SCHOOL OF ENGINEERING
JÖNKÖPING UNIVERSITY

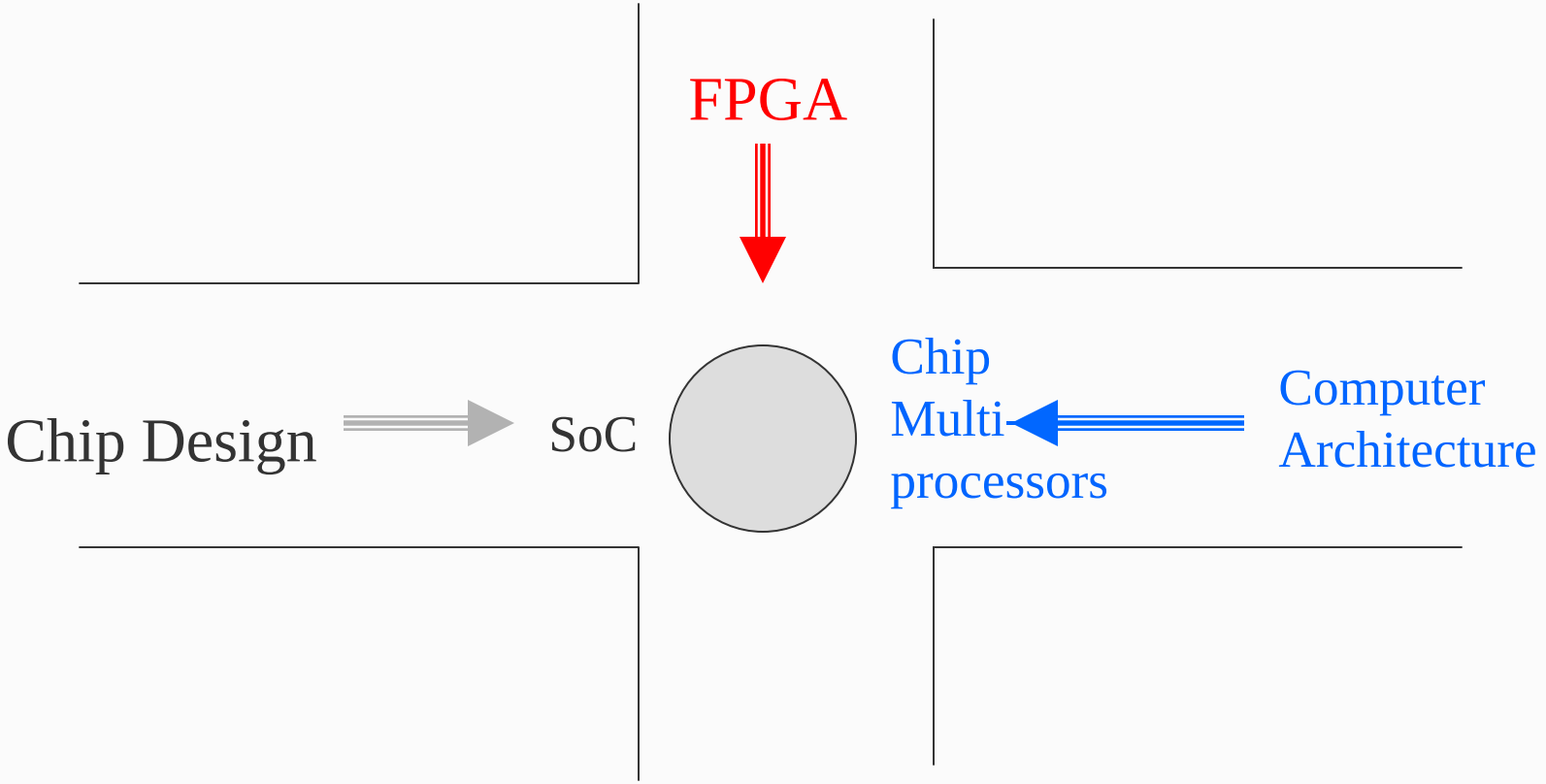
# Some issues in Application Specific Network on Chip Design

Shashi Kumar
Embedded Systems Group
Department of Electronics and Computer Engineering
School of Engineering, Jönköping University

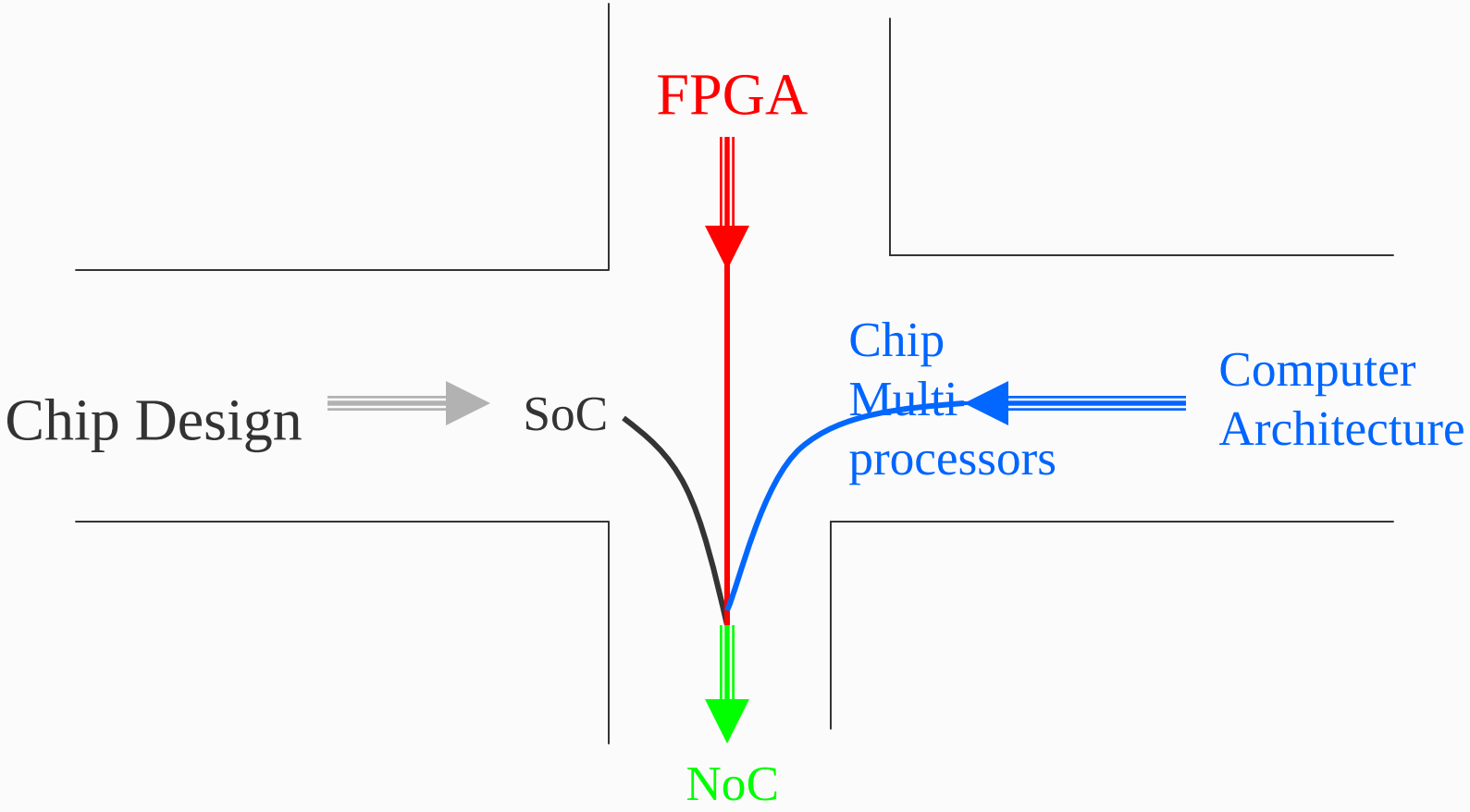
# Outline

- Introduction: rather long!
  - NoC Evolution
  - Case for Application Specific NoC
- A NoC Methodology
- Specializing NoC for an Application
  - Processor Selection and Evaluation for NoC
  - Link Optimization and QNoC
- Mapping Applications to NoC

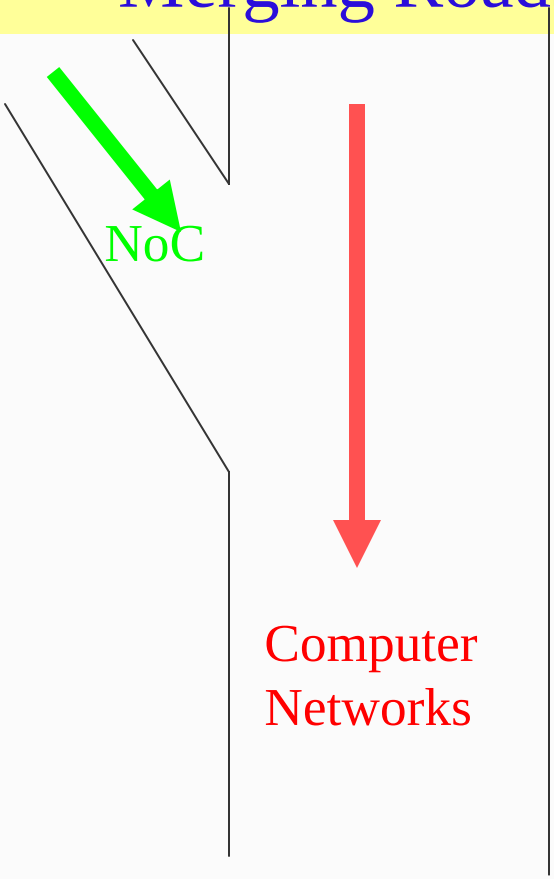
# An Interesting Cross-road



# An Interesting Cross-road



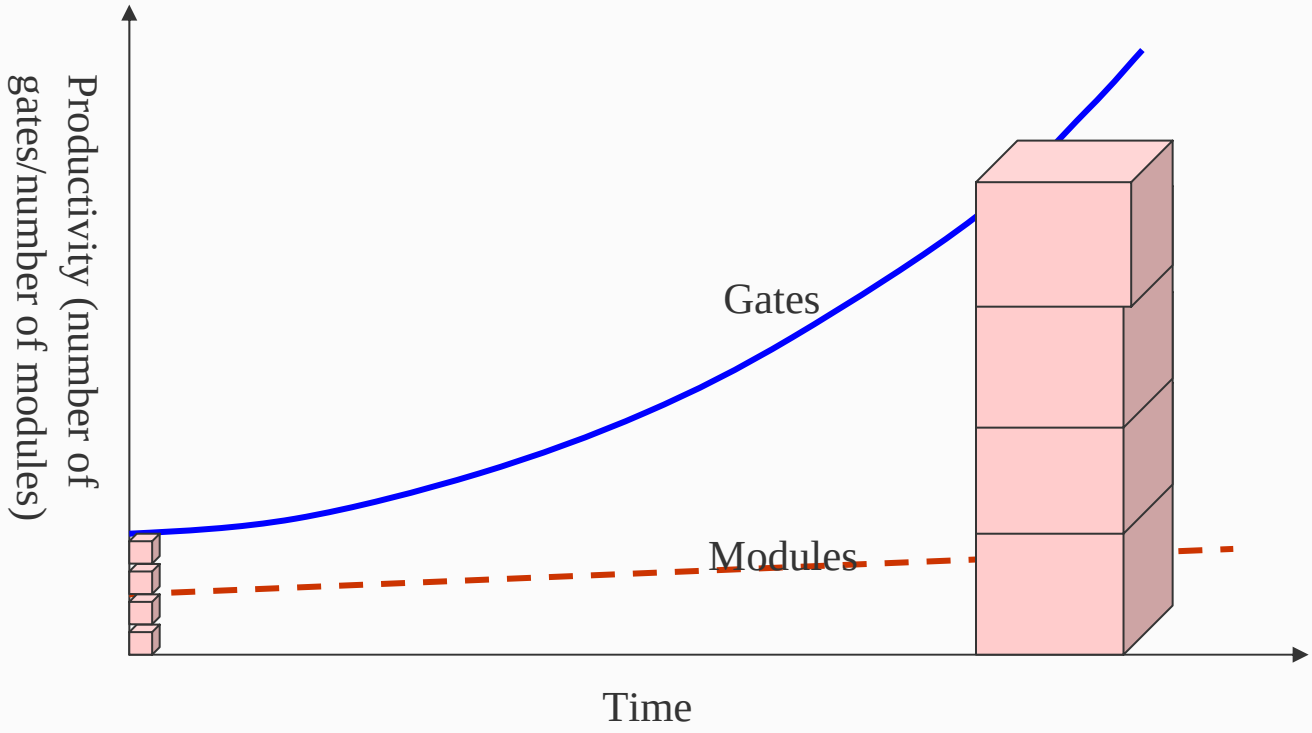
# Merging Roads



# A case for NoC

- Driving Forces
  - Exponentially increasing Silicon Capacity
  - Demand of products requiring large capacity
  - Market: Competitive in price and short Time-to-Market
- Techniques to handle large and complex system design
  - Reuse
  - Higher Level of abstraction
  - CAD tools
- NoC paradigm offers all the above features

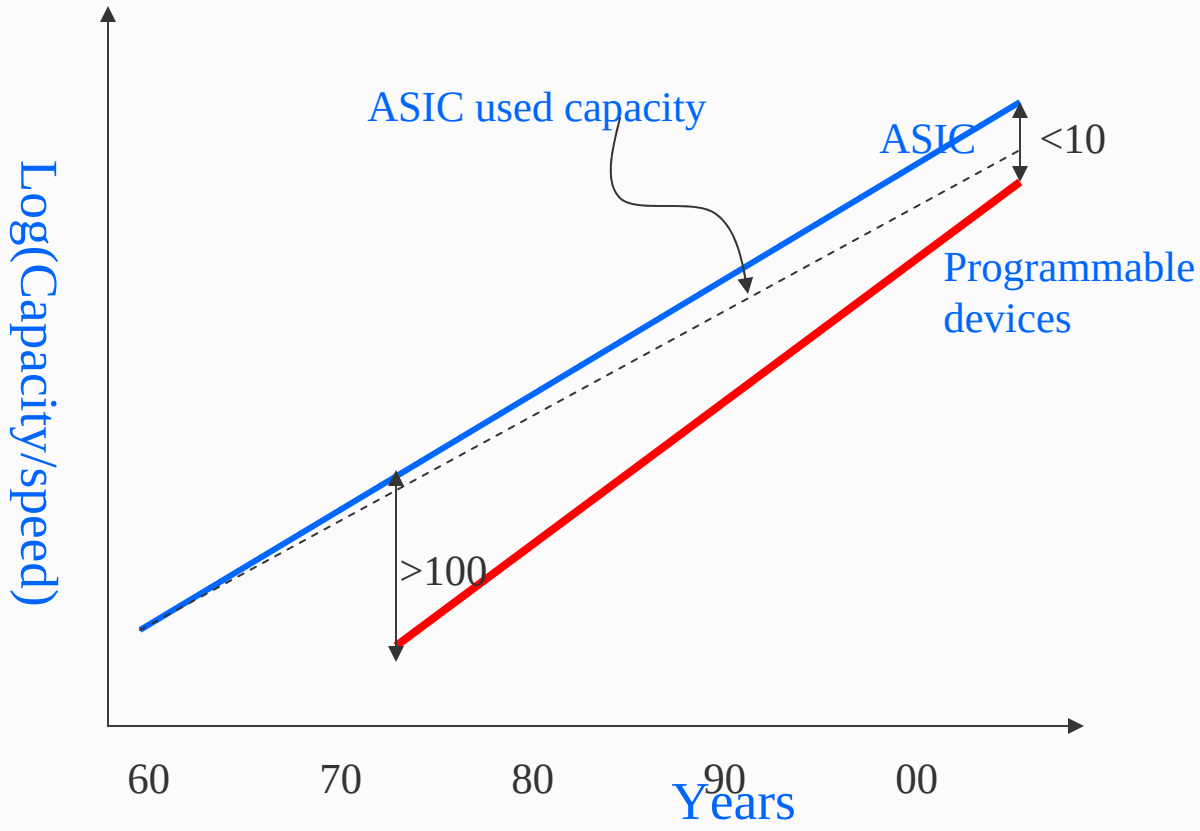
# Design Productivity and Level of Abstraction



