Scheduling complex streaming applications on the Cell processor

Mathias Jacquelin,

joint work with Matthieu Gallet and Loris Marchal

INRIA ROMA project-team
LIP (ENS-Lyon, CNRS, INRIA)
École Normale Supérieure de Lyon, France

Workshop on Multithreaded Architectures and Applications,
Atlanta, April 23, 2010.

Outline

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the ROMA team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the ROMA team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the ROMA team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques

Introduction: Steady-state Scheduling

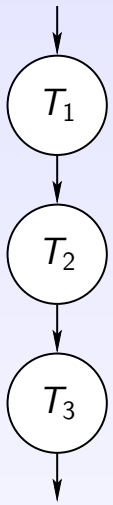
Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource

Introduction: Steady-state Scheduling

Rationale:

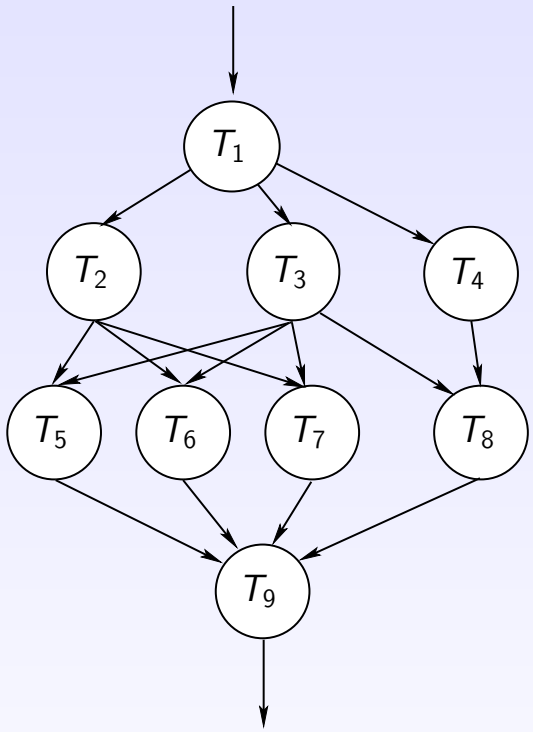
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

Rationale:

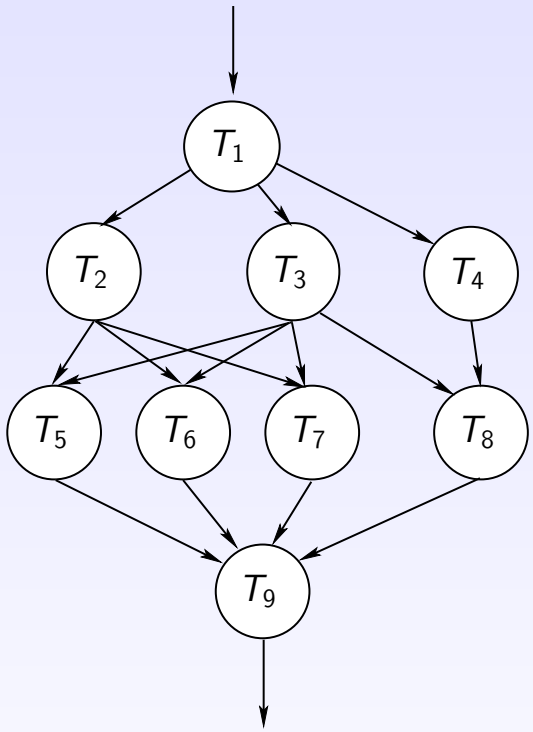
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

Rationale:

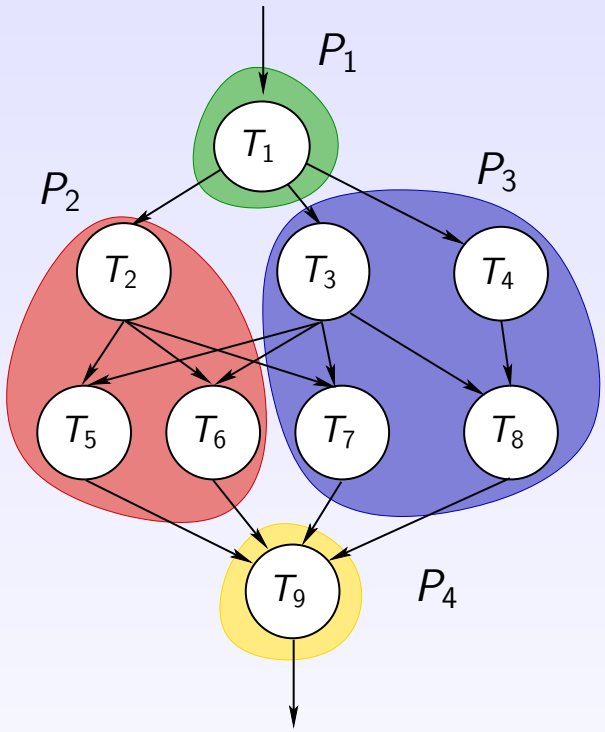
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

