Development of Dependable Network-on-Chip Platform (4)

Tomohiro Yoneda
National Institute of Informatics

Masashi Imai
Hirosaki Univ.

Takahiro Hanyu
Tohoku Univ.

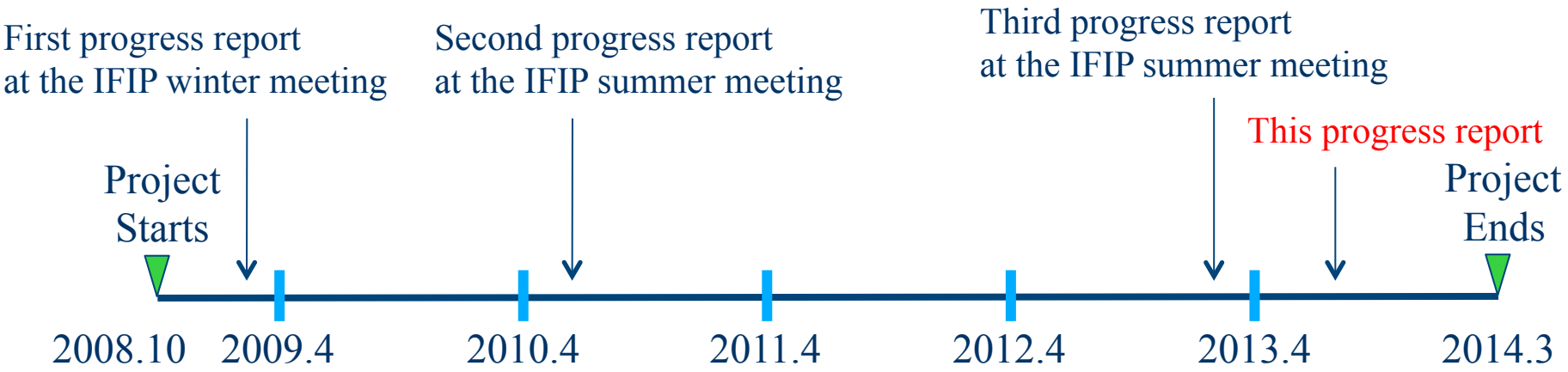
Hiroshi Saito
Univ. of Aizu

Kenji Kise
Tokyo Tech.

National Institute of Informatics
国立情報学研究所

Project summary

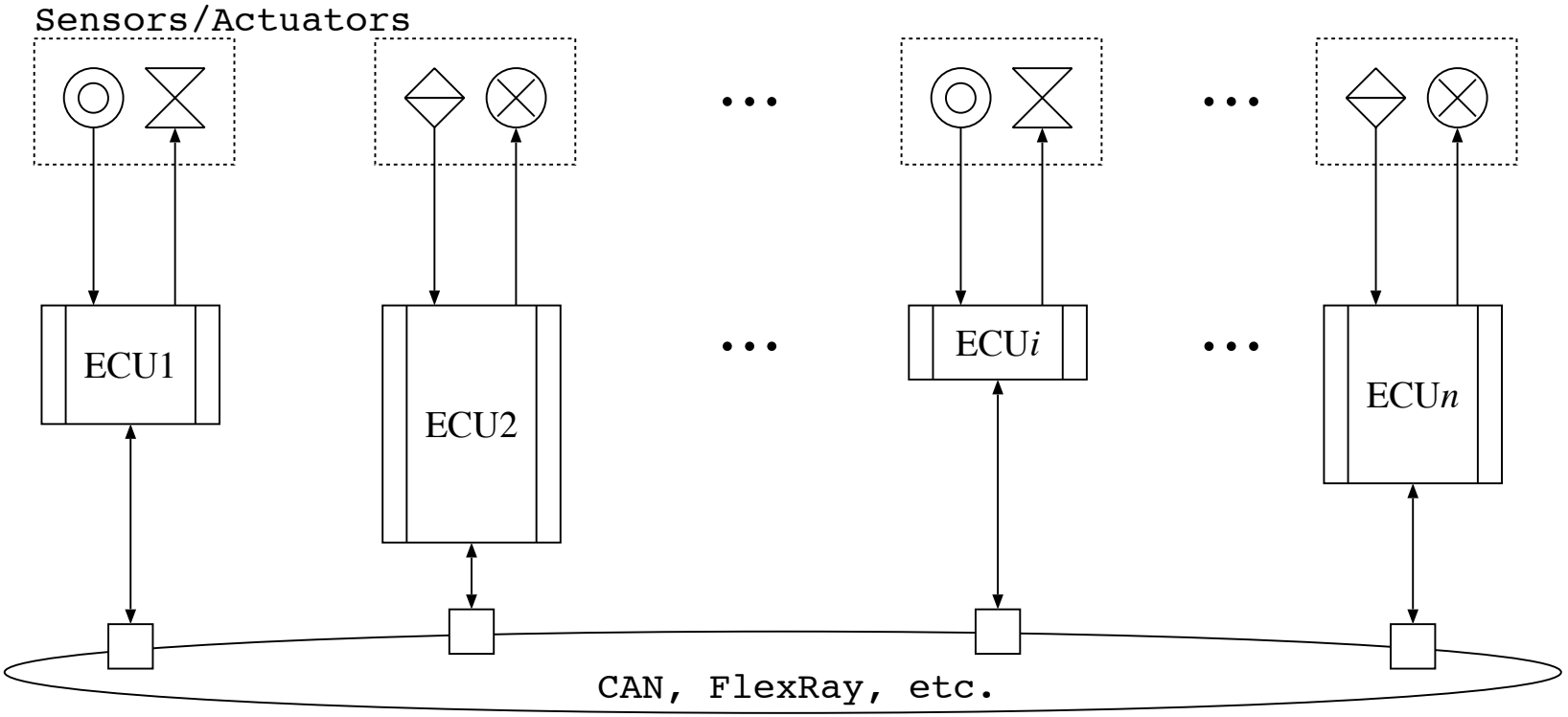
- 5.5 year National project (CREST)



- Goal
  - Platform for performing many and various tasks dependably, efficiently and adaptively
  - Demonstration in automotive control system area

