A Hybrid Dataflow Visual Programing Language

- Hong-Xin Wang

College of Computer Science and Software Engineering
Shenzhen University
Shenzhen, China

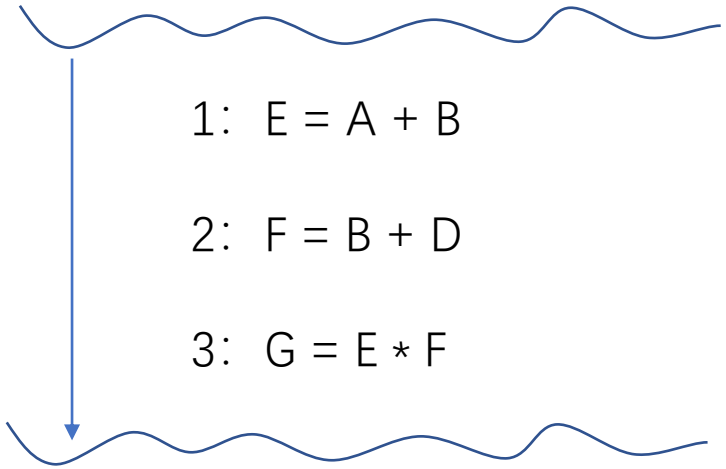
Content

- Background and Motivation
- Hybrid Dataflow Language
- Dataflow Visual Programing

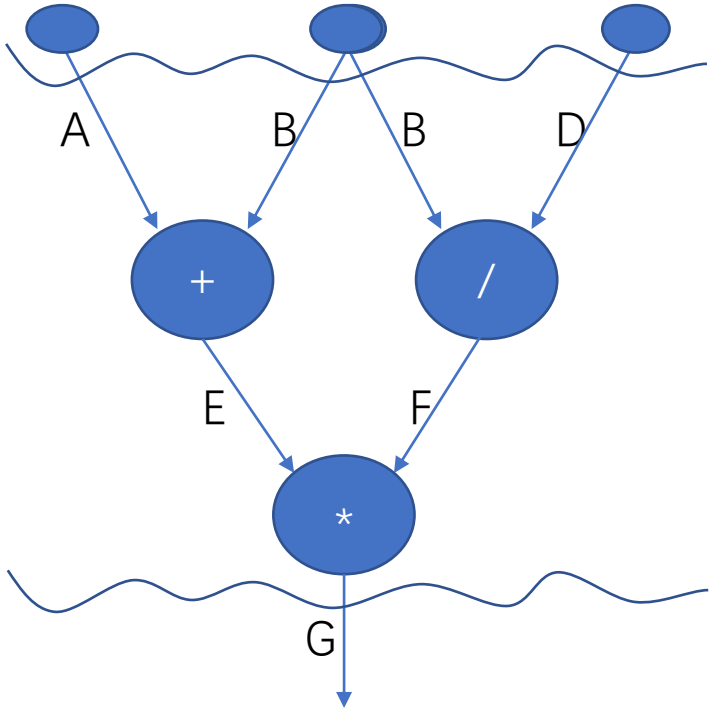
Background and Motivation

- Moore's Law gradually is failing
  - Single-core performance is close to its limit
  - Multi-core parallel acceleration
- Traditional Parallel Programming is hard
  - Synchronization tool
  - Dead lock problem