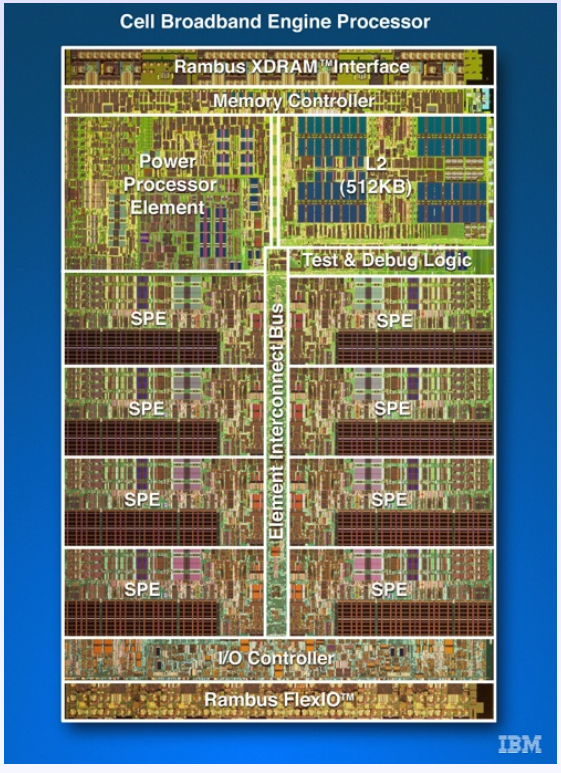
Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



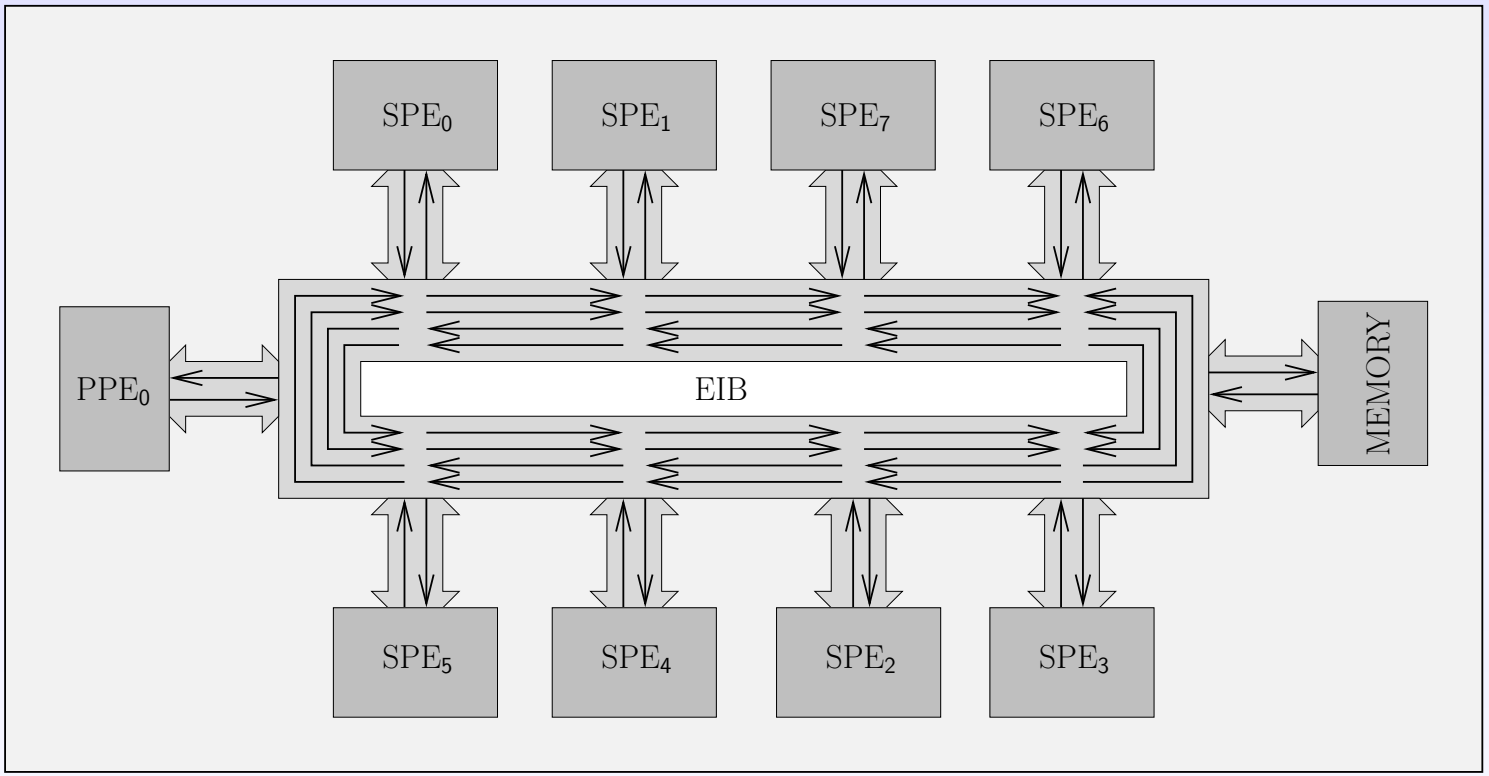
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



