GAPH
Grupo de Apoio ao Projeto de Hardware

HeMPS Platform

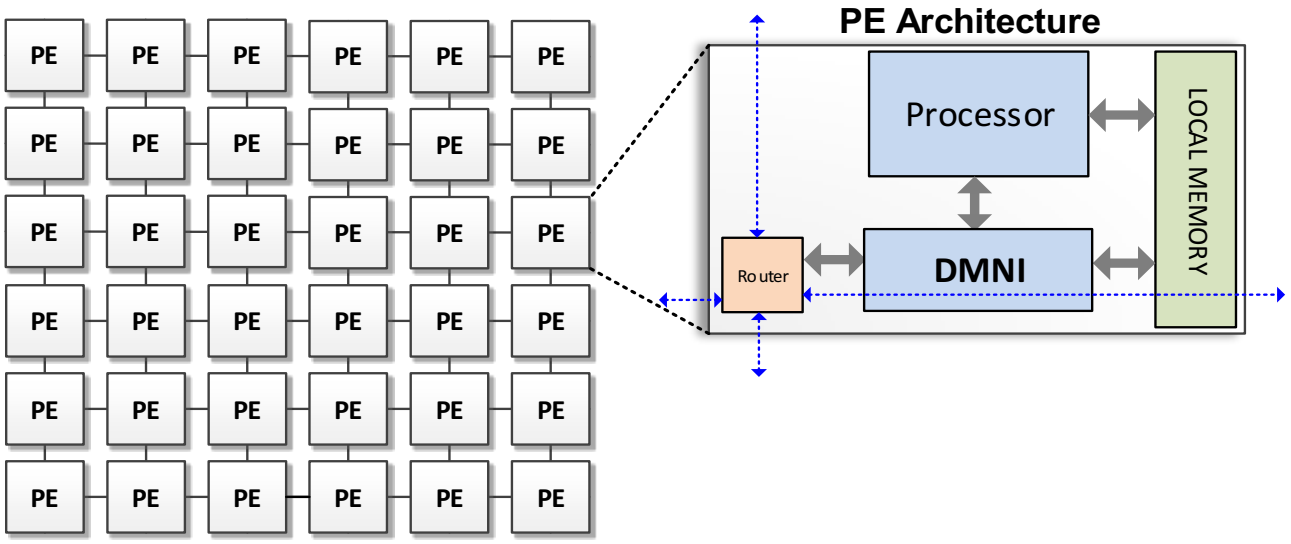
v7.3

Marcelo Ruaro, Eduardo Wachter, Guilherme Madalozzo, Guilherme Castilhos, André del Mestre

Fernando G. Moraes

PUCRS University, Computer Science Department, Porto Alegre, Brazil

Platform Overview



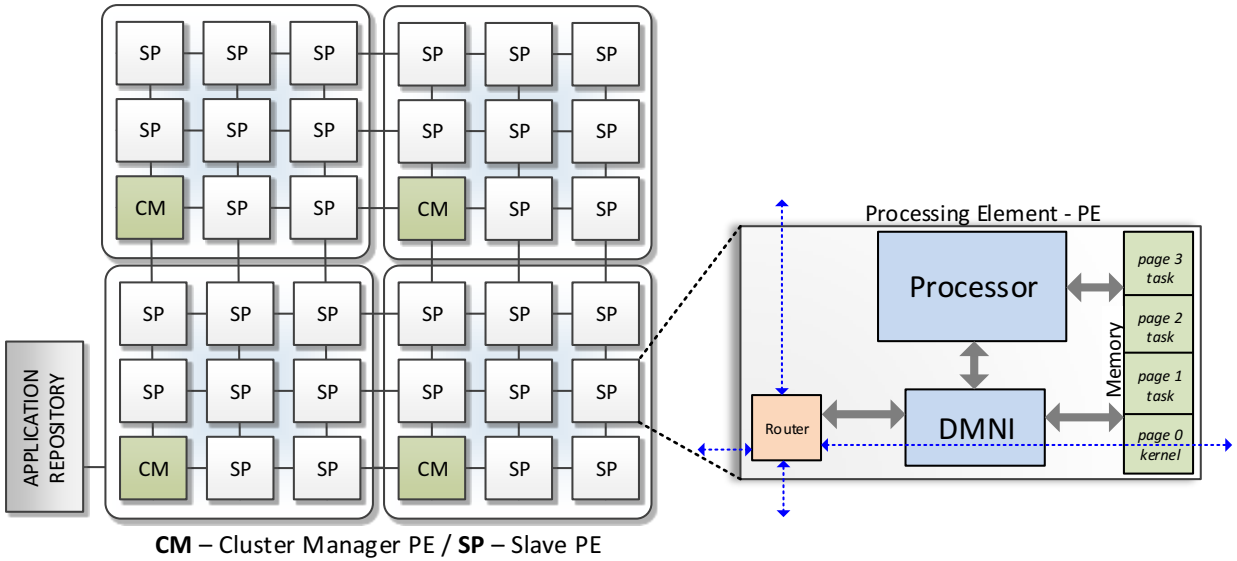
6x6 MPSoC instance

Homogeneous MPSoC

- Each PE has the same architecture

PE is composed of one processor, local memory, DMNI, and router

Platform Organization



CM – Cluster Manager PE / **SP** – Slave PE

Cluster-based organization

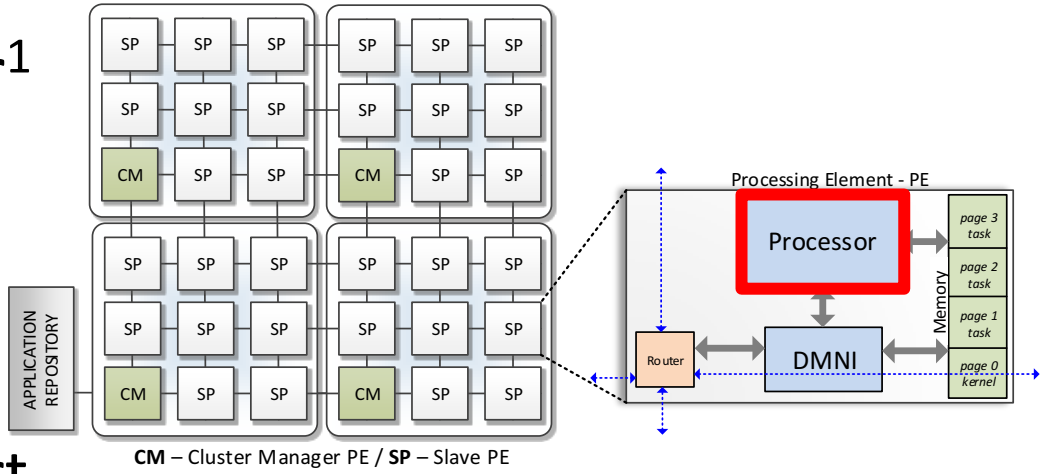
- Provides **scalability of management** and **traffic isolation**
- Reclustering is allowed
- Each cluster is managed by a **cluster manager (CM)**
- One CM is responsible for access a external repository containing the application task code

Architectural Features

Processor

Plama v2 microprocessor[1]

- 32 bits RISC
- 3-stage pipeline
- MIPS I ISA
- Add. pagination support
- ~~UART~~
- Memory mapped registers
- *syscall*

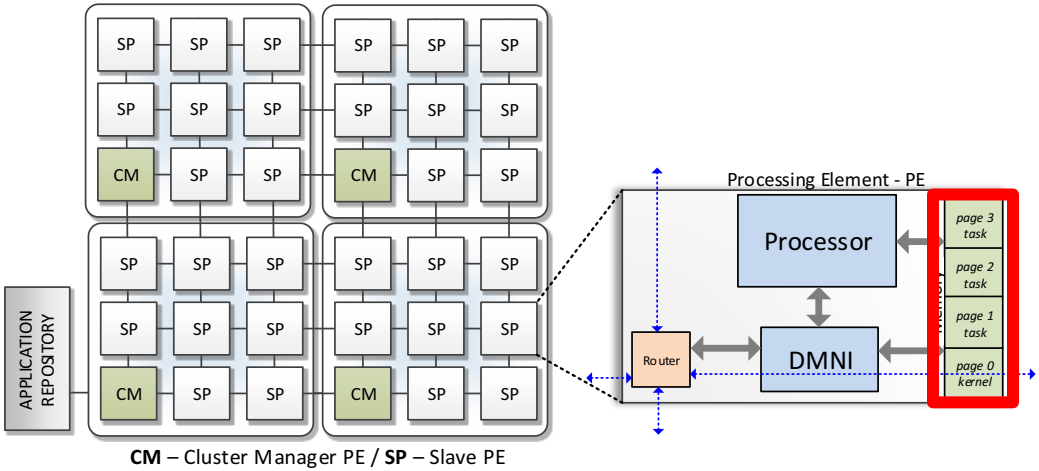


1. http://opencores.org/project,plasma

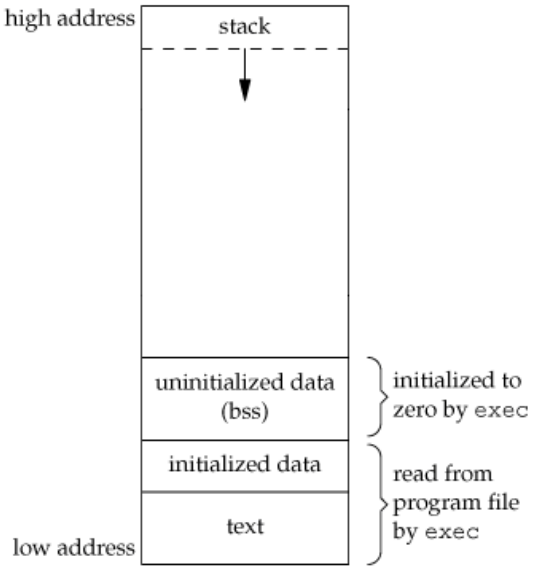
Local Memory

Scratchpad memory

- RAM
- Dual port
- Size is parameterizable
- Pages are logically managed

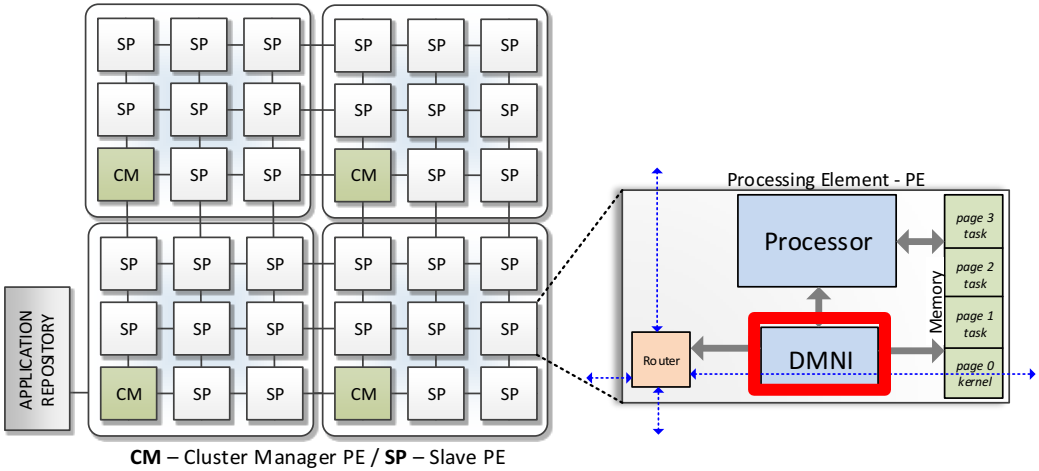


The memory implement a **true dual-port interface** enabling **simultaneous access of processor and DMNI**



DMNI

Direct Memory Network Interface[2]



The DMNI implements a **direct interface between the local memory and the NoC**.

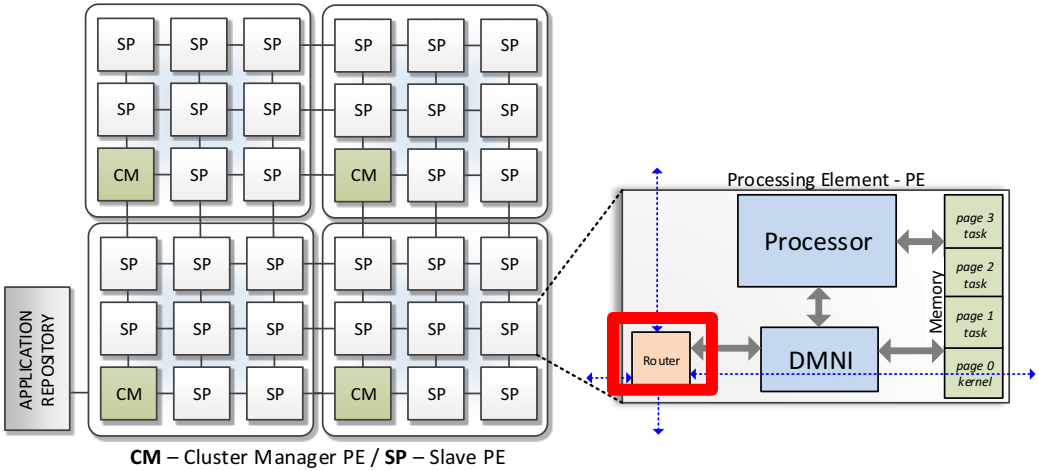
It is an approach specialized to design of NoC-based MPSoC systems

2. **DMNI: A Specialized Network Interface for NoC-based MPSoCs.** In: ISCAS, 2016.