Data flow programming



sequential logical dependencies



concurrent data dependencies

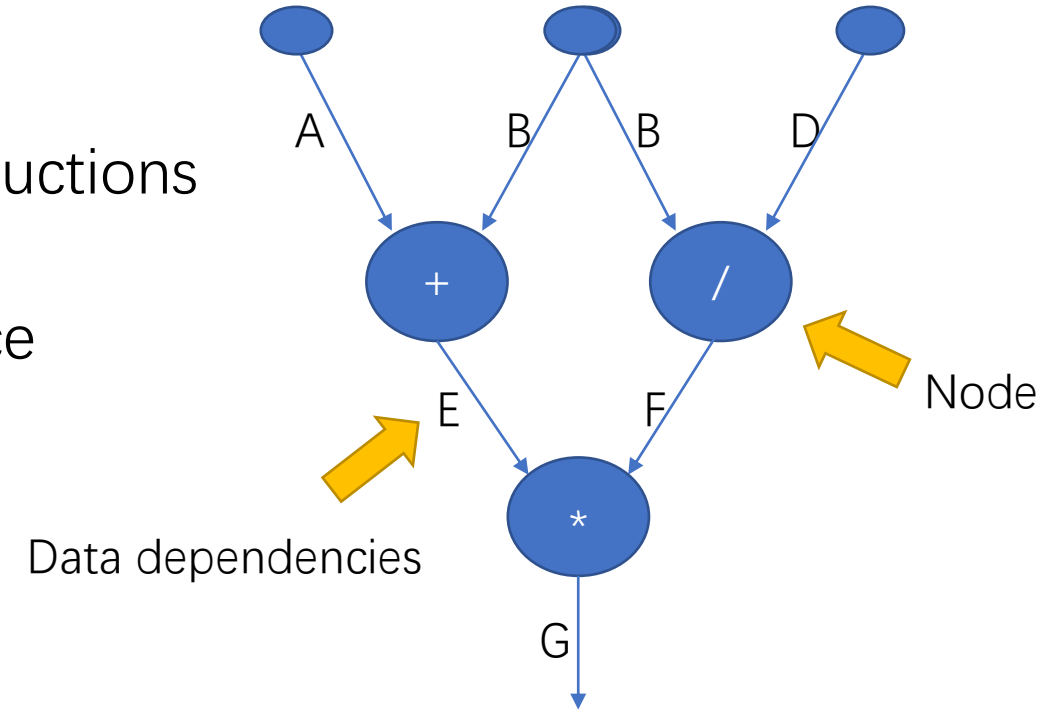
concurrency in E and F

Control flow vs Data flow

	Control flow programming	Data flow programming
Instruction level parallelism	automatic	automatic
Data level parallelism	manual	automatic
Thread level parallelism	manual	automatic

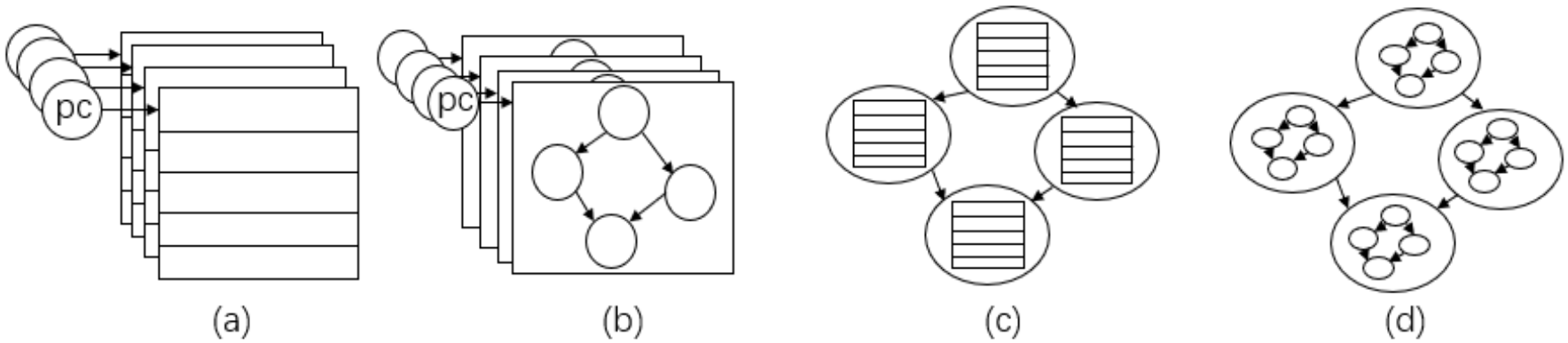
Hybrid Data flow Language

- The Pure Dataflow Model*
  - The nodes of the graph are primitive instructions
  - hard to handle complex data structure
  - failed to deliver the promised performance
- Hybrid Data flow Model



* Yazdanpanah, Fahimeh, et al. "Hybrid dataflow/von-Neumann architectures." IEEE Transactions on Parallel and Distributed Systems 25.6 (2013): 1489-1509.