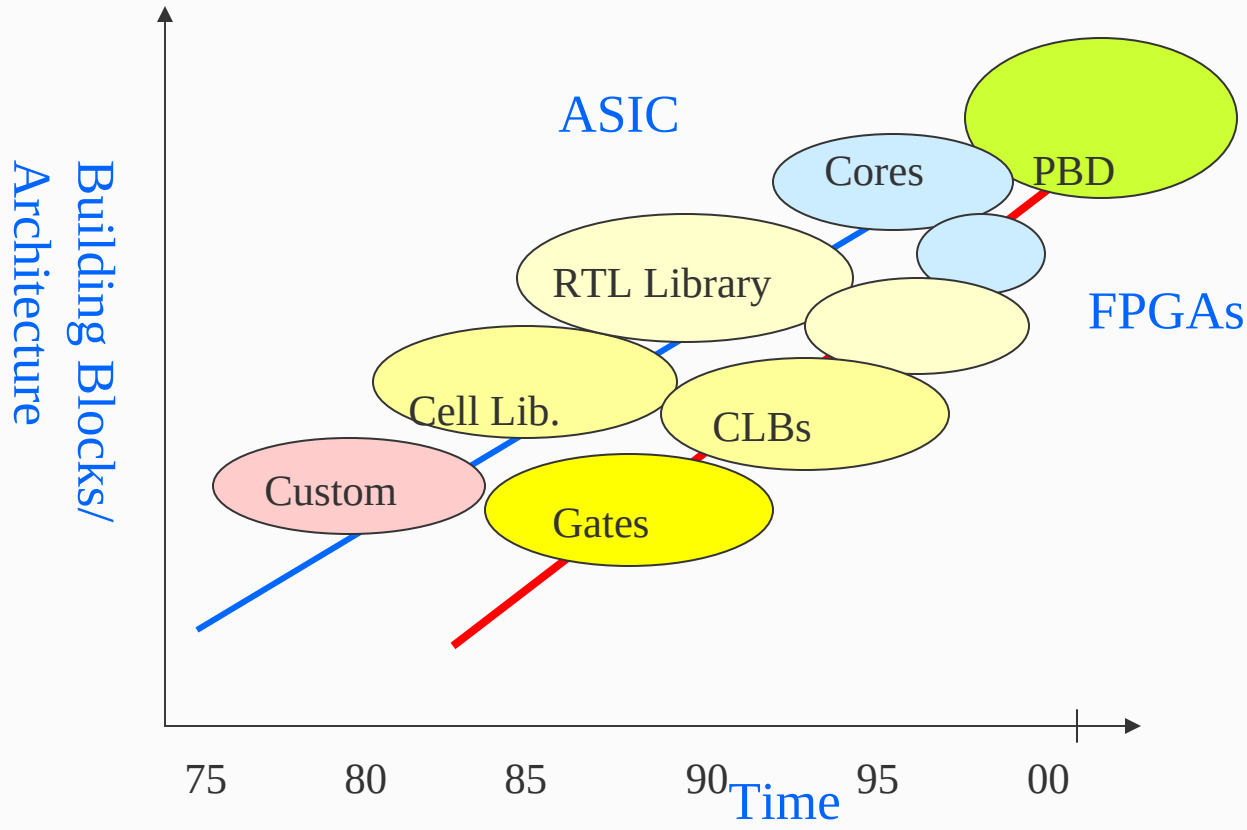
# Properties for REUSE

- Well defined external interfaces
  - Easy composability in different contexts
- Reuse means generalization which leads to under-utilization
  - Static Cost : More area
  - Dynamic cost: Consumes more power
- How to make Reuse efficient ?
  - Specialization:
    - Trim out the unnecessary resources
  - Higher utilization through programmability

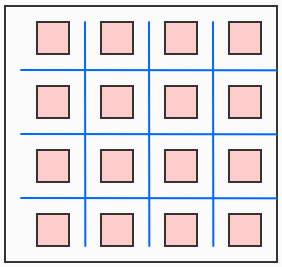
# Evolution of ASIC and Programmable Devices



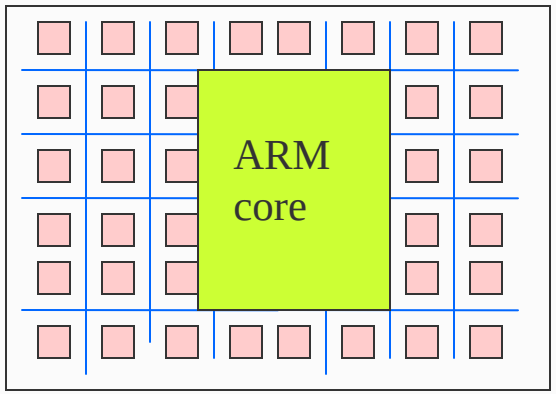
# Evolution: ASIC Vs. PLD Architectures



# Abstraction levels of programming: FPGA



- Program logic blocks and interconnections
- Bit file




- ARM core is programmable at higher level of programming

# Chip Multi-processor Road

Question: New computer architecture which can use such a high available capacity?

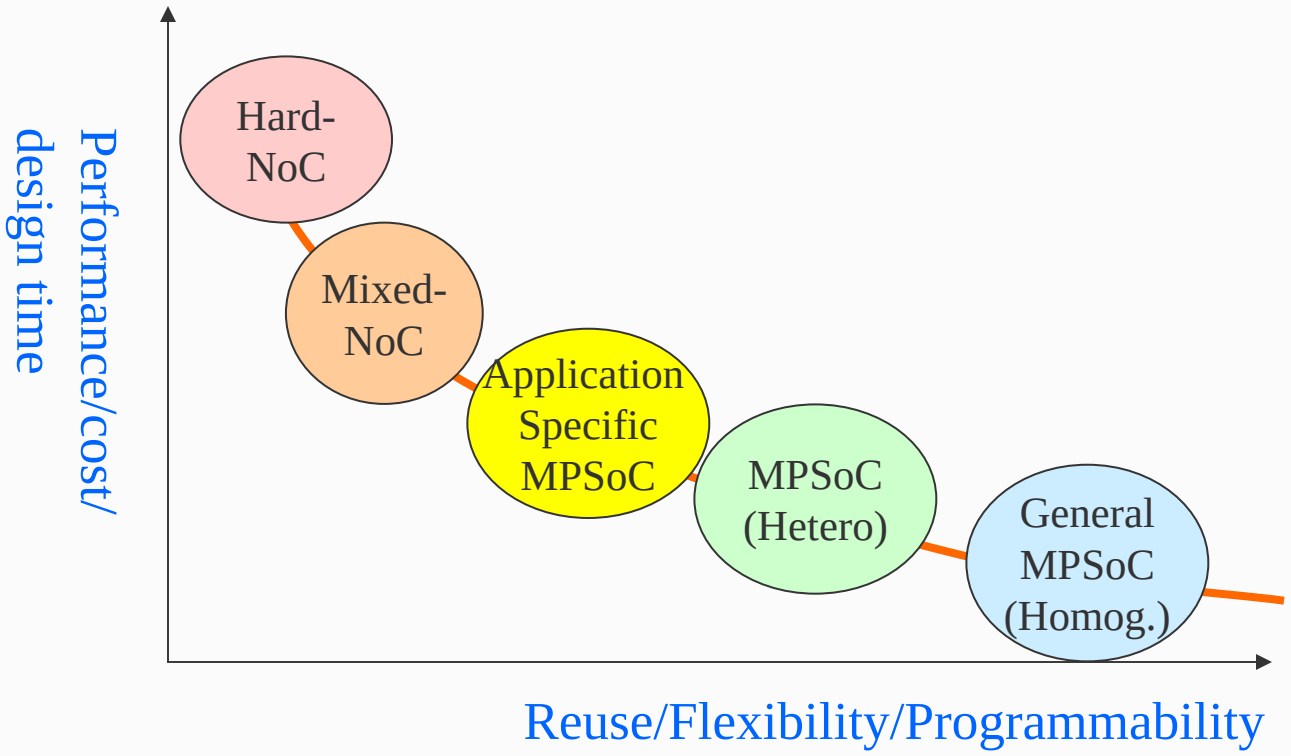
Options:

- Super-computer on a chip: Powerful instruction set; Higher accuracy, On-chip main memory, On-chip interfaces; ……
- Super-scalar Architecture
- Multi-threaded super-scalar architectures
- Multi-processor on a chip

# Multiprocessor SoC vs. NoC

- Multiprocessor SoC is a special NoC
  - All the cores are processors
    - Homogenous vs. Heterogeneous
  - Functionality of the system fully software programmable
- Other extreme : ASIC built using special function hardware IP cores
  - Functionality of the system quite fixed at the time of fabrication
- NoC covers the complete range between these extremes

# NoC Architecture Space



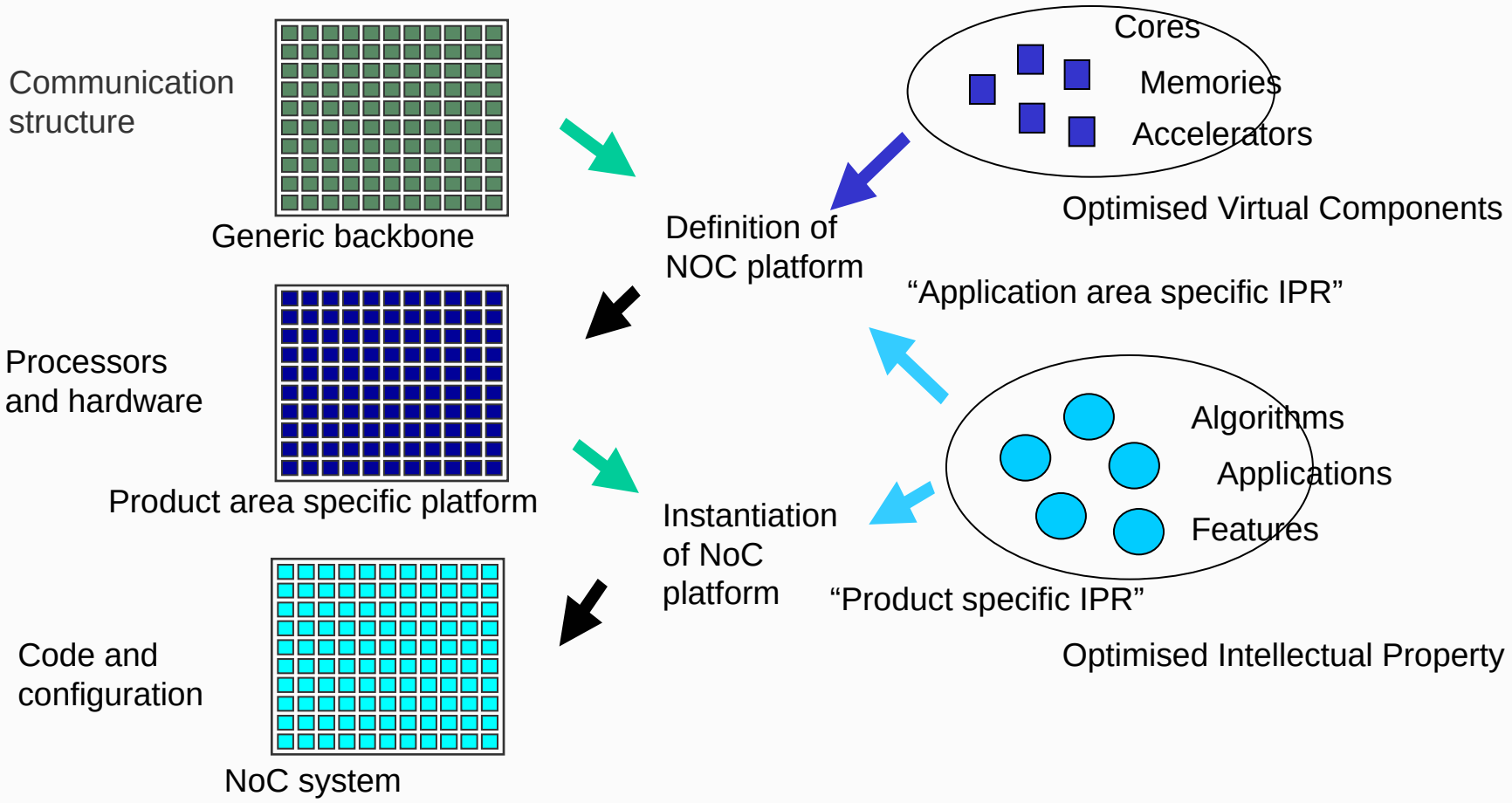
# Why Application Specific NOC ?

- Performance
  - Use of application specific cores
  - Specialize Size, Topology, Routers, Links, Protocols for the application
- Power Consumption
  - Optimal utilization of computing and communication resources
  - Optimal voltage and clock selection for computing and communication resources

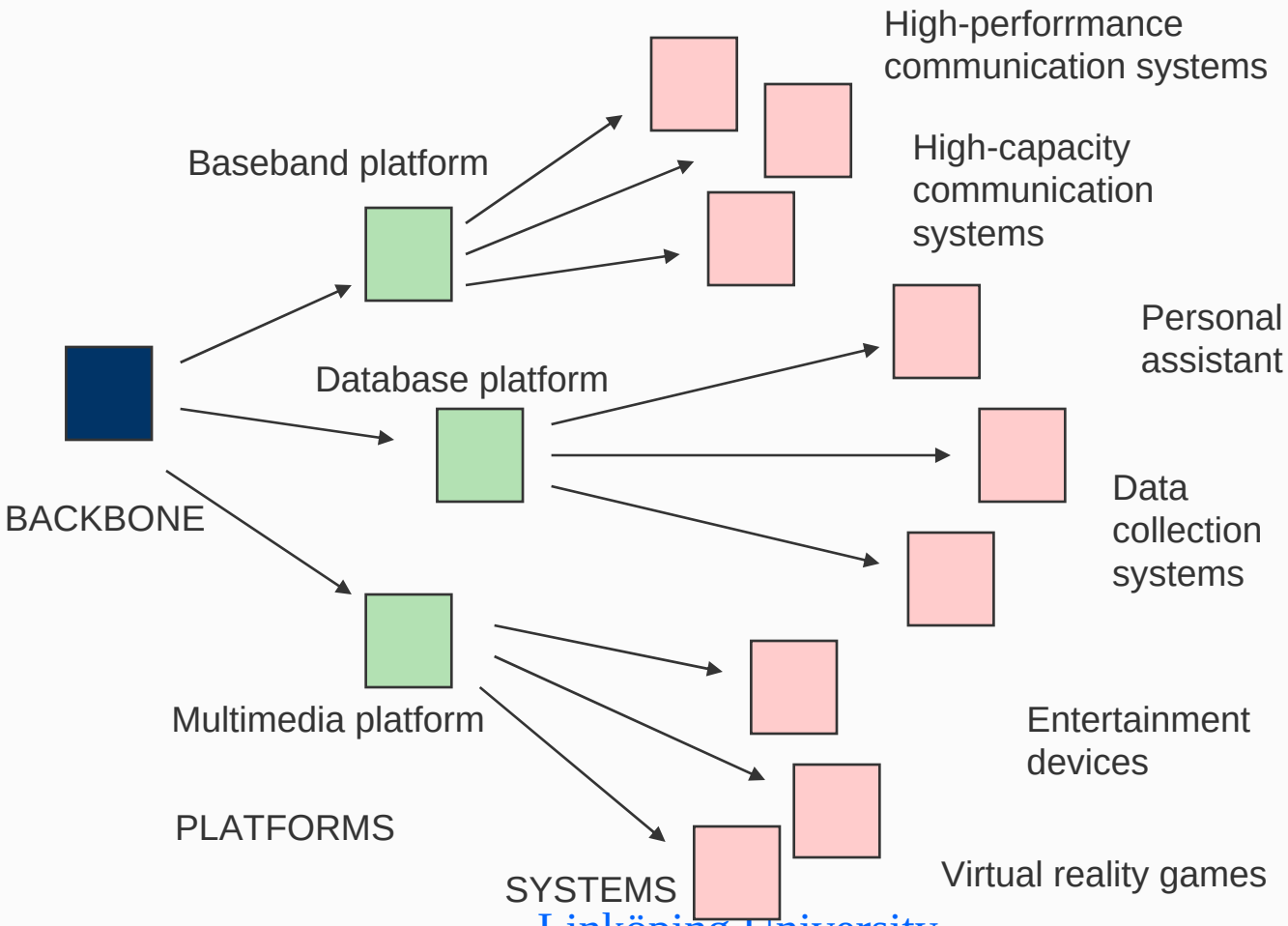
# Degrees of Freedom in Mixed NoC design