Router

Hermes NoC[3]

- XY addressing
- XY and WF routing
- Packet Switching
- Wormhole with credit-based flow control
- Takes 5 clock cycles to arbitrage and routing a packet

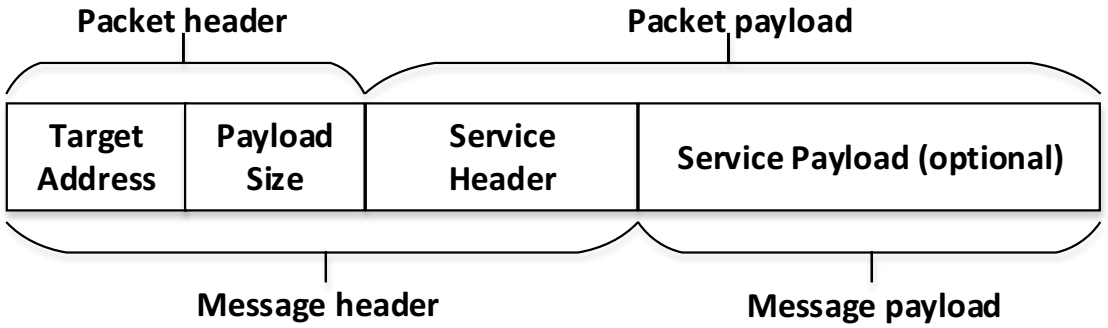


HeMPS v7.3 uses the simplest Hermes NoC implementation. There are several others Hermes derivations

- Asynchronous
- Virtual-channel
- Frequency Scaling
- Circuit-Switching
- Multicast, …

3. **HERMES: an infrastructure for low area overhead packet-switching networks on chip**. In Jornal of Integration on VLSI, 2004

NoC packet and message structure



From the NoC point of view, the packet has a **header** and a **payload**

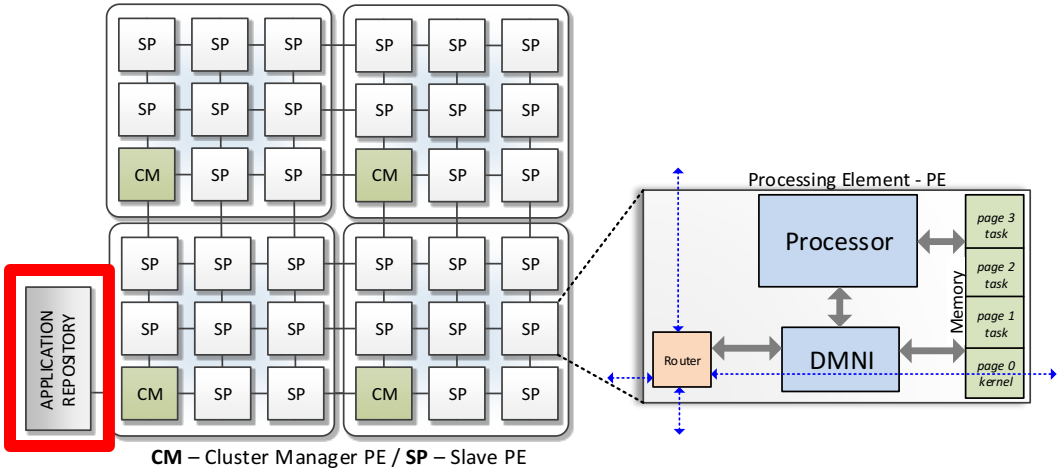
From a task point of view a message contains

- **Message header**
  - Encapsulates the packet and service header
- **Message payload**
  - Optional field. It may contain for example user data or an object code of a task

Application Repository

An external memory (off-chip)

Stores the application description and its task object code



Logical Features

Logical Features

Logical Features are implemented by software components

- **µkernels**
  - Slave
  - Manager
- **User's tasks**

Logical Features:

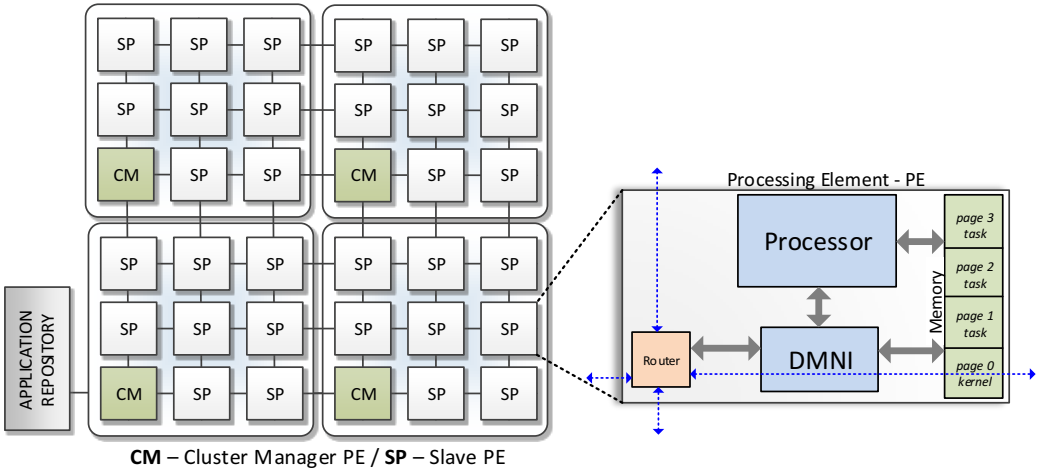
- **System Management**
- **User's Application Execution**

Logical Features

System Management

Cluster-based Management

Means that the system is logically divided into groups of processors managed by one **Cluster Manager (CM)**

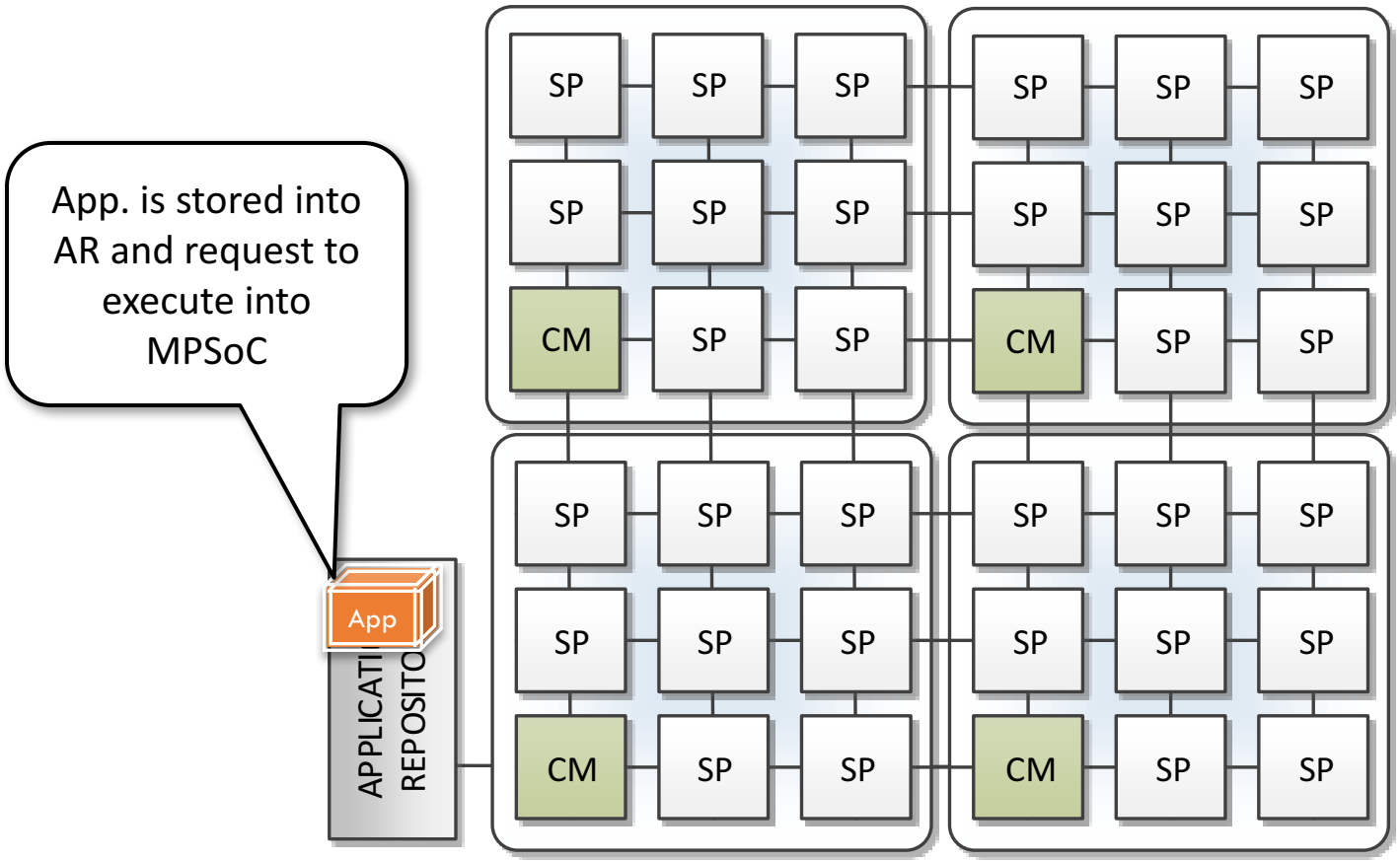


CM is a PE that runs the **manager µkernel**

Performs management functions

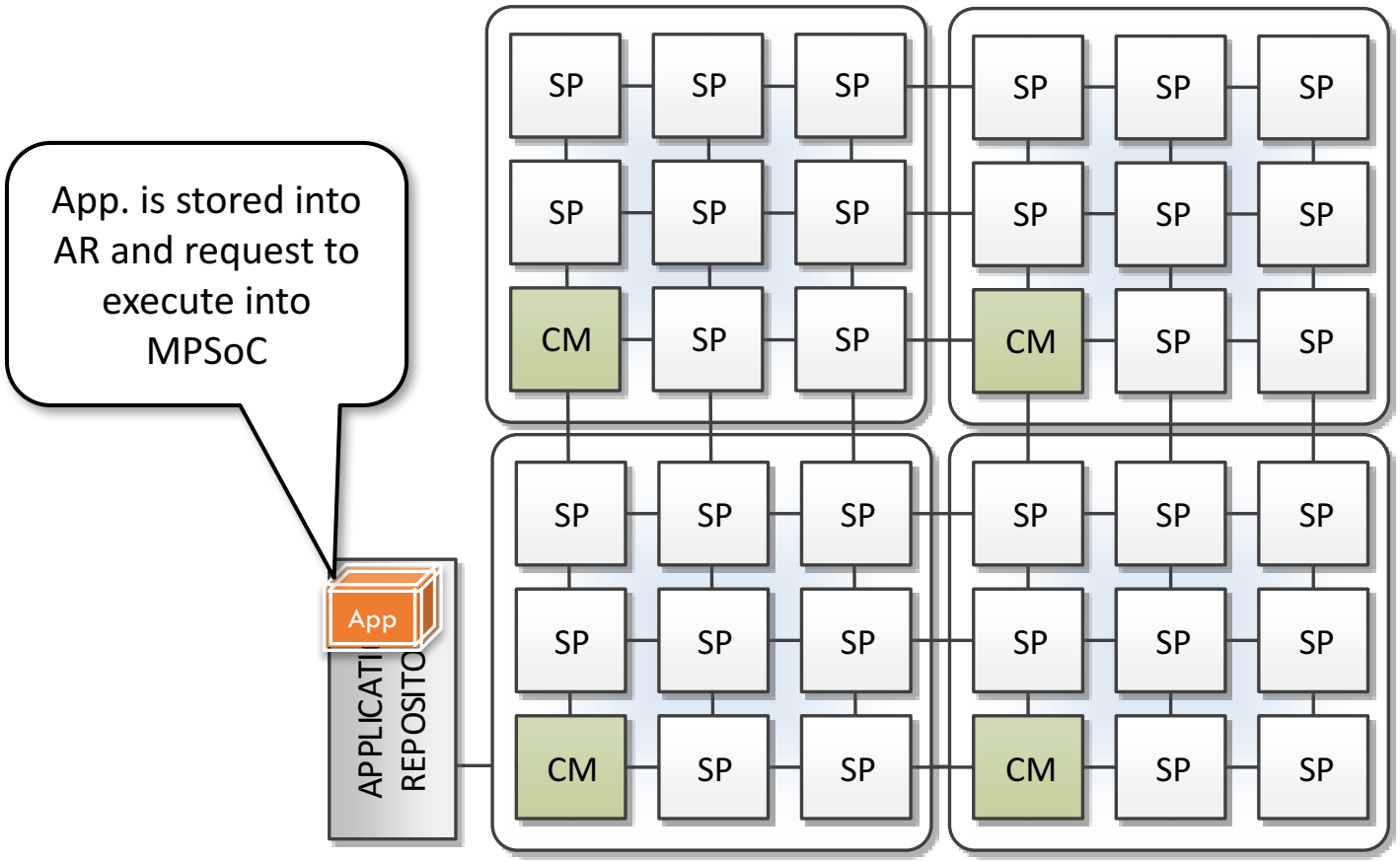
- Task mapping
- Task migration
- Reclustering