Way of Hybrid Data flow with Control flow*

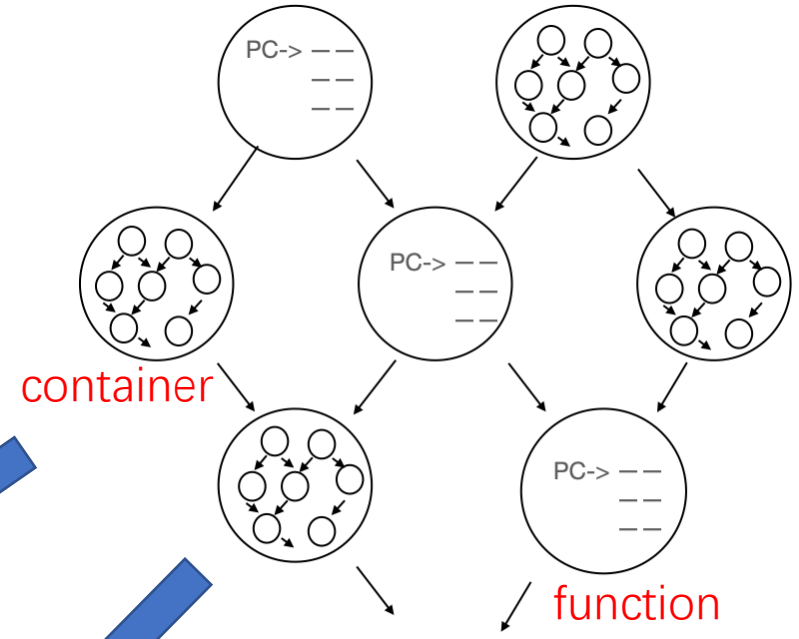


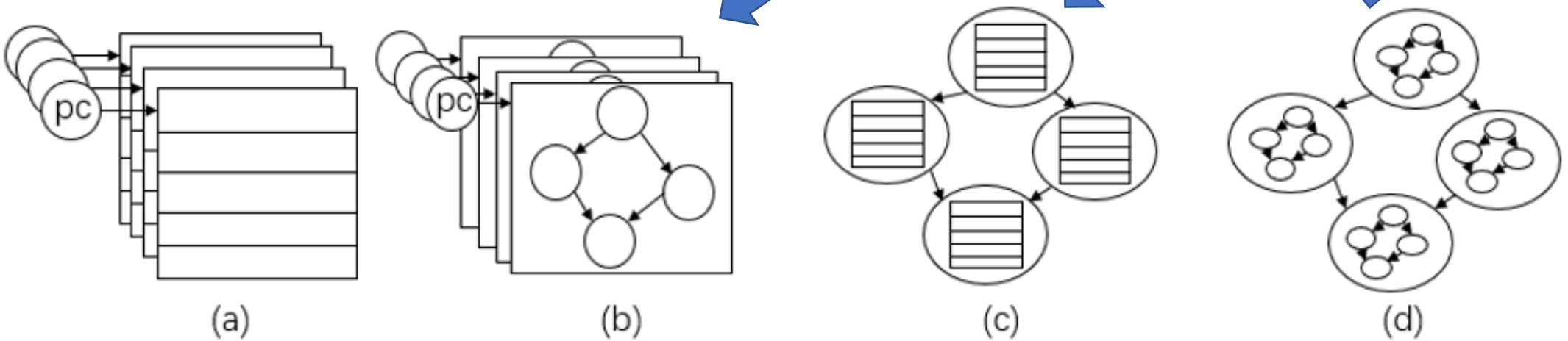
Enhanced control flow | control flow/dataflow | dataflow/control flow | enhanced dataflow

* Yazdanpanah, Fahimeh, et al. "Hybrid dataflow/von-Neumann architectures." IEEE Transactions on Parallel and Distributed Systems 25.6 (2013): 1489-1509.

Our Hybrid Execution Model

- The nodes of graph are container or function
- can naturally represent the latter three