- Topology and Size
- Core Selection and Core Positioning
- Core operating Voltage and Clock Speed
- Link specialization
- Router Design
- Protocol Specialization
- OS specialization

# ASNOC Design Methodology[1]



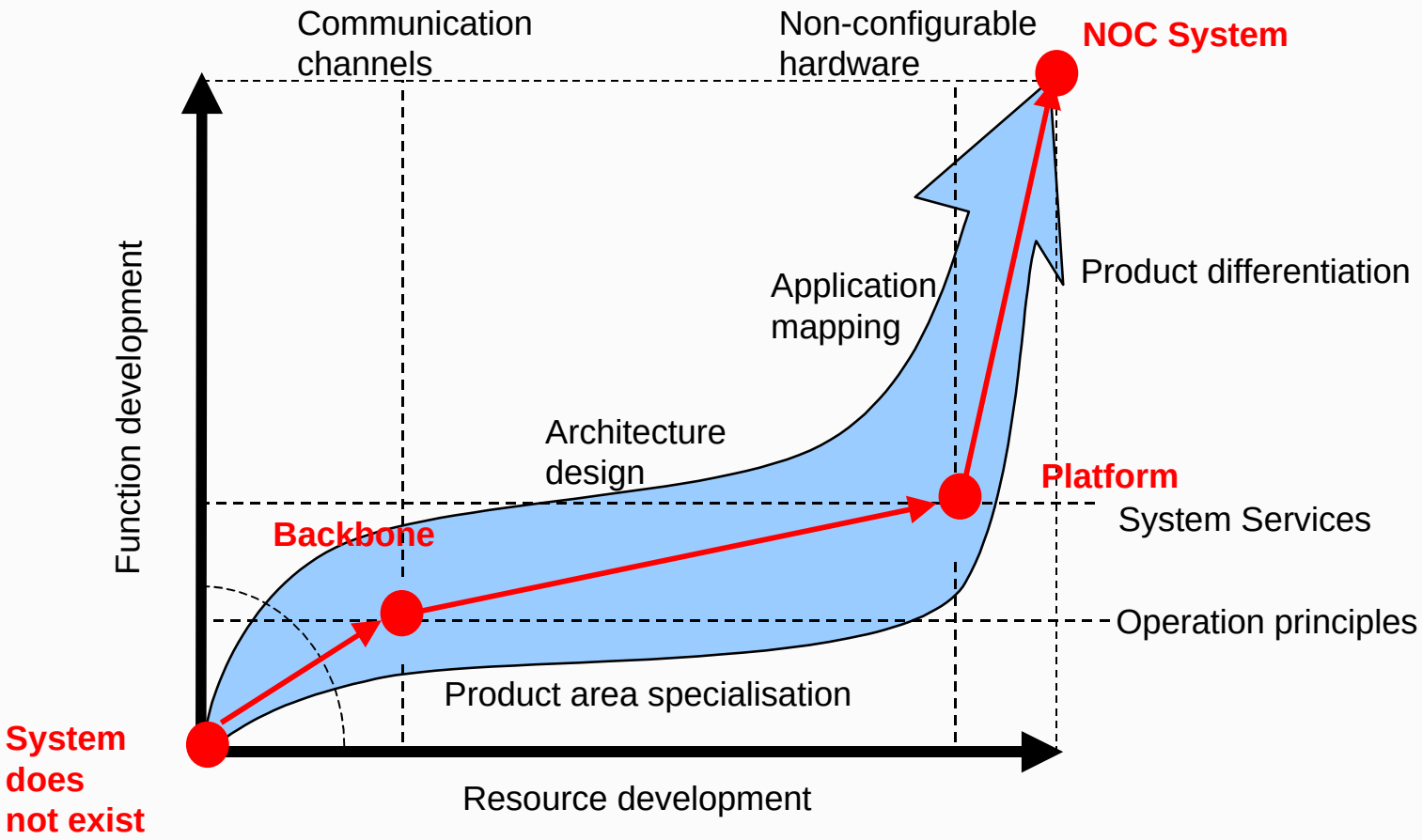
# Development of NOC based systems



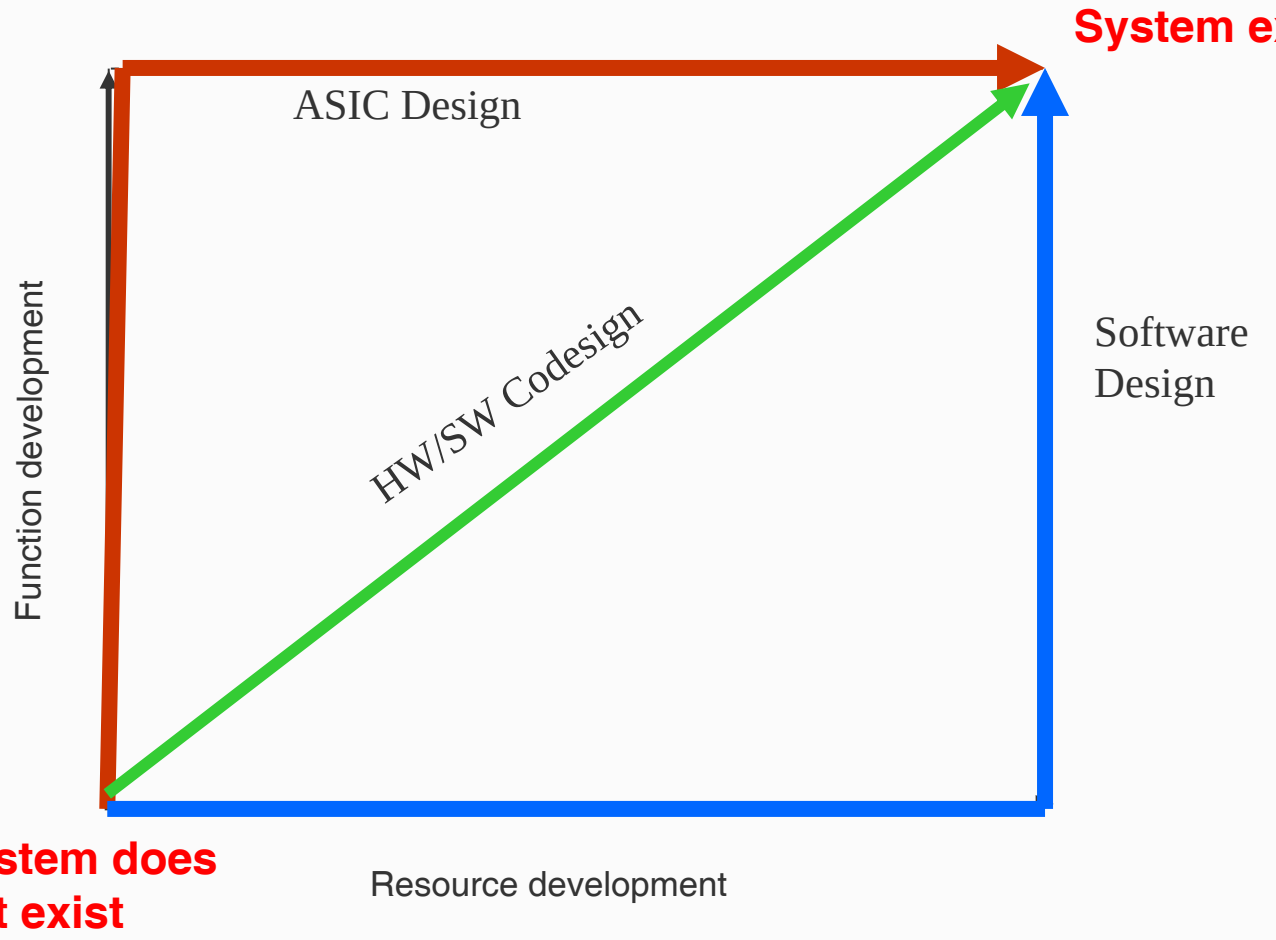
# Backbone-Platform-System Methodology

- Backbone Design
  - Topology
  - Switches, Channels and Network Interfaces (with generic parameters)
  - Protocols
- Platform Design
  - Scaling
  - Selection and placement of resources
  - Specialization of Switches, Channels, Network Interfaces and Protocols
  - Basic communication services
- Application Mapping
  - Programming functionality into resources
  - OS

# Development of ASNoC System



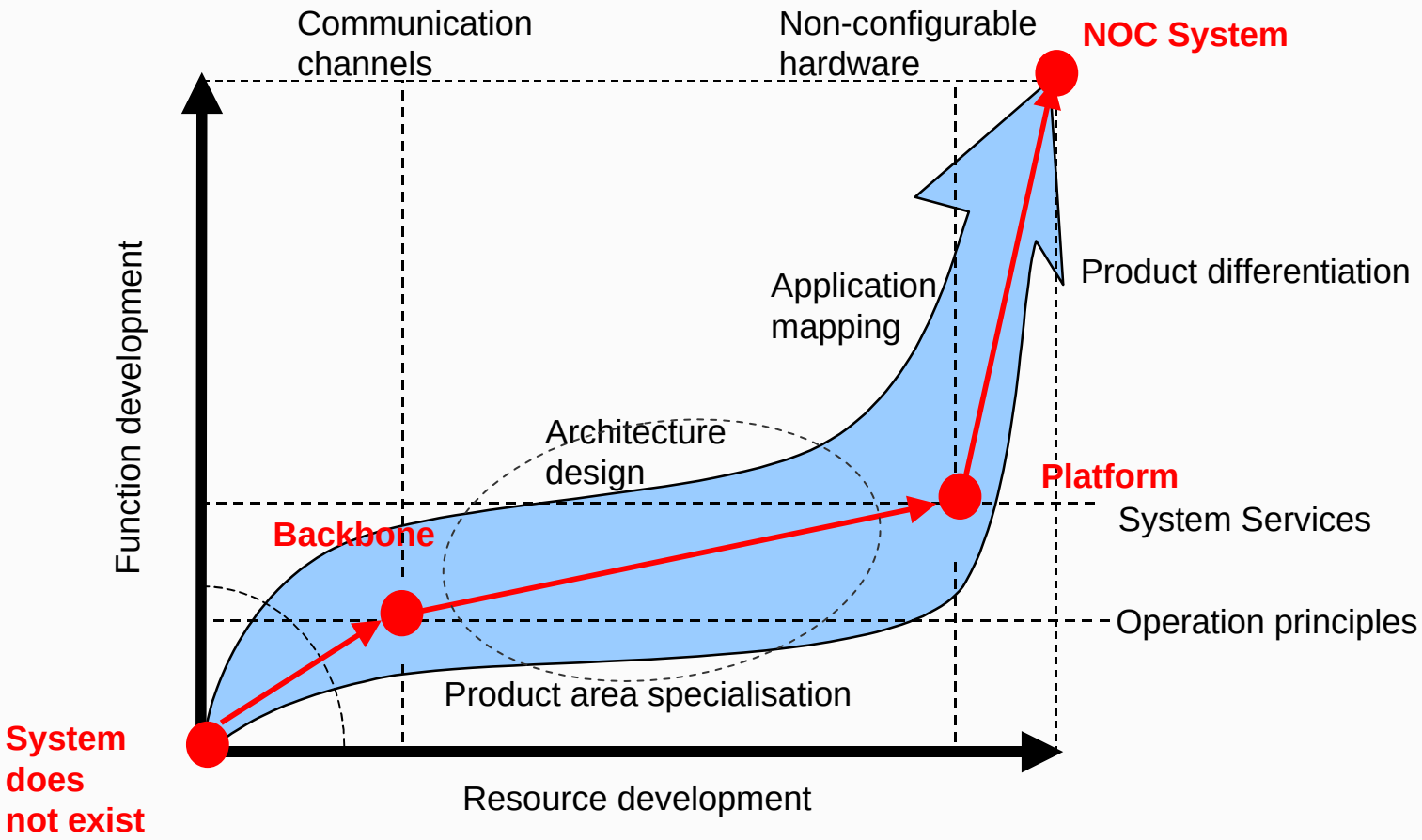
# Earlier Approaches



# Algorithm-Core Mappability Analysis[2,3]

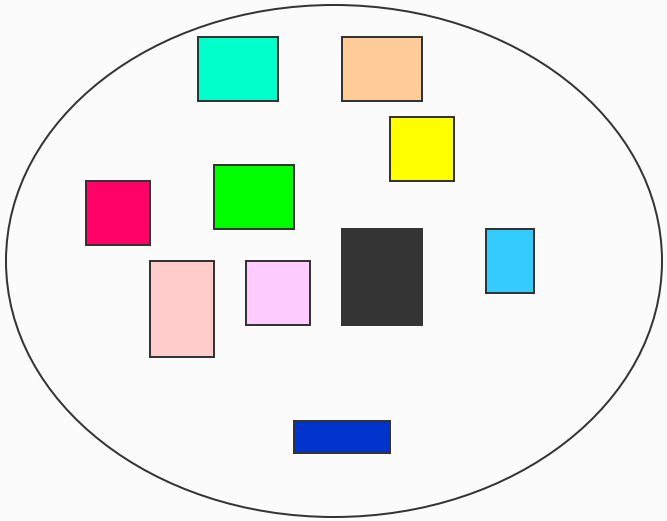
Based on Juha-Pekka's papers

# Development of ASNoC System

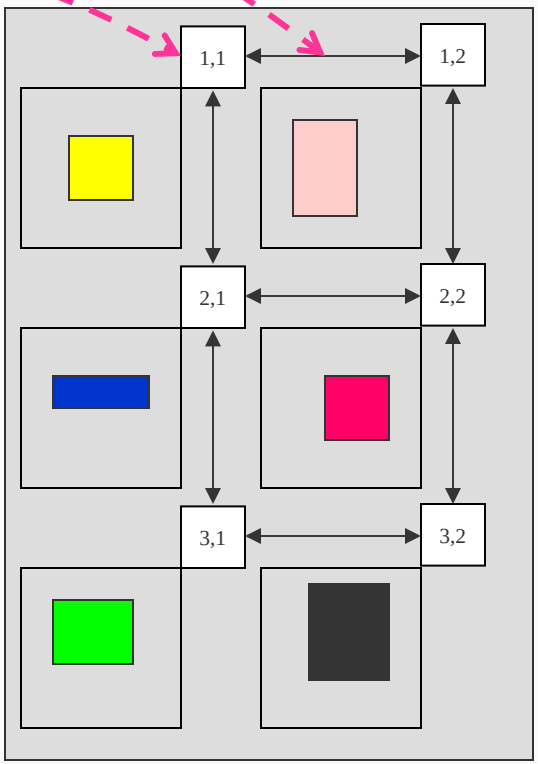


# Specialization of Mesh Topology NoC Architecture

- Specialize Back-Bone
- Specialize Resources



Available Cores

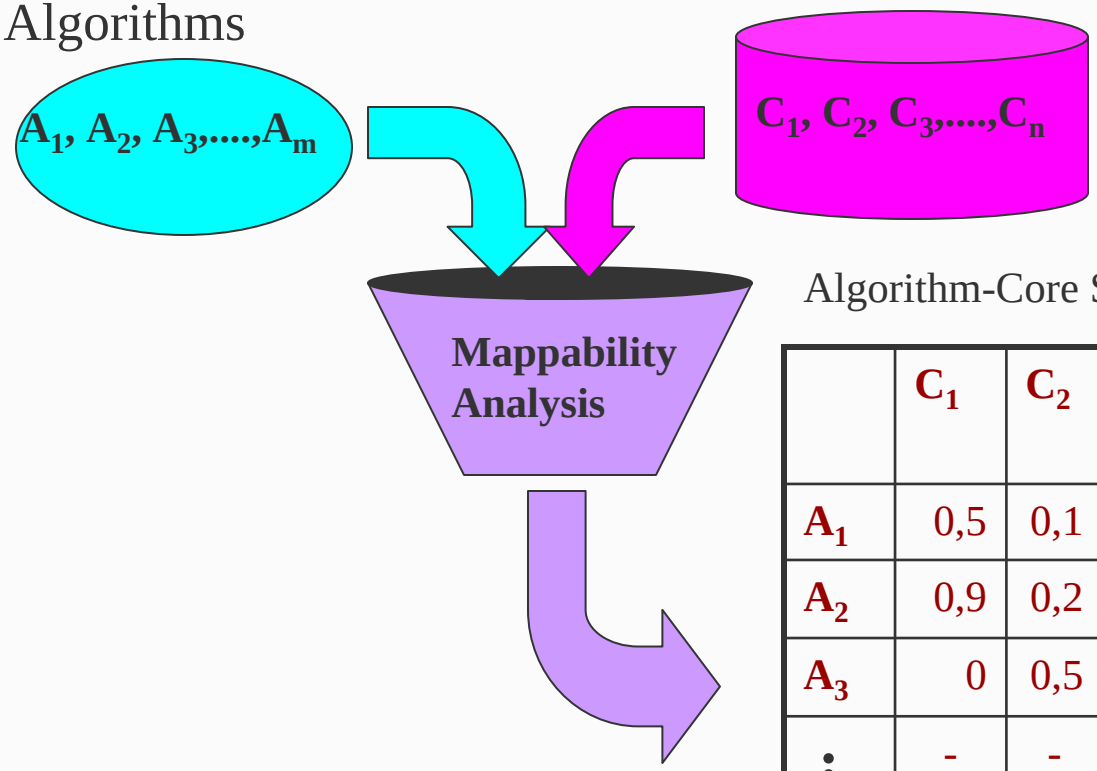


# Core Selection

- Task is similar to ”Processor Selection” in Embedded Systems
- Generalization of Processor Selection Problem
  - A core may be used for more than one algorithm/task
  - An algorithm may use more than one core
  - Set of Cores for a set of algorithms
    - Cluster/Region of cores