Task Mapping



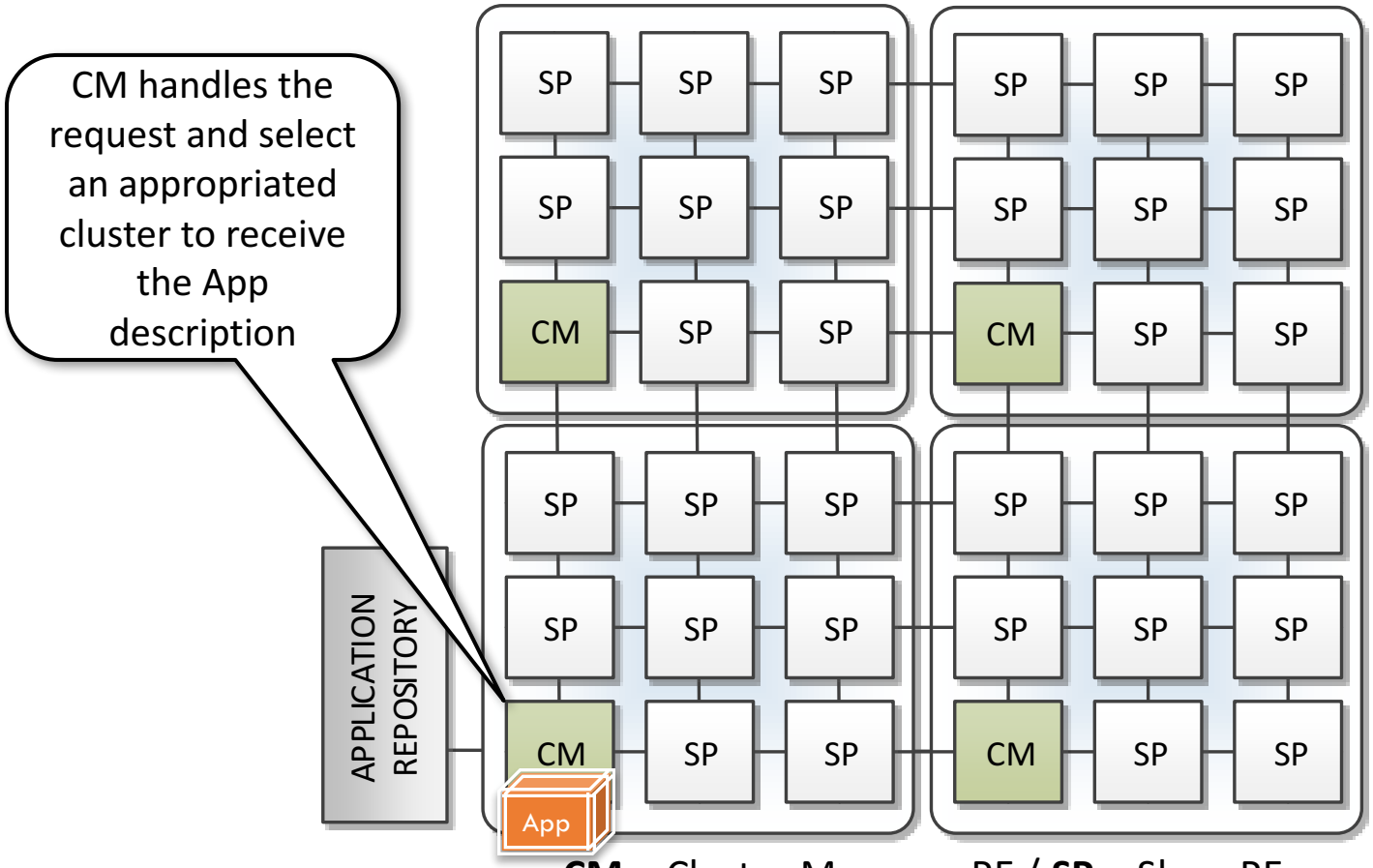
CM – Cluster Manager PE / **SP** – Slave PE

Task Mapping



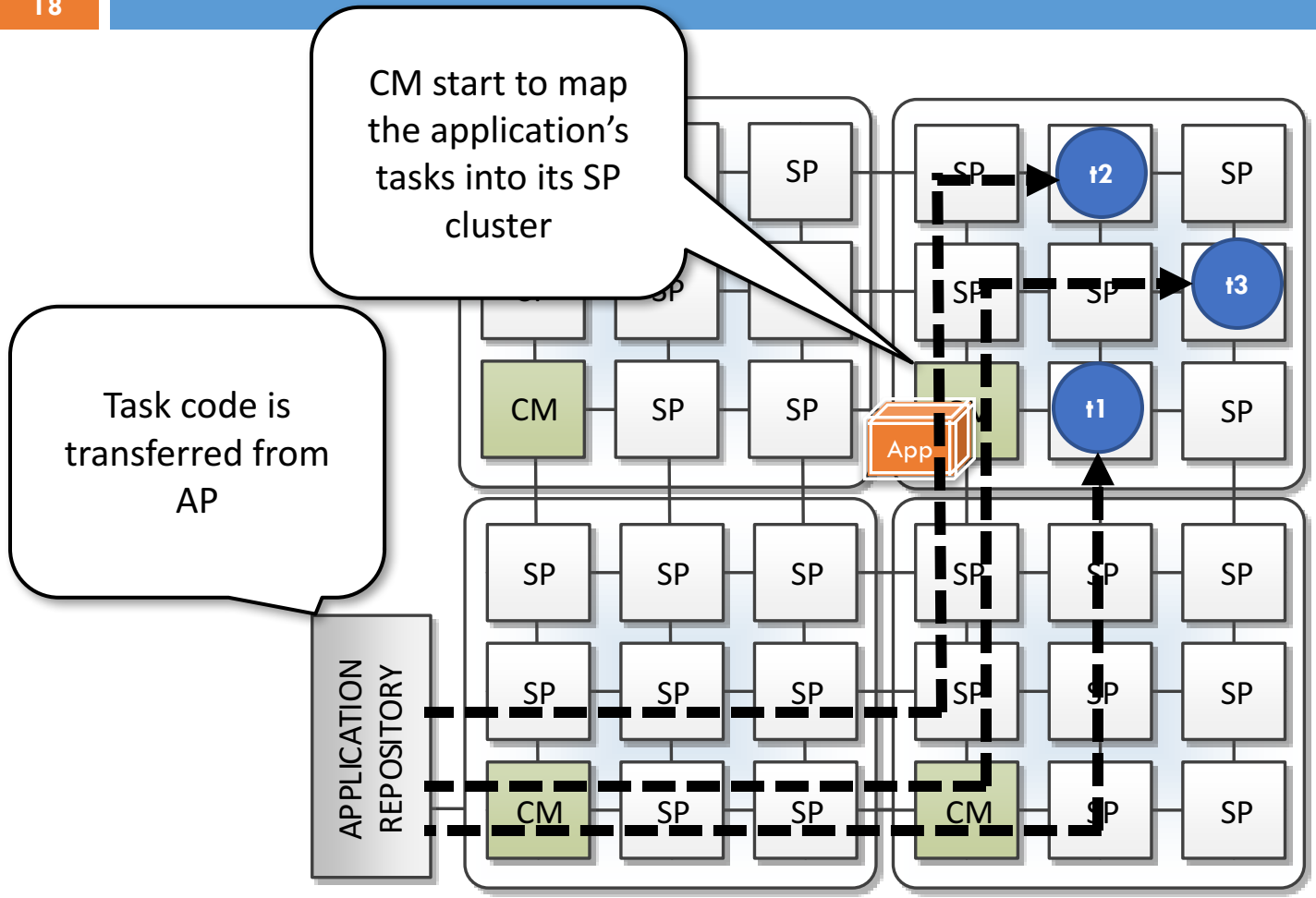
CM – Cluster Manager PE / **SP** – Slave PE

Task Mapping



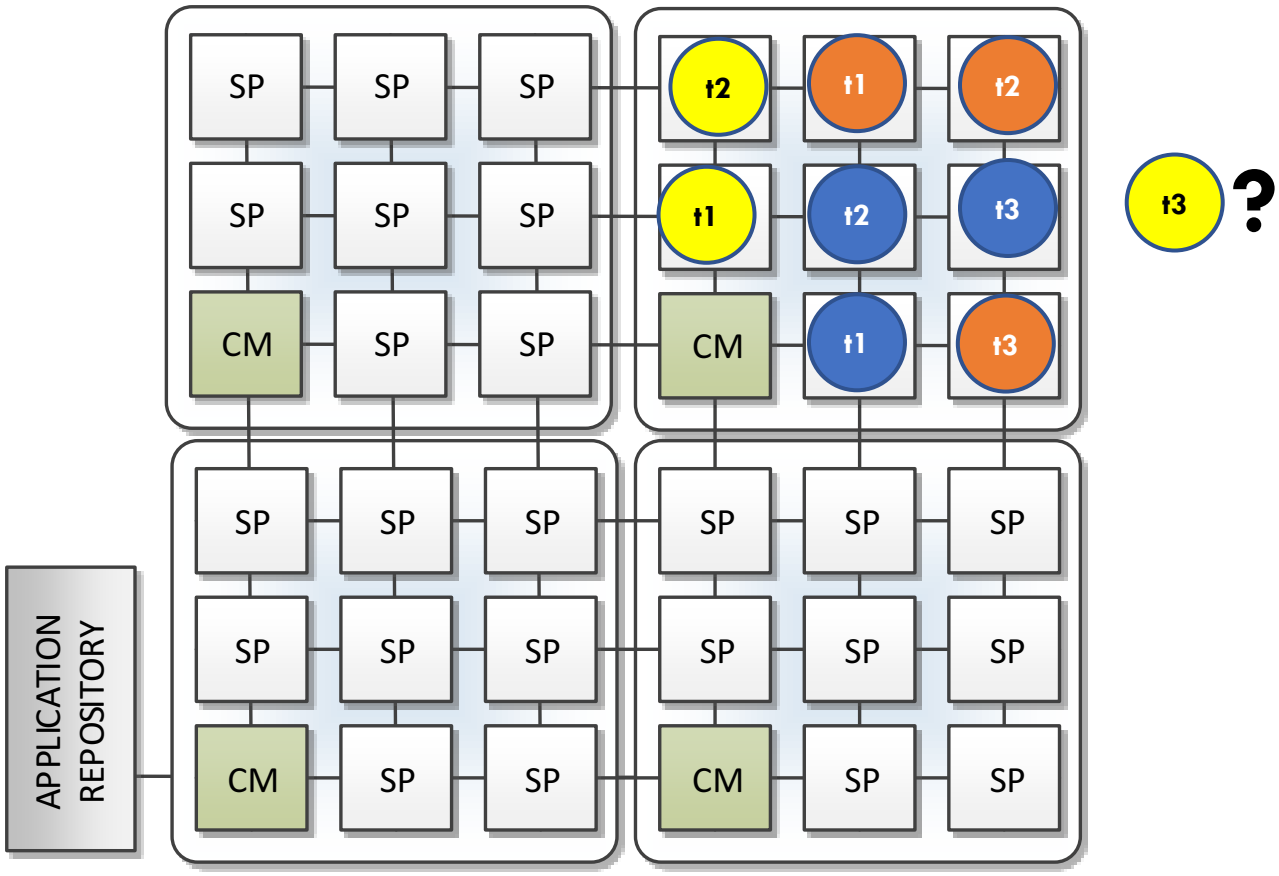
CM – Cluster Manager PE / **SP** – Slave PE

Task Mapping



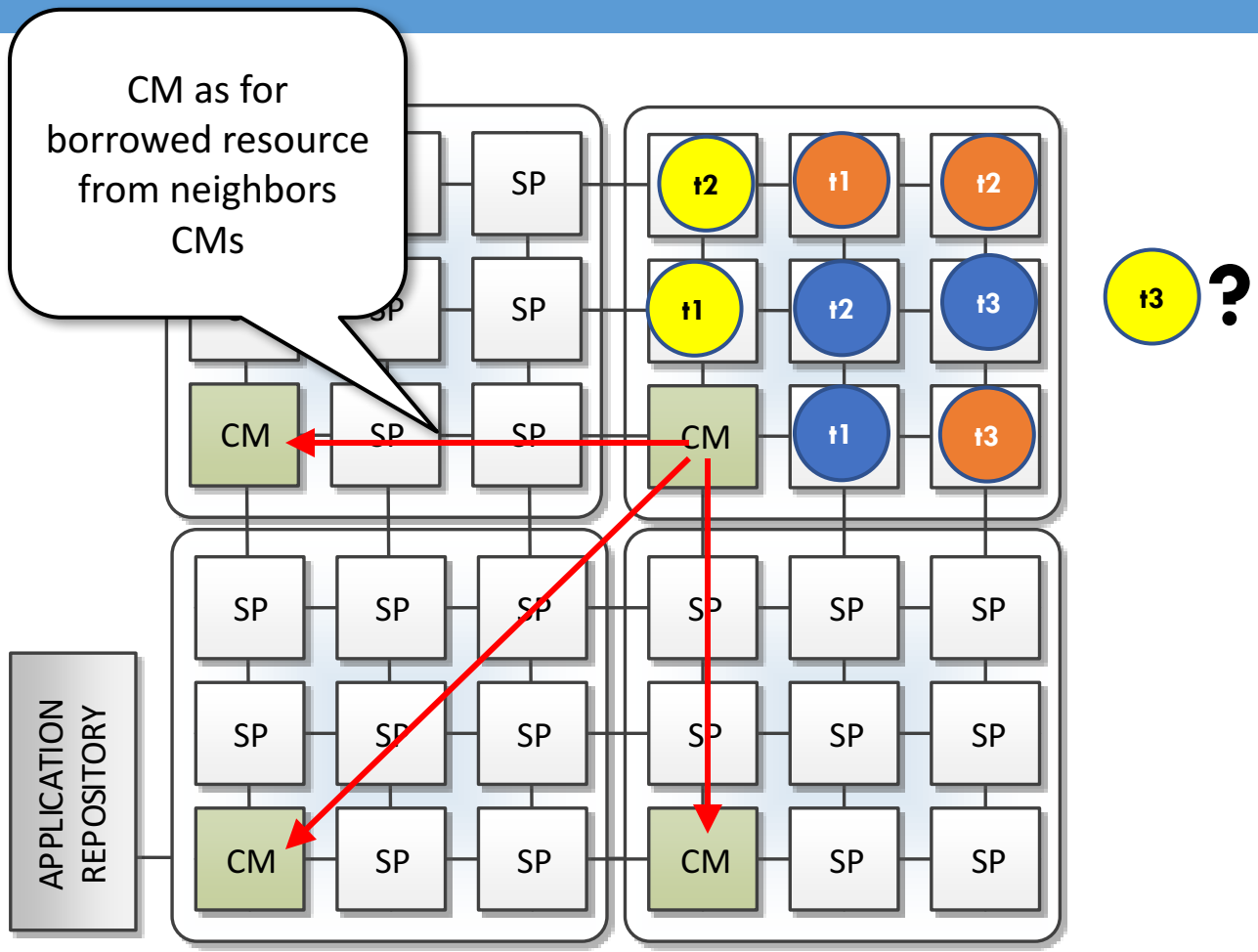
CM – Cluster Manager PE / **SP** – Slave PE