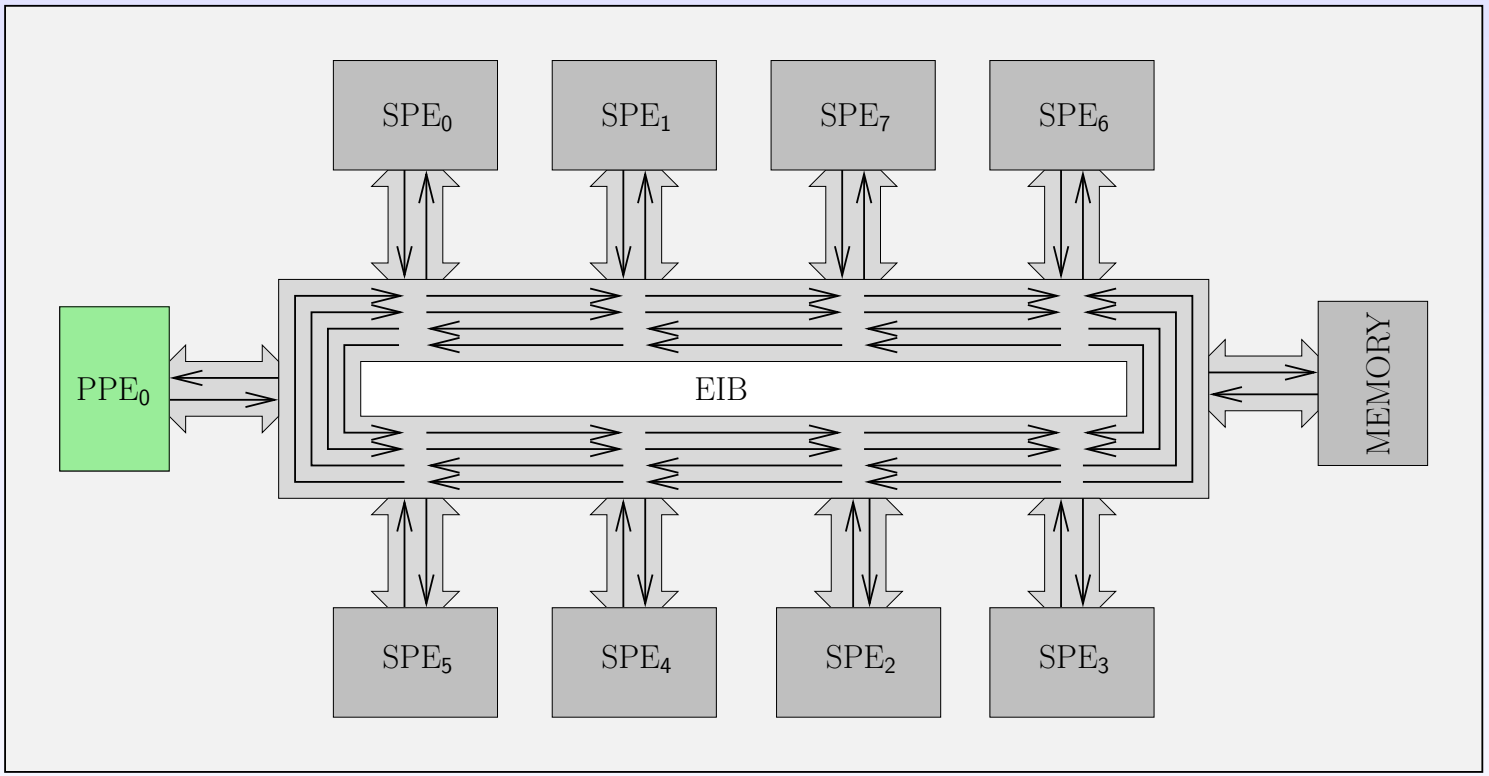
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

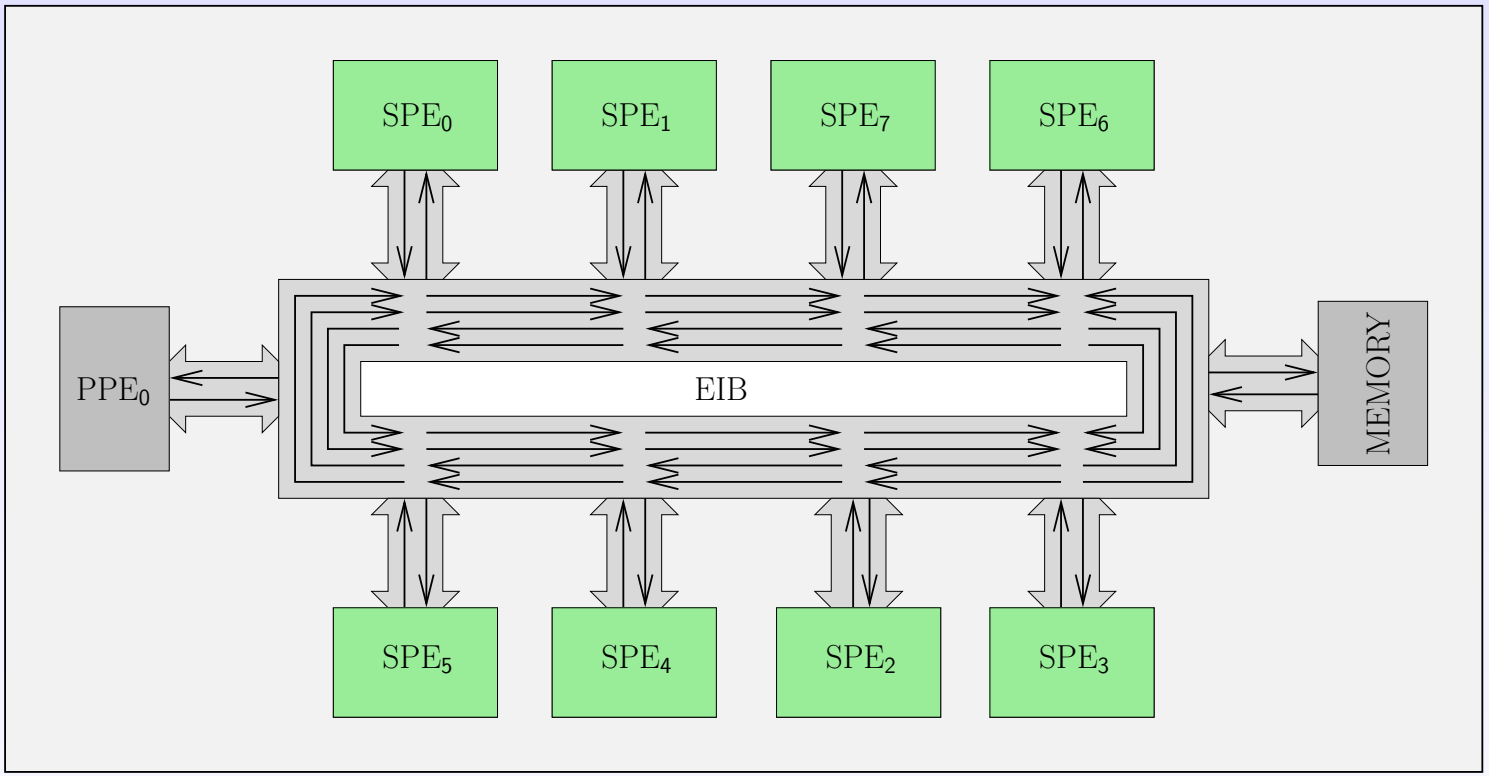
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 1 PPE core
  - VMX unit
  - L1, L2 cache
  - 2 way SMT