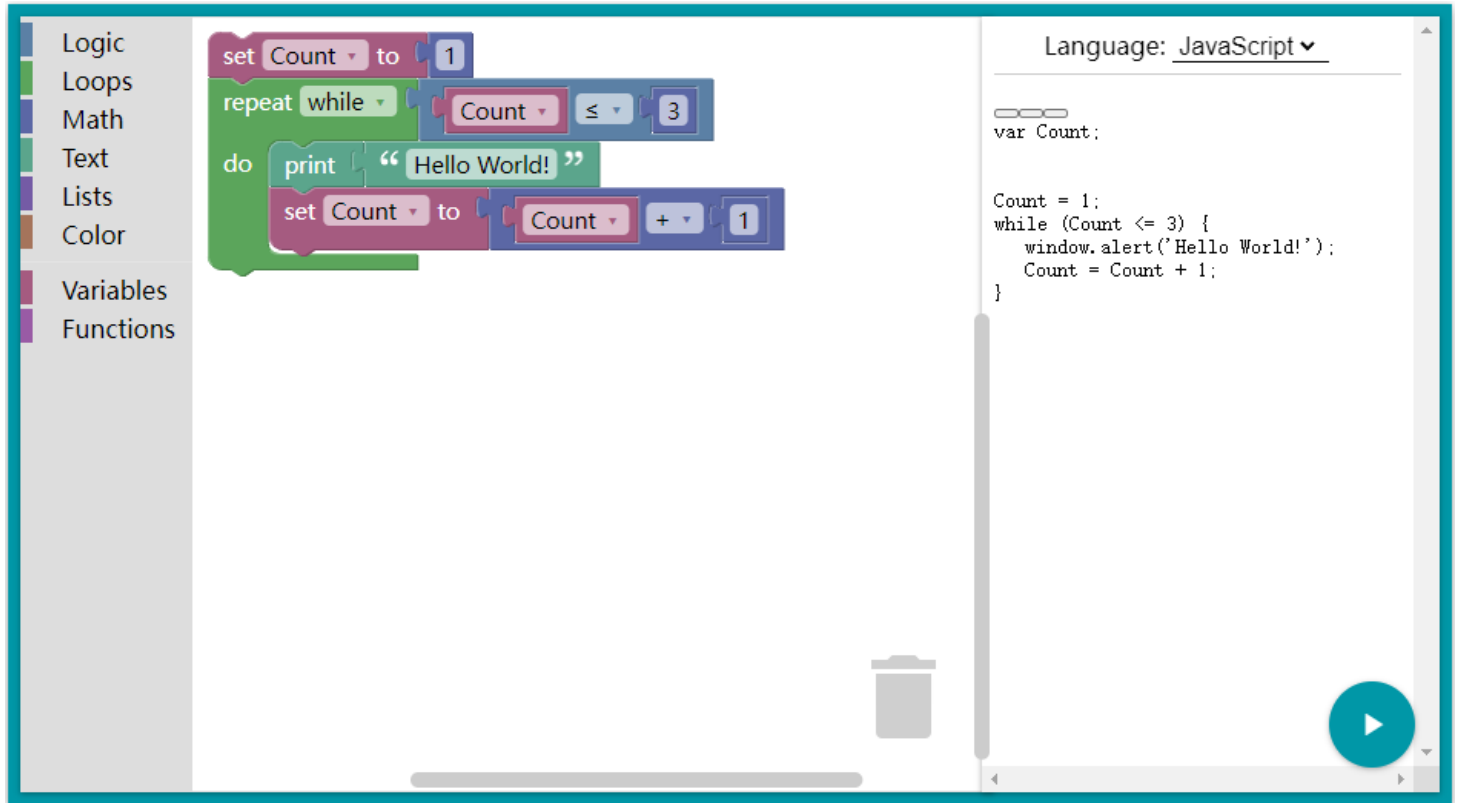




Enhanced control flow | control flow/dataflow | dataflow/control flow | enhanced dataflow