Reclustering



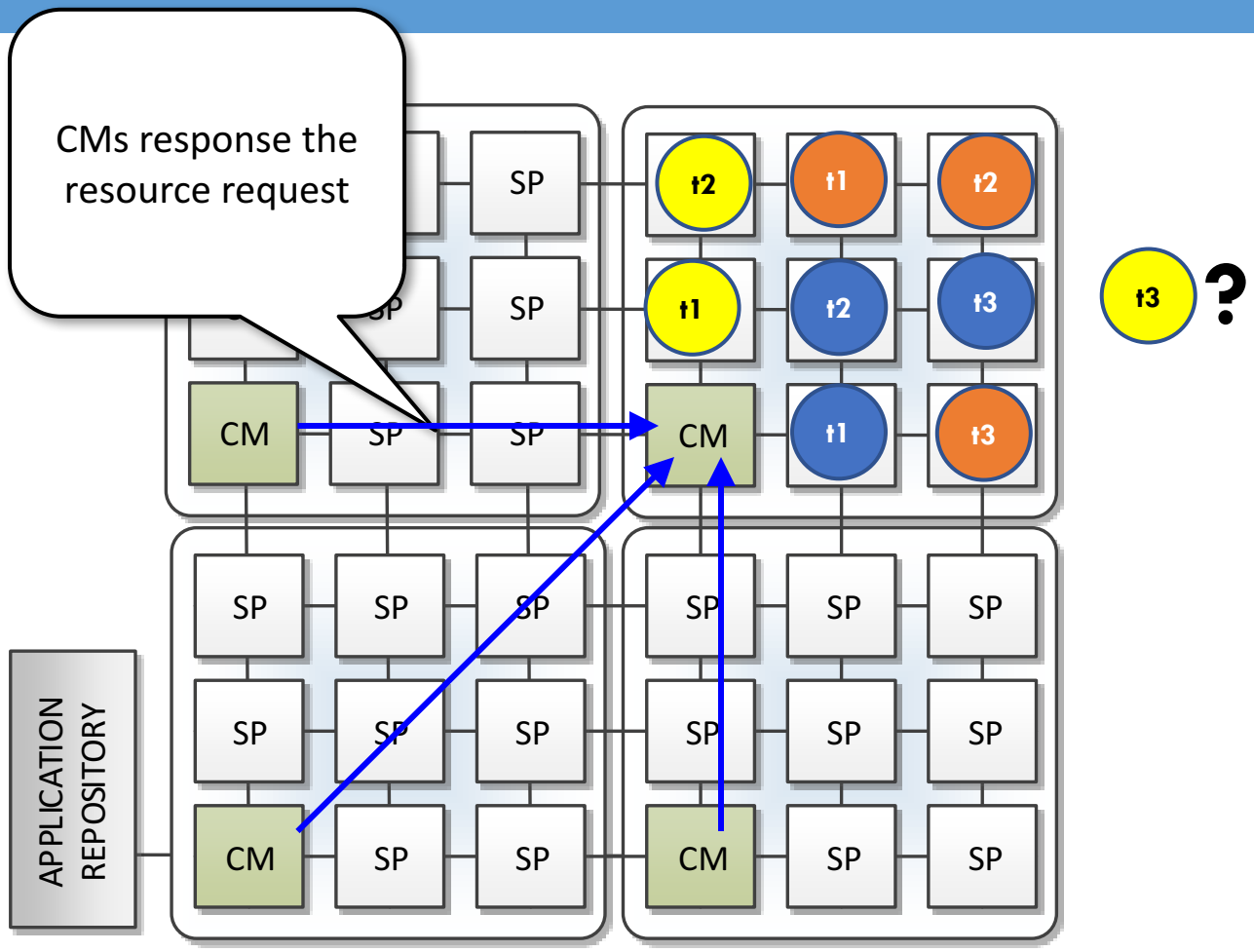
CM – Cluster Manager PE / **SP** – Slave PE

Reclustering



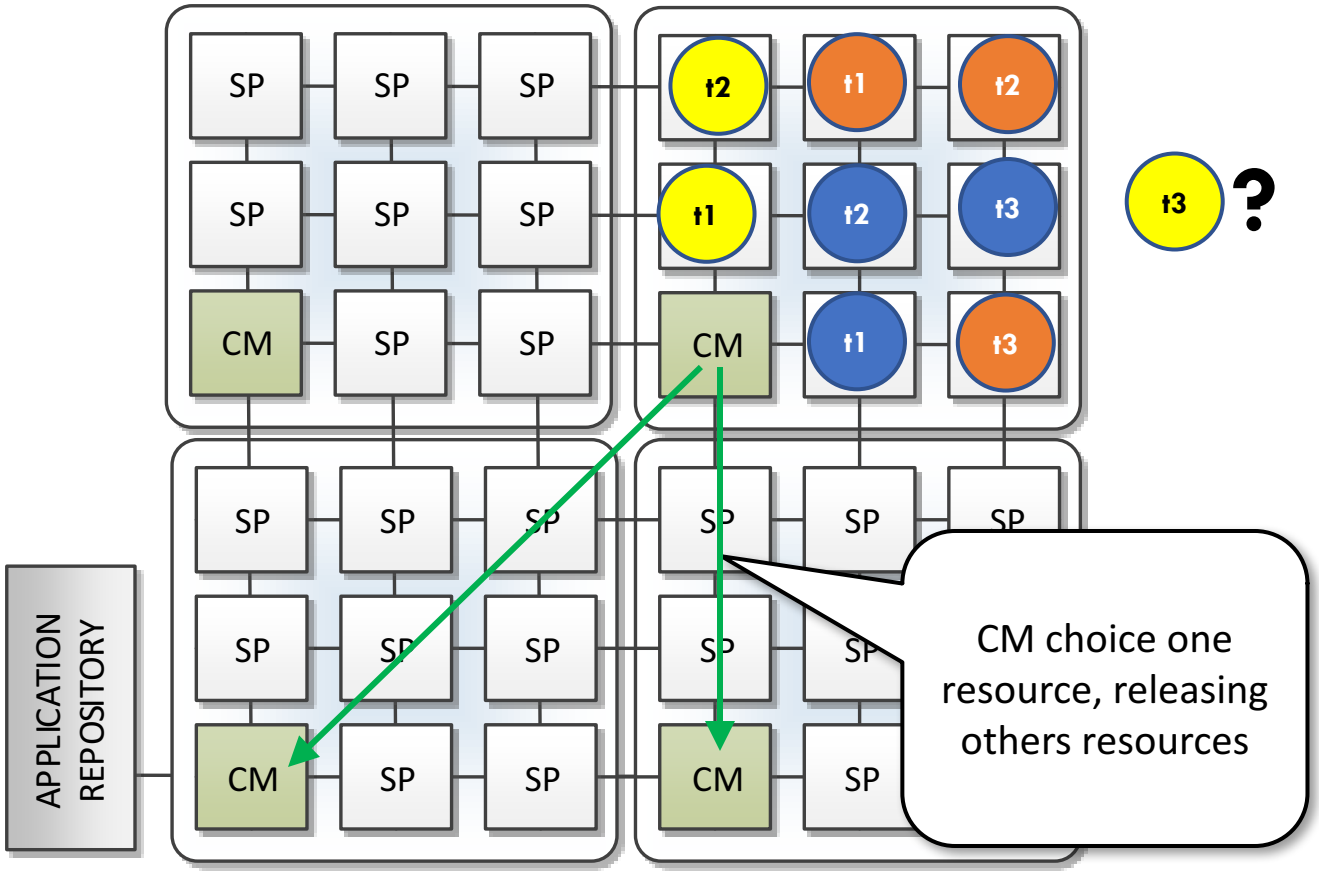
CM – Cluster Manager PE / **SP** – Slave PE

Reclustering



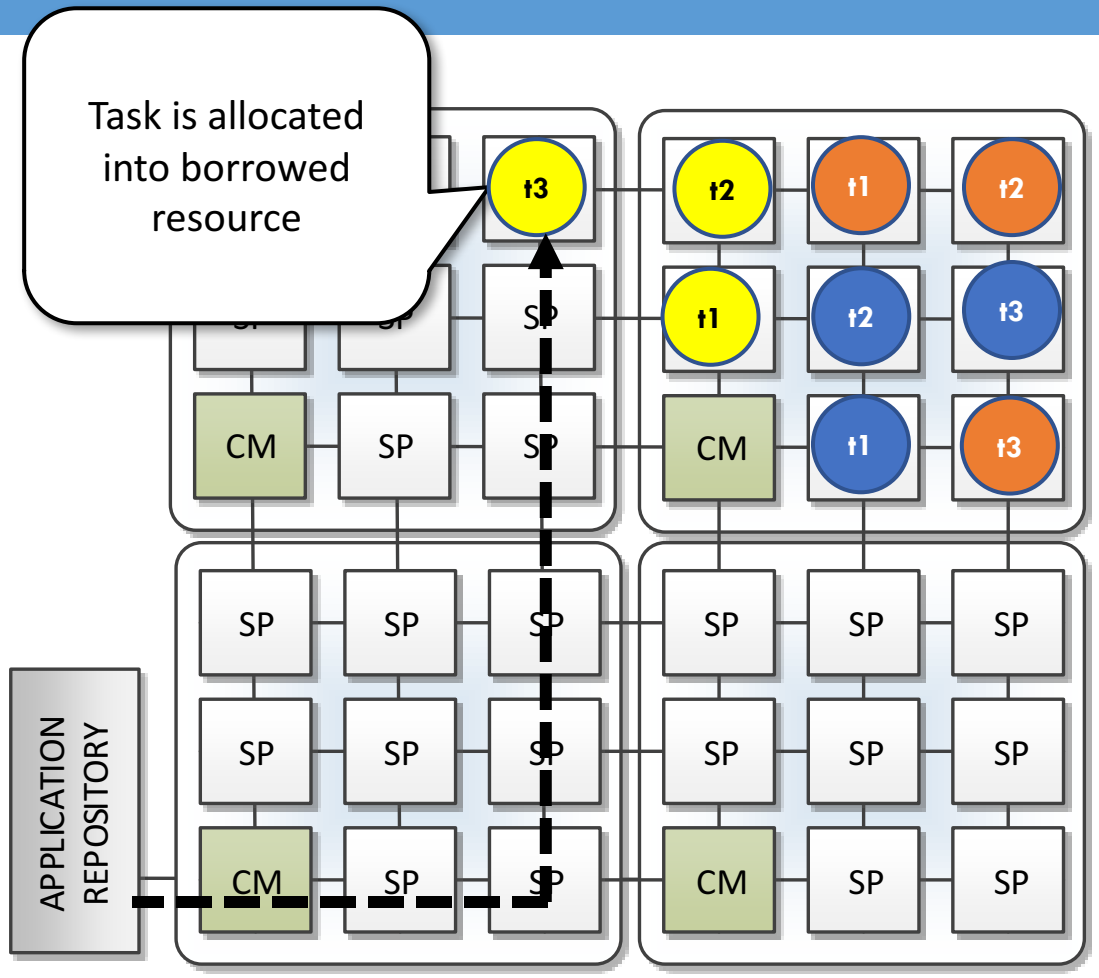
CM – Cluster Manager PE / **SP** – Slave PE