# Overview of the approach

Algorithms



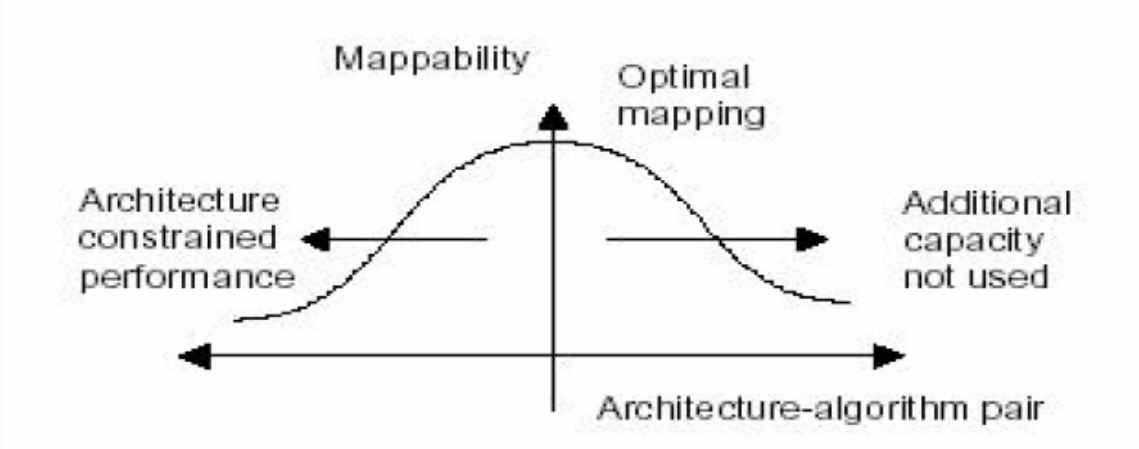

Available Cores

Algorithm-Core Suitability Index

| | $C_1$ | $C_2$ | $C_3$ | ... | $C_n$ |
|---|---|---|---|---|---|
| $A_1$ | 0,5 | 0,1 | 0,3 | - | .,. |
| $A_2$ | 0,9 | 0,2 | 0,6 | - | .,. |
| $A_3$ | 0 | 0,5 | 0,4 | - | .,. |
| ⋮ | - | - | - | - | .,. |
| $A_m$ | .,. | .,. | .,. | .,. | .,. |

# Core-Algorithm Mappability Analysis

- The goodness of a processor architecture-algorithm pair
  - Computational Capacity of architecture
  - Performance Requirement of algorithm
  - Optimal mapping implies that architecture does not constrain execution and does not have underutilization
  - CAMALA: Core Algorithm Mappability Analysis Approach Tool



# Mappability Analysis Goals

- Compare and Select the best core for a given algorithm
  - Select a set of cores for a set of algorithms
- Specialize a configurable core architecture for a given algorithm
  - Specialize the core for a set of algorithms
  - Architecture parameters
    - Number of registers
    - Number of functional units
    - Instruction set

# Mappability Parameters

- Instruction Set Suitability
- External Data Availability
- Internal Data Availability
- Control Flow Continuity
- Data Flow Continuity
- Execution Unit Availability

# Characterization of Algorithm and processor cores

| | Algorithm Characterization | Processor Core characterization |
|---|---|---|
| Instruction Set Suitability | Effectiveness of instructions w.r.t. operations in algorithm | Cost of instruction execution |
| External Data Availability | Number of memory accesses | Bus capacity of core |
| Internal Data Availability | Amount of temporary data to be stored | Number of registers |
| Control Flow Continuity | Number of branches | Branch penalty<br>Branch prediction |
| Data Flow Continuity | Possibility of data hazards | Handling of data hazards |
| Execution Unit Availability | Parallelism in algorithm | Number of functional units |

# Instruction Set Co-relation

- Effectiveness of instruction usage
  - Availability of instructions for operations used in the algorithm
    - If not available they have to be replaced by procedures
      - If floating point operation not available will lead to extra cost factor
  - Handling of operands of various data widths
- Methods used in compilers and High Level Synthesis used for this analysis

# Internal Data Availability Co-relation

- Effectiveness of register usage
- Algorithmic Requirement
  - Number of intermediate results in one scheduled step during execution
  - ASAP and ALAP on the data flow graph used to get this value
    - No constraint on resources: Maximum parallelism

e(a): Maximum number of intermediate results

e(c) = number of registers

# Control Flow Continuity Co-relation

- For the algorithm it depends on the branch instruction ratio
  - Ratio of branch instruction w.r.t. the total number of instructions
- For Core architecture it depends on
  - Branch prediction technique ( $P_e$)
  - Branch penality (D)
    - Degree of pipelining and super-pipeling

  $e(c) \sim (1- P_e)D$
- Good Matches
  - Small pipelines for algorithms with more branches
  - Long pipelines for algorithms with less branches

# Execution Unit Availability

- Operation level parallelism in algorithm vs. Number of functional units in the processor core
  - e(a) ~ number of instructions / number of steps in ASAP
  - e(c) ~ number of functional units

# Viewpoint-specific characterization

- The algorithm is compiled into a graph representation where each node is a block of code

For each computation node j in the algorithm

For each view-point i

$e_i(a_j)$: Algorithm characterization

$e_i(c)$: Core characterization

# Mappability of each view-point

$$m_{i,j}(c,a_j)=\begin{cases}\dfrac{e_i(c)}{e_i(a_j)}, e_i(c) \le e_i(a_j)\\ \dfrac{e_i(a_j)}{e_i(c)}, e_i(c) \ge e_i(a_j)\end{cases}$$

# Total Mappability of a view point

$$M_i(c,a) = \frac{\sum_{j=1\ldots|V|} (w_j \cdot m_{i,\,j}(c, a_j))}{\sum_{j=1\ldots|V|} w_j}$$

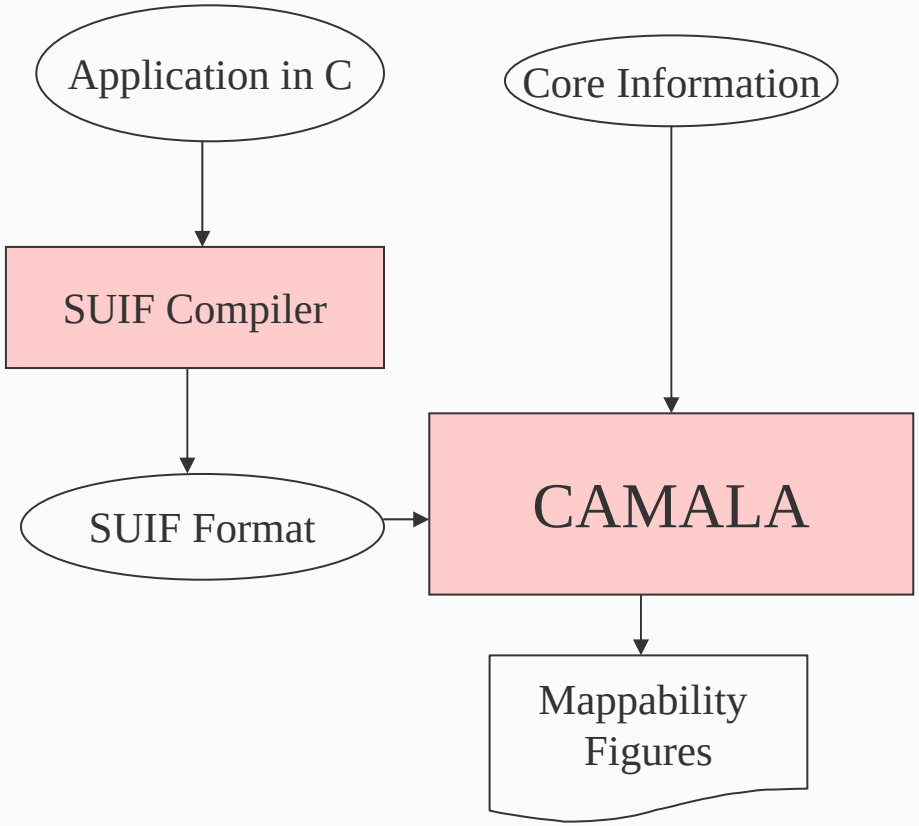
$w_j$ : number of times the block j is executed

# Total Mappability

$$M(c,a) = \sum_i w_i \cdot M_i(c,a)$$

$w_i$ : relative importance of different view points

# CAMALA Implementation



# Verification of Mappability Analysis

- Higher the total mappability between an algorithm and a core more suitable it is for the job
- Mappability analysis was verified using instruction set simulator Simplescalar
  - Goodness : Utilization x Speed-up

  Example : Processor with 2 functional units

  Utilization = 60% and Speed-up = 1.5

  Goodness = 0.60 x 1.5/2 = 0.45

# Experimental Results[

TABLE 1. Simulator configurations

| Config | Issue | Int alu | Int mul | FP alu | FP mul |
|---|---|---|---|---|---|
| ARM-1 | 1 | 1 | 1 | 1 | 1 |
| ARM-2 | 2 | 2 | 1 | 2 | 1 |
| ARM-4 | 4 | 4 | 1 | 4 | 1 |
| ARM-8 | 8 | 8 | 2 | 8 | 2 |

TABLE 2. Estimates and measurements

| Config | Dijkstra | | Quicksort | | SHA | | Stringsearch | |
|---|---|---|---|---|---|---|---|---|
| | Mappability | Goodness | Mappability | Goodness | Mappability | Goodness | Mappability | Goodness |
| ARM-1 | 0.391 | 0.284 | 0.584 | 0.285 | 0.383 | 0.336 | 0.383 | 0.270 |
| ARM-2 | 0.369 | 0.277 | 0.571 | 0.414 | 0.456 | 0.504 | 0.351 | 0.252 |
| ARM-4 | 0.305 | 0.143 | 0.591 | 0.294 | 0.406 | 0.426 | 0.286 | 0.145 |
| ARM-8 | 0.264 | 0.073 | 0.533 | 0.160 | 0.338 | 0.273 | 0.246 | 0.076 |

# Experimental Results: WLAN Modem

**Table 1. Core parameters.**

| Parameter | Values |
|---|---|
| Branch prediction technique, $P_T$ | no/static/dynamic |
| Dynamic prediction efficiency $P_e$ | 0-100% |
| Superpipeling degree, $D$ | 1-N |
| Bypassing , $D_B$ | yes/no |
| Number of execution paths, $E$ | 1-N |
| Number of registers $R$ | 0-N |
| Read buses $B_R$ | 0-N |
| Write buses $B_W$ | 0-N |
| Read/Write buses $B_D$ | 0-N |
| Program bus $B_P$ | yes/no |
| Bus width $B_w$ | 1-N |
| Floating point cost factor $C_f$ | 1-N |
| Word length cost factor $C_w$ | 1-N |

# Architectures for WLAN Modem Algorithms

D: Super-pipeling Degree

(1..20)

E: Number of Execution paths (1..6)

R: Number of registers (1..64)

B: Number of Data Buses

(1..6)

**Table 2. Average architecture parameters of best ten architectures for each WLAN modem algorithm and computational complexities of those algorithms.**

| Algorithm | *D* | *E* | *R* | *B* | MOPS |
|---|---|---|---|---|---|
| Convolutional encoding | 3.5 | 1.5 | 1.0 | 1.5 | 54 - 486 |
| Guard interval insertion | 7.5 | 2.0 | 4.0 | 2.0 | 80 |
| Interleaving | 7.5 | 2.0 | 3.0 | 2.0 | 12 - 72 |
| Modulation | 4.5 | 2.0 | 4.0 | 2.0 | 12 |
| Symbol filtering | 9.5 | 3.0 | 12.0 | 1.0 | 120 |
| Deinterleaving | 6.5 | 3.0 | 3.0 | 2.0 | 12 - 72 |
| Synchronization | 10.5 | 2.0 | 8.0 | 2.0 | 200 |
| FFT | 10.5 | 2.0 | 9.0 | 1.0 | 296 |
| Frequency error correction | 10.5 | 2.0 | 8.0 | 1.0 | 156 |
| Guard interval deletion | 7.0 | 2.0 | 4.0 | 1.0 | - |
| BPSK demodulation | 4.0 | 1.5 | 2.0 | 2.5 | 60 - 504 |
| QPSK demodulation | 4.5 | 1.0 | 3.0 | 2.0 | |
| QAM16 demodulation | 5.5 | 1.0 | 6.0 | 2.0 | |

# Multiprocessor for WLAN Modem

**Table 3. Mappings and estimated complexities of two and three processor core solutions.**

| Number of cores | Core type | Algorithms mapped to core | MOPS |
|---|---|---|---|
| 2 | 11-2-9-1 | Symbol filtering, FFT, frequency error correction | 572 |
| | 5-2-4-2 | Synchronization, encoding, demodulation, etc. | 1426 |
| 3 | 11-2-9-1 | Symbol filtering, FFT, frequency error correction | 572 |
| | 5-1-3-2 | All demodulation algorithms | 504 |
| | 5-2-4-2 | Synchronization, encoding, etc. | 922 |

# Summary

- Mappability Analysis technique provides possibility of fast selection/specialization of cores for an application or application set.
- Effect of memory organization on mappability are not considered
- Effect of inter-core communication organization on mappability not considered

