codesign environment

# Schedule

- 1. Introduction
- 2. System Modeling Language: System C *
- 3. HW/SW Cosimulation *
- 4. C-based Design *
- 5. Data-flow Model and SW Synthesis
- 6. HW and Interface Synthesis
  (Midterm)
- 7. Models of Computation
- 8. Model based Design of Embedded SW
- 9. Design Space Exploration
  (Final Exam)
  (Term Project)

# References

- Partitioning


[33] Hyunok Oh and Soonhoi Ha, "A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling," CODES Workshop, May 1999.

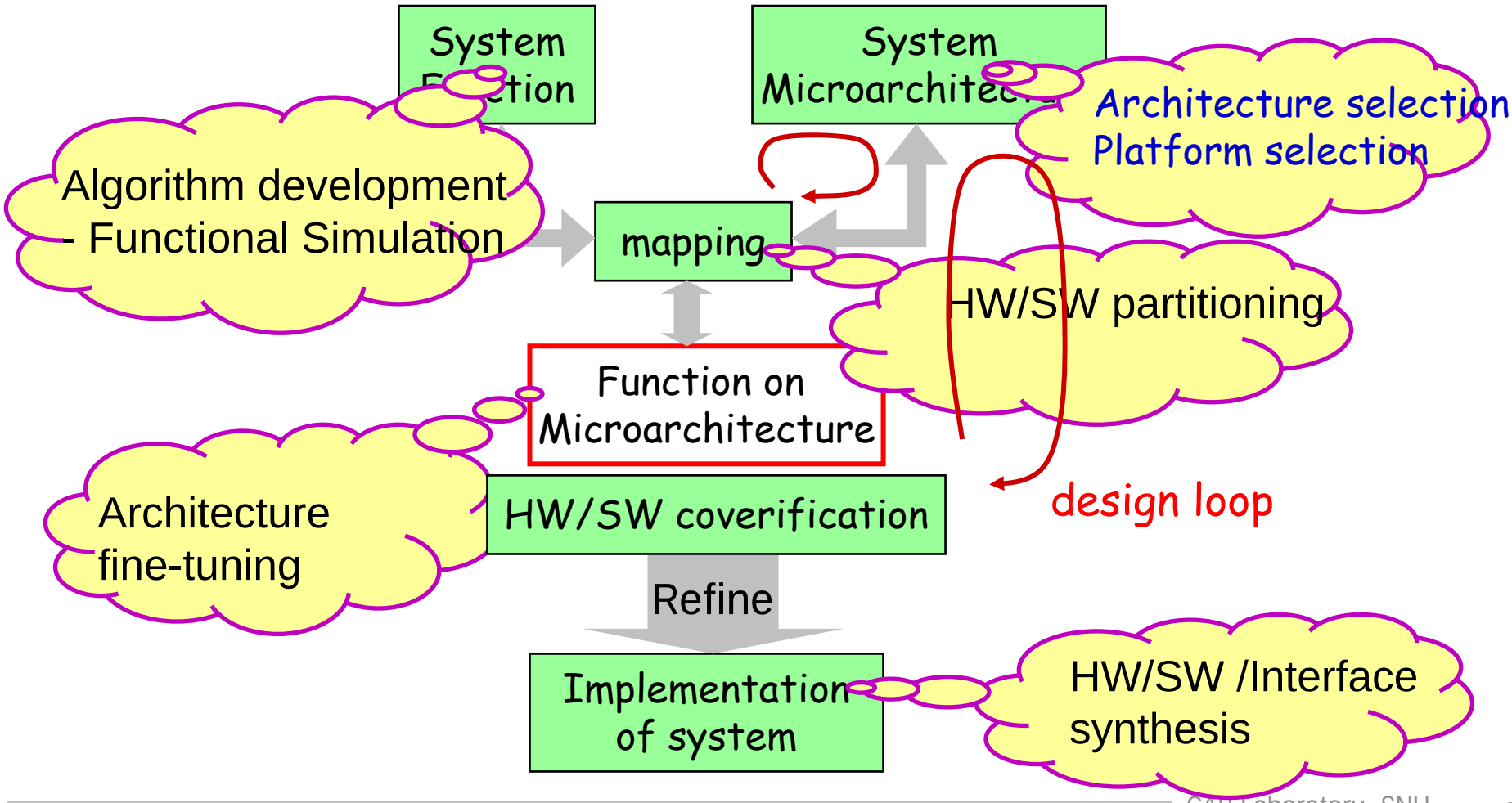
[34] Hyunok Oh and Soonhoi Ha, "Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints", CODES, 2002


- Communication Architecture Exploration


[35] Sungchan Kim, Chaeseok Im, Soonhoi Ha, "Schedule-Aware Performance Estimation of Communication Architecture for Efficient Design Space Exploration," IEEE Transactions on Very Large Scale Integration systems, Vol. 13 pp 539-552 May 2005

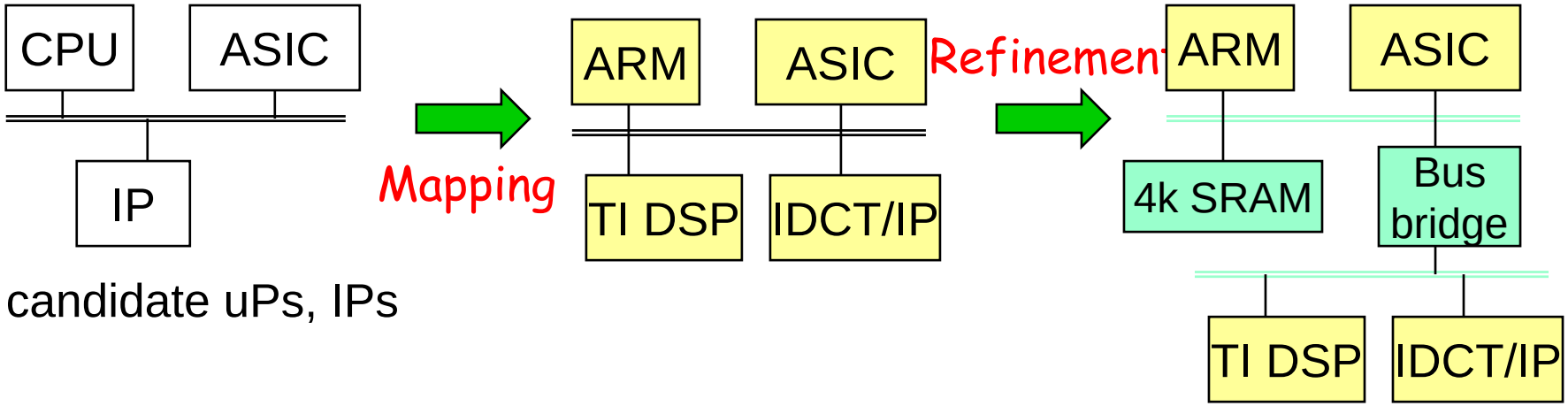
[36] SungChan Kim and Soonhoi Ha, "Efficient Exploration of Bus-Based System-on-Chip Architectures", IEEE Transactions on Very Large Scale Integration systems, Vol. 14 pp 681-692 July 2006

# Remind: Y-chart for DSE



# Outline of PeaCE Approach

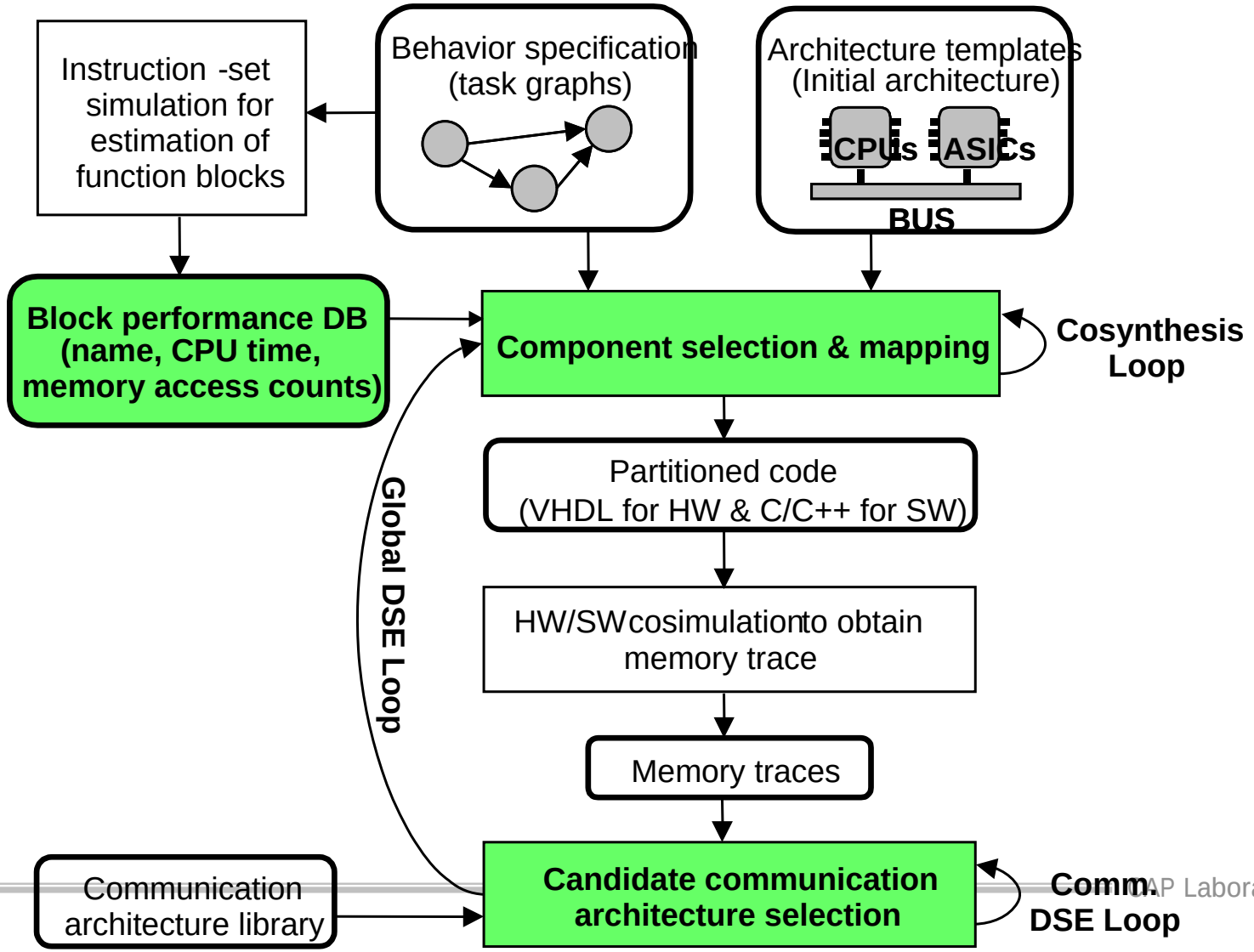
- **PeaCE**
  - Drag-and-drop architecture template
  - Partitioning: component selection and mapping
  - Architecture refinement: memory and channel refinement
    - Trace-driven performance evaluation (on-going work)
  - Time-accurate fast cosimulation: design validation



# Current Practice

- **Commercial Tools**
  - Mentor Platform Express, Synopsys System Studio, CoWare ConvergenSC
  - Drag-and-drop platform selection ← design library (architecture IP)
  - Manual mapping
  - Manual design refinement + communication synthesis
  - Fast performance evaluation: abstract modeling, virtual processor

# Two-step design space exploration in PeaCE



codesign environment

# Contents

- **Block Performance Estimation**
  - Block Execution Time Estimation
- **Power Estimation**
- **HW/SW Cosynthesis**
- **Communication Architecture Exploration**

# Block Performance Estimation

- **Assume that hardware performance is given.**
- **Software performance estimation is still needed. Why?**
  - Architecture dependency
    - Memory access time, processor type, etcs…
  - Compiler optimization
  - Data dependency
    - Software performance is usually data dependent

# Performance Estimation Method

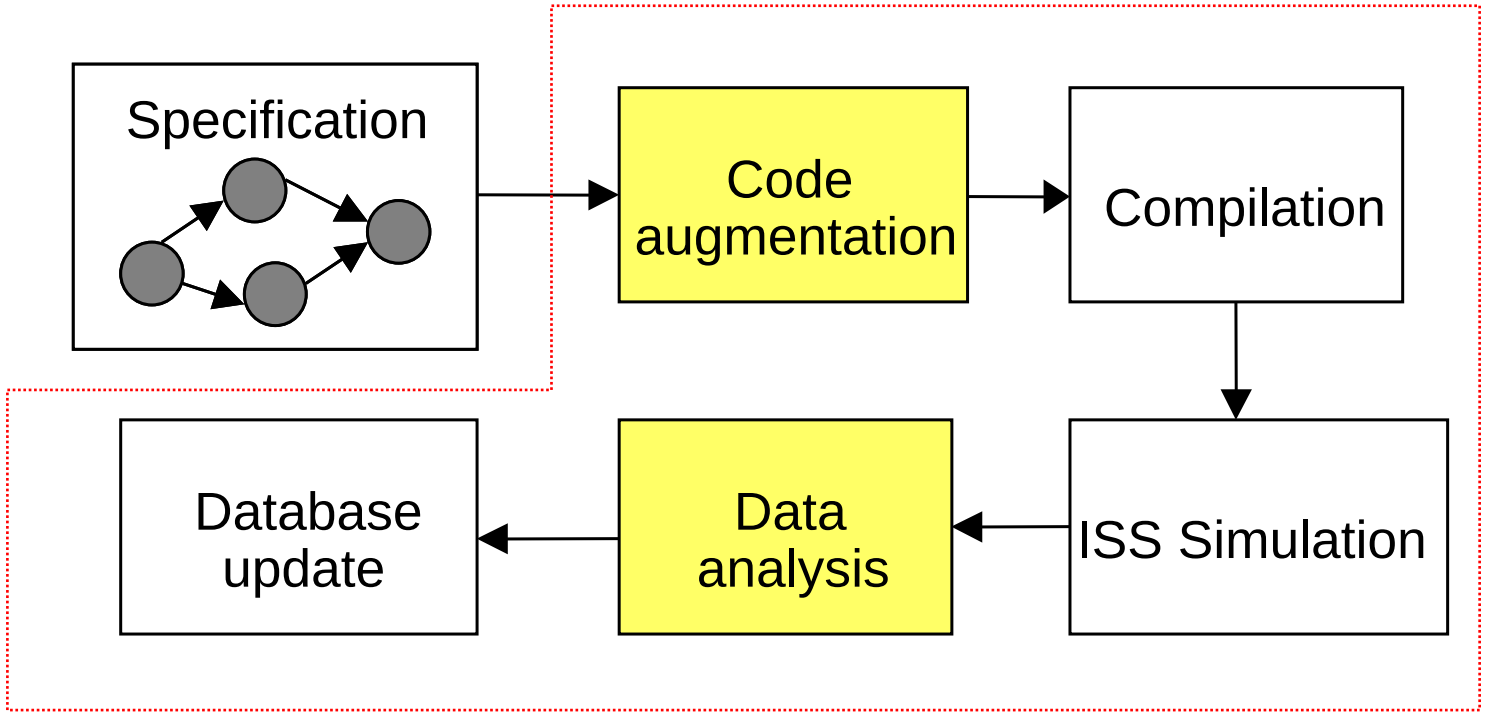
- **Simulation-based approach**
  - ISS (Instruction Set Simulator) or compiled simulation
  - Estimate average performance
  - Slow
- **Analytical approach**
  - Usually estimate the worst case performance
  - Fast
- **Hybrid approach**
  - Trace driven approach
  - Using a virtual processor

# PeaCE Approach

- **Simulation-based approach: Use ISS (Instruction Set Simulator)**
  - Long simulation time is tolerable before DSE
  - Architecture features are included
  - Use of target compiler with all optimization options turned on

- **Two possible scenarios**
  - Separate test bench for each functional block
    - Time consuming
    - Not easy to make test vectors
  - Use the application program to estimate the performances of all functional blocks at once for each processing element
    - Can consider application-specific data dependent behavior

# Proposed Scheme



# Code Augmentation

- **First try: add dummy function at the boundary of each function block and set break-points there in ISS**

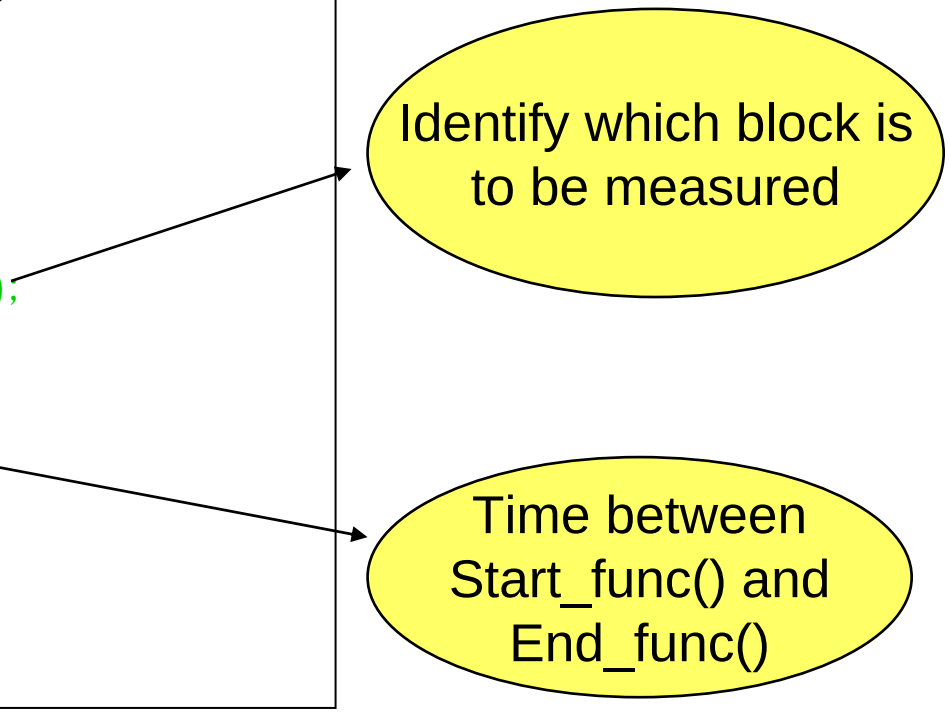
```
void Print_func(Block name, Factor value) {
Print Block name and Factor value;
}
void Start_func() {return;}
void End_func() {return;}
…
Print_func(Block name, Factor value);

/* the function block */

…
```

**Removed or repositioned after compiler optimization**

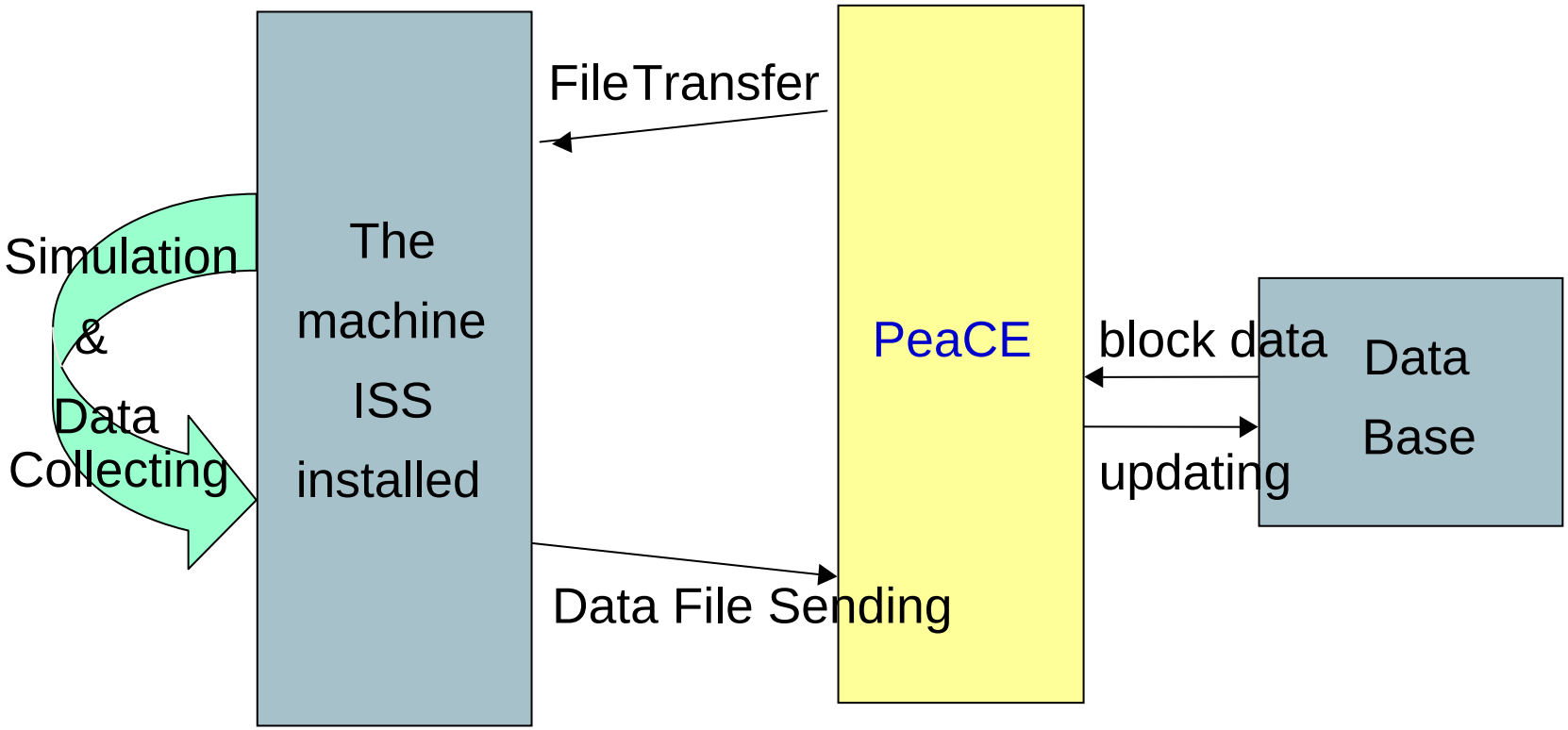
# Code Augmentation

- **Add control dependency: position change is not allowed**
- **Use "extern" declaration to hide the body of the dummies**

```
void print_func(Block name, Factor value)
{print Block name and Factor value;}
extern int Start_func();
extern void End_func();
…
print_func(Block name, Factor value);
If(Start_func()==true) {
/* the function block */
}
End_func();
…
```



# PeaCE + ISS + Database



# ISS Scripts (ex: Armulator)

```
void print_func(Block name, Factor value)
{print Block name and Factor value;}
extern int Start_func();
extern void End_func();
…
print_func(Block name, Factor value);
If(Start_func()==true) {
/* the function block */
}
End_func();
…
```

break-points
("b @print_func")

```
…
go
p $statistics_inc
go
p $statistics_inc
go
p $statistics_inc
…
```

Armsd -script arm_script > arm_out

# ISS Simulation Output

- **Example: Armulator**
  - Trace output file from “armsd”

```
……
armsd: step
Step completed at PC = 0x00008184 (main + 0x8c)
+008c 0x00008184: 0xe28f1f76  v... :    add      r1,pc,#0x1d8 ;
#0x8364
armsd: p $statistics_inc
struct {
  Instructions = 8, Core_Cycles = 22, Core_ID = 4, Core_IOnly = 7,
  Core_Idles = 0, Core_DOnly = 0, Total = 11}
armsd: go
Breakpoint #3 at PC = 0x000080a8 (print_func + 0)
```

# Data Analysis (Execution time)

```
XMgraph arm720T normal null 72 0 0 0 3171 0 0 0
Mpy arm720T normal 2 32 0 0 0 89 0 0 0
Add arm720T normal 2 32 0 0 0 58 0 0 0
Fork arm720T normal null 0 0 0 0 4 0 0 0
Abs arm720T normal null 40 0 0 0 24 0 0 0
Const arm720T normal null 20 0 0 0 9 0 0 0
```

for DB input

for user output

| Name | factor | count | total | min | max | m_total | m_min | m_max |
|---|---|---|---|---|---|---|---|---|
| Xmgraph | null | 10 | 30952 | 2441 | 3171 | 720 | 72 | 72 |
| Mpy | 2 | 10 | 890 | 89 | 89 | 320 | 32 | 32 |
| Add | 2 | 10 | 580 | 58 | 58 | 320 | 32 | 32 |
| Fork | null | 20 | 80 | 4 | 4 | 0 | 0 | 0 |
| Abs | null | 10 | 240 | 24 | 24 | 400 | 40 | 40 |
| Const | null | 10 | 90 | 9 | 9 | 200 | 20 | 20 |

# Data Analysis (Memory Access)

name processor factorValue count seq_r_min seq_r_max seq_r_toral seq_w_min seq_w_max seq_w_total non_r_min non_r_max non_r_total non_w_min non_w_max non_w_total

---

XMgraph arm720T null 10 2403 3069 29982 298 343 3385 0 0 0 0 0 0
Mpy arm720T 2 10 72 72 720 6 6 60 0 0 0 0 0 0
Add arm720T 2 10 55 55 550 2 2 20 0 0 0 0 0 0
Fork arm720T null 20 4 4 80 0 0 0 0 0 0 0 0 0
Abs arm720T null 10 26 26 260 0 0 0 0 0 0 0 0 0
Const arm720T null 10 11 11 110 2 2 20 0 0 0 0 0 0

# What information do we need?

- **Entire application performance estimation**

$$P_{est} = \sum_{B_k \text{ on } CP} n(k) \times \{P(k,i) + m(k) \times n_m + c(k,l) \times n_c\} \quad (1)$$

For a function block $B_k$,

$P(k,i)$ estimated performance number on PE

$m(k)$ memory access counts

$c(k,l)$ communication requirements to the next block

$n(k)$ number of invocations of block

$n_m$ memory access overhead

$n_c$ channel communication overhead

# Database Information

- **Block name**
- **Factor value: factors that affect the block performances**
  - (ex) adder: number of inputs, filter: number of taps
- **Processor and compiler option**
- **CPU performance**
  - with perfect cache assumption if 1st level cache exists
  - worst/best/average numbers
- **Memory access counts (after cache miss)**
  - sequential/non-sequential read/write counts
- **Code size**
- **Expected power**

# Experimental Environments

- **Application : H.263 Encoder**
- **Processor : arm720T 100Mhz**
  - 8k first level cache (write-through)
- **ISS : Armulator**
- **Input : QCIF(176*144) format 10 frame**
- **Architecture assumption**
  - Cache Miss penalty :
    - Sequential Read : 2 Cycles
    - Nonsequential Read : 5 Cycles
  - Write through cache / Write buffer
    - Write penalty = 0 (for ideal case)

# Expected Performance

$$\sum_i \{N_i * (t_i + sn_i * sp_i + nn_i * np_i)\}$$

- $N_i$ : execution count of Block i
- $t_i$ : CPU performance
- $sn_i$ : non sequential read miss counts
- $sp_i$ : non sequential read miss penalty
- $nn_i$ : sequential read miss counts
- $np_i$ : sequential read miss penalty

codesign environment

# CPU Performance Only

| Block Name | counts | maximum | total | (%) |
|---|---|---|---|---|
| Variable legnth coding | 990 | 1995135 | 3425854 | 1.75 |
| Motion Compensation | 10 | 396050 | 3812721 | 1.95 |
| IDCT | 5940 | 1676 | 1449361 | 0.74 |
| Quantization | 5940 | 1390 | 7372282 | 3.77 |
| DCT | 5940 | 2781 | 14276766 | 7.30 |
| Motion Estimation | 990 | 17274135 | 152333063 | 77.93 |
| Etc. | | | 12797937 | 6.55 |
| Sum | 10 | | 195467984 | 100.00 |

HW? Optimize?

Total execution cycles : 194507308 ➔ error : 0.494%

codesign environment

# Memory Access Counts

| Block Name | Count | NS Read | % | S Read | 비율(%) |
|---|---|---|---|---|---|
| VLC | 990 | 58434 | 4.77 | 175302 | 4.77 |
| Motion Comp. | 10 | 147517 | 12.04 | 442581 | 12.04 |
| IDCT | 5940 | 23866 | 1.95 | 71602 | 1.95 |
| Quantization | 5940 | 80501 | 6.57 | 241503 | 6.57 |
| DCT | 5940 | 245810 | 20.06 | 737430 | 20.06 |
| Motion Est. | 990 | 384957 | 31.42 | 1154861 | 31.42 |
| Etc. | | 283936 | 23.18 | 852379 | 23.19 |
| Total | 10 | 1225021 | 100.00 | 3675658 | 100.00 |

Total: Nonseq. Read = 1036444 ➔ error = 18.19%
Seq. Read = 3109332 ➔ error = 18.21%
Execution time: **208944405 vs 205908192** ➔ error = 1.47%

➡ Side effect from code augmentation !

