LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

- *Cpppc Project*

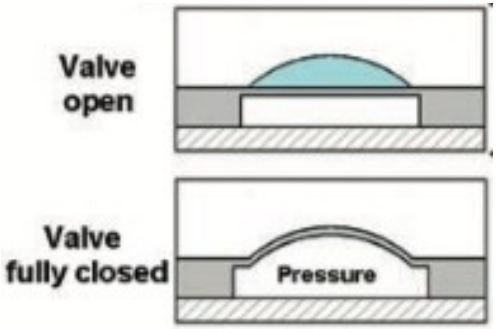
Microfluidic Large-Scale Integration (mLSI) Simulator

- Date: 12.10.2018
- Author: Mengchu Li

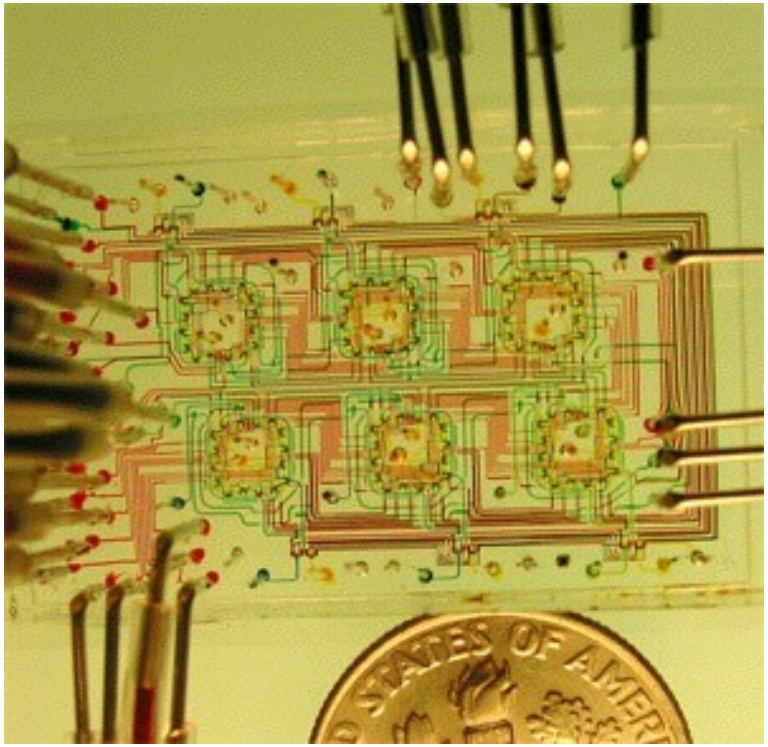
Introduction to mLSI

Manipulation of Continuous Liquid Flow through Micro-fabricated Channels.

- **Control Layer** ⇨ Pressure
- **Flow Layer** ⇨ Liquid
- Interface between Control and Flow Layer ⇨ **Valve**



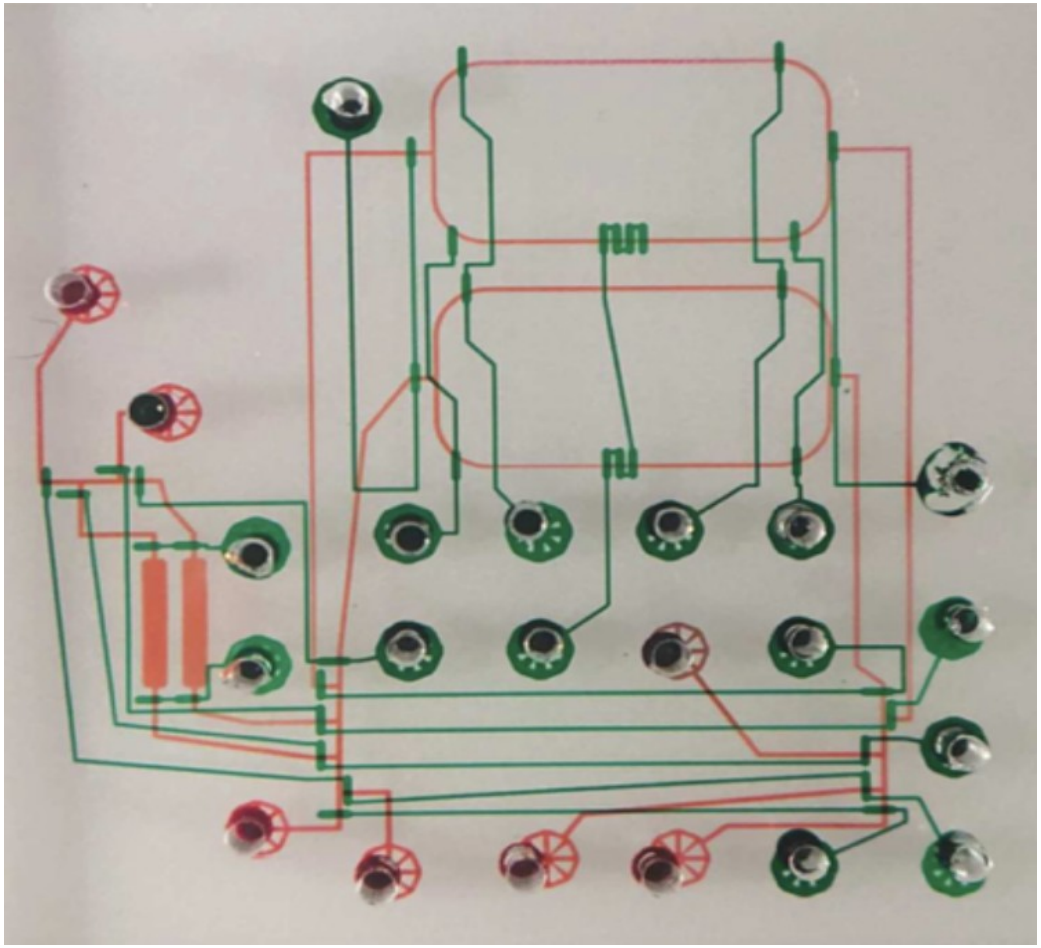
Valve Structure(2)

