Noxim Tutorial

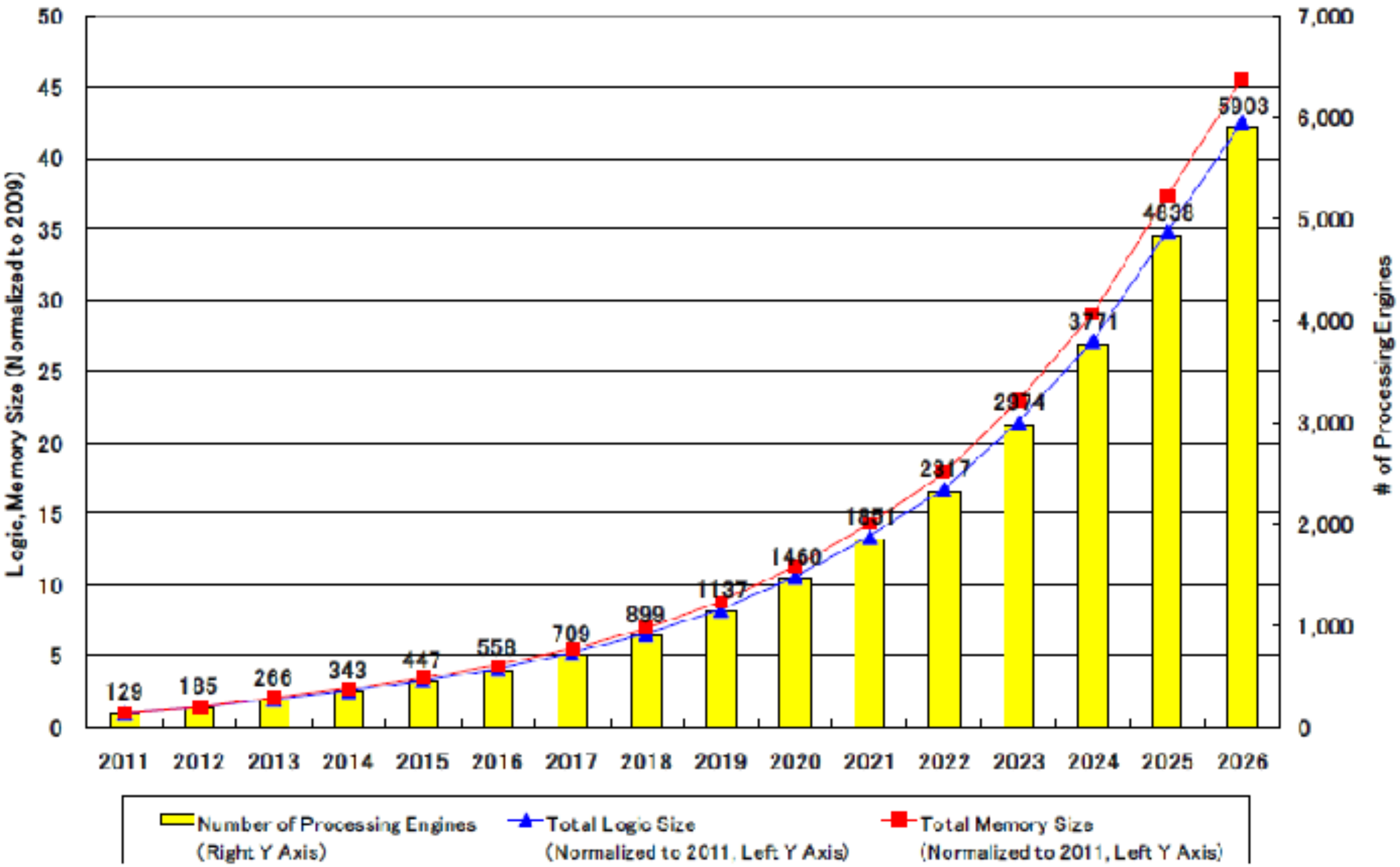
A SystemC cycle-accurate Simulator for On-Chip Networks

Davide Patti, Ph.D.
davide.patti@dieei.unict.it
University of Catania, Italy

Outline

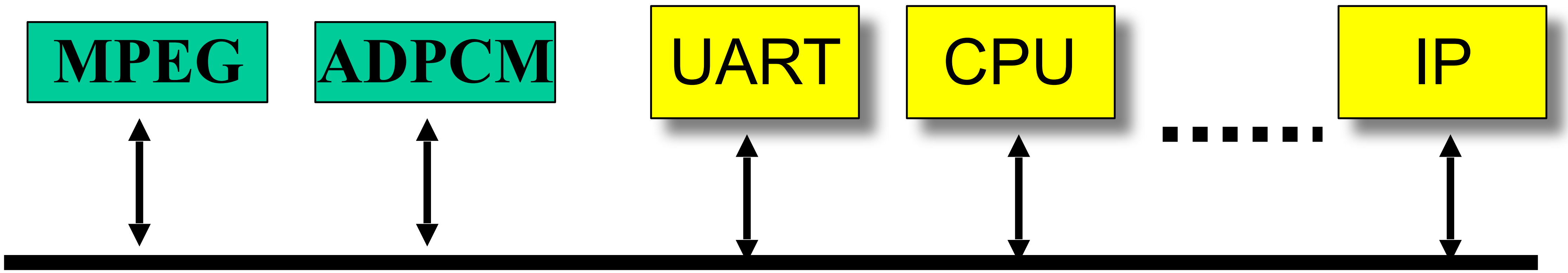
- Intro to Network-on-Chip
- Noxim simulator goals
- Reference Architecture
- Installation & Usage
- Noxim internals: modifying & adding features

Number of Processing Elements: Trend



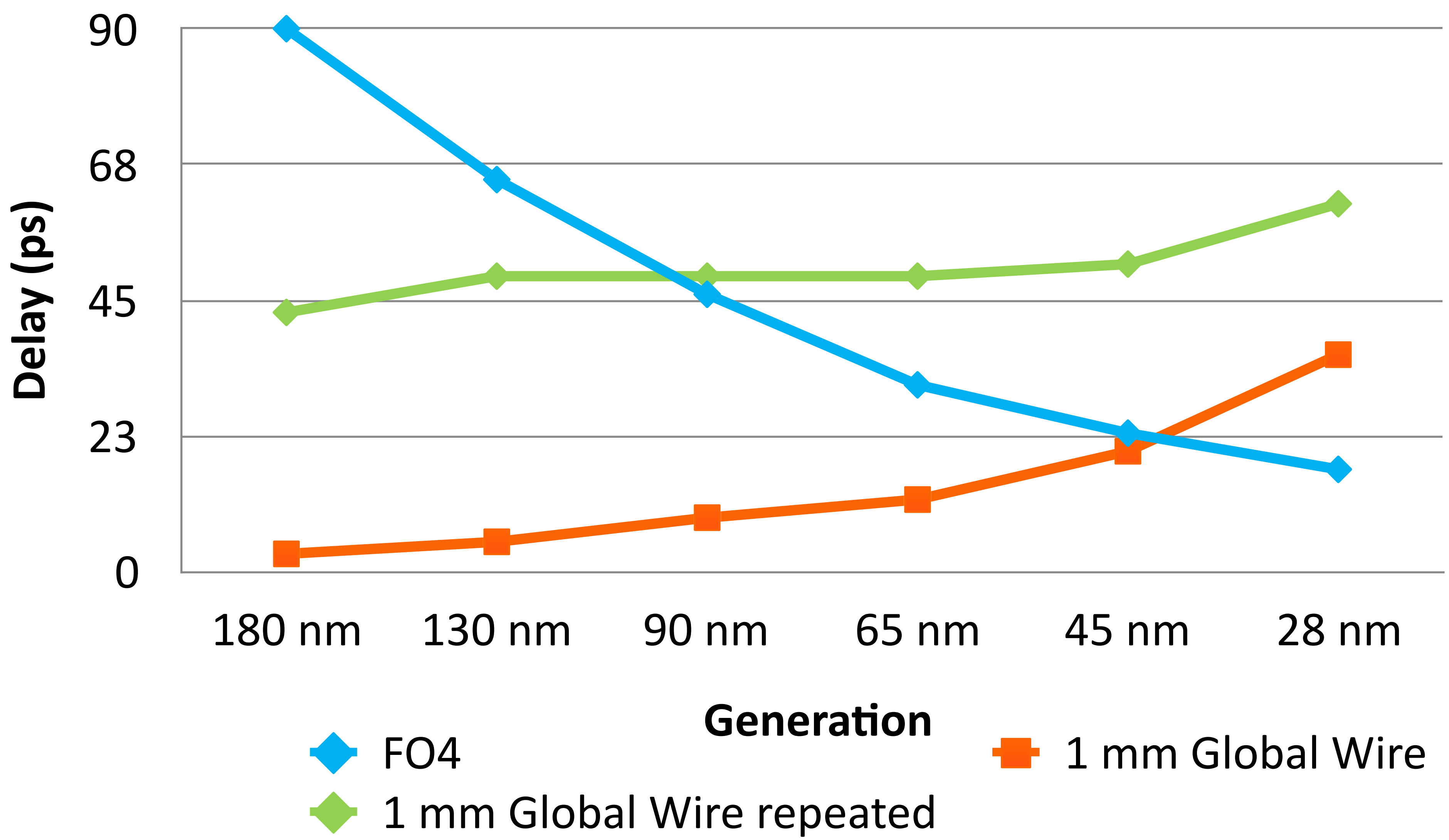
ITRS 2011 – System Drivers

Shared Bus Scalability



- More and more core to share the same bus
- Bus signal propagation delay: e.g. 22mm a 1ns delay
  - at 100MHz —>clock period 10ns, 1ns wire delay ok
  - at 1GHz —-> clock period 1ns is, comparable bus delay
- Smaller technology, lower voltage—> more noise

Transistor and Wire Delay Trend in CMOS



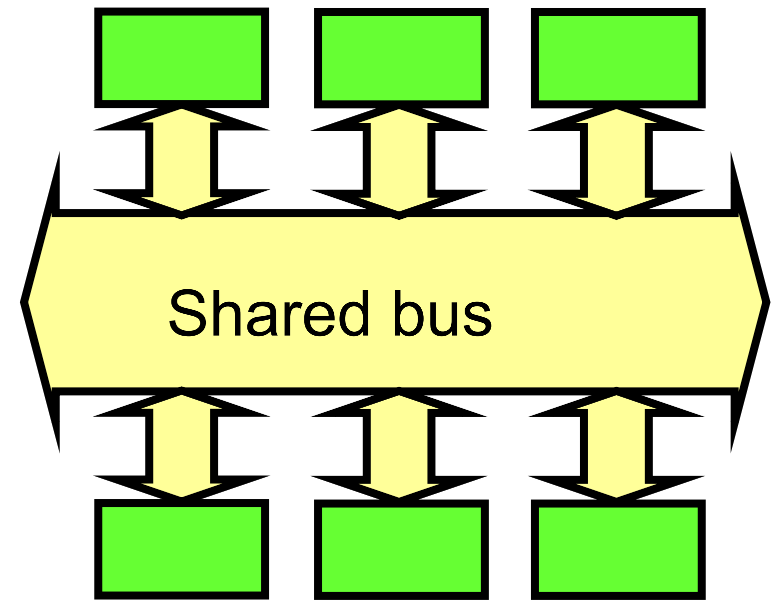
Ron Ho et al, "*The future of wires*", Proceedings of the IEEE , Volume 89 , Issue 4

Global Wiring

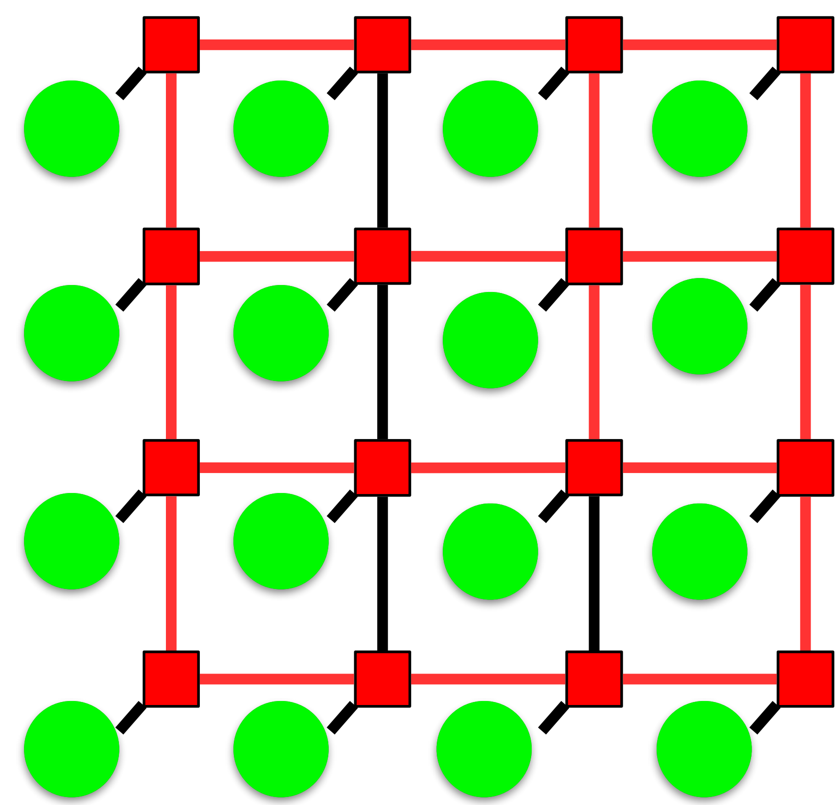
- Today, approximately 50% of dynamic power consumption is due to interconnects' RC
- This percentage will increase
- Global interconnect length doesn't scale with smaller transistors and local wires
  - ➔ Chip size remains relatively constant because chip function continues to increase
  - ➔ RC delay is increasing exponentially
- At 32nm, RC delay in 1 mm global wire at minimum pitch is 25x higher than intrinsic delay of a 2-input NAND fanout of 5