Reclustering



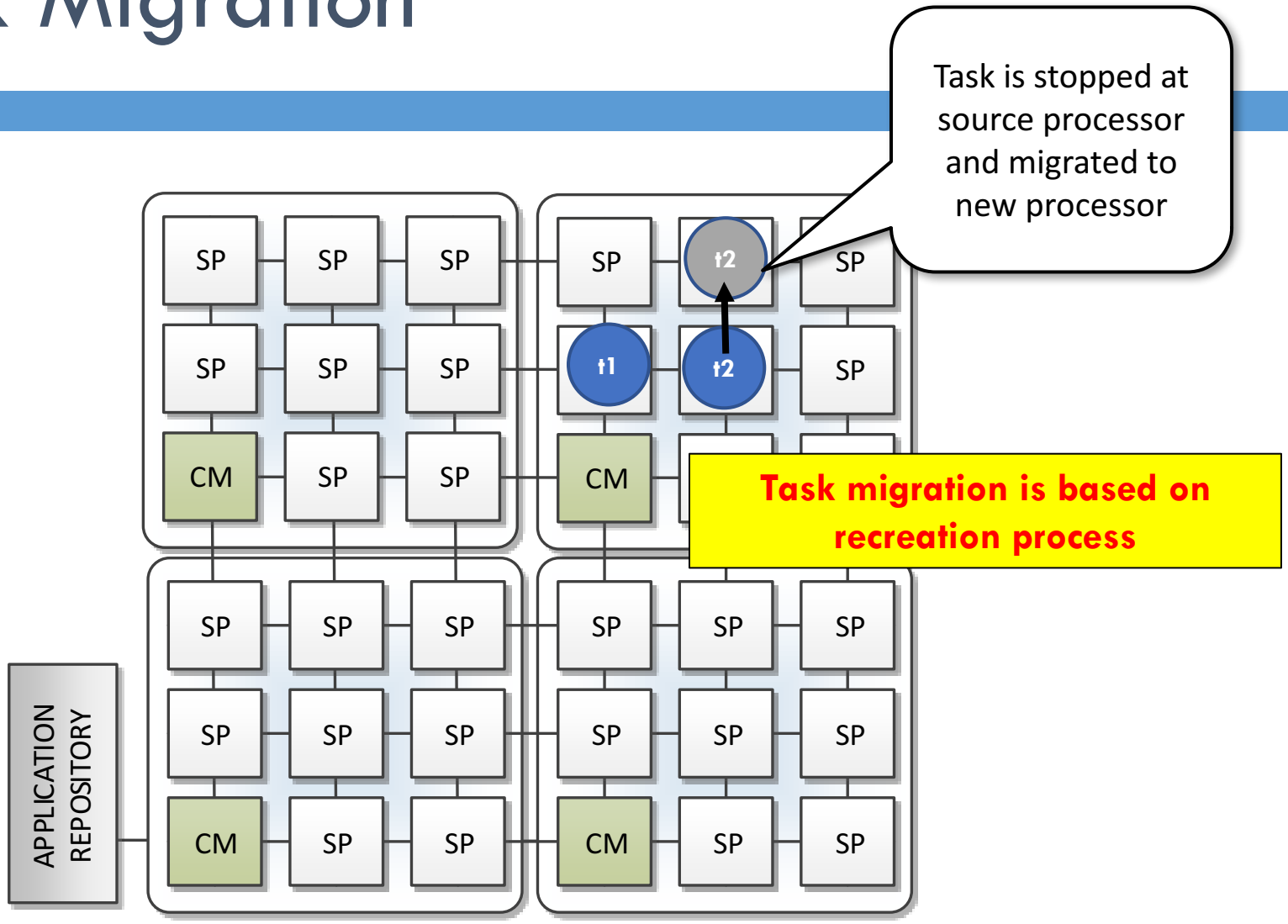
CM – Cluster Manager PE / **SP** – Slave PE

Reclustering



CM – Cluster Manager PE / **SP** – Slave PE

Task Migration

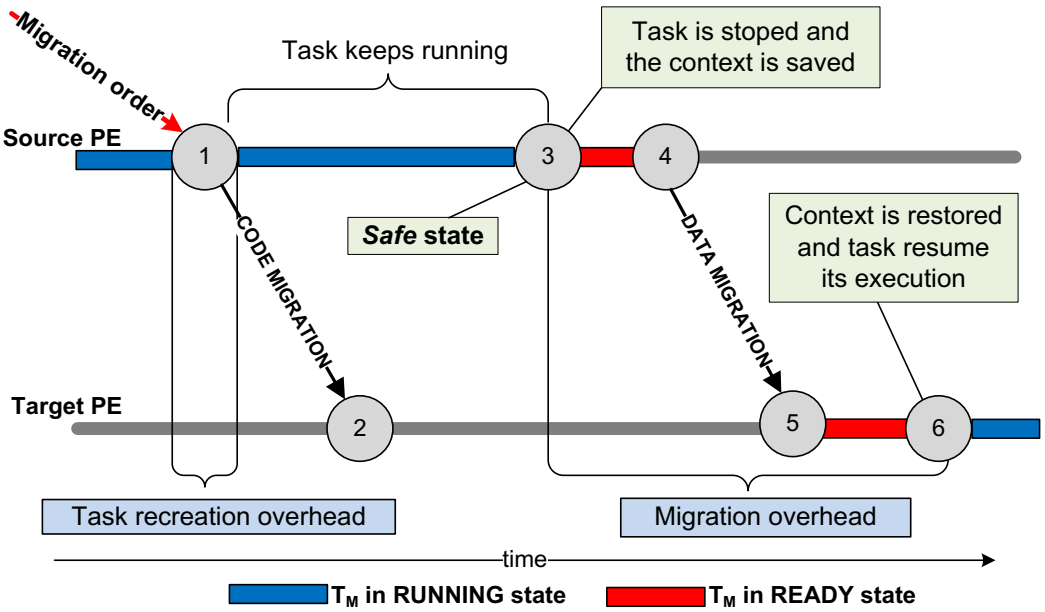


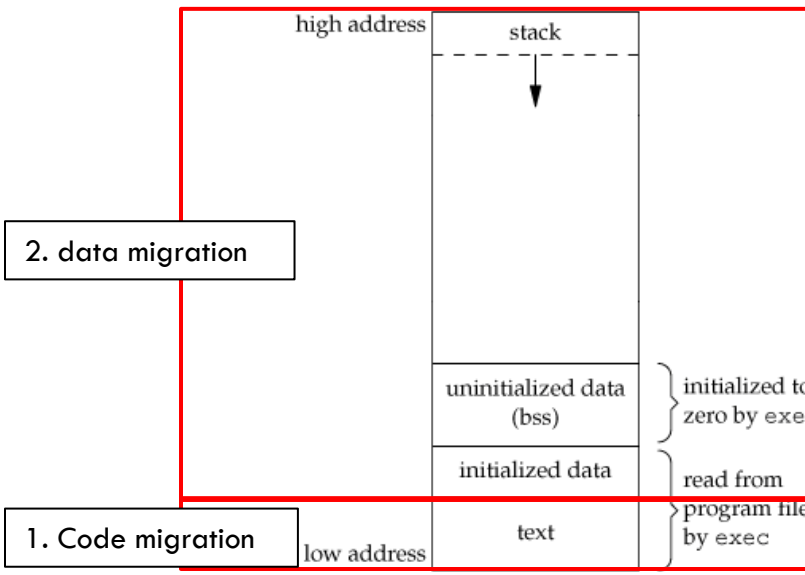
CM – Cluster Manager PE / **SP** – Slave PE

Task Migration

Migration occurs into steps

- Task **keeps running during its some migration steps**
- Task is only stopped when **safe points** are automatically identified by the migration process (software)
- Safe point are moment which the **task is not waiting for a message from another task**





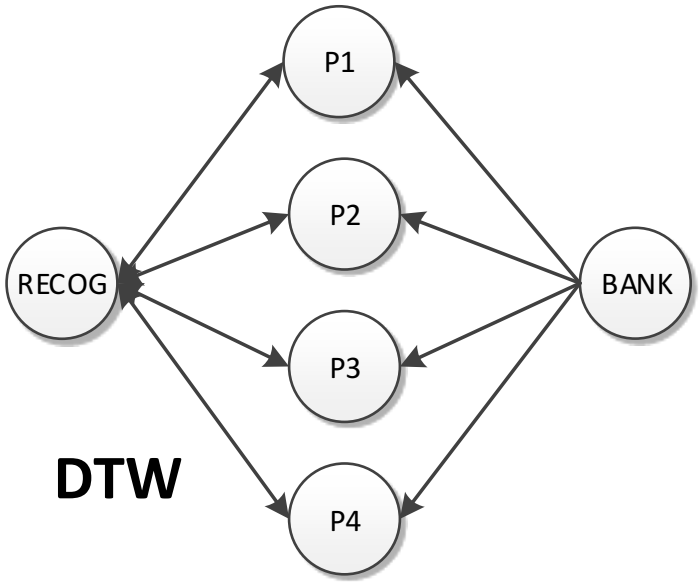
Logical Features

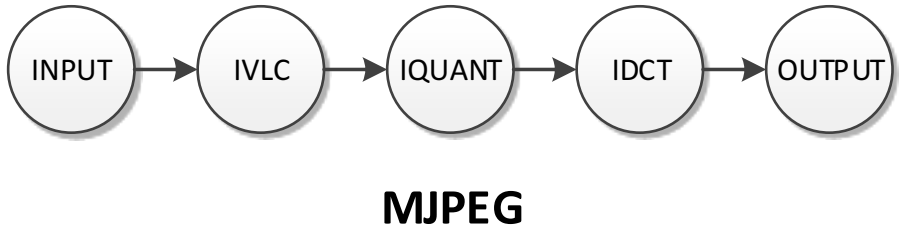
User's Application Execution

Application

An application is a **set of communicating tasks** (each task is a .c file)

Application are **described as a CTG: Communicating Task Graph**. Example of applications:





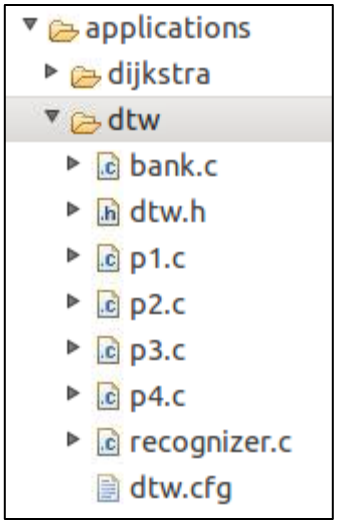
Task

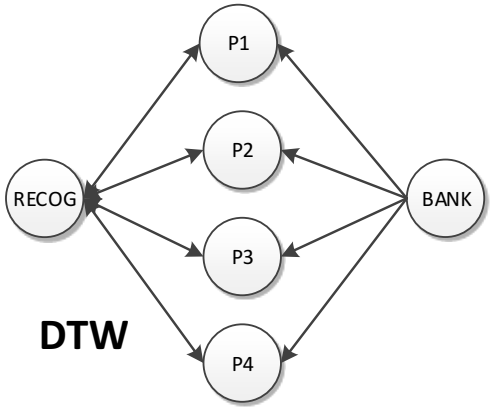
Task is a .c file which perform some computation and communication with other(s) task(s)

Example of a task code

```
int main(){

    int test[SIZE][SIZE];
    int pattern[SIZE][SIZE];
    int result, j;

    Receive(&msg, recognizer);

    Echo("Task P1 INIT\n");

    memcpy(test, msg.msg, sizeof(test));

    for(j=0; j<PATTERN_PER_TASK; j++){

        Echo("Task P1 FOR\n");

        memset(msg.msg,0, sizeof(int)*MSG_SIZE);

        Receive(&msg, bank);

        //Echo("Task P1 received pattern from bank\n");

        memcpy(pattern, msg.msg, sizeof(pattern));

        result = dynamicTimeWarping(test, pattern);

        msg.length = 1;

        msg.msg[0] = result;

        Send(&msg, recognizer);
    }

    Echo("Task P1 FINISHEDD IN\n");
    Echo(itoa(GetTick()));

    exit();
}
```

Example of an application task files





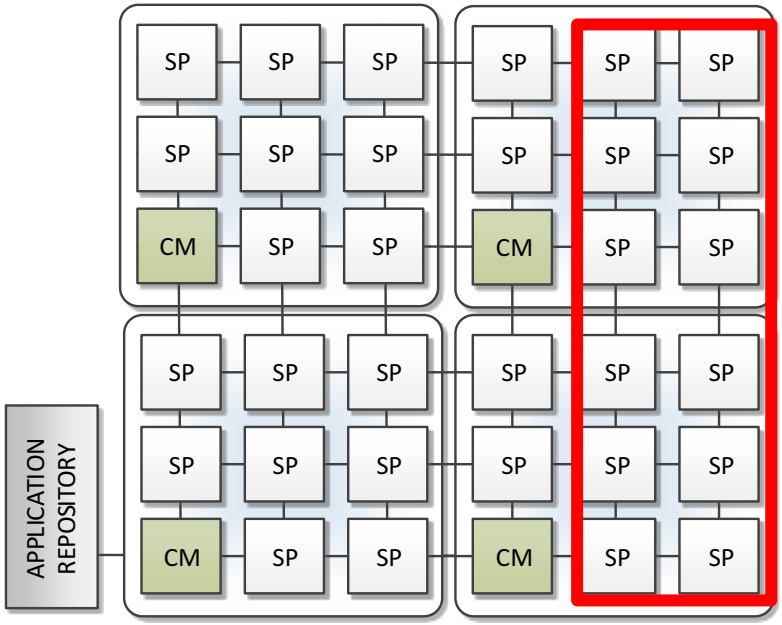
User's Application Execution

SP are dedicated to execute the user applications

SP is a PE that runs the **slave µkernel**

Performs support for user task execution

- TCB – Task Control Block
- Inter-task communication
- Scheduling
- Interruption Handling
- API – by System Calls
- Idle



CM – Cluster Manager PE / **SP** – Slave PE

API

HeMPS API (MPI-based)