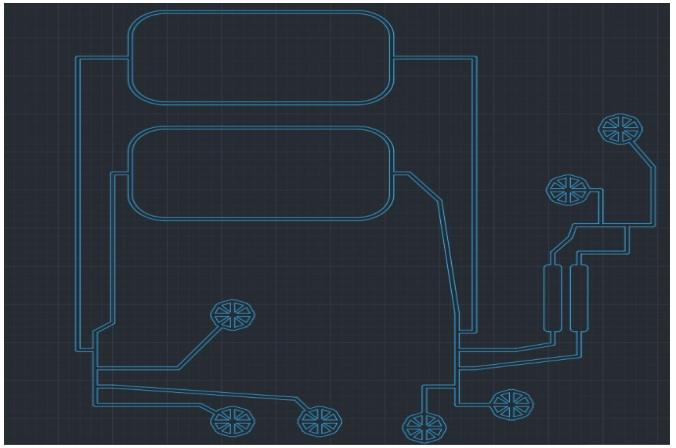


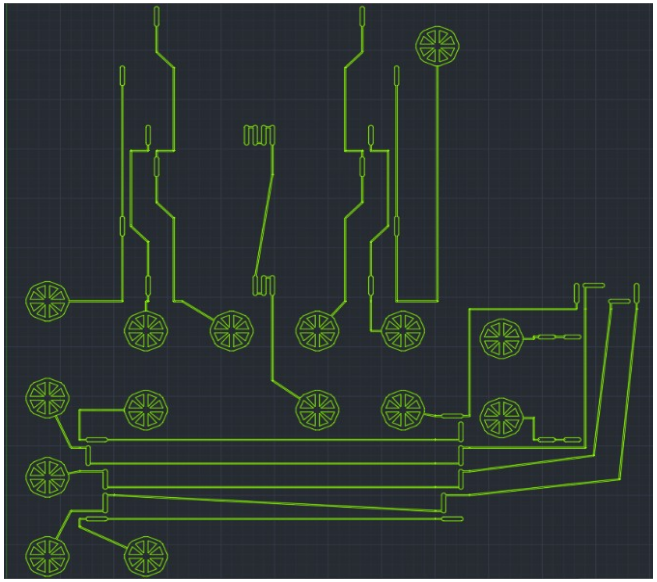
A continuous-flow microfluidic biochip(1)

(1) Department of bioengineering, Stanford University.
(2) Lee CC et al.,"Multistep Synthesis of a Radiolabeled Imaging Probe Using Integrated Microfluidics" Science. 16 December 2005

Another Demonstration







MLSI Simulator

MLSI is a developing field. Currently there is no simulation tool that can demonstrate the flow on the chip.