# Remaining Problems

- **Compensation for the increased cache miss penalty**
  - How to reduce the side effect?
- **Write through cache / write buffer does not imply write miss penalty = 0**
  - Write cycles increase read miss penalty
  - Not easy to formulate correctly: our experiments, 3 cycles write miss penalty should be added
  - How to consider this effect?
- **What is the best representative number?**
  - Average, best, worst, or what-else?

codesign environment

# Contents

- **Block Performance Estimation**
  - Block Execution Time Estimation
- **Power Estimation**
- **HW/SW Cosynthesis**
- **Communication Architecture Exploration**

# Component Power Estimation

- **Cache/SRAM memory**
  - Rely on knowledge of capacitances of each portion of cache design
  - CACTI by Wilton and Jouppi
- **Microprocessor**
  - instruction level power analysis
    (ex) lw,sw consume little more power than others
  - V. Tiwari's work
  - Non-ideal effects amortize differences: on the average, the CPU power is about the same independent of the instructions
- **Cycle-accurate power measurement**
  - CPU, memory (SRAM,DRAM), bus, FPGA,…
  - conducted by Prof. Chang, SNU

# System Power Estimation

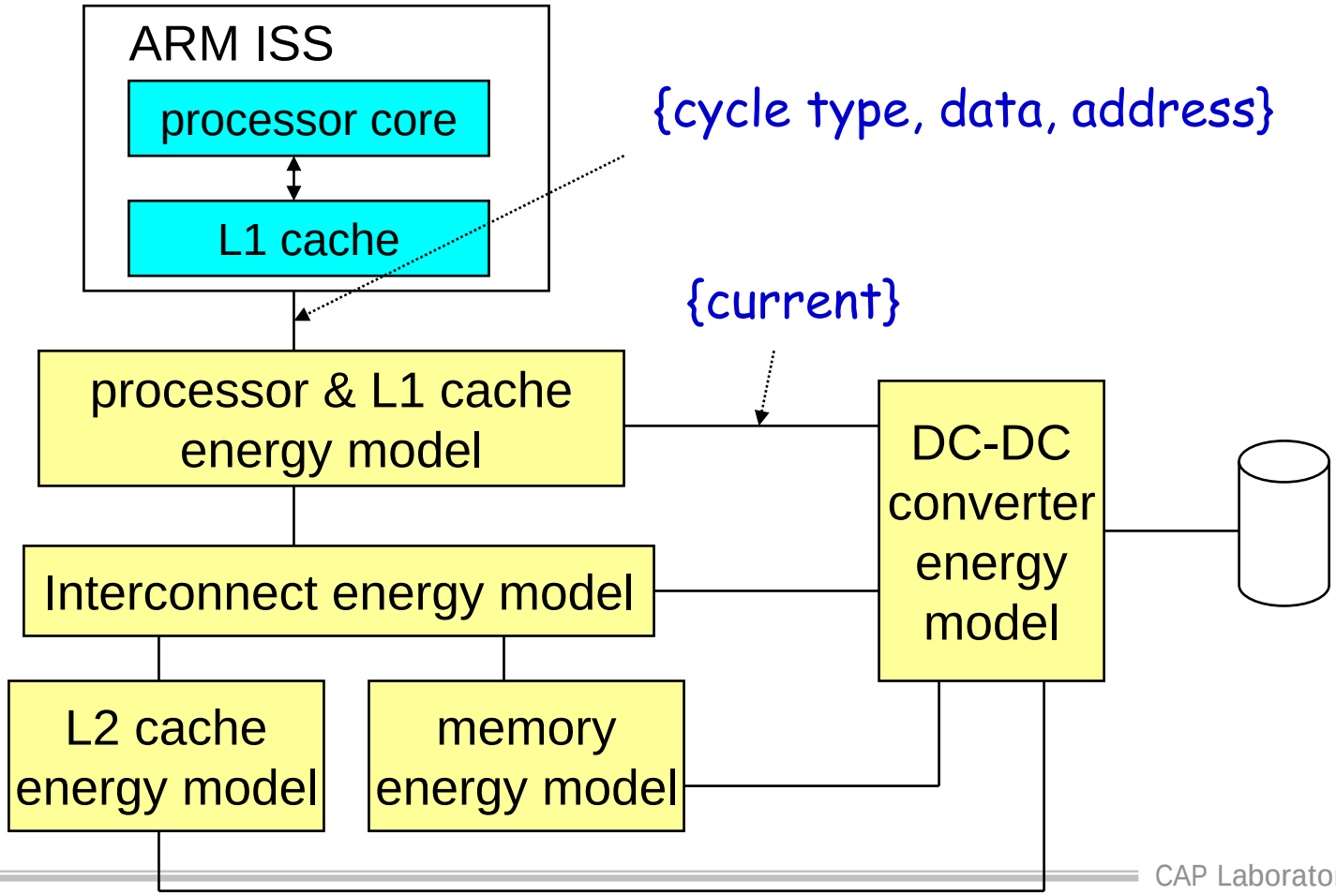
- **Lower-level power estimation**
  - Circuit-level, gate-level, architecture-level
  - Accurate, but expensive
  - Late in the design cycle
  - Usually for design validation, and late optimization

- **System-level power estimation**
  - Additive model (+ cycle accurate ISS for accurate modeling)

$$E_{System} = \sum_{i} E_{Component}(i) + E_{Static}$$

# Stanford Approach (1)

- **T. Simunic, L. Benini, and G. De Micheli (DAC '99)**



# Stanford Approach (2)

- **Processor**
  - active state
  - idle state

$$E_{p,a} = C_{p,a} V^2, \quad C_{p,a} = \frac{P_m}{V_m^2 f_m}$$

from data sheet

- **Memory and L2 cache**
  - active state

$$E_{m,a} = \frac{C_{m,a} V^2}{N_{wait} + 1}, \quad C_{m,a} = \frac{P_m}{V_m^2 f_m}, \quad N_{wait} = \frac{T_{mem}}{T_{cycle}}$$

  - idle state

$$E_{m,i} = T_{cycle} \sum P_i \delta_i$$

power x user given percent in idle state i

- **Interconnect and pins**
  - Line capacitance from manufacturer or measurement
  - Pin capacitance from data sheet

$$E_{line} = \frac{N_{switch} C_{switch} V^2}{N_{wait} + 1},$$

- **DC-DC converter**

# Stanford Approach (3)

- **Validation of simulation methodology**
  - SmartBadge prototype:
    - StrongARM-1000 (core 1.5V), FLASH and SRAM, all 3.3V
  - Dhrystone benchmark: error 5%
- **MPEG Decoder System Design Exploration**

| name | Inst. | Data | L2 cache |
|---|---|---|---|
| **original** | FLASH | SRAM | no |
| **L2 cache** | FLASH | BSDRAM | yes |
| **burst SRAM** | BFLASH | BSRAM | no |
| **burst SDRAM** | BFLASH | BSDRAM | no |

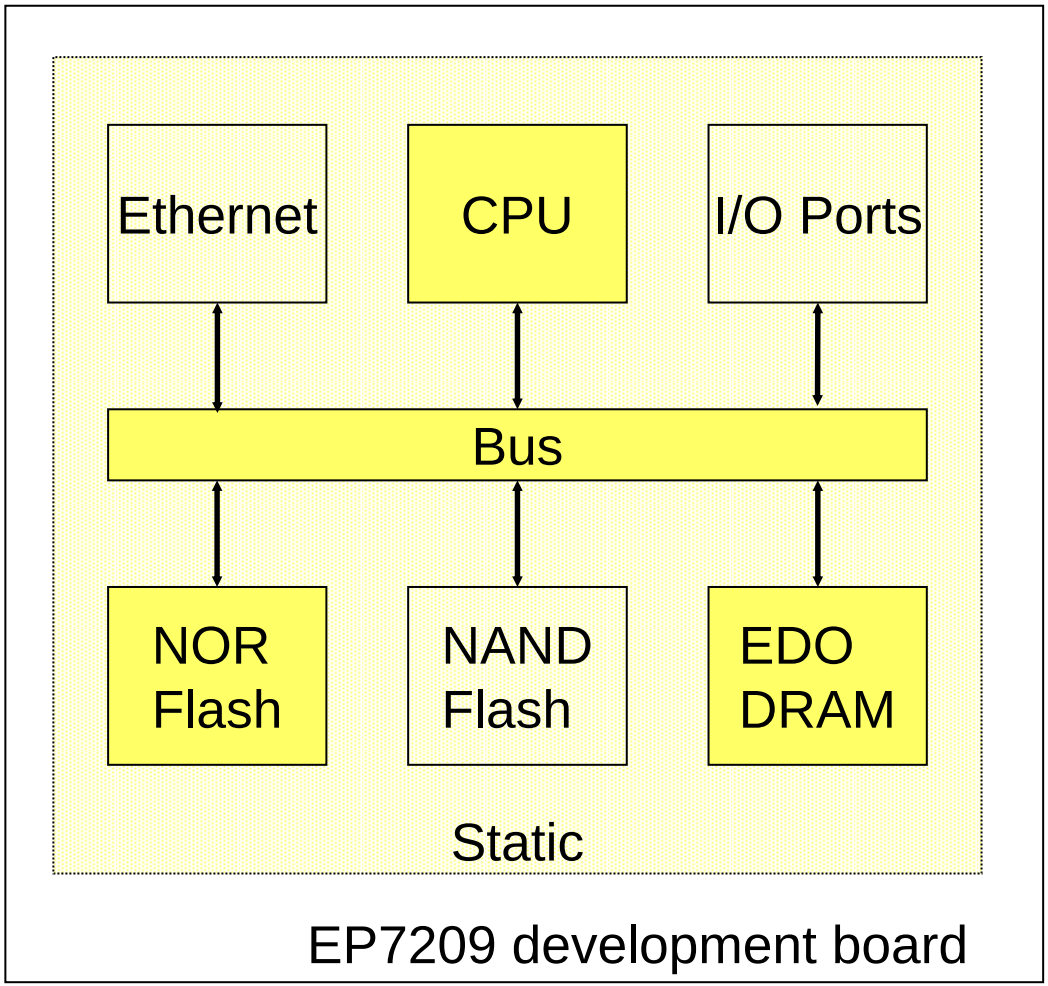
| configuration | I | P | B |
|---|---|---|---|
| **IPBB@30** | 2 | 3 | 7 |
| **IP@30** | 2 | 10 | 0 |
| **IPPI@30** | 4 | 8 | 0 |
| **IPPI@25** | 4 | 8 | 0 |

most energy-efficient

# System-level Power Estimation

- **Additive model**
  - $y = \beta 0 + \beta 1\, x1 + \beta 2\, x2 + \ldots + \beta n\, xn$
    - $\beta n$ : parameter, xn : estimates, y : response
- **Modeling accuracy depends on parameter accuracy**
- **Three ways to obtain the parameter values**
  - data sheet
    - not always possible. In many cases, range is given.
  - individual measurement
    - usually not feasible.
  - regression analysis
    - need many data sets
    - not for design space exploration: hardware should be given
    - but for design optimization once technology is given

# Experimental System




$$E_{System} = E_{Cpu} + E_{Memory} + E_{Bus} + E_{Static}$$

# Component Power Modeling

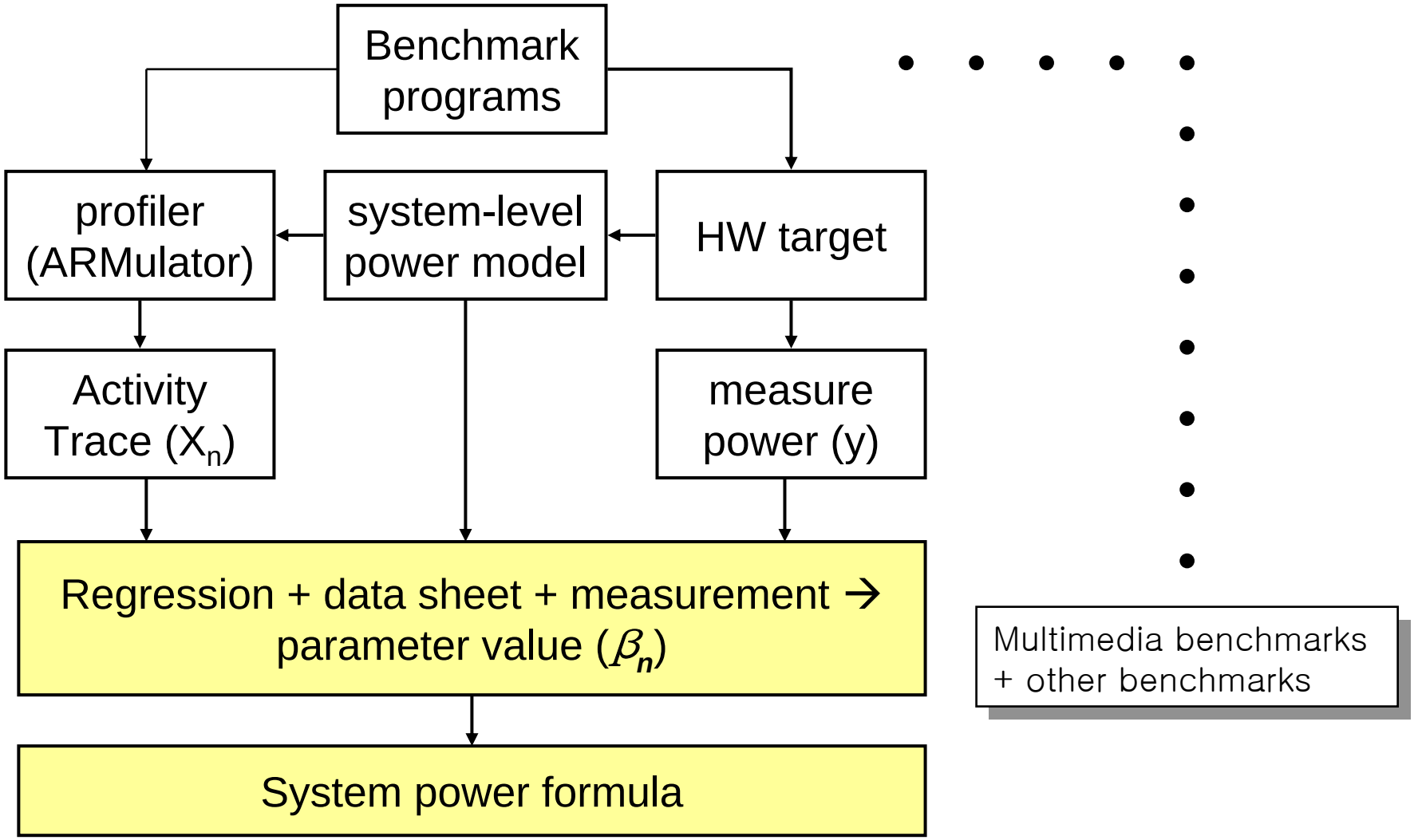
- **Processor**
  - Active power is given from the data sheet
  - Idle power is obtained from measurement
  - $E_{Cpu} = P_{CpuActive} \times T_{CpuActive} + P_{CpuIdle} \times T_{CpuIdle}$
- **Memory**
  - Non-sequential/sequential, read/write
  - $E_{Memory} = E_{Nonsequential\_Read} \times N_{Nonsequential\_Read} + E_{Nonsequential\_Write} \times N_{Nonsequential\_Write} + E_{Sequential\_Read} \times N_{Sequential\_Read} + E_{Sequential\_Write} \times N_{Sequential\_Write}$

# Component Power Modeling

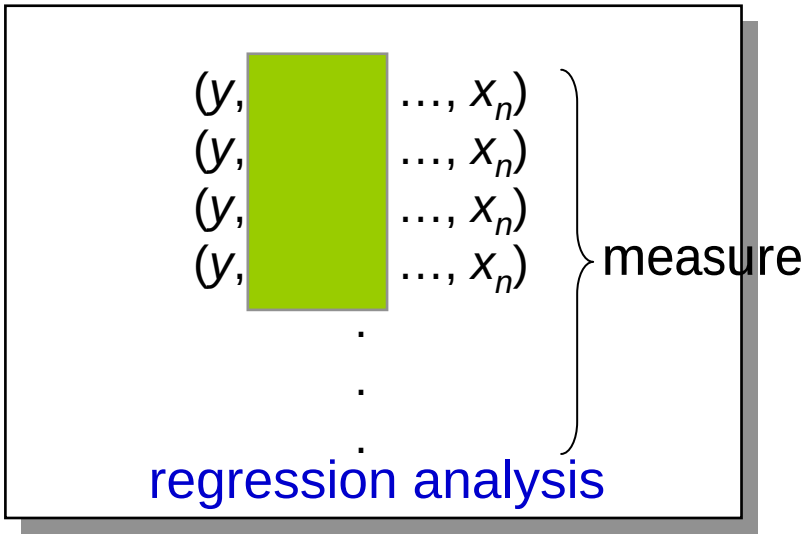
- **Bus**
  - Depends on which type of bus is used
    - float bus - Hamming distance
    - pull-up bus – number of 0/1
  - Include bus-driver power
  - In case of float bus $E_{Bus} = E_{HammingDistance} \times N_{BitToggles}$

- **Static power**
  - Total sum of idle components
  - Power from static currents

# Power Estimation Flow



# Solution Space Reduction




Required data set: N~2*N

Not easy to obtain 2*N independent data set

We reduce the number of parameters from 11 to 5

$$y = \beta_0 + \square + \ldots + \beta_n x_n$$

$$y - (\beta_1 x_1 + \beta_2 x_2) = \beta_0 + \ldots + \beta_n x_n \quad \bullet \bullet \bullet$$

# Experiments

from data sheet

| | |
|---|---|
| $P_{ProcActive}$ | 0.170 |
| $P_{NorFlash\_Nonsequential\_Read}$ | 0.132 |
| $P_{NorFlash\_Sequential\_Read}$ | 0.132 |

measurement

| | |
|---|---|
| $P_{ProcIdle}$ | $0.347 \times P_{ProcActive}$ |
| $P_{RamNonsequentialWrite}$ | $P_{RamNonsequentialRead}$ |
| $P_{RamSequentialWrite}$ | $P_{RamSequentialRead}$ |
| $P_{DataBus}$ | $2.06 \times P_{AddressBus}$ |

Regression test

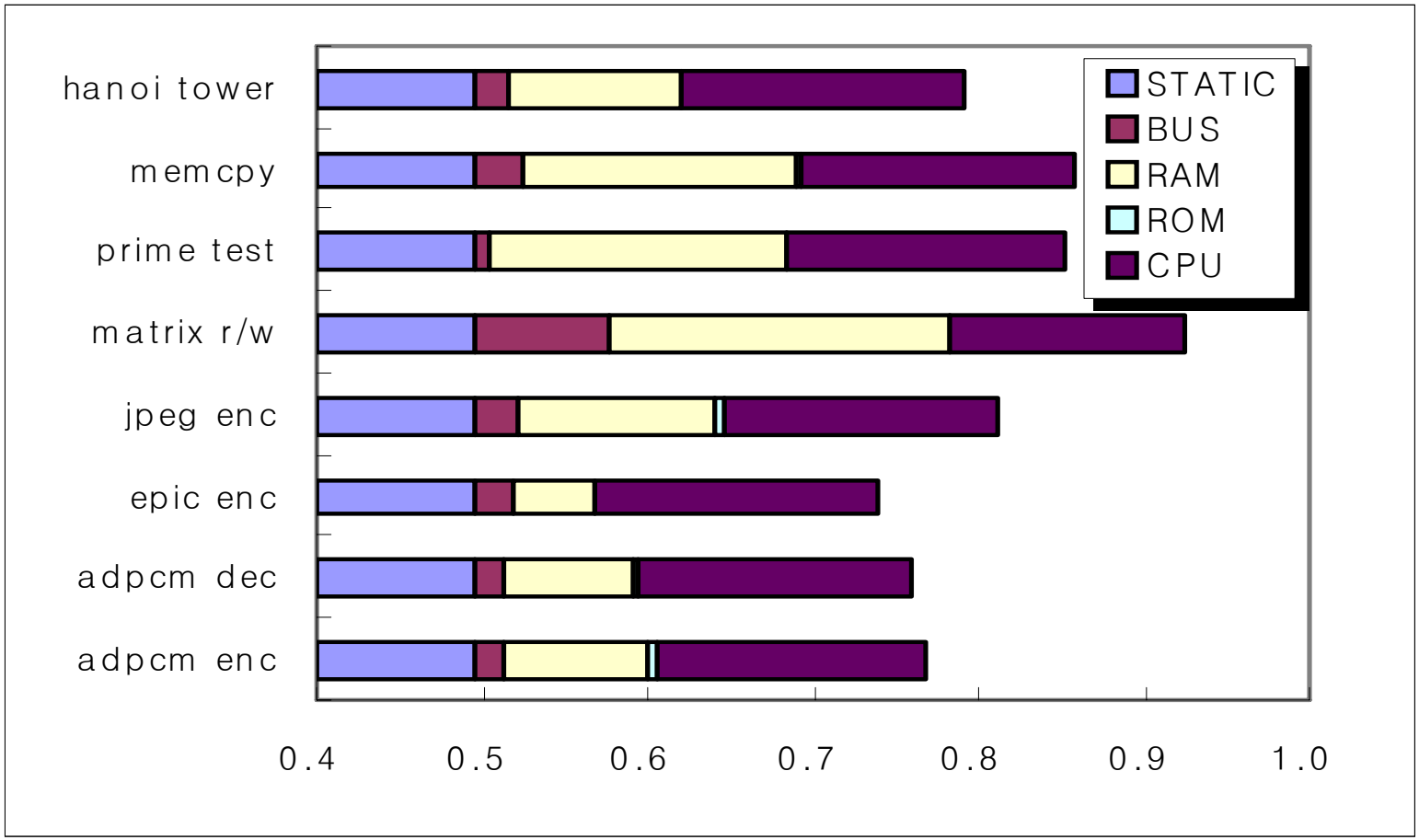
| | |
|---|---|
| $P_{RamNonSequentialRead(Write)}$ | $2.927 \times 10^{-8}$ |
| $P_{RamSequentialRead(Write)}$ | $5.448 \times 10^{-9}$ |
| $P_{DataBus}$ | $1.792 \times 10^{-10}$ |
| $P_{AddressBus}$ | $3.692 \times 10^{-10}$ |
| $P_{Static}$ | 0.496 |

# Experimental Results

- **From benchmark programs**

| Unit : mW | estimates | actual | error(%) |
|---|---|---|---|
| adpcm enc | 0.7617 | 0.7795 | -1.4 |
| adpcm dec | 0.7516 | 0.7590 | -0.1 |
| epic enc | 0.7271 | 0.7430 | -0.7 |
| jpeg enc | 0.7997 | 0.8320 | -2.5 |
| mp3 enc | 0.7476 | 0.7613 | -1.8 |
| matrix r/w | 0.9308 | 0.9263 | -0.1 |
| prime test | 0.8366 | 0.8191 | 4.0 |
| memcpy | 0.8265 | 0.8307 | 3.2 |
| Hanoi tower | 0.7906 | 0.8300 | -4.7 |

# Power Consumption Analysis



# Validation: More Experiments

- **For other programs**

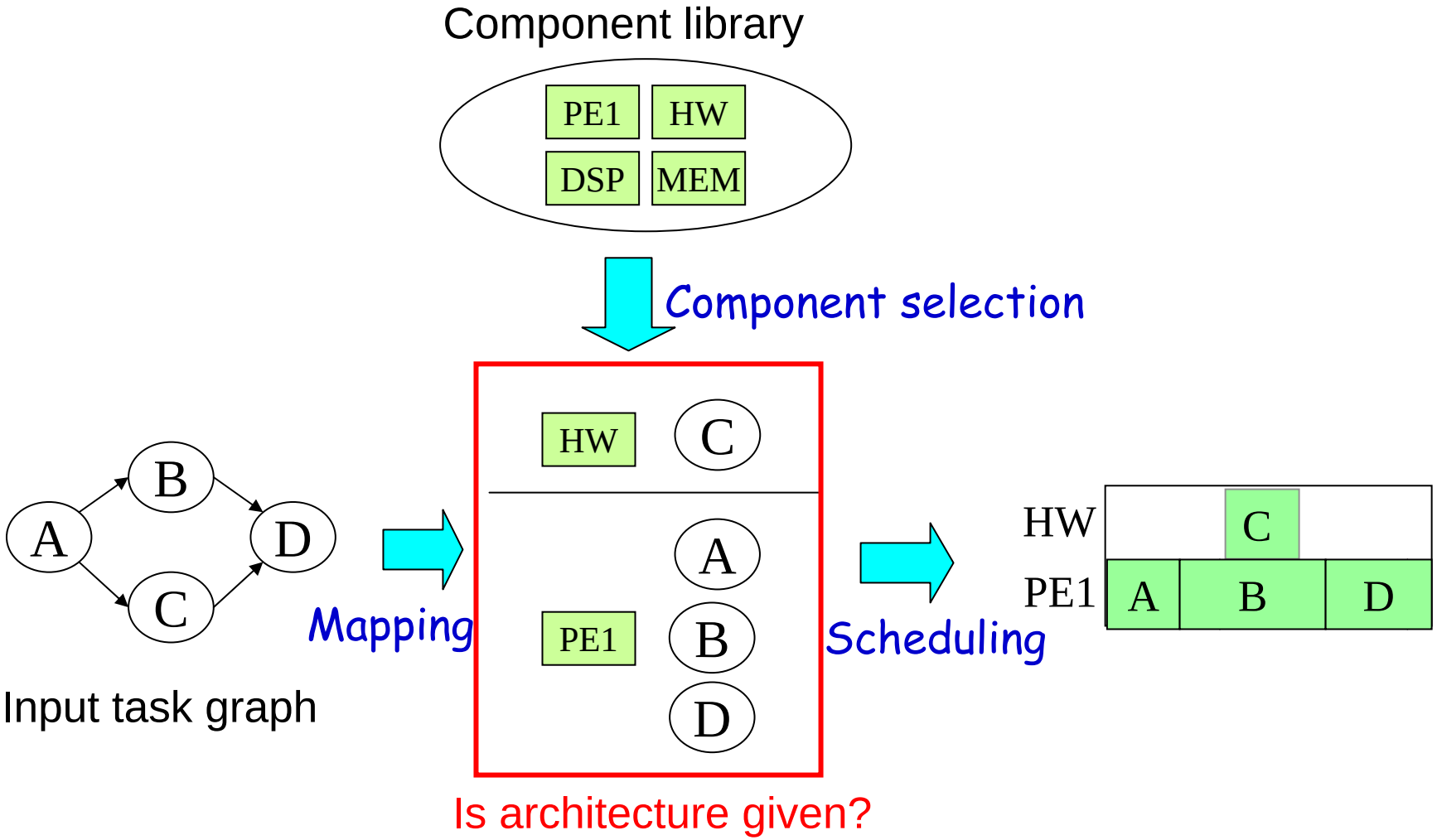
| Unit : mW | estimates | actual | error(%) |
|---|---|---|---|
| mpeg dec | 0.8645 | 0.8940 | -3.2 |
| h.263 dec | 0.7841 | 0.7742 | 1.4 |
| insert sort | 0.7609 | 0.7920 | -3.8 |
| shell sort | 0.7735 | 0.7689 | 0.8 |

codesign environment

# Contents

- **Block Performance Estimation**
  - Block Execution Time Estimation
- **Power Estimation**
- **HW/SW Cosynthesis**
  - Platform and component selection
  - Hardware/software partitioning
- **Communication Architecture Exploration**

# Cosynthesis Problem



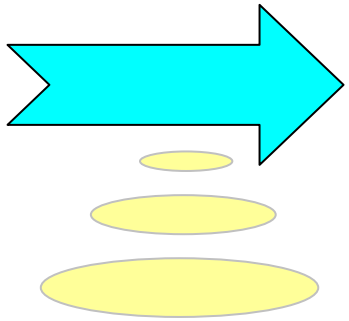
# Partitioning Problem

Problem space

Constraints
(HW area, power, timing, memory)

Input specification
(fixed acyclic task graph, time-shared multiple task graph)

Partitioning algorithms



Solution space

Fixed architecture

Synthesizable architecture
(component selection, interface synthesis, memory structure synthesis)

**Heuristics : list scheduling, clustering**
**hardly extensible**
**well-formulated techs: simulated annealing, genetic algorithm)**
**mathematical programming : ILP**
**too time consuming**

codesign environment

# Partitioning

- **Approaches**
  - software-oriented: from SW to HW to satisfy the timing constraint
  - hardware-oriented: from HW to SW to reduce the cost
- **Grain size**
  - task level
  - operation level, basic block level
  - hierarchical
- **Issues**
  - fixed target architecture or unknown target architecture
  - IPC overhead consideration
  - partitioning with scheduling
  - pipelining and/or parallelism
- **Partitioning is a well-known NP-hard problem in its simpler form!**

