - It is not possible to define your own scheduling and gathering policies
- If you don't need a strict input/output ordering then it is generally better to implement your own policy by re-defining the Emitter and the Collector

- Considering again the ClassWork2, try to replace the ff_Farm with the ff_OFarm in all examples (pay attention to the ff_OFarm class interface for the farm_square3.cpp version)

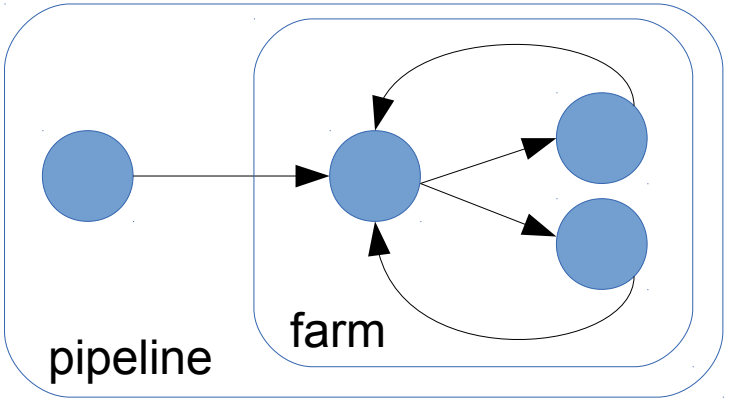
More on the *ff_farm*

- Auto-scheduling:
  - myFarm.set_scheduling_ondemand(<optional-value>)
- Possibility to implement user's specific scheduling strategies (ff_send_out_to)
  - *ff_send_out_to.cpp* example in the tutorial tests
- Master-Worker computation:
  - farm without the collector node
  - Workers send the results back to the Emitter



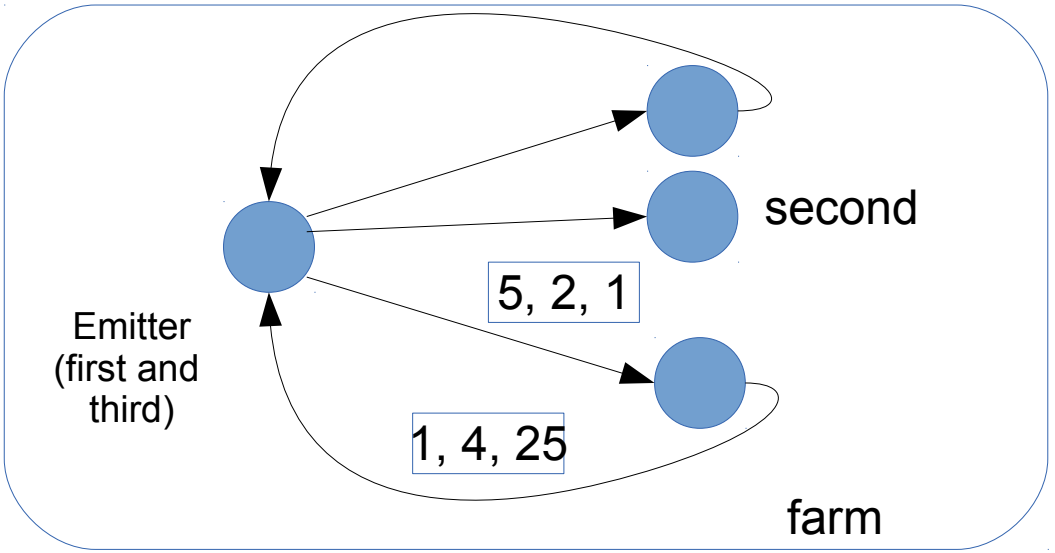
master-worker skeleton

- Let's see the *feedback.cpp* example in the tutorial tests
  - The termination protocol is a bit more complex (you need to use *eosnotify*)



FastFlow farm (again classWork2)

- Master-worker version:



- See the farm_square4.cpp file in the ClassWork2 folder

Introduction to Class Work 3: using ff_Pipe and ff_Farm

- Simple file compressor using *miniz.c*:
  - The sequential implementation of the compressor is given (*simplecomp.cpp*) together with an utility program for decompressing the files (*compdecomp.cpp*).
  - The task is to modify the sequential code and implement a 3-stage pipeline version in which the first stage reads from the command line a list of files to compress, the second stage compresses each input file in memory and finally the third stage writes the compressed memory file into the disk (in a separate folder).
    - icc simplecomp.cpp -o simplecomp
    - To decompress a file use the compdecomp program (first you have to compile the compdecomp.cpp file):
      - *./compdecomp d <compressed-file>*
  - Then implement the second stage by using an ff_Farm
  - All files needed are in the **~spm1501/public/ClassWork3** folder of the course machine