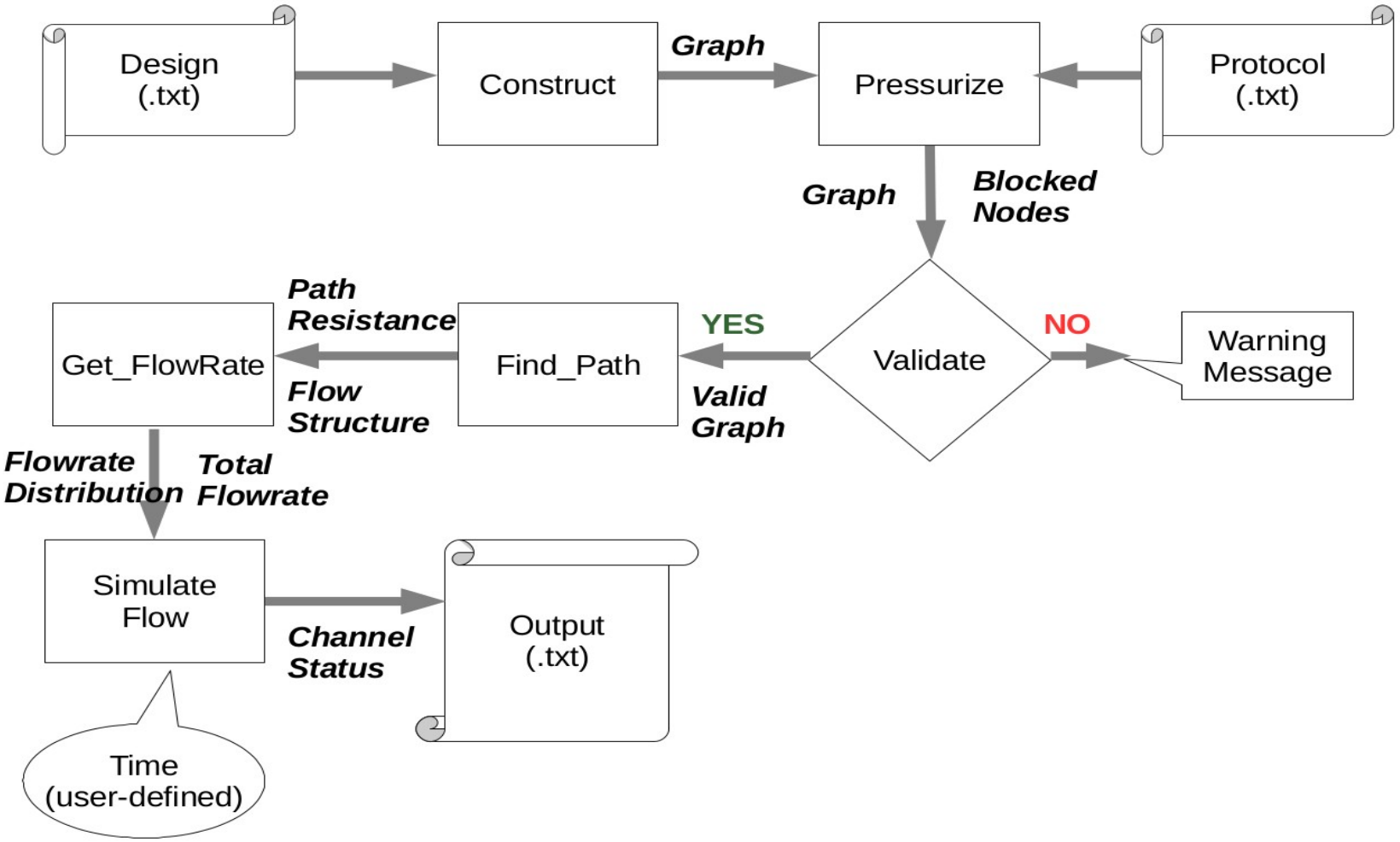
- **Focus:** Correlation of valves and channels

- **Input 1**: text description of chip structure
- **Input 2**: text description of application protocol
- **Input 3**: a user-defined number that specifies time