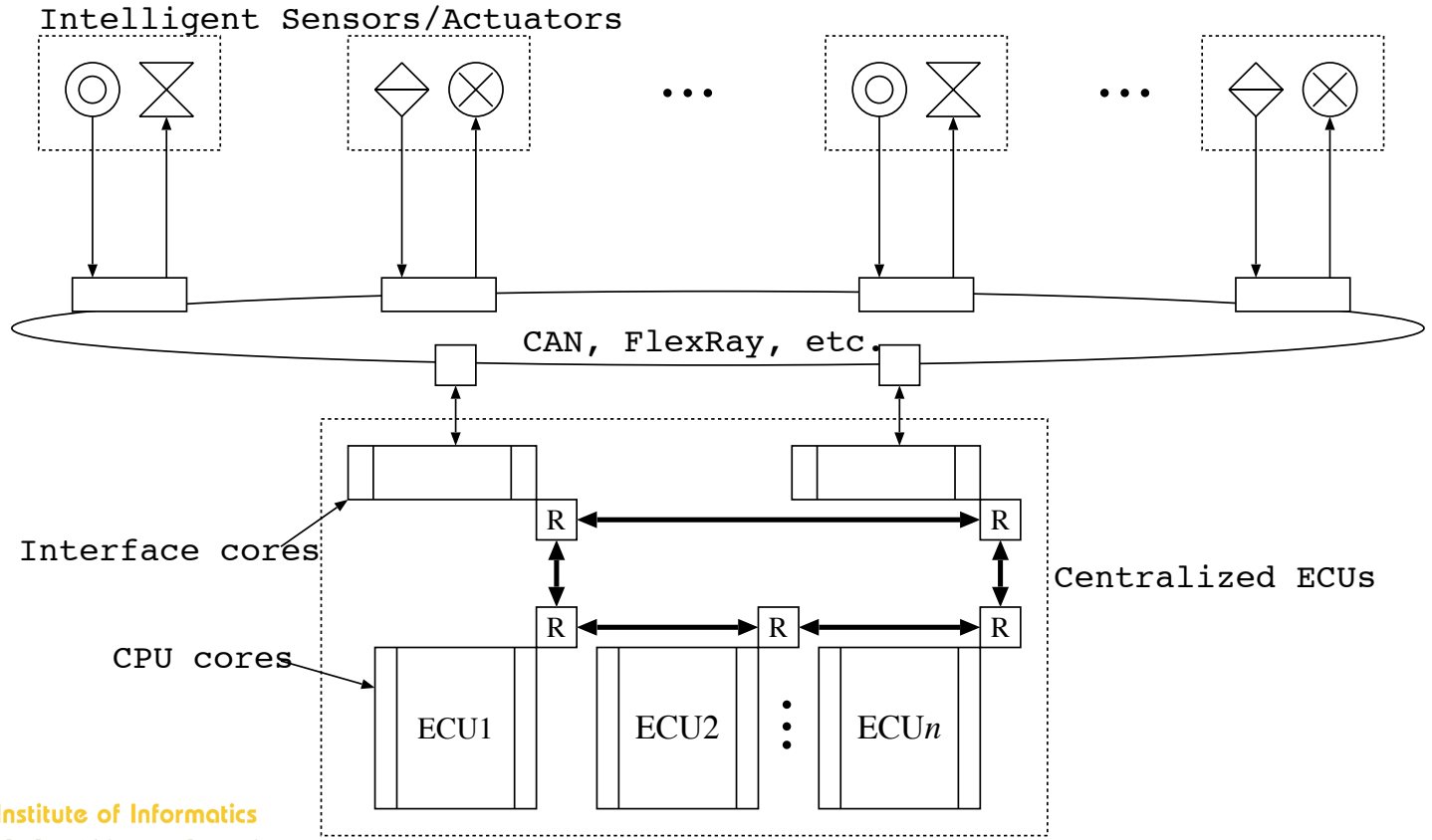
Backgrounds

- Recent cars are equipped with many ECUs
  - Conventional ECU configuration



Backgrounds

- Recent cars are equipped with many ECUs
  - Centralized ECU approach

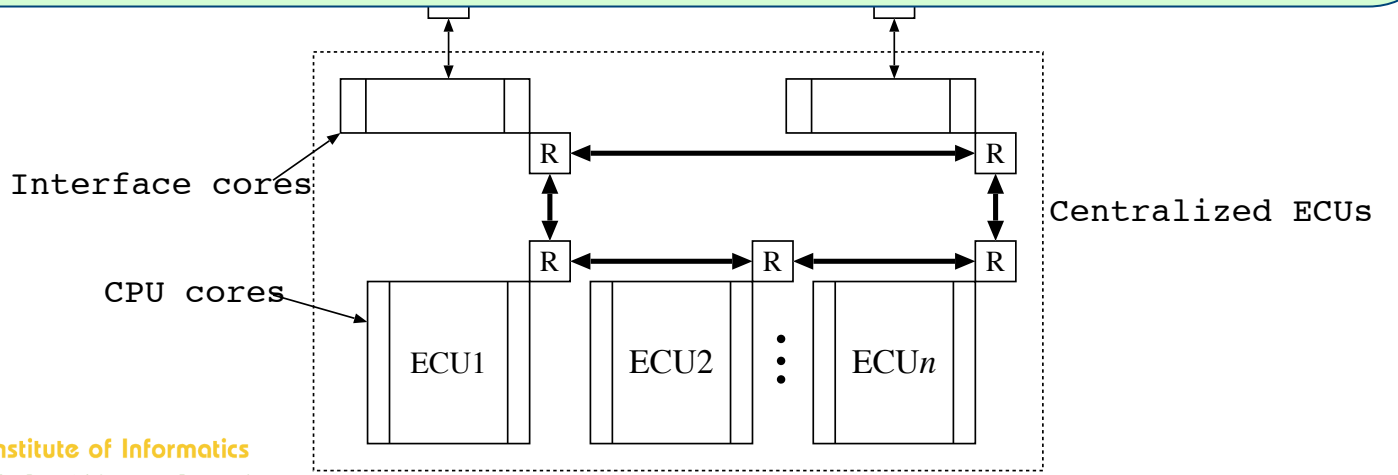


Backgrounds

- Recent cars are equipped with many ECUs
  - Centralized ECU approach

Any ECU can access any sensors/actuators

ECUs efficiently used by balancing loads
Tasks continuously executed even if some ECUs become faulty
(i.e., faulty ECU does not result in malfunction of its specific functions)

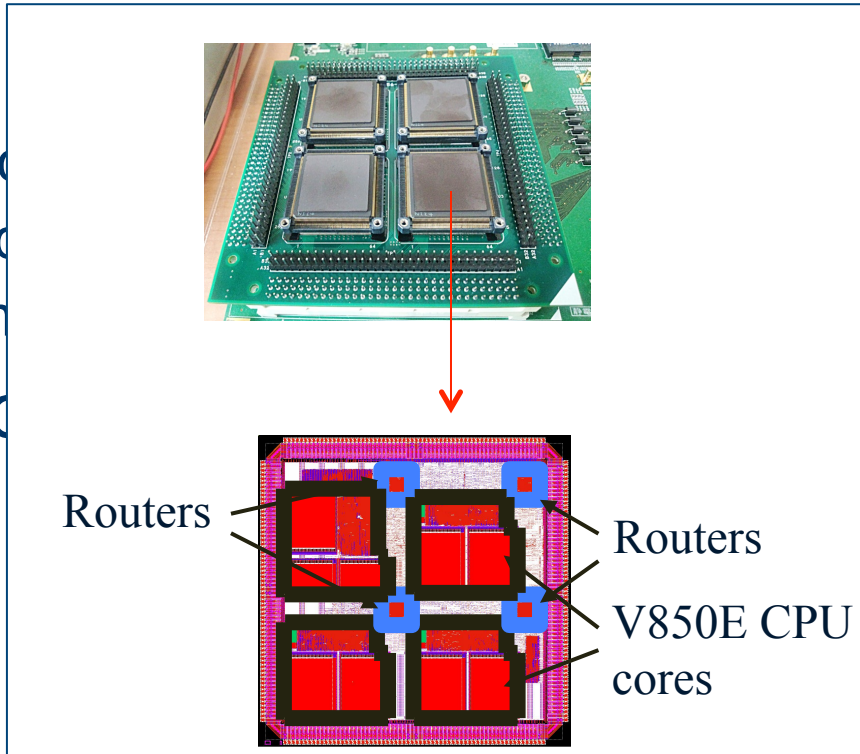


Outcome

- Hardware platform
  - Multi-Chip NoC
    - Fully asynchronous on-chip network
    - Dependable, adaptive, deadlock-free routing
    - Efficient inter-chip communication technology
- Dependable task execution
  - Modified Pair & Swap
- Task allocation
  - Redundant allocation, redundant scheduling
- Demonstration of the proposed approach
  - Practical automotive application