Shared Bus vs Network-on-Chip

- **Shared bus**
  - → Low area
  - → Poor scalability
  - → High energy consumption

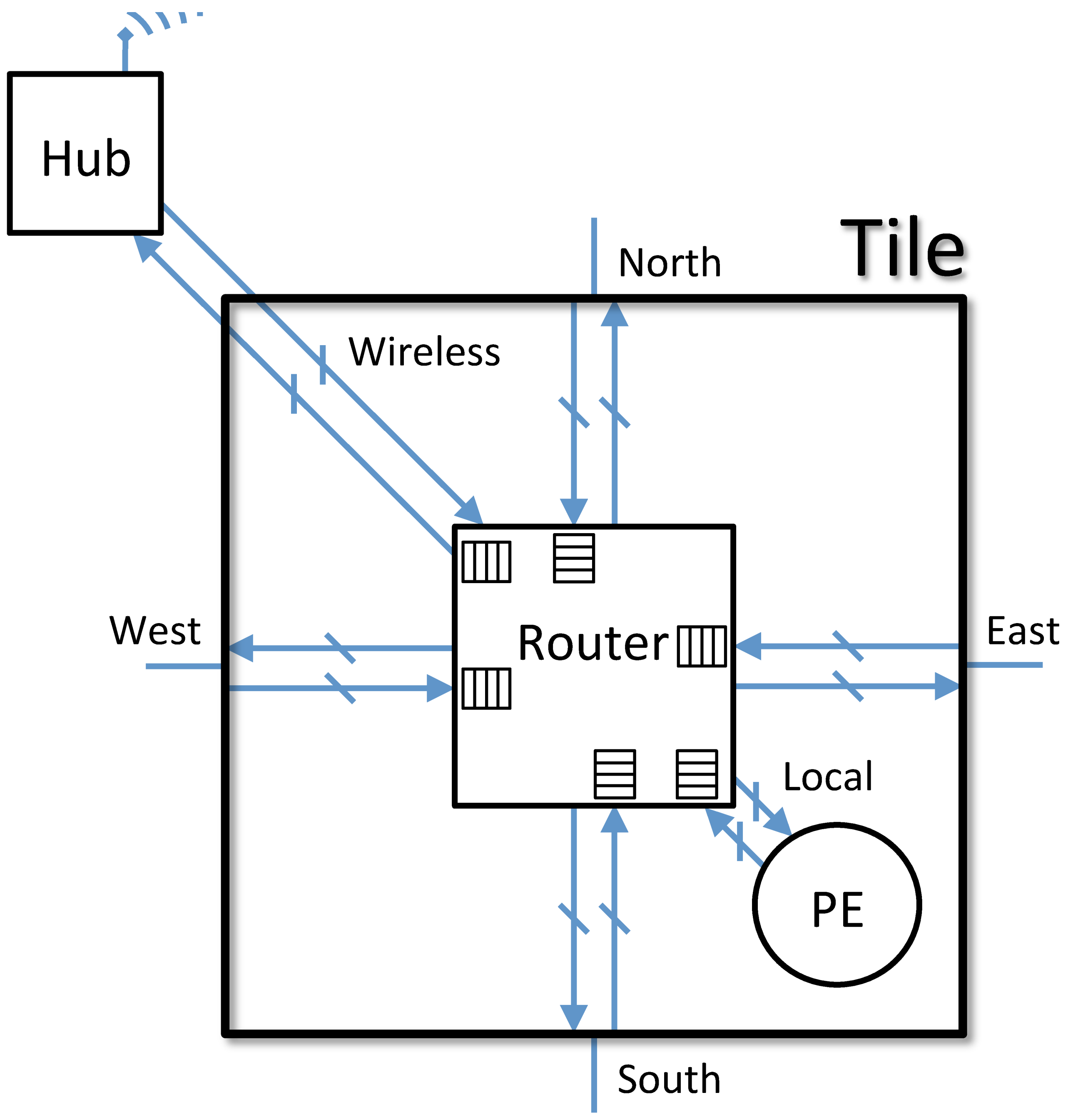


- **Network-on-Chip**
  - → Mesh of Routers (in red)
  - → Each **Processing Element** connected to a **Router**
  - → Scalability and modularity
  - → Low energy consumption
  - → Increase of design complexity



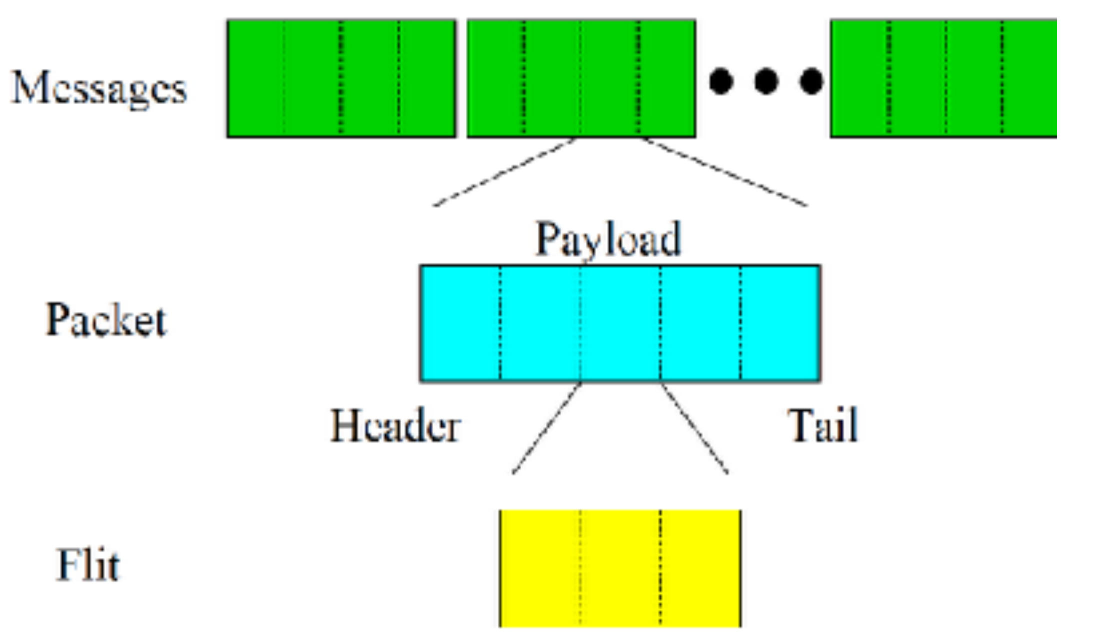
PE+Router= Tile Node

- Default scenario: Mesh of *Tile* nodes
- Each Tile contains a *Router* and a *Processing Element (PE)*
- Each Tile is connected to the 4 neighbours
- Optionally, some nodes can be connected to Radio-hub allowing wireless transmissions