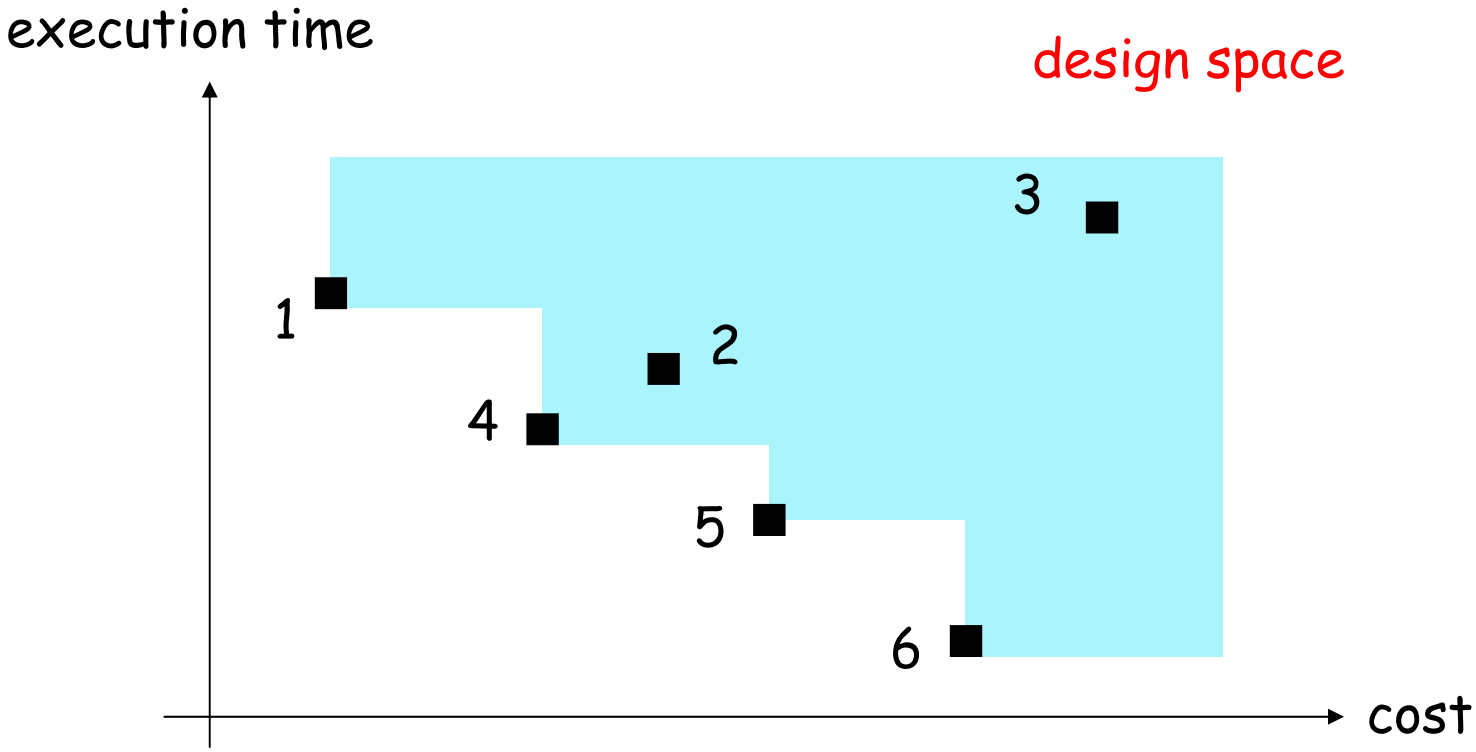
# Objective/Cost Functions

- **Primary objectives (properties of overall system)**
  - Cost (price)
  - Power dissipation
  - Speed
- **Secondary objectives (properties of a part of system)**
  - Utilization of computation and communication resources
  - Memory-specific metrics (e.g. cache misses)
  - Testability
- **Combined metrics**
  - Energy-delay product
  - Speed-area ratio



Courtesy by Prof. Taewhan Kim

# Pareto-optimal designs




- Design 2 is dominated by design 4
- Design 3 is dominated by all other designs
- Designs 1, 4, 5, 6: Pareto-optimal designs



Courtesy by Prof. Taewhan Kim

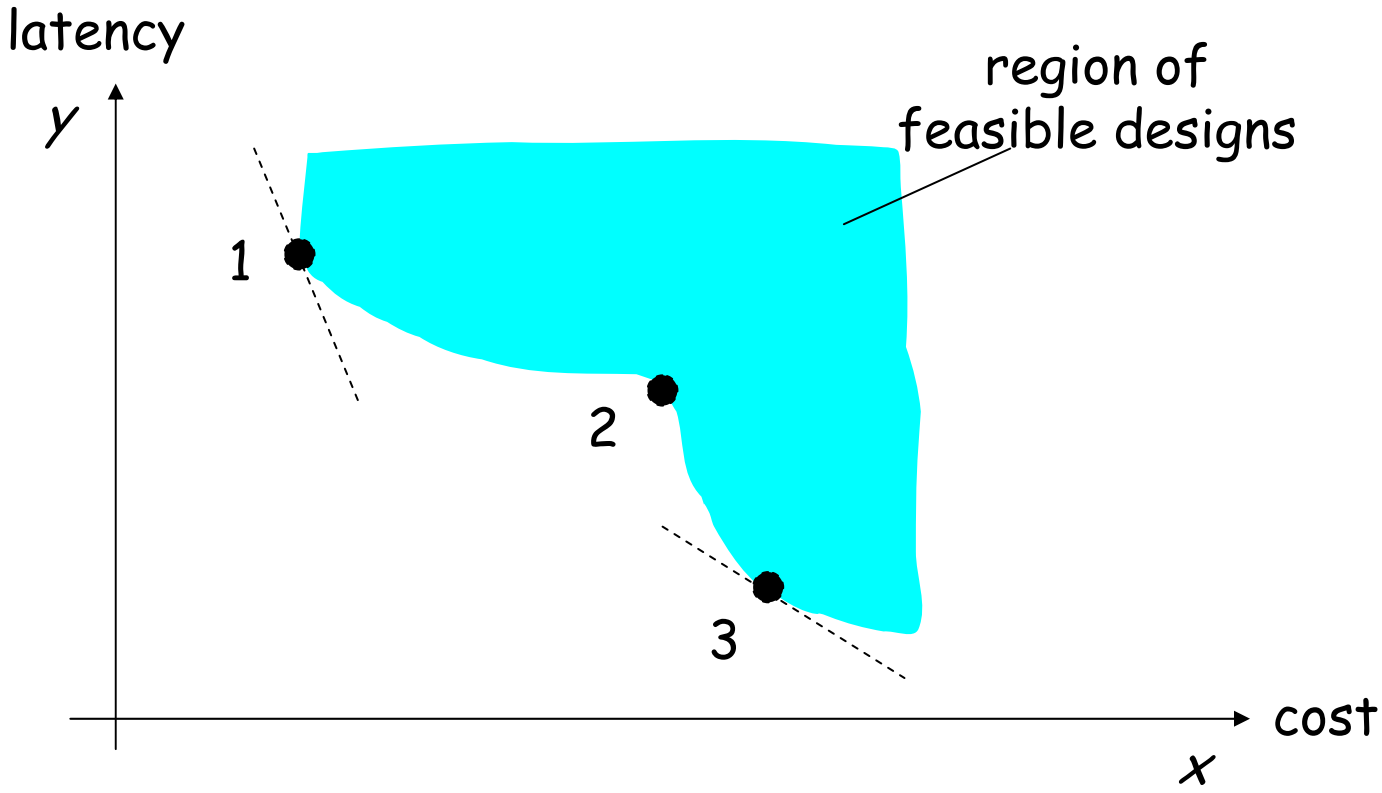
# Finding Pareto-optimal designs

- **Decision making before search**
  - The designer decides how to aggregate different objectives into a single objective (cost) function before the actual search is performed.
  - Examples: a weighted sum of objectives, a reduced number of objectives
  - Some design space may not be reachable at all
- **Search before decision making**
  - The search is performed to find a set of Pareto-optimal solutions. Only after the search, additional criteria or preferences are applied to find an optimal solution for a given problem
  - slow



Courtesy by Prof. Taewhan Kim

# Finding Pareto-optimal designs by a weighted sum of objectives



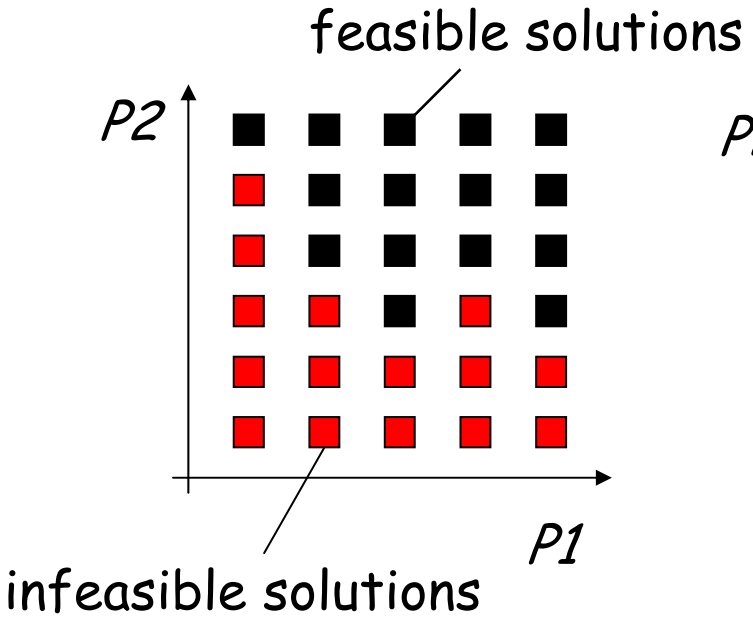

- Objective function: $f(x,y) = a * x + b * y$
- Design 2: unreachable



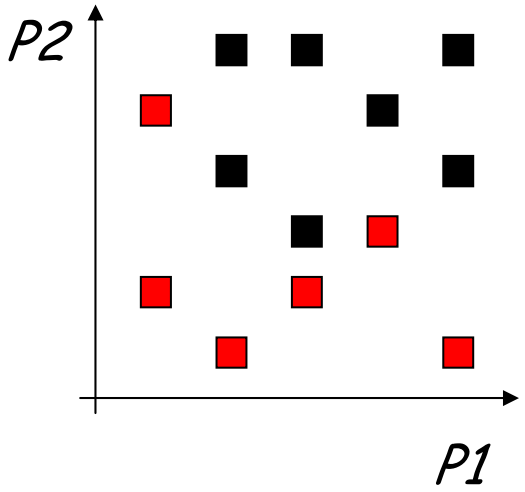
Courtesy by Prof. Taewhan Kim

# Strategies for Covering Design Space

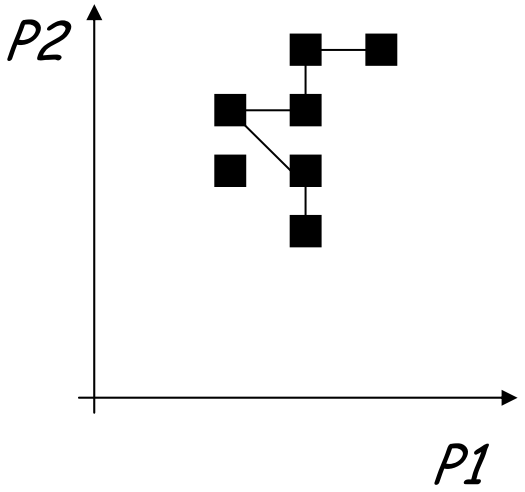
Exhaustive search




Random sampling




Knowledge-based (guided) search






Courtesy by Prof. Taewhan Kim

# Pruning the Design Space

- **Hierarchical exploration**
  - Step 1: Interesting (sub) regions of design space are identified and evaluated accurately.
  - Step 2: Higher-level design space exploration uses the refined information of the regions.
  - (ex) *branch-and-bound*
- **Sampling of the design space**
  - Good choice for an unbiased exploration where an exhaustive search is prohibitive.
  - (ex) *genetic algorithm, simulated annealing*
- **Subdividing the design space into independent parts for optimization**
  - Reduces the number of possible designs by dividing the optimization problem into independent subproblems



Courtesy by Prof. Taewhan Kim

# (Early) Partitioning Works

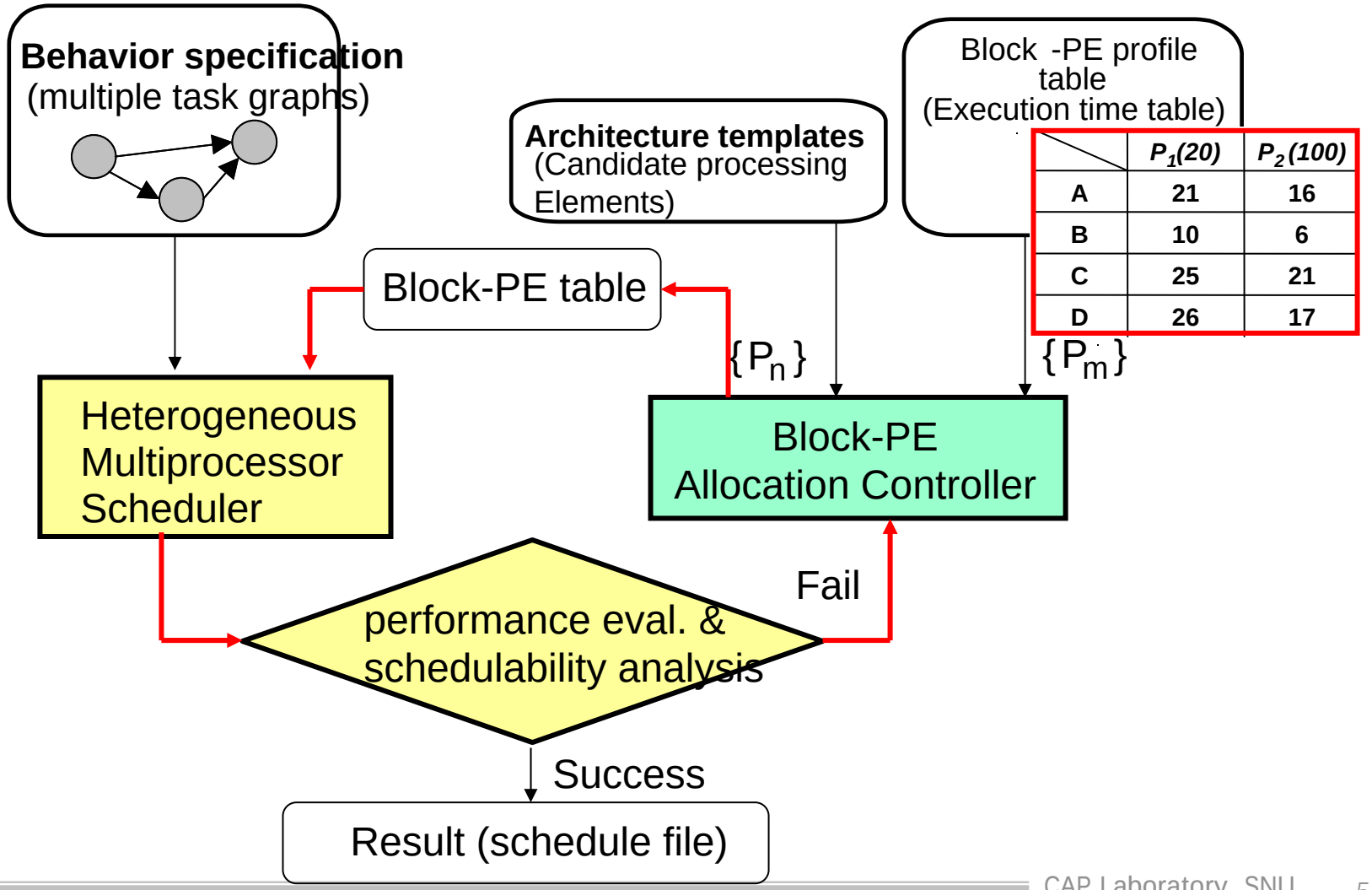
Courtesy by Prof. Taewhan Kim

| Group | Spec. | Application | Grain | Algorithm | Model | Architecture |
|---|---|---|---|---|---|---|
| Vulcan [Gupta 93] | Hardware C | Data oriented | Fine (instructions) | Iterative improvement | DFG set | Standard |
| Cosyma [Henkel 93] | C | Data oriented | Fine (basic blocks) | Simulated annealing | Syntactic tree | Standard |
| Ptolemy [Lee 97] | Silage | Real time | Coarse (tasks) | Constructive heuristic | CDFG | DSP based embedded system |
| [Vahid 97] | SpechCharts | Control | Coarse (processes) | Clustering, *min-cut*, S. Annealing | SLIF (access graph) | Standard |
| Lycos [Madsen 97] | C, VHDL | | Fine (basic blocks) | Dynamic programming | | |
| Cadlab [Peng 97] | VHDL | | Coarse (loops, processes) | S. Annealing Tabu search | Process graph | Standard |
| DDEL [Vermuri98] | C, VHDL | Data oriented | Coarse (tasks) | Genetic | Task graph | Standard |
| [Wolf 97] | Object oriented | Real time | Coarse (tasks) | Heuristic | Task graph | Parametric |

# Need More Research

- **Larger problem space should be considered**
  - New constraints
    - two-sided timing constraints
    - word-length optimization
    - QoS optimization
    - multiple constraints: memory, power, cost
  - wider input specifications
    - data parallelism
    - dynamic behavior
- **Automatic graph transformation**
  - pipelining, retiming, unfolding
- **Reasonably fast solution is needed**

# Proposed Cosynthesis Framework




| | $P_1(20)$ | $P_2(100)$ |
|---|---|---|
| A | 21 | 16 |
| B | 10 | 6 |
| C | 25 | 21 |
| D | 26 | 17 |

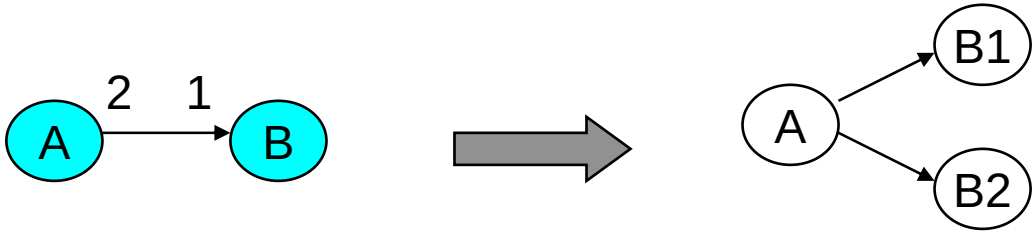
codesign environment

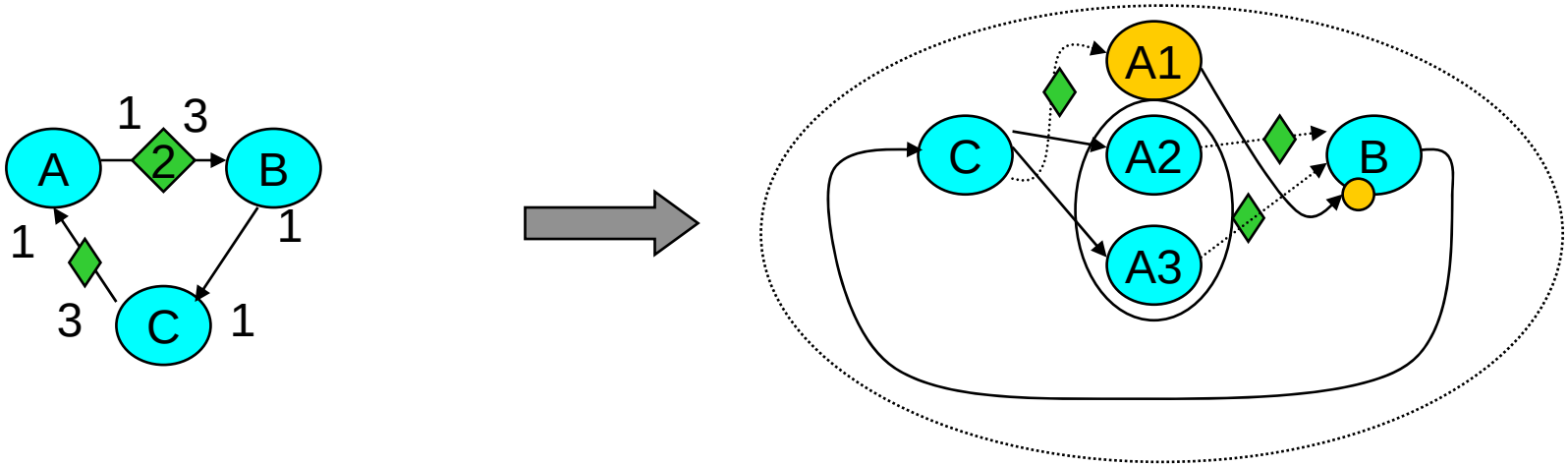
# The Characteristics

- **Input**
  - Directed Acyclic Graph
  - Task deadlines
- **Separation of concerns**
  - Component selection
  - Heterogeneous multiprocessor scheduling
  - Evaluation (including schedulability analysis)
- **Communication cost is considered in the HMP scheduler**
- **Resource sharing between multiple tasks are considered**

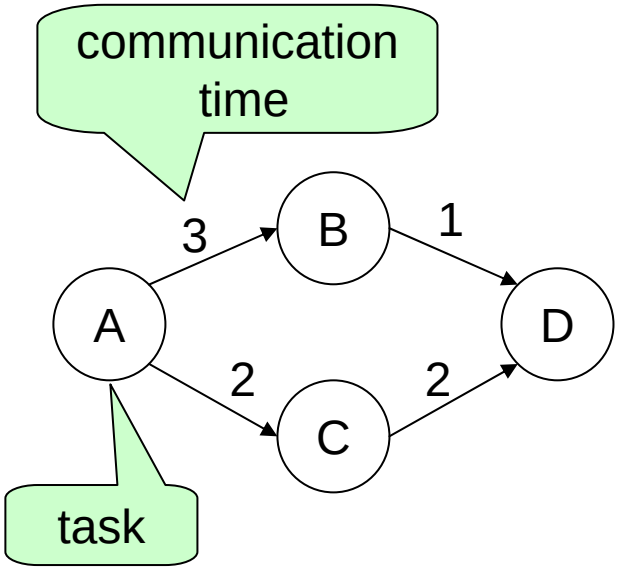
# ExpandedGraph

- **From SPDF to an acyclic graph**



# Inputs




The given time constraint = 32

**Acyclic Input Graph**

implementation

cost

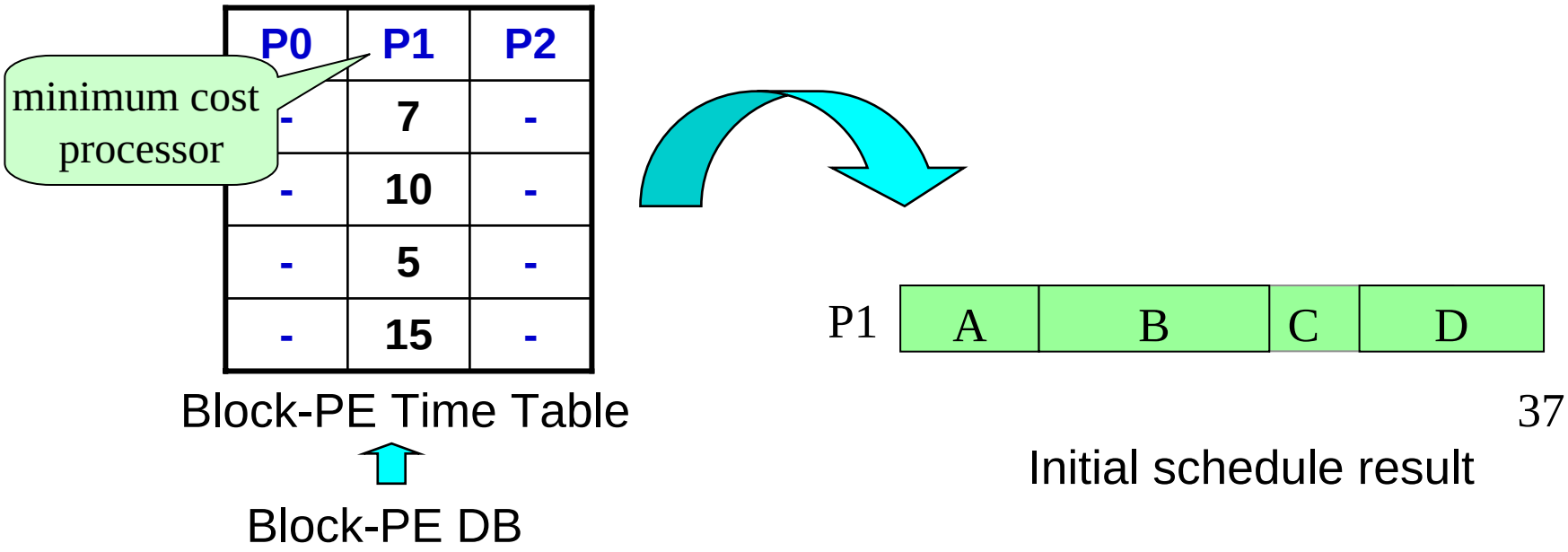
| | P0(HW) | | | P1(1) | P2(5) |
|---|---|---|---|---|---|
| | B0 | B1 | B2 | | |
| A | 3(4) | 2(6) | 1(10) | 7 | 2 |
| B | 4(5) | 2(8) | 1(10) | 10 | 3 |
| C | 2(3) | 1(5) | | 5 | 2 |
| D | 5(10) | 3(15) | | 15 | 5 |

execution time

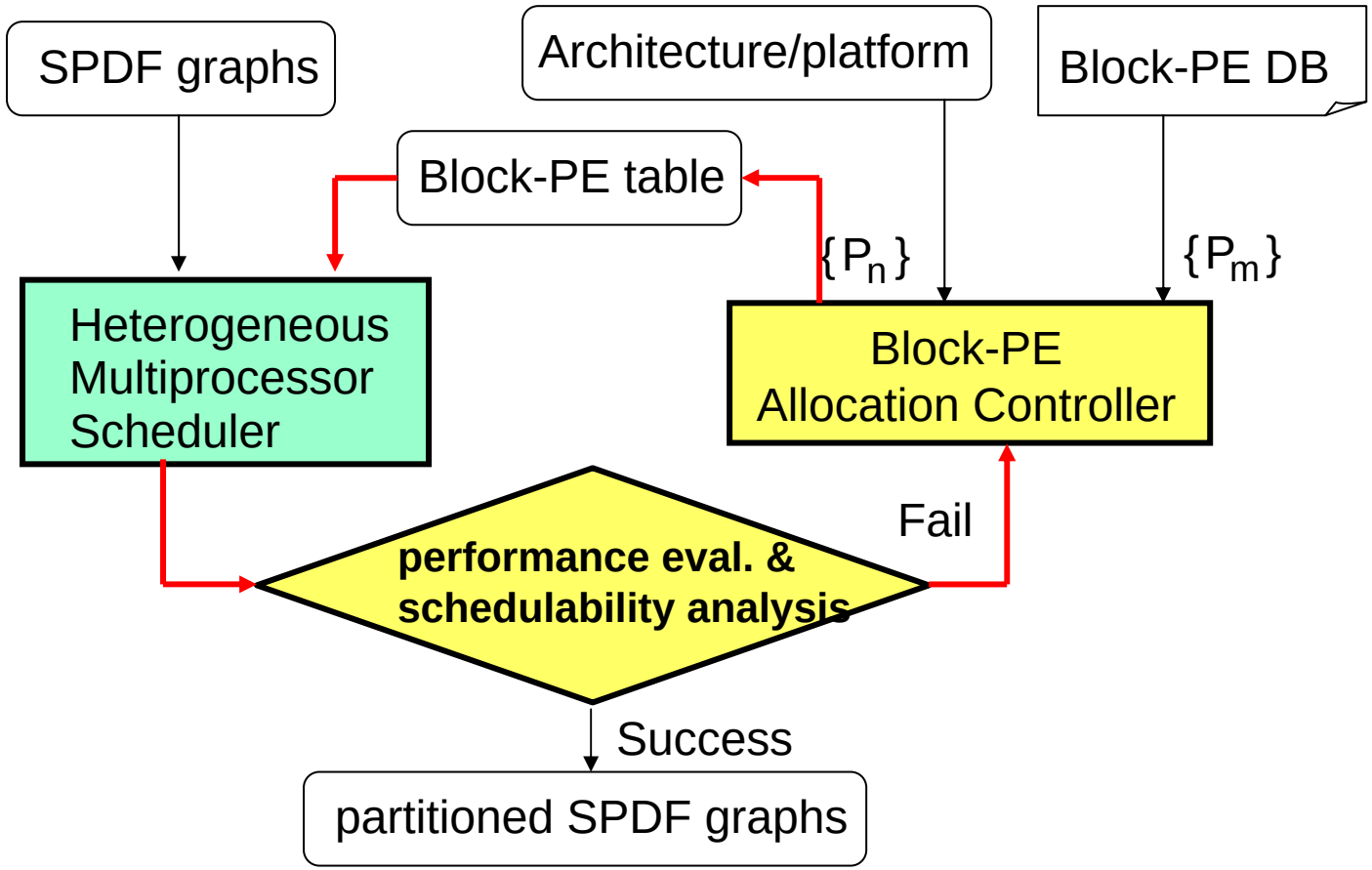
**Block-PE DB**

# Initial Partitioning

- **Terminologies**
  - lock & unlock
    - A task is called “locked” out of a PE if it cannot be scheduled on a processing elements, and “unlocked” otherwise.