- **Output**: flow channel status of the given chip at the specified time

- *Quick demonstration of the simulator*

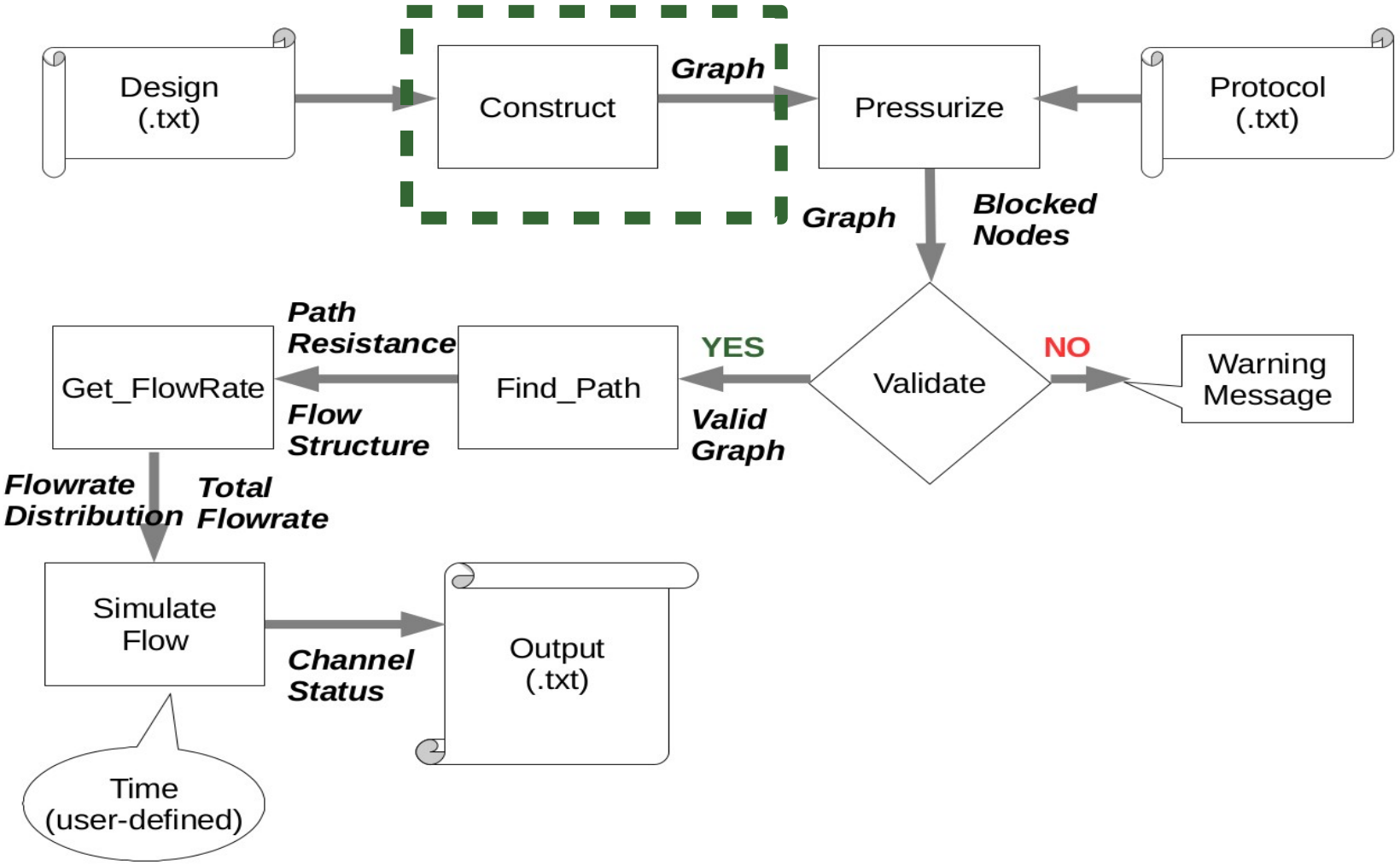
System Diagram



Code Demonstration of the System Diagram


```
int main(int argc, char* argv[]) {
  Graph basic_graph = construct(argv[1]);
  std::map<int, Graph::Node *> blocked_nodes = pressurize(argv[2], basic_graph);
  Graph valid_graph = validate(blocked_nodes, basic_graph);
  if (valid_graph.flag() == false)
    return 1;
  std::tuple<std::map<Graph::Edge, std::pair<Graph::Edge *, double>>,
             std::map<std::pair<Graph::Node*, Graph::Node *>, double>,
             double> seq_par_res = find_path(valid_graph);
  std::pair<std::map<Graph::Edge *, double>,double> flow_collection = get_flow_rate(seq_par_res);
  std::cout<<"Please give the execution time: ";
  double time;
  std::cin>>time;
  std::map<Graph::Edge *, double> utilization = simulate_flow(flow_collection, valid_graph, time);
  output(valid_graph, seq_par_res, flow_collection, utilization, time);
}
```

