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

- **PeaCE**
  - [30] Dohyung Kim, Minyoung Kim and Soonhoi Ha, "A Case Study of System Level Specification and Software Synthesis of Multi-mode Multimedia Terminal", Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), Newport Beach, CA, USA Oct 2003
- **Models for Embedded Software**
  - [31] Jie Liu and Edward A. Lee, “Timed Multitasking for Real-Time Embedded Software,” IEEE Control Systems Magazine, pp. 65-75, February 2003.
    [32] T.A.Henzinger, B.Horowitz, and C.M.Kirsch, “Embedded Control Systems Development with Giotto,” LCTES (Languages, Compilers, and Tools for Embedded Systems) Proceedings, pp. 64-72, June, 2001.

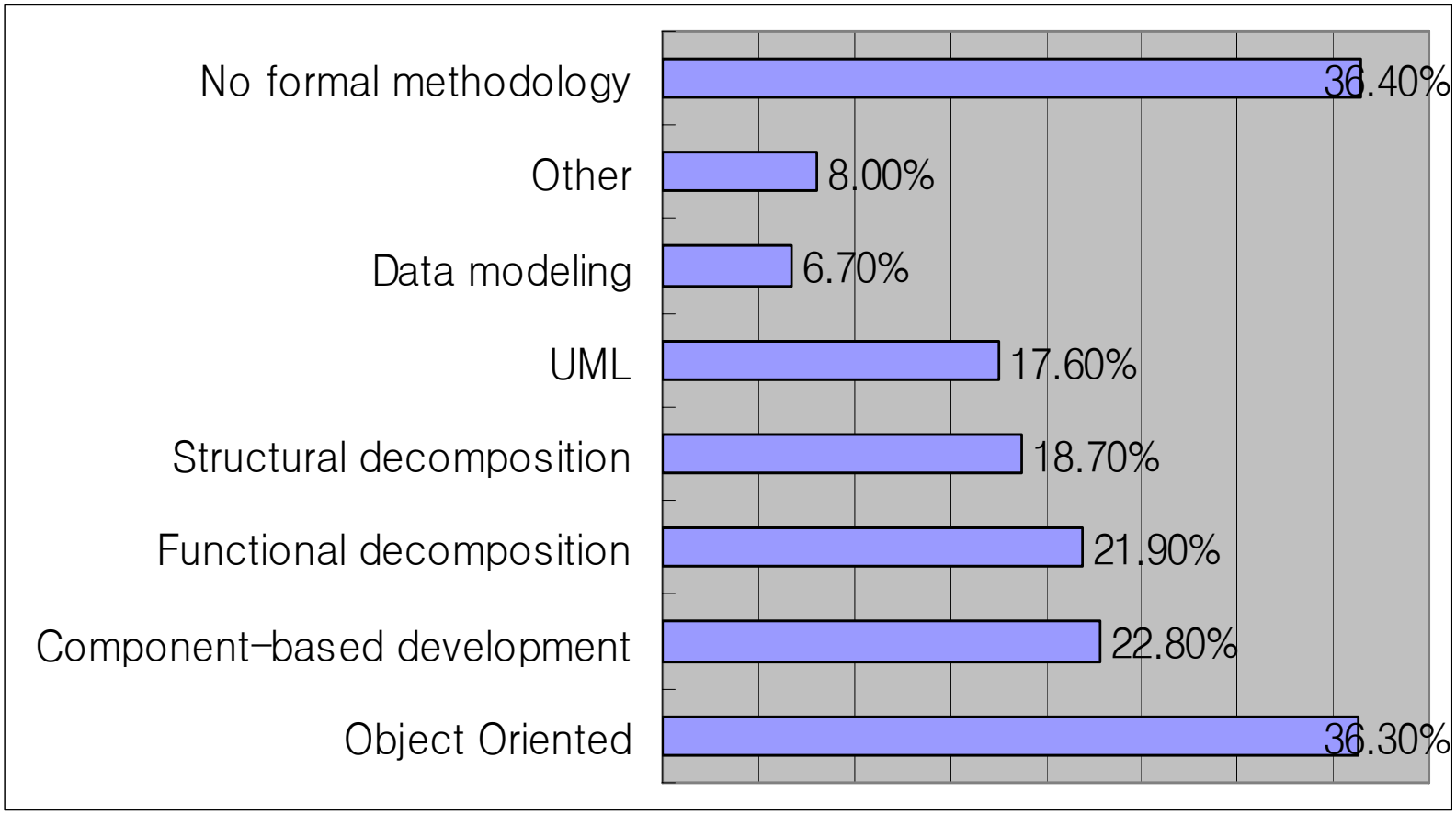
Outline

- Introduction
- Port-based Model
- TM Model
- HOPES approach

Embedded Software Design

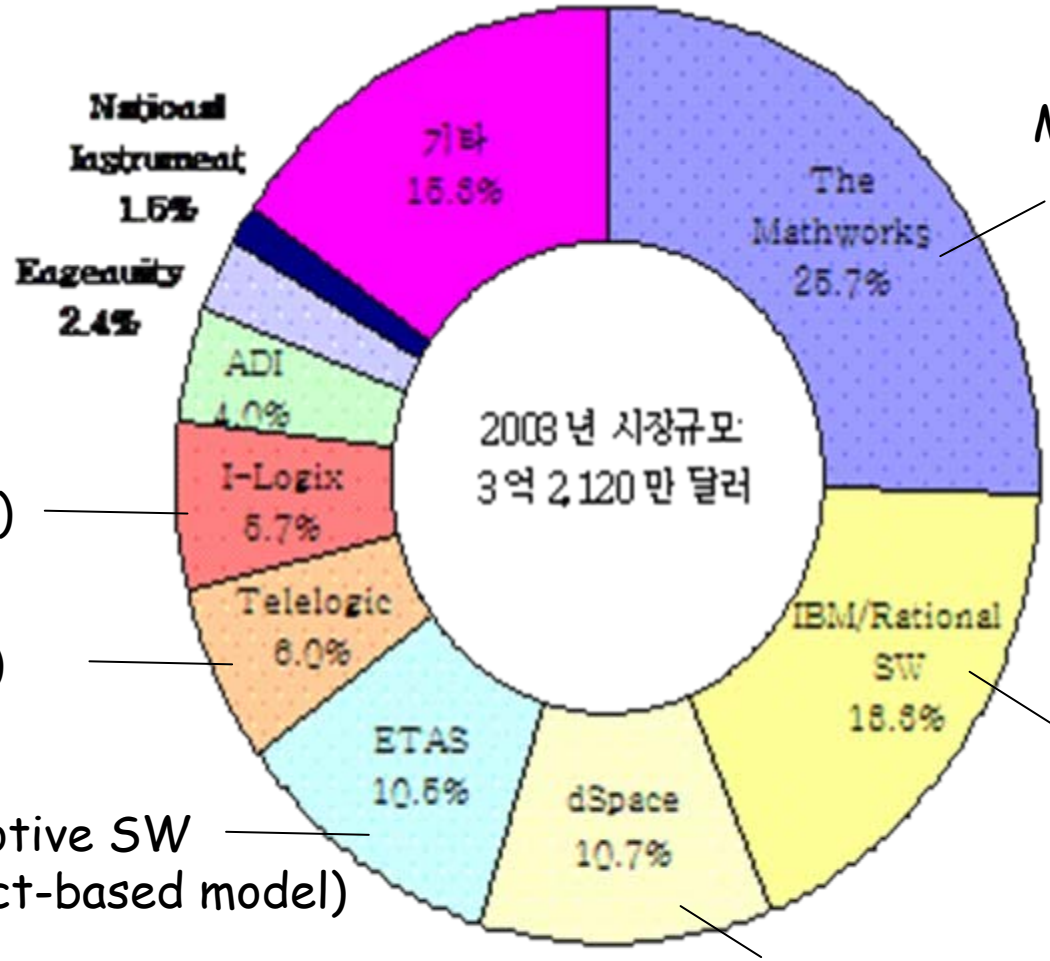
- **Embedded Software Platform Design**
  - Operating System
  - Middleware
- **Application Software Design**
  - **Automatic Code Generation**
  - Source code management, documentation
- **Software Development Environment**
  - Compiler
  - Debugger

Embedded SW Design Methodologies



Gartner Dataquest (January 2003):
Percent of respondents N = 448 (multiple responses)

Tools for Embedded SW Design



Matlab/Simulink/ Stateflow

Rhapsody (UML)

TAU (UML)

Rose RT (UML)

Automotive SW
ASCET (Object-based model)

Automotive SW (Simulink/Stateflow)