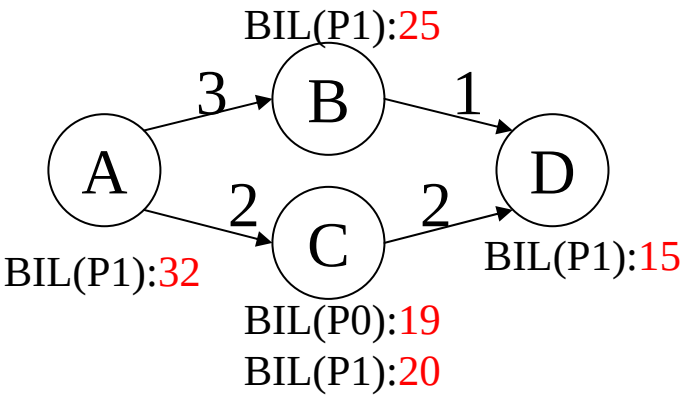
# Next …


SPDF graphs
Architecture/platform
Block-PE DB
Block-PE table
{Pn}
{Pm}
Heterogeneous Multiprocessor Scheduler
Block-PE Allocation Controller
Fail
performance eval. & schedulability analysis
Success
partitioned SPDF graphs

# Heterogeneous Multiprocessor Scheduling Algorithm

- **Any HMP Scheduler can be used.**
- **BIL Scheduler**
  - List scheduling
  - Static level
    - BIL(i,j) = ExTime(i,j) + max[min(BIL(d,j),min(BIL(d,k)+C(i,d)))]
    - i : node, j : processor, d : child node of i, C(i,d) : the amount of IPC overhead between i and d
  - Dynamic level
    - BIM(i,j) = BIL(i,j) + T(i,j)
    - T(i,j) : node i cannot be scheduled on j before T(i,j)

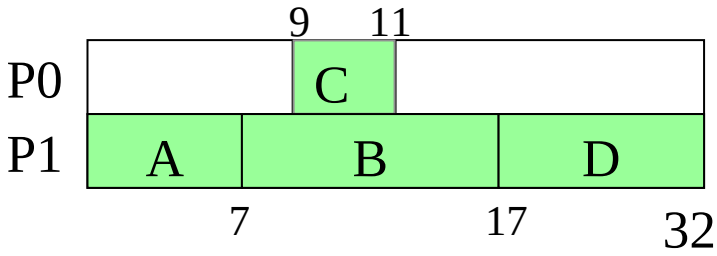
# BIL Computation



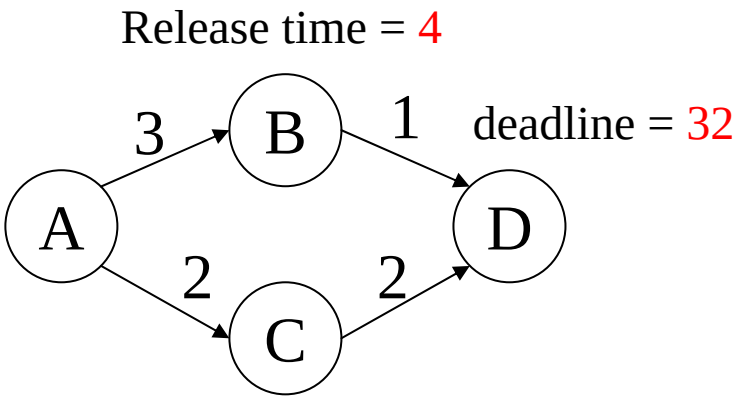

BIL(B,P1) = ExTime(B,P1) + BIL(D,P1)
= 10+15 = 25
BIL(C,P0) = ExTime(C,P0) + BIL(D,P1) + C(C,D)
= 2+15+2 = 19
BIL(C,P1) = ExTime(C,P1) + BIL(D,P1)
= 5+15 = 20
BIL(A,P1) = ExTime(A,P1) +
max[min{BIL(C,P0)+C(A,C), BIL(C,P1)},
BIL(B,P1)]
= 7 + max(min(21,20),25) = 7+25= 32

## Schedule

A is scheduled on P1
B is scheduled on P1
BIM(B,P1) = 25+7 = 32
BIM(C,P0) = 19+2+7=28,
BIM(C,P1)=20+7=27
C is scheduled on P0
BIM(C,P0) = 19+(7+2)=28
BIM(C,P1) = 20+17=37
D is scheduled on P1
BIM(D,P1) = 15+17=32

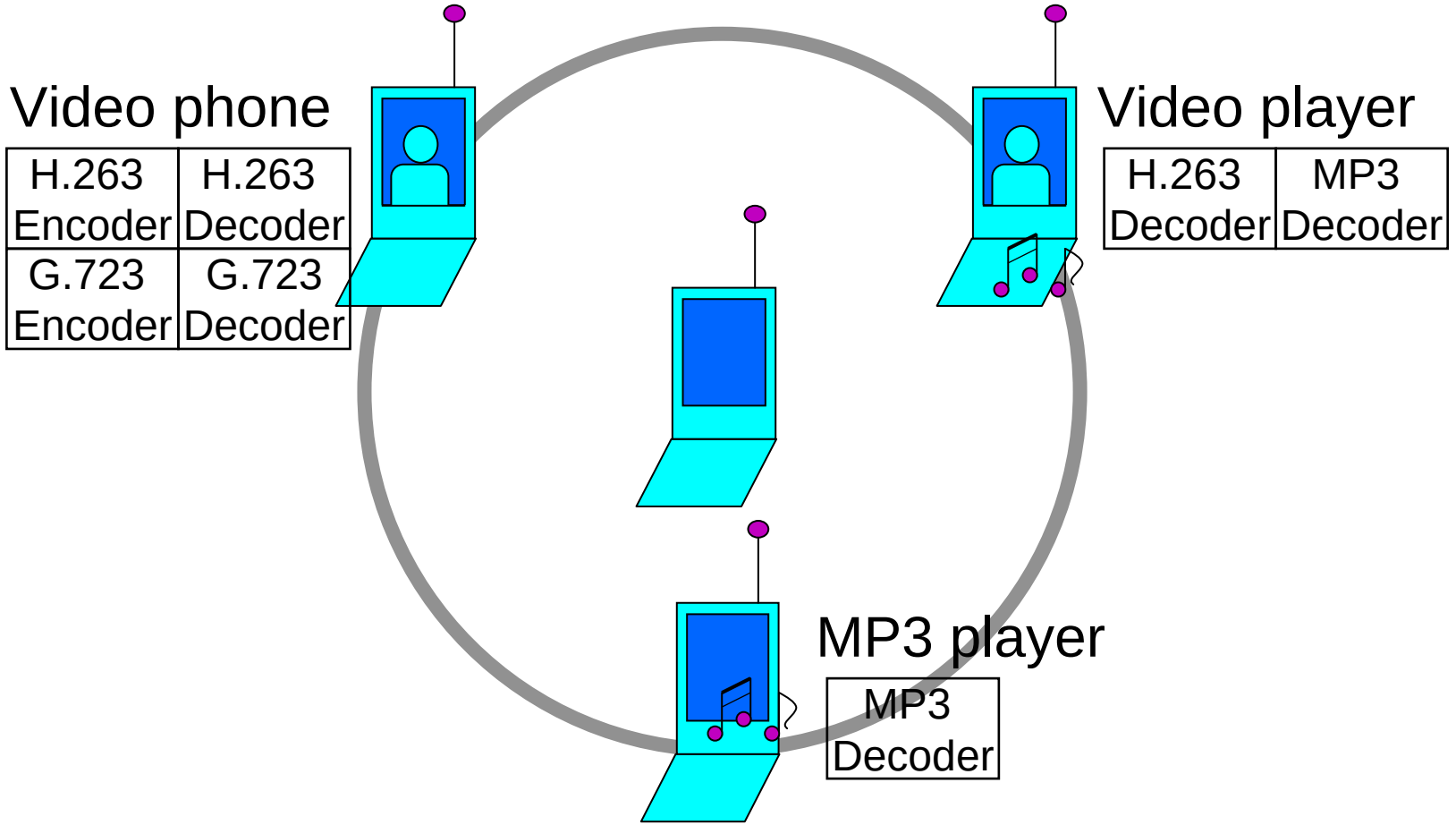


# Partitioning With Two-Sided Timing Constraints




- **Release time**
  - a node cannot begin until its release time
- **Deadline**
  - a node must be completed by its deadline
- **Revision of HMP scheduler**

# Extention: Multi-Mode Multi-Task Embedded System



# Multi-Mode Multi-Task System

Multi-Mode

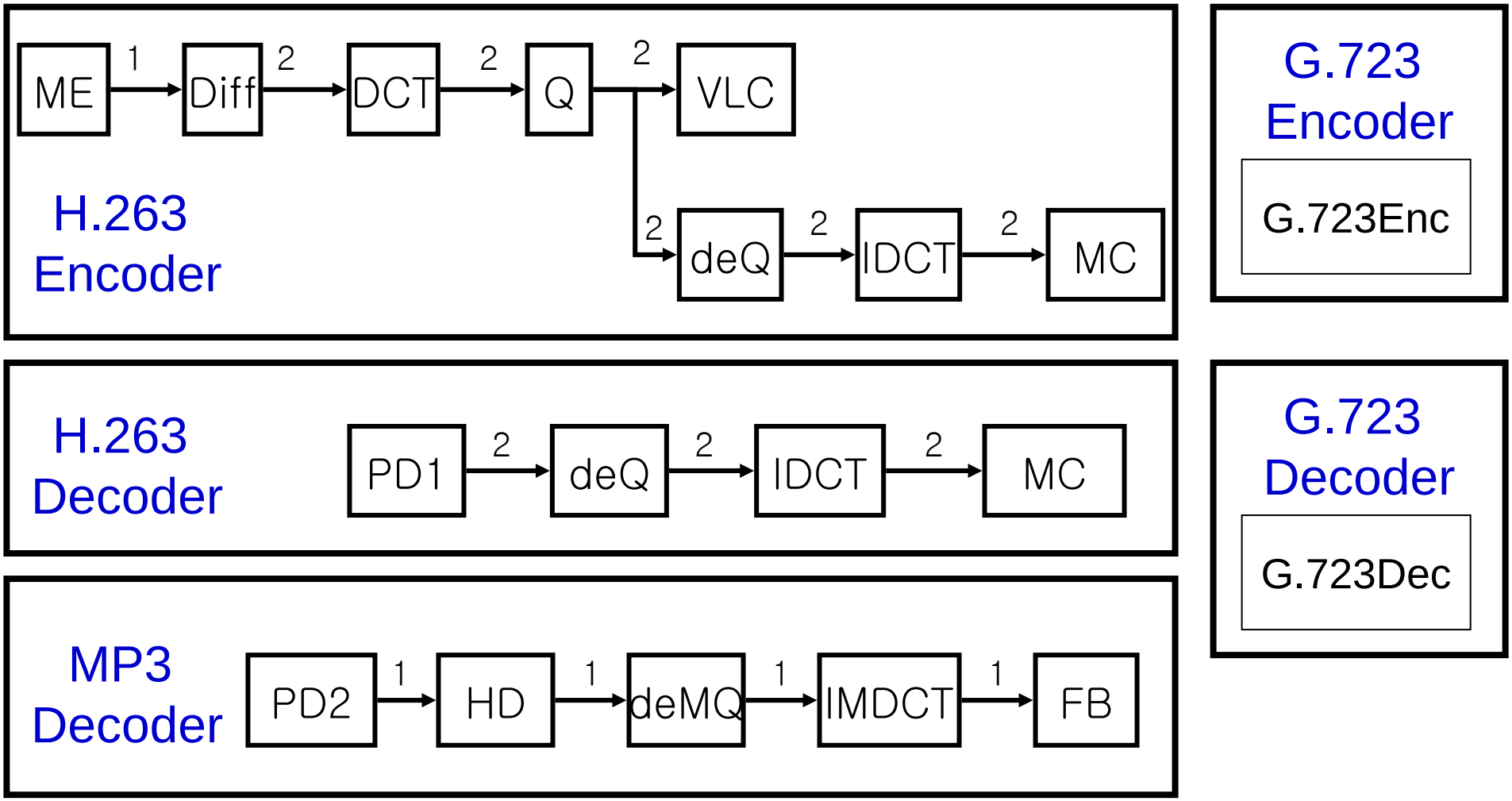
Multi-Task

Multiple modes by dynamically refiguring the system functionality

Each mode comprises a set of real-time tasks that should be scheduled within time constraints

All modes should satisfy the schedulability

# Task Specification



codesign environment

# Mode Specification

| Mode | Video phone | | | | Video player | | MP3 |
|---|---|---|---|---|---|---|---|
| Task | H.263 Encoder | H.263 Decoder | G.723 Encoder | G.723 Decoder | H.263 Decoder | MP3 Decoder | MP3 Decoder |
| Period | 100 | 100 | 25 | 25 | 50 | 40 | 40 |
| Deadline | 100 | 100 | 25 | 25 | 50 | 40 | 40 |

# Block-PE DB

| | HW: time (cost) | | P1:ARM (100) | P2:ARM2(900) |
|---|---|---|---|---|
| MC | $MC_{hw1}$:2.2(10) | $MC_{hw2}$:1(30) | 17 | 8.5 |
| ME | $ME_{hw}$ : 17 (100) | | 518 | 259 |
| DCT | $DCT_{hw}$ : 5.6 (20) | | 17 | 8.5 |
| IDCT | $IDCT_{hw}$ : 5.8 (20) | | 18 | 9 |
| PD1 | | | 4.4 | 2.2 |

...

# Related Work (1)

- **Multi-mode multi-task systems**
  - Y. Shin, D. Kim and K. Choi at DAC'00
  - Objective
    - How much we should reduce the execution time of each task
  - Example
    - The execution time of H.263 decoder should be reduced to 17.75 ms from 33.6 ms
    - That of H.263 encoder should be reduced to 68.65 ms from 596.8 ms
  - They ignored the function module sharing possibility between tasks.

# Related Work (2)

- **Multi-task systems**
  - No research for cosynthesis of multi-mode multi-task systems
  - Cosynthesis technique for multi-task system is applied to each mode separately
    - J. Hou and W. Wolf at CODES'96
  - No resource sharing between multiple modes

# Experimental Result

| Without considering multi-mode | | | | | With considering multi-mode | | | |
|---|---|---|---|---|---|---|---|---|
| Used PE's | | | Cost | Mode | Cost | Used PE's | | |
| $ME_{hw}$ | $IDCT_{hw}$ | $P1$ | 220 | Video phone | 220 | $P1$ | $IDCT_{hw}$ | $ME_{hw}$ |
| $MC_{hw1}$ | $FB_{hw}$ | $P1$ | 115 | Video player | 120 | $P1$ | $IDCT_{hw}$ | |
| | | $P1$ | 100 | MP3 player | 100 | $P1$ | | |
| $ME_{hw}$ | $IDCT_{hw}$ | $P1$ | 235 | Total | 220 | $P1$ | $IDCT_{hw}$ | $ME_{hw}$ |
| $MC_{hw1}$ | $FB_{hw}$ | | | | | | | |

# Characteristics of Our Approach

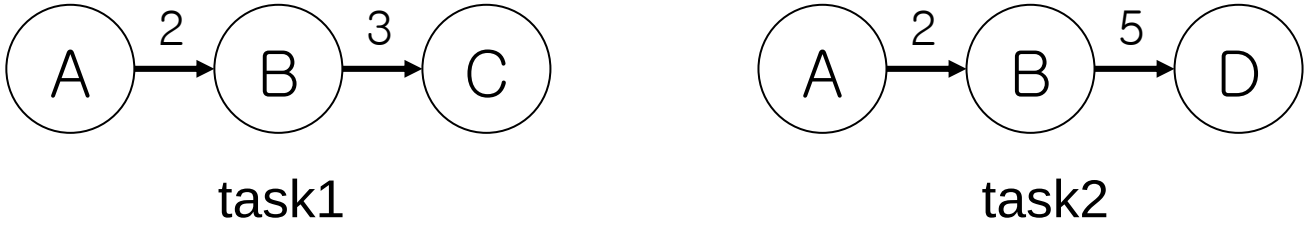
- **Scheduling unit is a function module**
  - Common modules can be shared between multiple tasks
- **Real-time scheduler is assumed**
  - Schedulability test based on utilization

# Example : Mode Specification

| Mode | Mode1 | | Mode2 |
|---|---|---|---|
| Task | task1 | task2 | task2 |
| Period | 40 | 60 | 30 |
| Deadline | 40 | 60 | 30 |

# Example : Task Specification & Module-PE Profile Table

## Task Graph




## Module-PE profile table

| | P0(HW): time (cost) | | P1 (10) | P2(60) |
|---|---|---|---|---|
| A | $A_{hw}$ : 1 (12) | | 7 | 2 |
| B | $B_{hw1}$: 2 (5) | $B_{hw2}$ : 1 (15) | 8 | 3 |
| C | $C_{hw1}$: 2 (10) | $C_{hw2}$ : 1 (20) | 10 | 5 |
| D | $D_{hw}$ : 4 (10) | | 16 | 5 |

# Initial Allocation

Heterogeneous Multiprocessor Scheduler

Block-PE Allocation Controller

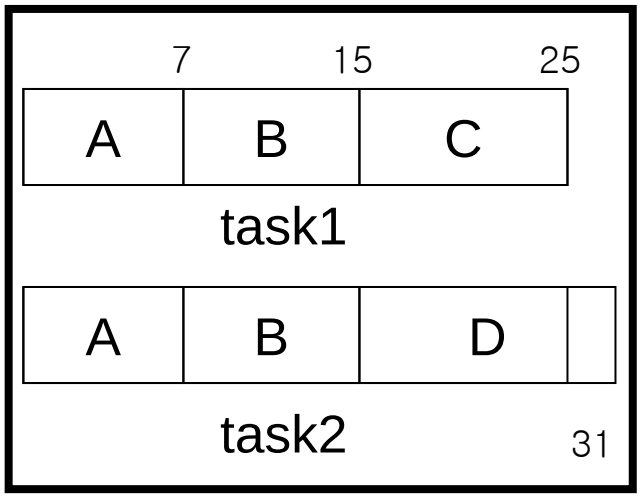
| | P0 | P1 | P2 |
|---|---|---|---|
| A | ∞ | 7 | ∞ |
| B | ∞ | 8 | ∞ |
| C | ∞ | 10 | ∞ |
| D | ∞ | 16 | ∞ |

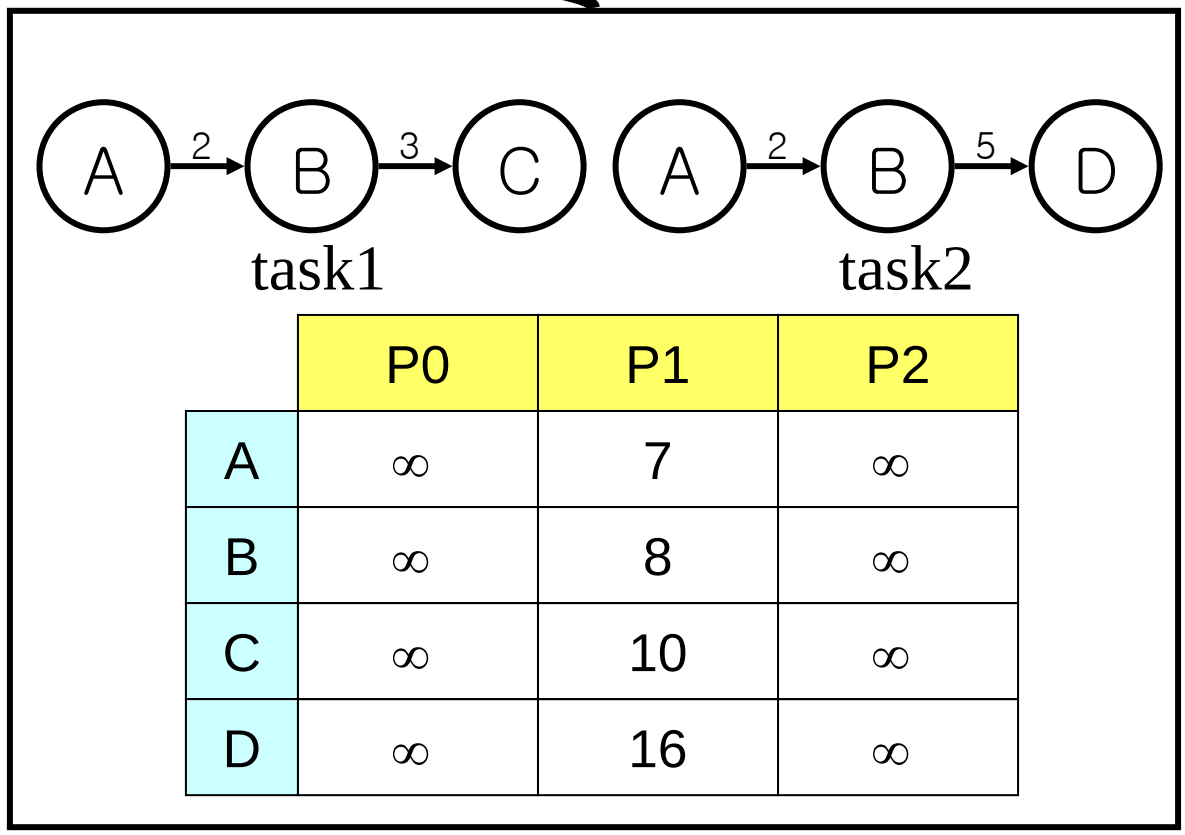
Allocate P1

Initially reduced Block-PE table

# Scheduling

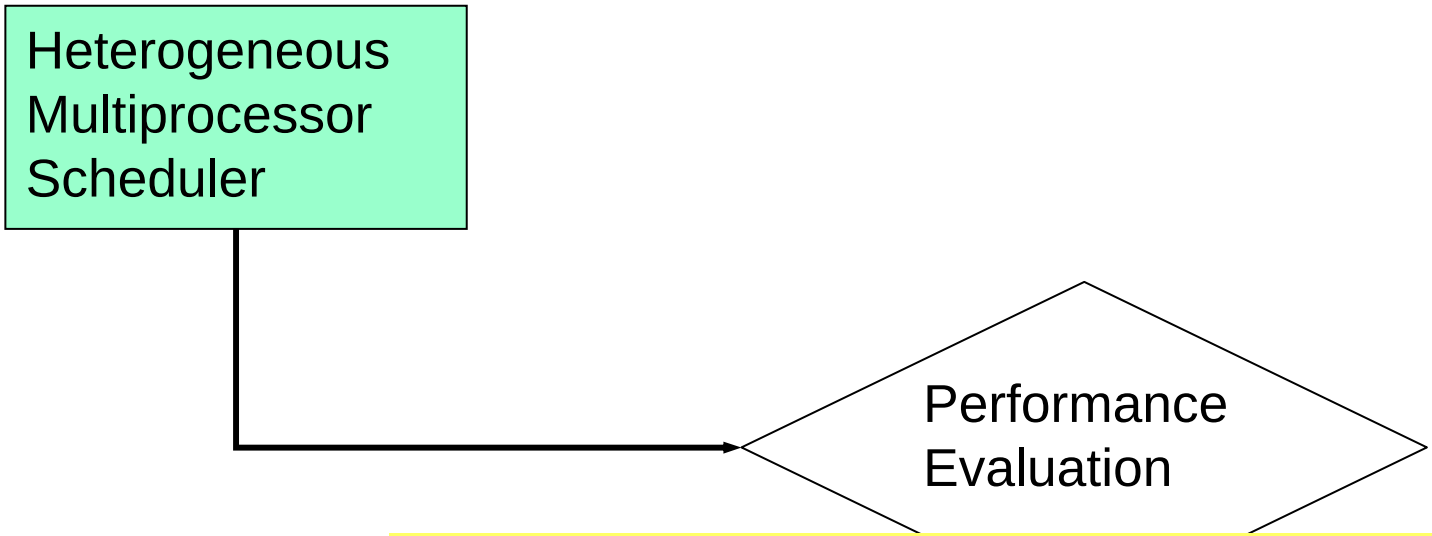
Heterogeneous Multiprocessor Scheduler



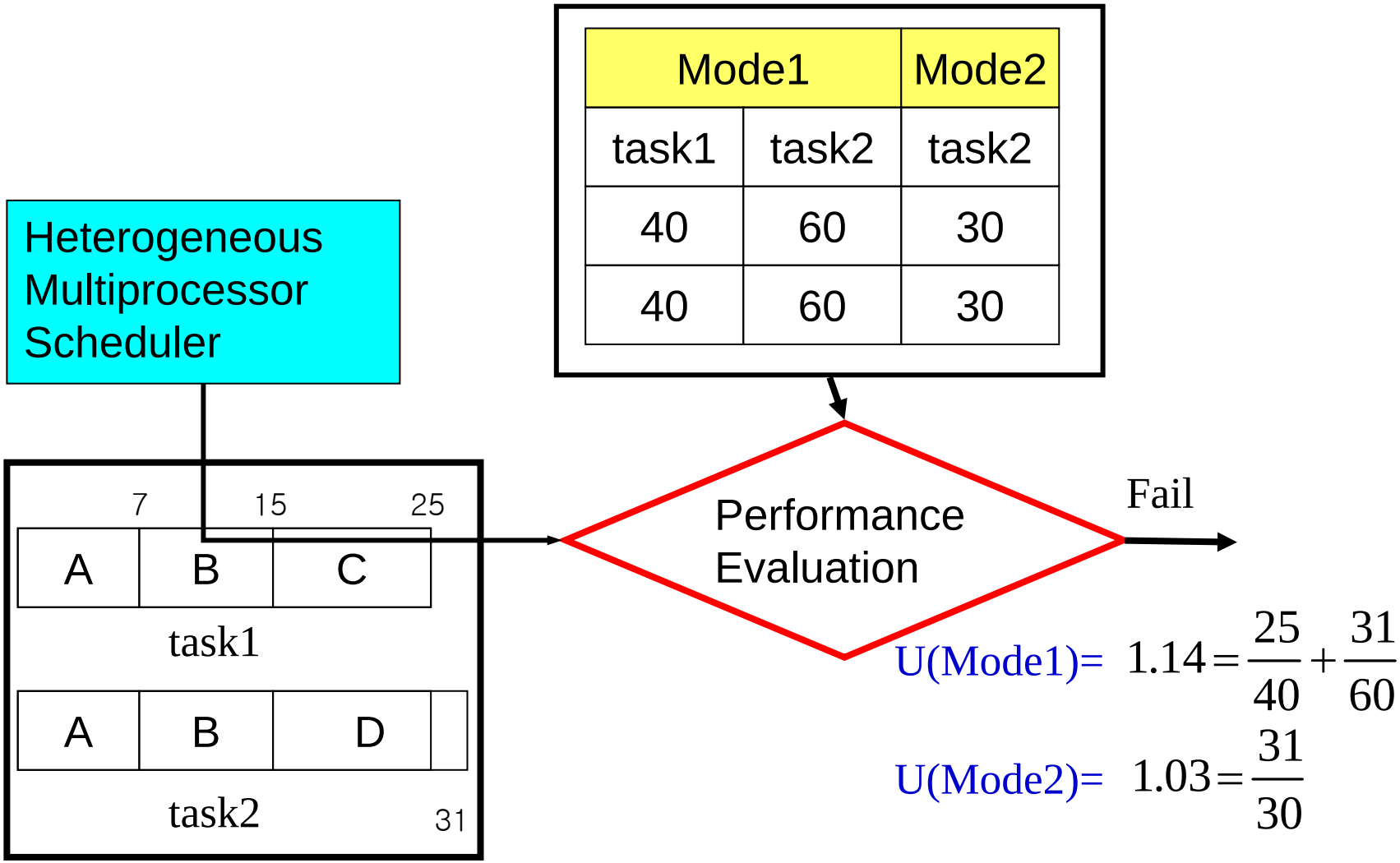

| | P0 | P1 | P2 |
|---|---|---|---|
| A | ∞ | 7 | ∞ |
| B | ∞ | 8 | ∞ |
| C | ∞ | 10 | ∞ |
| D | ∞ | 16 | ∞ |

# Utilization Test

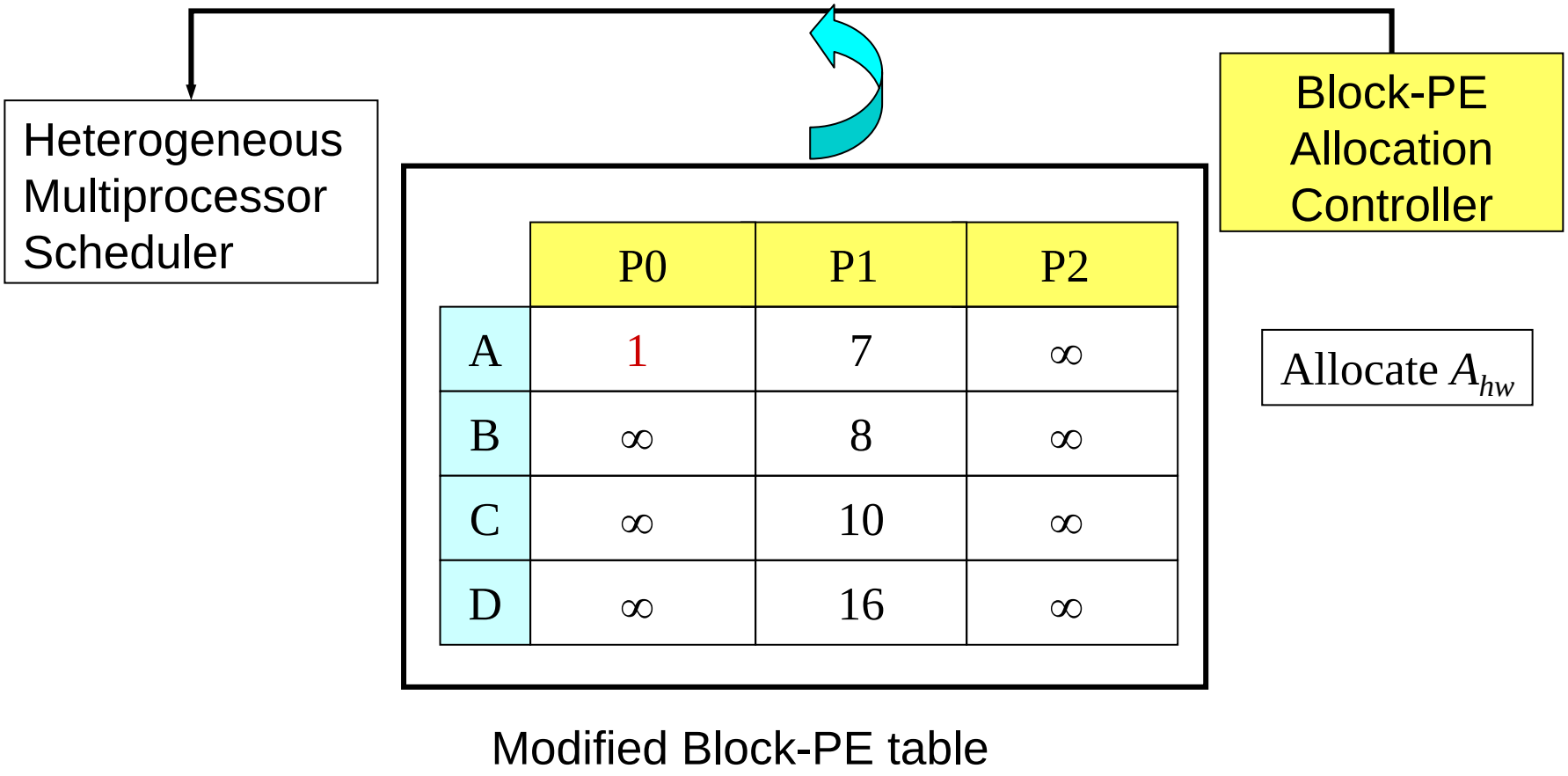



$$U(\text{Mode}) = \text{sum of } \frac{\text{Schedule length}}{\text{period}} \text{ for all tasks}$$