Today Popular Visual Programing

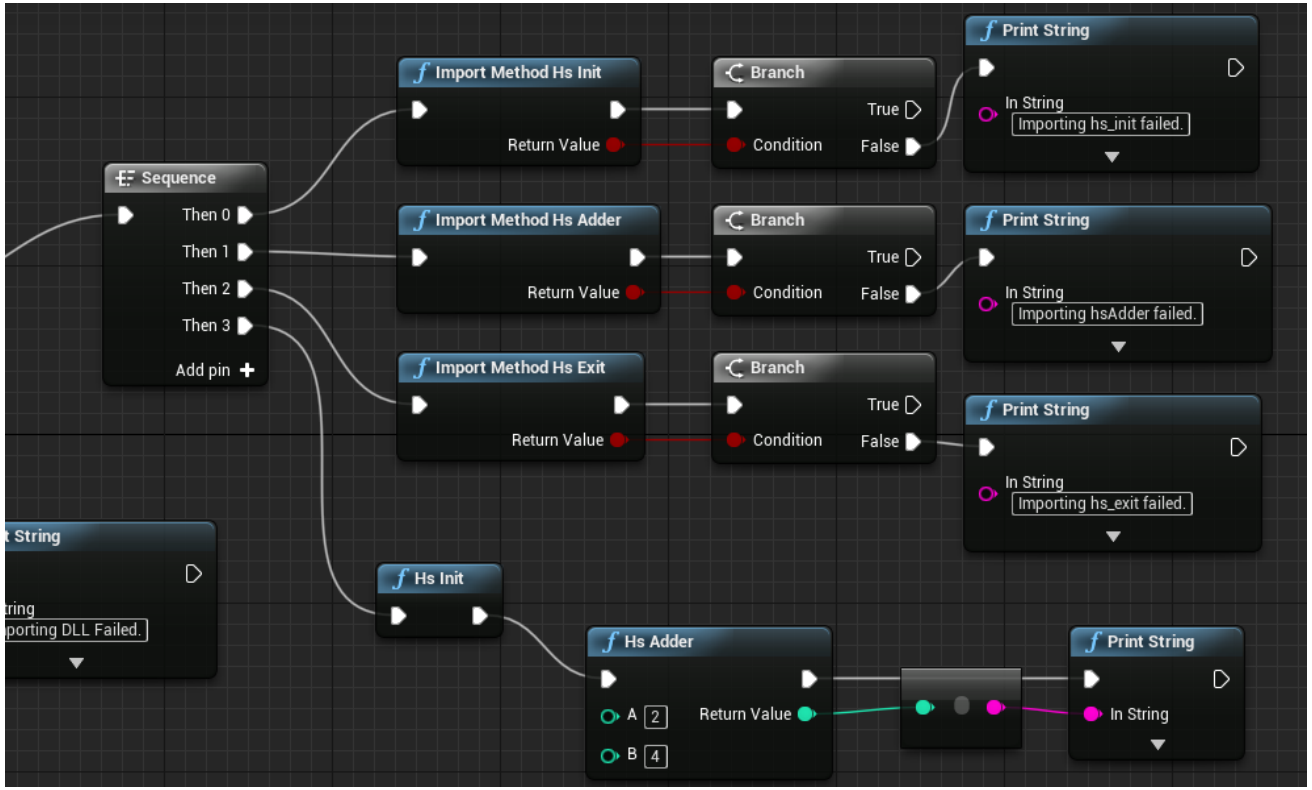
- Block-based
- Control flow model



Google's Blockly

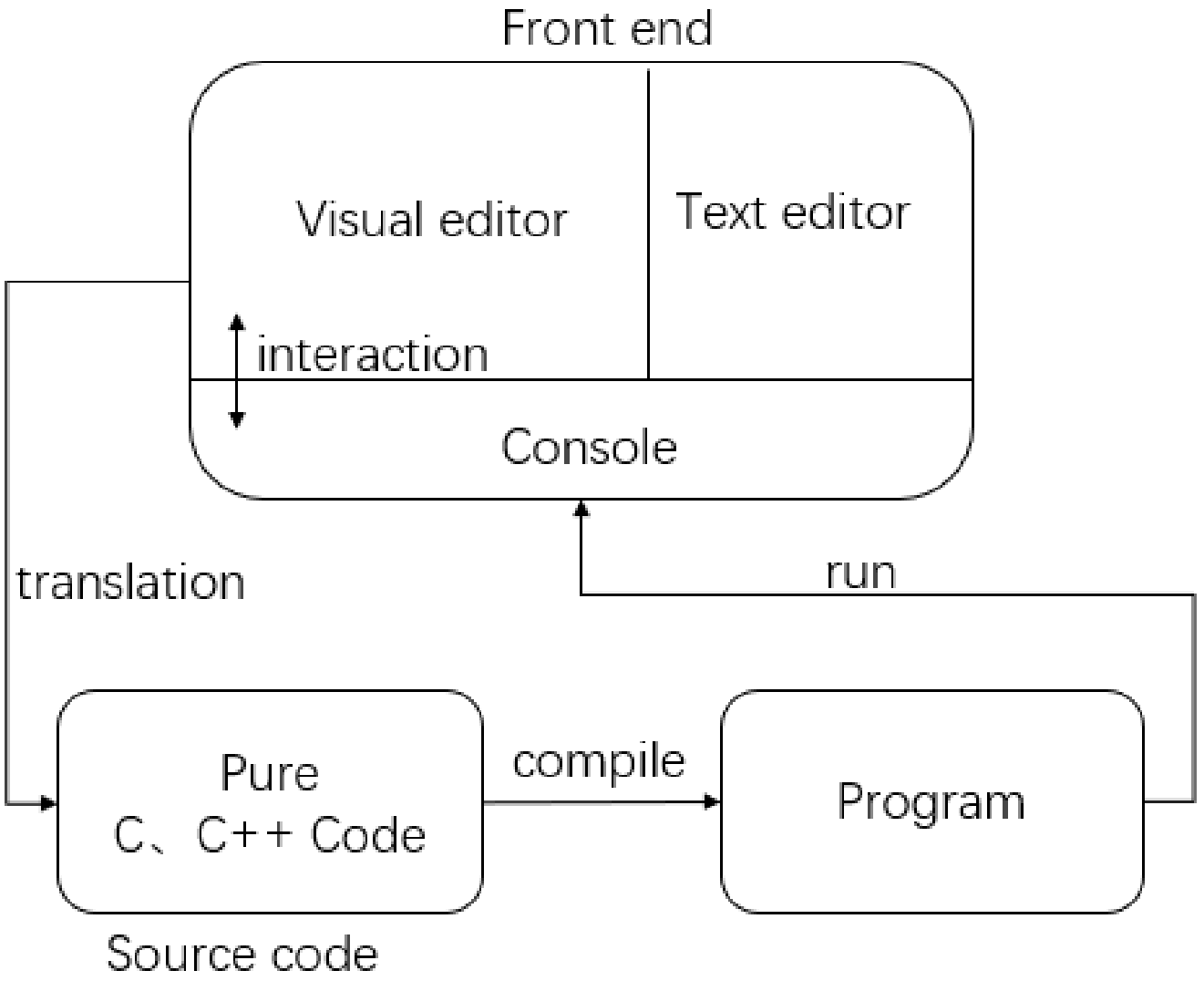
Today Popular Visual Programing

- Flow-based
- Dataflow model
- Function Fixed node
- Not for general-purpose programming