System Diagram



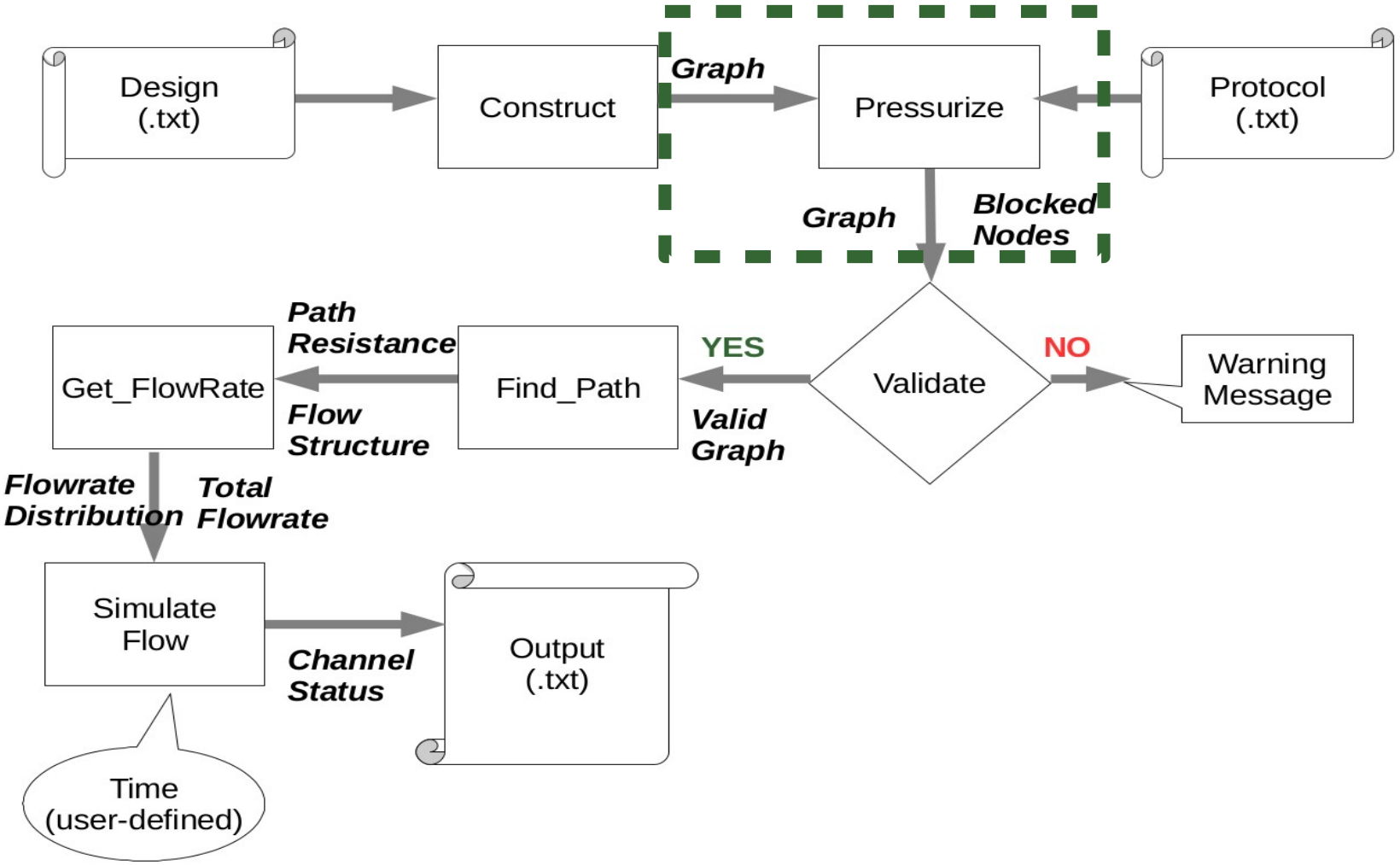
Insight Into the Graph Concept

A graph contains nodes and edges, which interact with each other

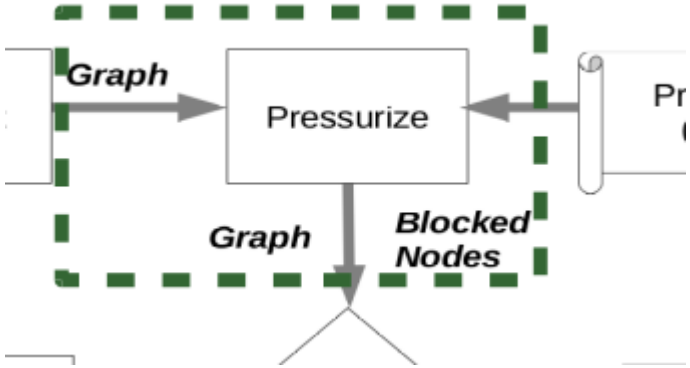
```
Class Graph{
    Class Node;
    Class Edge;

    Class Edge{
        int _index;
        std::pair<Node *, Node *> _ends;
        double _length;
    }
     private:
     std::map<int, Node> _nodes;
     std::map<int, Edge> _edges;
}
```

```
Class Node{
    int _index;
    char _type;
    bool _status;
    std::set<Edge *> connections();
    std::set<Edge *> in();
    std::set<Edge *> out();
}
```