# Quality of Service NoC (QNoC) [4,5]

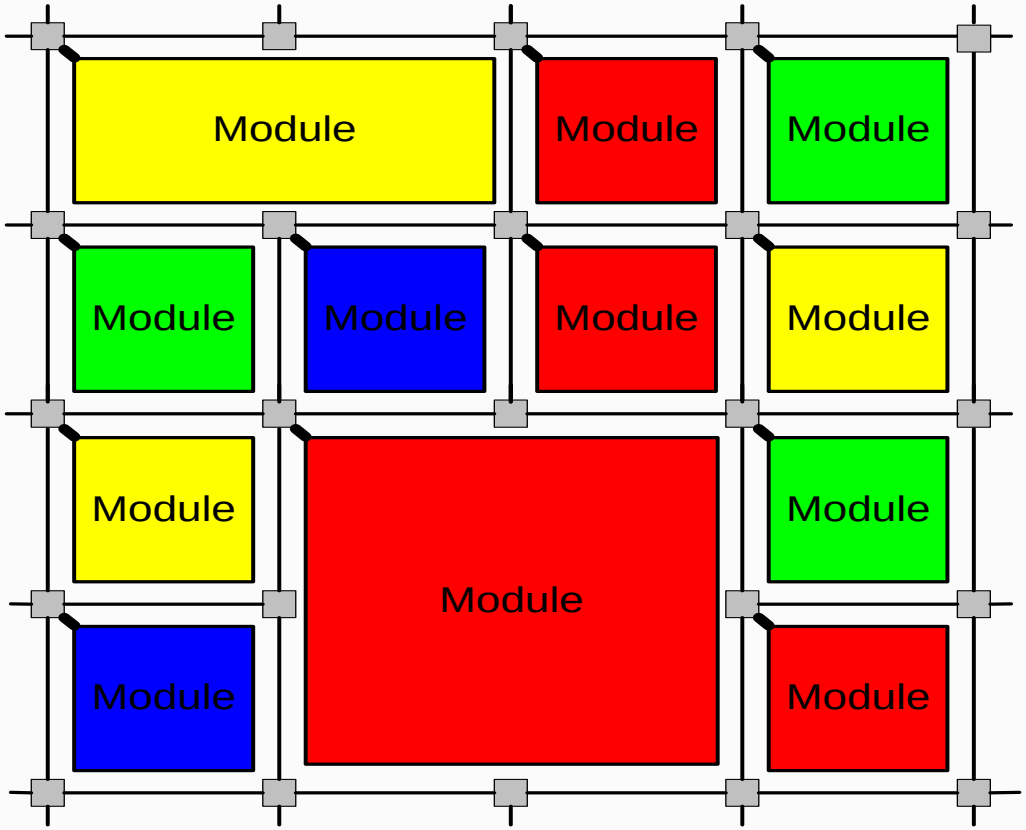
Based on Evegeny Bolotin's papers and presentations

# Goals of QNoC

- Design an Application Specific NoC which provides required communication performance (QoS) at minimum cost
  - Communication QoS: Required throughput and end to end delay guarantee between communicating cores
    - Specialize links in NoC
  - Cost
    - Area: links, buffers and routers are trimmed
    - Power Consumption: shortest path routing

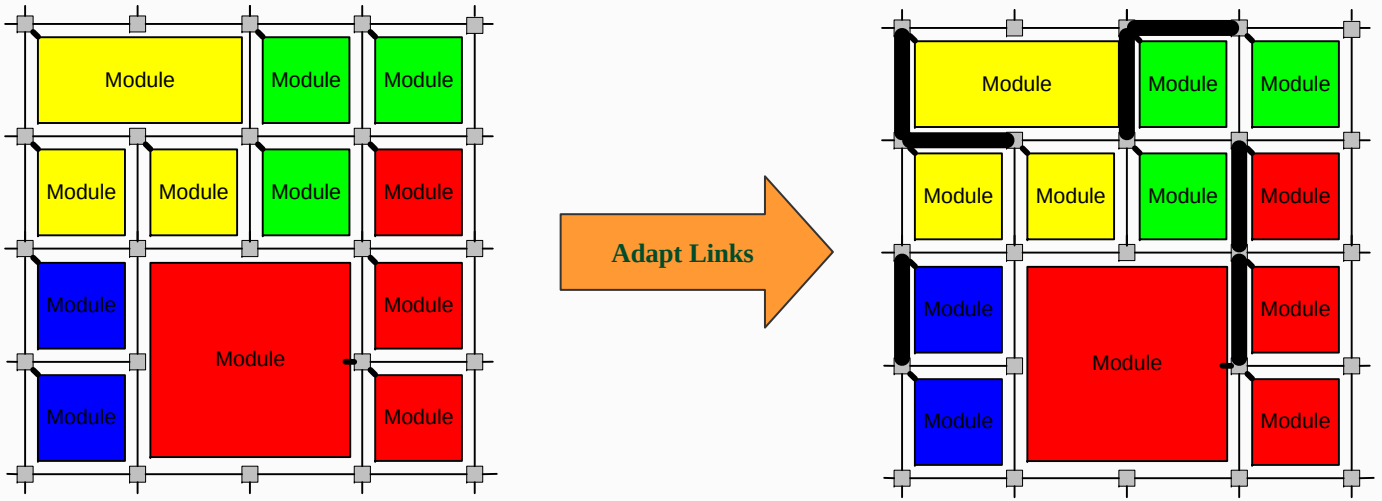
# Quality of Service NoC (QNoC) Architecture

- Irregular Mesh Topology




Figures taken from Evegeny Bolotin's Presentation

# Specializion of Links




Figures taken from Evegeny Bolotin's Presentation

- Link bandwidth is adjusted according to expected traffic
- Bandwidth is adjusted by either number of wires or the dat frequency

# QNoC Service Levels

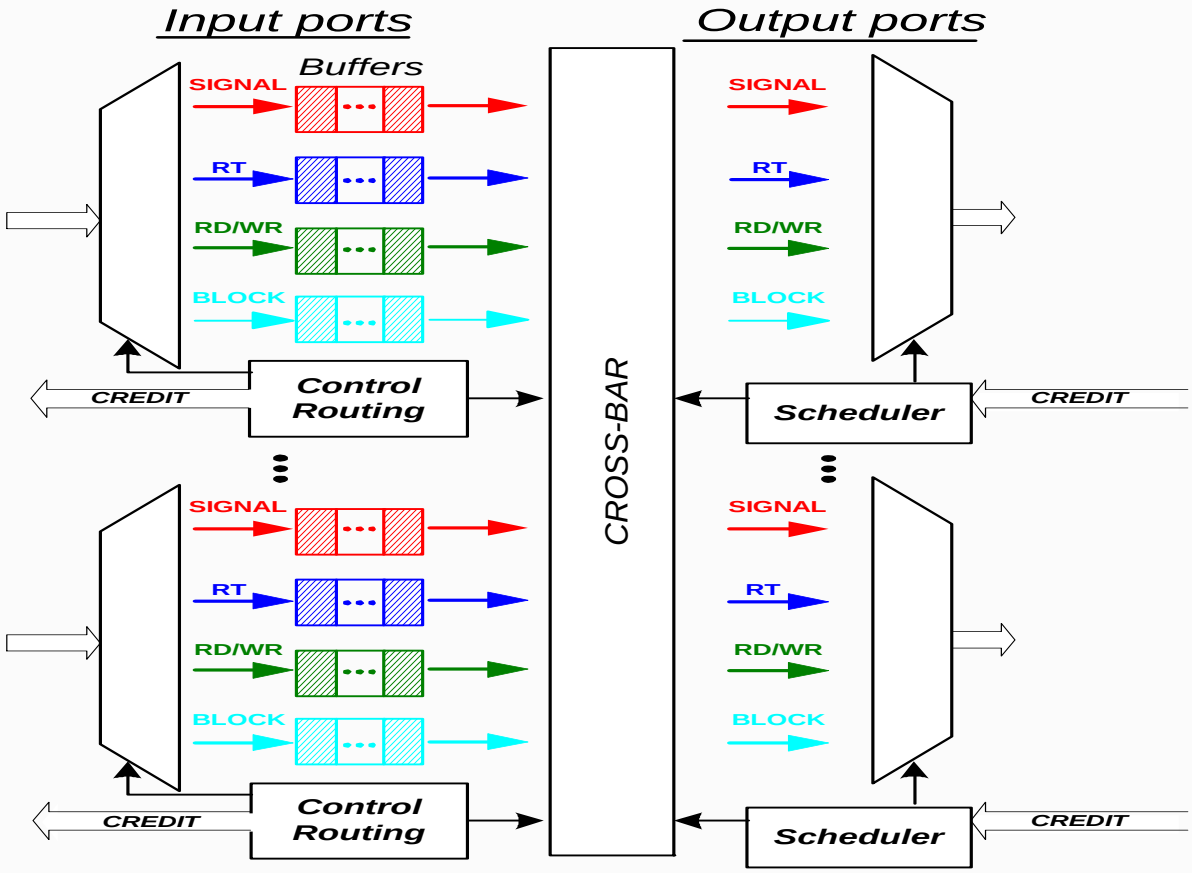
- Four different service levels with different priorities
  - Signaling
    - Short urgent packets
    - Suitable for control signals and Interrupts
  - Real-Time
    - Guaranteed bandwidth and latency
    - Useful for streamed audio or video processing
  - Read/Write ( RD/WR)
    - Provides Bus Semantics
  - Block Transfer
    - Large blocks of data
    - DMA transfers
- Priority

  Signaling >Real Time > Read/Write > Block Transfer

# QNoC: Routing Algorithm

- Static shortest X-Y coordinate based routing
  - Avoids deadlocks
  - No reordering at end points
- Wormhole packet forwarding with credit based flow control
- Packets of various priorities are forwarded in an interleaved manner according to packet priorities
- A high priority packet can pre-empt a low priority long packet

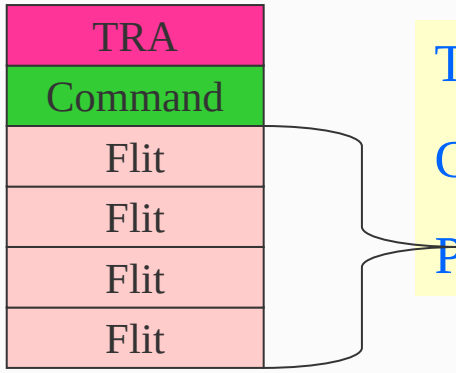
# QNoC: Router Architecture [4]




Slide taken from Evegeny Bolotin's Presentation

# QNoC Packet Format

## ➢ Arbitrary Sized Packets




TRA: Target Routing Address

Command: Info. about payload

Payload: Arbitrary Length

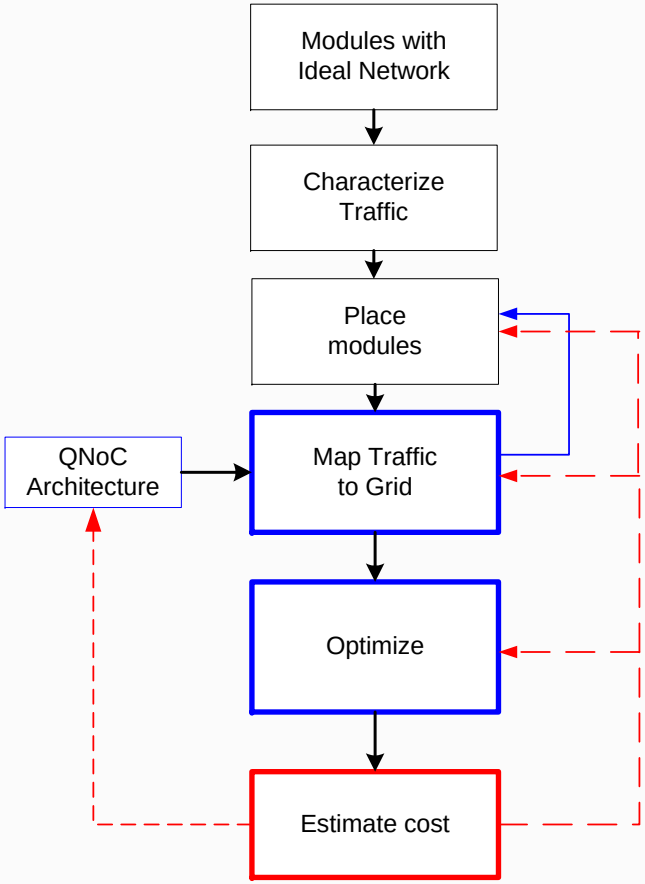
Flit Types:

FP (full packet): a one-flit packet

EP (end of packet): last flit

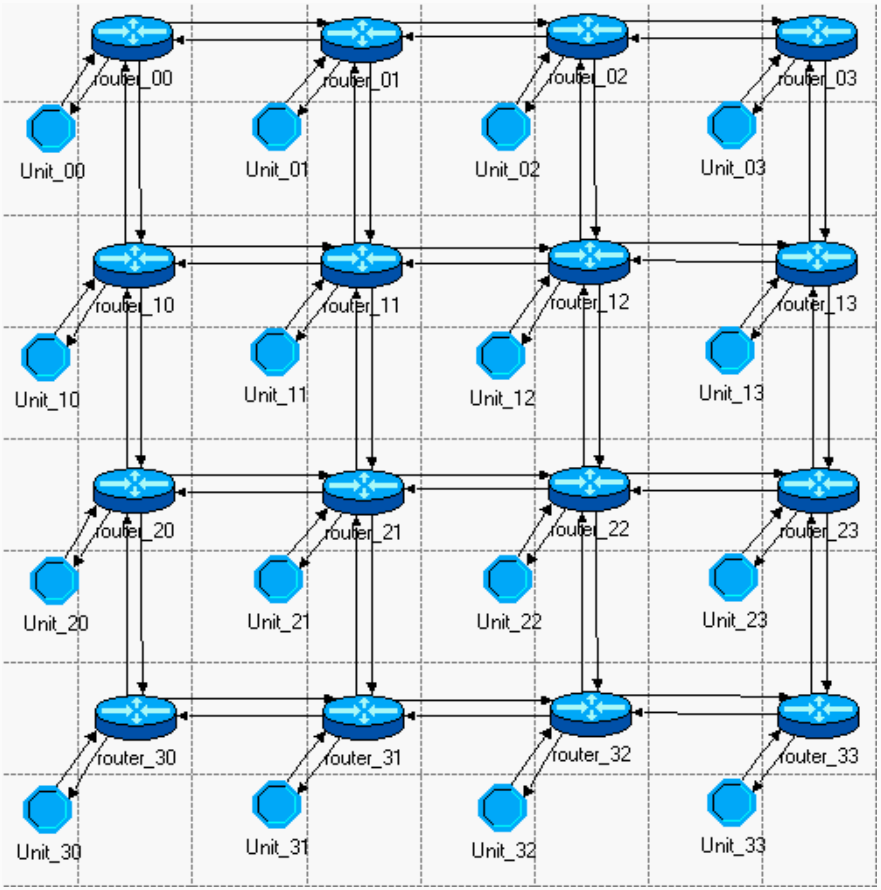
BDY (body): a non-last flit

# QNoC Design Flow




Figures taken from Evegeny Bolotin's Presentation

# Design Example




Figures taken from Evegeny Bolotin's Presentation

# Design Example[4]

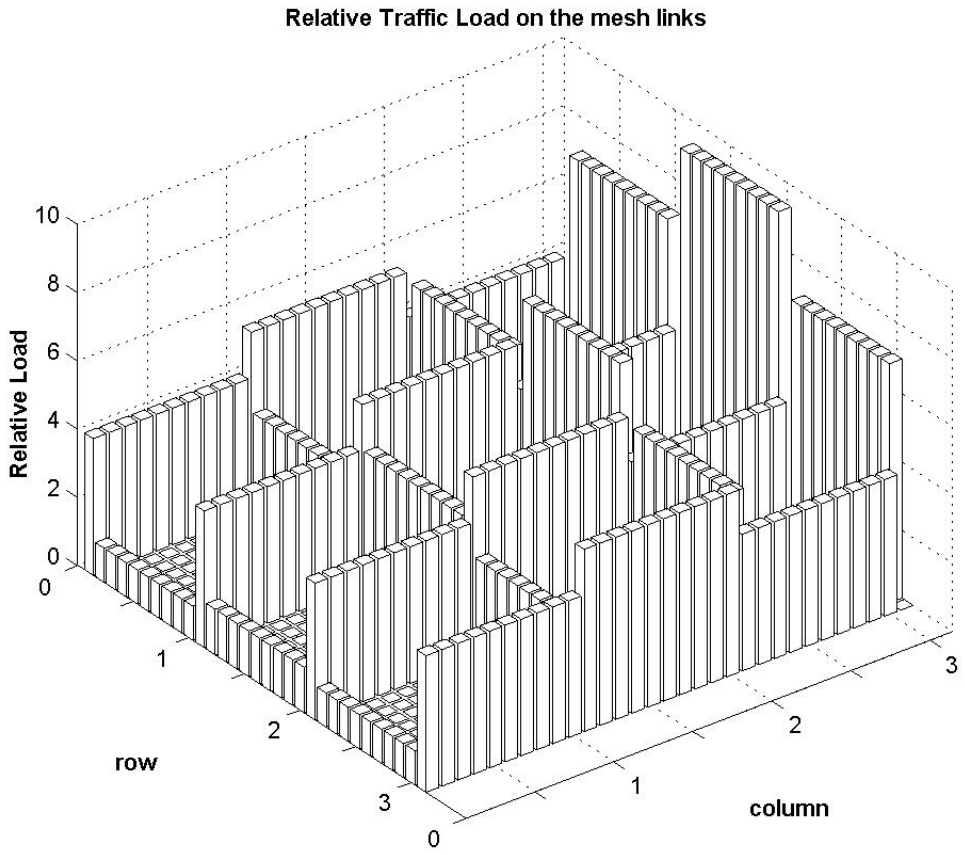
Table taken from Evegeny Bolotin's Presentation