Unreal engine blueprint

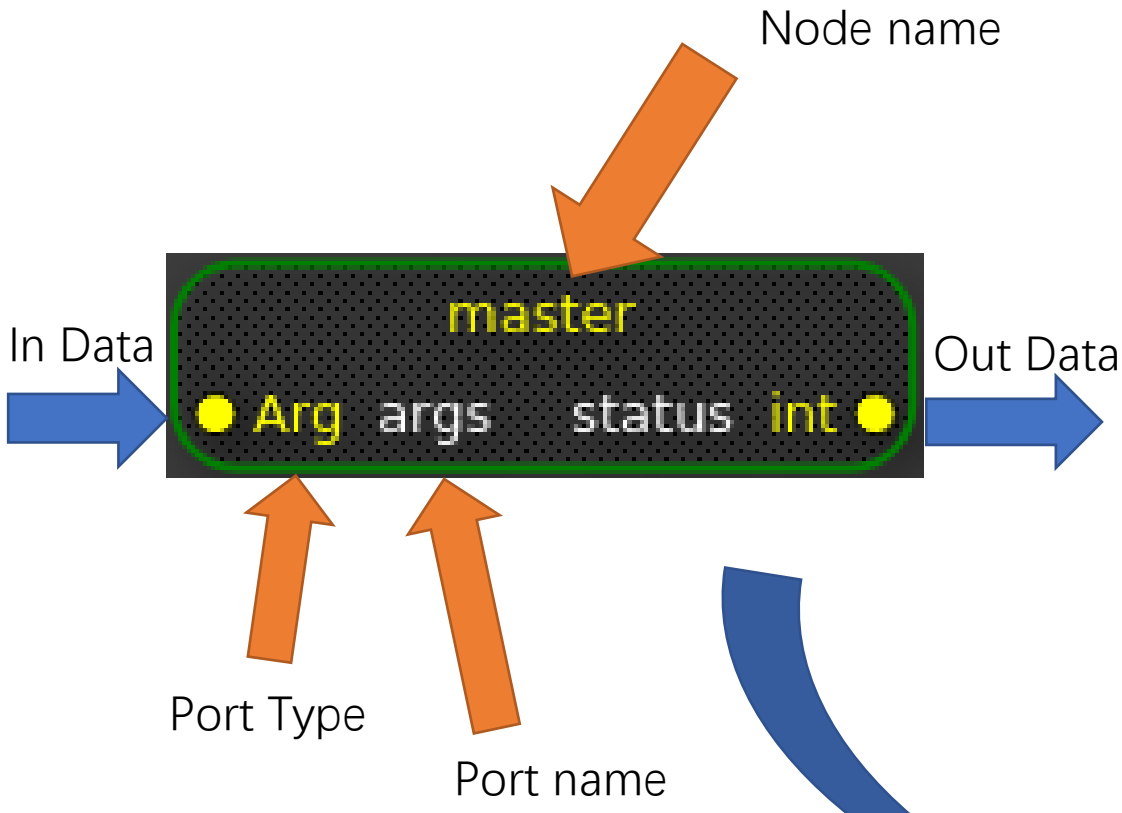
Our Graphical User Interface



Data flow Visual Programing

1. Define data structure for Node's Data transfer
2. Create function and container node and its port
3. Connect port
4. Programming function node