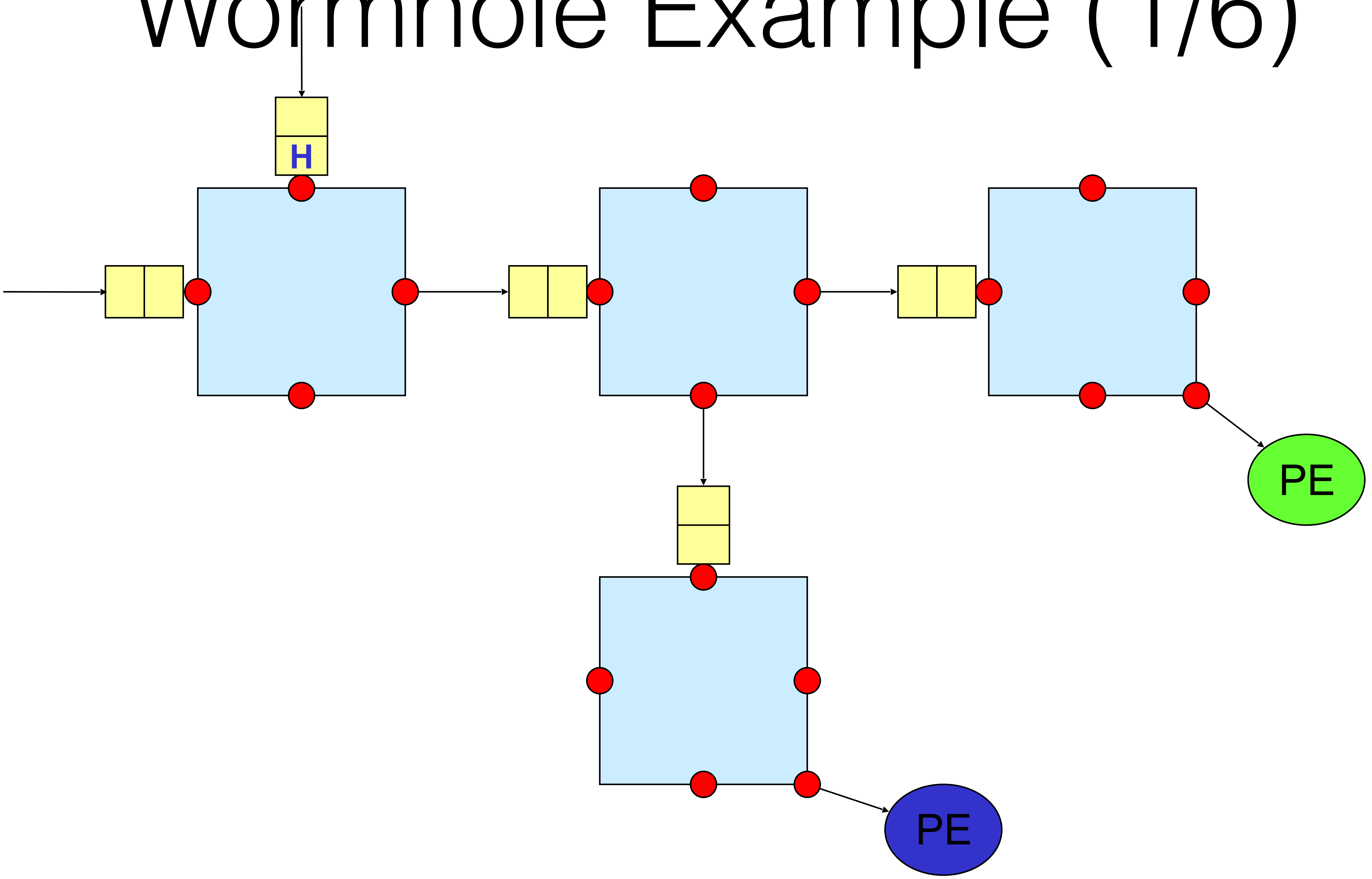


Wormhole & Flits

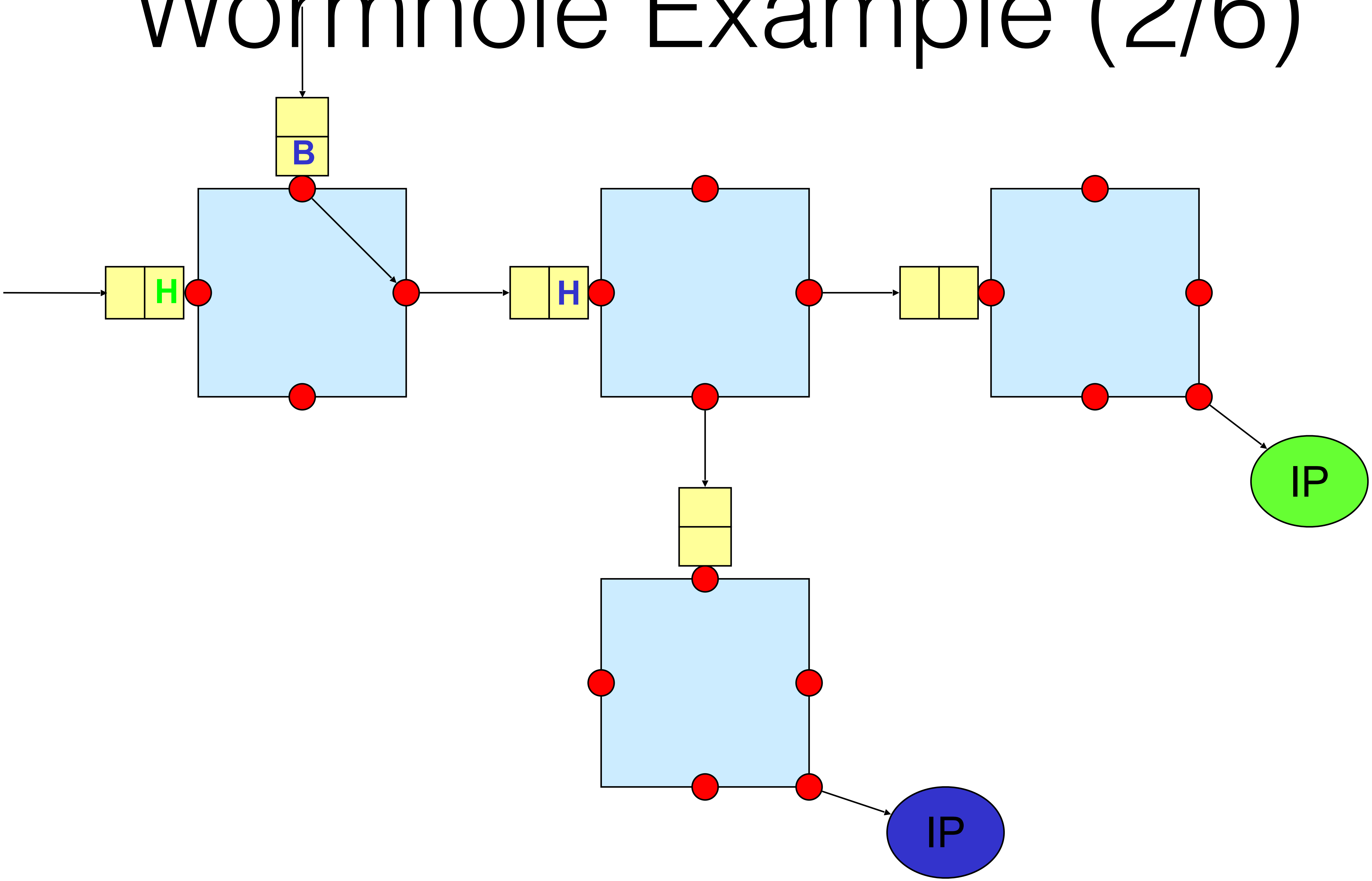
- While travelling from source—>destination cannot store the packet at each intermediate node
  - Lot of resources (large buffers)
  - Delay for storing and restoring to retransmit
  - Solution: **wormhole**



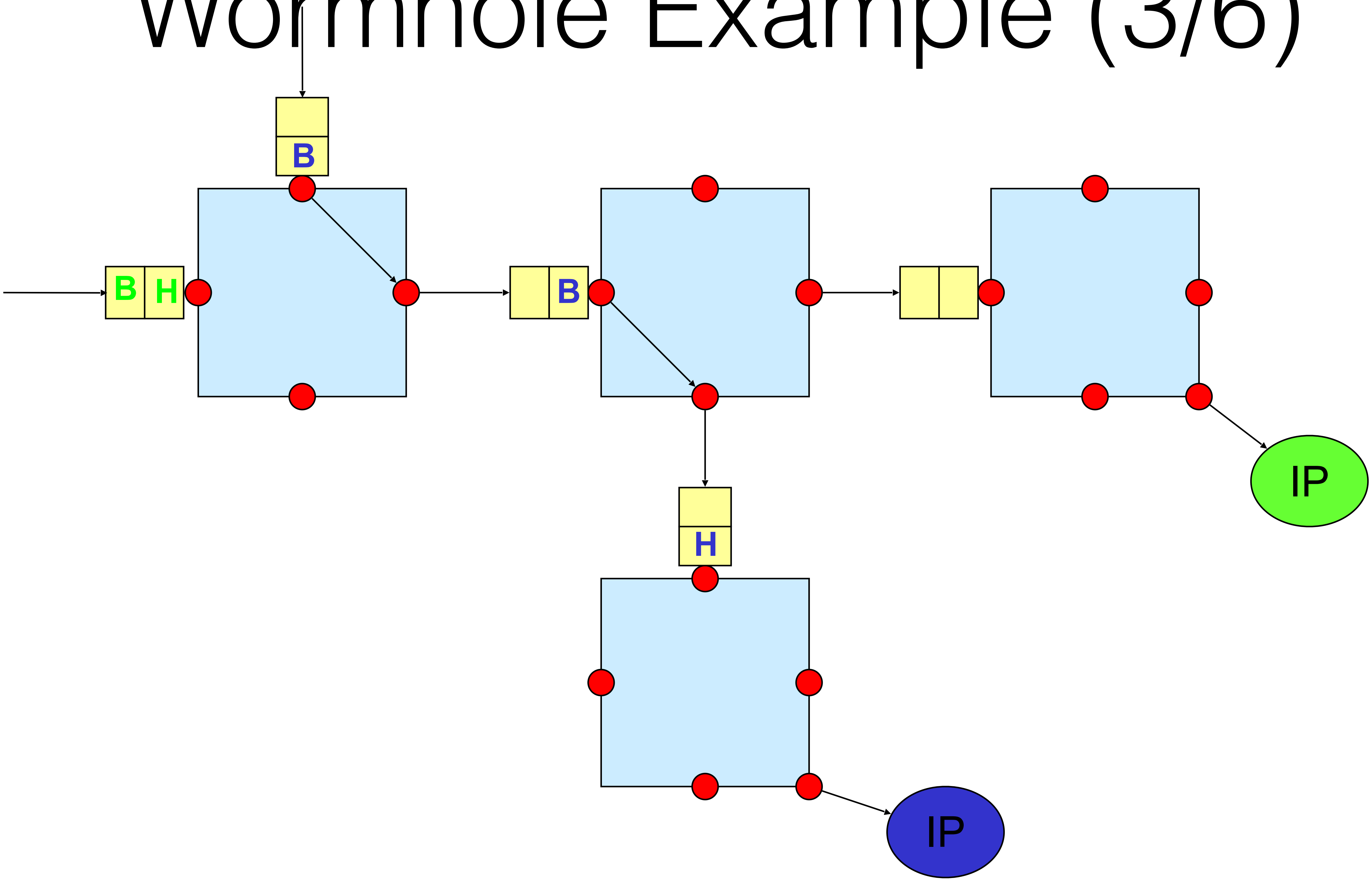
Wormhole Example (1/6)



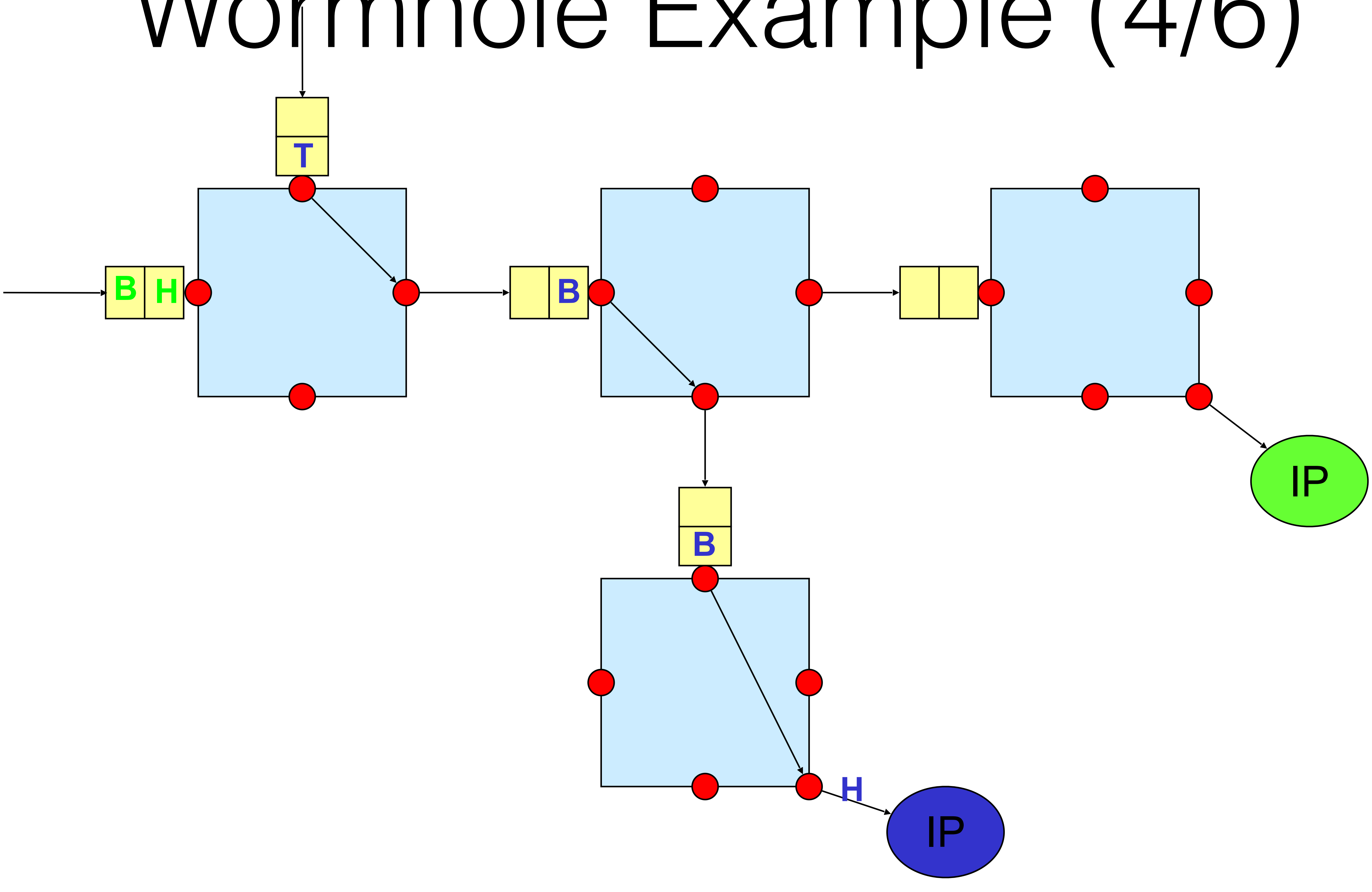
Wormhole Example (2/6)



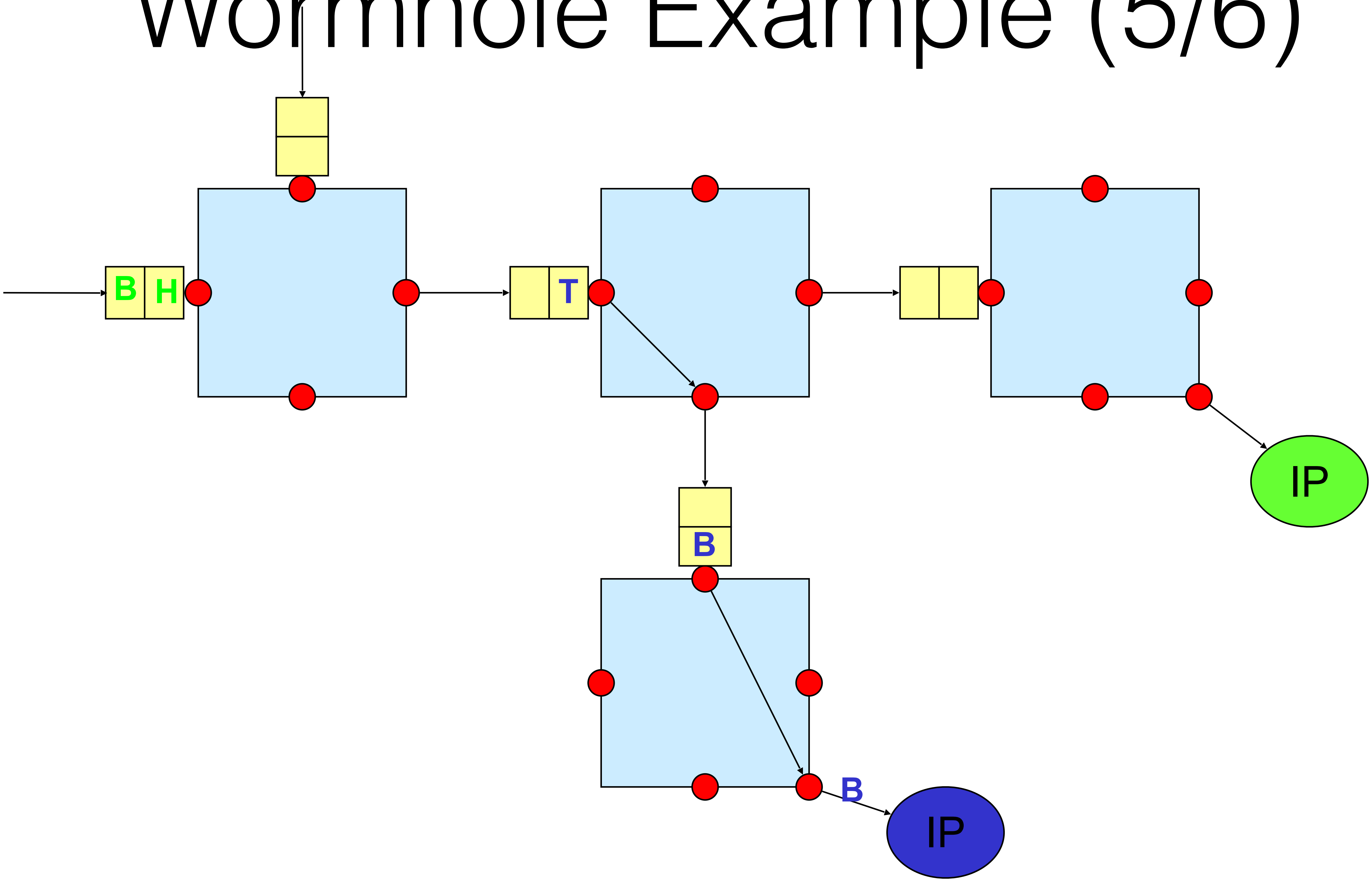
Wormhole Example (3/6)