<자료>: VDC, 2004. 7. 재구성

(그림 2) 세계 임베디드 설계 자동화 도구 시장 업체별 점유율

Embedded SW Development

- **No formal methodology: C programming**
- **Object-oriented modeling**
  - UML-based method is popular
  - Passive components interact principally through transfer of control (method calls)
- **Actor-oriented modeling**
  - Actors are concurrent, in charge of their own actions
  - Actors interact principally through exchange of data
  - Simulink, Ptolemy II, PeaCE approach falls into this category

Actor-Oriented Component Frameworks

- Simulink (The MathWorks)
- Labview (National Instruments)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- …

Codesign Tools: Embedded SW

- **UML based tools**
  - IBM Rational Rose RT
  - Telelogic TAU: SDL model inside

  - Object-oriented model
  - Well-documented SW production
  - Not executable model itself
  - Scenario-based performance estimation
  - Production-level C code is not achievable yet

codesign environment

Simulink

- **de facto standard in many industrial application domains, particularly automotive control.**
- **Started purely as a simulation environment**
- **Has a multitude of semantics depending on user-configurable options, informally and sometime only partially documented**

- **Simulink-based Tools**
  - The Mathworks: Real-Time Workshop
  - dSPACE: TargetLink
    - *Generate code only for blocks of the dSpace-provided Simulink library*

Design Challenges of Embedded SW

- **Interfacing to the real world – real-time requirement**
  - Reacts to physical and user-interaction events.
  - Both value and time affect the physical outputs of embedded systems: “time” is the first class citizen
- **Increasingly networked**
  - Downloadable modules that dynamically reconfigure the system
- **Increased complexity - concurrency**
  - Need of aggregate components performs computation on limited and competing resources
  - Parallel programming is needed for MPSoC
- **Performs computation on limited and competing resources – resource constraints**

Conventional Approach

- **Separation of Functionality (at design time) and Timeliness (at run-time)**

- **Procedures**
  - Termination computation
- **Object Orientation**
  - Passive

- **RTOS**
  - Resource manager: allocate resources and schedule task activation
  - Real-time scheduling: assign priorities to tasks and execute the highest priority task.
    - *RM (rate monotonic) scheduling: static priority assignment for periodic tasks*
    - *EDF (earliest deadline first) scheduling: dynamic priority assignment*

Facts of Real-Time Scheduling

- **Assumptions on tasks and resources (for RM and EDF)**
  - A single and arbitrarily preemptible resource (CPU)
    - *When multiple resources are being managed, optimal scheduling becomes NP-hard*
  - Fixed and known task execution times
  - Each task must be completed before receiving the next trigger
  - Independent tasks

- **Priority inversion problem**
  - High-priority task is blocked by low-priority tasks indefinitely due to critical section management
  - Solution: Priority inheritance and priority ceiling protocols
    - *Raise the task priority to the ceiling priority when it enters into a critical section*
    - *Overhead is not trivial (VxWorks, QNX). Some lightweight real-time kernel does not support them*

More pitfalls

- **Preemptive execution with static priority is fundamentally fragile**
  - Timing behavior of tasks may be very sensitive to task activation time and their actual execution time
  - Schedulability analysis considers the worst-case execution times, which is often too conservative
  - Schedulability analysis does not provide useful information on the timing behavior.
  - Pushing for fast response time may not be optimal

Need better program specification for real-time embedded software

Need of Better Abstraction

- **Timeliness**
  - "priority" is not enough.
- **Concurrency**
  - Threads, processes, semaphores, and monitors are like assembly language in abstraction
- **Liveness**
  - Turing paradigm is not good for error condition
  - Correctness includes "timeliness" also.
- **Interfaces**
  - Procedures are terminating computations: poor match for embedded system
  - Processes and threads are extremely weak for compositions
- **Heterogeneity**
  - Periodic, or aperiodic events
  - Different timing requirements and tolerances
- **Reactivity**
  - QoS demands may change as conditions change

Desired Program Specification

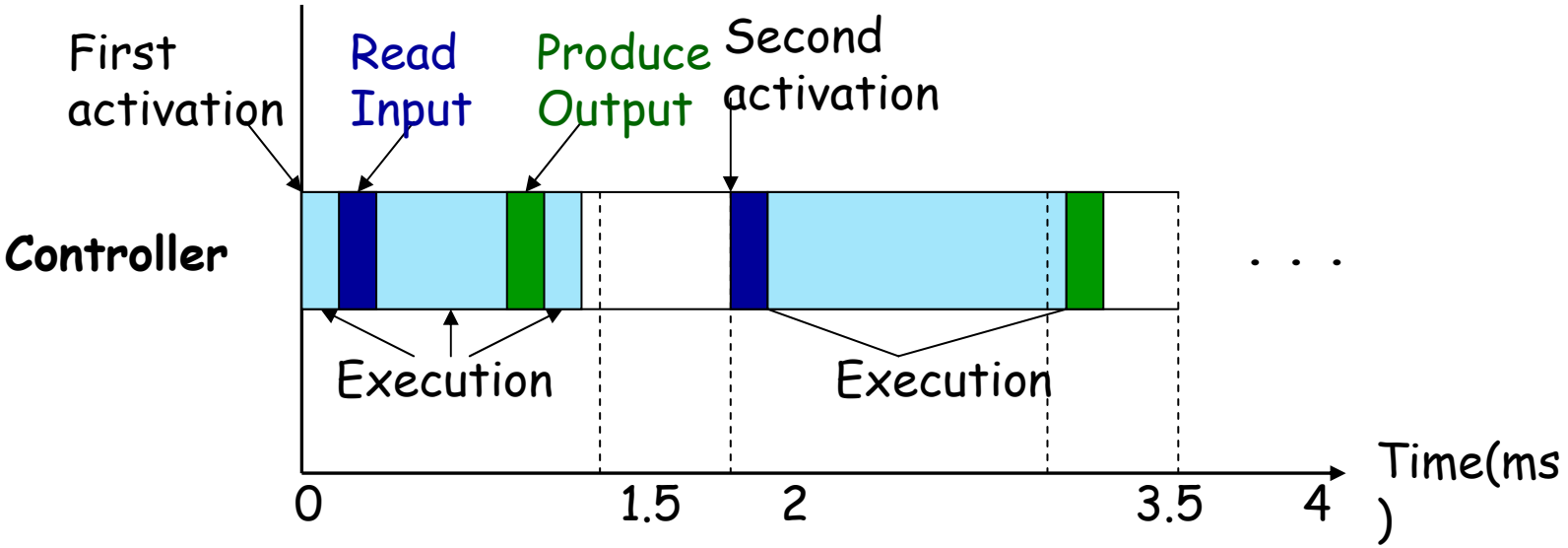
- **A Component-Based SW Architecture satisfying the followings:**
  - Expose resource utilization and concurrent interaction among SW components
  - Bring the notion of time and concurrent interactions to the programming level.
  - Specification, compilation, and execution mechanism preserve timing properties throughout the software life cycle.
- **Time-triggered approach**
  - Port-based objects
  - Giotto.
- **Timed multitasking (TM model in Ptolemy II)**

Outline

- Introduction
- Port-based Model
- TM Model
- HOPES approach

Port-Based Objects

- **Time triggered (applied to communication of channels of tasks)**
- **Communication channel forms a global data space**
  - It has a state semantics meaning that the value of a variable is preserved as long as it is not overwritten



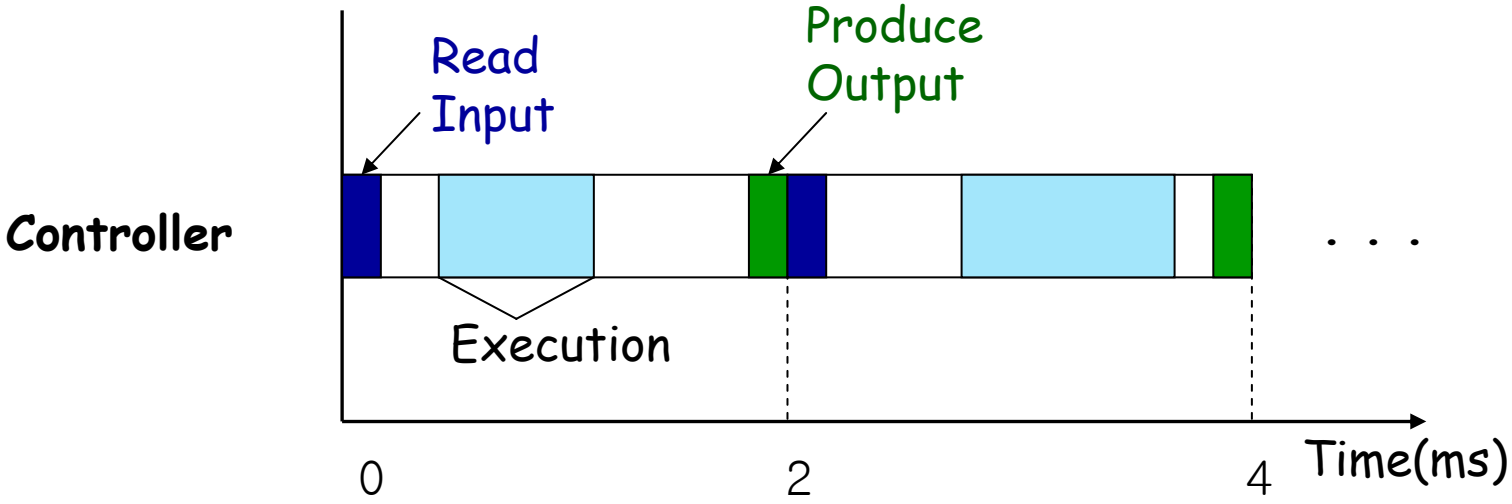
An Execution trace of a controller

Port-Based Objects(Cont'd)

- The execution time of a PBO task may vary from activation to activation.
- The time when the inputs are consumed and the outputs are produced may not be regular.
- R/W operations are atomic.
- All computations within the PBOs is based on their internal variables.
- Synchronization (monitors and locks) is managed by OS, not inside the objects.