Main and container Node

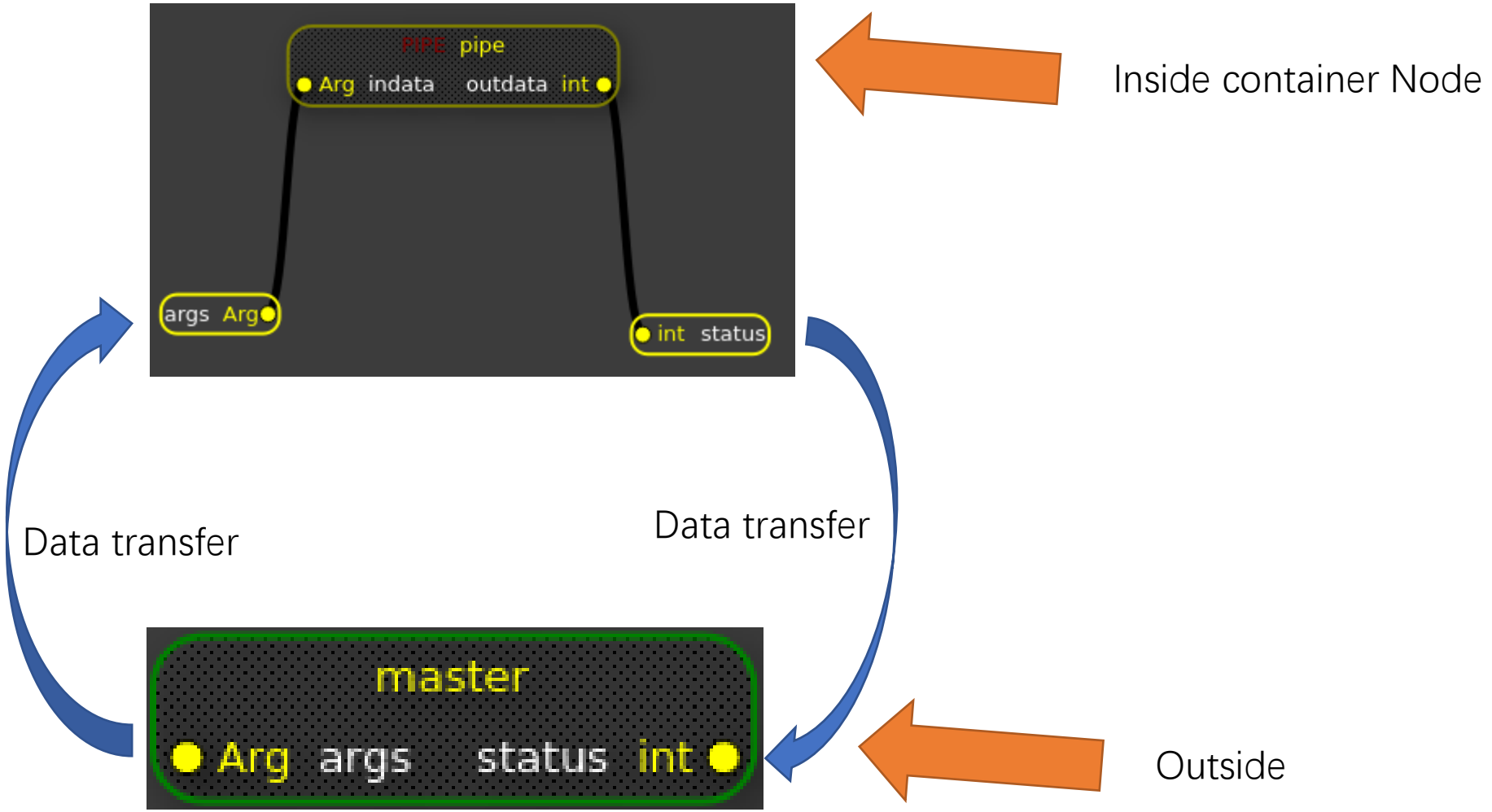


Defined C data structure

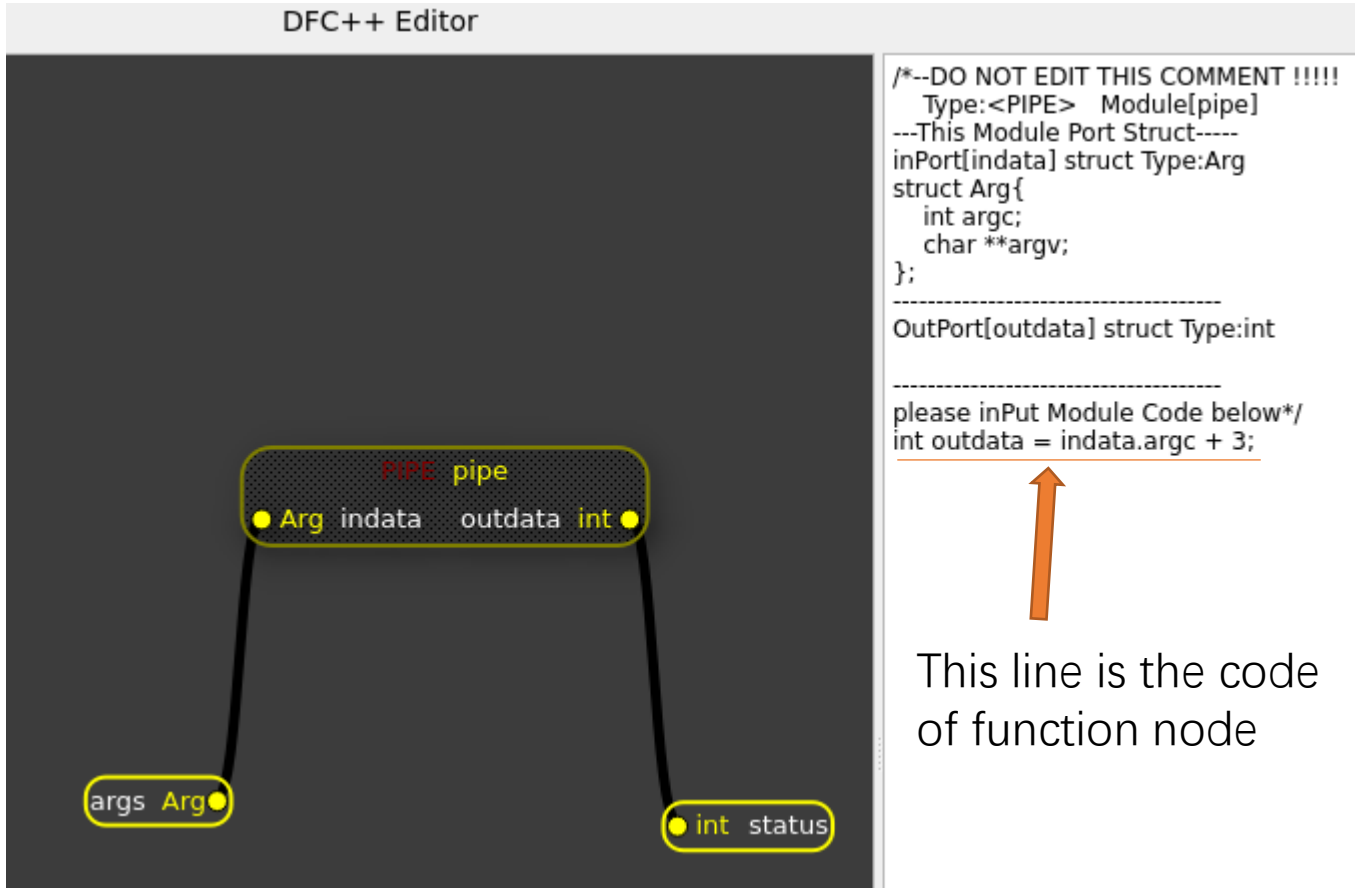
```c
struct Arg{
    int argc;
    char ** argv;
};

int main(int argc, char** argv){
    ...
    ...
    ...
    return status;
}
```

