gem5-Garnet

Syam Sankar
Ph.D Scholar (Prime Minister’s Research Fellow)
MARS Lab, Dept. of CSE, IIT Guwahati

- The default network model in Ruby is the **simple** network.
  - hop-by-hop network traversal (switches and links)
  - monitoring the available buffers and available bandwidth in output links before sending

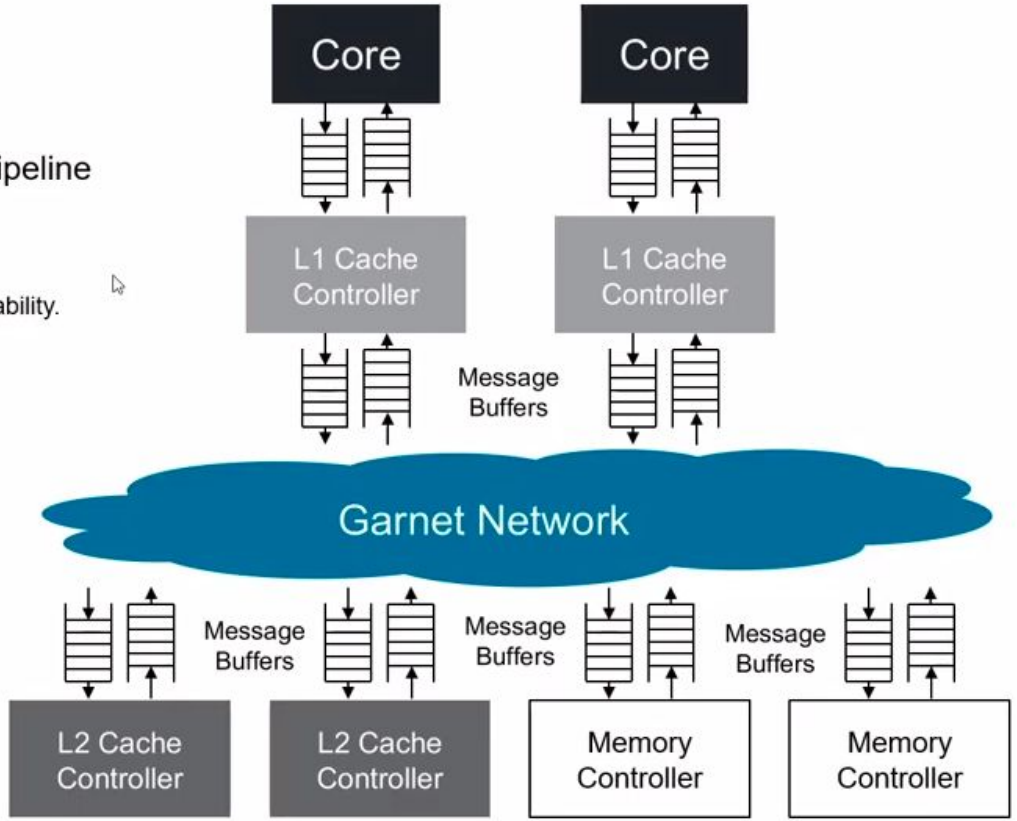
Related Files:

- **src/mem/ruby/network/Network.py**
- **src/mem/ruby/network/simple**
- **src/mem/ruby/network/simple/SimpleNetwork.py**