# Utilization Test



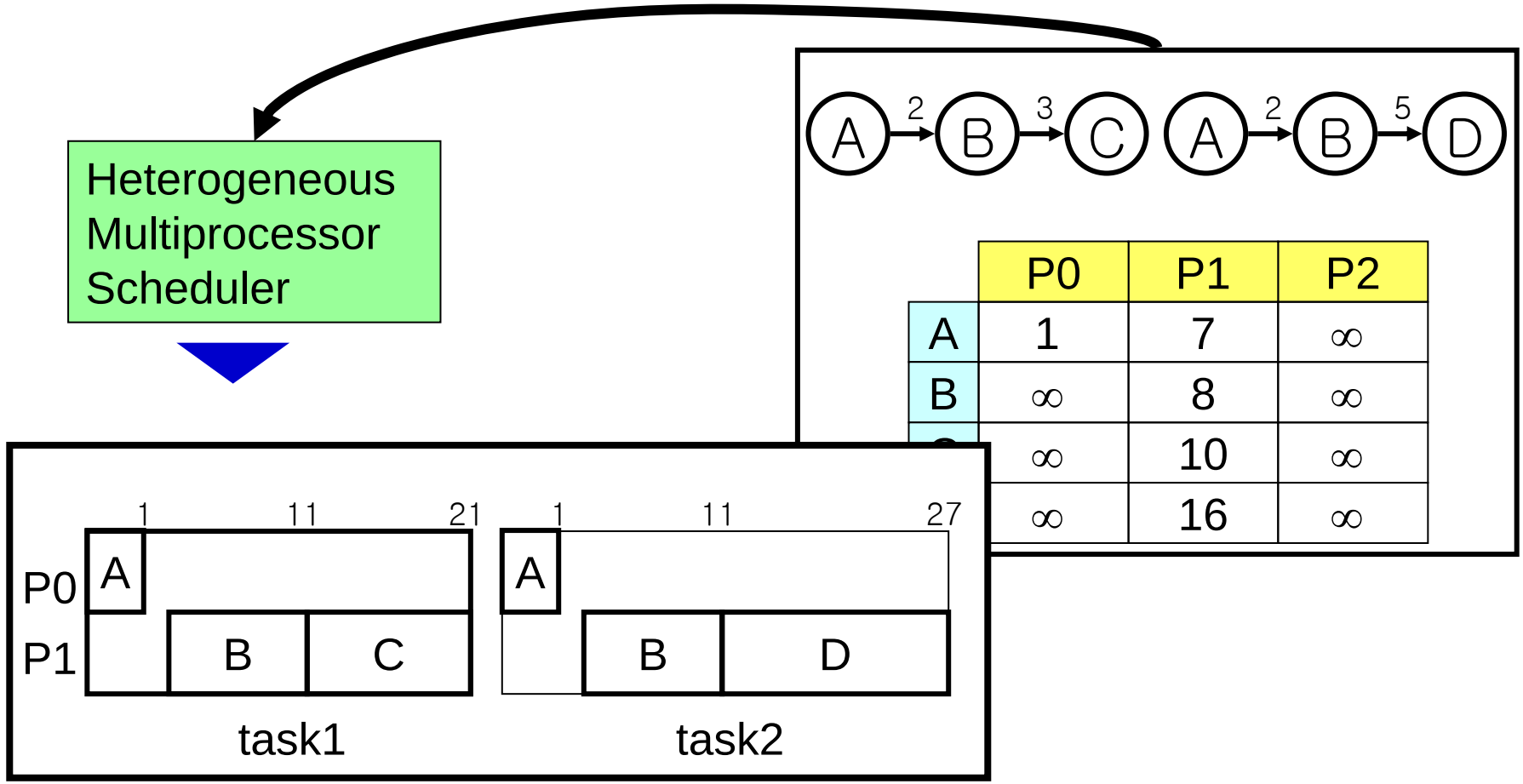
# Module-PE Allocation



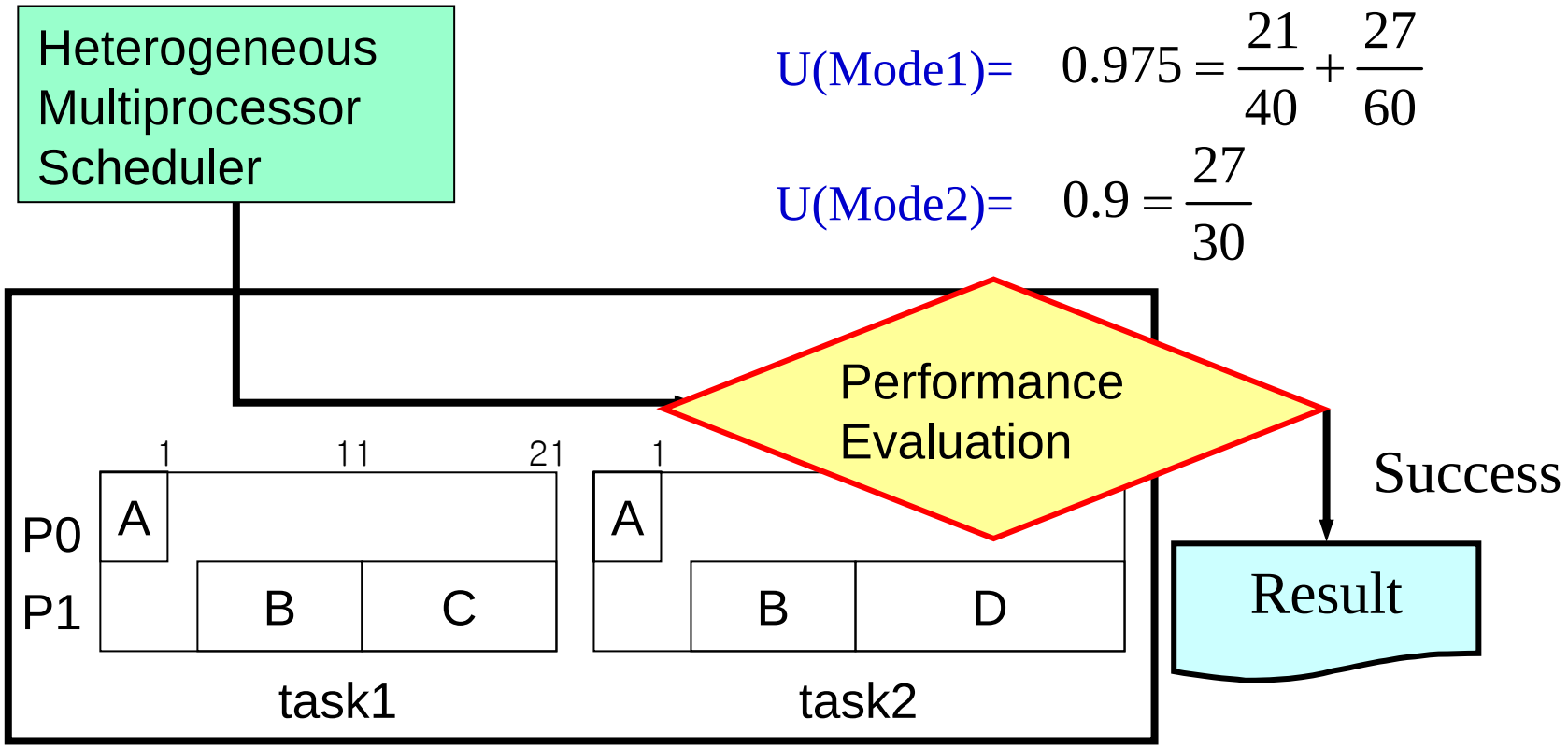

| | P0 | P1 | P2 |
|---|---|---|---|
| A | 1 | 7 | ∞ |
| B | ∞ | 8 | ∞ |
| C | ∞ | 10 | ∞ |
| D | ∞ | 16 | ∞ |

Modified Block-PE table

# Schedule


A
2
B
3
C
A
2
B
5
D
Heterogeneous Multiprocessor Scheduler
P0
P1
P2
A
1
7
∞
B
∞
8
∞
∞
10
∞
∞
16
∞
1
11
21
27
P0
P1
A
B
C
A
B
D
task1
task2

# Utilization Test



# Experimental Result (1)

| Without considering multi-mode | | | | | With considering multi-mode | | | |
|---|---|---|---|---|---|---|---|---|
| Used PE's | | | Cost | Mode | Cost | Used PE's | | |
| $ME_{hw}$ | $IDCT_{hw}$ | $P1$ | 220 | Video phone | 220 | $P1$ | $IDCT_{hw}$ | $ME_{hw}$ |
| $MC_{hw1}$ | $FB_{hw}$ | $P1$ | 115 | Video player | 120 | $P1$ | $IDCT_{hw}$ | |
| | | $P1$ | 100 | MP3 player | 100 | $P1$ | | |
| $ME_{hw}$ | $IDCT_{hw}$ | $P1$ | 235 | Total | 220 | $P1$ | $IDCT_{hw}$ | $ME_{hw}$ |
| $MC_{hw1}$ | $FB_{hw}$ | | | | | | | |

# Experimental Result (2)

Hou's Example

| Mode | Mode1 | | Mode2 | | Mode3 | |
|---|---|---|---|---|---|---|
| Task | task1 | task2 | task1 | task3 | task3 | task4 |
| Original period | 240 | 240 | 240 | 200 | 200 | 380 |
| Relaxed period | 240 | 240 | 2000 | 2000 | 200 | 380 |

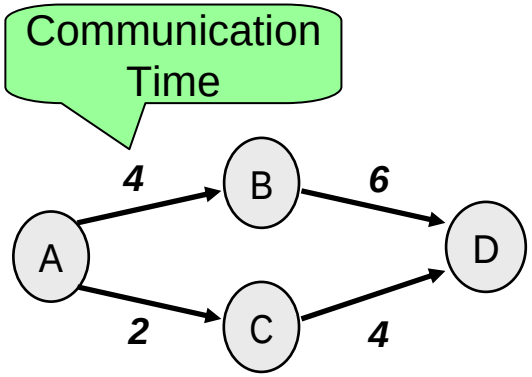
Without considering multi-mode

| Cost | Mode | Cost |
|---|---|---|
| 170 | Original period | 170 |
| 190 | Relaxed period | 170 |

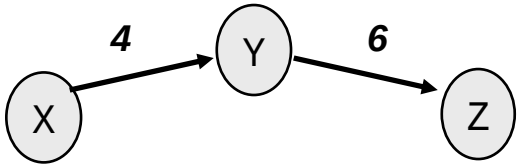
With considering multi-mode

# Extension is still needed

- **Motivational example**




$Task_1$




$Task_2$

| | $Task_1$ | $Task_2$ |
|---|---|---|
| Period | 60 | 80 |
| Deadline | 60 | 80 |

| | $P_1(20)$ | $P_2(100)$ |
|---|---|---|
| A | 21 | 16 |
| B | 10 | 6 |
| C | 25 | 21 |
| D | 26 | 17 |

Execution time

Cost

| | $P_1(20)$ | $P_2(100)$ |
|---|---|---|
| X | 14 | 8 |
| Y | 15 | 10 |
| Z | 20 | 15 |

# HMP Results

| | $P_1$ | $P_2$ |
|---|---|---|
| A | 10 | 7 |
| B | 13 | 10 |
| C | 4 | 3 |
| D | 3 | 15 |

Original

| | $P_1$ | $P_2$ |
|---|---|---|
| A | 44 | 35 |
| B | 58 | 42 |
| C | 34 | 36 |
| D | 68 | 39 |

BIM*

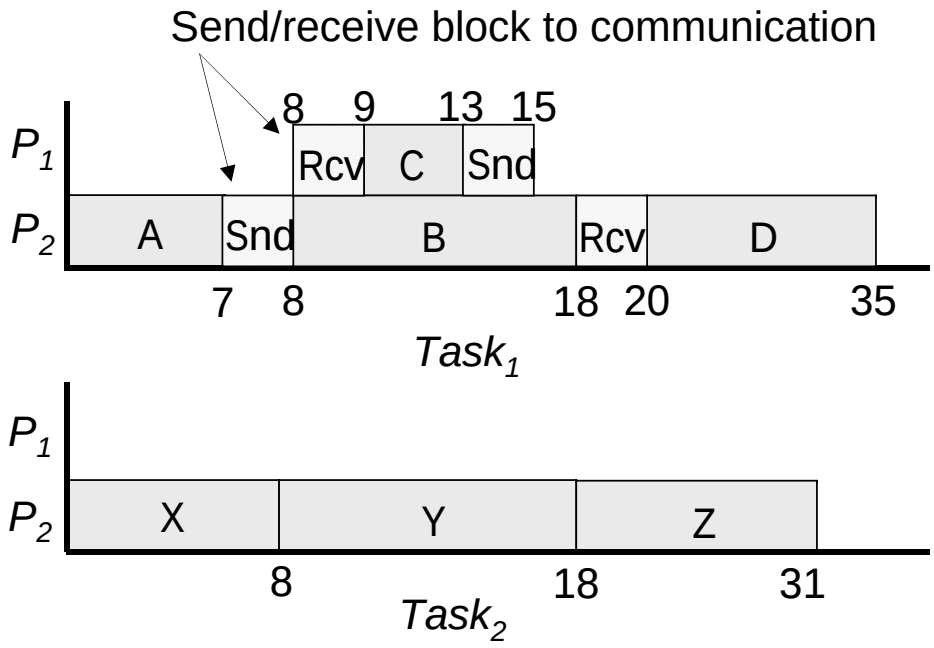
| | $P_1$ | $P_2$ |
|---|---|---|
| X | 14 | 8 |
| Y | 15 | 10 |
| Z | 16 | 13 |

Original

| | $P_1$ | $P_2$ |
|---|---|---|
| X | 80 | 67 |
| Y | 82 | 68 |
| Z | 80 | 69 |

BIM*

The partition result from HMS algorithm



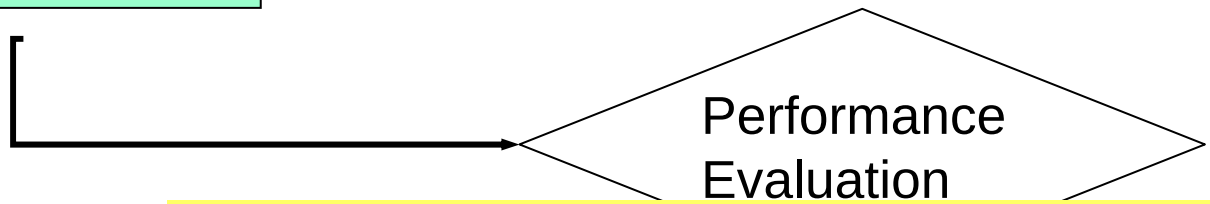

Schedule diagram

- **HMP Scheduler: schedule the task graph to have the minimum execution time ➡ all tasks tend to use faster processor**

# Schedulability Analysis

Heterogeneous Multiprocessor Scheduler

| | $Task_1$ | $Task_2$ |
|---|---|---|
| Period | 60 | 80 |
| Deadline | 60 | 80 |




$$U(Mode) = \text{sum of } \frac{\text{Schedule length}}{\text{period}} \text{ for all tasks}$$

- **Schedulability test is failed because the utilization of processor is larger than 1 (35/40 + 31/60 = 1.4) !**

codesign environment

# Motivation

- **Limits of previous framework**
  - Assume that each task monopolizes all resources.
  - Schedulability analysis based on “utilization factor”

➡ Not suitable for general multi-tasking applications

- **Extension of the framework**
  - Allow multiple tasks to run concurrently
  - Schedule-based schedulability analysis

codesign environment

# Related works

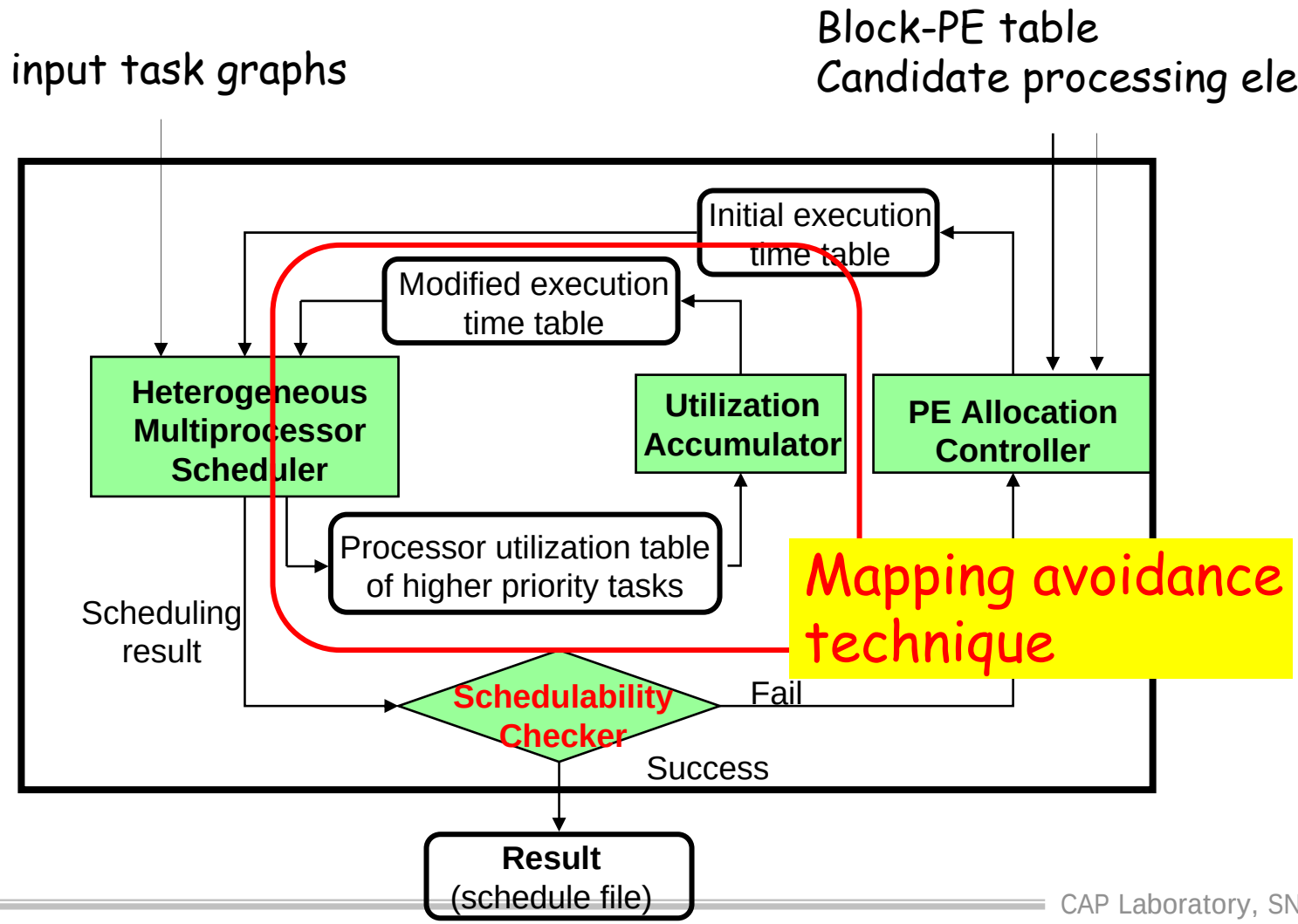
- **Partitioning technique**
  - Genetic Algorithm
    - EMOGAC [8] (extension version of MOGAC [7])
      - R. P. Dic and N. K. Jha
      - Static schedule until their hyper-period
  - Heuristic Iterative Method
    - COSYN [9]
      - B. P. Dave, G. Lakshminarayana and N. K. Jha
  - Branch-and-Bound
    - P. Pop et al [6]
      - Integrated the schedulability analysis into the partitioning algorithm
- **Disadvantages**
  - Tightly coupled with the application size such as the number of tasks, size of task, and the hyper-period
  - Method itself is not applicable to handle aperiodic tasks

codesign environment

# Related works

- **Schedulability analysis**
  - Worst-case response time bound
    - Tindell at al. [10]
      - Considered the task with static offset
    - J.C. Palencia et al. [11] (extension of Tindell's approach)
      - Allows the dynamic offset.
    - Disadvantage
      - Over-estimation
  - Simulation-based approach
    - Slomka's [12]
      - Calculates all possible preemption delay
    - Disadvantage
      - Does not allow multiple predecessors of a task for easy analysis of the worst delay.
  - Current implementation:Timed Multitasking analysis approach

# Proposed Extensions



# Mapping Avoidance Technique

- **Key Idea**
  - Adjust the estimated execution of each block on a processor considering the utilization factor of the processor by pre-assigned tasks.

- **Procedure**

(1) Sort the input tasks in order of its priority from highest to the lowest
  - The task with early deadline has higher priority

(2) Select the top-priority task among the remaining tasks

(3) Update the module-PE timetable base on the processor utilization by 'Utilization Accumulator'

(4) Apply HMS algorithm to get the shortest schedule length

(5) Repeats (2) to (4) until all tasks are partitioned

# Mapping avoidance scheduling

- **Terms**
  - k: the index of a processor element
  - $n_{ij}$: *j*-th node(function block) of the *i*-th task graph
  - $E^{new}(n_{ij}, P_k)$: estimated execution time of node $n_{ij}$ on processor $P_k$
  - $E(n_{ij}, P_k)$: measured execution time of node $n_{ij}$ taken from the performance database
  - $proc(n_{ij})$: the processor index of node $n_{ij}$ after mapping is performed
  - $hp(\tau_i)$: the set of task graphs with a higher priority than task $\tau_i$
  - $T_i$: period of the task $\tau_i$
  - $Comm(n_{ij})$: the sum of communication time of node $n_{ij}$
  - Util ( $\tau_i$, $P_k$): the accumulated processor utilization

  - $E^{new}(n_{ij}, P_k)$, $Comm(n_{ij})$ can be changed while the partitioning algorithm is running.

# Mapping avoidance scheduling

- **The accumulated processor utilization**

$$Util(\tau_i, P_k) = \sum_{i \in hp(\tau_i)} \left( \frac{1}{T_i} \cdot \sum_{proc(n_{ij}) \wedge P_k} (E(n_{ij}, P_k) + Comm(n_{ij})) \right)$$

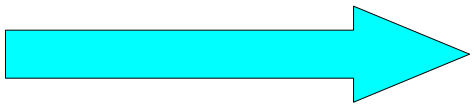
- **The change time of each node**

$$E^{new}(n_{ij\ j}, P_k) = \frac{E(n_{ij}, P_k)}{1 - Util(\tau_i, P_k)}$$

| | $P_1$ | $P_2$ |
|---|---|---|
| X | 14 | 8 |
| Y | 15 | 10 |
| Z | 16 | 13 |

Original

$$Util(Task_1, P_1) = 0.18$$
$$Util(Task_1, P_2) = 0.88$$



| | $P_1$ | $P_2$ |
|---|---|---|
| X | 17 | 66 |
| Y | 18 | 83 |
| Z | 19 | 108 |

The changed time table

# Proposed framework behavior

| | $P_1$ | $P_2$ |
|---|---|---|
| A | 10 | 7 |
| B | 13 | 10 |
| C | 4 | 3 |
| D | 3 | 15 |

Original

| | $P_1$ | $P_2$ |
|---|---|---|
| A | 44 | 35 |
| B | 58 | 42 |
| C | 34 | 36 |
| D | 68 | 39 |

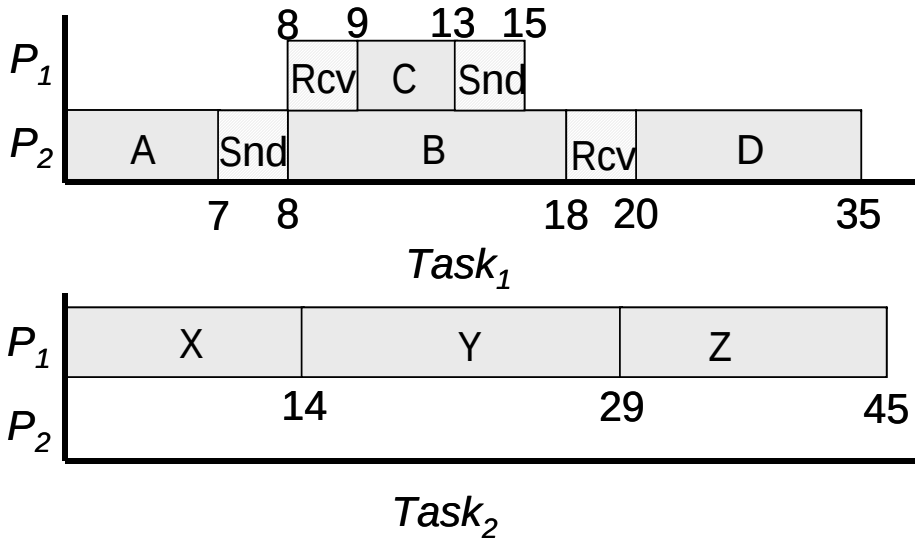
BIM*

| | $P_1$ | $P_2$ |
|---|---|---|
| X | 14 | 8 |
| Y | 15 | 10 |
| Z | 16 | 13 |

Original

| | $P_1$ | $P_2$ |
|---|---|---|
| X | 87 | 129 |
| Y | 79 | 123 |
| Z | 78 | 132 |

BIM*




- **All blocks in $Task_2$ are mapped onto $P_1$.**
- **This result can be scheduled by TM schedulability analysis**

# Schedulability analysis

- **Timed Multitasking**
  - Assumptions
    - Periodic task
    - Their priorities are statically and uniquely assigned at compile time
    - The release time of each task is static and known beforehand
    - Each node(function block) of a task is mapped to a single processing element
- **Schedule**

(1) All tasks are sorted by their priorities

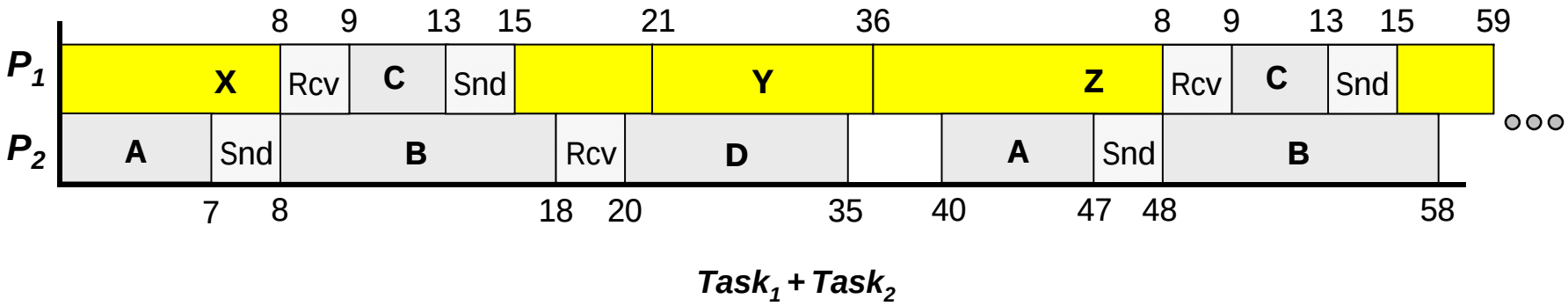
(2) Select the top-priority task first and iteratively scheduled by the repetition number within hyper period.