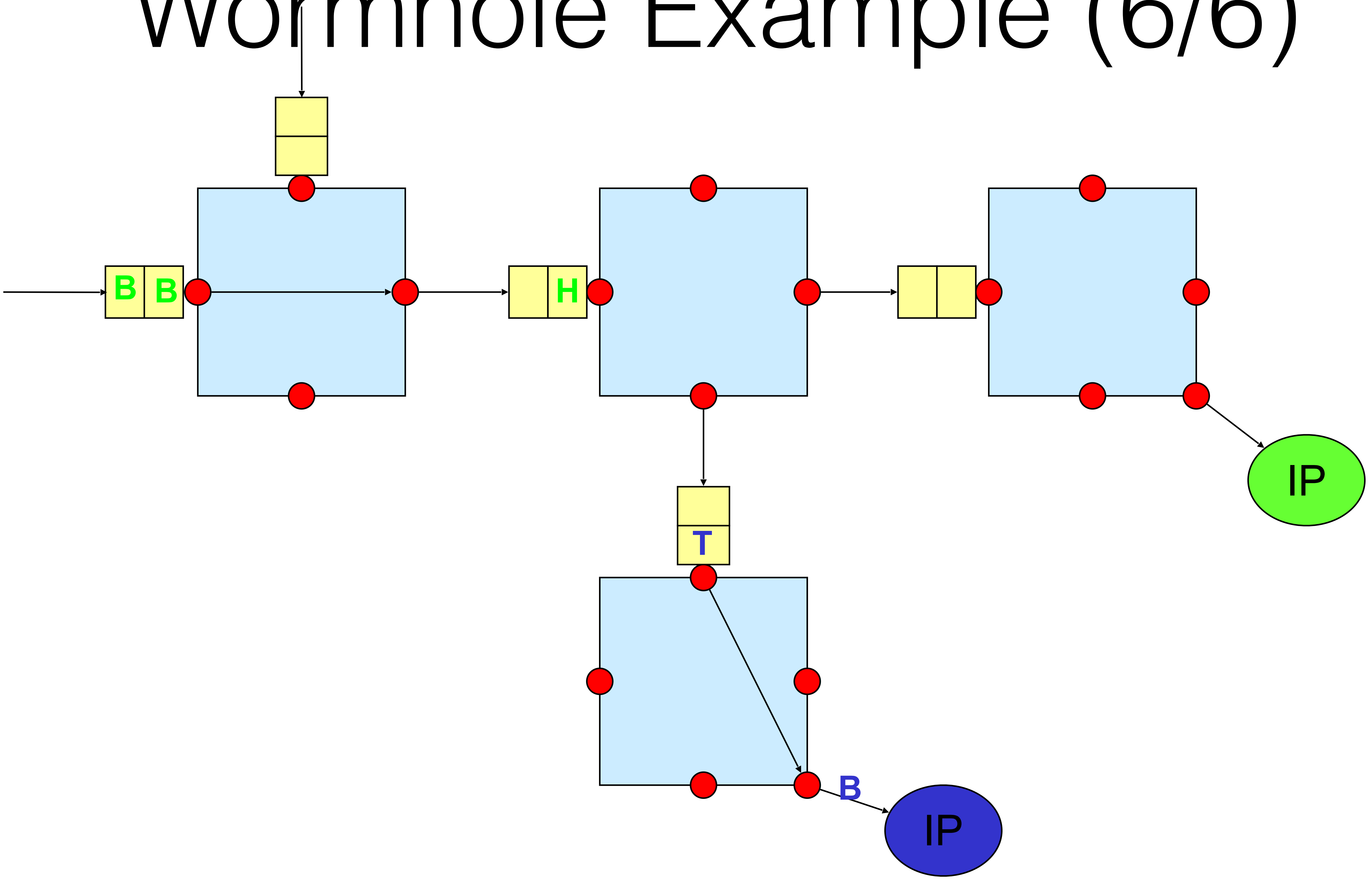
Wormhole Example (4/6)



Wormhole Example (5/6)

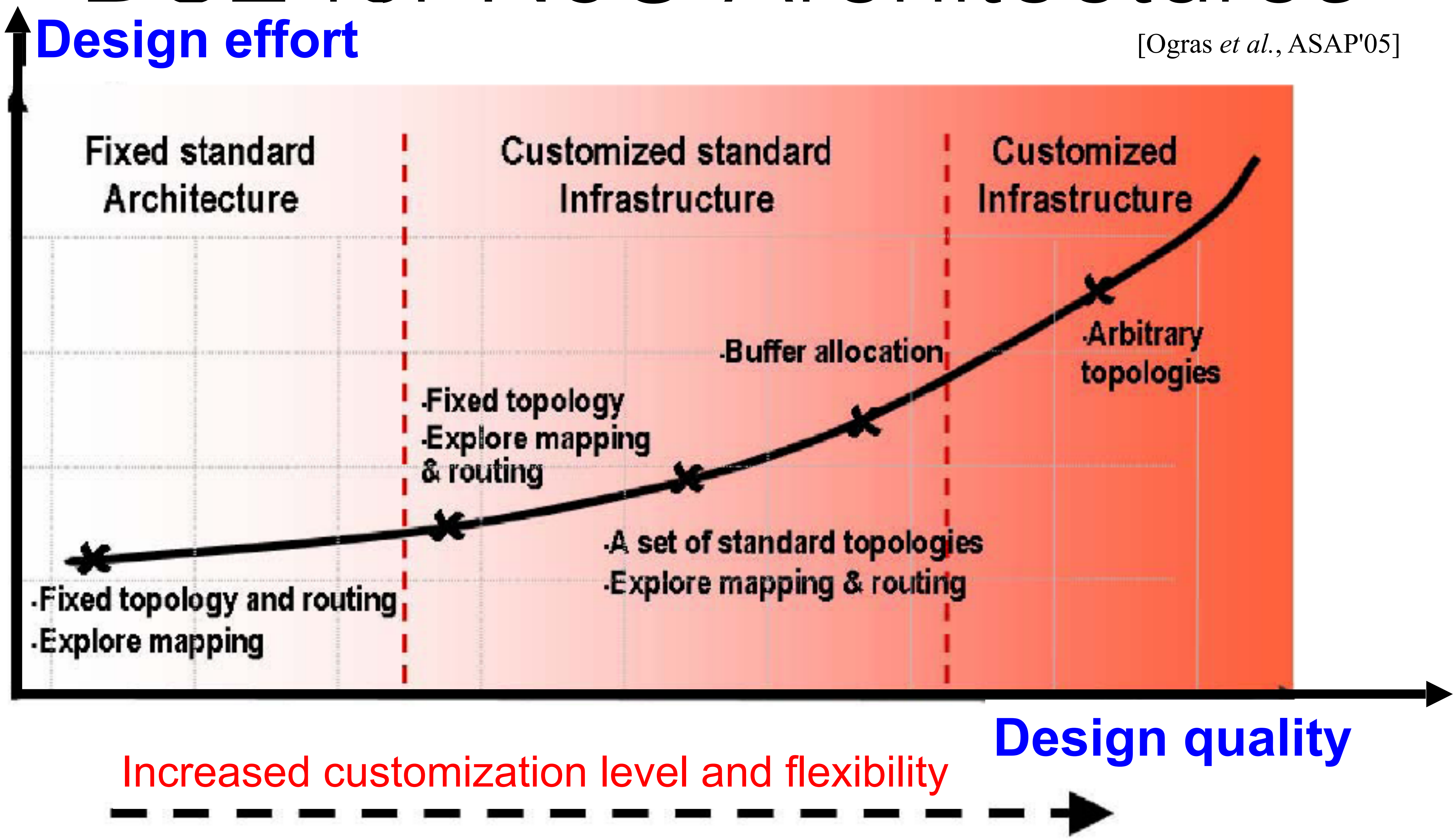


Wormhole Example (6/6)

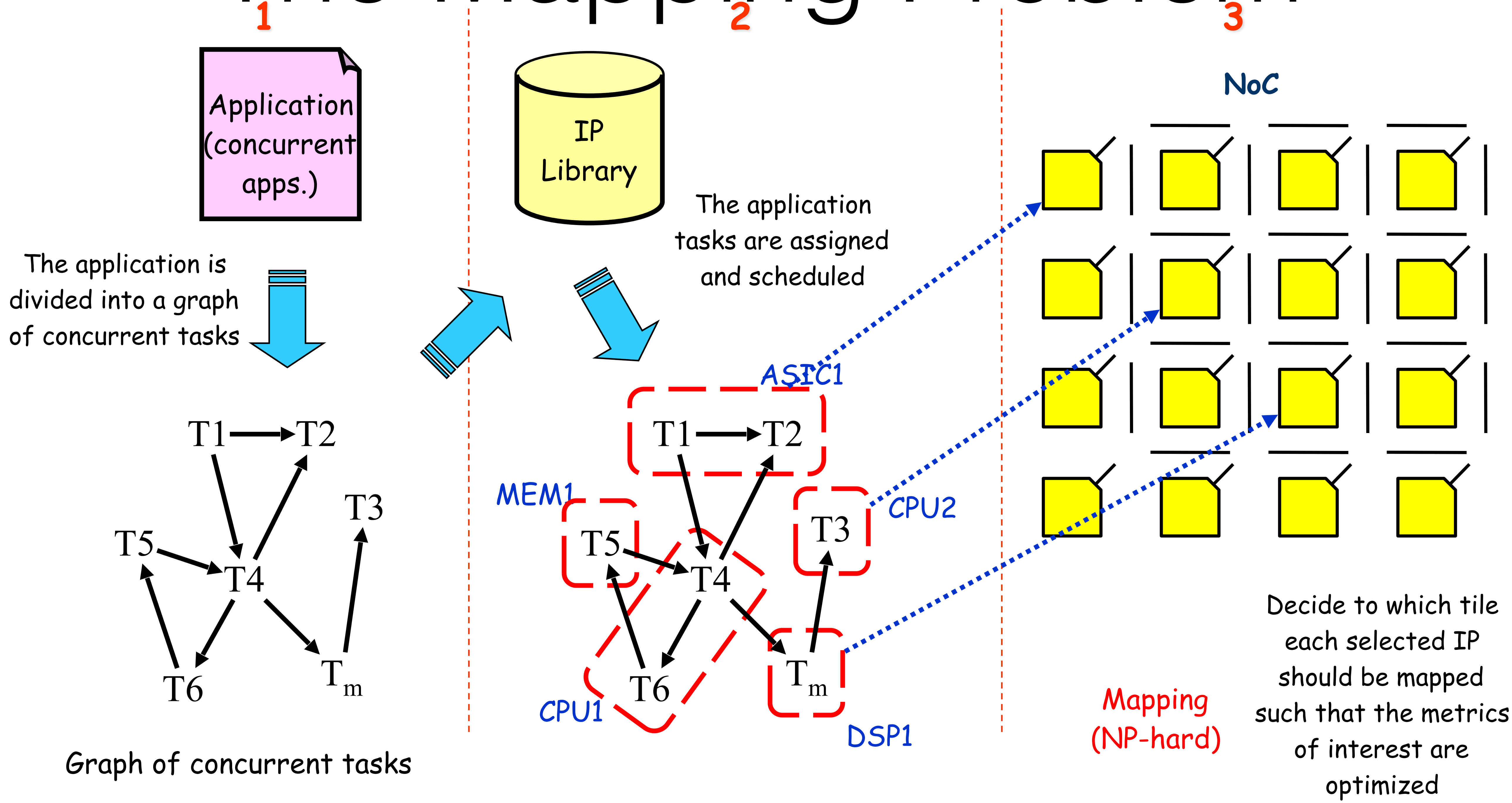


DSE for NoC Architectures

[Ogras *et al.*, ASAP'05]