Outcome

- Hardware platform
  - Multi-Chip NoC
    - Fully asynchronous on-c
    - Dependable, adaptive, c
    - Efficient inter-chip comm
- Dependable task exec
  - Modified Pair & Swap
- Task allocation
  - Redundant allocation,
- Demonstration of the proposed approach
  - Practical automotive application

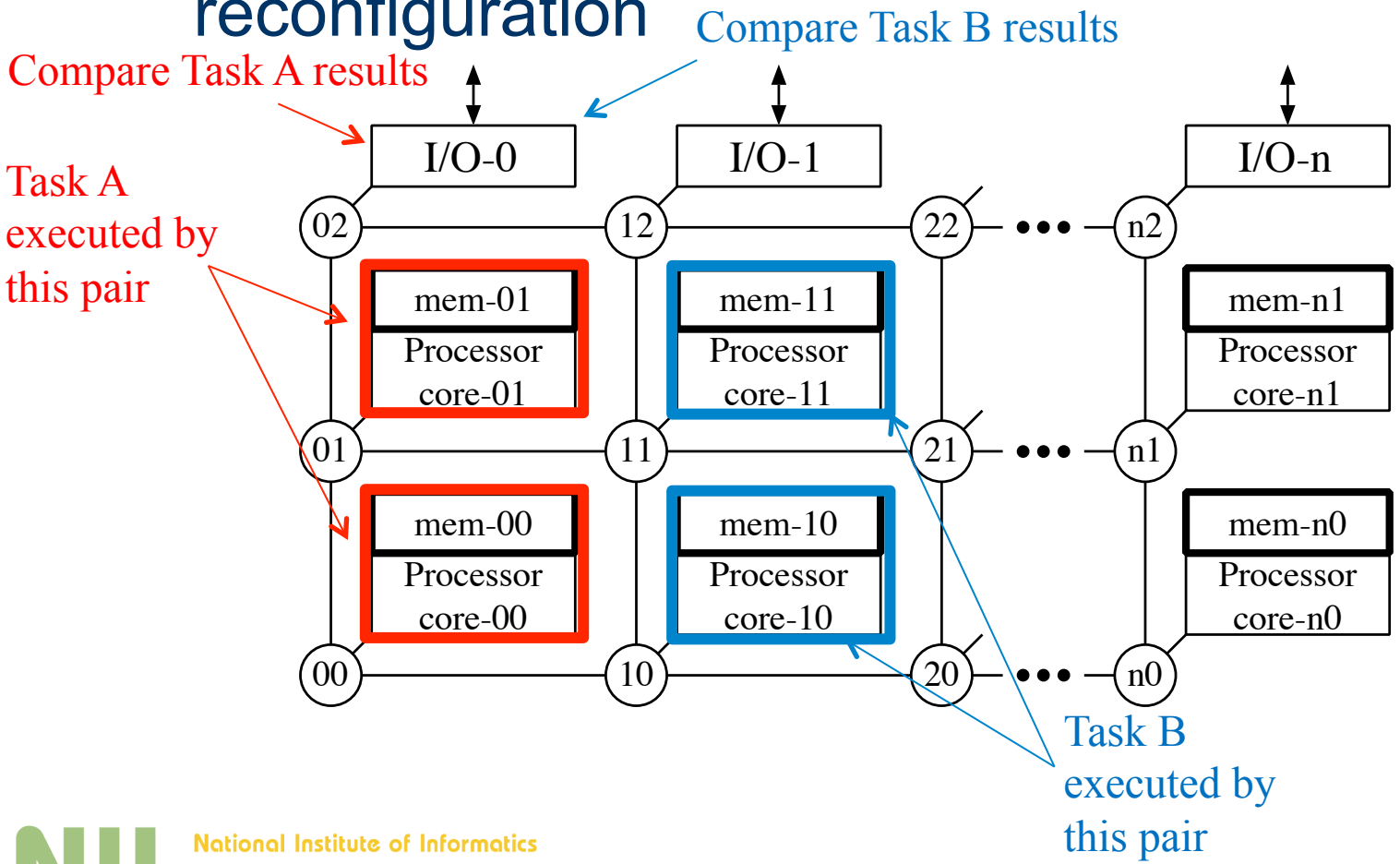


Outcome

- Hardware platform
  - Multi-Chip NoC
    - Fully asynchronous on-chip network
    - Dependable, adaptive, deadlock-free routing
    - Efficient inter-chip communication technology
- Dependable task execution
  - Modified Pair & Swap
- Task allocation
  - Redundant allocation, redundant scheduling
- Demonstration of the proposed approach
  - Practical automotive application