(3) Select the next highest priority task and schedule it using the available time slots maintaining the mapping decision.

(4) Repeats (3) until there is no task to be scheduled

# Schedulability analysis

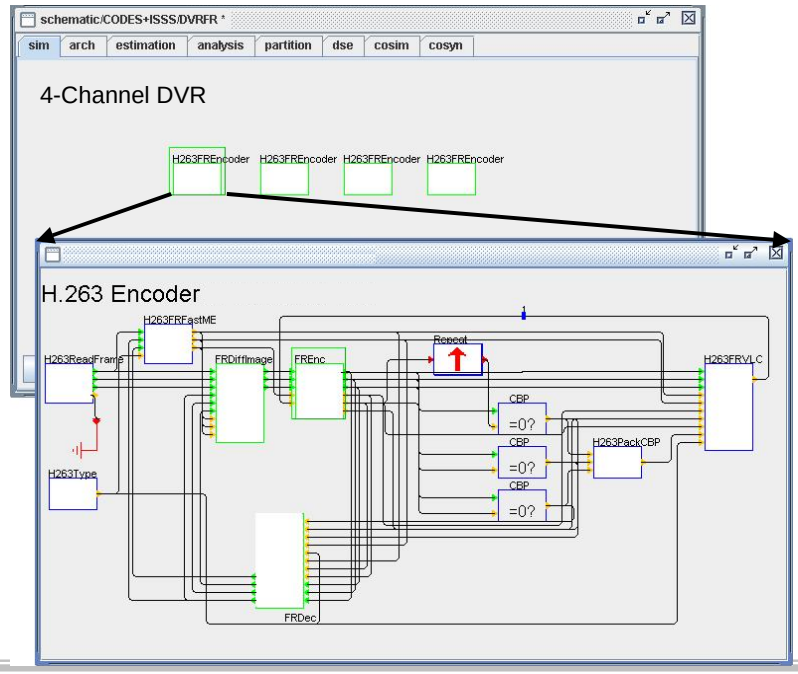
- **The diagram of TM scheduling**




***Task*$_1$ + *Task*$_2$**

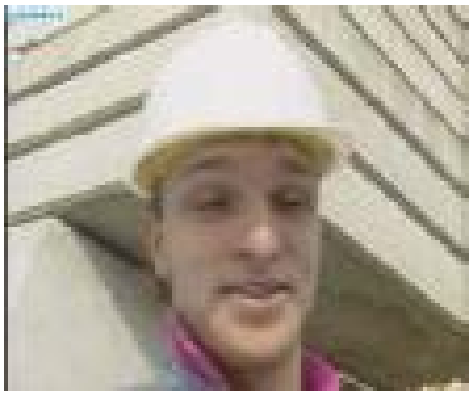
# Experiments

- **(1) 4-Channel DVR (Digital Video Recorder)**
  - Comprised of four H.263 Encoders
  - They are all the same timing constraints which is represented the number of bus cycles.
  - Specified on PeaCE (Ptolemy extension as Codesign Environment)



# Experiments

- **Performance estimation for each processor**
  - We used 133Mhz ARM720T, 266Mhz ARM920T, ARM926ej-s, and 399Mhz ARM1020T with L1 cache only
  - Bus cycle is 133Mhz
  - ‘armcc’ compiler with –02 option in ADS 1.2 on Xeon 2.8Ghz dual CPU for processor simulator
  - Input file: FOREMAN.QCIF



| Resource | | Schedule Length (Bus Cycles) |
|---|---|---|
| Core Type | Cost | |
| ARM720T | 100 | 45105050 |
| ARM926ej-s | 200 | 20368071 |
| ARM920T | 200 | 19098007 |
| ARM1020T | 400 | 10480947 |
| FPGA (MC, ME) | 4000 | - |
| FPGA(IDCT, DCT, VLC) | 2000 | - |

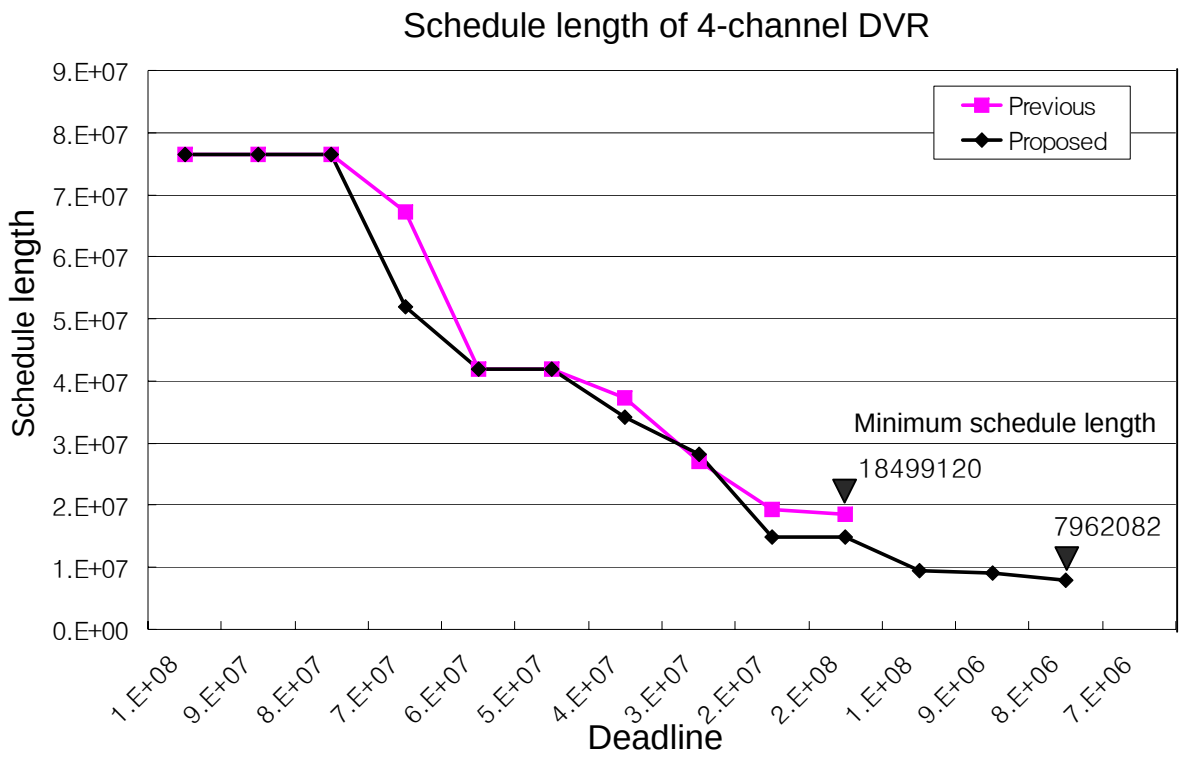
codesign environment

# Partitioning Results

| Previous | | Deadline (frame/sec) | Proposed | |
|---|---|---|---|---|
| Partition result | Cost | | Cost | Partition result |
| A9(1)* | 200 | 100,000,000 (1.3) ~ 80,000,000 (1.6) | 200 | A9(1) |
| A9(2) | 400 | 70,000,000 (1.9) | 400 | A9(2) |
| A10(1) | 400 | 60,000,000 (2.2) | 400 | A10(1) |
| A10(1) | 400 | 50,000,000 (2.7) | 400 | A10(1) |
| A10(2) | 800 | 40,000,000 (3.3) | 800 | A9(2), A9e(2) |
| A10(2), ME | 4800 | 30,000,000 (4.4) | 1000 | A9(3), A9e(2) |
| A10(3), ME, DCT, VLC | 9200 | 20,000,000 (6.7) | 1800 | A10(4), A7(2) |
| A10(3), ME, DCT, IDCT, VLC | 11200 | 19,000,000 (7.0) | 1800 | A10(4), A7(2) |
| N.A. | - | 10,000,000 (13.3) | 12800 | A10(7), MC, DCT, IDCT |
| N.A. | - | 9,000,000 (14.8) | 14800 | A10(7), MC, ME, DCT, IDCT |
| N.A. | - | 8,000,000 (16.7) | 16000 | A10(5), MC, ME, DCT, IDCT, VLC |
| N.A. | - | 7,000,000 (19.0) | - | N.A. |

# Experiments

- **Minimum schedule length**
  - Previous approach framework: 18499120 cycle (7.4 frame/sec)
  - Proposed approach framework: 7962082 cycle (16.9 frame/sec)
  - Performance improvement: 2.3 times



# Experiments

- **(2) Hou & Wolf's example**
  - Proposed approach runs much faster than EMOGAC with some performance penalty.
  - Advantages
    - It is too slow to run the heavy real example since EMOGAC is GA approach. Therefore it is not applicable to fast design framework.
    - Generates too long and complicate executable code.

| Example | EMOGAC | | COSYN | | Previous | | Proposed | |
|---|---|---|---|---|---|---|---|---|
| | Price | CPU** Time(s) | Price | CPU Time(s) | Price | CPU Time(s) | Price | CPU Time(s) |
| H&W 1&2 (u) | 140 | 600 | 170 | - | 170 | 0.1 | 150 | 0.13 |
| H&W 1&3 (u) | 170 | 4440 | 170 | - | 170 | 0.1 | 170 | 0.13 |
| H&W 3&4 (u) | 140 | 600 | N.A. | - | 170 | 0.1 | 170 | 0.13 |

** In case of EMOGAC, each example was run on a 1.4GHz AMD Athlon Thunderbird CPU and the CPU time is given in [8] of paper reference.

# Key Benefits of Our Approach

## Extensibility

Design constraints → Performance evaluation

Multiple design objectives → Performance evaluation

More PE's → Module-PE allocation

## Adaptability

Mapping and scheduling algorithm

Schedulability test for a given real-time scheduler → performance evaluation

# Extension of Platform Selection

- **Slight modification of module-PE allocation controller**
  - (S1) Select the cheapest platform first
  - (S2) Allocate more hardware IPs if necessary to satisfy the timing constraints through iteration
  - (S3) Compute the resultant system cost
  - (S4) Choose the next platform with cheaper cost and go to (S2) if exists
  - (S5) Select the cheapest solution

codesign environment

# Summary

- **HW/SW cosynthesis framework**
  - multi-mode and multi-task embedded system with real-time constraints
  - Selection of PE, mapping and scheduling of modules and schedulability test
  - Task sharing between modes and HW sharing between tasks
- **Key benefits of the proposed framework**
  - Extensibility
  - Adaptability
- **More extension is needed**
  - Data parallelism
  - Multi-objectives

codesign environment

# Contents

- **Block Performance Estimation**
  - Block Execution Time Estimation
- **Power Estimation**
- **HW/SW Cosynthesis**
- **Communication Architecture Exploration**

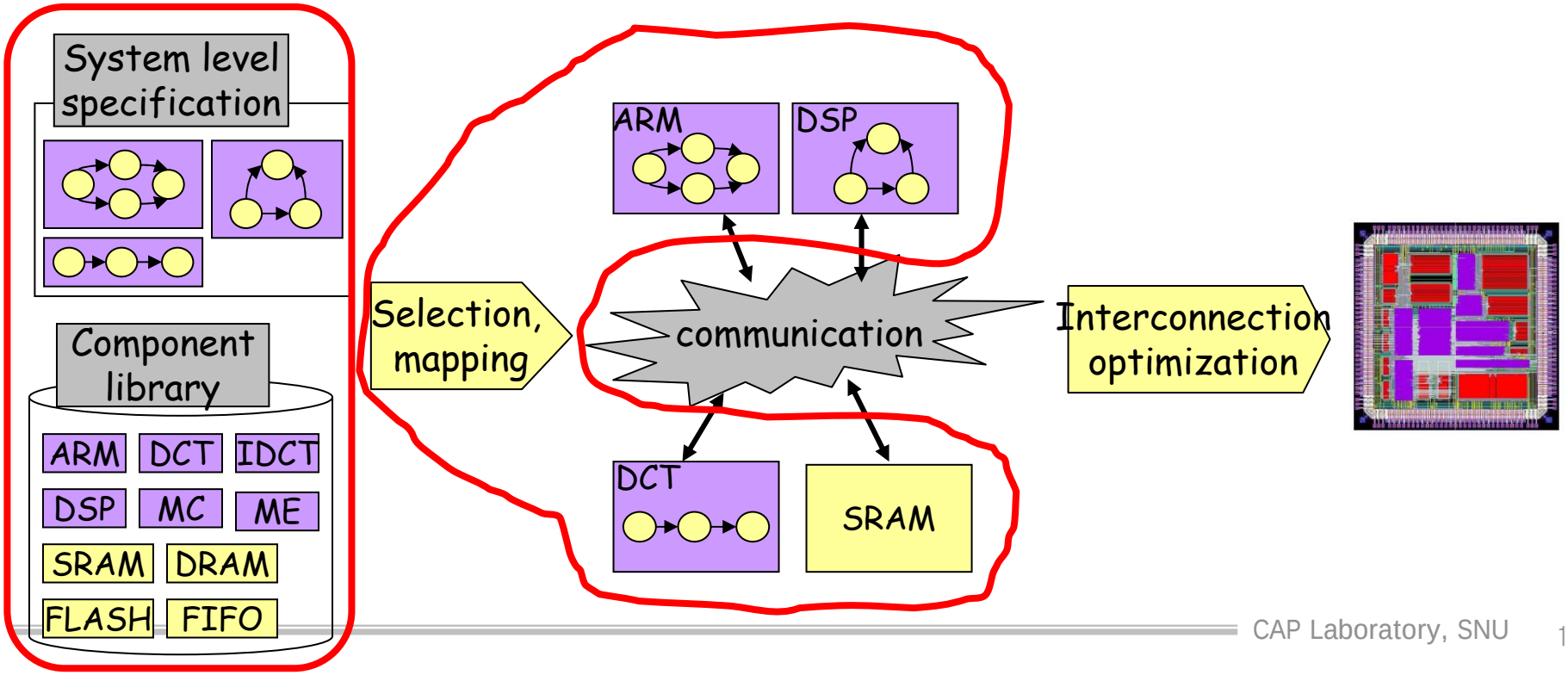
codesign environment

# Introduction

- **The communication architecture design is one of the most critical step in system level design**
- **Separation of computation and communication**
  - Allows the independent exploration communication architecture
- **Large design space for communication architectures**
  - Number of buses, bus topology
  - Mapping of processing elements, priority assignment
  - Significant performance improvement by just communication architecture optimization
    - Up to 78% in our experiments !!
- **Simulation based approach is popular, but it is too slow to explore the large design space**
  - The need for fast and efficient exploration methodology

# Problem definition

- **Find the performance-optimized bus-based communication architecture with the given**
  - System level specification of functional behavior
  - Component selection and mapping of functional behavior to the selected components

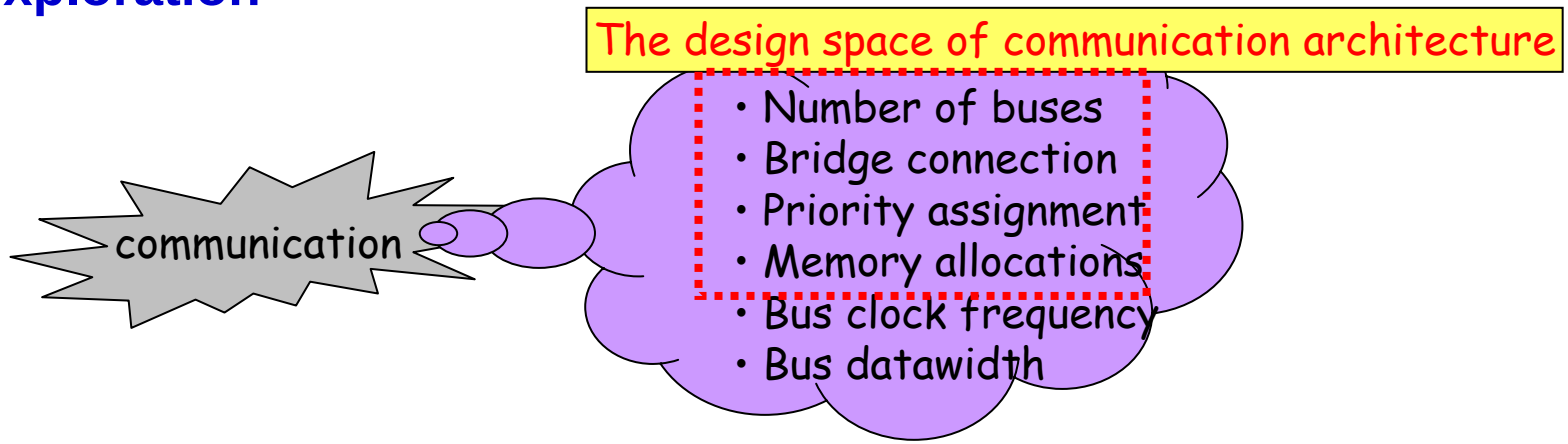


# Comm. DSE Approaches

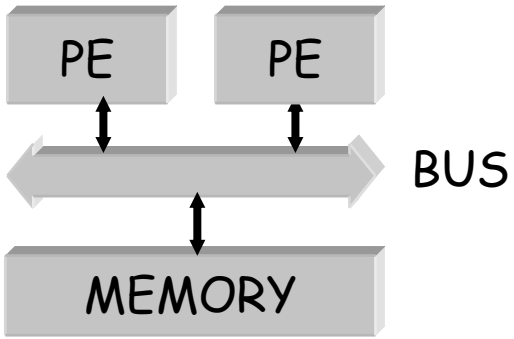
- **Static estimation based approach**
  - Pros : Time-efficient
  - Cons : Inaccurate, can be applied to limited architecture
- **Simulation based approach**
  - many commercial tools
  - Pros : Accurate
  - Cons : Too time consuming
- **Hybrid Approach**
  - K. Lahiri and Sujit Dey in UCSD
  - Pros : Accurate and time-efficient
  - Cons : Consider only the shared memory access

# Our scope

- **Many design axes exist for communication architecture exploration**



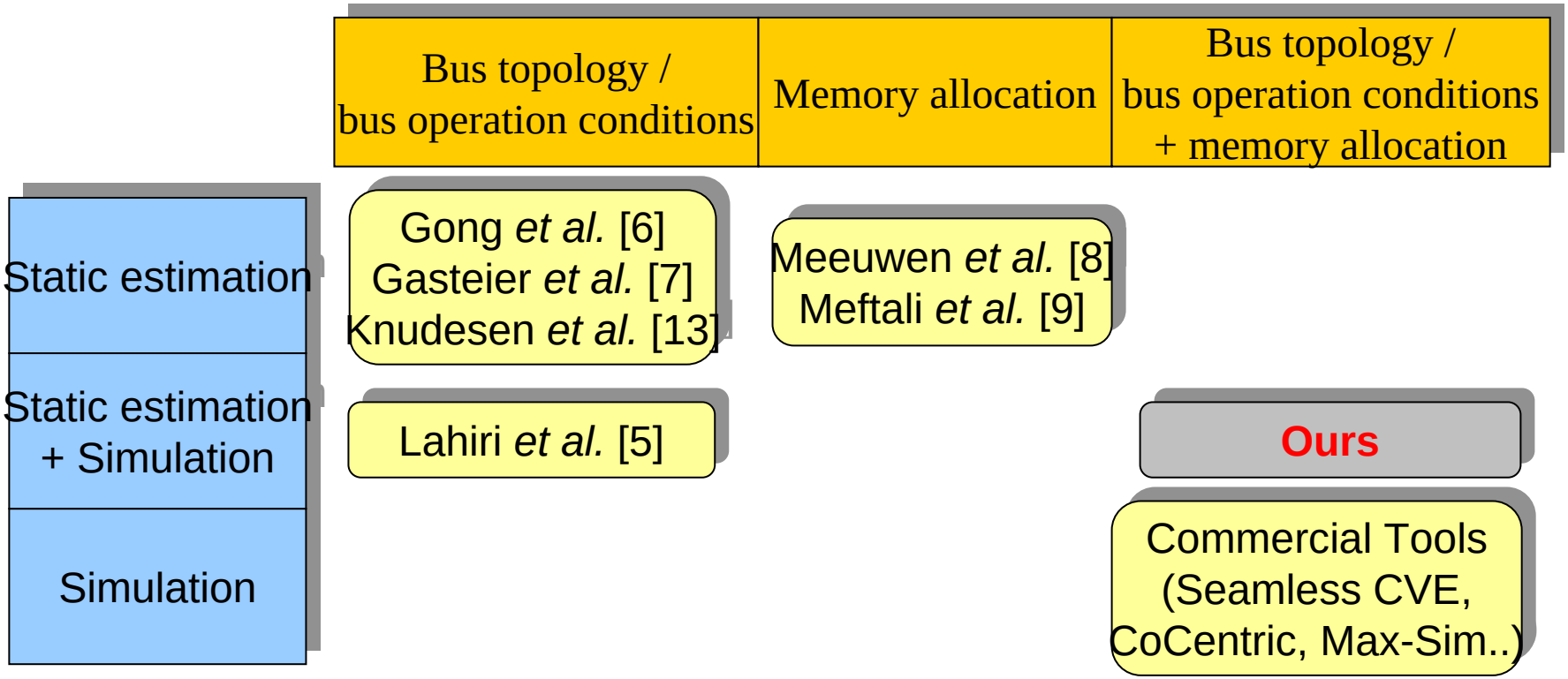

- **Target communication architecture**




• One physical memory in a bus
• Shared memory communication between difference processing elements
• No restriction on separation between local and shared memory

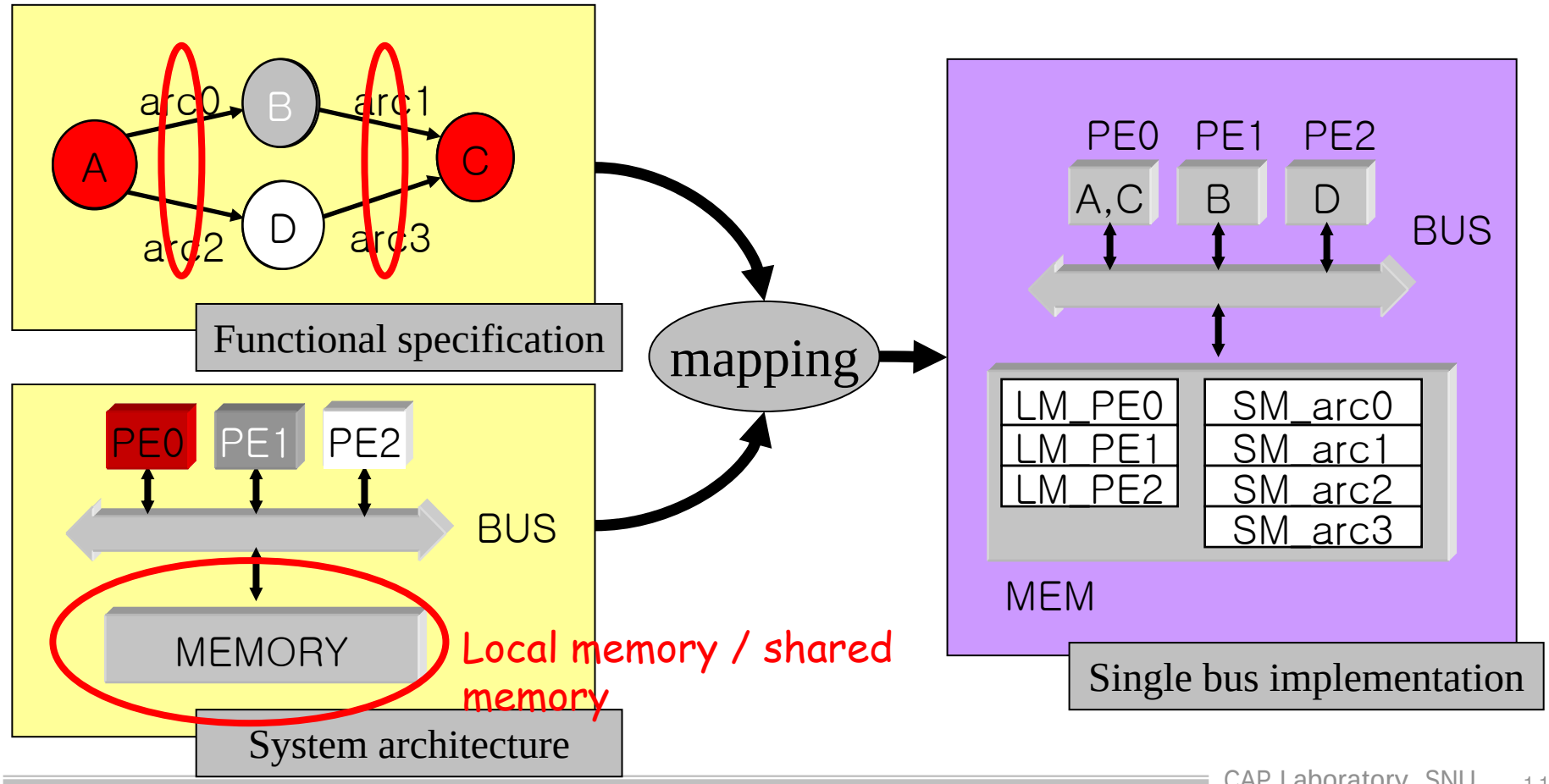
# Related work

- **Classification with regards to**
  - The coverage on the design space with various design axes
  - Evaluation method