Garnet Network Simulator

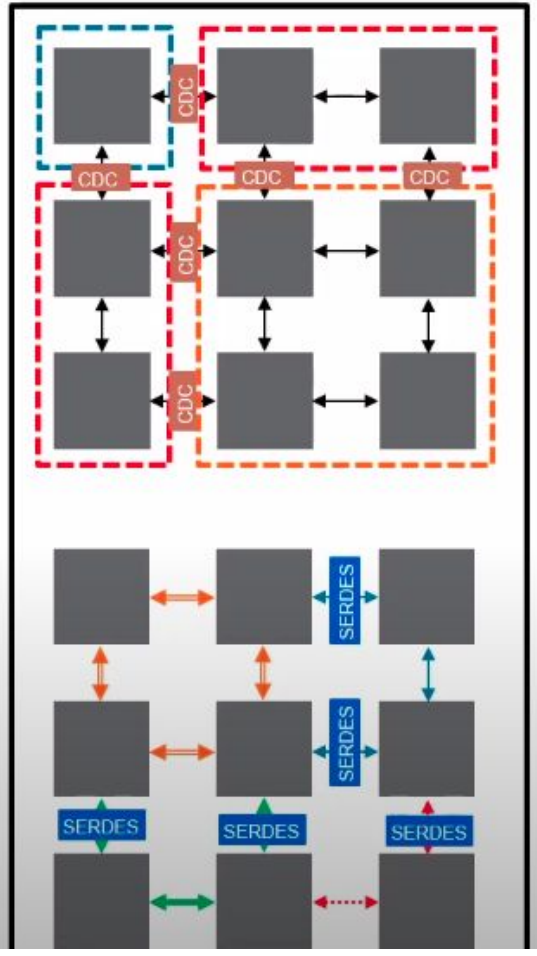
- Detailed NoC Model
  - Part of Ruby Memory System in gem5
  - Garnet 1.0 released in 2009: Fixed 5-stage router pipeline
  - Garnet 2.0 released in 2016: Configurable routers
  - Garnet 3.0 includes HeteroGarnet
    - Supports diverse heterogeneous networks with complete configurability.

- Features of Garnet 2.0
  - Configurable topology
  - Detailed router pipeline model
  - Credit-based flow control
  - Configurable routing policies
  - Synthetic traffic simulations



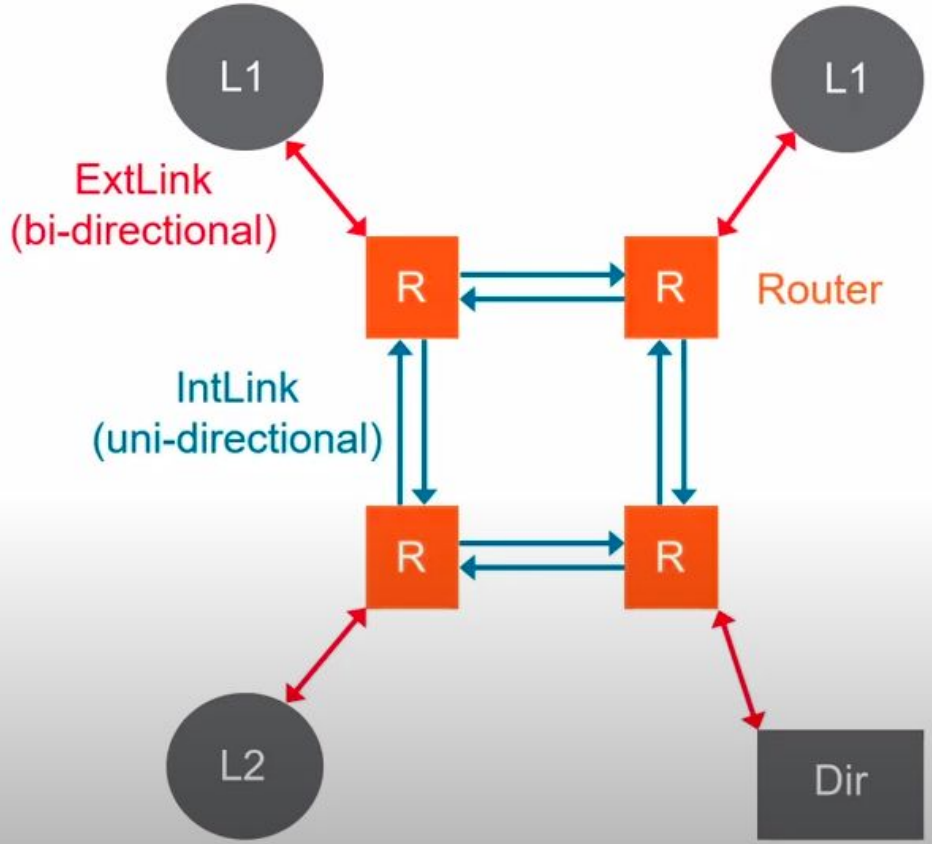
HeteroGarnet – Enabling Heterogeneous Networks