CELL brief introduction

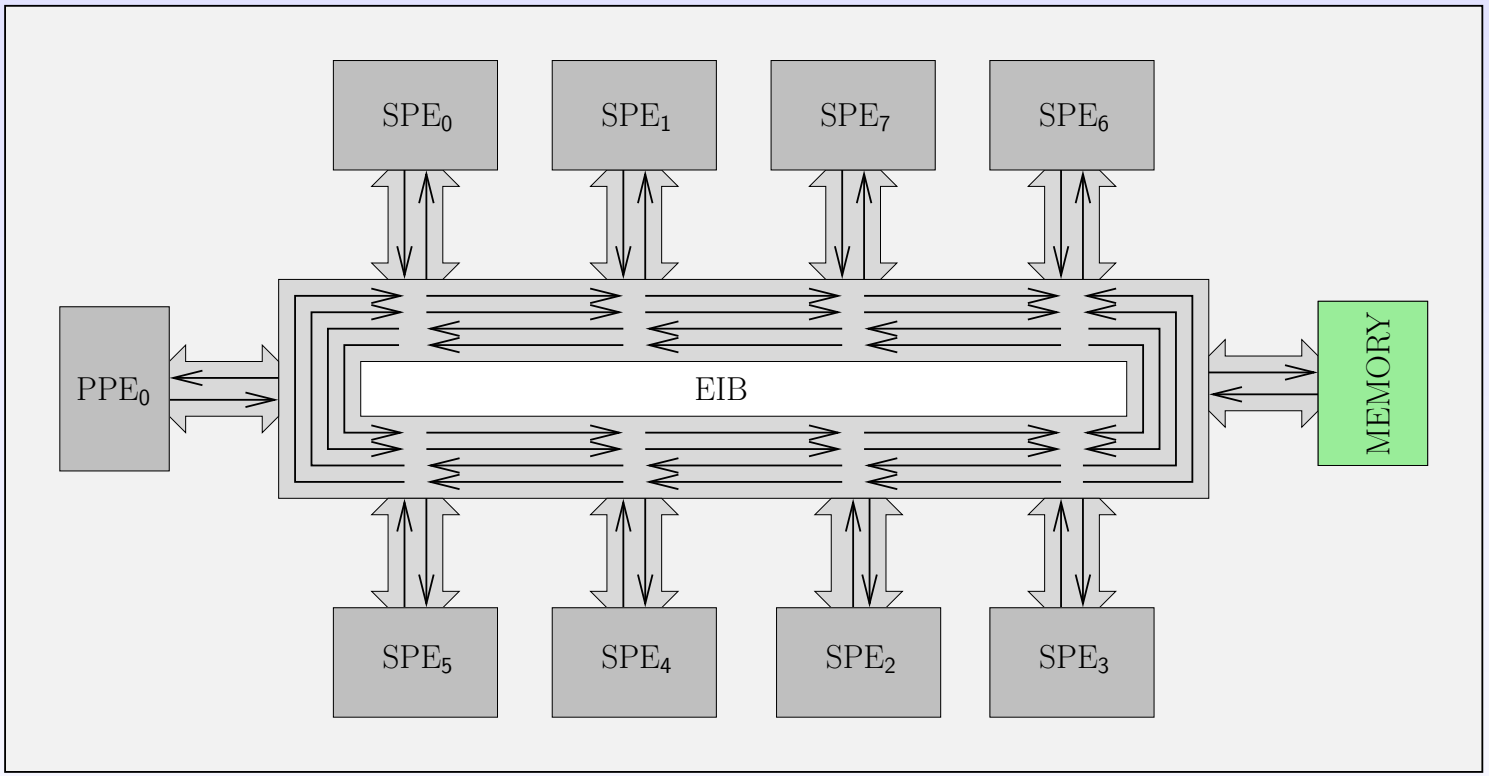
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 8 SPEs
  - 128-bit SIMD instruction set
  - Local store 256KB
  - Dedicated Asynchronous DMA engine