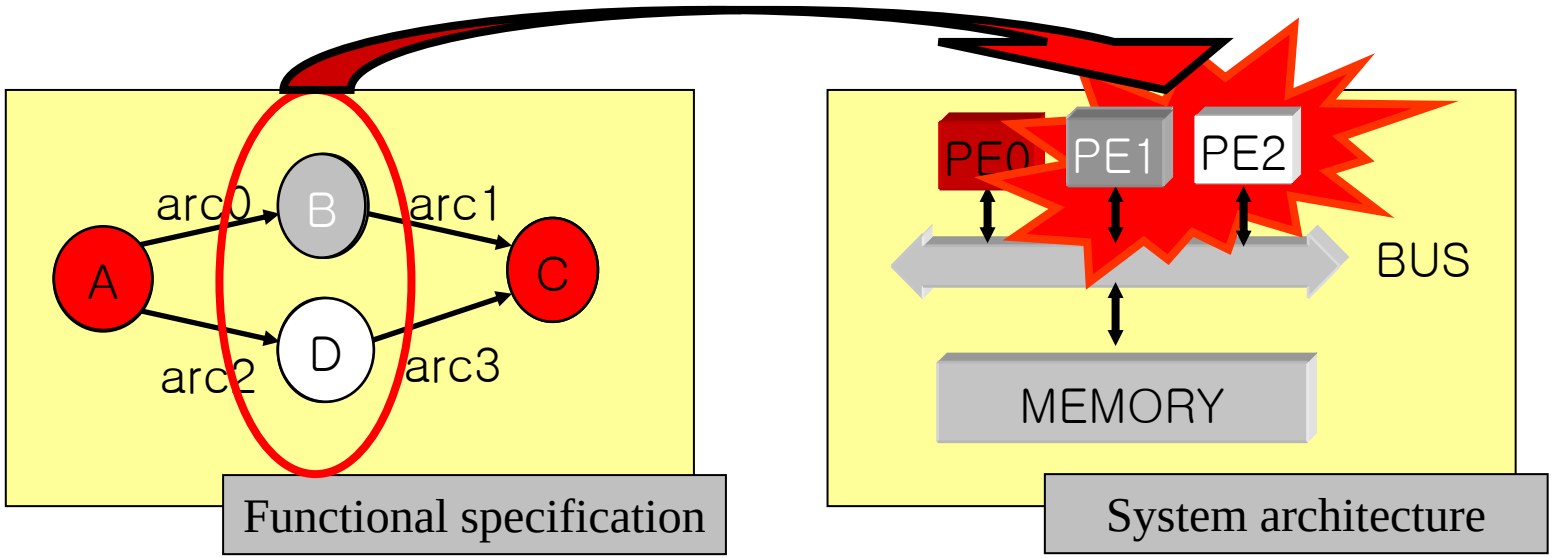
# Basic idea of proposed exploration

- **Motivational example**



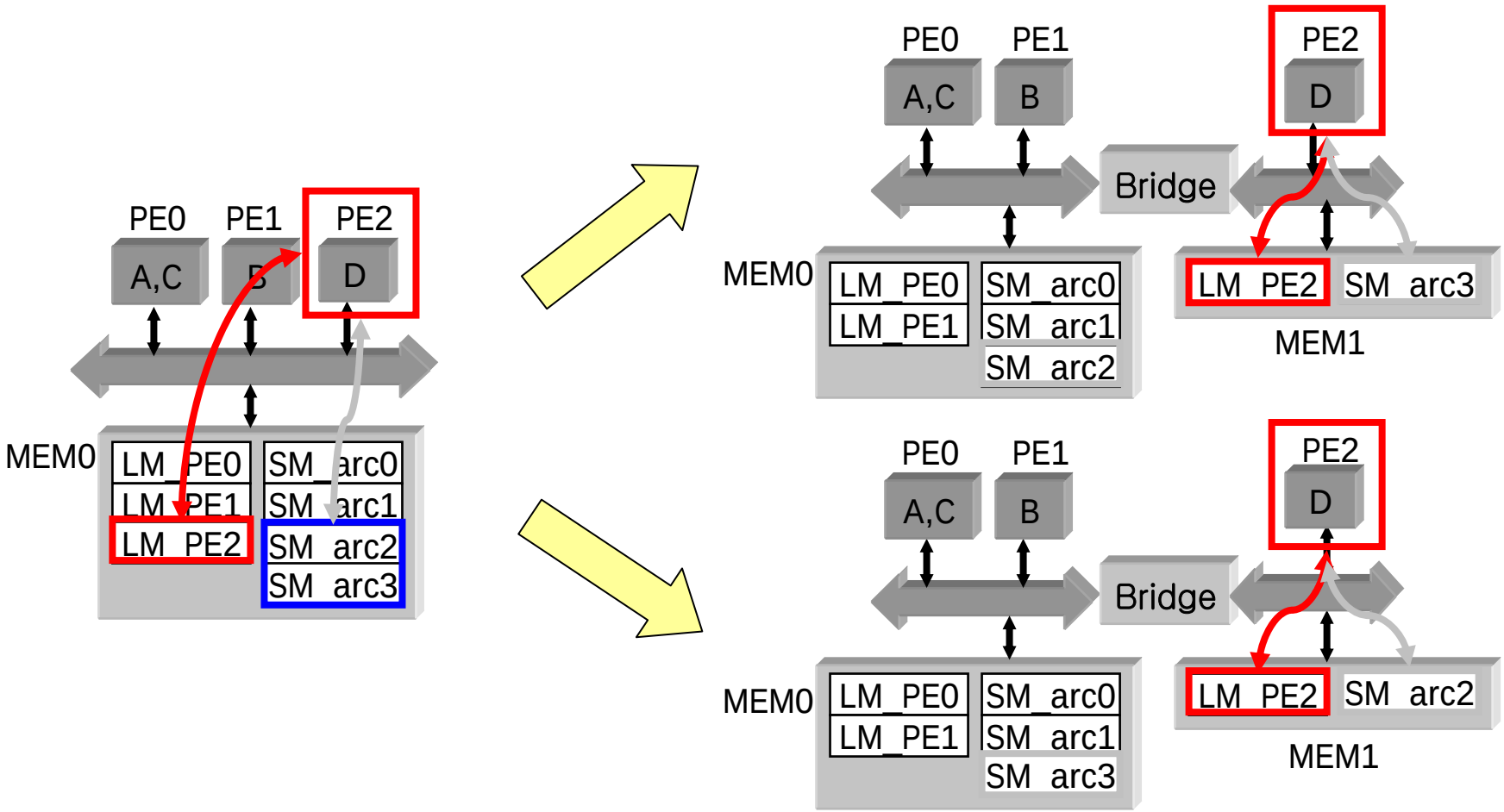
# Basic idea of proposed exploration (cont'd)

- **Poor performance due to excessive bus conflict of the initial bus architecture**
- **Scattering the traffic by simultaneous bus access from processing elements PE0, PE1, and PE2**



# Basic idea of proposed exploration (cont'd)

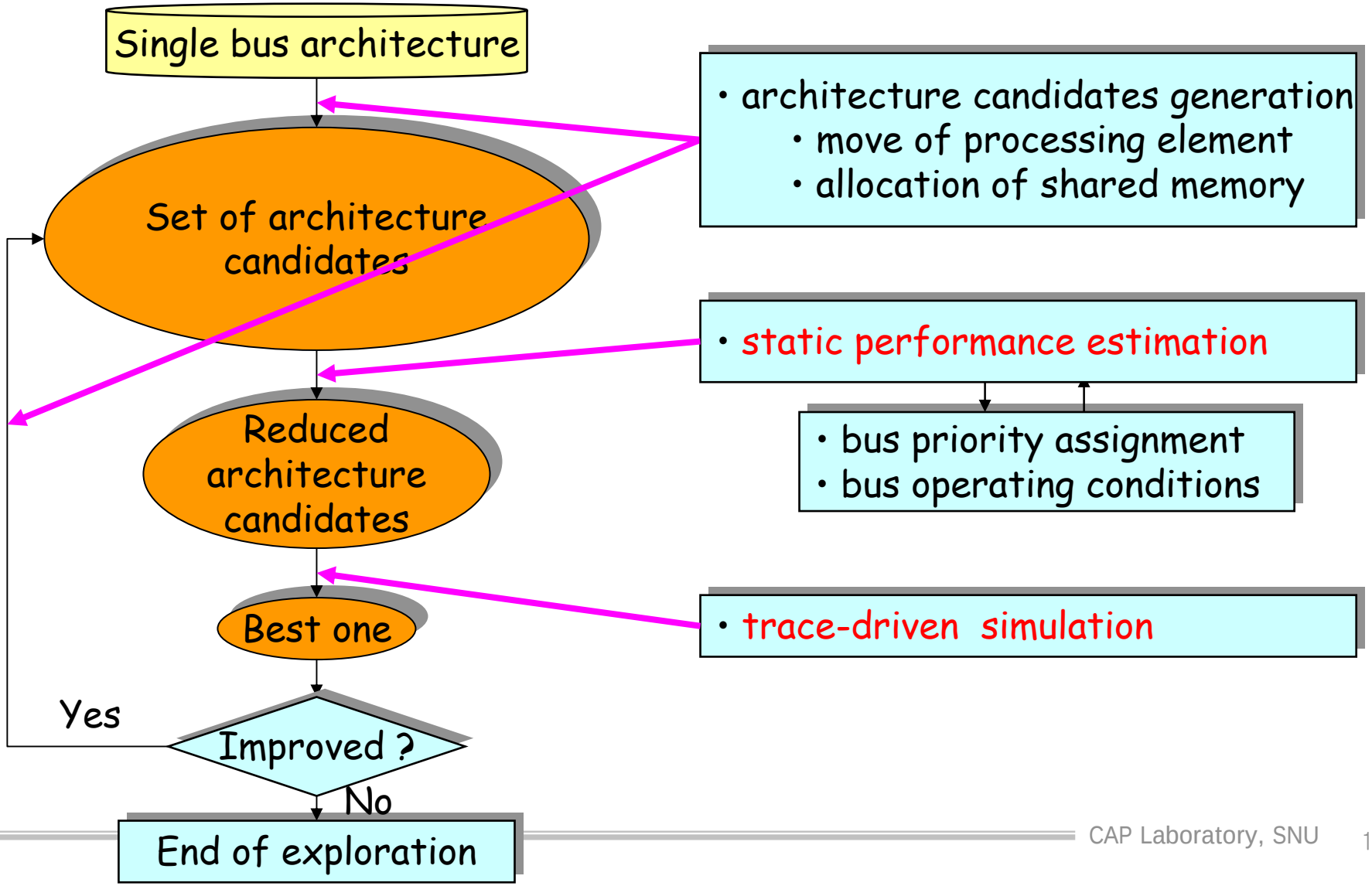
- Generating the design space of communication architectures



# Features of propose technique

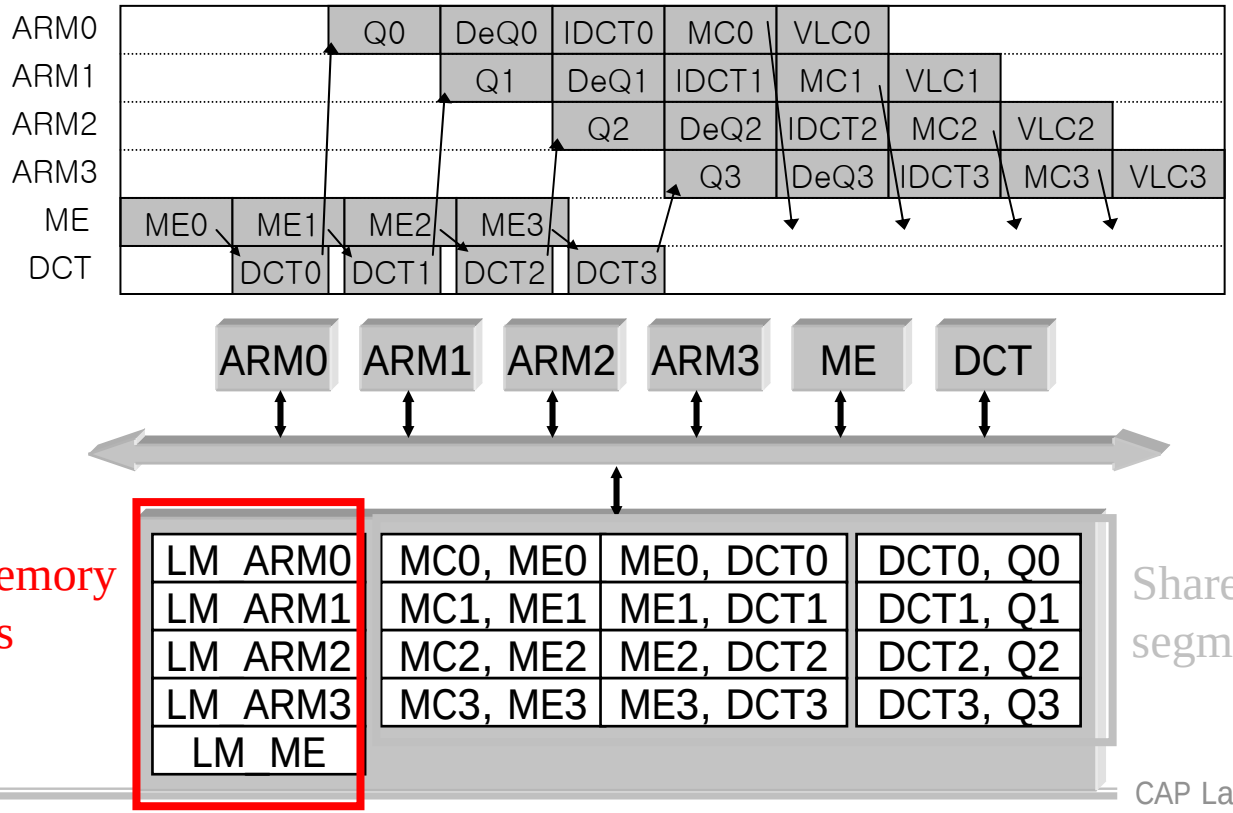
- **Capability of visiting as large design space as possible**
  - Bus topology including multiple buses connected with bridges
  - Memory allocation of shared memory segment for inter-component communication
  - Priority assignment of processing elements on each bus after decision on bus topology
- **Fast and efficient evaluation of the design space**
  - An iterative two step exploration
  - 1st step: Rapid reduction of the large design space to smaller one using static estimation with acceptable accuracy
  - 2nd step: Accurate evaluation of the reduced design space from the first step using cycle-accurate trace-driven simulation

# Proposed exploration flow



# Exploration of a preliminary example

- **4-channel DVR system**
  - Encodes 4 external raw bit streams using 4 H.263 encoders
  - 4 ARM processors, dedicated hardware for ME and DCT

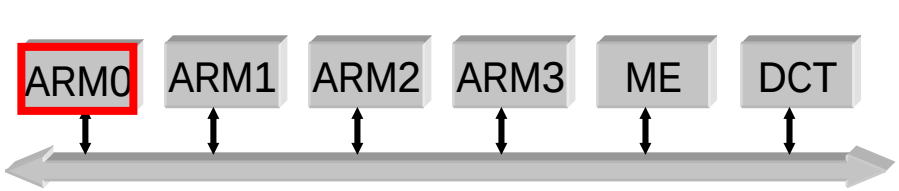


# Generating architecture candidates

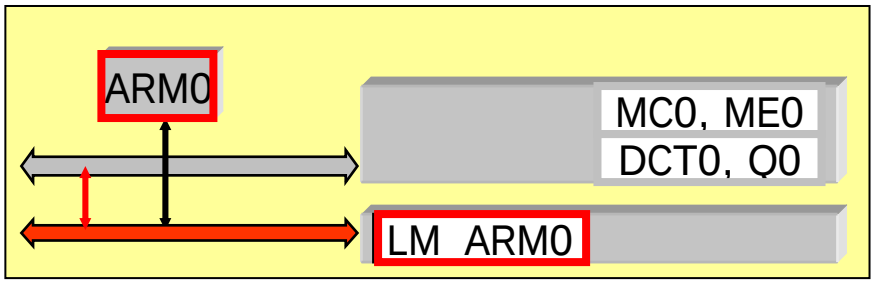
- **The seed architecture**
  - Single bus architecture for initial iteration
  - The best architecture of the previous iteration
- **Exploring bus topology**
  - Move of processing elements and shared memory allocation
- **Exploring bus operating conditions for a given bus topology**
  - Priority assignment of each processing elements on a bus

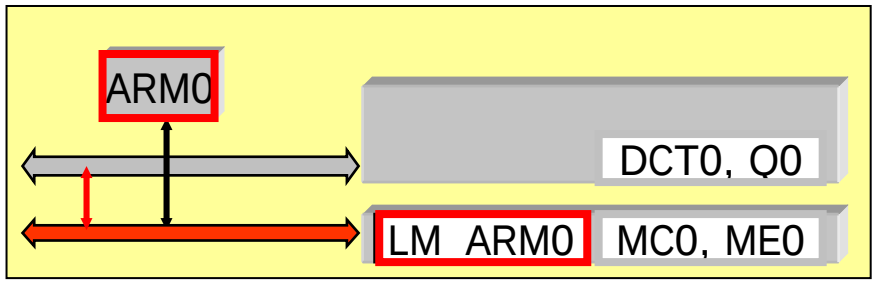
# Exploring bus topology

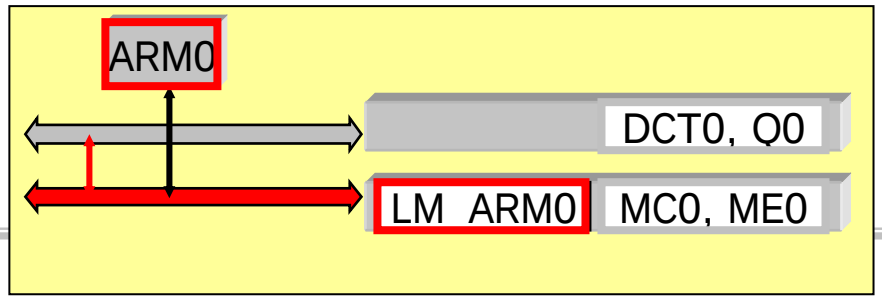
- **Move of ARM0 to a new bus and allocation of its associated shared memory segments**

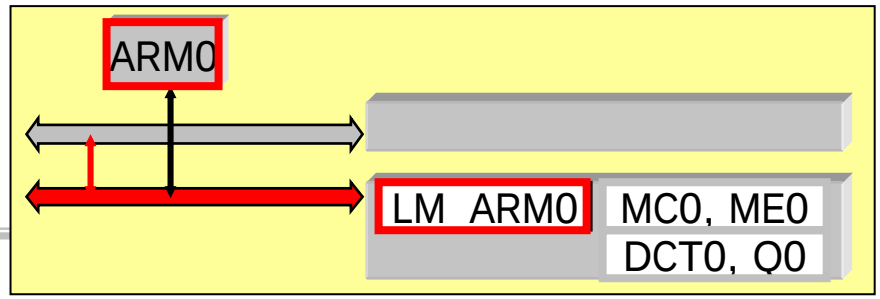



| | | | |
|---|---|---|---|
| LM_ARM0 | MC0. ME0 | ME0, DCT0 | DCT0. O0 |
| LM_ARM1 | MC1, ME1 | ME1, DCT1 | DCT1, Q1 |
| LM_ARM2 | MC2, ME2 | ME2, DCT2 | DCT2, Q2 |
| LM_ARM3 | MC3, ME3 | ME3, DCT3 | DCT3, Q3 |
| LM_ME | | | |



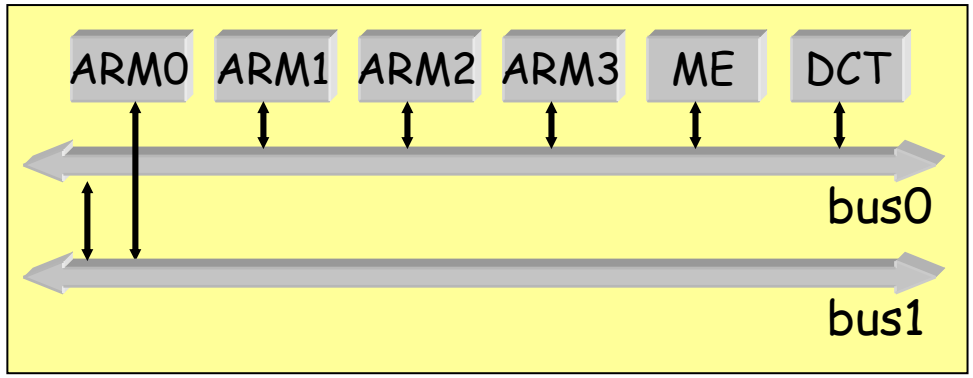
# Exploring bus topology (cont'd)

- **The factors that affect the size of design space by exploring bus topology**
  - The number of processing elements
  - The number of shared memory segments
- **The number of generated architectures from single-bus architecture of 4-channel DVR**

| PE being moved | ARM0 | ARM1 | ARM2 | ARM3 | ME | DCT |
|---|---|---|---|---|---|---|
| # of shared memory | 2 | 2 | 2 | 2 | 8 | 8 |
| # of generated architectures | $2^2$ | $2^2$ | $2^2$ | $2^2$ | $2^8$ | $2^8$ |
| Total | 528 | | | | | |

# Exploring bus operation conditions

- **Bus operation conditions for a given bus topology**
  - Data-width
  - Operation clock frequency
  - Priority assignment of processing elements
- **Priority assignment**
  - Exhaustive search always finds the optimum, but impractical




| | Exhaustive assignment | |
|---|---|---|
| | bus0 | bus1 |
| # of bus masters | 5 PEs + 1 bridge | 1 PE + 1 bridge |
| # of assignment in a bus | 6! | 2! |
| # of assignment in architecture | 6! × 2! = 2880 | |

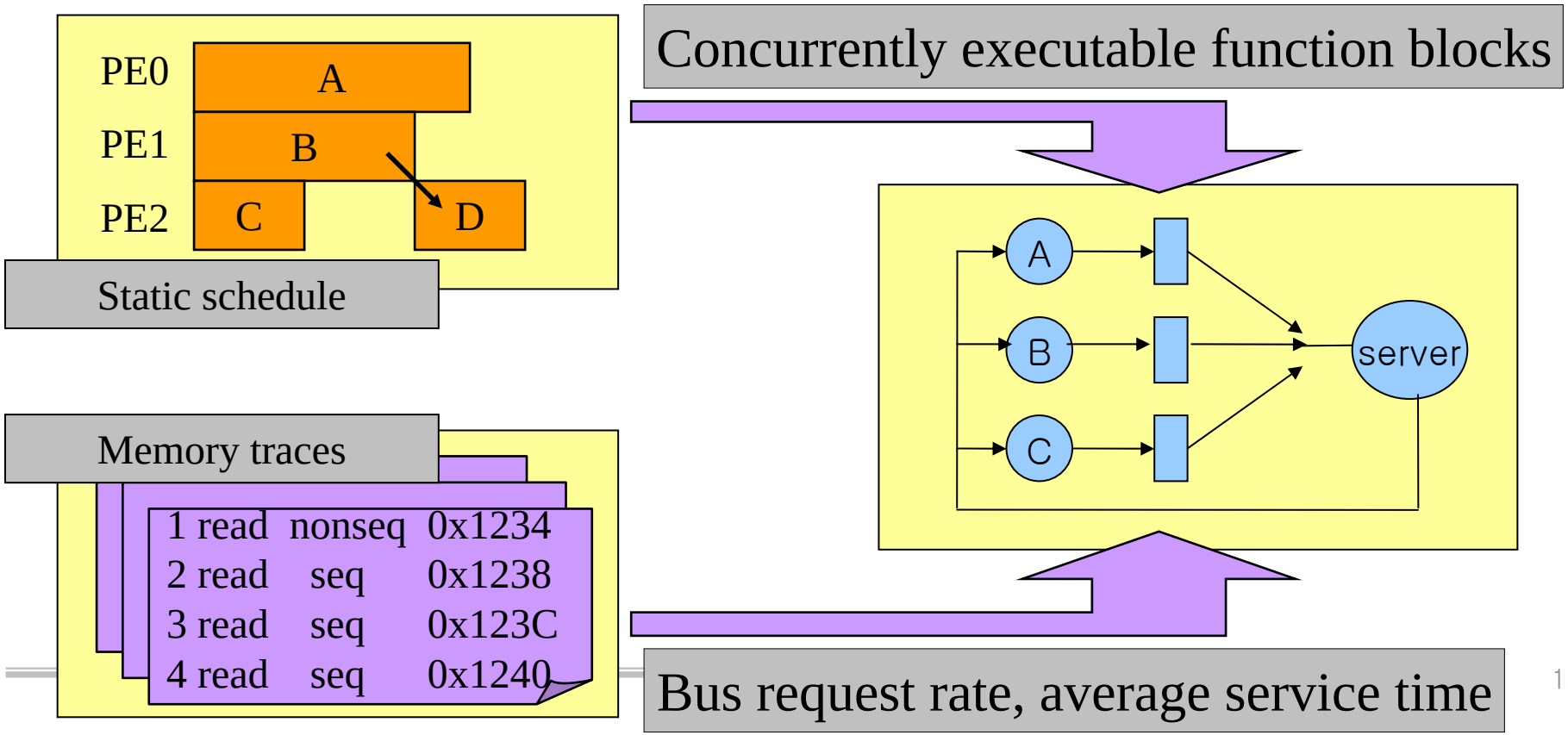
# Heuristic on priority assignment

- **Higher priority to the processing element with**
  - More memory access, i.e. higher bandwidth
  - More critical through the entire execution of an application
- **Fine tuning of priority assignment**
  - Escape local optimum
  - Swap the priority of two processing elements in a bus

| | Exhaustive | | Heuristic | |
|---|---|---|---|---|
| | bus0 | bus1 | bus0 | bus1 |
| # of bus masters | 5 PEs + 1 bridge | 1 PE + 1 bridge | 5 PEs + 1 bridge | 1 PE + 1 bridge |
| # of assignment in a bus | $6!$ | $2!$ | $\binom{6}{2}$ | $\binom{2}{2}$ |
| # of assignment in architecture | $6! \times 2! = 2880$ | | $\binom{6}{2}+\binom{2}{2}=31$ | |

# Static performance estimation

- **Queuing model**
  - Performance evaluation considering dynamic bus conflict
  - Processing elements: customers
  - Bus: single server

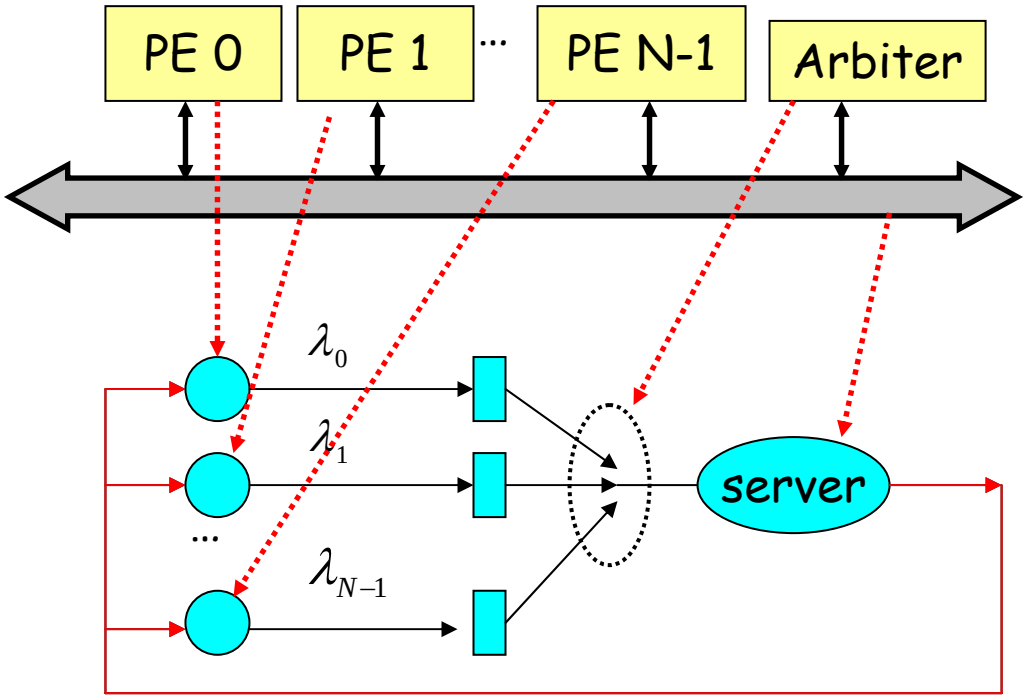


# Base Queuing Model For A Single Bus

- **Performance estimation of bus?**
  - Some waits may occur due to dynamic bus conflicts
  - The key is to estimate the wait due to bus conflict
  - We can estimate average wait of each processing element using queuing model of bus
  - The average time for one memory access is the summation of
    - The average wait for the use of bus
    - Bus protocol specific overhead
    - Memory access time

# Base Queuing Model For A Single Bus

- **Basic notation and idea from Brandwajn**
  - The queuing model of SCSI I/O single bus
  - Fixed priority based arbitration, non-preemptive



# Base Queuing Model For A Single Bus

$\lambda_i$ : The rate at which component $i$ issues a bus request in idle state

$\theta_i$ : The request rate of component $i$ that is actually seen on the bus

$1/\mu_i$ : The mean service time of a server

$u_i = \theta_i / \mu_i$ : The bus utilization of component $i$

$k_i$ : The average number of component $i$ waiting for the use of a bus

$w_i$ : The average wait of component $i$ to grant the use of a bus

$$\theta_i = (1 - k_i - u_i)\lambda_i = (1 - k_i - \theta_i / \mu_i)\lambda_i \quad (1)$$

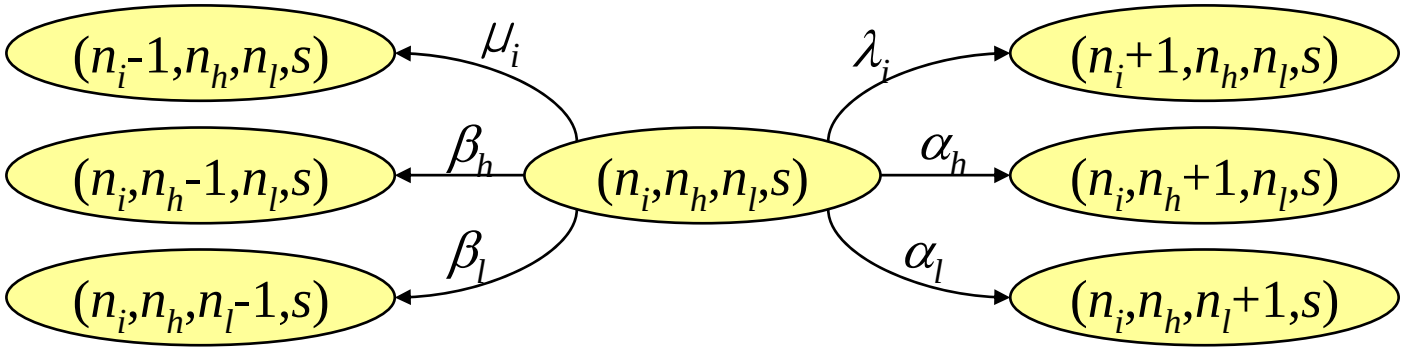
$$k_i = \theta_i \cdot w_i \quad (2)$$

# Base Queuing Model for A Single Bus

- **The steady state probability of processing element i : p(ni,nh,nl,s)**
  - ni: the number of request that processing element i issues
  - nh: The number of requests from the processing elements that have higher priority than processing element i
  - nl: The number of requests from the processing elements that have lower priority than processing element i
  - s: The class of the processing element that uses a bus