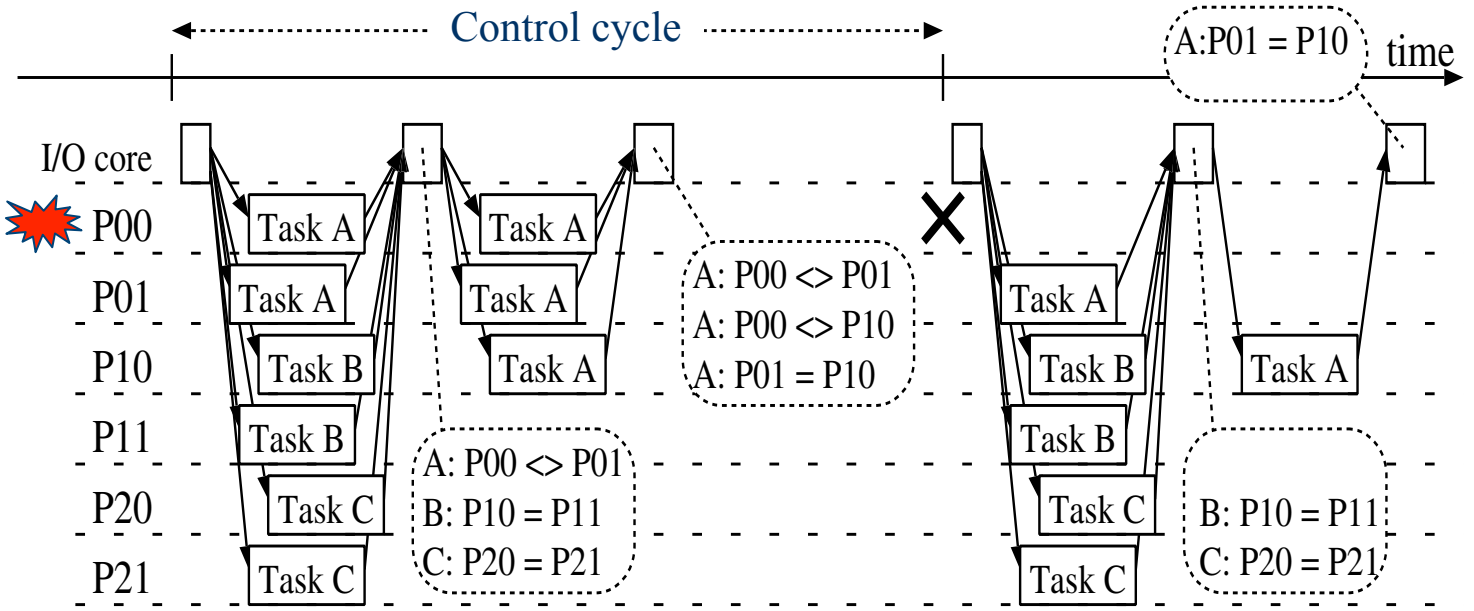
Dependability in Processor level

- Duplicated execution, comparison, and pair-reconfiguration



Modified Pair & Swap

- Duplicated execution, comparison, and pair-reconfiguration

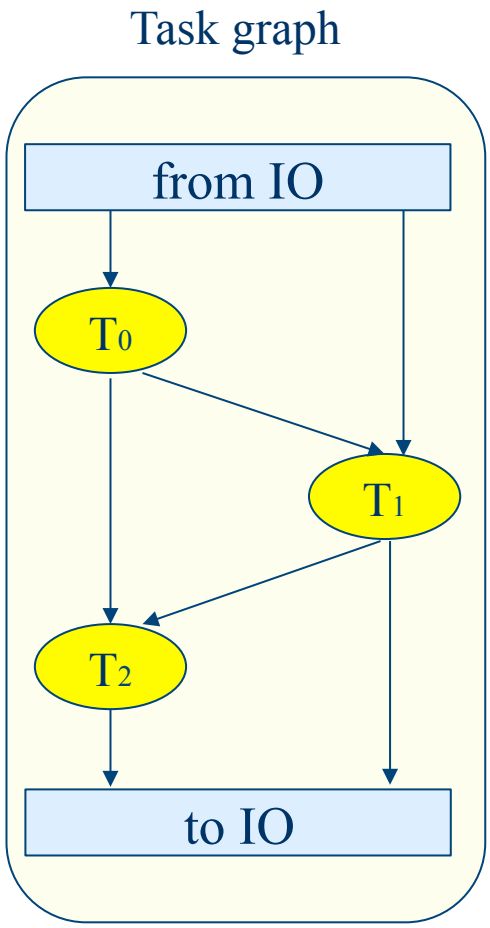


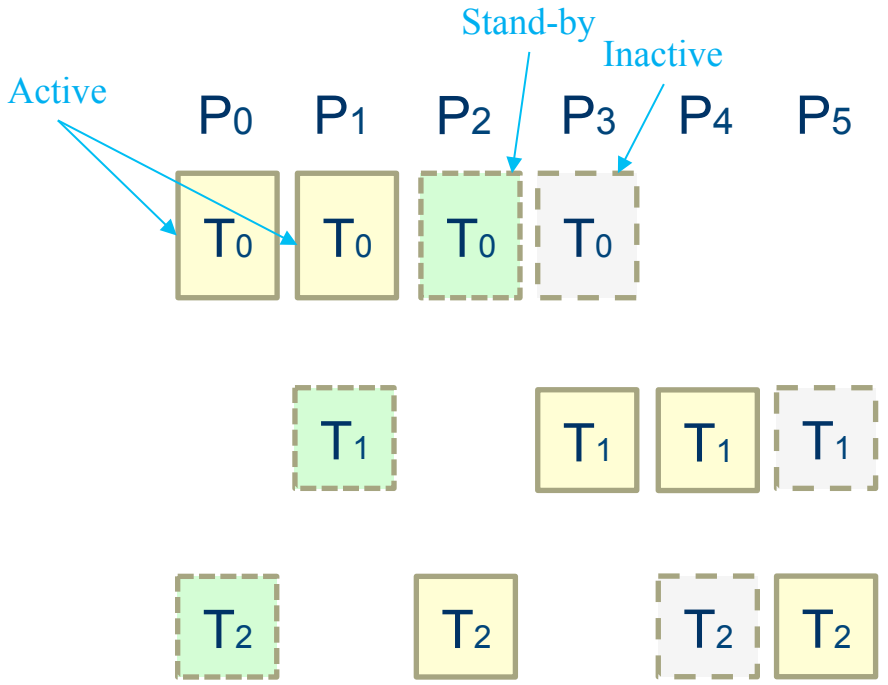
Idea is simple!

Implementation is not so simple!

- Issues to be considered
  - Copies of tasks should be loaded in several cores, considering possible faulty patterns
  - Third core should always know every information of the task execution for TMR configuration
  - For each faulty patterns, every task execution should be done within the control cycle
  - Programmers do not want to think about duplicated or triplicated task execution