The Mapping Problem



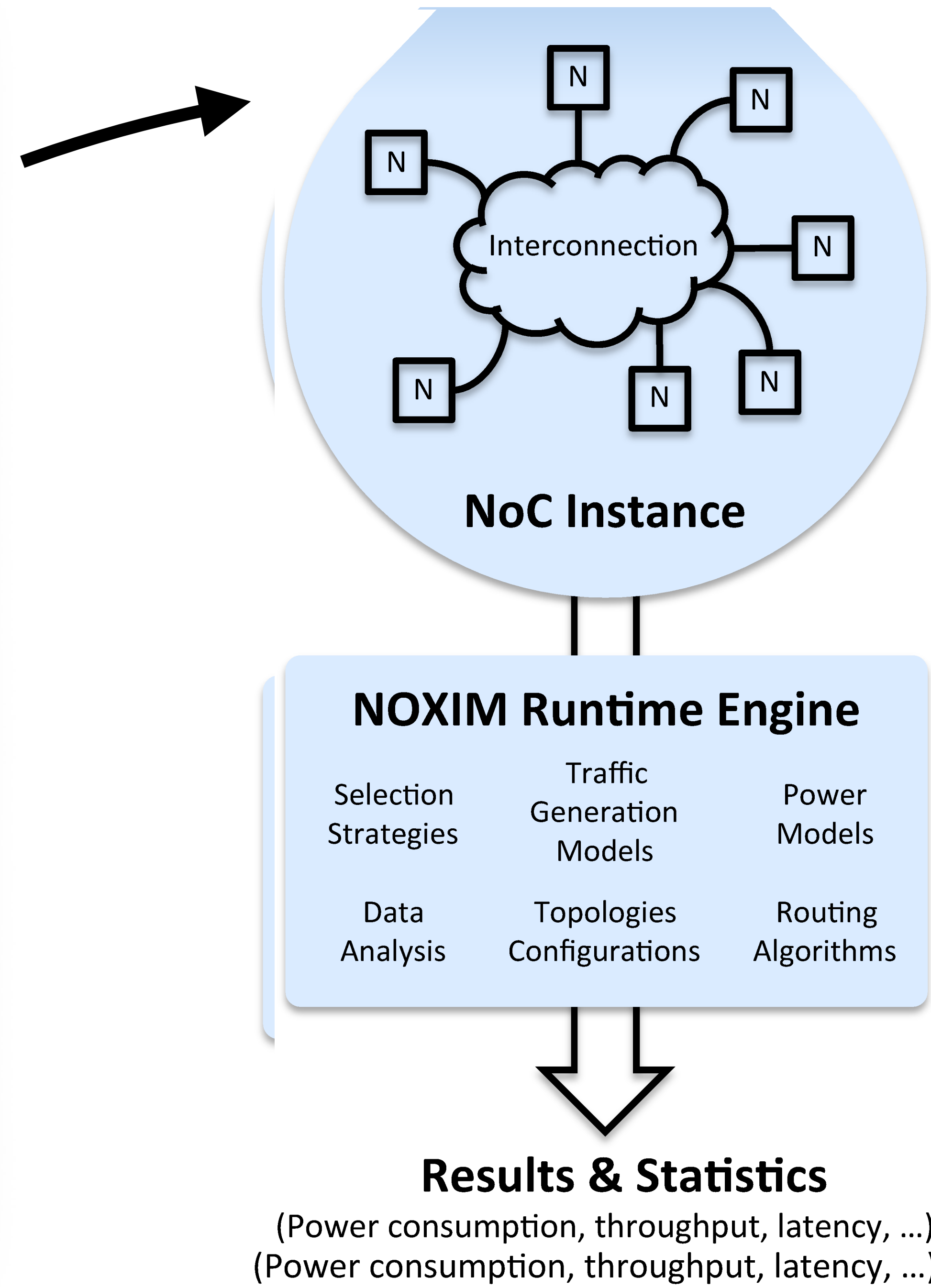
Noxim: NoC Simulator

- Cycle accurate, Open Source:
  - **`https://github.com/davidepatti/noxim`**
- SystemC signals level simulation in C++
- Performance & Power estimation
- Modular plugin-like addition of Routing/Selection strategies
- Wireless transmissions simulation

Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2016. Cycle-Accurate Network on Chip Simulation with Noxim. ACM Trans. Model. Comput. Simul. 27, 1, Article 4 (August 2016), 25 pages. DOI: https://doi.org/10.1145/2953878

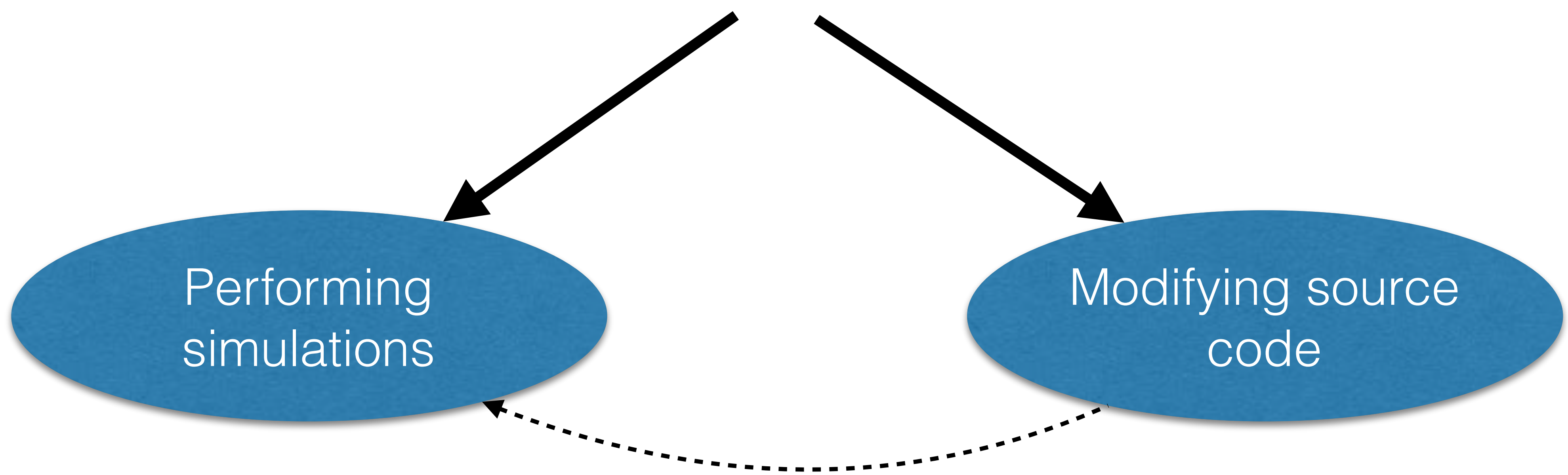
Noxim Simulation Flow

- **Topology & Structure**
  - size (number of nodes)
  - buffers
  - wireless channels
- **Workload**
  - packet size
  - injection rate
  - traffic distribution
- **Dynamic behaviour**
  - routing strategy
  - wireless access
- **Simulation Runtime**
  - duration (cycles)
  - statistics collection



What can be done?

The are two main approaches when using Noxim

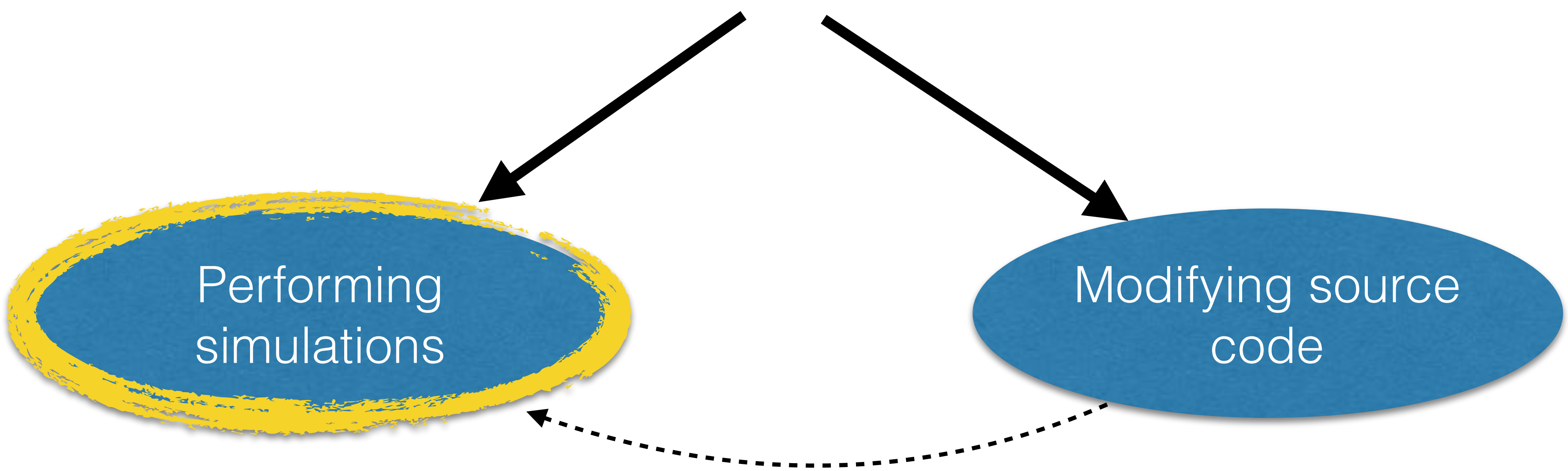


- Design Space Exploration
- Effect of Traffic Patterns
- Power Estimation
- Comparison Analysis

- New Routing Algorithms
- Different topologies
- Power Saving Strategies
- …whatever else

What can be done?

The are two main approaches when using Noxim



- Design Space Exploration
- Effect of Traffic Patterns
- Power Estimation
- Comparison Analysis

- New Routing Algorithms
- Different topologies
- Power Saving Strategies
- …whatever else

Installing Noxim

- Get an Ubuntu Linux distribution (or a virtual machine). Open the shell command line and execute (in a single line):

```
bash <(wget -qO- --no-check-certificate https://raw.githubusercontent.com/davidepatti/noxim/master/other/setup/ubuntu.sh)
```

- **Automagically does everything:** it downloads SystemC, Noxim source code, compiles everything, performs a quick simulation and the end
- **ALTERNATIVE**: (other GNU/Linux, UNIX, MaCOS), get the source code:
    - `git clone https://github.com/davidepatti/noxim.git` and follow INSTALL.txt instructions.
- Windows: it is possible. Some people did it. Not supported (by me)

Directory Structure

- **`config_examples:`** NoC configuration files, just make a copy of **`default_config.yaml`** if you want to experiment
- **`other:`** deprecated stuff, work in progress, skip it
- **`bin:`** noxim executable, results & logs
- **`doc:`** installing instructions, license etc..
- **`src:`** only go here if you want to modify Noxim

Quick & dirty Simulation

1. Open the terminal shell
2. Go to the **`noxim/bin`** directory
3. type:

```
./noxim -config ../config_examples/default_config.yaml
```

4. Enjoy results

Simple Results Format

```
Noxim simulation completed.
 ( 11000 cycles executed)
% Total received packets: 1458
% Total received flits: 11665
% Received/Ideal flits Ratio: 1.01259
% Average wireless utilization: 0
% Global average delay (cycles): 11.5528
% Max delay (cycles): 70
% Network throughput (flits/cycle): 1.29611
% Average IP throughput (flits/cycle/IP): 0.0810069
% Total energy (J): 2.12248e-06
%   Dynamic energy (J): 1.47232e-07
%   Static energy (J): 1.97525e-06
```

- Throughput in flits: number of flit per packets may change dynamically, while bits per flit usually does not (bus width)
- TIP: For a more detailed output, add -detailed to command line

Configuration File

- Cool results, but what did I simulated?
- Go to `noxim/config_examples` and open `default_config.yaml`
- See comment for parameter description. Make sure to make a backup copy of the original file when experimenting new values.
- EXERCISE 1: create a my_config.yaml, change the packet size and see the effect