# Base Queuing Model For A Single Bus

- **State transition diagram at the steady state**




$$\alpha_h \approx (N_h - n_h)\gamma_h \quad \text{where,} \sum_{j=0}^{i-1}\theta_j = \left\{ N_h - \sum_{j=0}^{i-1}(k_j + u_j) \right\}\gamma_h \qquad (1)$$

$$\beta_h \approx \sum_{j=0}^{i-1}\left( \mu_i \cdot u_j \Big/ \sum_{k=0}^{i-1} u_k \right)$$

- To get $k_i$

$$\sum_{n_h=0}^{N_h}\sum_{n_l=0}^{N_l}\sum_{S} p(1, n_h, n_l, s) = u_i + k_i \qquad (2)$$

codesign environment

# Our Extensions

- **Aims for improving the estimation accuracy**
  - Simultaneous events
  - System level task scheduling

# Extension 1 : Simultaneous Events

- **Processor-memory system bus is quiet different from I/O bus**
  - I/O bus (e.g. SCSI bus, IDE bus…)
    - The bus request interval is long enough compared with the service time
    - It is possible to estimate accurately without considering the requests more than two at the same time
  - Processor-memory system bus
    - Bus request interval is shorter than the service time.
    - Bus requests occur very frequently
    - More than one request at most event sampling point(e.g. clock edge)
- **The consideration for the simultaneous events is required**

# Extension 1 : Simultaneous Events

- **Consideration on the simultaneous events**
  - The more accurate estimation becomes, the more modeling complexity increase
- **The trade-off between accuracy and complexity**
  - At most two simultaneous events can be issued
- **Synchronous system bus**
  - The state transition rate should be replaced with the state transition probability within a clock period

$$\text{Example Transition}: (n_i, n_h, n_l, s) \rightarrow (n_i, n_h + 1, n_l + 1, s)$$

$$\text{Transtion Rate} \quad : (1-\lambda_i) d_h d_l (1-S)$$

$$\text{where } d_h = 1-(1-\gamma_h)^{N_h - n_h}, \; d_l = 1-(1-\gamma_l)^{N_l - n_l}$$

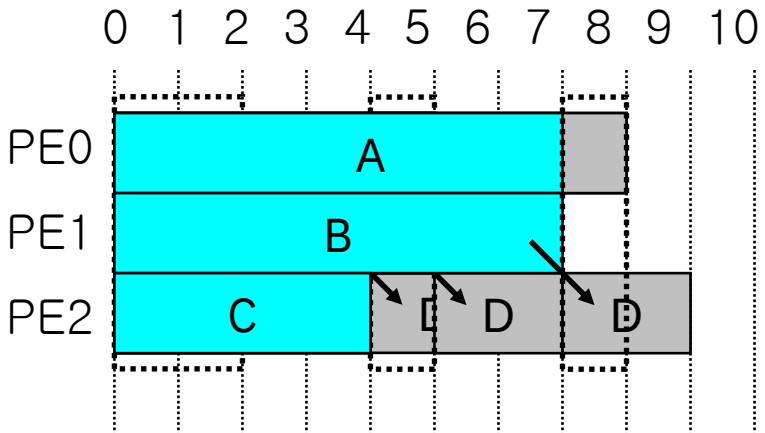
# Extension 1 : Simultaneous Events

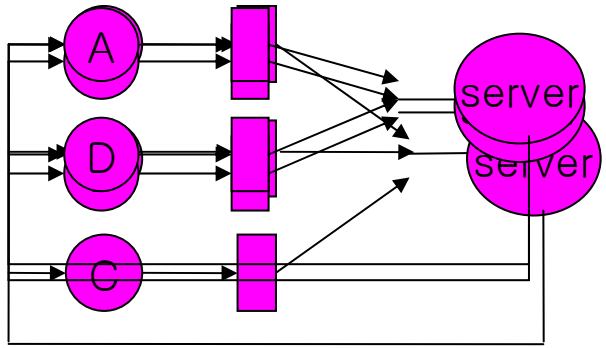
- **All possible transition from the state (ni,nh,nl,s)**

| Type 1 | | Type 2 | |
|---|---|---|---|
| Next state | Transition rate | Next state | Transition rate |
| $(n_i+1, n_h, n_l, S)$ | $\lambda_i(1-d_h)(1-d_l)(1-S)$ | $(n_i+1, n_h+1, n_l, S)$ | $\lambda_i d_h(1-d_l)(1-S)$ |
| $(n_i, n_h+1, n_l, S)$ | $(1-\lambda_i)d_h(1-d_l)(1-S)$ | $(n_i, n_h+1, n_l+1, S)$ | $(1-\lambda_i)d_h d_l(1-S)$ |
| $(n_i, n_h, n_l+1, S)$ | $(1-\lambda_i)(1-d_h)d_l(1-S)$ | $(n_i+1, n_h, n_l+1, S)$ | $\lambda_i(1-d_h)d_l(1-S)$ |
| $(n_i-1, n_h, n_l, S)$ | $\mu_i(1-d_h)(1-d_l)$ | $(n_i-1, n_h+1, n_l, S)$ | $\mu_i d_h(1-d_l)$ |
| $(n_i, n_h-1, n_l, S)$ | $(1-\lambda_i)\beta_h(1-d_l)$ | $(n_i-1, n_h, n_l+1, S)$ | $\mu_i(1-d_h)d_l$ |
| $(n_i, n_h, n_l-1, S)$ | $(1-\lambda_i)(1-d_h)\beta_l$ | $(n_i+1, n_h+1, n_l, S)$ | $\lambda_i\beta_h(1-d_l)$ |
| | | $(n_i, n_h-1, n_l+1, S)$ | $(1-\lambda_i)\beta_h d_l$ |
| | | $(n_i+1, n_h, n_l-1, S)$ | $\lambda_i(1-d_h)\beta_l$ |
| | | $(n_i, n_h+1, n_l-1, S)$ | $(1-\lambda_i)d_h\beta_l$ |

# Extension 2: Making Use of Scheduling

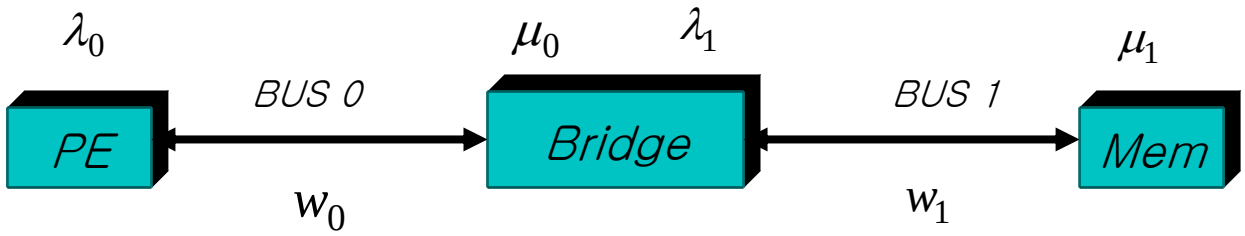
- **Simple static estimation**
  - Assuming that the bus requests are distributed evenly over the entire system execution
  - Main reason of inaccuracy
- **Proposed technique**
  - Use system level task scheduling
    - Disclose the active duration of bus requests



# Extension 2: Making Use of Scheduling

- Multiple bus system connected with the bridges




- **When PE at BUS 0 accesses Mem at BUS 1 though Bridge**
  - Bridge is seen
    - As a memory at BUS 0

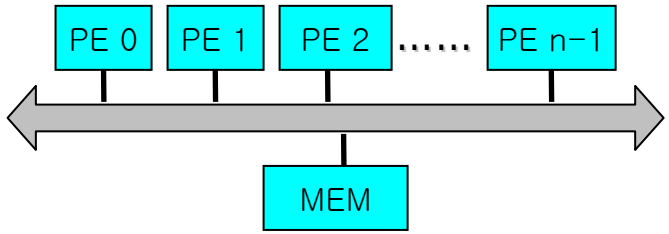
$$\frac{1}{\mu_0} = w_1 + O_{bridge} + \frac{1}{\mu_1} \qquad (1)$$

    - ✓ As a processing element at *BUS 1*

$$\frac{1}{\lambda_1} = \frac{1}{\lambda_0} + O_{bridge} + w_0 \qquad (2)$$

# Experiment 1 : Set Up

- **Estimation on various architecture configuration for simple system level task scheduling**
  - Scheduling and target communication architecture


    - Adjustable parameters
      - ✓ Number of processing elements
      - ✓ Arrival rate
      - ✓ Mean service time

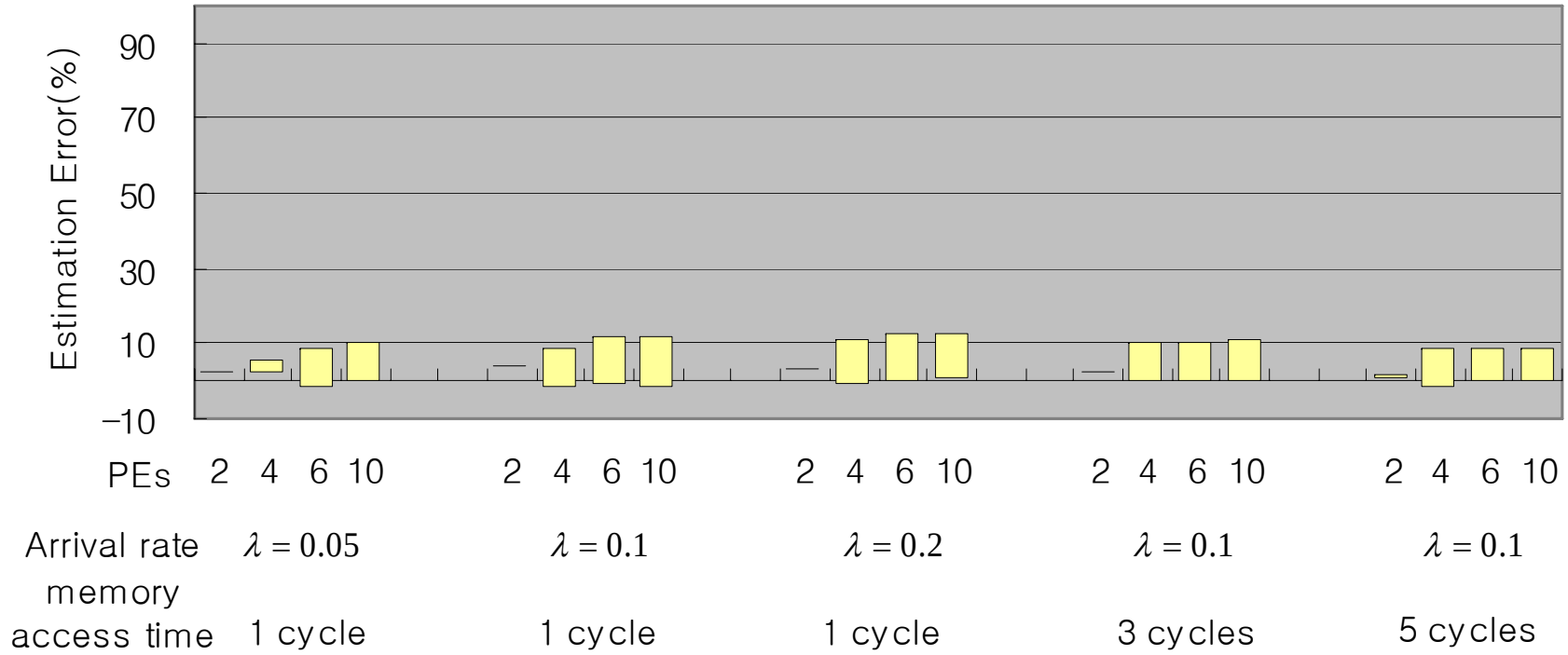
# Experiment 1 : 1st Result

- **The comparison of base queuing model and our extension considering simultaneous events according to the bus request rate**
  - The number of processing elements : 2, 4, 6, 10

| | $\lambda_i = 0.1$ | | | $\lambda_i = 0.2$ | | |
|---|---|---|---|---|---|---|
| Error(%) | Base | Extension | Improved | Base | Extension | Improved |
| Minimum | 0.73 | 0.21 | 71.22 | 2.40 | 2.41 | -0.18 |
| Maximum | 16.02 | 12.14 | 24.19 | 16.73 | 12.25 | 26.75 |
| Average | 7.01 | 4.20 | 40.05 | 6.90 | 5.65 | 18.19 |

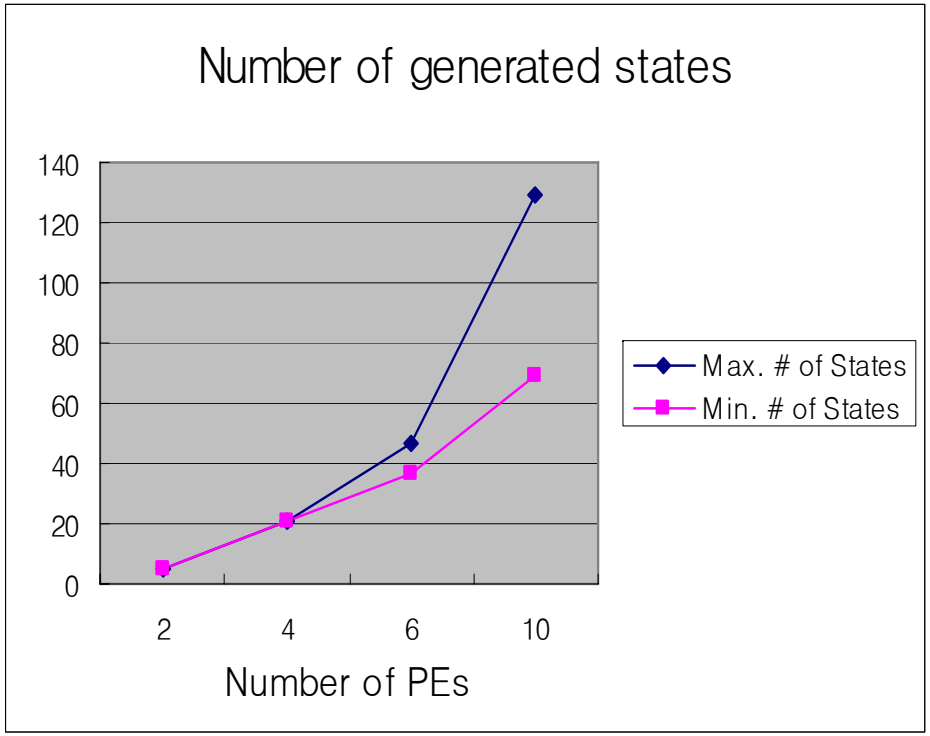
# Experiment 1 : 2nd Result

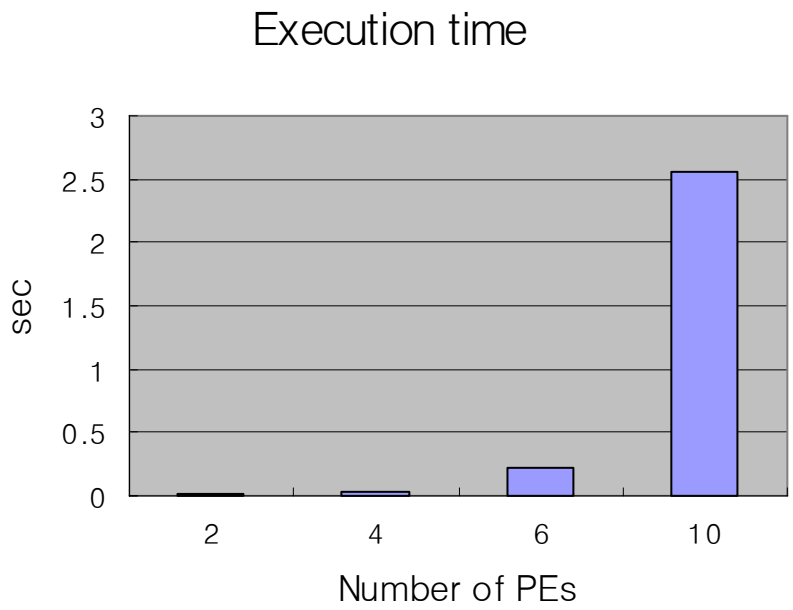
- **Estimation errors compared with the simulation according to the various communication architecture parameters**



# Experiment 1 : Final Result

- **The complexity of proposed technique**
  - Xeon 1.8GHz, 1GB Memory, Linux



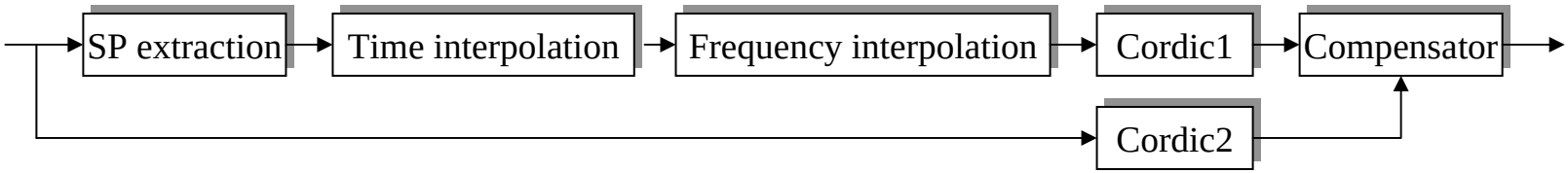



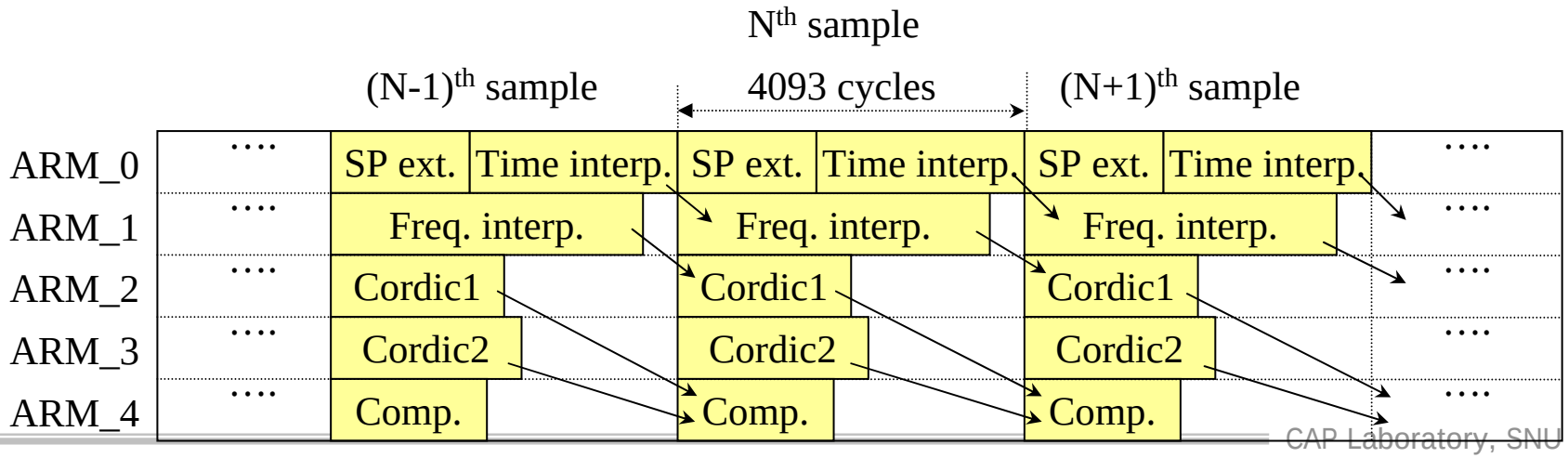
# Trace-driven simulation

- **In-house trace-driven simulator**
- **Bus protocol**
  - Simplified AMBA AHB burst read/write
  - No pipelined address/data bus
  - No split-transaction
  - No error-handling

# Experiment – Setting up

- **Exploration of two preliminary applications**
  - 4-channel DVR
  - The equalizer of OFDM DVB-T Receiver
- **The equalizer of OFDM DVB-T Receiver**



# Experiment – Results

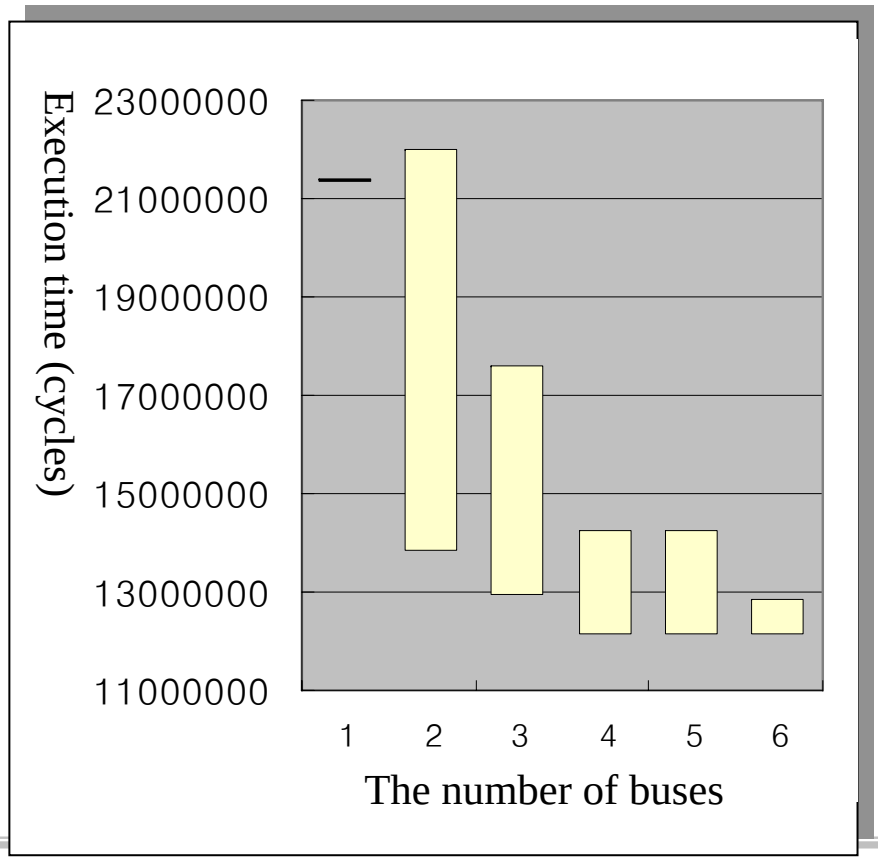
- **Exploration summary according to the number of buses**

| Application | DVR | | Equalizer | |
|---|---|---|---|---|
| # of shared mem | 14 | | 4 | |
| # of buses | # of arch | Improvement | # of arch | Improvement |
| 1 | 11 | 1 | 7 | 1 |
| 2 | 5912 | 1.5511 | 167 | 1.5871 |
| 3 | 3753 | 1.6585 | 107 | 1.7664 |
| 4 | 2119 | 1.7651 | 68 | 1.7661 |
| 5 | 1058 | 1.7686 | 12 | 1.7671 |
| 6 | 260 | 1.7689 | | |
| Total | 13112 | | 360 | |
| Pruning ratio(%) | 99.6 | | 94.4 | |
| Estimation/arch in 1st step (sec) | 0.67 | | 0.06 | |
| Total exe. (sec) | 12063 | | 24 | |

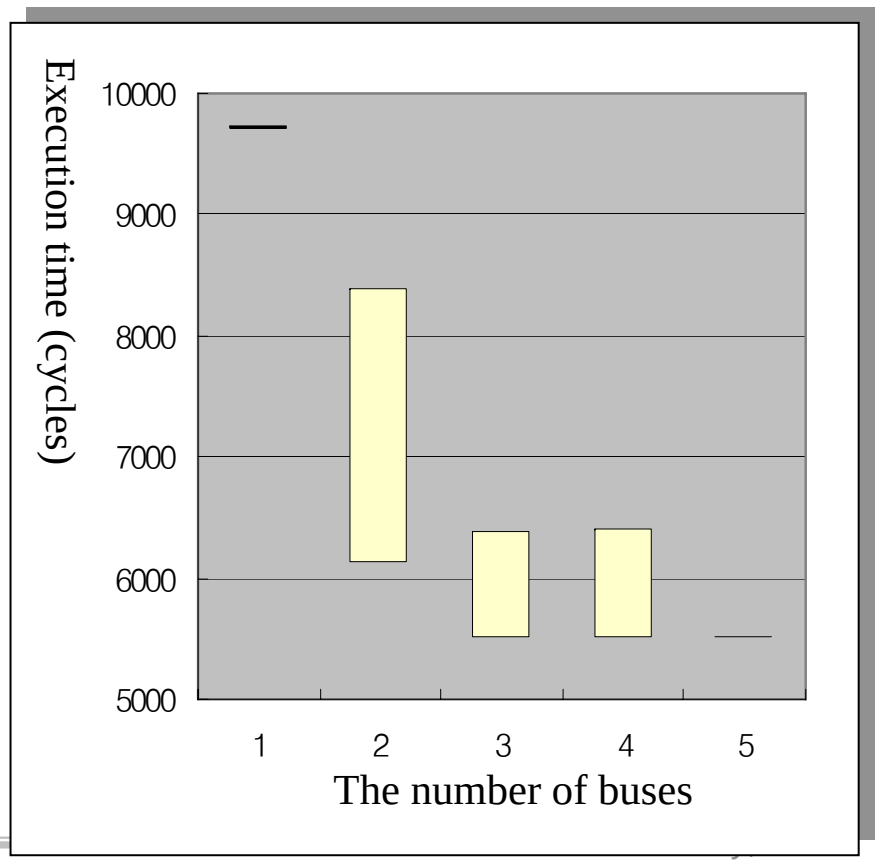
# Experiment – Results (cont'd)

- **Performance variation according to the number of buses**

4-channel DVR




The equalize of OFDM receiver



codesign environment

# Conclusion

- **Iterative two-step exploration technique**
  - 1st step: Rapid reduction of the large design space using static performance estimation
  - 2nd step: Accurate performance evaluation on the reduced design space
  - The validation with two preliminary applications
- **Future works**
  - More complicated memory architecture
  - Consideration on system power consumption

# Questions

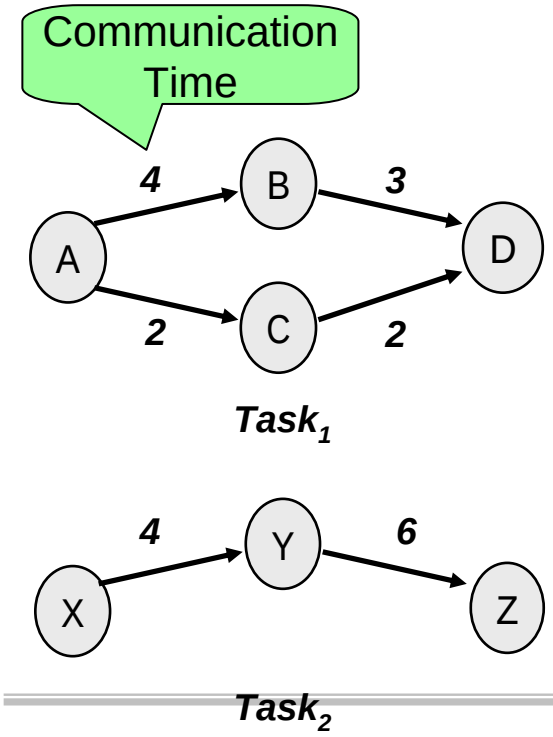
- 1. Explain a popular method of estimating the system power.
- 2. Explain how DSE is performed in PeaCE.
- 3. What is “pareto optimal design”? .
- 4. Compare the HW/SW partitioning techniques. What are the characteristics fo PeaCE approach?
- 5. PeaCE uses a queuing analysis to reduce the design space of bus architecture. Explain the meaning of the following formula.

$$\theta_i = (1 - k_i - u_i)\lambda_i = (1 - k_i - \theta_i / \mu_i)\lambda_i$$

$$k_i = \theta_i \cdot w_i$$

# Questions (cont.)

- **5. Find an optimal HW/SW cosynthesis result through PeaCE heuristic**
  - (a) In case only a task is running at any time.
  - (b) In case two tasks can run concurrently.




| | $Task_1$ | $Task_2$ |
|---|---|---|
| Period | 60 | 40 |
| Deadline | 60 | 40 |

| | $P_1(20)$ | $P_2(40)$ |
|---|---|---|
| A | 20 | 10 |
| B | 10 | 6 |
| C | 20 | 10 |
| D | 10 | 7 |

Cost

| | $P_1(20)$ | $P_2(40)$ |
|---|---|---|
| X | 10 | 5 |
| Y | 15 | 7 |
| Z | 20 | 10 |

Execution time