- Multiple clock domains
  - Each building block within network can be configured to be in a clock domain.
- Heterogeneous link widths
  - Each routers and link can be configured to support a distinct flit width
- Multiple sub-networks
  - Multiple mutually-exclusive networks could co-exist in a simulation
- Clock-domain crossing units
- Serializer-deserializer units



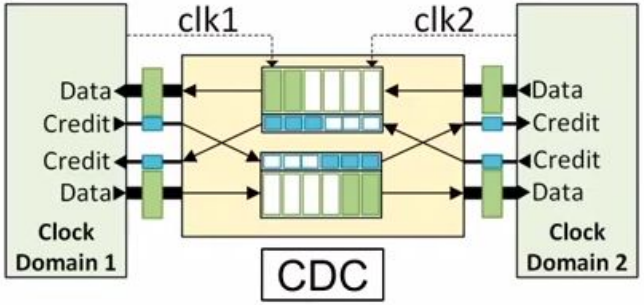
Building Topology in Garnet/HeteroGarnet

- Each topology is defined using a python file located in `configs/topologies`
- Building blocks on Garnet
  - Router: Detailed router model with crossbar
  - ExtLink: Used to connect to external controllers
  - IntLink: Used to connect routers within network
- Each block has configurable parameters.

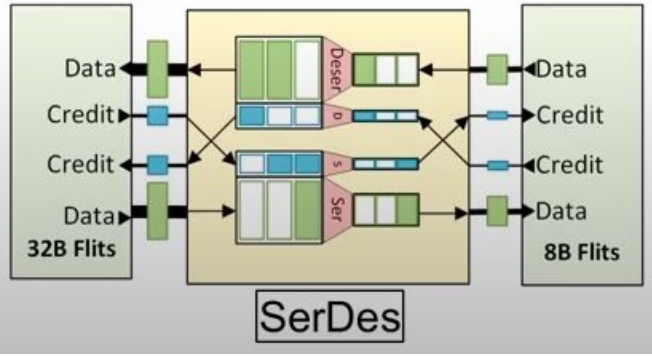


HeteroGarnet - CDC and SerDes Units

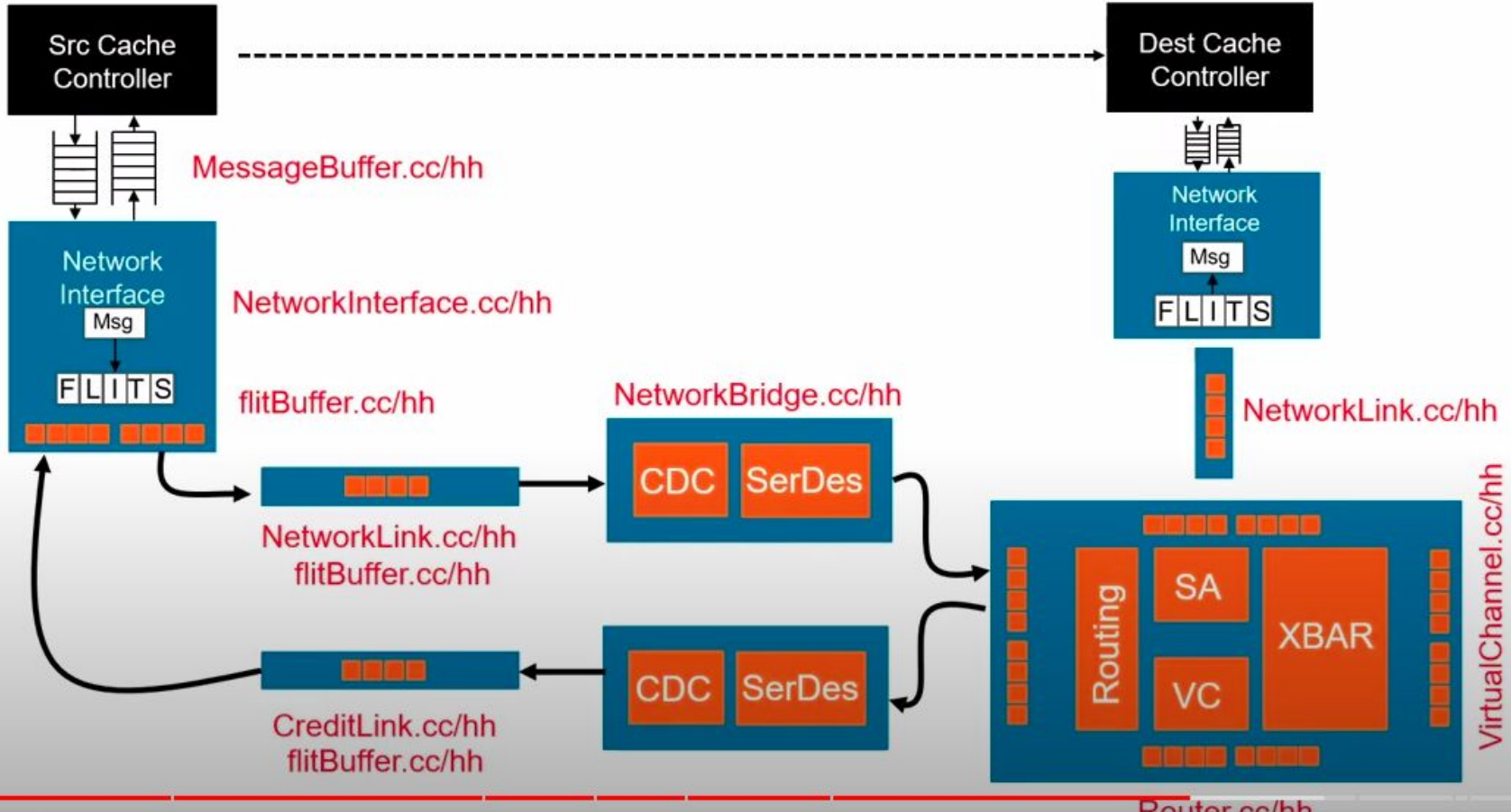
- Clock Domain Crossing (CDC)
  - Enables transmission of flits across clock domains
  - Supports flow-control credit crossings
  - Configurable latency
    - Static or Dynamic (Function of clock domain frequencies)
  - Enabled by setting flags on the links
    - Internal Links: `src_cdc` and/or `dst_cdc`
    - External Links: `ext_cdc` and/or `int_cdc`