System Diagram

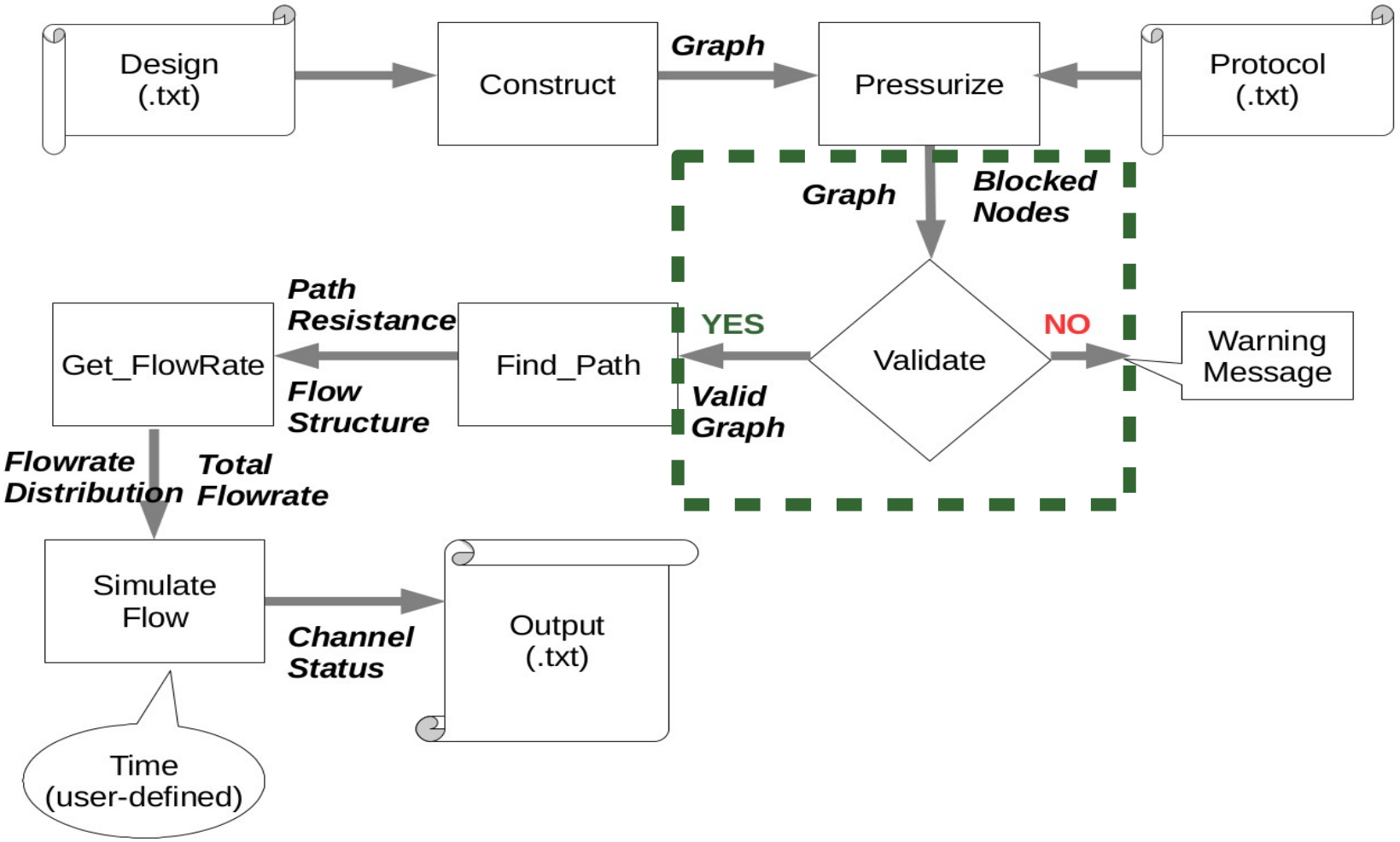


Pressurization




```
std::map<int, Graph::Node *> pressurize(std::string filename, Graph & g){
  read_protocols(filename, g);
  std::map<int, Graph::Node *> blocked_nodes;
  block_nodes(blocked_nodes, g);
  return blocked_nodes;
}
```

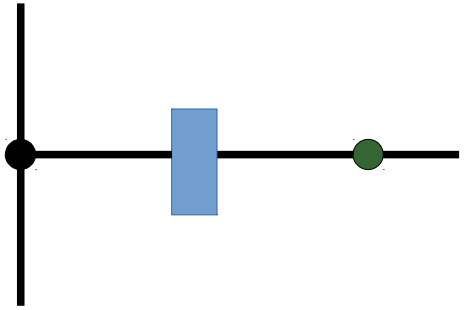
System Diagram



Validate


```
Graph validate(std::map<int, Graph::Node *> & blocked_nodes, Graph & g){
  Graph valid_graph = initialize(blocked_nodes, g);
  int count = 0;
  while (count != valid_graph.nodes().size()){
    count = valid_graph.nodes().size();
    delete_invalid_nodes(valid_graph);
    delete_invalid_branch(valid_graph);
    delete_intermediate_node(valid_graph);
  }
  Graph::Node * departure = find_departure(valid_graph);
  if (departure == nullptr){
    std::cout<<"There is no open inlet and thus no feasible flow path."<<std::endl;
    valid_graph.set_flag(false);
    return valid_graph;
  }
  refine_edges(valid_graph);
  guide_direction(valid_graph, departure);
  return valid_graph;
}
```