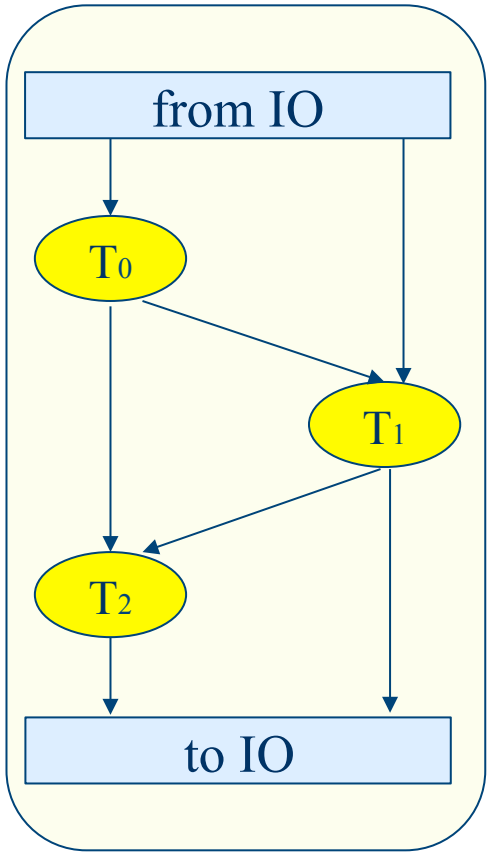
Static / Redundant Task Allocation

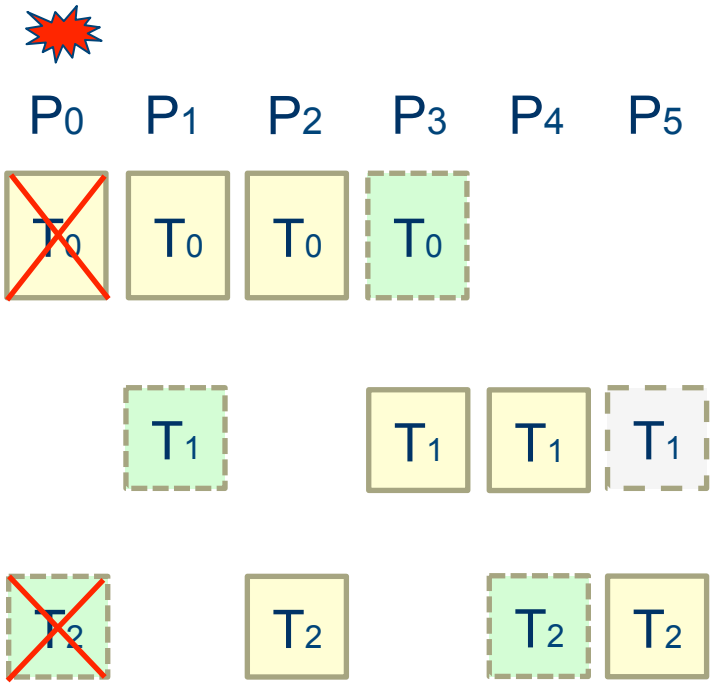




Static / Redundant Task Allocation

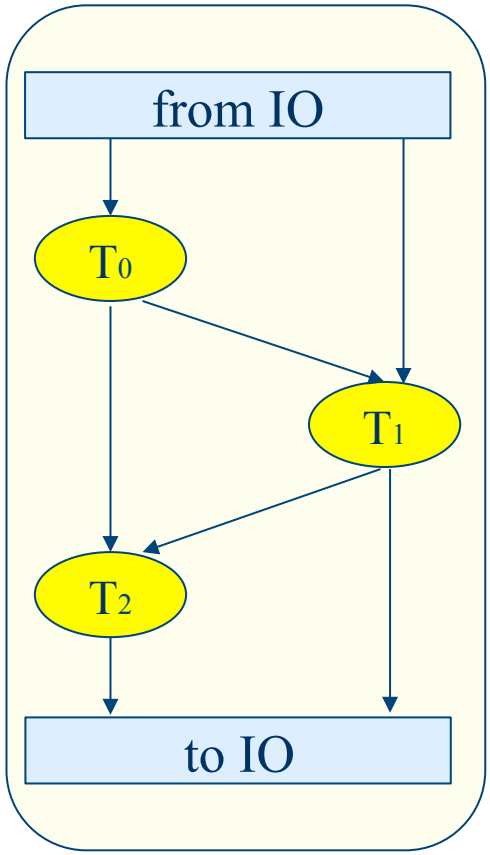
Task graph

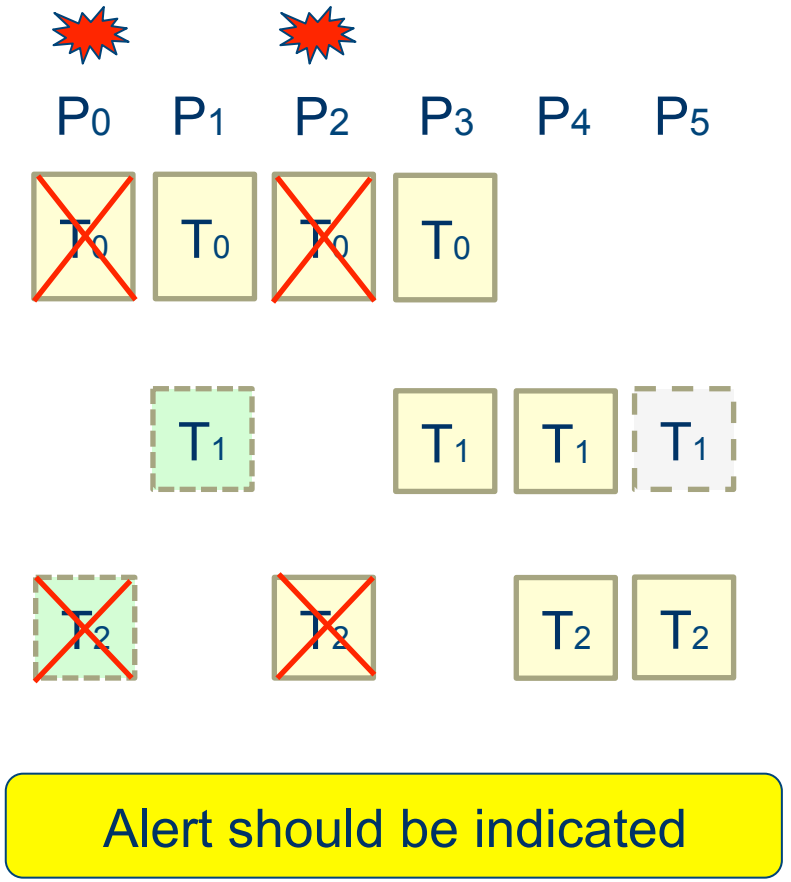




Static / Redundant Task Allocation

Task graph

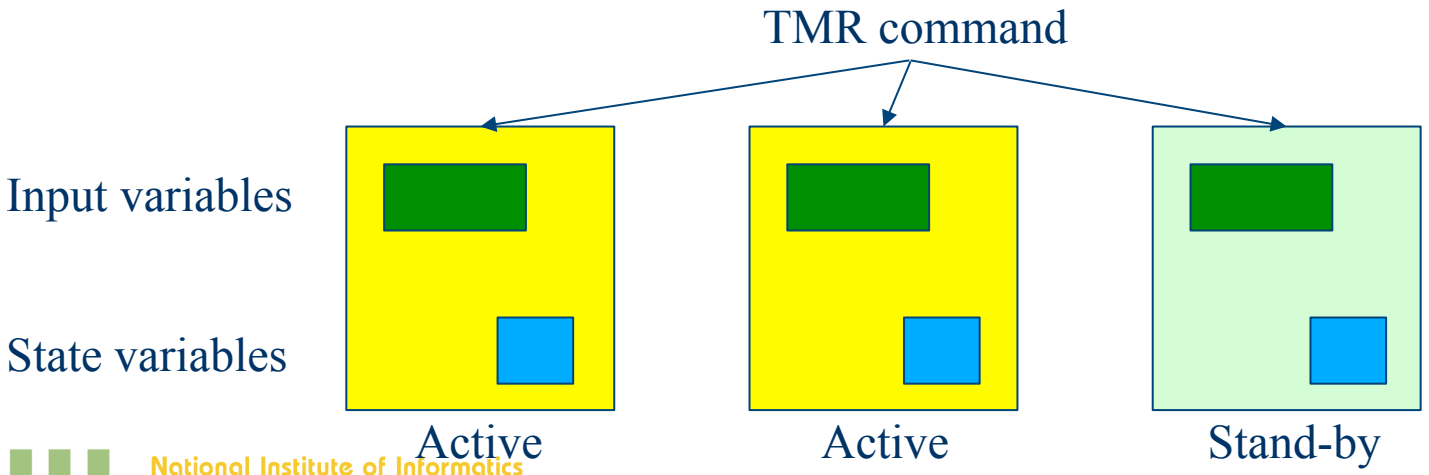




Alert should be indicated

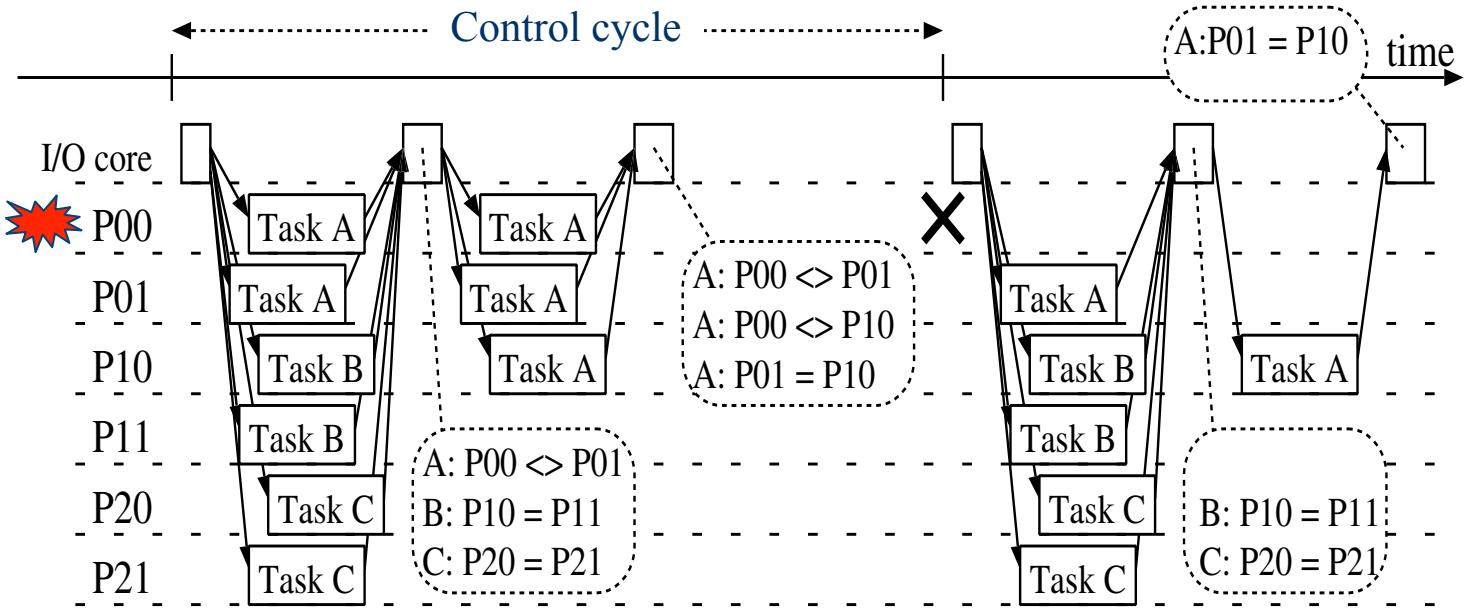
Temporary TMR configuration

- Active tasks are also re-executed