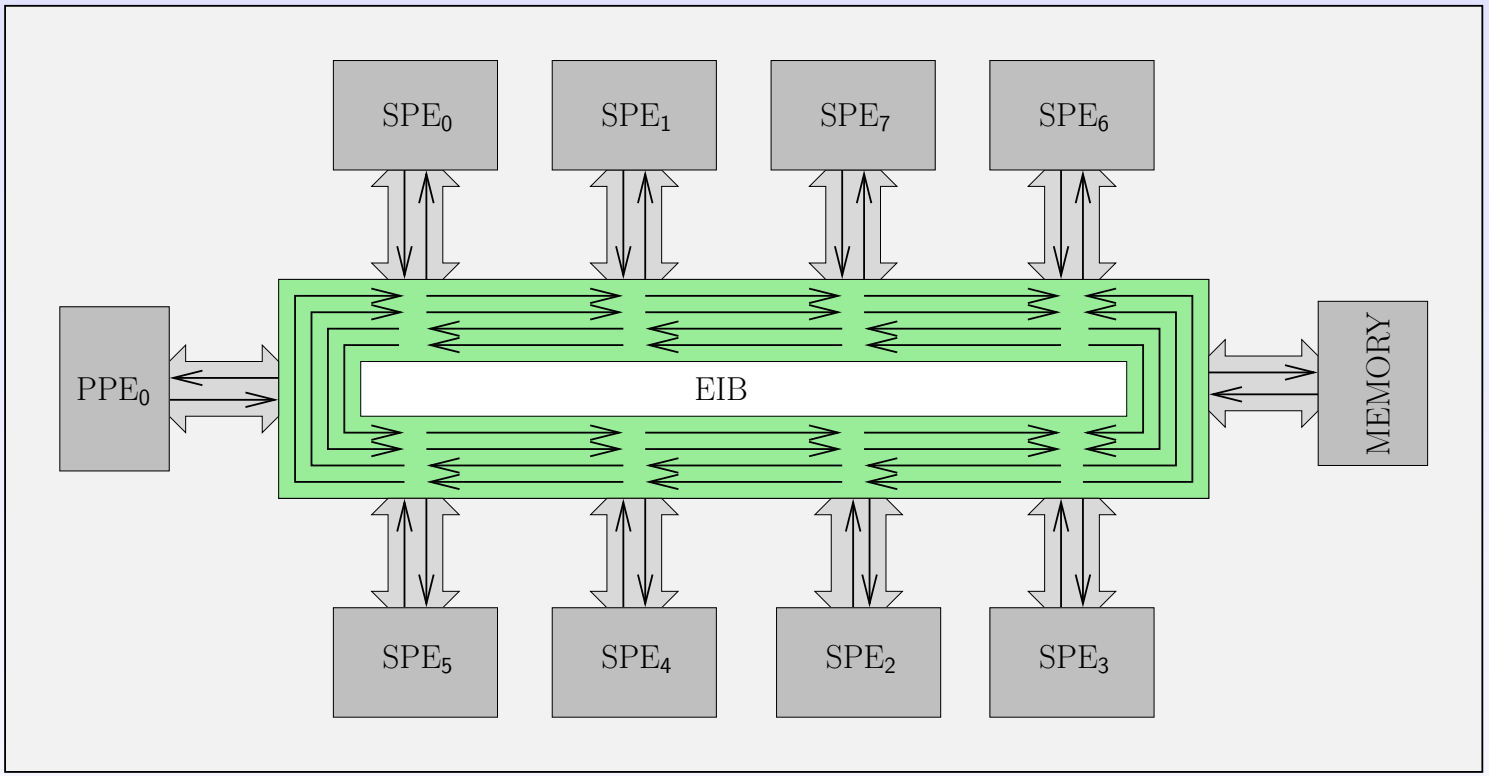
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