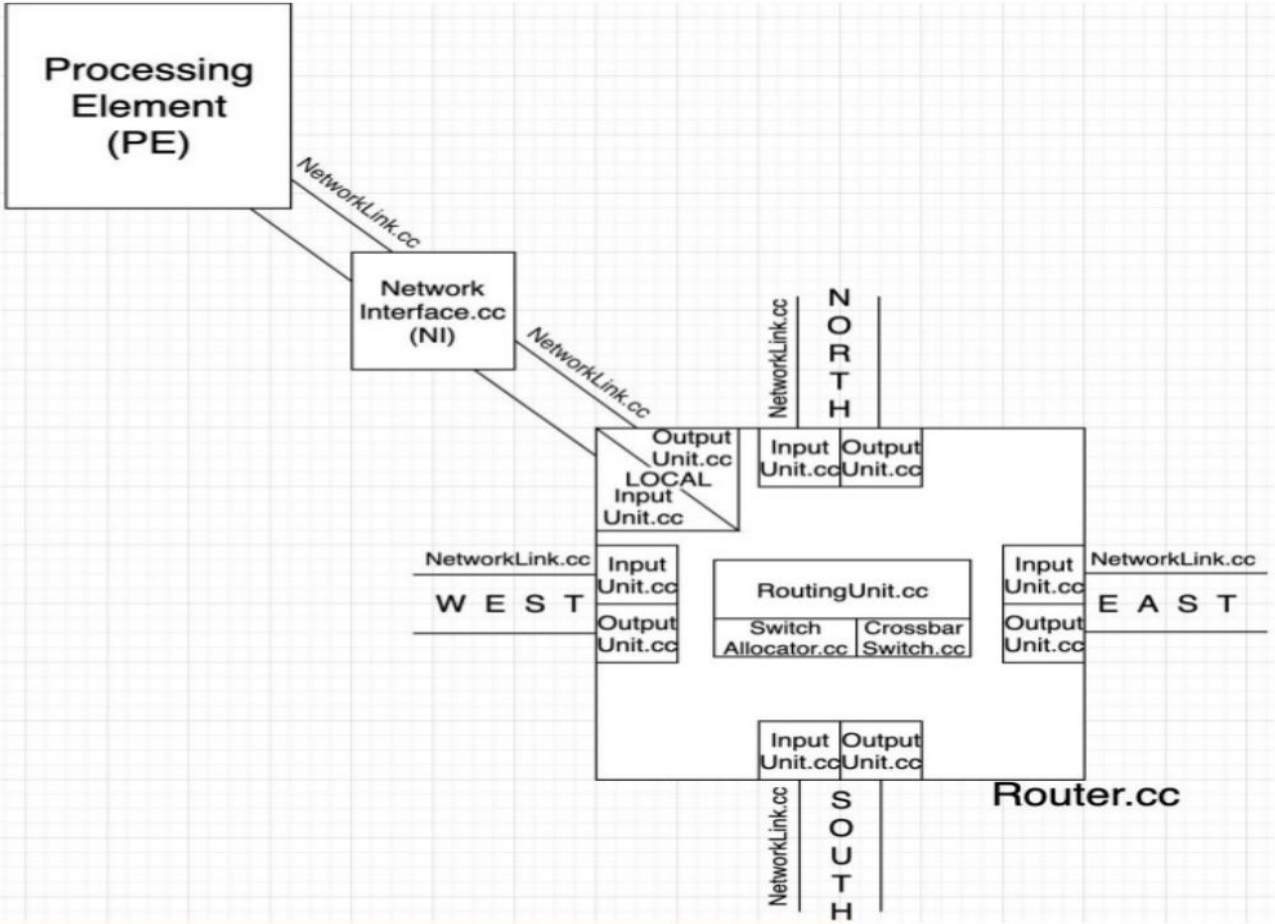
- Serializer-Deserializer (SerDes)
  - Enables serialization or deserialization of flits
  - Supports flow-control credit crossings
  - Configurable latency
  - Enabled by setting flags on the links
    - Internal Links: `src_serdes` and/or `dst_serdes`
    - External Links: `ext_serdes` and/or `int_serdes`



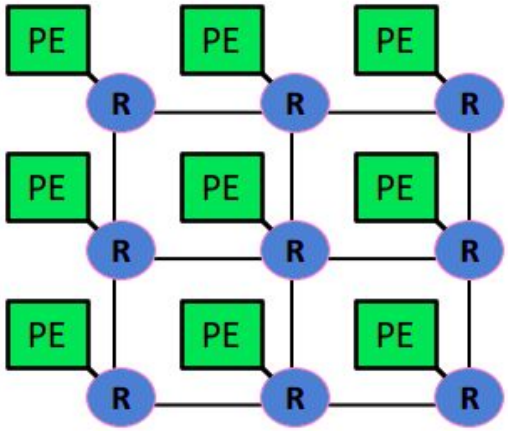
Life Cycle of a Message through HeteroGarnet