  - Transient errors can be masked
- Only “TMR command” is given
  - Quick execution is possible
  - IO core does not have to keep data



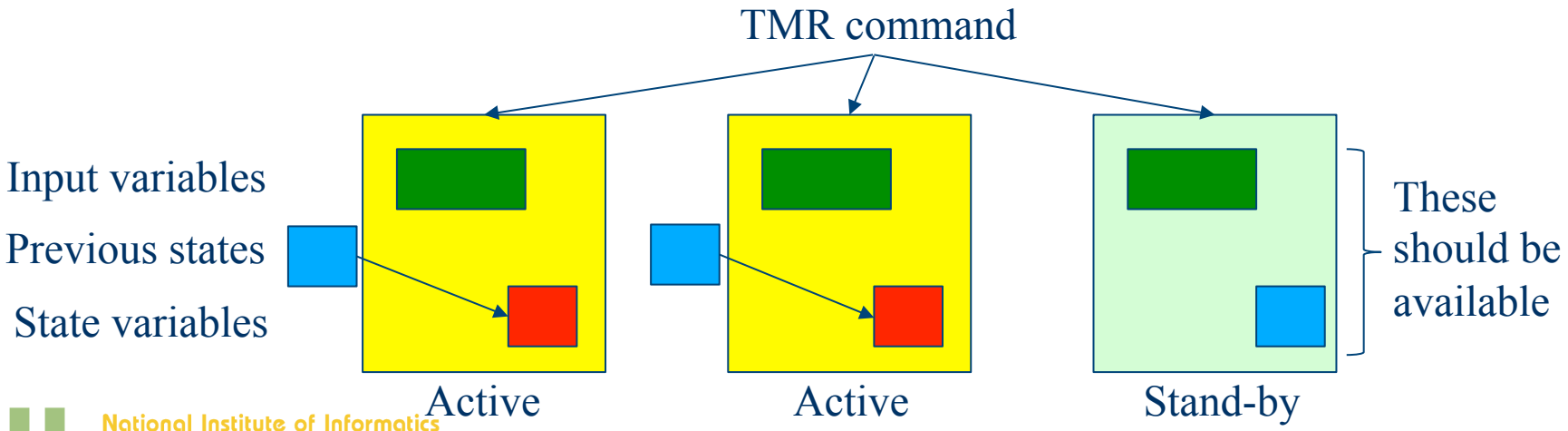
Modified Pair & Swap

- Duplicated execution, comparison, and pair-reconfiguration



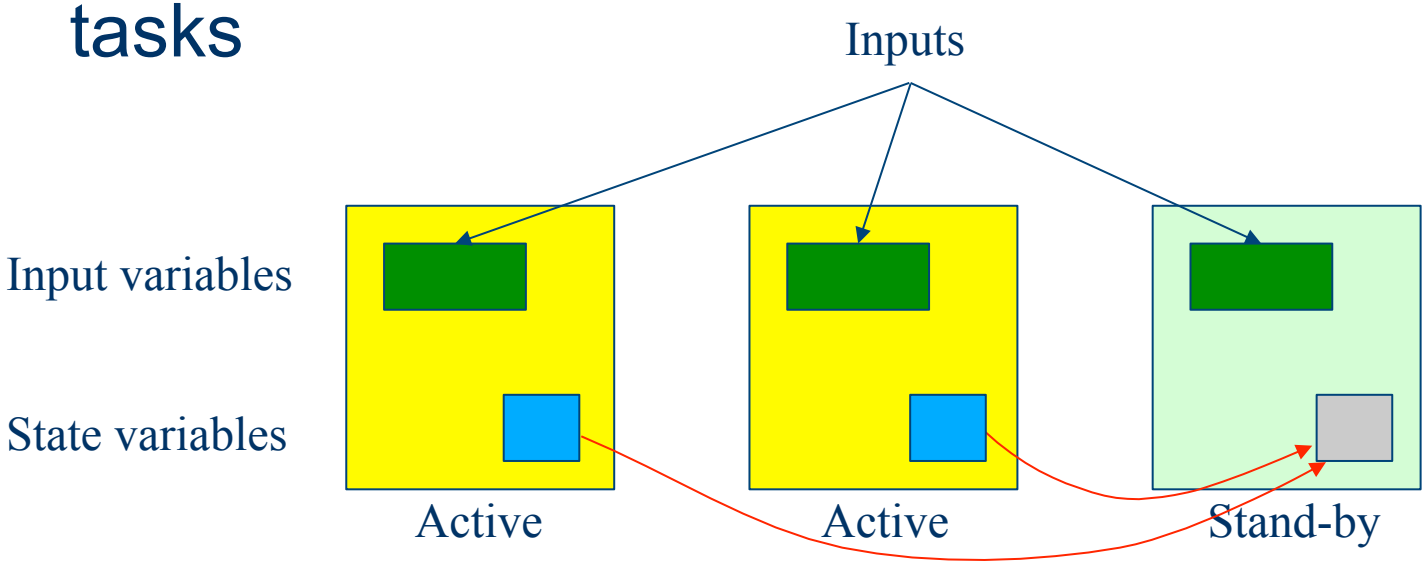
Temporary TMR configuration

- Active tasks
  - should roll back their state variables
- Stand-by task
  - should have correct input variables and state variables