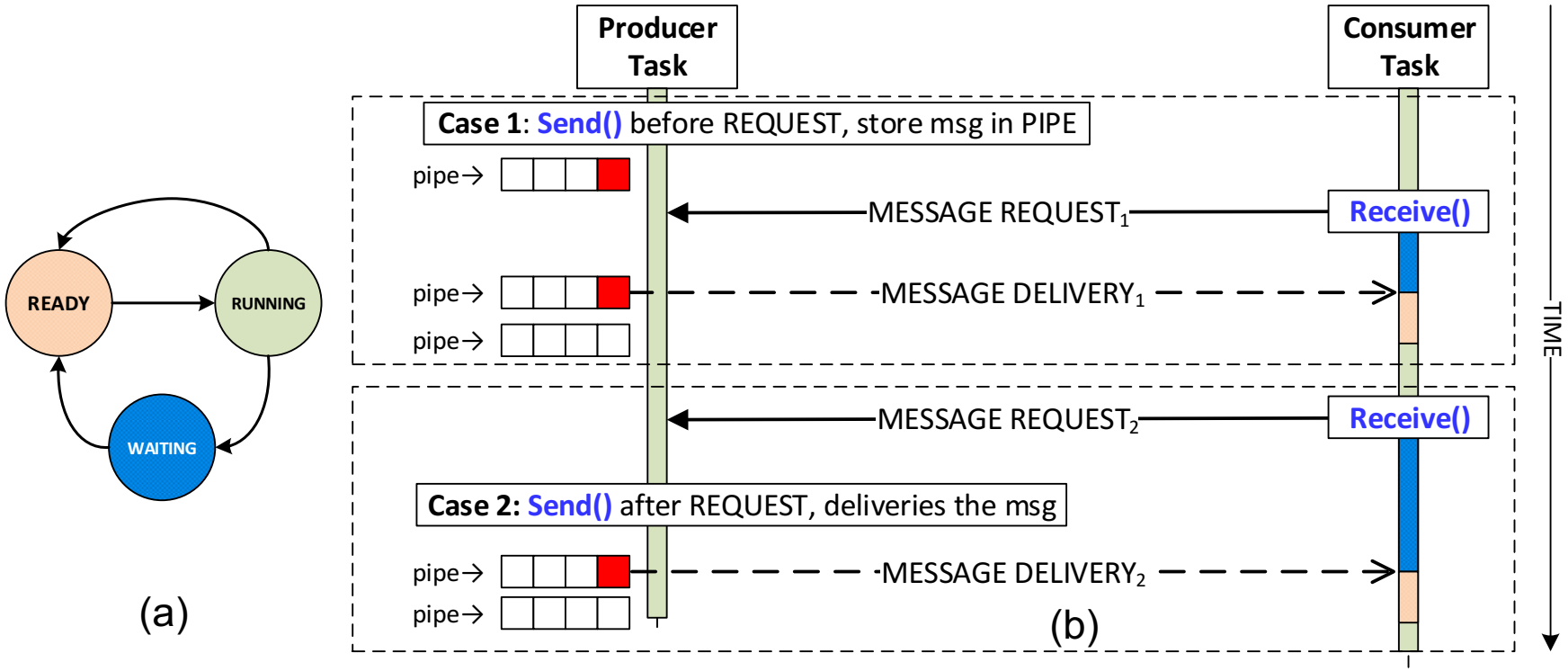
- void **Send**(Msg * msg, unsigned int target _task_ID)
- void **Receive** (Msg * msg, unsigned int source _task_ID)
- unsigned int **GetTick**(void)
- void **Echo**(char * string)
- void **Exit**(char * string)

```
typedef struct {
    int length;
    int msg[MSG_SIZE];
} Message;
```

Task communicate using **Send** and **Receive** primitives

Inter-task Communication

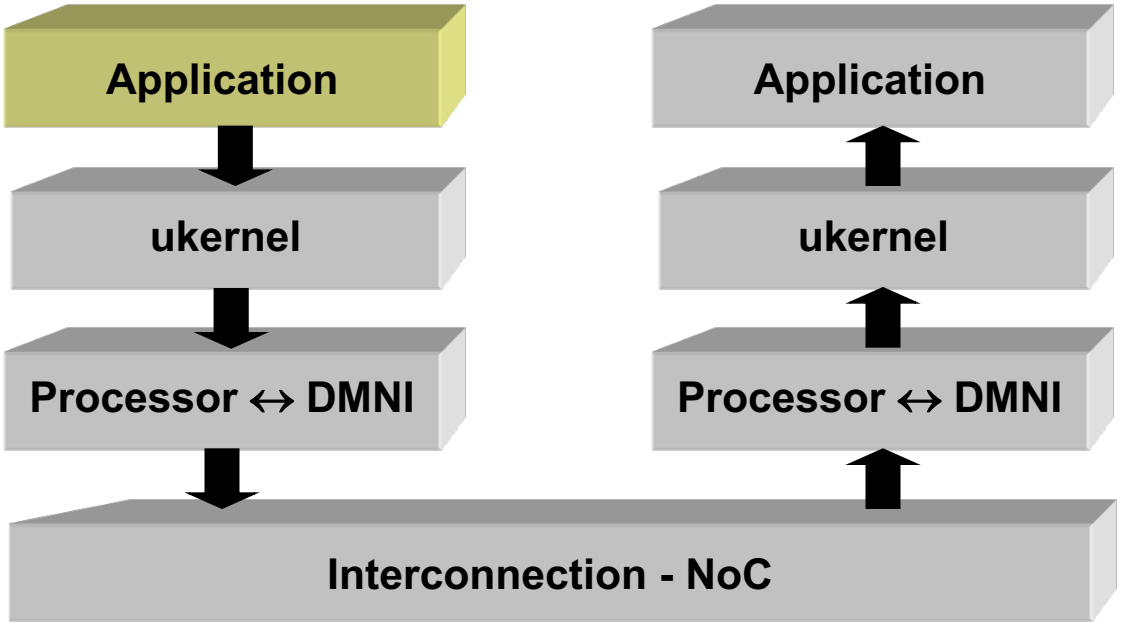
Task communicate using **Send** and **Receive** primitives



Communication Layers

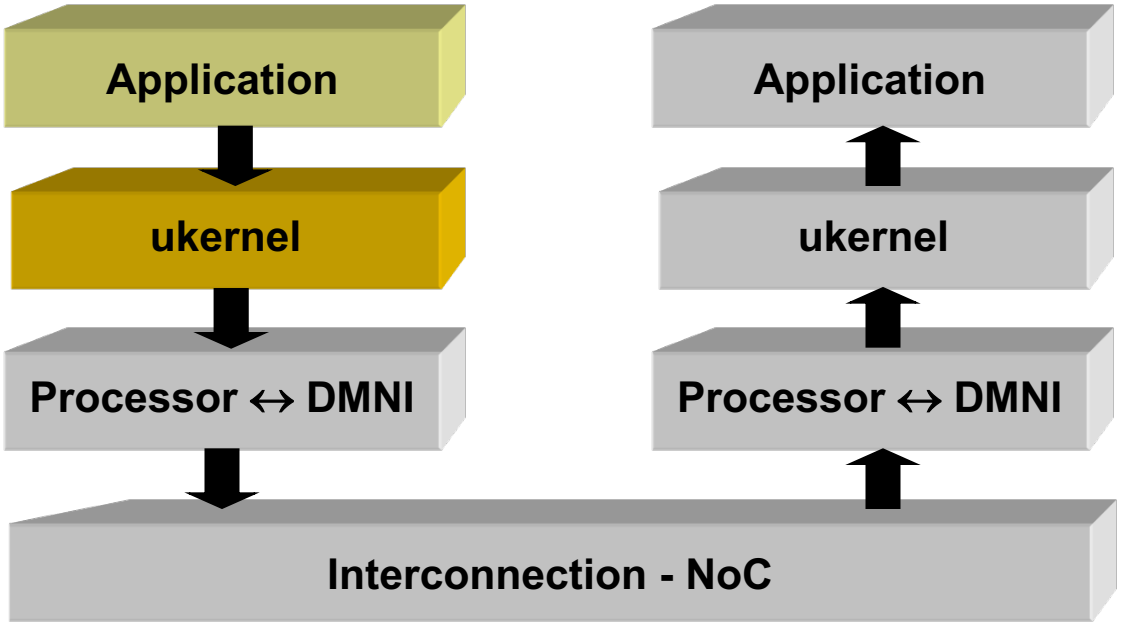
Application

- Send the messsage by calling the **Send** API primitive



Communication Layers

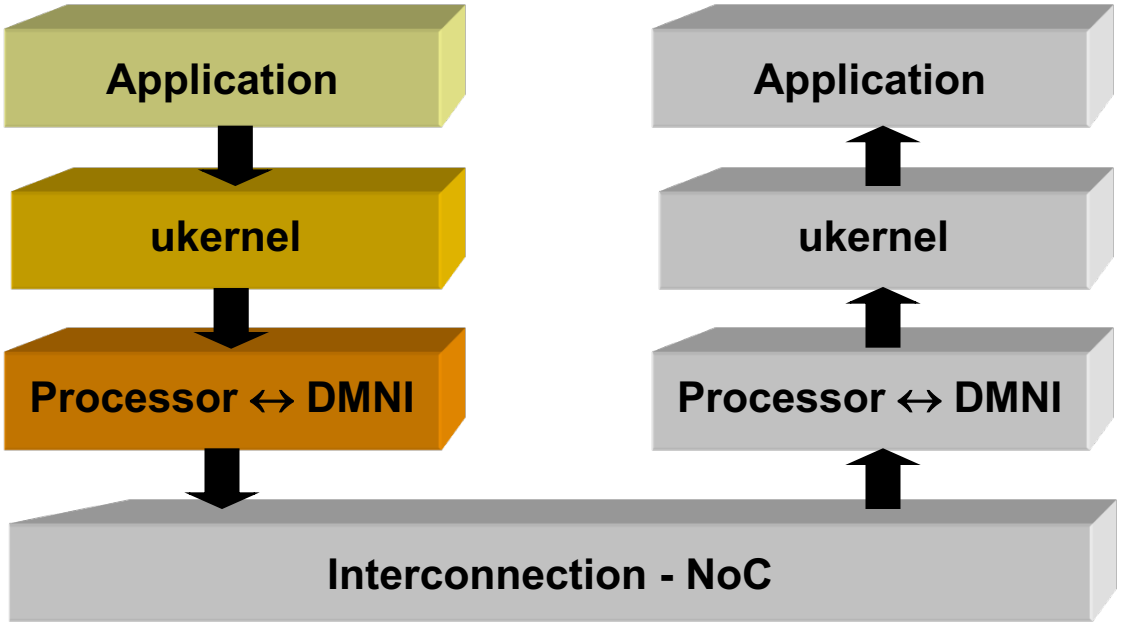
ukernel programs the DMNI to **send** memory block in a packet format



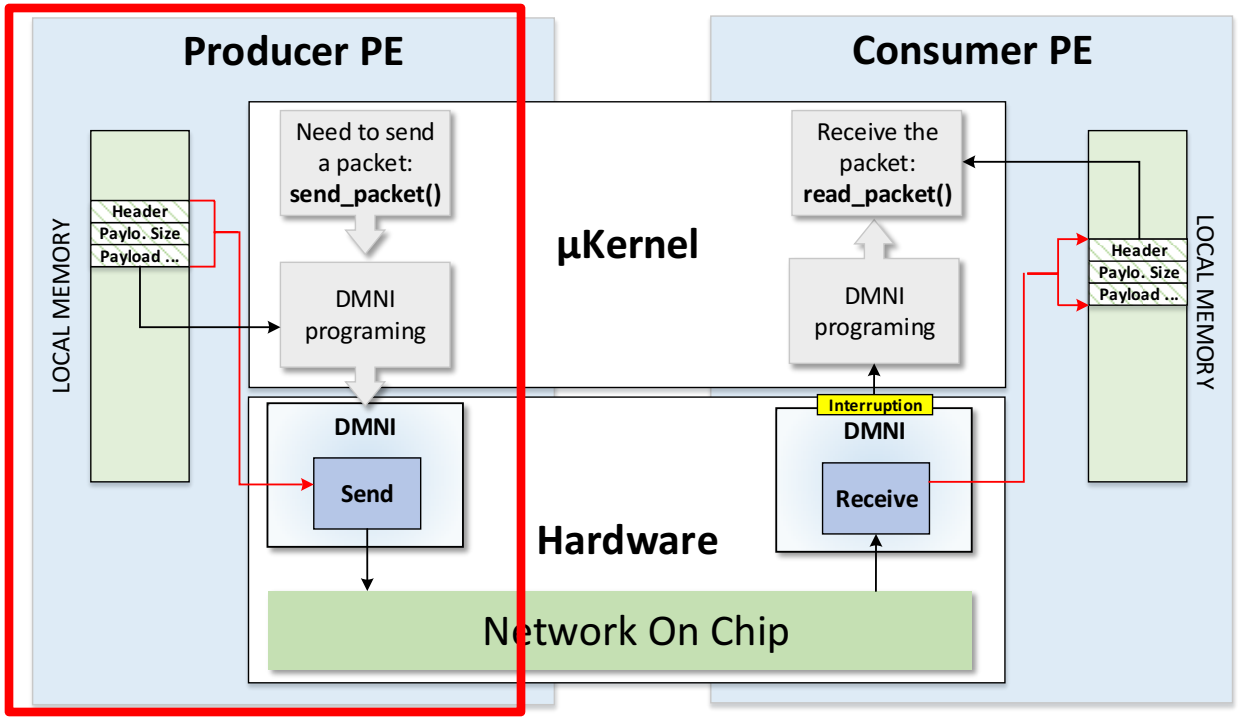
Communication Layers

DMNI (send) copies packet from memory and inject into NoC

- Can perform serialization
- Must to implement the NoC flow control



Communication Layers



send_packet(): function that programs DMNI to copy a memory block to the NoC

- Assumes that the memory block is in the format of a NoC packet

DMNI - **Send**

Objective: copy a memory block injecting into the NoC

- The particular feature of this module is the possibility to **transfer two memory blocks with one software programming**

send_packet() API is responsible for to expose the DMNI send feature to the software by configuring MMR