Garnet design: an overview



On-Chip Networks

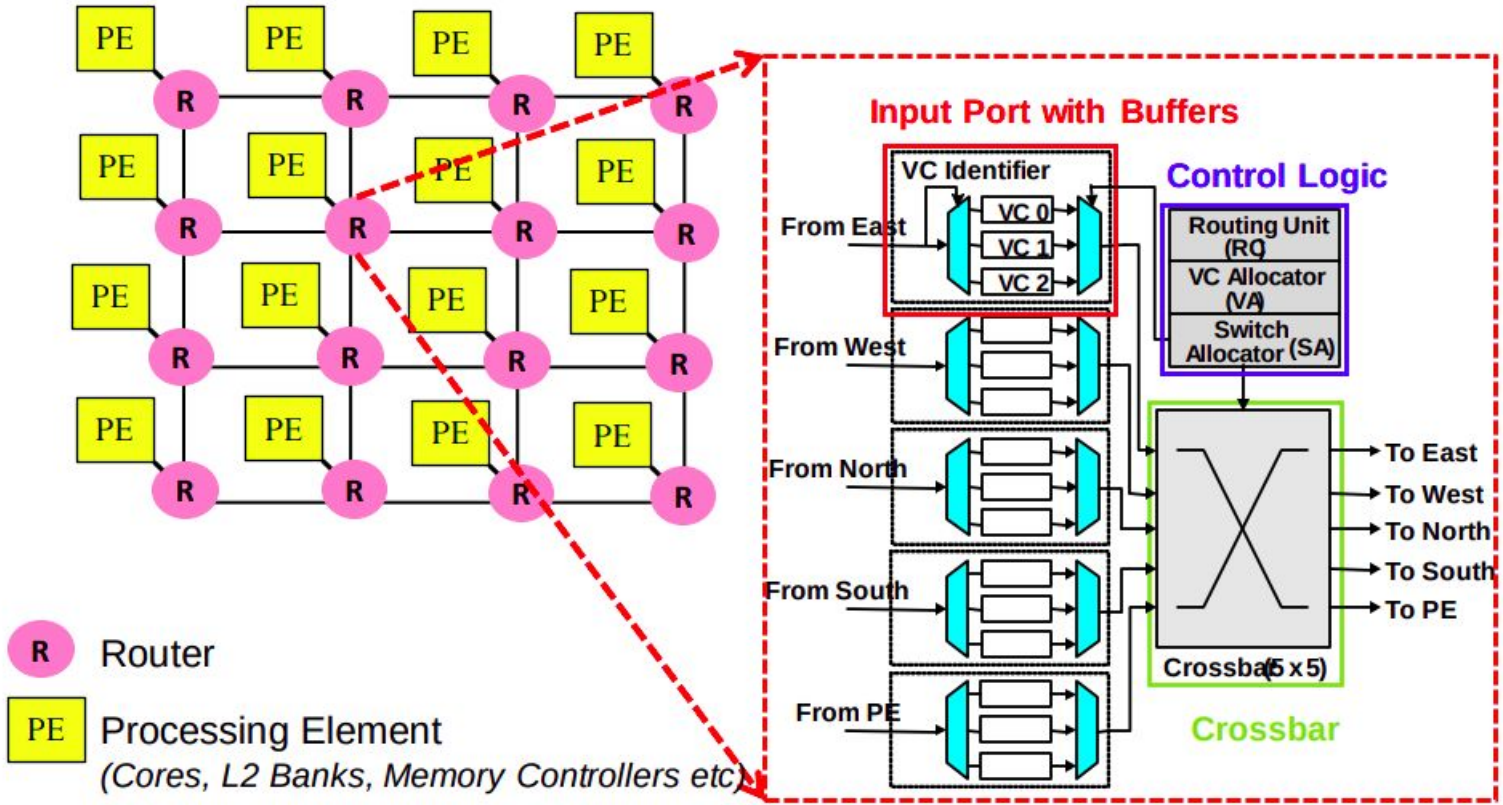


- Connect **cores, caches, memory controllers, etc**
  - Buses and crossbars are not scalable
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**

R Router

PE Processing Element
(Cores, L2 Banks, Memory Controllers, etc)