Giotto

- **Time triggered (computation and communication).**
- **All inputs are obtained at start time, outputs are produced at stop time: communication only occurs between task executions**
- **Program model for hard real-time embedded software**
  - For every actor, the design specifies a WCET which constrains the execution time of that actor in the model



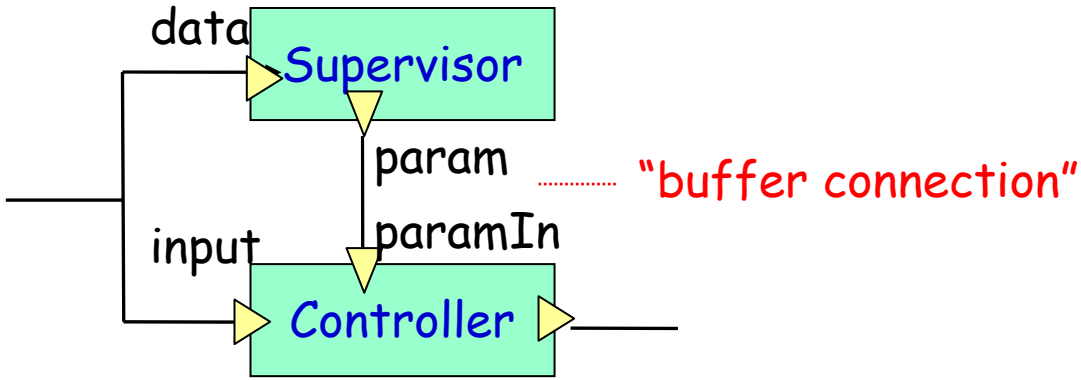
An Execution trace of a controller

Timed Multitasking

- **Event driven approach**
- **TM model is simple**
  - The activation of a task depends either on other tasks or on interrupts.
  - Deterministic timing properties : control of start time and stop time is easy.
- **Actor-oriented programming in Ptolemy II**
  - Actor represents a sequence reactions where a reaction is a piece of computation
  - Actor has state
  - Actor has ports: supports many communication machanisms
    - *Actor can only communicate with other actors and the physical world through ports: internal states is not directly accessible from outside*

Actor-Oriented Programming

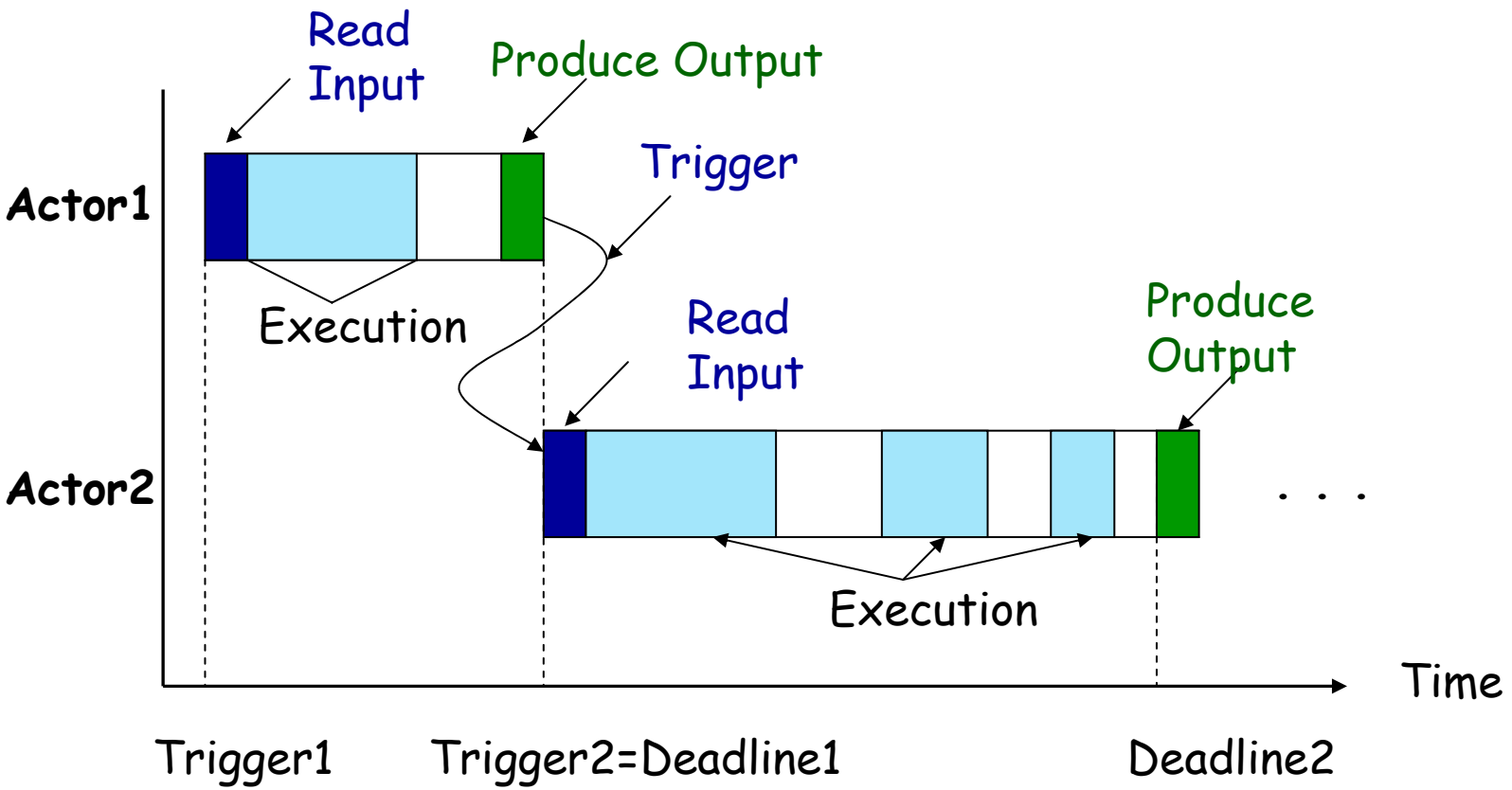
- **Unlike method calls in object-oriented models, interaction with the ports may not directly transfer the flow of control to the actor**
  - Control flow inside the actor could be independent of its data communication
  - Major distinction between Ptolemy actors and SW processes
    - *Another distinction is state hiding*



TM Programming Concepts

- **Actors + Annotations**
- **Trigger Condition**
  - "responsible" – once triggered, the actor does not need any additional data to complete its (finite) computations
  - Event semantics – every piece of data is produced and consumed exactly once: usually implemented by FIFO queues
- **Deadlines**
  - Computational results are produced if and only if the deadline is reached
  - If a deadline is not specified, the results will be produced as soon as the computation has finished
- **Execution Time**
  - May or may not be WCET:
  - "overrun handler" is triggered by the run-time system when the actor is about to miss the deadline – nonpreemptible.

TM Programming Concepts



An illustration of the execution of an actor in the TM model

Overrun handling

- **Allow the declared task execution time relaxed to better utilize the resources**
- **Needed if it is not possible to guarantee that all deadlines of all reactions can be met.**
- **How to handle overruns is application dependent**
  - In all cases, the actor should be ready for the next activation after running the overrun handler

Software Synthesis

- **Scheduling Analysis**
  - If trigger conditions are predictable, the execution time and deadlines can be fed to schedulability and priority assignments.
  - No need to directly specify the priority of actors
  - TM model is robust to scheduling policies.

- **Code Generation**
  - Provide the interface and scheduling code
  - First, distinguish ISR (Interrupt Service Routine) and task actors.
    - *ISR : a source actor (a port that transfers external events into the model.*
    - *An ISR is synthesized as an independent thread.*
    - *Tasks : triggered entirely by events produced by peer actors.*
  - Interface generation step : C file that contains the template of all methods listed and task data structure.