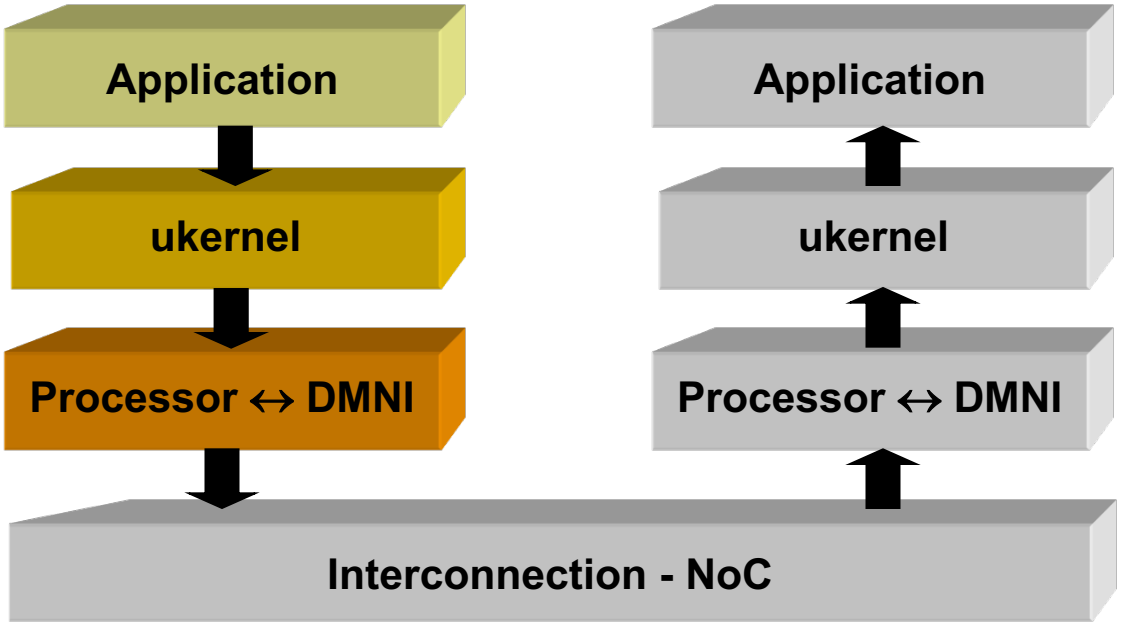
```
1.  void send_packet(mem_size_1, mem_addr_1, mem_size_2,
    mem_addr_2){
2.    while (MemoryRead(DMNI_SEND_ACTIVE));
3.    MemoryWrite(DMNI_SIZE, mem_size_1);
4.    MemoryWrite(DMNI_ADDRESS, mem_addr_1);
5.    if (mem_size_2 > 0){
6.      MemoryWrite(DMNI_SIZE_2, mem_size_2);
7.      MemoryWrite(DMNI_ADDRESS_2, mem_addr_2);
8.    MemoryWrite(DMNI_OP, READ);
9.    MemoryWrite(DMNI_START, 1);
10. }
```

Fig. 5 – *Send_packet()* function, executed in the µkernel of the processor.

Communication Layers

NoC – Network on Chip

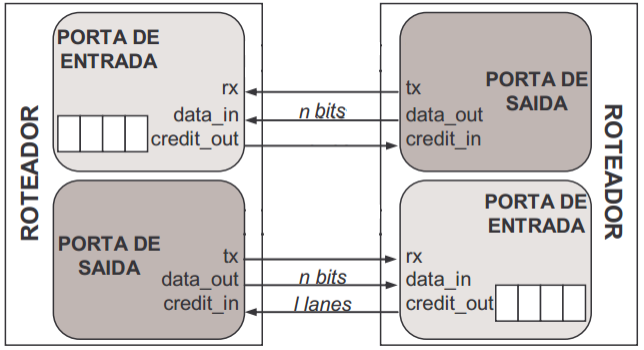
- Send the packet to the destination PE
- Packet is divided in flits

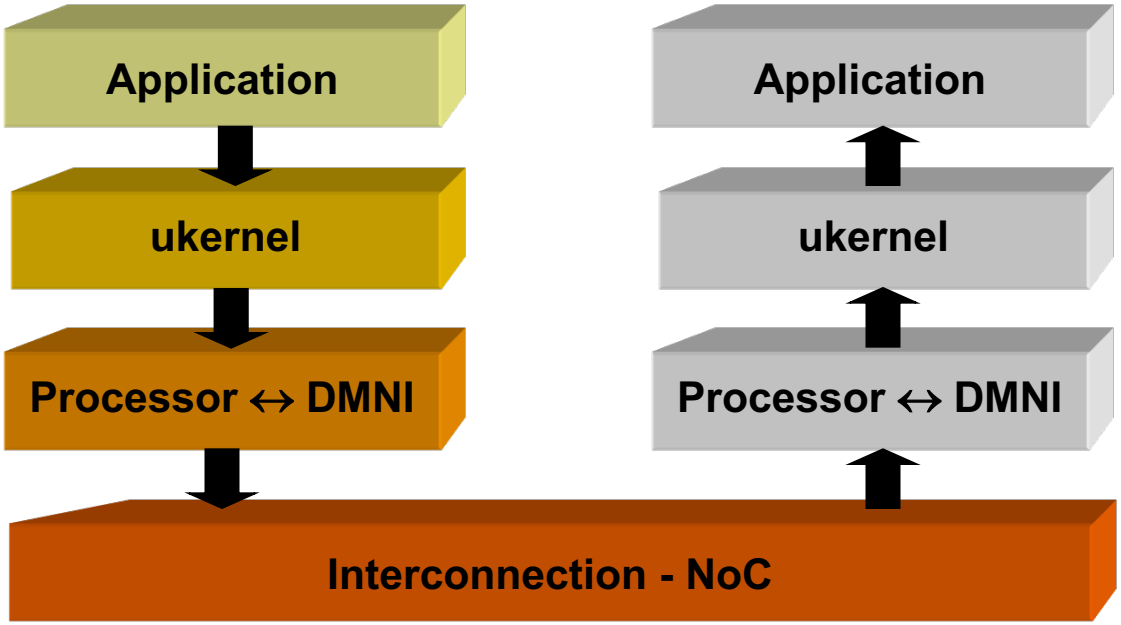


Communication Layers

NoC – Network on Chip

- Send the packet to the destination PE
- Packet is divided in flits

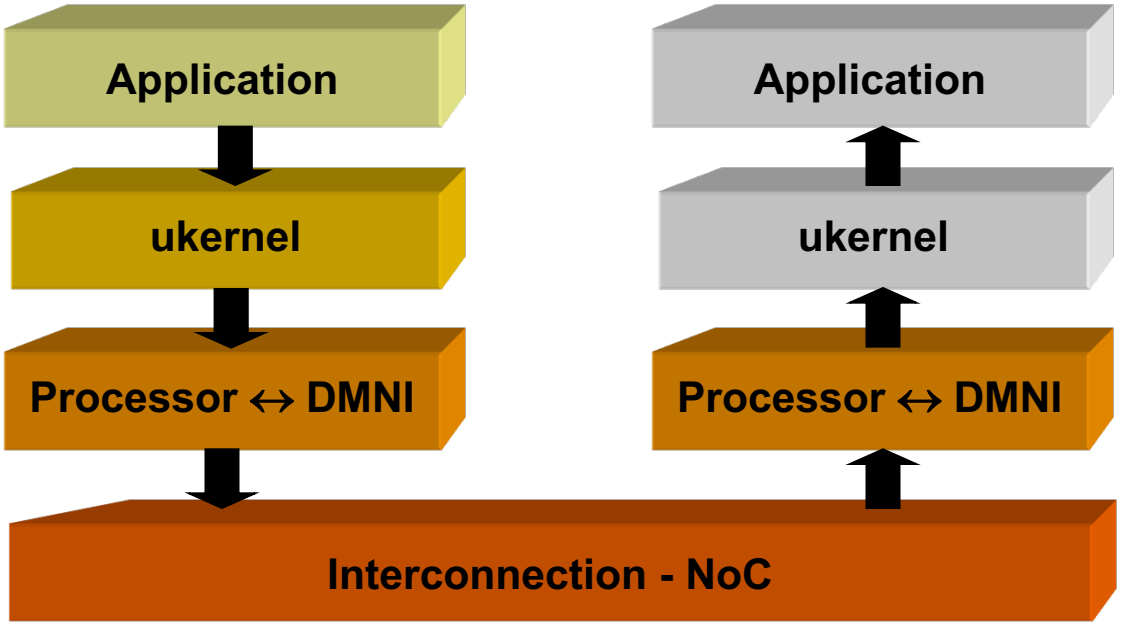




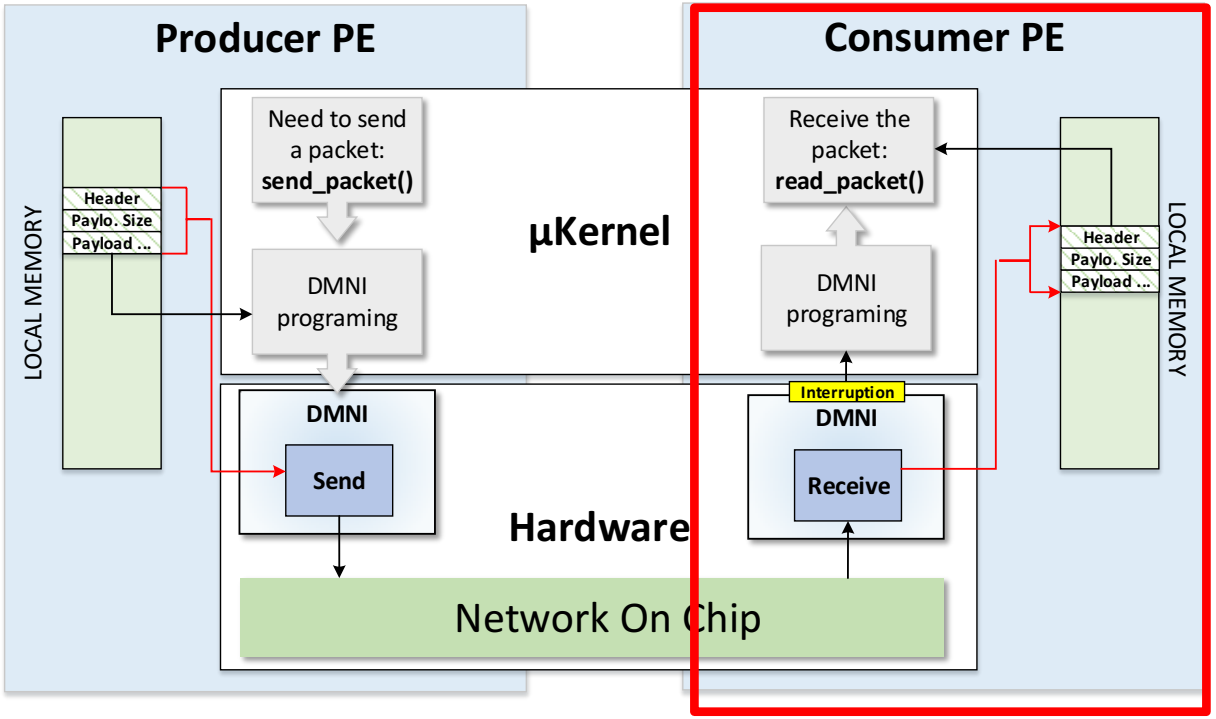
Communication Layers

DMNI (receive) copies packet from NoC and transfers into memory

- Can perform deserialization
- Must to implement the NoC flow control



Communication Layers



read_packet(): function that programs the DMNI to copy a NoC packet to a memory block

- Fired by a interruption

DMNI - **Receive**

Objective: receiving a NoC packet coping to a specified memory address

- Also it generates a *software interruption* when detects a incoming packet

receive_packet() API is responsible for expose the DMNI receive feature to the software by configuring MMR

- Called through a *software interruption*, generated when a incoming packet is detected by DMNI

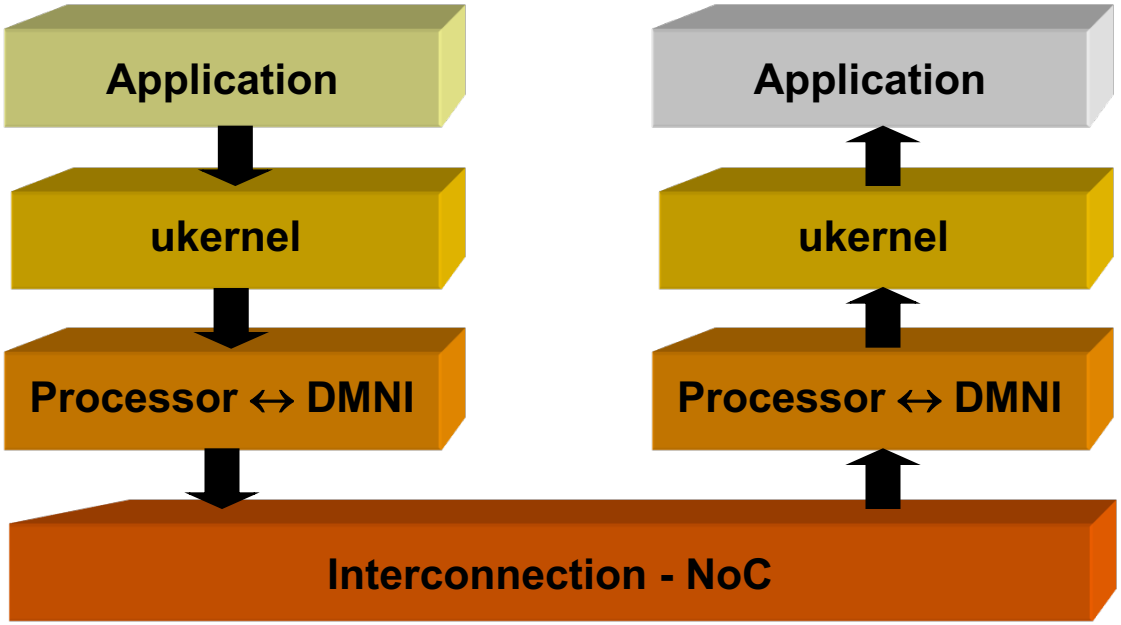
```
1.  void read_packet (init_addr, packet_size)
2.      MemoryWrite(DMNI_SIZE, packet_size);
3.      MemoryWrite(DMNI_ADDRESS, init_addr);
4.      MemoryWrite(DMNI_OP, WRITE);
5.      MemoryWrite(DMNI_START, 1);
6.      while (MemoryRead(DMNI_RECEIVE_ACTIVE));
7.  }
```

Fig. 9 – *Read_packet()* function, executed in the μkernel of the processor.