It would execute like this

Sub DAG and function Node



Function node programing

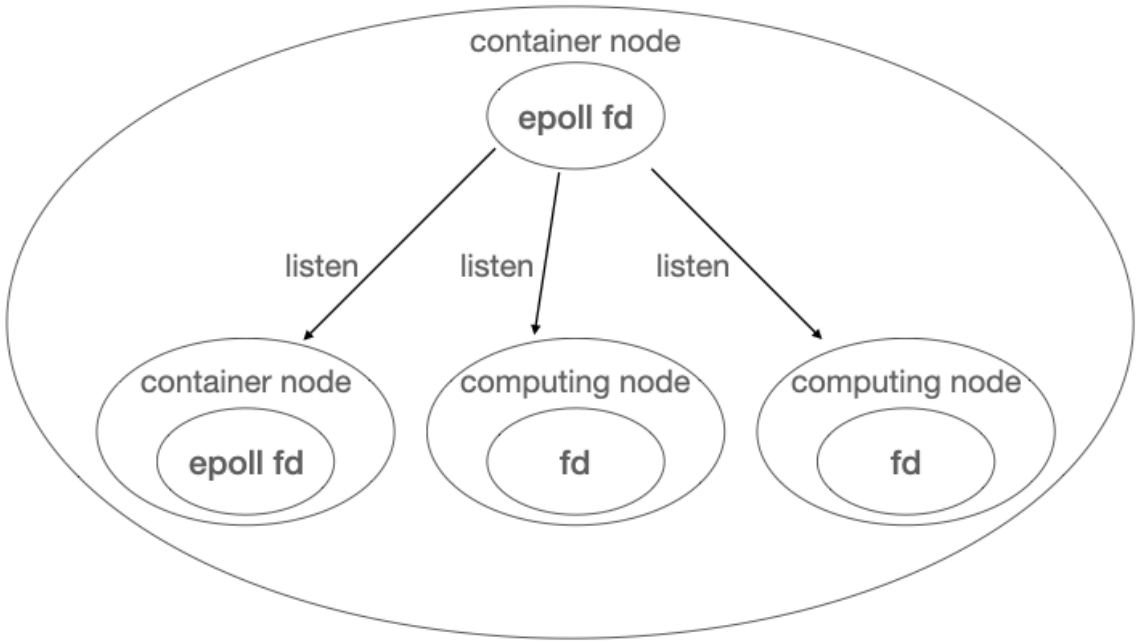


This line is the code of function node

Editor will describe Port structure

User can program below

1. Use In Port data freely
2. Must create Out Port data.



Node Event

- break the fire rule of dataflow execution
  - directly run the node and notify IO event.
  - realized the iterative structure
  - express the nondeterminism
  - Enable Nodes respond to I/O events

Translation Engine

- Translate it into text-based C++ code
  1. Include dataflow library head files
  2. Define user data structure
  3. Define function for function node
  4. Create DAG in main function and execute

1
```
#include "module.h"
#include <iostream>
```

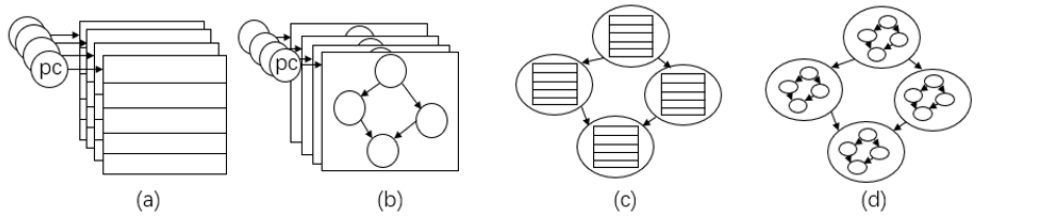
2
```
struct Arg{
    int argc;
    char **argv;
};
```

3
```
void PIPE(ModuleData& in, ModuleData& out){
    auto indata = any_cast<Arg>(in.get("indata"));
//-----------------user code--------------------//

int outdata = indata.argc + 3;
//^^^^^^^^^^^^^^^user code^^^^^^^^^^^^^^^^^^^//
    out.set("outdata",make_any<int>(outdata));
}
```

4
```
int main(int argc,char *argv[])  {
    Module master ("master");
    master.addModule("master_pipe",PIPE);
    master.addEdge("master_pipe", "master", "outdata", "status");
    master.addEdge("master", "master_pipe", "args", "indata");
    Arg arg{argc,argv};
    ModuleData dataIn;
    ModuleData dataOut;
    dataIn.set("args", arg);
    master(dataIn,dataOut);
    return any_cast<int>(dataOut.get("status"));
}
```

Our contribution

- An Expressive dataflow model
  - can naturally represent the latter three



- Simple and Easy to use
  - only need know C/C++ language and simple dataflow rule
  - All code will translate to C++ code for looking into it
- IO listening for Node
  - Expands dataflow programming in network

Thanks for listening