TM Runtime System

- **Highest priority task to dispatch trigger events and execute reactions**
- **Strictly obey the deadlines of each task if they are specified, as timer interrupts**
  - An overrun reaction can be terminated either gracefully or abruptly

TM Runtime System

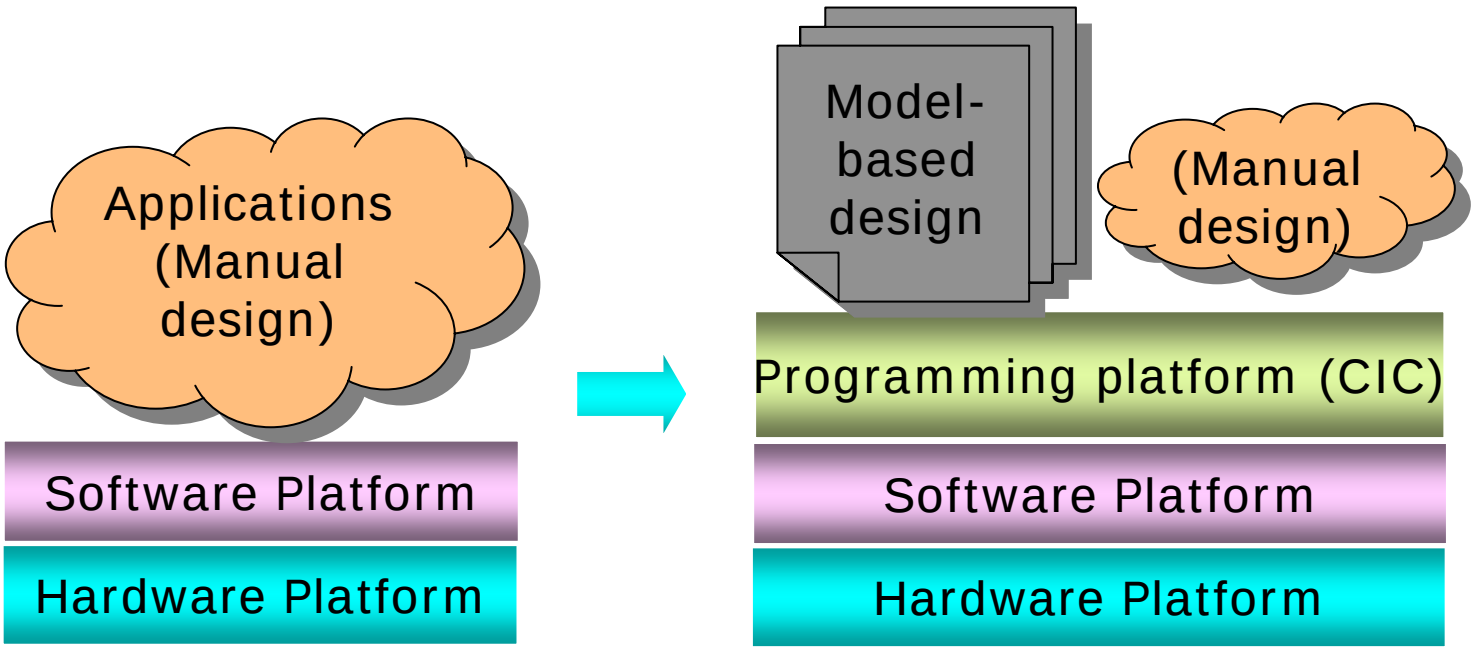
- **Highest priority task to dispatch trigger events and execute reactions**
- **Strictly obey the deadlines of each task if they are specified, as timer interrupts**
  - An overrun reaction can be terminated either gracefully or abruptly

codesign environment

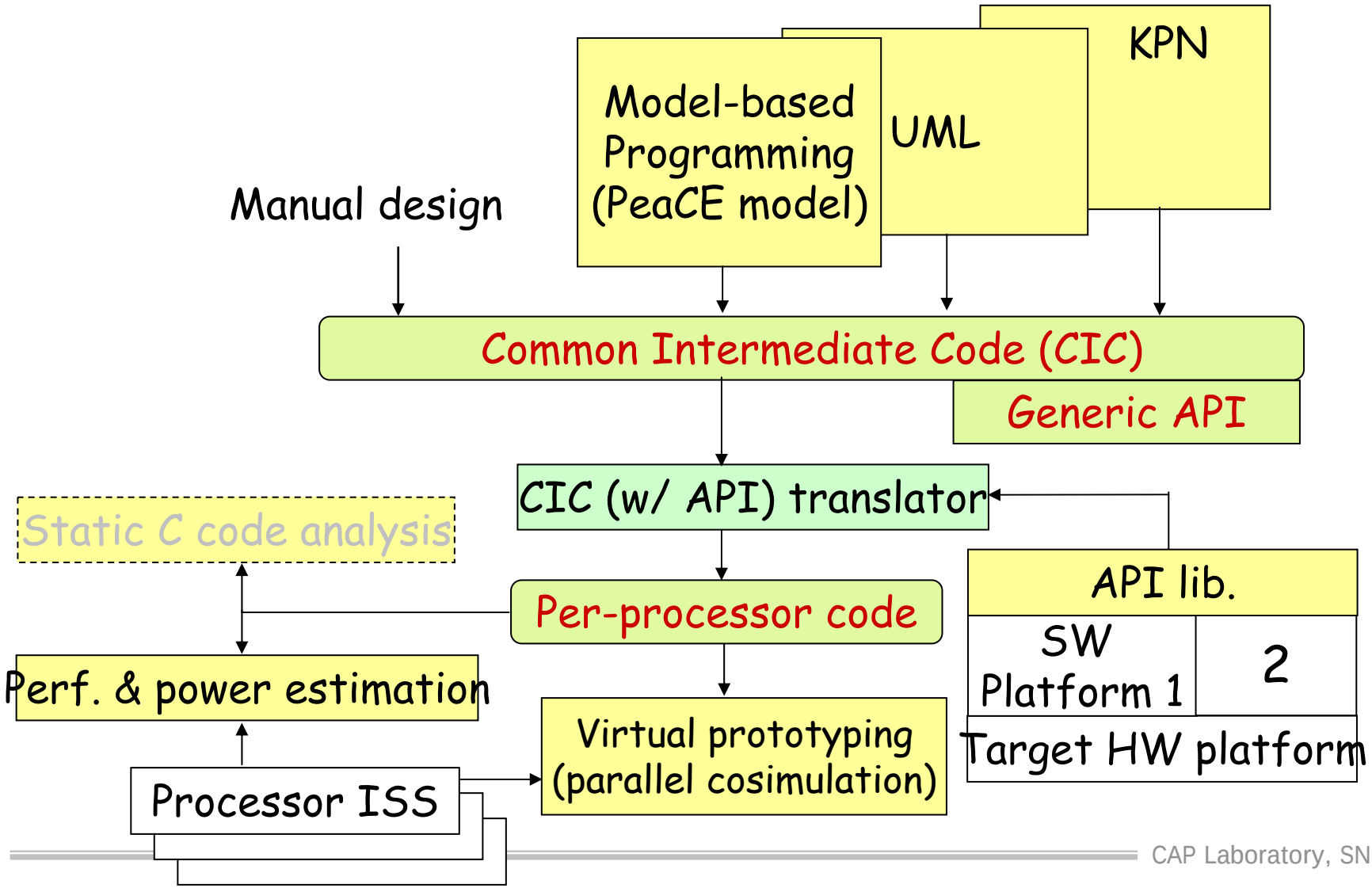
Outline

- Introduction
- Port-based Model
- TM Model
- HOPES approach

Basic Idea


Applications
(Manual
design)
Software Platform
Hardware Platform
Model-
based
design
(Manual
design)
Programming platform (CIC)
Software Platform
Hardware Platform