## Representative Design Example, each module contains 4 traffic sources:

| ***Traffic Source*** | ***Traffic interpretation*** | ***Average Packet Length [flits]*** | ***Average Inter-arrival time [ns]*** | ***Total Load per Module*** | ***ETE requirements For 99.9% of packets*** |
|---|---|---|---|---|---|
| *Signaling* | Every 100 cycles each module sends interrupt to a random target | 2 | 100 | 320 Mbps | 20 ns (several cycles) |
| *Real-Time* | Periodic connection from each module: 320 voice channels of 64 Kb/s | 40 | 2 000 | 320 Mbps | 125 µs (Voice-8 KHz frame) |
| *RD/WR* | Random target RD/WR transaction every ~25 cycles. | 4 | 25 | 2.56 Gbps | ~150 ns (tens of cycles) |
| *Block-Transfer* | Random target Block-Transfer transaction every ~12 500 cycles . | 2 000 | 12 500 | 2.56 Gbps | 50 µs (Several tx. delays on typ. bus) |

# Uniform Scenario - Observations

## Calculated Link Load Relations:



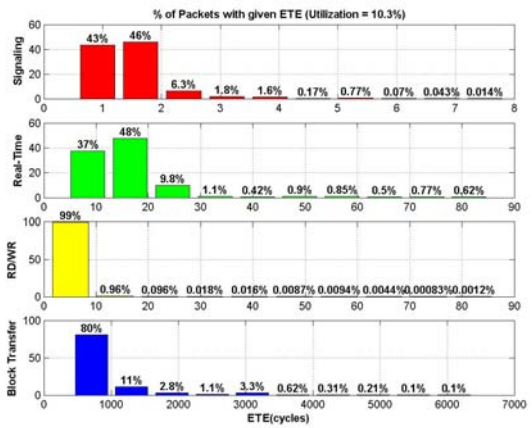

Figures taken from Evegeny Bolotin's Presentation

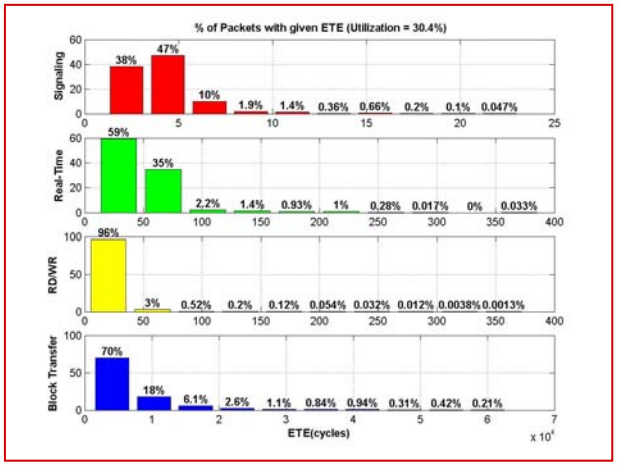
# Uniform Scenario - Observations

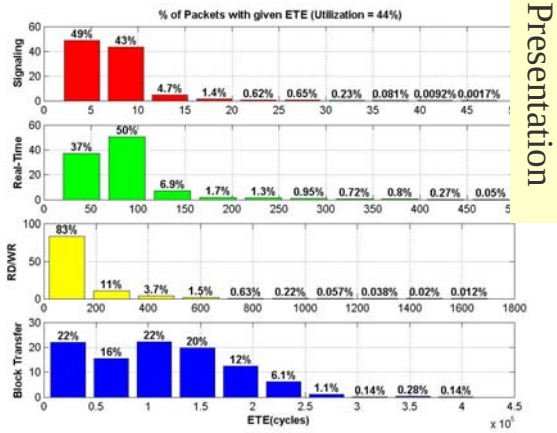
Various Link BW allocations:

| Allocated Link BW [Gbps] | Average Link Utilization [%] | Packet ETE delay of packets [ns or cycles] | | | |
|---|---|---|---|---|---|
| | | Signaling (99.9%) | Real-Time (99.9%) | RD/WR (99%) | Block-Transfer (99%) |
| 2560Gbps | 10.3 | 6 | 80 | 20 | 4 000 |
| *850Gbps* | *30.4* | *20* | *250* | *80* | *50 000* |
| 512Gbps | 44 | 35 | 450 | 1 000 | 300 000 |

Desired QoS



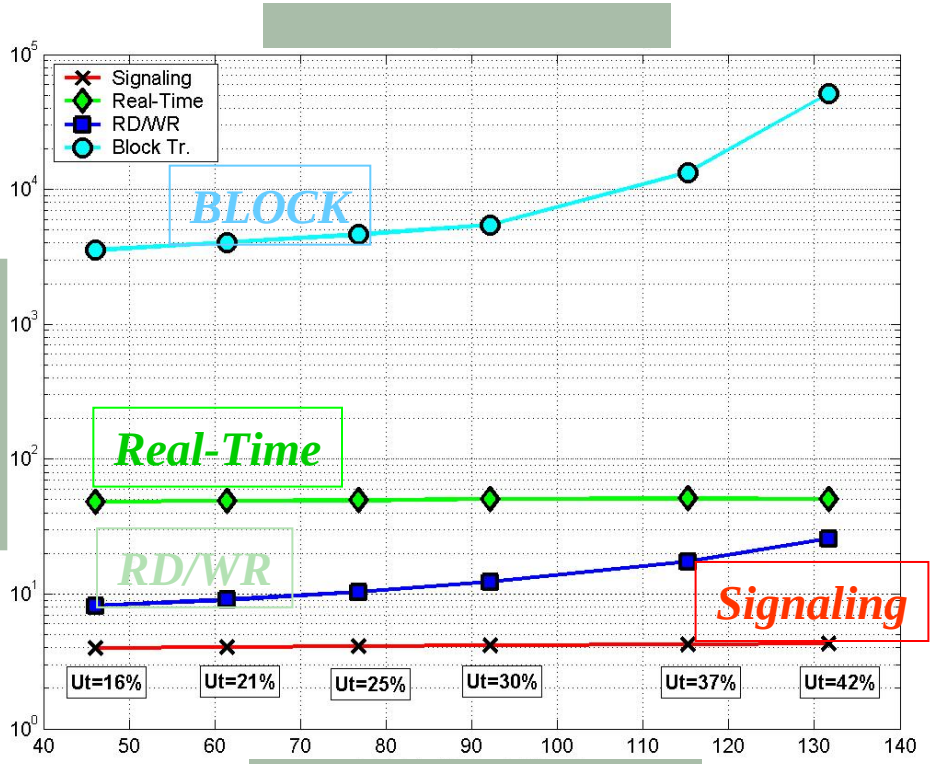

Figures taken from Evegeny Bolotin's Presentation

# Uniform Scenario - Observations

Figures taken from Evegeny Bolotin's Presentation

## Fixed Network Configuration -Uniform Traffic

Network behavior under different traffic loads?



# Alternatives for connecting n cores[4,5]

## For achieving the same Communication Bandwidth

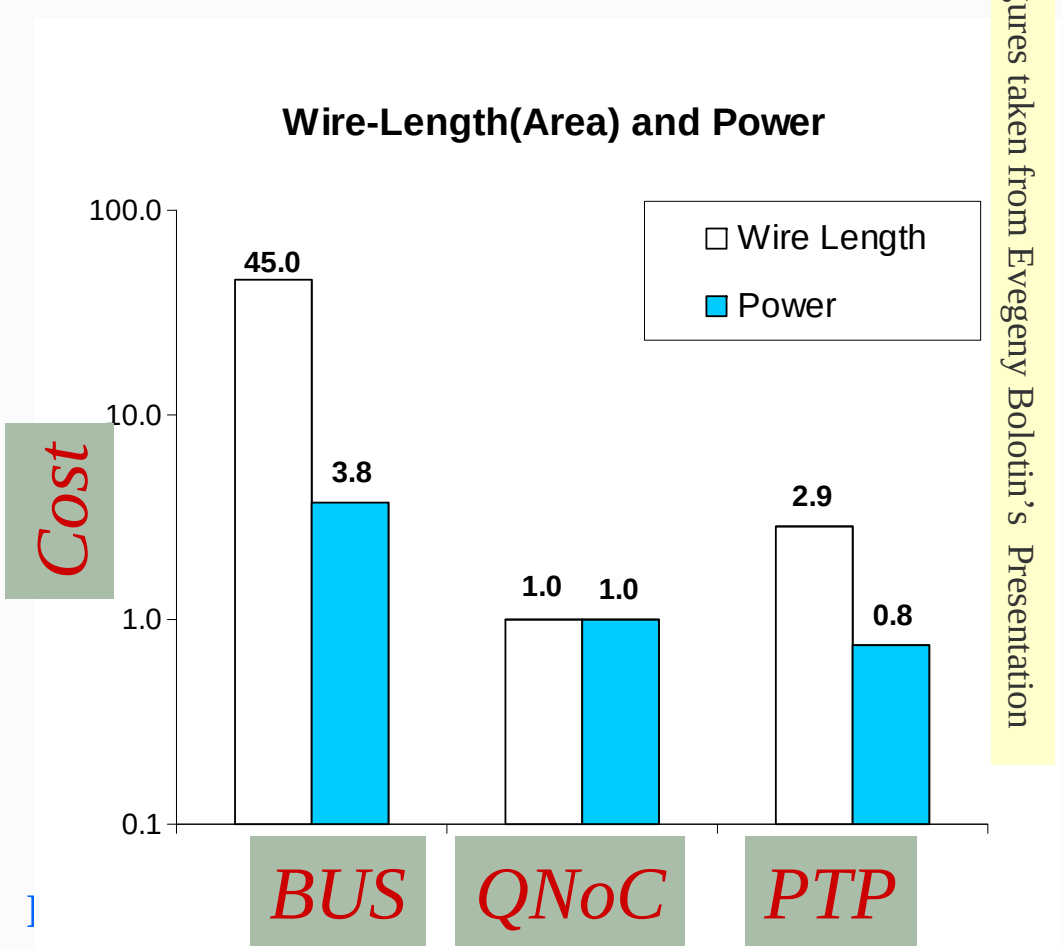
| *Arch* | *Total Area* | *Power Dissipation* | *Operating Frequency* |
|---|---|---|---|
| ***NS-Bus*** | $O\left(n^3\sqrt{n}\right)$ | $O\left(n\sqrt{n}\right)$ | $O\left(\frac{1}{n^2}\right)$ |
| ***S-Bus*** | $O\left(n^2\sqrt{n}\right)$ | $O\left(n\sqrt{n}\right)$ | $O\left(\frac{1}{n}\right)$ |
| ***NoC*** | $O(n)$ | $O(n)$ | $O(1)$ |
| ***PTP*** | $O\left(n^2\sqrt{n}\right)$ | $O\left(n\sqrt{n}\right)$ | $O\left(\frac{1}{n}\right)$ |

Table taken from Evegeny Bolotin's Presentation

# QNoC vs. Alternative Solutions
## (4x4 mesh, uniform traffic)

Uniform scenario (Same QoS):

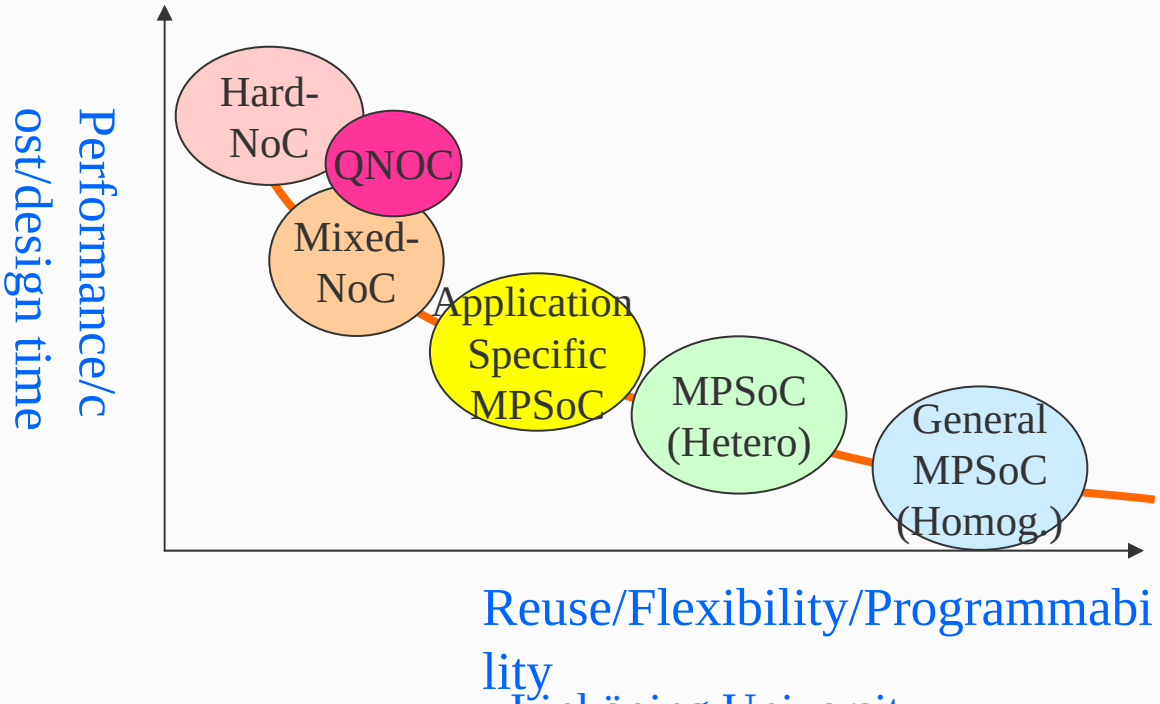
| Arch. | Frequency | Utilization | Av. Link Width |
|---|---|---|---|
| QNoC | 1GHz | 30% | 28 |
| Bus | 50 MHz | 50% | 3 700 |
| PTP | 100MHz | 80% | 6 |




Figures taken from Evegeny Bolotin's Presentation

# Summary about QNoC Approach

- Lower cost and higher performance are the driving forces behind Application Specific NoC approaches like QNoC