Validate - Propagation of the Blocked status

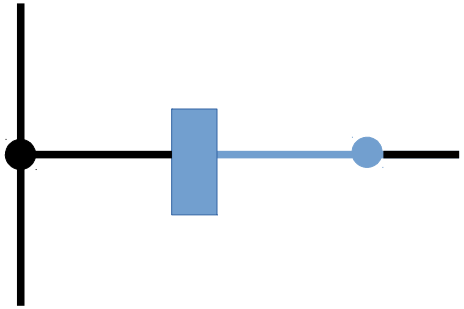


Case 1: The investigated node is not a branching point

⇩

Both the node and the connecting edge will be blocked

Validate - Propagation of the Blocked status

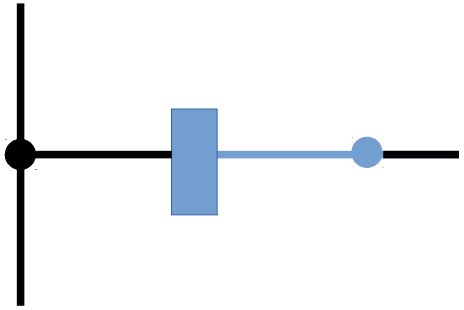


Case 1: The investigated node is not a branching point

⇩

Both the node and the connecting edge will be blocked

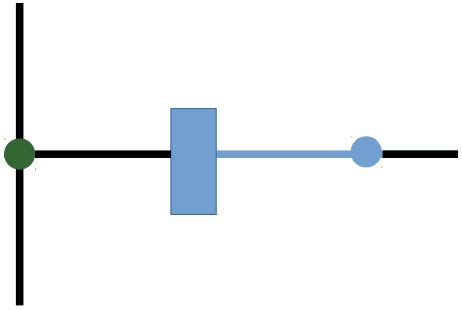
Validate - Propagation of the Blocked status



Case 1: The investigated node is not a branching point

⇩

Both the node and the connecting edge will be blocked

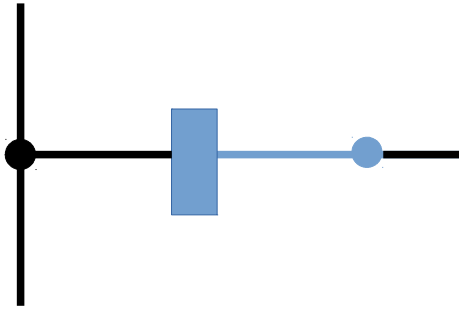


Case 2: The investigated node is a branching point

⇩

the node will not be blocked but the connecting edge will be blocked

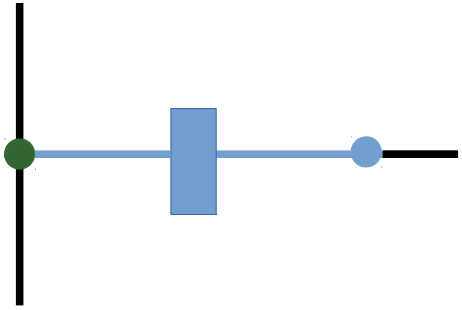
Validate - Propagation of the Blocked status