HOPES Proposal

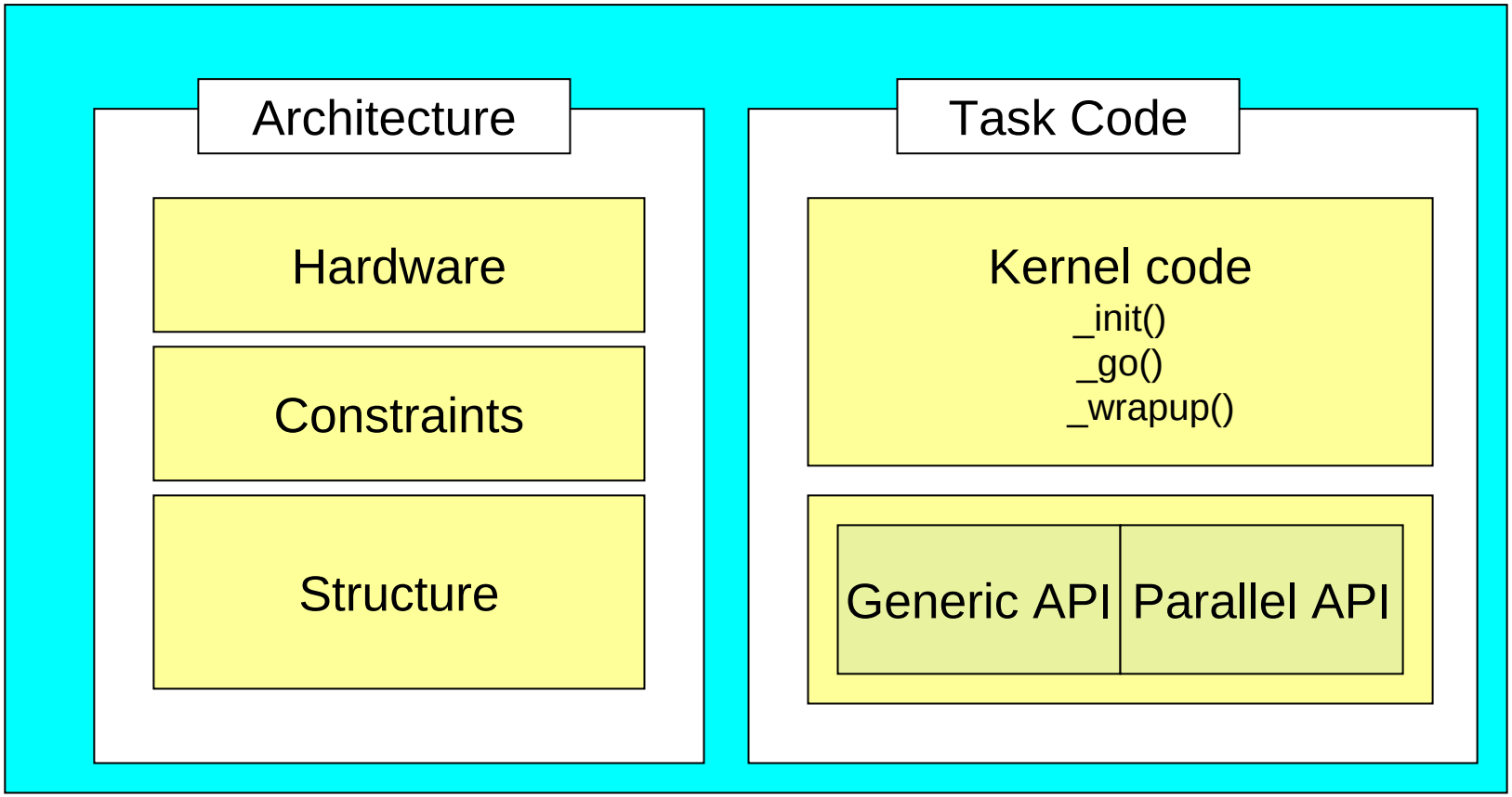


Key Techniques

- **SW design techniques**
  - Model-based specification
  - Partitioning and automatic code generation from specification
  - Parallel programming (task parallelism, data parallelism)
  - Generic APIs (independent of OS and target architecture)

- **SW verification techniques: 3-phase verification**
  - At specification level: static analysis of program specification and functional simulation
  - After code generation (optional): static analysis of C code
  - At simulation time: debugging on virtual prototyping environment

CIC Format

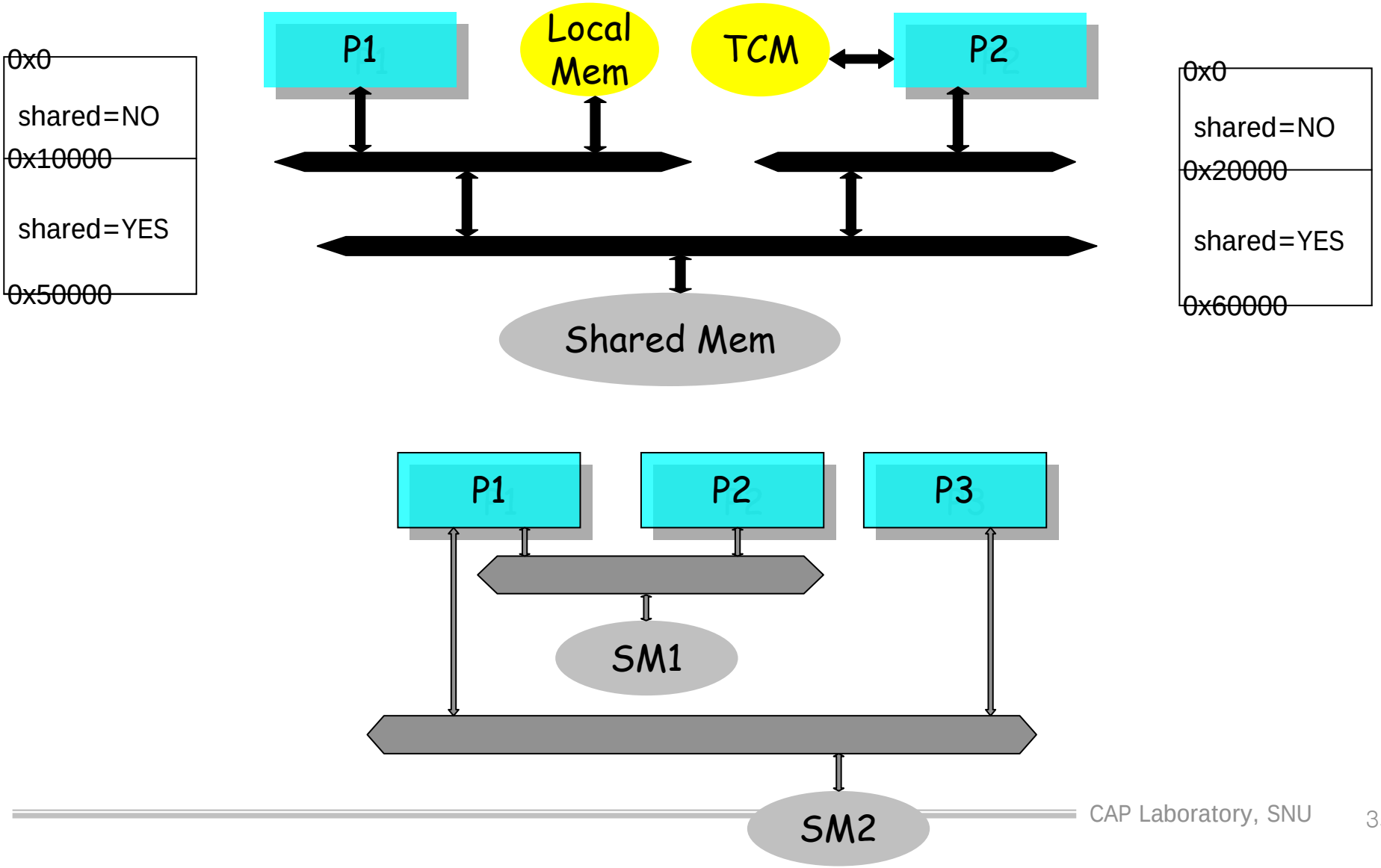

Architecture
Hardware
Constraints
Structure
Task Code
Kernel code
_init()
_go()
_wrapup()
Generic API
Parallel API

DivX player CIC xml (Hardware)

```
<?xml version="1.0" ?>
 <CIC_XML>
  <hardware>
   <processor name="arm926ej-s0">
    <index>0</index>
    <localMem name="lmap0">
      <addr>0x0</addr>
      <size>0x10000</size>  // 64KB
    </localMem>
    <sharedMem name="shmap0">
      <addr>0x10000</size>
      <size>0x40000</size>  // 256KB
      <sharedWith>1</sharedWith>
    </sharedMem>
    <OS>
      <support>TRUE</support>
    </OS>
   </processor>
```

```
   <processor name="arm926ej-s1">
    <index>0</index>
    <localMem name="lmap1">
      <addr>0x0</addr>
      <size>0x20000</size>  // 128KB
    </localMem>
    <sharedMem name="shmap1">
      <addr>0x20000</size>
      <size>0x40000</size>  // 256KB
      <sharedWith>0</sharedWith>
    </sharedMem>
    <OS>
      <support>TRUE</support>
    </OS>
   </processor>
  </hardware>
```