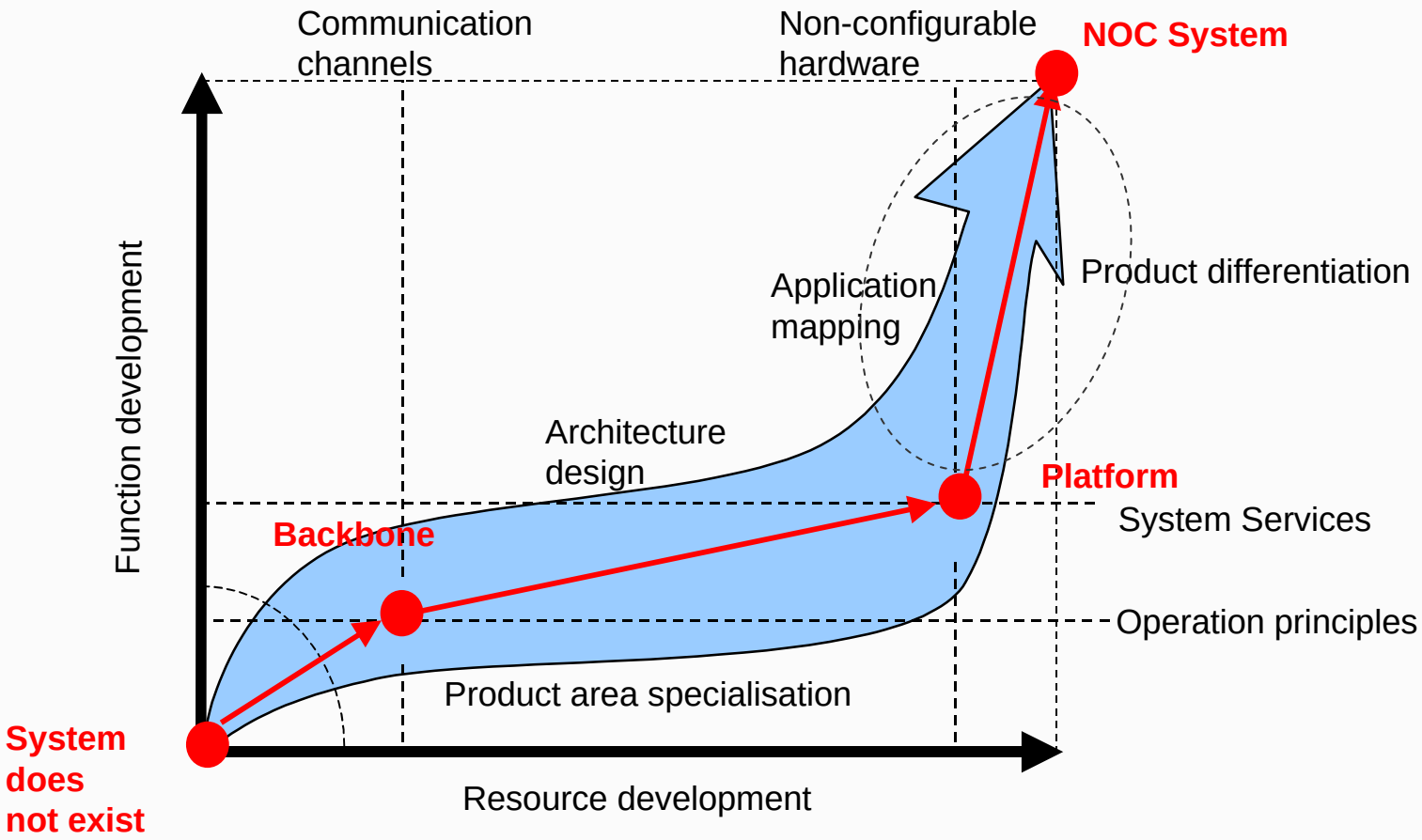


# Mapping Applications to NoC Platforms

# Various Options

- Architecture-Application Co-development
  - Optimized in terms of performance and cost
  - High Development Time
  - Less Flexible
- Mapping sequential code to a Fixed Platforms
  - Low development time and cheap
  - Medium Performance
  - Low utilization of resources
- Mapping parallel code through emulation of one network on another network

# Application Mapping



# Task Graph as an Application Model

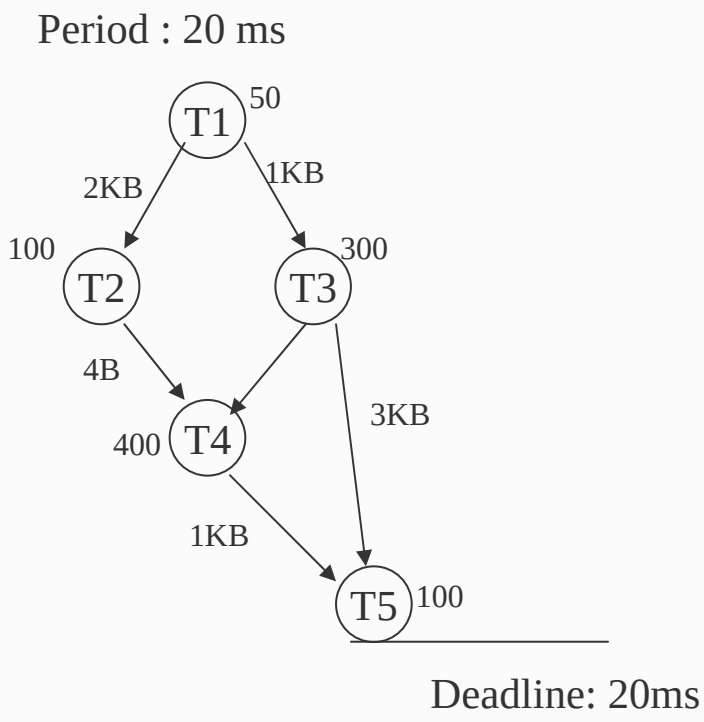
- Most common graph model used for representing applications for multi-processor systems
  - Application is partitioned into smaller units called tasks
- Similar in properties to Data Flow graph used for high level synthesis of ASICs
- It can represent
  - Concurrency among computations
  - Data Dependencies among computations
  - Communication among computations
  - Control Dependencies among Computations
  - Temporal Dependencies among computations
  - Non-determinism among computation

# Task Graph

- Task Graph is a weighted directed acyclic graph (DAG)
  - Nodes represents computational tasks
    - Weight on a node may represent different features of the computation : Number of instructions, Execution Time etc.
  - Edges between tasks represent dependencies
    - Weight on an edge may represent size of data to be communicated between tasks or some temporal information

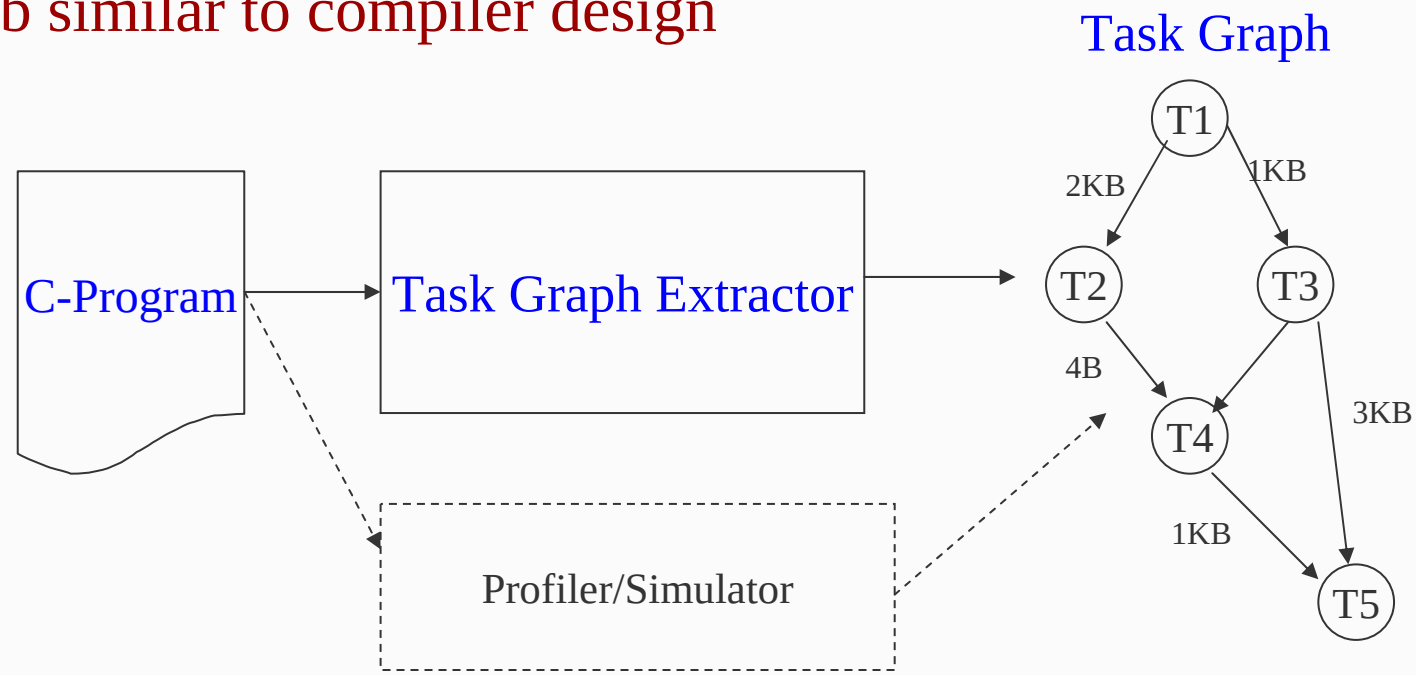
# Task Graph Example

- Parameters
  - Granularity of nodes
  - Period and deadlines
- Granularity of Nodes
  - High Granularity
    - Less available parallelism
    - Less communication cost
  - Low Granularity
    - Higher available parallelism
    - More communication cost
  - Hierarchical Task Graphs
    - Task node itself is a task graph at lower level



# Task Graph Extraction from Sequential Code

- Job similar to compiler design

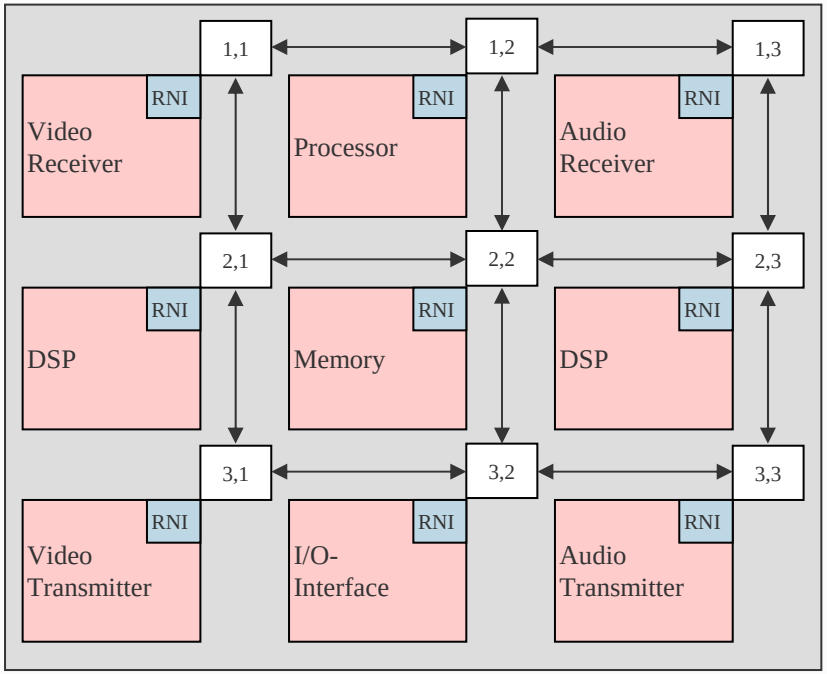


# Concurrent Applications as Multi-Task Graph (MTG)

- NoC is expected to integrate more than 50 processor size resources in a few years
  - Can concurrently run many applications
- Each application can be represented by a task graph called Single Task Graph (STG)
- We call aggregation of many STGs as a Multi-Task Graph (MTG).
  - There is no data or temporal dependencies among various STGs
  - Relative importance of STGs may be specified in some form

# Fixed NoC Platforms

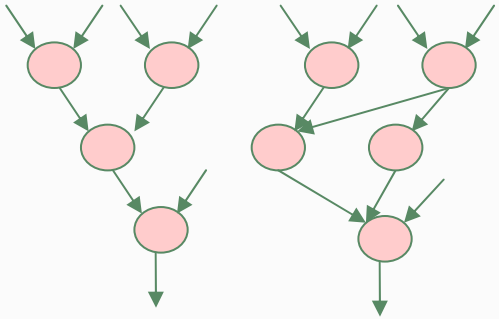
- A NoC Platform is fixed if
  - Topology and Size of NoC is decided
  - Communication Protocols are decided
  - Resources for all the slots have been selected



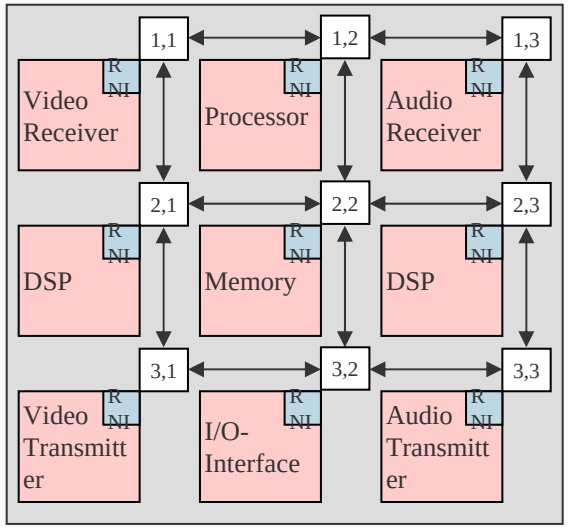
# Mapping and Scheduling Problem

Mapping: For every task decide the core in NoC where it will be executed

Scheduling: For every task decide the starting time of its execution




Multi-Task Graph



# Issues : Static Vs. Dynamic Mapping & Scheduling

- Static or Off-line Mapping and Scheduling
  - The resource on which the task will be run decided before run-time
  - Starting time for execution of each task on the resource is also decided
  - Normally based on worst case estimates of task execution time
- Dynamic or On-line Mapping and Scheduling
  - Task assignment to resources as well ordering of their execution is done at run time
    - Non-preemptive or Preemptive scheduling
  - Based on actual execution time of tasks
  - Which resource runs mapping and scheduling algorithm?
- Dynamic mapping and scheduling can lead to better performance

# Objective Functions

- Primary Objective for real-time application is to meet all hard deadline
  - Find a mapping and scheduling of tasks on the computing platform such that the performance requirements are met
- Secondary Objectives
  - Power consumption minimization
  - Soft deadline are met as much as possible

# Tang's Lei's 2-step genetic Algorithm [6]

- Solves mapping and scheduling as a single integrated problem
  - Static mapping and static and non-preemptive scheduling
  - Fixed NoC architecture
- Objective
  - Each task graph can meet its individual execution deadline
  - Maximize the overall execution performance

# Inputs to the Mapping Algorithm

- Weighted Task Graph
  - Weight on each edge corresponds to amount of data communicated from source node to destination node
- Deadline for execution of each STG
- Task Execution Time Table
  - Worst case execution Time on each core

| Task/Core | $C_1$ | $C_2$ | ---------- | $C_M$ |
|---|---|---|---|---|
| $T_1$ | 100 | ∞ | | 75 |
| $T_2$ | 200 | 50 | | 150 |
| -------- | | ∞ | | |
| $T_N$ | 50 | ∞ | | 40 |

# Mapping Issues and Assumption

- A task graph node has different execution time on different resources
  - There may be many copies of the same resource in the network
  - Every task can be executed by at least one resource
  - A task is executed without pre-emption
- The execution time of an STG depends on:
  - The type and the position of resources on which its tasks are executed.
  - Execution of tasks of various STGs may be interleaved
- There is enough local memory with every core to store all the required data for all the tasks executing on them

# Estimation of STG delay

- Delay of STG corresponds to the delay of critical path in the task graph
  - Sum of execution time of tasks and edges on the critical path

$$T_k = T_{critical-path,k} = \sum_{V_i \in critical-path_k} Tv_i + \sum_{E_j \in critical-path_k} Te_j$$

# Edge delay

- Two communicating vertices of MTG may get mapped on two different NoC resources.
- An edge delay in MTG depends on:
  - Mapping of task nodes to resources
  - Data size
  - Router Design including protocols
  - Network traffic situation at that time

# Edge delay (assumption)

- In mesh based network using a dynamic routing algorithm, delay of $E_i$ going from node $(x_1, y_1)$ to $(x_2, y_2)$ can be coarsely estimated as

$$Te_i = k_e \cdot w_i \cdot (|x_1 - x_2| + |y_1 - y_2|)$$

where $w_i$ is the size of data to be communicated

$K_e$ absorbs all architectural parameters

# MTG effective delay: Objective Function

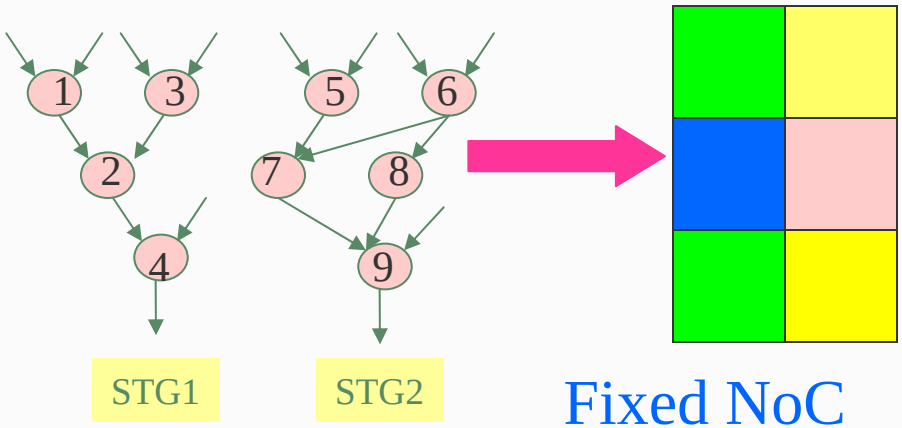
- Since the main optimizing goal is to meet every STG's deadline constraint
- we define a normalization, called MTG Effective Delay, to reflect every STG's contribution to the overall Objective Function and then combine them all as:

$$T_{MTG} = D_{\max} \times \max\{\frac{T_0}{D_0}, \cdots, \frac{T_{(c-1)}}{D_{(c-1)}}\}$$

Where $T_i$ and $D_i$ are execution delay in a mapping and deadline for $STG_i$. $D_{max}$ is the largest deadline.