On-chip Networks



Lifecycle of a Network Traversal

- NetworkInterface.cc::wakeup()
  - Every NI connected to one coherence protocol controller on one end, and one router on the other.
  - receives messages from coherence protocol buffer in appropriate vnet and converts them into network packets and sends them into the network.
    - garnet2.0 adds the ability to capture a network trace at this point [under development].
  - receives flits from the network, extracts the protocol message and sends it to the coherence protocol buffer in appropriate vnet.
  - manages flow-control (i.e., credits) with its attached router.
  - The consuming flit/credit output link of the NI is put in the global event queue with a timestamp set to next cycle. The eventqueue calls the wakeup function in the consumer.

- NetworkLink.cc::wakeup()
  - receives flits from NI/router and sends it to NI/router after m_latency cycles delay
  - Default latency value for every link can be set from command line (see configs/network/Network.py)
  - Per link latency can be overwritten in the topology file
  - The consumer of the link (NI/router) is put in the global event queue with a timestamp set after m_latency cycles. The eventqueue calls the wakeup function in the consumer.

- Router.cc::wakeup()
  - Loop through all InputUnits and call their wakeup()
  - Loop through all OutputUnits and call their wakeup()
  - Call SwitchAllocator's wakeup()
  - Call CrossbarSwitch's wakeup()
  - The router's wakeup function is called whenever any of its modules (InputUnit, OutputUnit, SwitchAllocator, CrossbarSwitch) have a ready flit/credit to act upon this cycle.

- InputUnit.cc::wakeup()
  - Read input flit from upstream router if it is ready for this cycle
  - For HEAD/HEAD_TAIL flits, perform route computation, and update route in the VC.
  - Buffer the flit for (m_latency - 1) cycles and mark it valid for SwitchAllocation starting that cycle.
    - Default latency for every router can be set from command line (see configs/network/Network.py)
    - Per router latency (i.e., num pipeline stages) can be set in the topology file.

- OutputUnit.cc::wakeup()
  - Read input credit from downstream router if it is ready for this cycle
  - Increment the credit in the appropriate output VC state.
  - Mark output VC as free if the credit carries is_free_signal as true

- SwitchAllocator.cc::wakeup()
  - Note: SwitchAllocator performs VC arbitration and selection within it.
  - SA-I (or SA-i): Loop through all input VCs at every input port, and select one in a round robin manner.
    - For HEAD/HEAD_TAIL flits only select an input VC whose output port has at least one free output VC.
    - For BODY/TAIL flits, only select an input VC that has credits in its output VC.
  - Place a request for the output port from this VC.
  - SA-II (or SA-o): Loop through all output ports, and select one input VC (that placed a request during SA-I) as the winner for this output port in a round robin manner.
    - For HEAD/HEAD_TAIL flits, perform outvc allocation (i.e., select a free VC from the output port.
    - For BODY/TAIL flits, decrement a credit in the output vc.
  - Read the flit out from the input VC, and send it to the CrossbarSwitch
  - Send a increment_credit signal to the upstream router for this input VC.
    - for HEAD_TAIL/TAIL flits, mark is_free_signal as true in the credit.
    - The input unit sends the credit out on the credit link to the upstream router.
  - Reschedule the Router to wakeup next cycle for any flits ready for SA next cycle.

- CrossbarSwitch.cc::wakeup()
  - Loop through all input ports, and send the winning flit out of its output port onto the output link.
  - The consuming flit output link of the router is put in the global event queue with a timestamp set to next cycle. The eventqueue calls the wakeup function in the consumer.

- NetworkLink.cc::wakeup()
  - receives flits from NI/router and sends it to NI/router after m_latency cycles delay
  - Default latency value for every link can be set from command line (see configs/network/Network.py)
  - Per link latency can be overwritten in the topology file
  - The consumer of the link (NI/router) is put in the global event queue with a timestamp set after m_latency cycles. The eventqueue calls the wakeup function in the consumer.

Thank you