Typical usage style

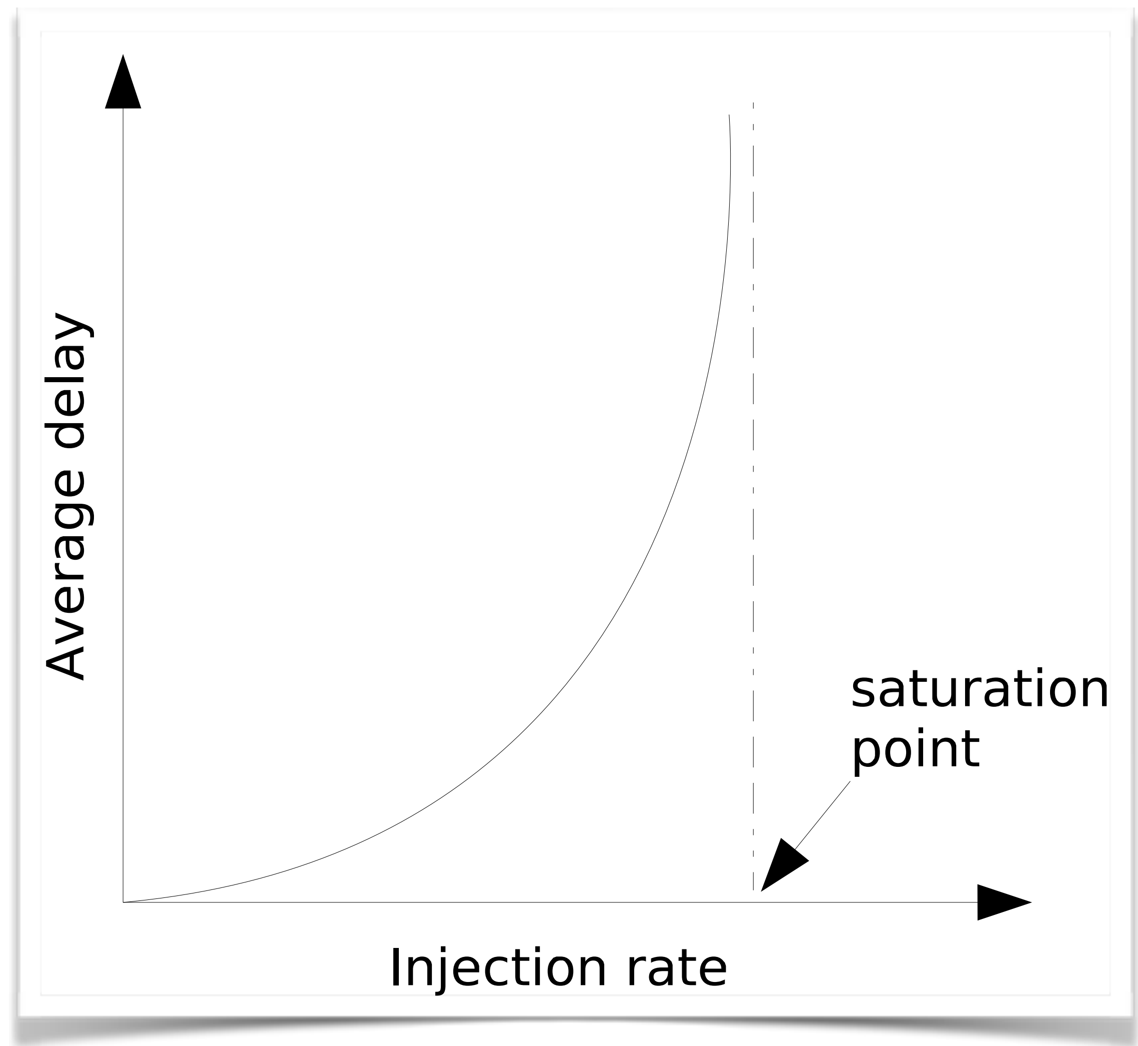
- So I have to edit a file every time? No! use command line to overwrite something at runtime (see `./noxim -help`)

ASSUMPTION: command line options overwrite those of YAML configuration file!

- You use the YAML file to set a baseline NoC configuration, i.e., most of them will remain unchanged
- Then use the command line to change the parameters you are exploring, e.g., traffic rate, simulation length, traffic pattern, routing algorithms
- EXERCISE 2: try different simulation lengths using the command line option… which option?

Example: testing saturation

- Let’s sat you have a NoC and want to test the **saturation point**
- It’s a common practice: saturation point measures the “quality” of a NoC
- Many things can lead to congestion: buffers too small, inefficient routing of packets, of simply too much workload
- Beyond saturation results are not useful anymore (e.g. router power consumption get lower, but only because they cannot move anything)



Checking Saturation

```
./noxim -config ../config_examples/default_config.yaml  -pir 0.001 poisson

…
% Global average delay (cycles): 7.45455
% Max delay (cycles): 24
% Network throughput (flits/cycle): 0.136889
% Average IP throughput (flits/cycle/IP): 0.00855556
```

- The blue numbers seems to be good, but they only tell the story of what have been delivered
- **Throughput**: how the network is able to process the requested packet injection rate (pir)?
- We have set a pir to 0.001 (1 packet every 1000 cycles). Since packets are set to 8 flits size (see YAML config), the resulting **0.008 flits/node/cycle corresponds to 0.001 packet/node/cycle of througthput, i.e. similar to the requested pir.**
- Numbers don't match exactly, because "pir" is a probabilistic value (longer sims).

Checking Saturation

- **Pir 0.5 —> Insanely saturated scenario, nothing make sense anymore**

```
% Total received packets: 4766
% Total received flits: 38098
% Received/Ideal flits Ratio: 0.0661424
% Average wireless utilization: 0
% Global average delay (cycles): 5100.47
% Max delay (cycles): 9443
% Network throughput (flits/cycle): 4.23311
% Average IP throughput (flits/cycle/IP): 0.264569
% Total energy (J): 2.4664e-06
%  Dynamic energy (J): 4.9115e-07
%  Static energy (J): 1.97525e-06
```

- So, why should I mess things to go into congestion? —-> The breakpoint at which begins to happen is a measure of “quality”

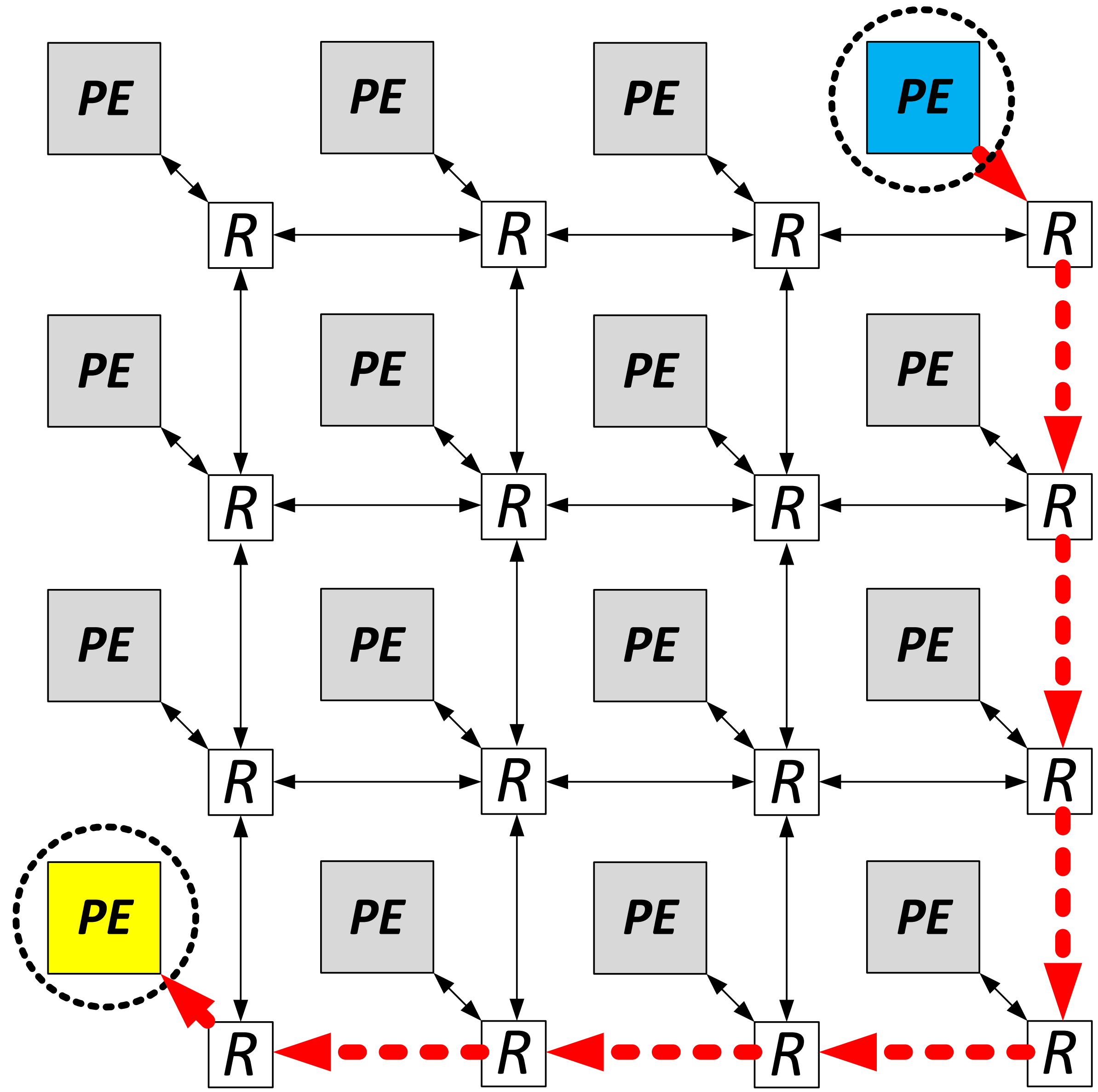
- **Congestion checklist —>** (1) huge average delay (2) throughput far less than expected (3) increasing sim length make things worse

- EXERCISE 3: Starting from the 0.5 value, decrease the packet injection rate to find the congestion breakpoint