# Mapping Algorithm: Basic Idea

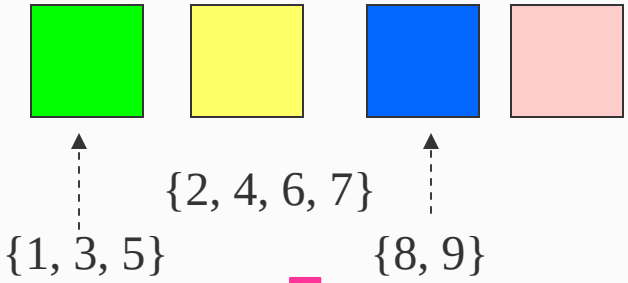
## Two Steps




MTG

Fixed NoC

## Step 1




## Step 2

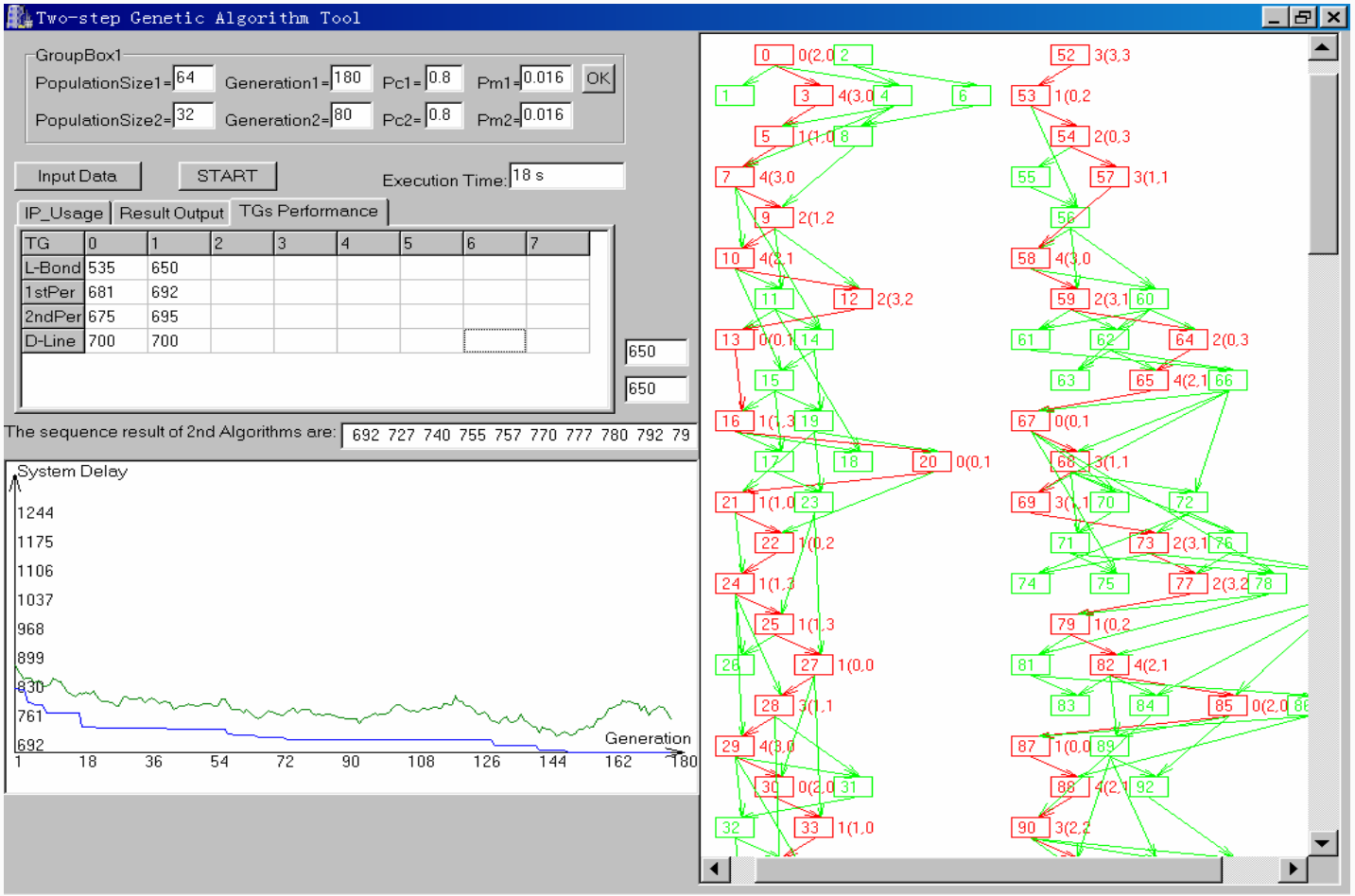
# The First Step: Partition nodes according to types

- We assume that there are a few types of NoC resources
- For each task in MTG find the type of resource it should be executed on
  - Non-trivial problem, since we want to use maximum parallelism ( maximum utilization of resources)
    - Insisting on use of fastest resource for a task can delay start time of the task or other tasks
    - To avoid communication delay it may be better to execute connected tasks in the same resource/neighboring resources
- This step is implemented as a genetic algorithm and generates many candidate solutions
  - This is used as the initial population for the second step

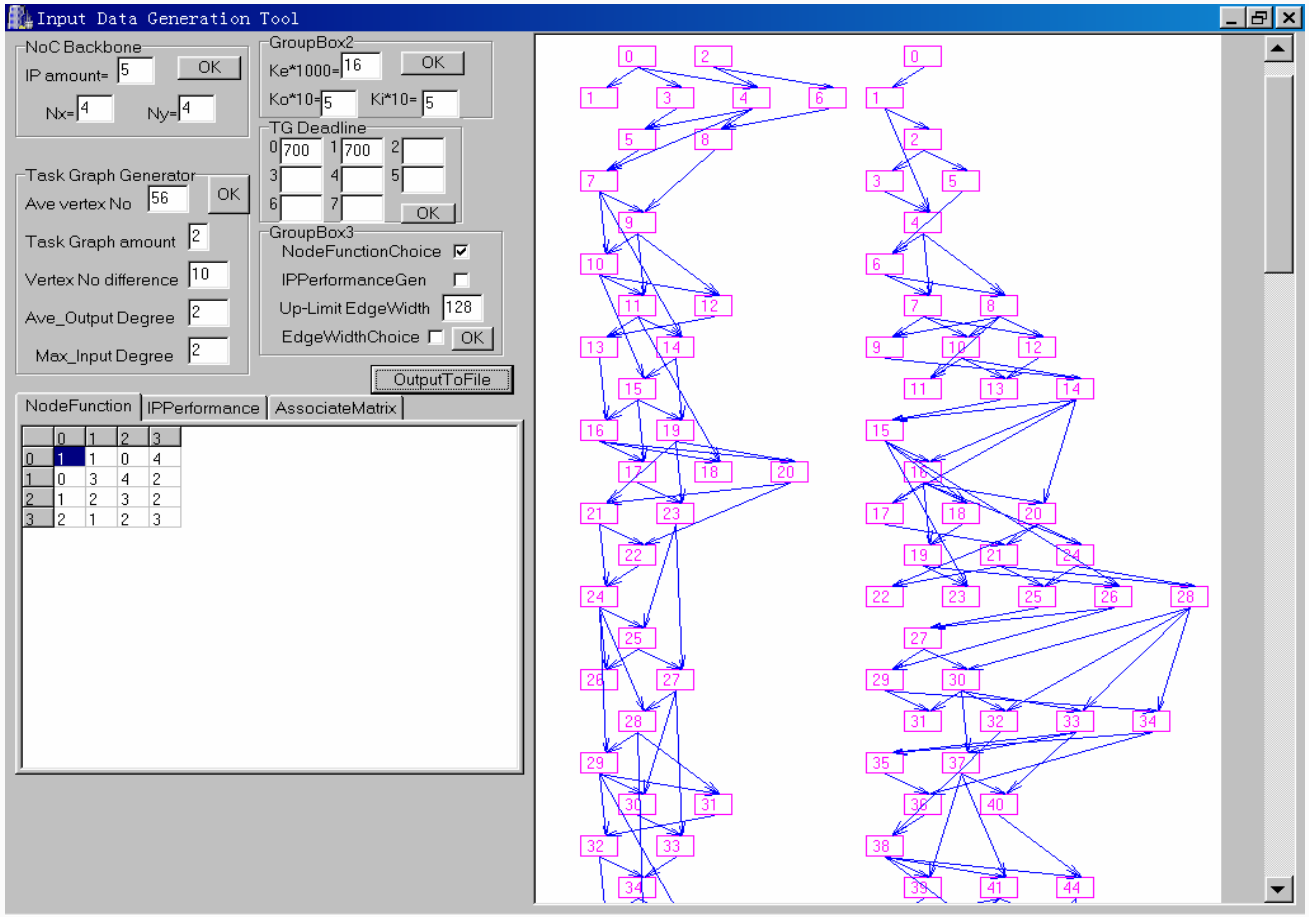
# The Second step: Binding of Position

- In each of the candidate solution
  - For each task node decide the exact resource of the type decided in first step
    - If the number of resources of a type is exactly one then the choice is trivial.
    - Choice will affect the execution time of individual STG
- This binding problem is also hard!
- This step is also implemented using a separate genetic algorithm

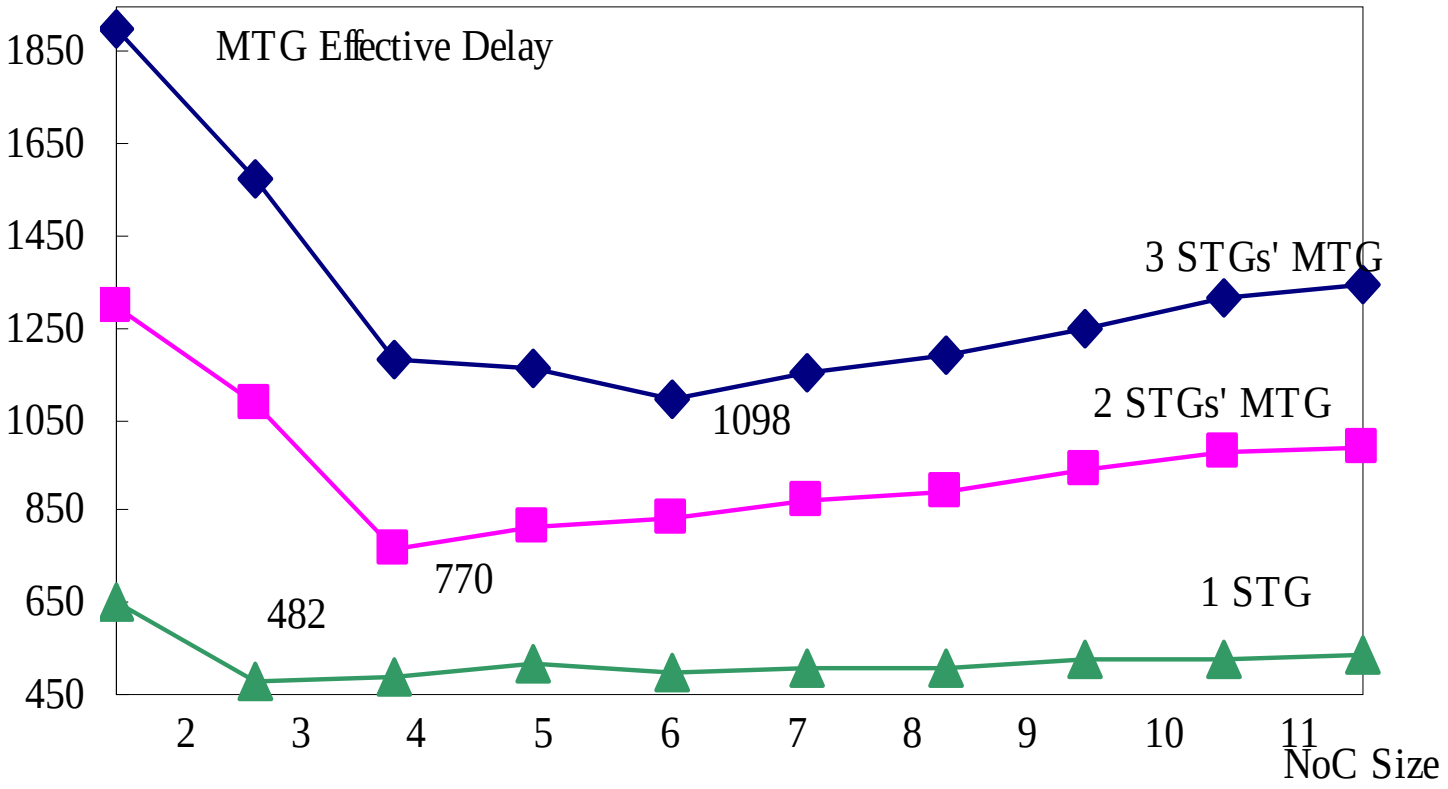
# Task Graph Mapping Tool



# Input Generation Tool



# MTG performance variations with NoC size




MTG performance variations with NoC size changing

# Conclusions

- NoC research is in its infancy. Many tools will be required for ASNoC Design
  - Simulators for evaluation of design choices
  - Performance and power estimators
  - Tools for mapping and scheduling applications on the NoC
  - Communication libraries for coding applications
- There is a lack of availability of real large applications which can be used you evaluate current research proposals
  - Researchers still use random traffic and random task graphs

# Important References

1. Shashi Kumar et. al., *"A Network on Chip Architecture and Design Methodology"*, SVLSI 2002, Pittsburgh, 2002.
2. Juha-Pekka Soininen et. al. ," *Extending Platform based design to Network on Chip Systems*", Proceedings of the 16th International Conference on VLSI Design 2003, India, Jan. 2003.
3. Juha-Peka Soininen et. al. *"Mappability Estimation Approach for processor Architecture Evaluation", Proceedings of the 20th NORCHIP 2002, Nov. 2002, Copenhagen, Denmark.*
4. Evegeny Bolotin et. al., *"QNoC: QoS Architecture and Design Process for Network on Chip*", Journal of System Architecture 50 (2004), 105-128.
5. Evegeny Bolotin et. al., *"Cost considerations in Network on Chip*", Integration – The VLSI Journal, special issue on Network on Chip, Volume 38, Issue 1, October 2004, pp. 19-42.
6. Tang Lei, Shashi Kumar, *" A two step genetic algorithm for mapping task graphs to a NoC Architecture"*, Proceedings DSD 2003, Turkey, Sept. 2003.
7. Ruxandra Pop and Shashi Kumar, *"A survey of techniques for mapping and scheduling applications to Network on Chip Systems*", Research Report 04:4, School of Engineering, Jönköping University, Dec. 2004, ISSN 1404-0018.