Case 1: The investigated node is not a branching point

⇩

Both the node and the connecting edge will be blocked



Case 2: The investigated node is a branching point

⇩

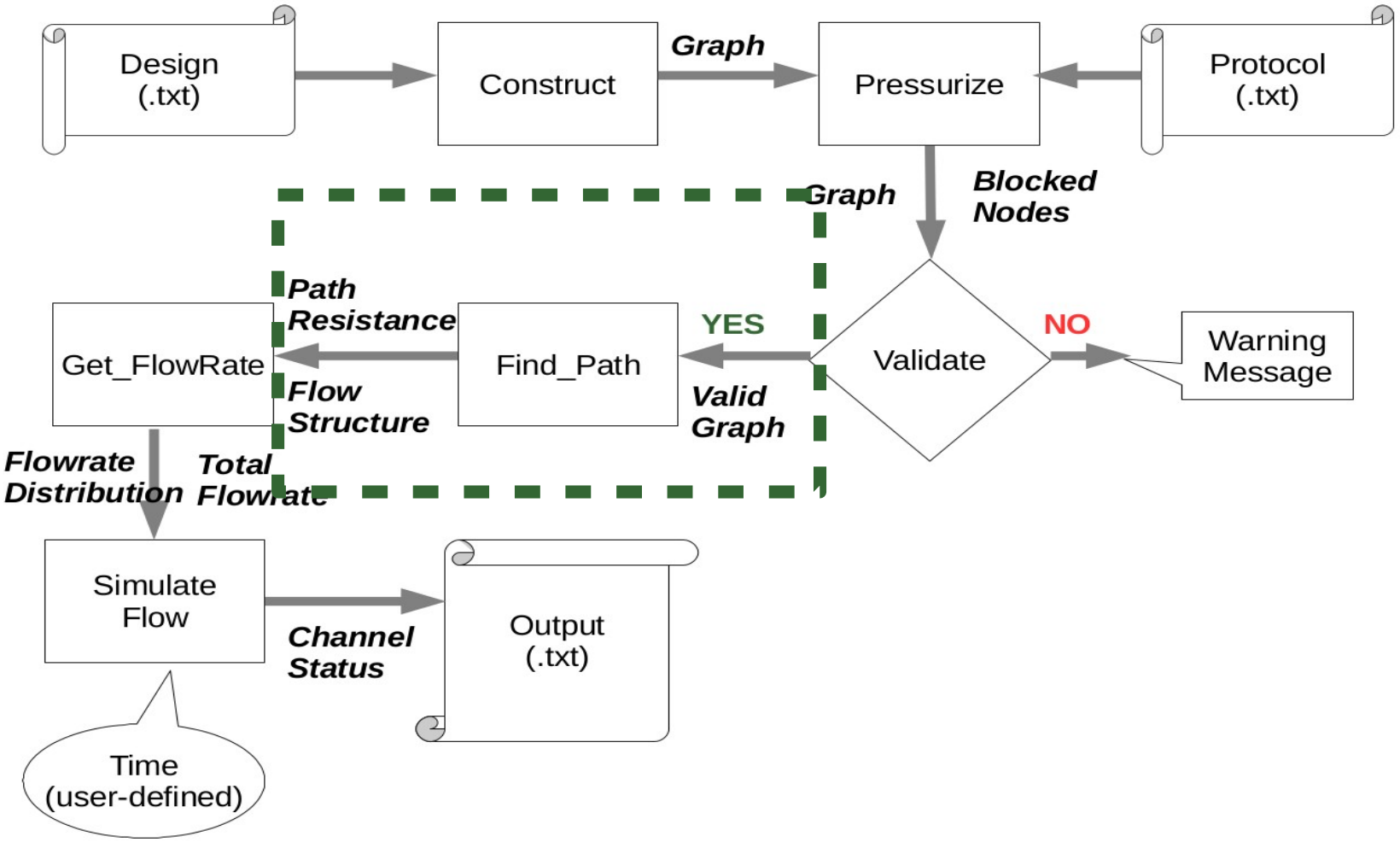
the node will not be blocked but the connecting edge will be blocked

...

Validate


```
Graph validate(std::map<int, Graph::Node *> & blocked_nodes, Graph & g){
  Graph valid_graph = initialize(blocked_nodes, g);
  int count = 0;
  while (count != valid_graph.nodes().size()){
    count = valid_graph.nodes().size();
    delete_invalid_nodes(valid_graph);
    delete_invalid_branch(valid_graph);
    delete_intermediate_node(valid_graph);
  }
  Graph::Node * departure = find_departure(valid_graph);
  if (departure == nullptr){
    std::cout<<"There is no open inlet and thus no feasible flow path."<<std::endl;
    valid_graph.set_flag(false);
    return valid_graph;
  }
  refine_edges(valid_graph);
  guide_direction(valid_graph, departure);
  return valid_graph;
}
```

System Diagram



Find_Path


```
std::tuple<std::map<Graph::Edge, std::pair<Graph::Edge *, double>>,
           std::map<std::pair<Graph::Node*, Graph::Node *>, double>,
           double> seq_par_res = find_path(valid_graph);
```

- **Output:** A tuple consisting of
  - Sequentially connected edges and their resistance;
  - In parallel connected edges and their resistace;
  - Total resistance
- If two edges share the same end nodes, it means that these two edges are in parallel. In this case, these edges can be merged as a new edge with smaller resistance
- After introducing new edges, there may be new sequentially connected edges, the end between these two edges is redundant. In this case, delete the node and merge the edge
- Iteratively refine until no more change happens