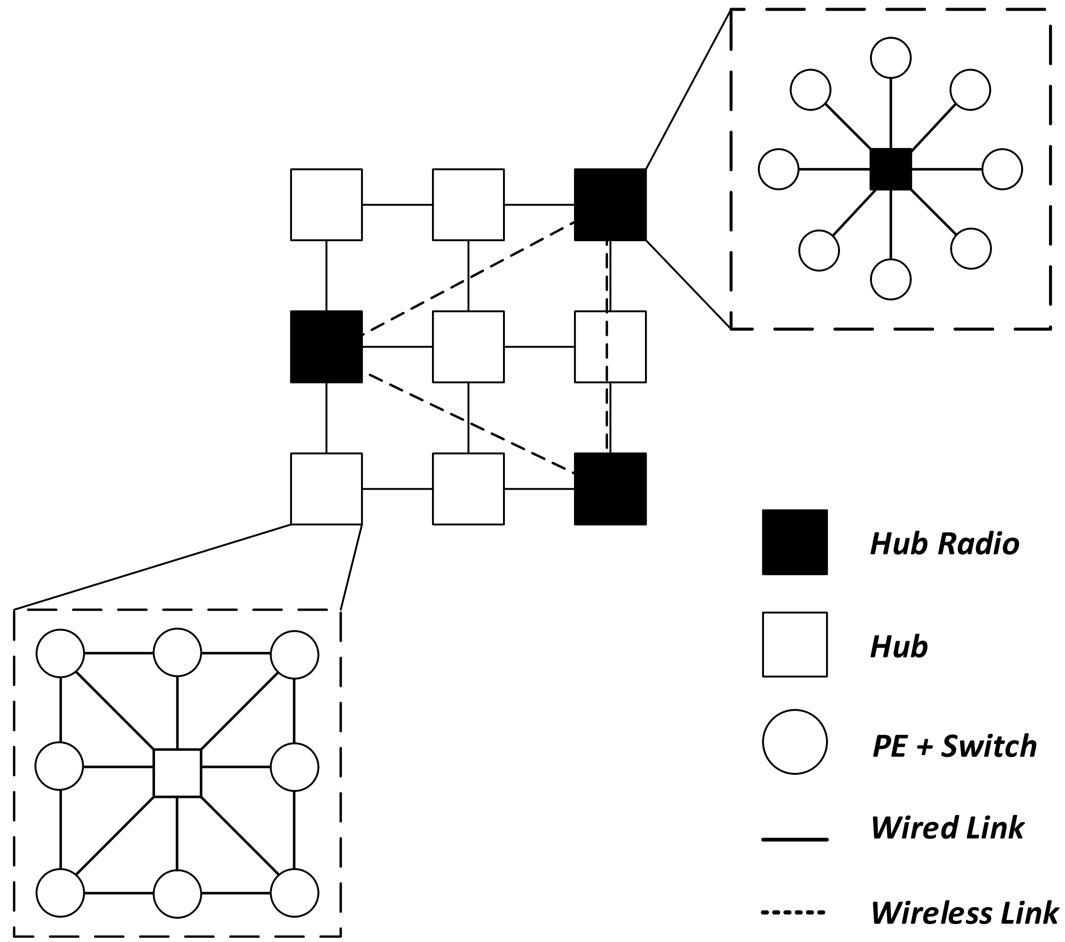
Wireless Motivation

$$D = 2\left(\sqrt{n} - 1\right)$$

$$E_{flit} = n \cdot E_{switch} + n \cdot E_{link}$$



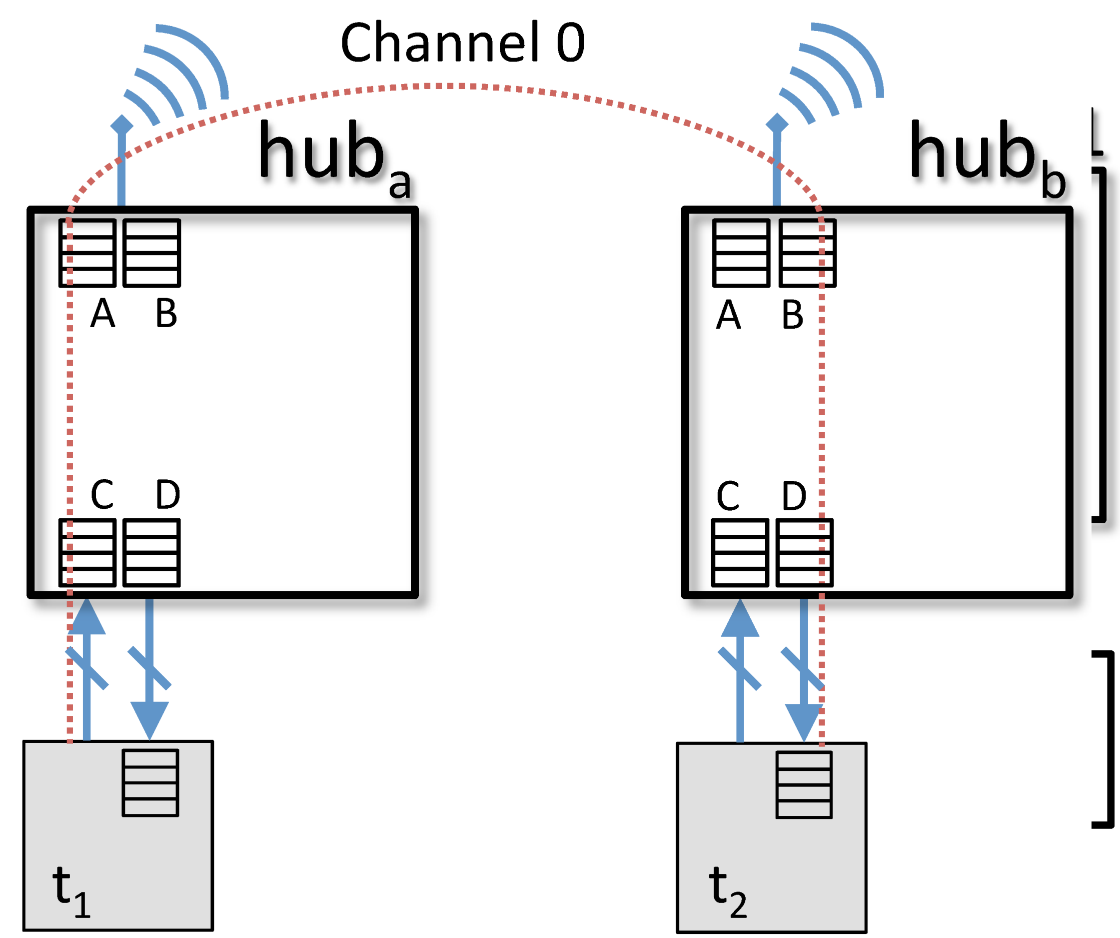
Wireless NoC Architecture



S.Deb *et al, "Wireless NoC as Interconnection Backbone for Multicore Chips: Promises and Challenges",* IEEE Journal on Emerging and Selected Topics In Circuits And Systems, Vol. 2, No. 2, June 2012

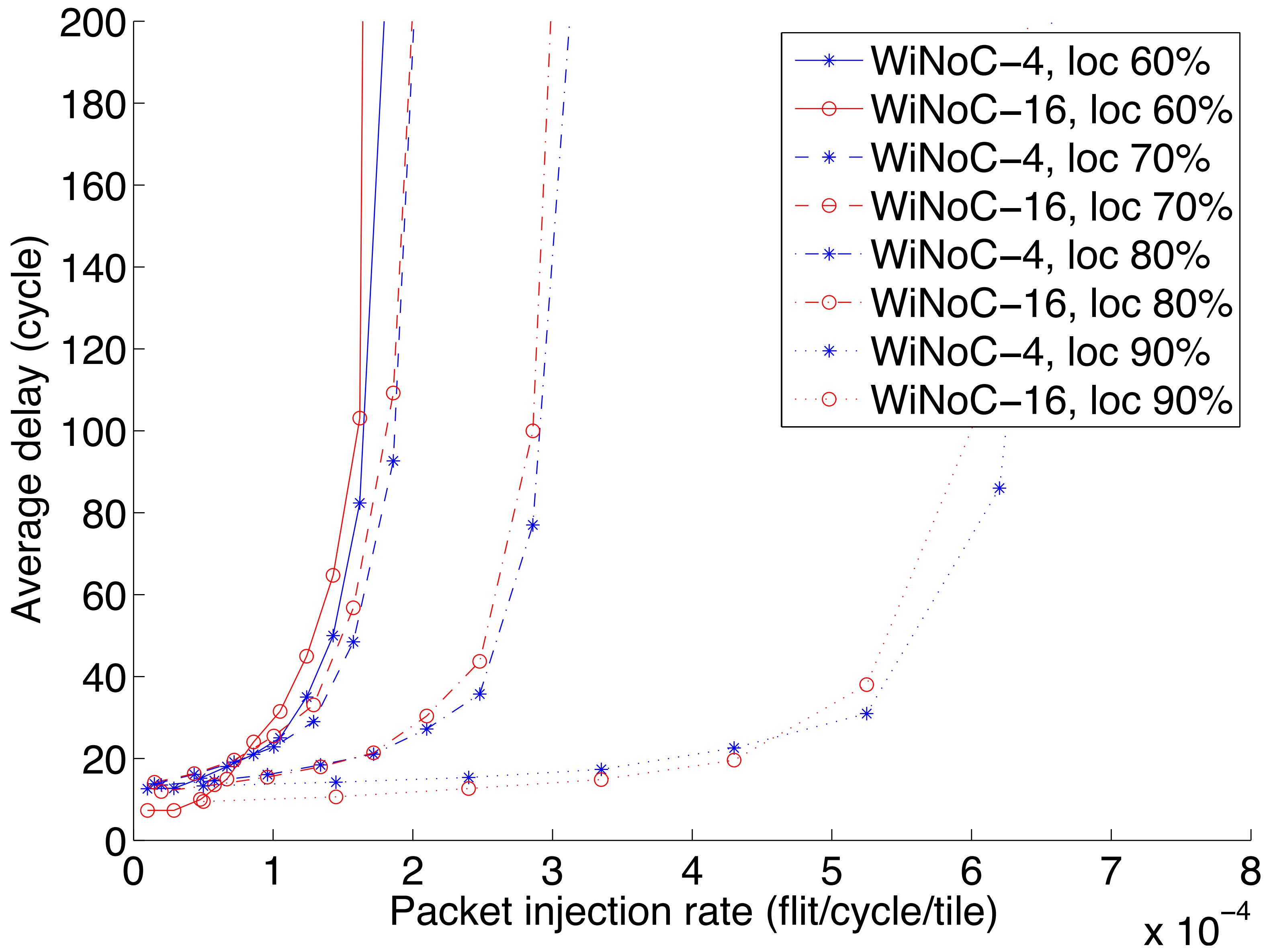
Wireless Connections

- **Tile-Hub** connections are the usual wired connections
- **Hub-Hub** connections are wireless: no physical wire to be modelled
- SystemC Transaction Level Modelling (TLM) has been chosen: account the flit transmission delay using data rate, flit length etc…
- (A) antenna_buffer_TX, (B) antenna_buffer_RX, (C) buffer_from_tile, (D)buffer_to_tile[i]



- TIP: wireless makes no sense with default small 4x4 network…take a look to more complex 256_*.yaml configurations
- EXERCISE 4: set an hub-tile interconnection in bigger 256 nodes YAML config file and enable -winoc option

Delay vs Locality



Power Estimation

- **Additive model:** each component has been implemented in HDL and average dynamic/static power has been estimated
- At each cycle the energy of a given element “e”:

$$E(e,c) = \alpha(e,c) \times P_{avg}(e) \times T_{CK}$$

- Where $P_{avg}(e)$ is the average dynamic power and $\alpha(e,c)$ is the activity function: 0 if e is not active in cycle c, 1 otherwise
- Static power is added without caring about activity
- TIP: Checkout numbers in noxim/bin/power.yaml
- EXERCISE: enable -detailed to get power breakdown. Choose one of the values and try to change sim configuration to increase or decrease it

Enabling Logs

- edit `noxim/bin/Makefile`

- remove the “#” in DEBUG line (uncomment)

```
DEBUG    :=  -g -DDEBUG
```

- `make clean`

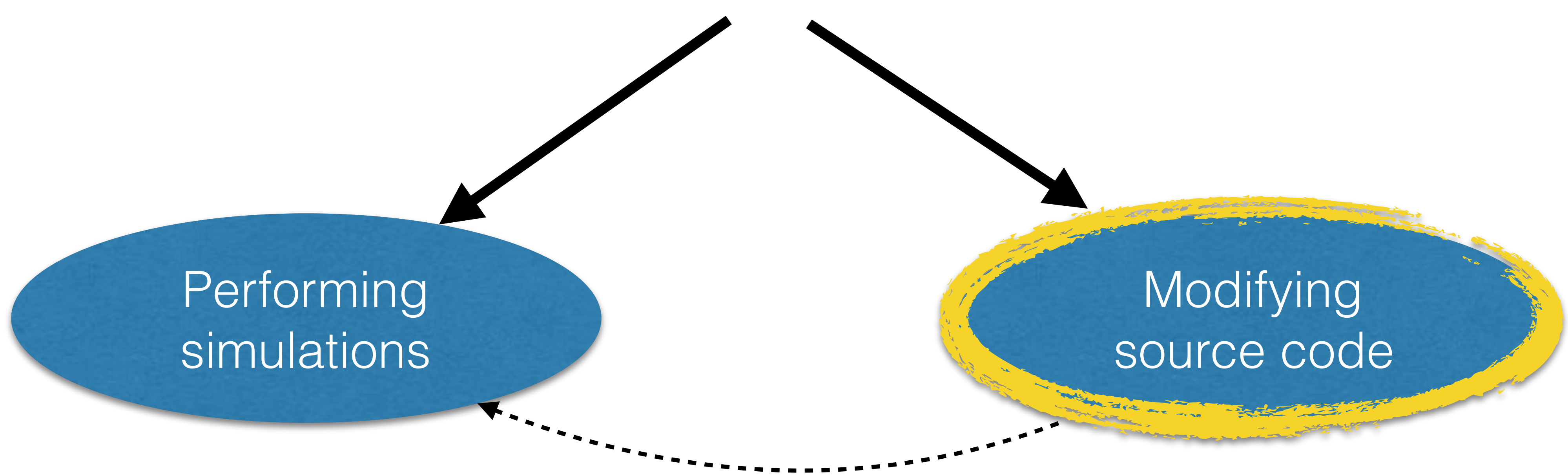
- `make`

- Redirect the output to some file:

```
./noxim -config ../config_examples/default_config.yaml > my_log.txt
```

What can be done?

The are two main approaches when using Noxim



- Design Space Exploration
- Effect of Traffic Patterns
- Power Estimation
- Comparison Analysis

- New Routing Algorithms
- Different topologies
- Power Saving Strategies
- …whatever else

Modifying Noxim

- Prerequisites: C++, SystemC (hardware related parts)
- Edit source code is in **`noxim/src`**
  - …but launch "make" from **`noxim/bin`**
  - **`sc_main()`** in **`Main.cpp`**: where everything starts…
- EXERCISE 5: just write something before simulation starts in main.cpp and recompile
- TIP: To understand what a piece of code is about, always start by looking *.h (header files)