Hardware Architectures



DivX player CIC xml (Constraint)

```
<constraints>
 <memory>16MB</memory>
 <power>50mWatt</power>
  <mode name="default">
  <task name="AviReaderI0">
   <period>120000</period>
   <deadline>120000</deadline>
   <priority>0</priority>
    <subtask name="arm926ej-s0">
    <execTime>186</execTime>
    </subtask>
   </task>
   <task name="H263FRDivxI3">
   <period>120000</period>
   <deadline>120000</deadline>
   <priority>0</priority>
    <subtask name="arm926ej-s0">
    <execTime>5035</execTime>
    </subtask>
   </task>
   <task name="MADStreamI5">
   <period>120000</period>
   <deadline>120000</deadline>
   <priority>0</priority>
    <subtask name="arm926ej-s0">
    <execTime>13</execTime>
    </subtask>
  </task>
 </mode>
</constraints>
```

DivX player CIC xml (Structure)

```
<structure>
 <mode name="default">
  <task name="AviReaderI0">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>AviReaderI0_arm926ej_s0.cic</fileName>
   </subtask>
  </task>
  <task name="H263FRDivxI3">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>H263FRDivxI3_arm926ej_s0.cic</fileName
   </subtask>
  </task>
  <task name="MADStreamI5">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>MADStreamI5_arm926ej_s0.cic</fileName>
   </subtask>
  </task>
 </mode>
```

```
 <queue>
  <name>mq0</name>
  <src>AviReaderI0</src>
  <dst>H263FRDivxI3</dst>
  <size>30000</size>
 </queue>
 <queue>
  <name>mq1</name>
  <src>AviReaderI0</src>
  <dst>MADStreamI5</dst>
  <size>30000</size>
 </queue>
</structure>
</CIC_XML>
```

Task Code: Structure

- **Task kernel code**
  - _init(): before main loop
  - _go(): in the main loop
  - _wrapup(): after main loop

(example) h263dec.cic

// header file

// global declaration and definition

// procedure definition

h263dec_init() { … }

h263dec_go() { … }

h263dec_wrapup() { … }

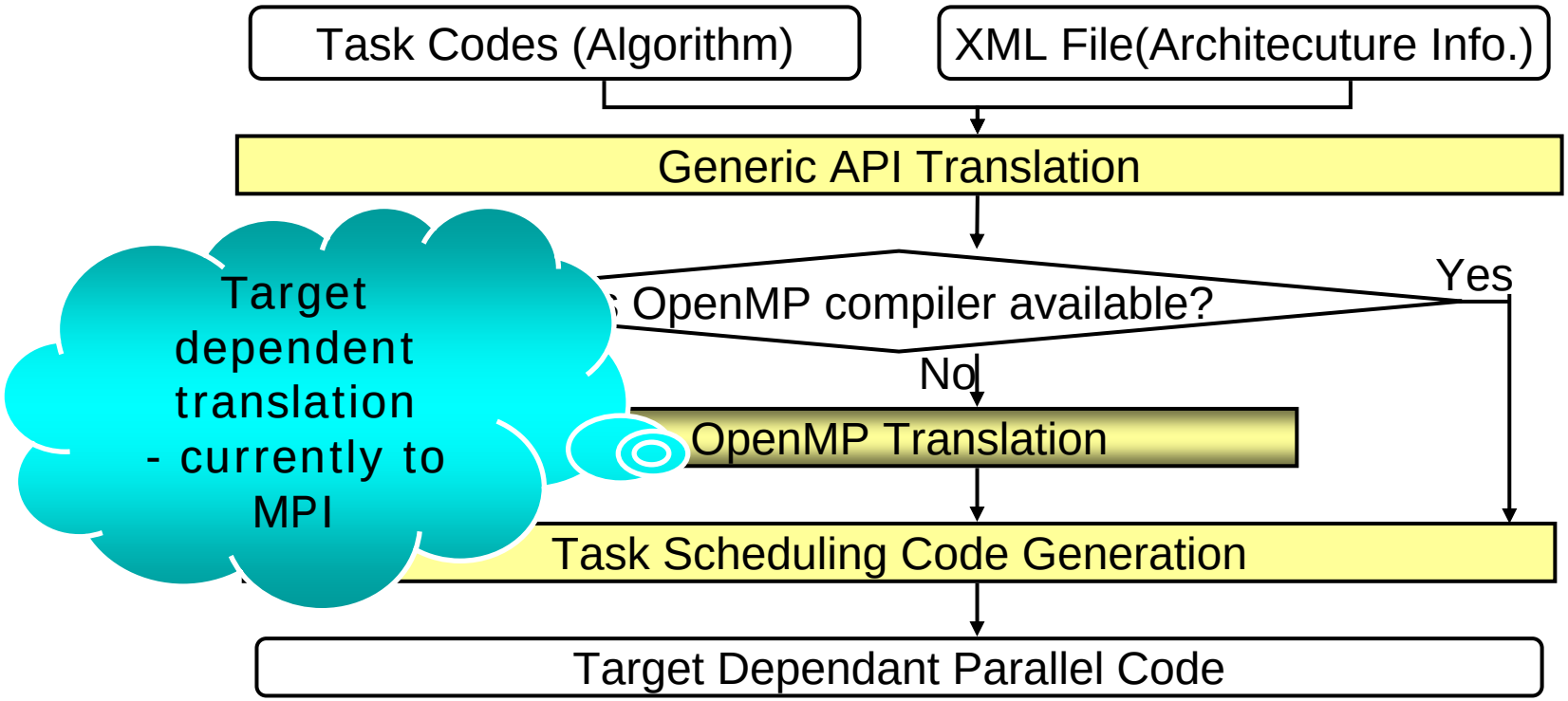
```
// Task Initialize
Task_init();

// Main Body
while(1) {
  Task_go();
}

// Finishing
Task_wrapup();
```

CIC Code Generation

- **Separate CIC task codes for partitioned tasks**
  - Explicit functional parallelism → task_name.cic
- **openMP programming for data parallelism**
- **Generic API for platform independent programming**

```
void h263decoder_go (void) {
        ...
         l = MQ_RECEIVE("mq0", (char *)(ld_106->rdbfr),
2048);
         ...
         # pragma omp parallel for
         for(i=0; i<99; i++) {
              //thread_main()
            ....
         }
          // display the decode frame
            dither(frame);
}
```

CIC Translator



Generic API

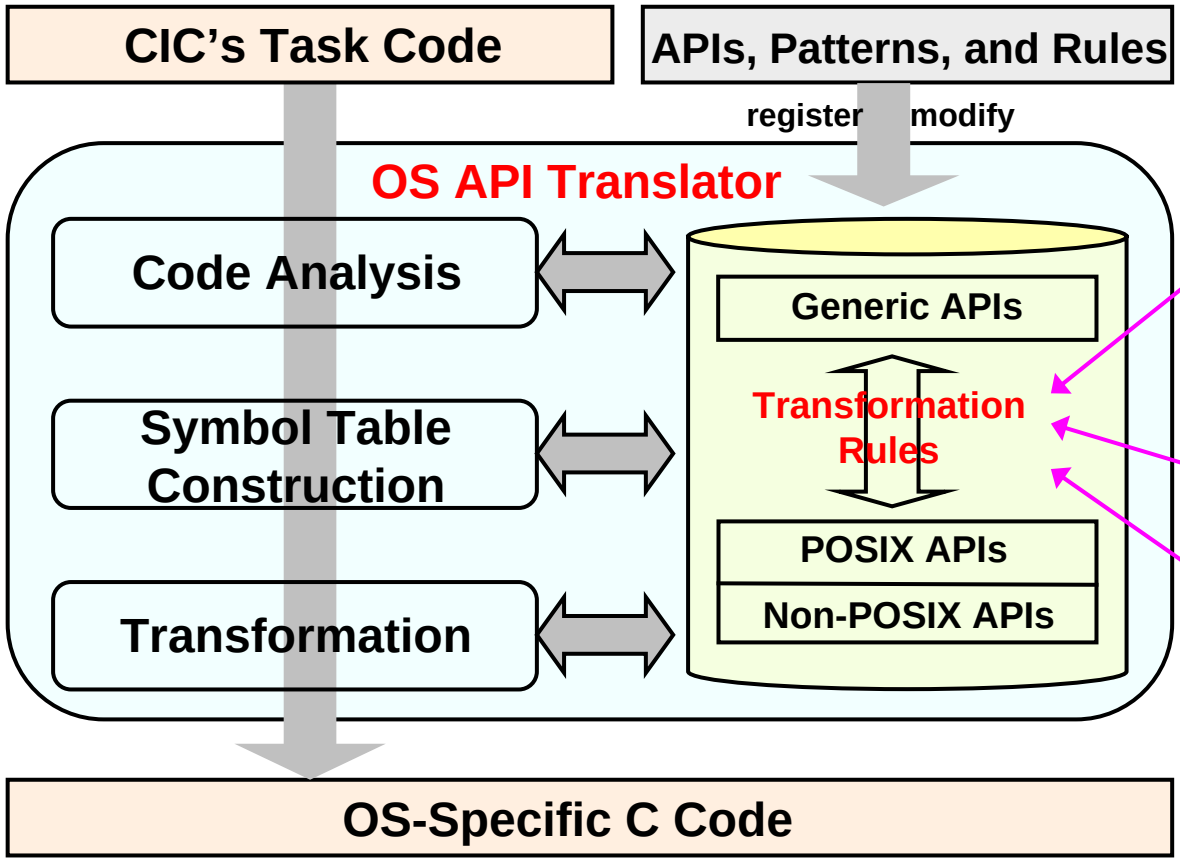
- **Generic API**
  - OS-indepednent API
  - Abstraction from IEEE POSIX 1003.1-2004
  - Selected OS services + C library functions

```
file = OPEN("input.dat", O_RDONLY);
…
READ(file, data, 100);
…
CLOSE(file);
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
…
file = open("input.dat", O_RDONLY);
…
read(file, data, 100);
…
close(file);
```

```
#include <stdio.h>
…
file = fopen("input.dat", "r");
…
fread(data, 1, 100, file);
…
fclose(file);
```