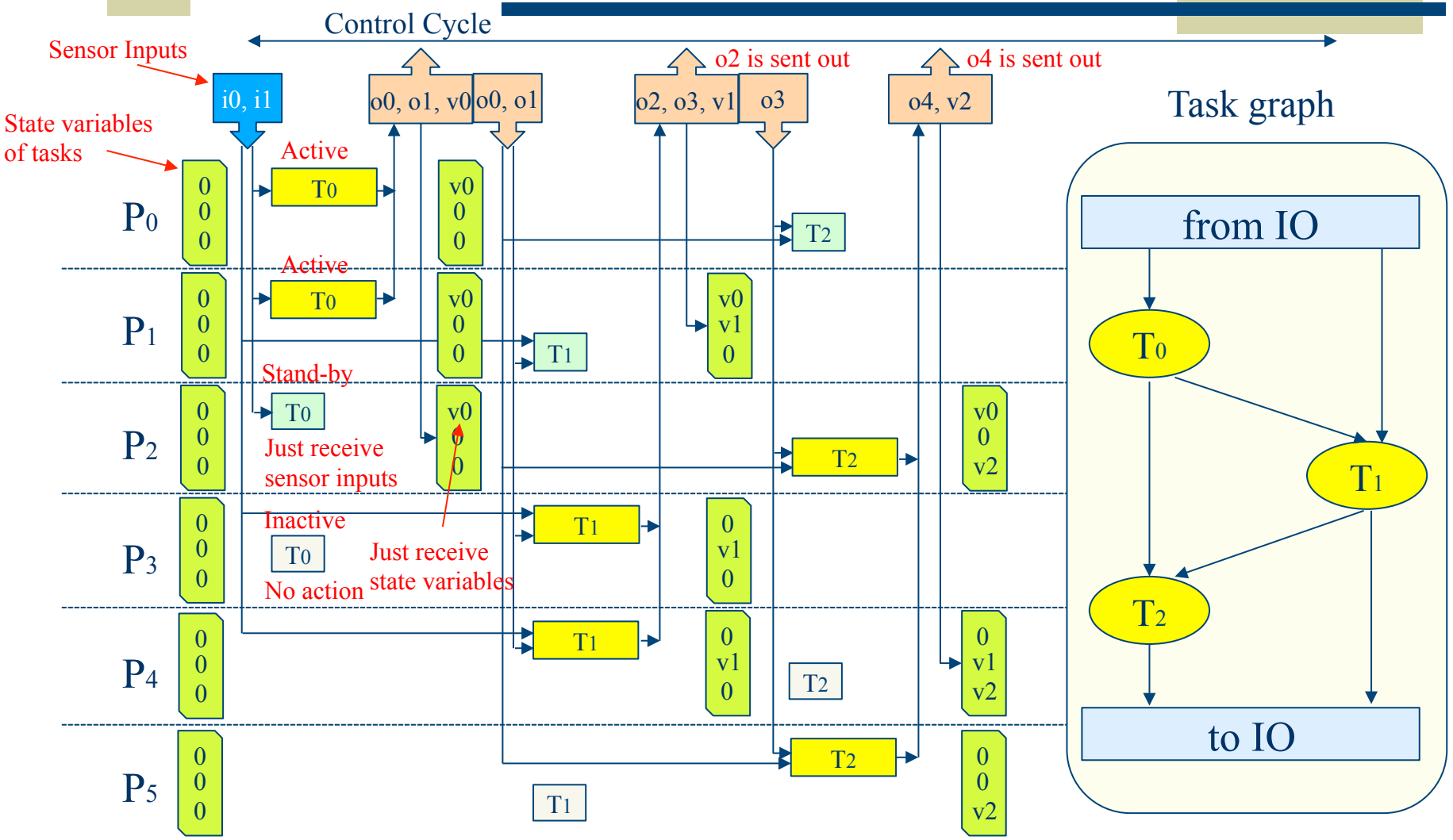


Temporary TMR configuration

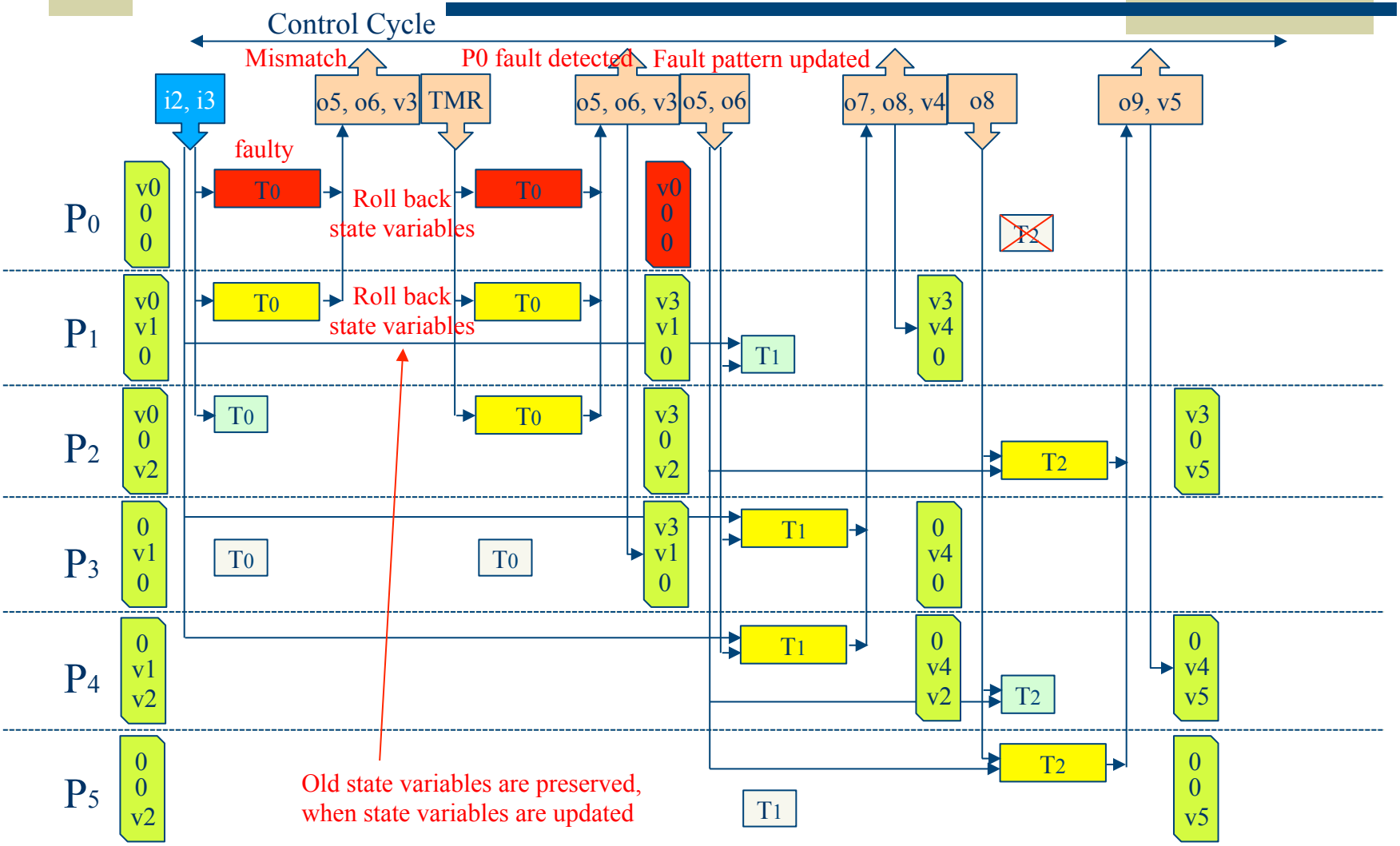
- To prepare for TMR configuration, stand-by task usually
  - Receives all input data given to active tasks
  - Receives the state variables updated by active tasks