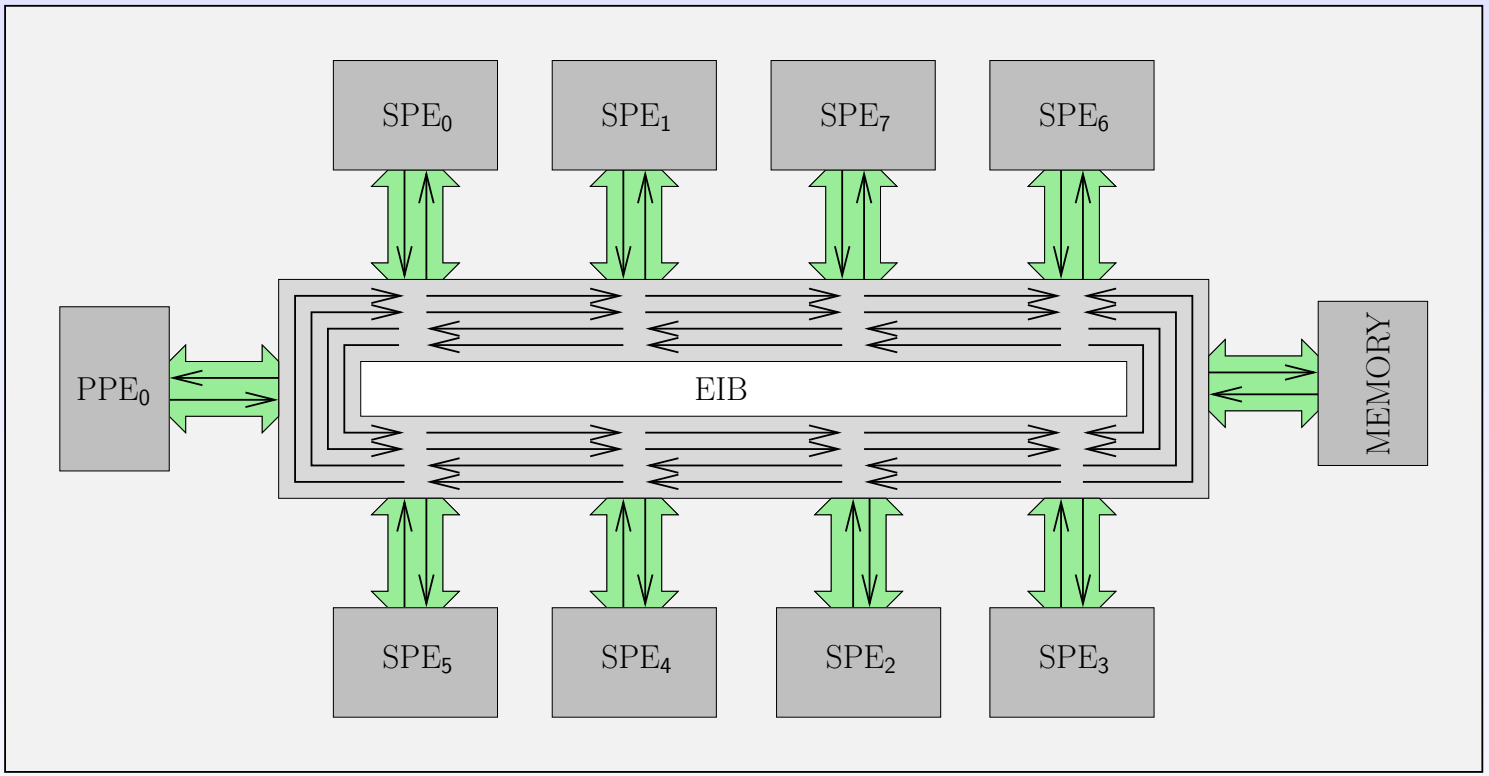
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- Element Interconnect Bus (EIB)
  - 200 GB/s bandwidth

CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 25 GB/s bandwidth

Outline

Platform modeling

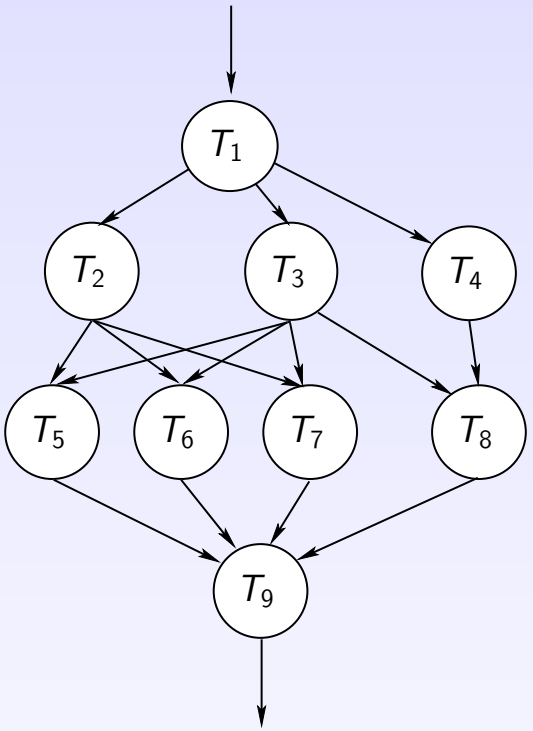
Simple CELL modeling:

- 1 PPE and 8 SPE: 9 processing elements $P_1, \ldots, P_9$, with *unrelated* speed,
- Each processing element access the communication bus with a (bidirectional) bandwidth $b = (25GB/s)$,
- The bus is able to route all concurrent communications without contention (in a first step),
- Due to the limited size of the DMA stack on each SPE:
  - Each SPE can perform at most 16 simultaneous DMA operations,
  - The PPE can perform at most 8 simultaneous DMA operations to/from a given SPE.
- Linear cost communication model:
  a data of size S is sent/received in time S/b

Application modeling

Application is described by a directed acyclic graph:

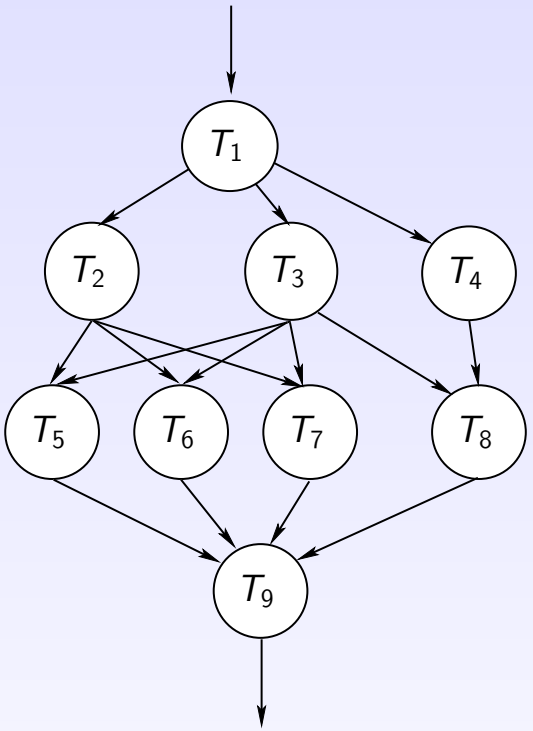
- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,
- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,