Communication Layers

ukernel

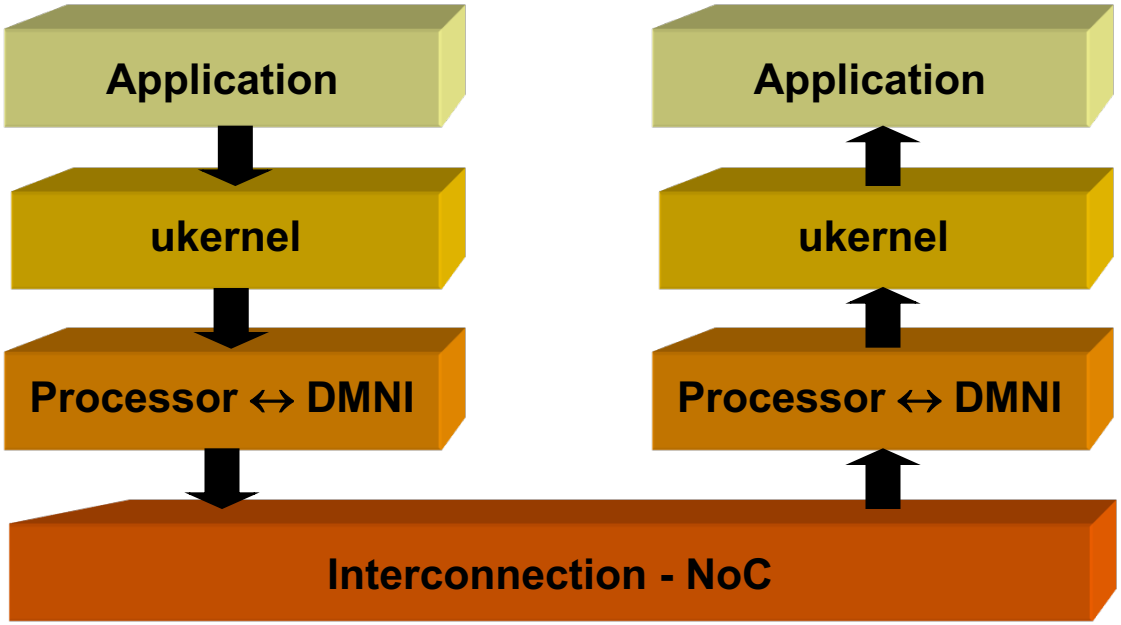
- Handle packets from DMNI by implementing a interruption handling mechanism (OS_InterruptServiceRoutine)
- Is responsible to program the DMNI



Communication Layers

Application

- Receive the packet by calling the **Receive** primitive API



Debugging

Debugging

Debugging can be performed from two perspective

From the system developer viewpoint

- By using the **HeMPS Debugger Tool (HDT)**

From the user viewpoint

- By using **Deloream**
  - Currently integrated into HDT

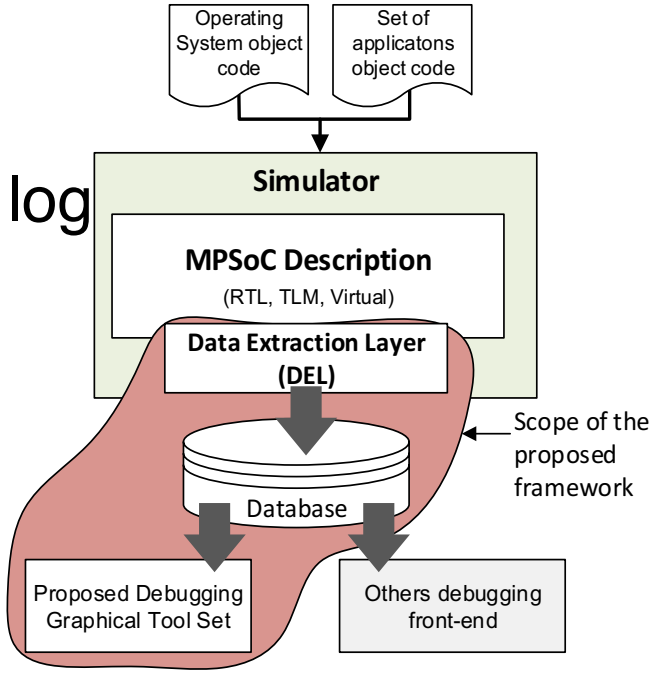
Debugging Framework

Data Extraction (back-end)

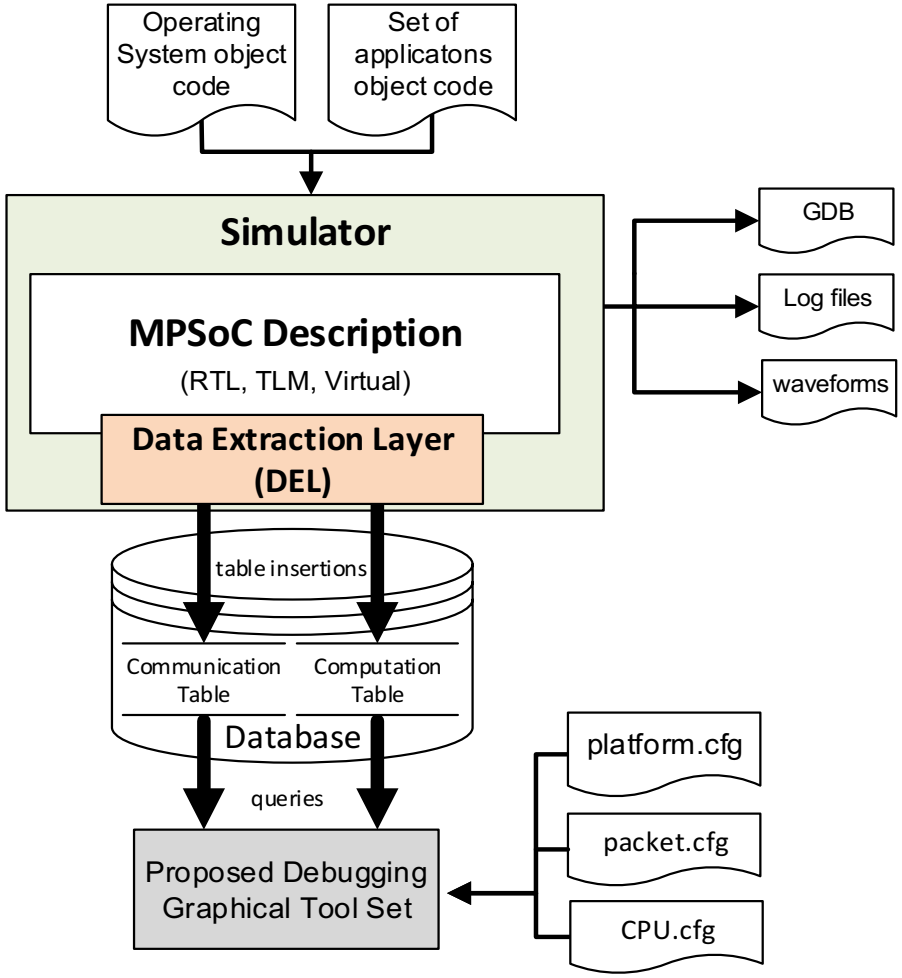
- Extracts simulated data from platform
- Inserts into a DB or generated log files
- Data extraction following a standard to be generic

Graphical Debugging (front-end)

- Read extracted data from DB or log files
- Enable easy debugability by the graphical features



Overview


Operating System object code
Set of applicatons object code
Simulator
MPSoC Description
(RTL, TLM, Virtual)
Data Extraction Layer (DEL)
GDB
Log files
waveforms
table insertions
Communication Table
Computation Table
Database
queries
Proposed Debugging Graphical Tool Set
platform.cfg
packet.cfg
CPU.cfg

Main View

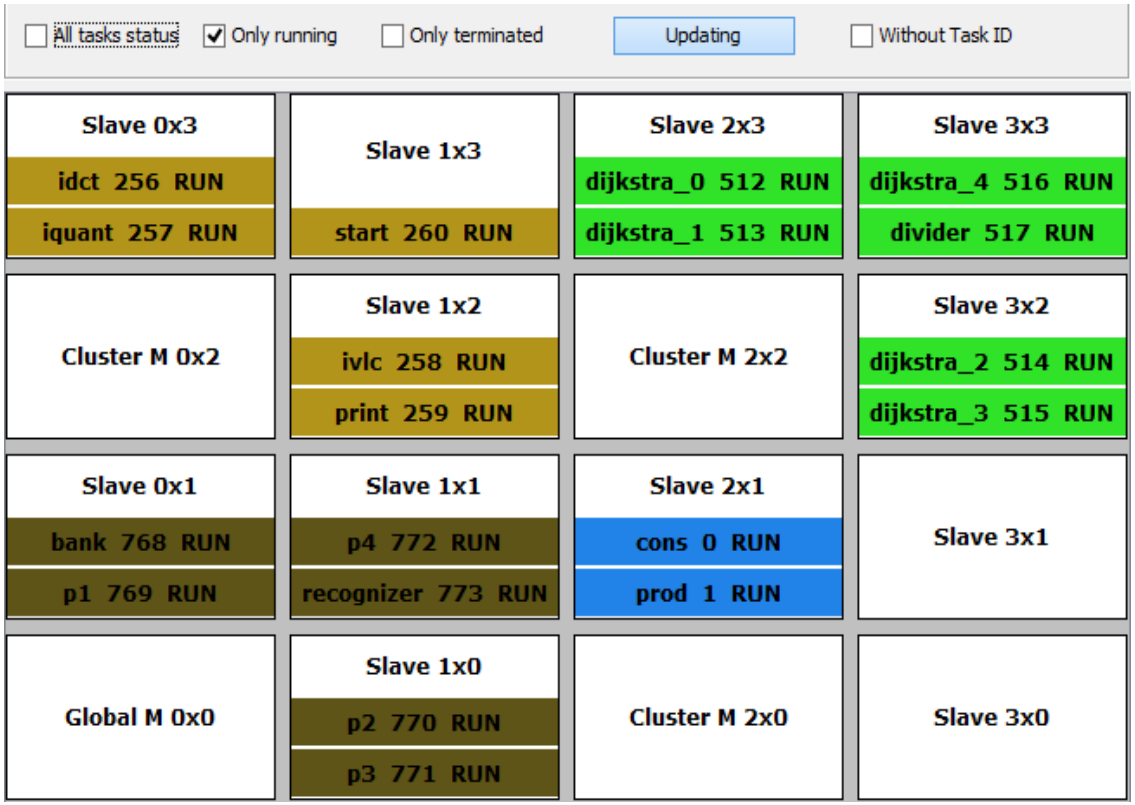
Debug:

- Communication flows
- Routing Algorithms
- Link utilization
- Management Protocols
- Parallel communications

Mapping View

Debug

- Task mapping algorithm
- PEs occupation
- Task execution status

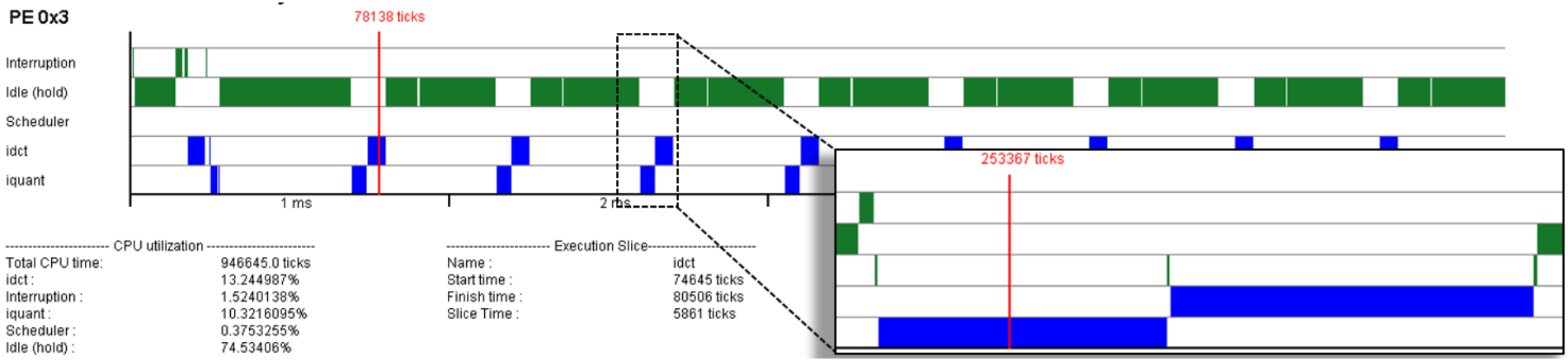


CPU Utilization View

Enables the designer to verify the CPU use by different software parts over the time

Debug

- Scheduling algorithms
- OS and task bugs
- Other software malfucntions



Deloream

Used to debug the **application's tasks logs**

It is a **debugging** tool for **the application developer**