Noxim Source Files

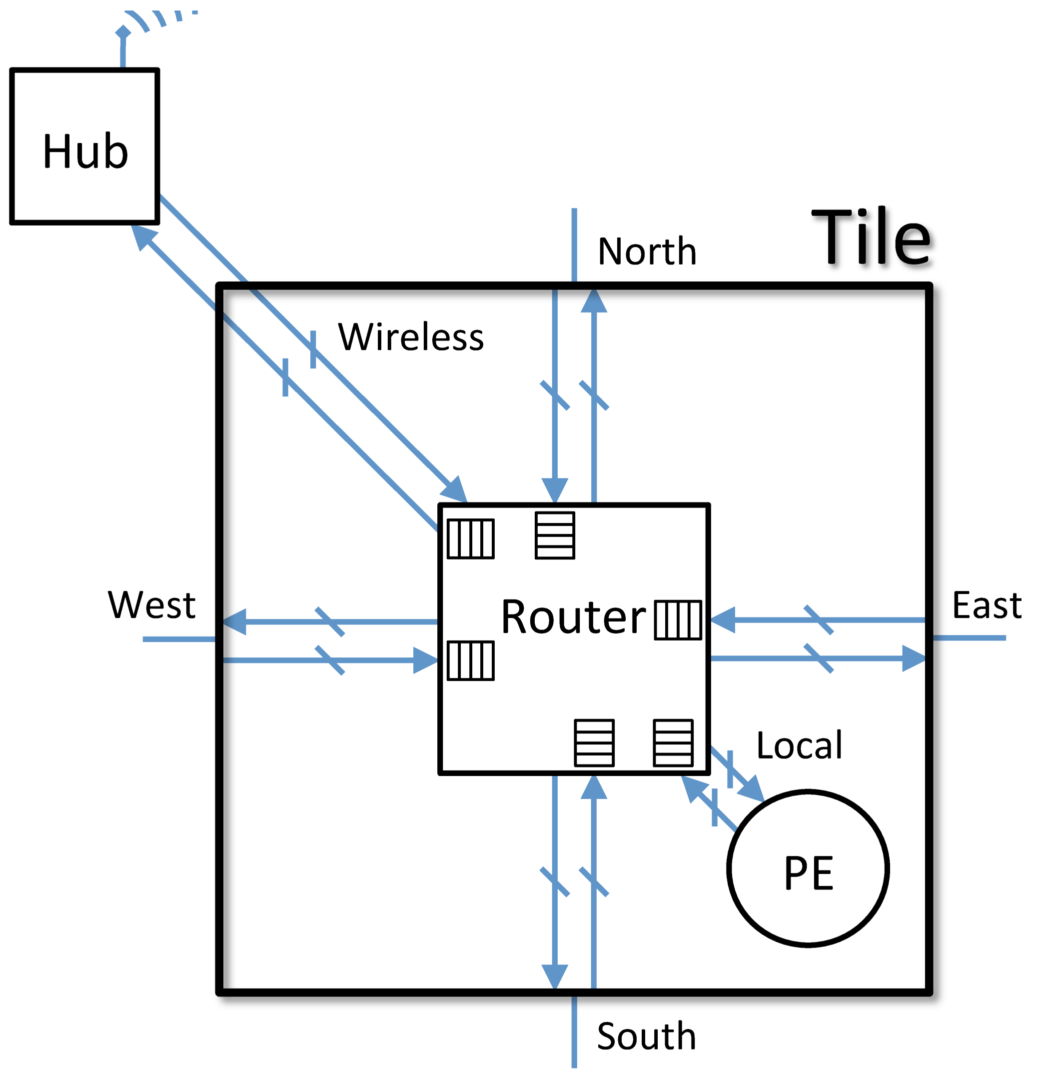
- **Architectural Elements:** `Buffer.h, Router.h, LocalRoutingTable.h, Tile.h, NoC.h, GlobalRoutingTable.h, Power.h, ProcessingElement.h`
- **Simulator Management & Utilities:** `ConfigurationManager.h, DataStructs.h, GlobalStats.h, Stats.h, Utils.h, GlobalParams.h,GlobalTrafficTable.h ReservationTable.h`
- **Wireless Communication:** `Initiator.h, Target.h, Hub.h, TokenRing.h, Channel.h`
- **Routing algorithms can be found in:** `src/routingAlgorithms`

- For each of these `*.h`, actual implementation is in `.cpp`

- TIP: The *.h files defining a "`SC_MODULE`" are related to NoC hardware components

Architecture/Source Code

- Default scenario: Mesh of Tiles nodes
- Each Tile contains a Router and a Processing Element
- Each Tile is connected to the 4 neighbours
- Optionally, some nodes can be connected to Radio-hub allowing wireless transmissions



- EXERCISE 6: (re)discover the hierarchy Tile = (Router + PE) in the code

Five quick Rules for SystemC Hardware components

1. `"SC_MODULE"` in header `*.h` file —> **ok, it's hardware**

2. **Interconnection wires?** Look at `sc_in<…>` and `sc_out<…>`

3. **What happens at each clock cycle?** checkout for "sensitive" "`SC_METHOD`"s inside the constructor

```
SC_CTOR(Router) {
    SC_METHOD(rxProcess);
    sensitive << clock.pos();
```

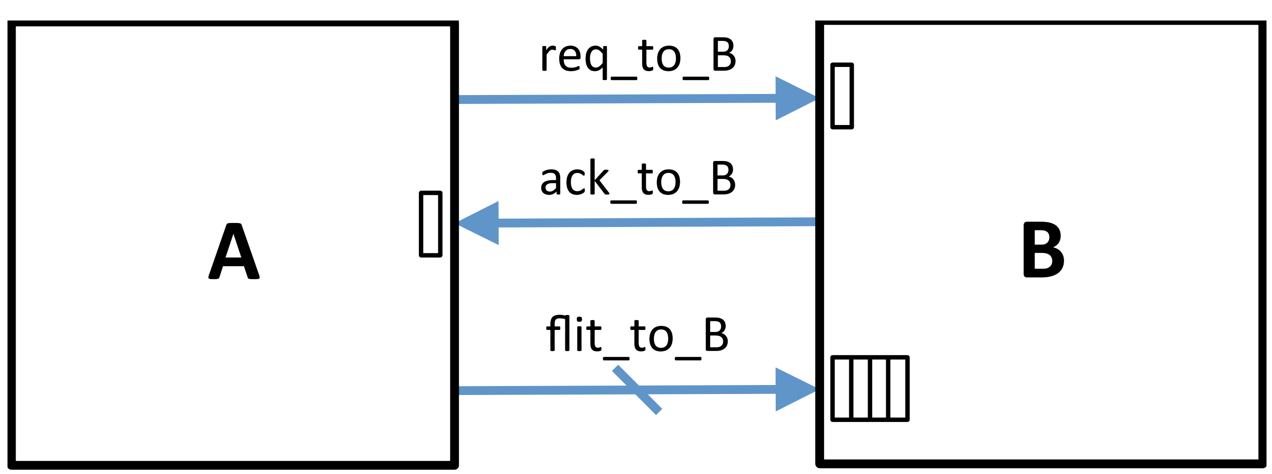
4. Do not assume any temporal ordering between sensitive methods in a clock cycle. **All values at the end of the cycle should be coherent!**

5. **Who writes/reads wires?** go to `*.cpp`, search sensitive methods implementation and how the change I/O wires with `read()/write()` calls

EXERCISE 7: Analyse Router component according to these rules

3 Wired Connections?

- Cycle accuracy and Signal level simulation required more than simple function call
- Each connection requires 3 signals, not one: two booleans ("req/ack") are for protocol, the other one for the actual data
- TIP: Look at this last one signal to "understand" the interconnection
- EXERCISE 8: Choose a logical interconnection in Tile.h and find its 3 signals inside the code...



From Tile.h

```
SC_MODULE(Tile)
{
…..
    Router *r;
    ProcessingElement *pe;

    sc_in <Flit> flit_rx[DIRECTIONS];
    sc_in <bool> req_rx[DIRECTIONS];
    sc_out <bool> ack_rx[DIRECTIONS];

    sc_out <Flit> flit_tx[DIRECTIONS];
    sc_out <bool> req_tx[DIRECTIONS];
    sc_in <bool> ack_tx[DIRECTIONS];
```

What else?

- `DataStructs.h:` is the file containing conceptual data structures of Noxim: Packet, Flit, Coordinates etc..
    - EXERCISE 9: Add to the packet format a bool field invented by you!
- `GlobalParams.h:` where all running configuration is stored, accessible from everywhere in the code, e.g. `Globalparams::my_parameter`
    - TIP: to add your own simulation parameter, use the most similar one as example (i.e. a number or a string?).
    - EXERCISE 10: create a new parameter and add it to `GlobalParams.h`, `ConfigurationManager.cpp`, `GlobalParams.cpp` and `my_config.yam`

GlobalParams::

- You'll find these double nested in several places, "do this for every tile node)

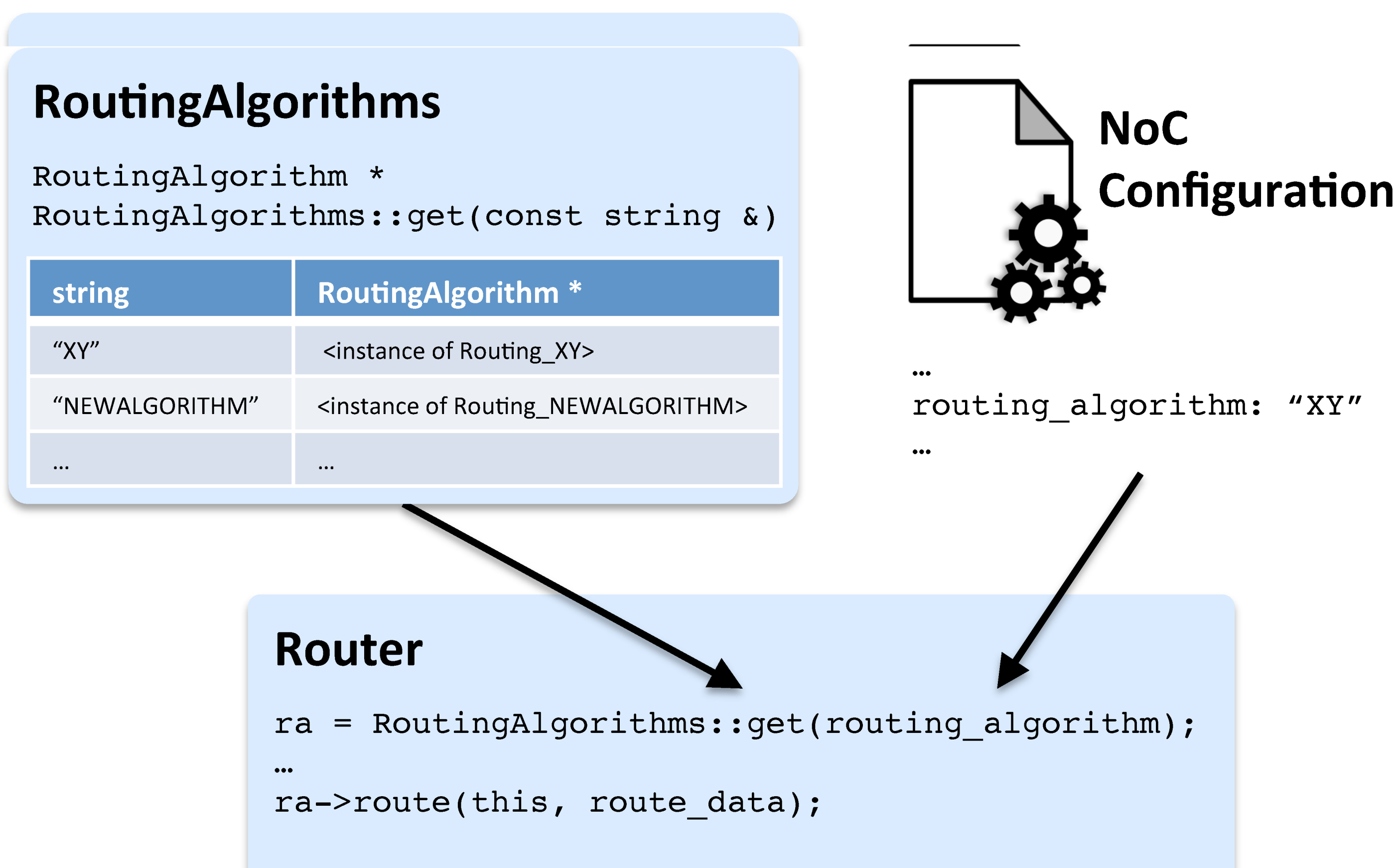
```
for (int y = 0; y < GlobalParams::mesh_dim_y; y++)
    {
        for (int x = 0; x GlobalParams::mesh_dim_x; x++)
            {
```

- Everywhere in the code

```
if (GlobalParams::some_parameter == some_value)
{
   do …
```

Adding a Routing Algorithm

- Looking at `Router.cpp` implementation, a generic `route()` function is invoked.
- The actual code executed depends on configuration



Routing "PIZZA" in 6 Steps

1. Go to the **`noxim/src/RoutingAlgorithms`** source directory and create your files from other existing ones:

```
cp Routing_XY.cpp Routing_PIZZA.cpp
cp Routing_XY.h Routing_PIZZA.h
```

2. Edit **`Routing_PIZZA.h Routing_PIZZA.cpp`** and replace every occurrence of **`XY`** with **`PIZZA`**

3. Put your implementation logic of PIZZA in **`Routing_PIZZA.cpp:`**

```
Routing_PIZZA::route(Router * router, const RouteData & routeData) {
```

5. Edit **`power.yaml`** to put static & dynamic power cost of doing **`PIZZA`**

6. Go to **`noxim/bin`** and recompile by typing "make": notice how only your code is recompiled!

Future Works

- Parallelisation of Noxim. Mapping SystemC threads to different host-machine threads [*Roth et al. 2013; Sinha et al. 2012*]
- More advanced trace traffic support based on real application traffic streams
- Wireless Broadcast support
- Replace Alternate Bit Protocol
- Analytic proof of deadlock freedom for all the supported topologies
- Virtual Channels (done better…)

Bibliography

- Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2016. Cycle-Accurate Network on Chip Simulation with Noxim. ACM Trans. Model. Comput. Simul. 27, 1, Article 4 (August 2016), 25 pages. DOI: https://doi.org/10.1145/2953878

- C. Roth, H. Bucher, S. Reder, F. Buciuman, O. Sander, and J. Becker. 2013. A SystemC modeling and simulation methodology for fast and accurate parallel MPSoC simulation. In Integrated Circuits and Systems Design (SBCCI), 2013 26th Symposium on. 1–6. DOI:http://dx.doi.org/10.1109/SBCCI.2013.6644853

- R. Sinha, A. Prakash, and H. D. Patel. 2012. Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs. In 17th Asia and South Pacific Design Automation Conference. 455–460. DOI:http://dx.doi.org/10.1109/ASPDAC.2012.6164991