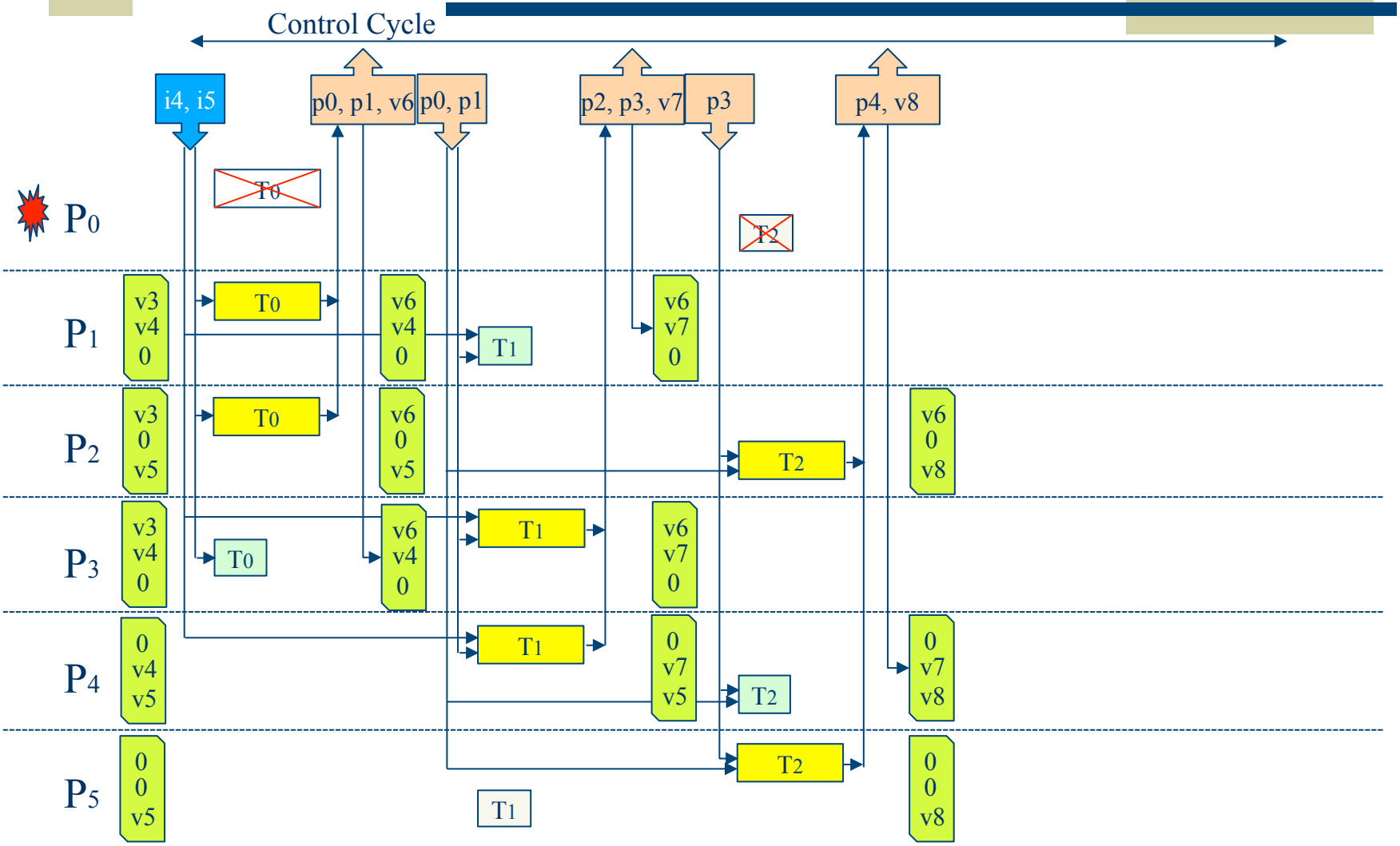
Example of task execution



Example of task execution



Example of task execution



Implementation is not so simple!

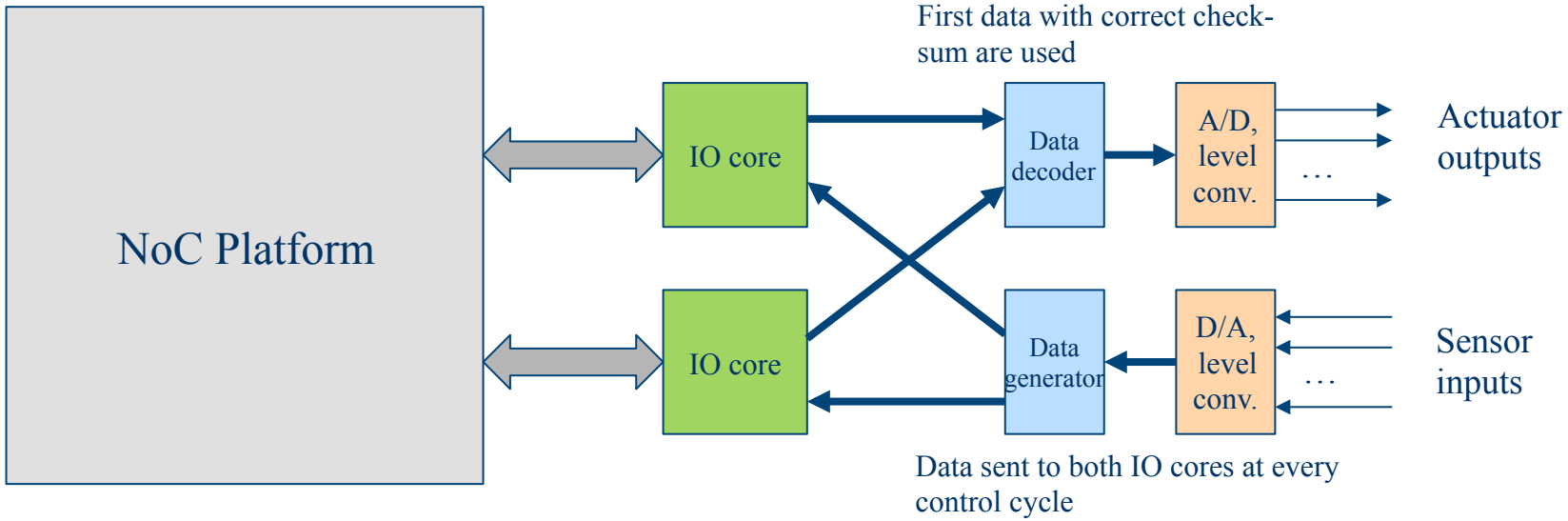
- Issues to be considered
  - Copies of tasks should be loaded in several cores, considering possible faulty patterns
  - Third core should always know every information of the task execution for sudden TMR configuration
  - For each faulty patterns, every task execution should be done within the control cycle
  - Programmers do not want to think about duplicated or triplicated task execution

Tool support

- Given by users
  - Simplex simulink program
  - Task declaration (by specifying atomic subsystems)
  - # of task copies allocated
  - # of processer cores available
- Front-end GUI tool supports
  - Allocation of multiple task copies to redundant processor cores with timing and memory constraints
- Back-end tool supports (ongoing work)
  - C code generation for simulink codes
  - Wrapper code templates for receiving and sending data as well as handling TMR configuration

IO core duplication (ongoing work)

- IO core plays simple but important roles
  - Implemented by hardware or a small processor
  - Simple crash fault assumed
    - Fixed duplex configuration



Summary

- For our dependable NoC based platform
  - Implementation of dependable task execution (i.e., modified Pair & Swap)
  - Task allocation for modified Pair & Swap
    - Front-end tool
- Ongoing work
  - Back-end tool
  - Implementation of duplicated IO core