Internal Structure of API Translator



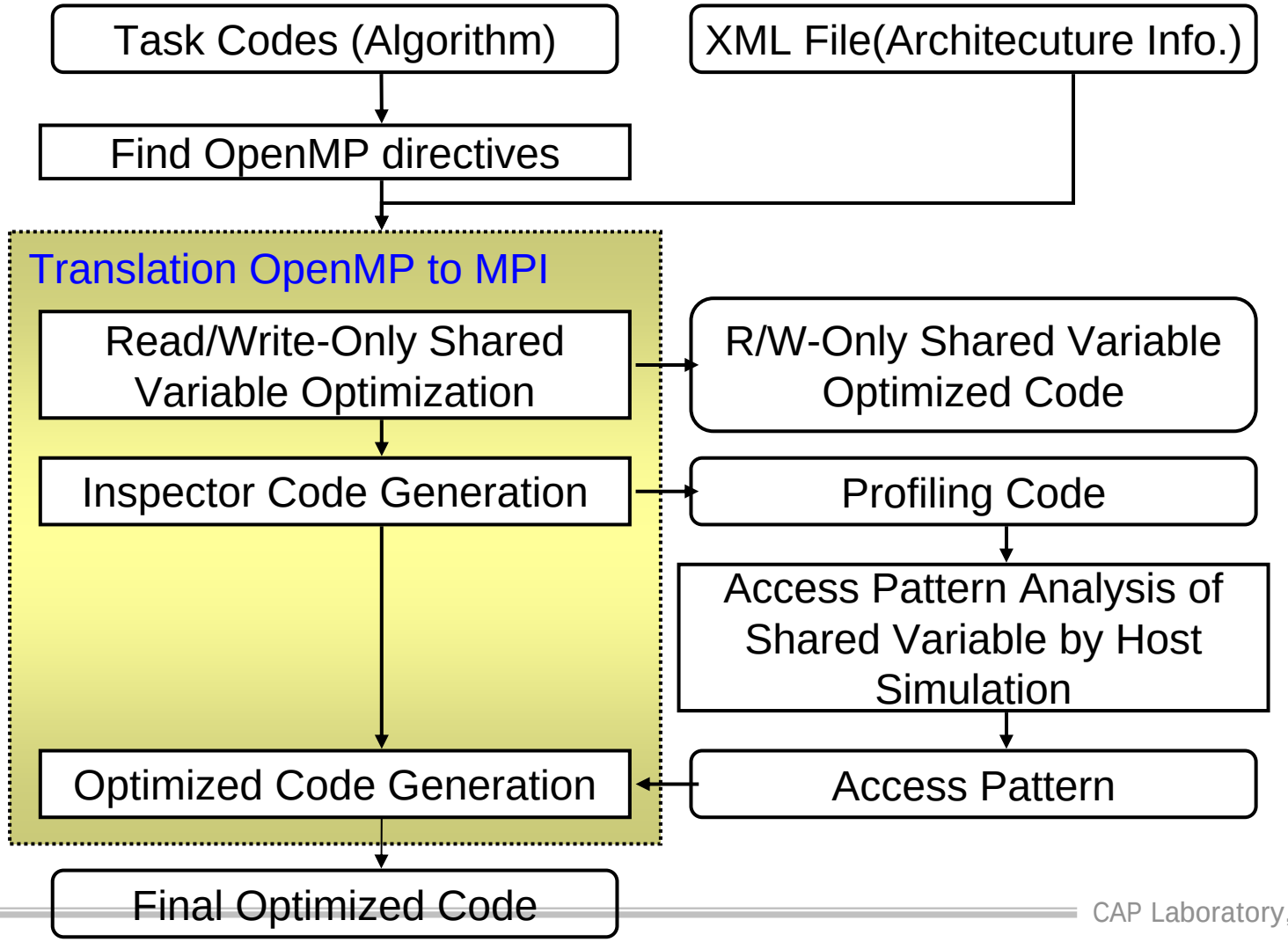
Translation pattern and rules

Pattern mapping
- 1-to-1, 1-to-m, n-to-m
- initialize/finalize
- acquire/release

parameters
- type, number, value

Variable names
- naming of additional variables

OpenMP to MPI Translator



Naive Translation

```
1  /* Simple example of an OpenMP program */
2  int main() {
3    int i;
4    int a[1000];
5    int b[1000];
6  #pragma omp parallel shared(a,b) private(i)
7  #pragma omp_hopes read_only(a) write_only(b)
8   {
9  #pragma omp for
10    for(i = 0; i < 1000; i++)
11      b[i] = a[i] + i;
12   }
13 }
```

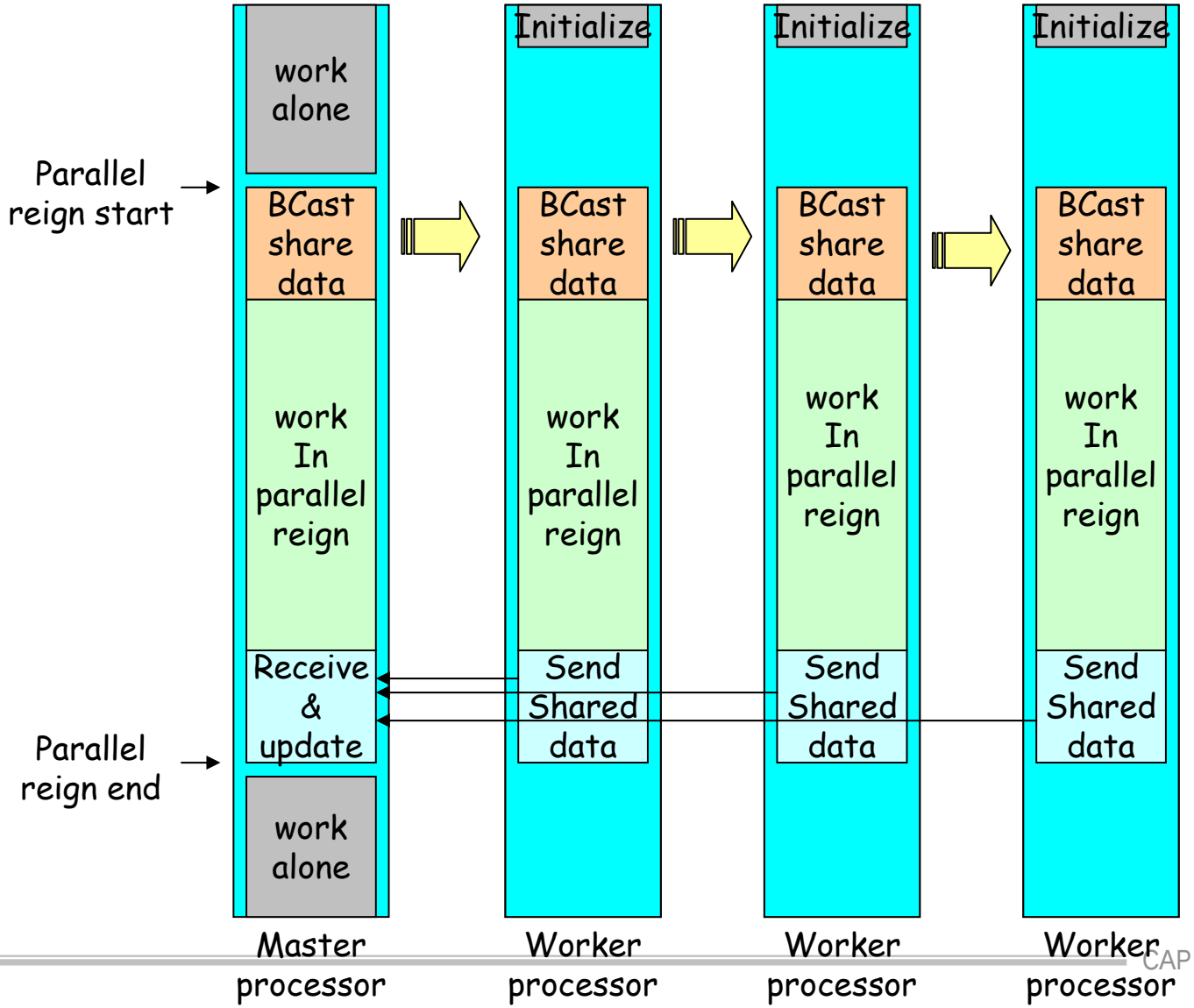
master

```
1  /* Simple example of an OpenMP program */
2  int main() {
3    int i;
4    int a[1000], b[1000];
5    {
6      int rank,numtasks;
7      int argv = 0;
8      MPI_Init(argv);
9      …
10     MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
11     MPI_Bcast(&b,1000,MPI_INT,0,MPI_COMM_WORLD);
12     MPI_Barrier(MPI_COMM_WORLD);
13     {
14       …
15       int  temp_a[1000];
16       int  temp_b[1000];
17       int  other_temp_a[1000];
18       int  other_temp_b[1000];
19       1) save initial data of shared variables in temporary variables
20       for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
21          (*((b)+(i)))=((*((a)+(i)))+(i)));
22       2) update result data into each shared variable
23       for(_temp_l=0;_temp_l<(2-1); _temp_l++){
24         MPI_Recv(&other_temp_a,1000,MPI_INT,_temp_l+1,
               0,MPI_COMM_WORLD,&stat);
25         MPI_Recv(&other_temp_b,1000,MPI_INT,_temp_l+1,
               1,MPI_COMM_WORLD,&stat);
26       3) compare received data with initial data, and update modified value
27       }
28       MPI_Barrier(MPI_COMM_WORLD);
29       MPI_Finalize();
30     }
31 }
```

slave

```
1  void worker_1(int argv)  {
2    int a[1000], b[1000];
3    int i;
4    …
5    MPI_Init(argv);
6    ..
7    MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
8    MPI_Bcast(&b,1000,MPI_INT,0,MPI_COMM_WORLD);
9    MPI_Barrier(MPI_COMM_WORLD);
10   {
11       for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*
12           (rank+1))/2)+0;(i)++)  (*((b)+(i)))=((*((a)+(i)))+(i)));
13     MPI_Send(&a,1000,MPI_INT,0,0,MPI_COMM_WORLD);
14     MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
15   }
16   MPI_Barrier(MPI_COMM_WORLD);
17 }
```

Worker model of execution



Read/write-only Variable Optimization

Master

```
1-9     …
10-11   MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
12      MPI_Barrier(MPI_COMM_WORLD);
13      {
14        …
15-16     int  temp_b[1000];
17-18     int  other_temp_b[1000];
19        1) save only b's initial value in temporary variables
20        for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)    (*((b)+(i)))=((*((a)+(i)))+(i)));
22        2) update result of b
23        for(_temp_l=0;_temp_l<(2-1); _temp_l++){
24-25         MPI_Recv(&other_temp_b,1000,MPI_INT,_temp_l+1, 1,MPI_COMM_WORLD,&stat);
26        3) compare received data with initial data and update modified one
27      }
28-31   …
```

Worker

```
1-6     …
7-8     MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
9       MPI_Barrier(MPI_COMM_WORLD);
10      {
11        for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
12          (*((b)+(i)))=((*((a)+(i)))+(i)));
13-14     MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
15      }
16-17   …
```

Inspector Code Optimization(1)

- **Inspector code generation**

Master

```
1-9     …
        MPI_Barrier(MPI_COMM_WORLD);
        {
            ...
            master_inspector((void*)a,2,1,sizeof(int ),0);
            master_inspector((void*)b,2,2,sizeof(int ),0);
        }
12      MPI_Barrier(MPI_COMM_WORLD);
20       for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*
21            (rank+1))/2)+0;(i)++)  (*((b)+(i)))=((*((a)+(i)))+(i)));
        master_inspector((void*)a,2,1,sizeof(int ),1);
        master_inspector((void*)b,2,2,sizeof(int ),1);
28-31   …
```

Worker

```
1-6     …
9        MPI_Barrier(MPI_COMM_WORLD);
        {
            ...
            int t_a[1000], t_b[1000];
            initialize_table(t_a,1000);
            initialize_table(t_b,1000);
           for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
           {
              inspector(t_a,(void*)a,(void*)(a+i),sizeof(int ));
              inspector(t_b,(void*)b,(void*)(b+i),sizeof(int ));
           }
           recv_initial_data(t_a,(void*)a,1000,sizeof(int ),1,rank,0);
           recv_initial_data(t_b,(void*)b,1000,sizeof(int ),2,rank,0);
           MPI_Barrier(MPI_COMM_WORLD);
11         for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
12           (*((b)+(i)))=((*((a)+(i)))+(i)));
           recv_initial_data(t_a,(void*)a,1000,sizeof(int ),1,rank,1);
           recv_initial_data(t_b,(void*)b,1000,sizeof(int ),2,rank,1);
        }
16-17 …
```

Inspector Code Optimization(2)

- **Optimized Code**

Master

```
1-9     …
12      MPI_Barrier(MPI_COMM_WORLD);
13      {
14      ...
        MPI_Send(&a[500],500,MPI_INT,1,0,MPI_COMM_WORLD);
20      for((i)=(((1000-0)*(rank))/4)+0;(i)<(((1000-0)*(rank+1))/4)+0;(i)++)
21          (*((b)+(i)))=((*((a)+(i)))+(i)));
23-26   MPI_Recv(&a[500],500,MPI_INT,1,0,MPI_COMM_WORLD,&stat);
27      }
28-31   …
```

Worker

```
1-6     …
9       MPI_Barrier(MPI_COMM_WORLD);
10      {
            ...
            MPI_Recv(&a[500],500,MPI_INT,0,0,MPI_COMM_WORLD,&stat);
11          for((i)=(((1000-0)*(rank))/4)+0;(i)<(((1000-0)*(rank+1))/4)+0;(i)++)
12              (*((b)+(i)))=((*((a)+(i)))+(i)));
13-14       MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
15      }
16-17   …
```

Scheduling Code (1): w/o OS

```
1   typedef struct {
2      void (*init)();
3      int (*go());
4      void (*wrapup)();
5      int period;
6      int time_count;  /* this variable indicates the next invocation time of a task */
7   } task;
8   int taskNum = 2;
9   task taskInfo[] = { {task1_init, task1_go, task1_wrapup, 100, 0}
10     , {task2_init, task2_go, task2_wrapup, 200, 0}};
11  …
12  int main() {
13     init();           /* {task_name}_init() functions of all tasks are called */
14     scheduler(); /* scheduler code */
15     wrapup();     /* {task_name}_wrapup() functions of all tasks are called */
16     return 0;
17  }
```

codesign environment

Scheduling Code (2): w/o OS

```
1   void scheduler() {
2     while(all_task_done()==FALSE) {
3       int taskId = get_next_task();
4       if (taskInfo[taskId]->go()==0) {
5         taskInfo[taskId]->timeCount += taskInfo[taskId]->period;
6       }
7     }
8   }
9   /* return the number of task id that should be executed */
10  int get_next_task() {
11    for all task in this processor
12      find the tasks that has the smallest value of time_count variable
13    if (the number of found tasks > 1) {
14      for found tasks that has the smallest value of time_count variable
15        select the task that is not executed for the longest time in found tasks
16    }
17    return task_id;
18  }
```

Scheduling Code (3): w/ OS

```
1   void *thread_task_0_func (void *argv) {
2     …
3     task_0_go();
4     get_time(&time);
5     sleep(task_0->next_period – time);    //sleep during remained time
6     …
7   }
8   int main() {
9     …
10    pthread_t thread_task_0;
11    sched_param thread_task_0_param;
12    …
13    thread_task_0_param.sched_priority = 0;
14    pthread_attr_setschedparam (…, &thread_task_0_param);
15    …
16    task_init();   /* In this function, {task_name}_init() of each is called  */
17    pthread_create (&thread_task_0, &thread_task_0_attr, thread_task_0_func, NULL);
18    …
19    task_wrapup();   /* In this function, {task_name}_wrapup() of each task is called */
20  }
```

Summary

- **Embedded SW needs to consider**
  - Real-time constraints
  - Resource constraints
  - Parallel programming
- **Emerging embedded SW development methodologies**
  - Actor-based modeling
- **HOPES is a newly launched project to make a embedded software development environment for MPSoCs**
  - Support of diverse models
  - Target independent environment + target specific libraries
  - 3-phase verification: model-level, C code static analysis, run-time simulation
  - Integration of software modules at various stages
  - http://peace.snu.ac.kr/hopes