Application modeling

Application is described by a directed acyclic graph:

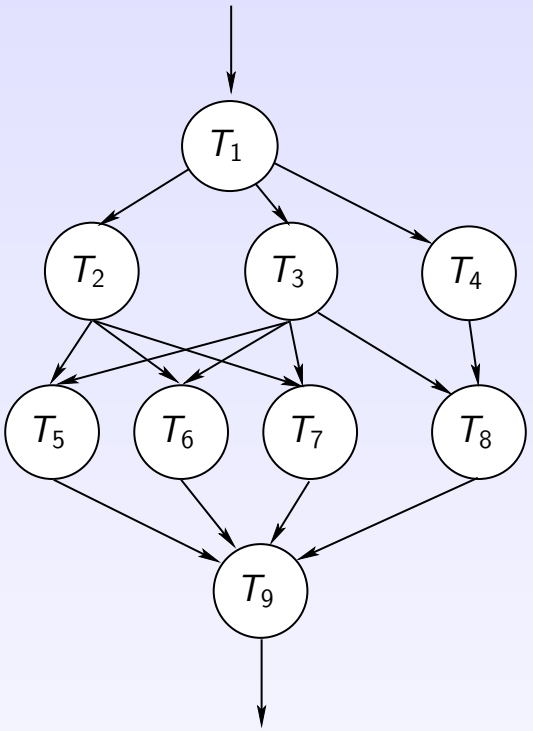
- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,
- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,



Application modeling

Application is described by a directed acyclic graph:

- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,



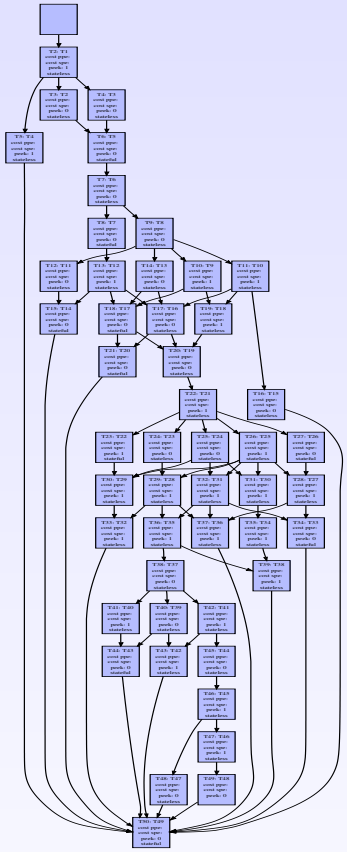
- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,

Target application: any DAG

- Today, we will focus on three random task graphs:

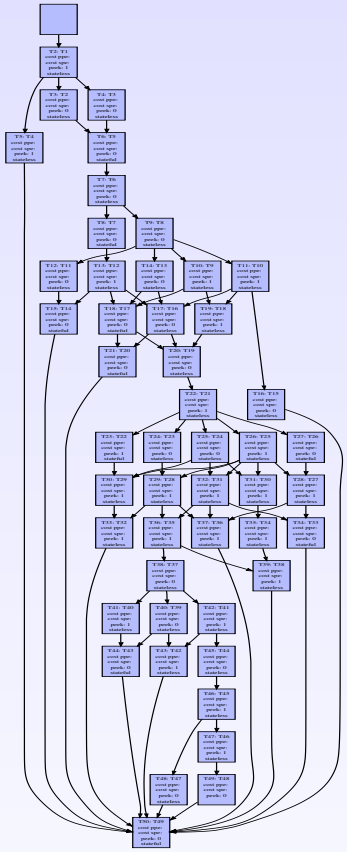
Target application: any DAG

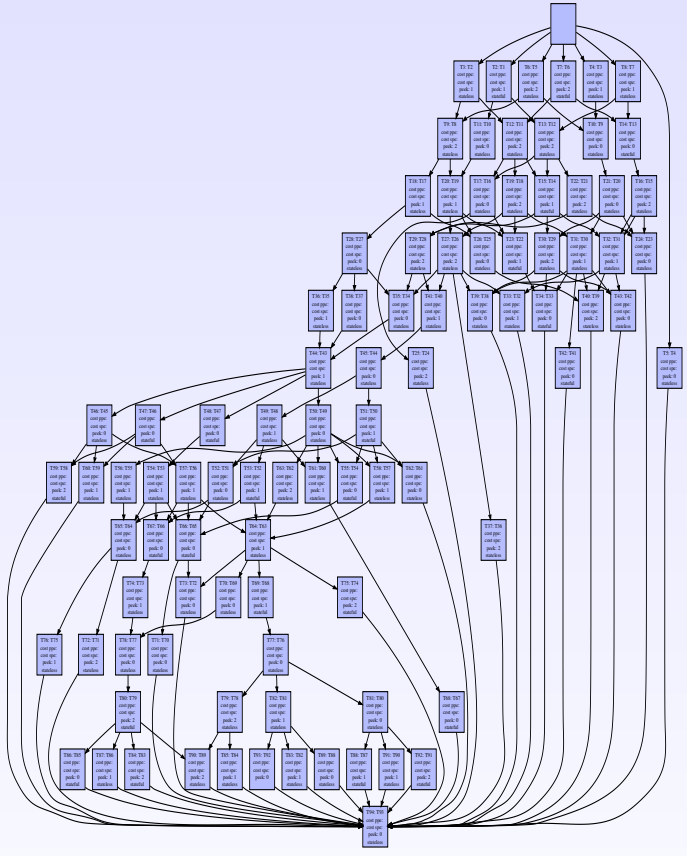
- Today, we will focus on three random task graphs:



Target application: any DAG

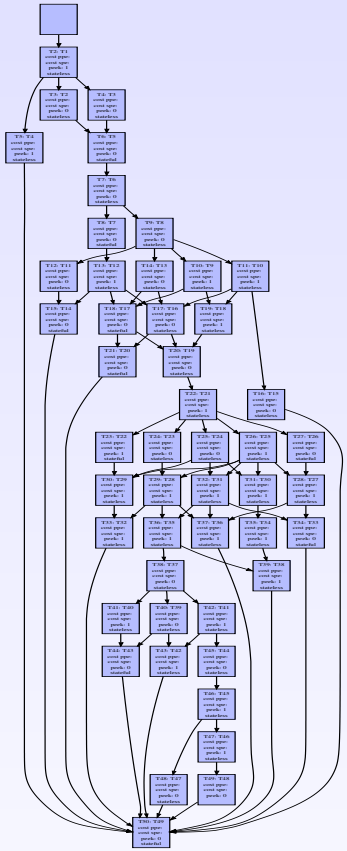
- Today, we will focus on three random task graphs:

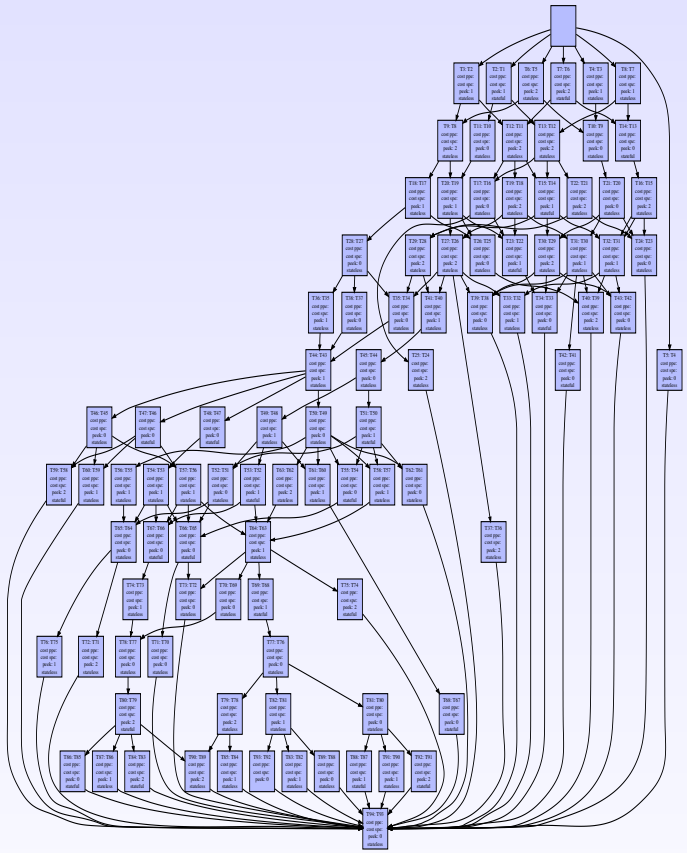




Target application: any DAG

- Today, we will focus on three random task graphs:





And a simple chain graph (50 tasks)

Outline

How to compute an optimal mapping

- Ojective: maximize throughput ρ
- Method: write a linear program gathering constraints on the mapping
- Binary variables: $\alpha_i^k = \begin{cases} 1 & \text{if } T_k \text{ is mapped on } P_i \\ 0 & \text{otherwise} \end{cases}$
- Other useful binary variables: $\beta_{i,j}^{k,l} = 1$ iff file $T_k \rightarrow T_l$ is transfered from P_i to P_j

Constraints 1/2

On the application structure:

- Each task is mapped on a processor:

$$\forall T_k \quad \sum_i \alpha_i^k = 1$$

- Given a dependency $T_k \rightarrow T_l$, the processor computing T_l must receive the corresponding file:

$$\forall (k,l) \in E, \forall P_j, \quad \sum_i \beta_{i,j}^{k,l} \geq \alpha_j^l$$

- Given a dependency $T_k \rightarrow T_l$, only the processor computing T_k can send the corresponding file:

$$\forall (k,l) \in E, \forall P_i, \quad \sum_j \beta_{i,j}^{k,l} \leq \alpha_i^k$$

Constraints 2/2

On the achievable throughput $\rho = 1/T$:

- On a given processor, all tasks must be completed within T:

$$\forall P_i, \quad \sum_k \alpha_i^k \times t_i(k) \leq T$$

- All incoming communications must be completed within T:

$$\forall P_j, \quad \frac{1}{b}\Big(\sum_k \alpha_j^k \times \text{read}_k + \sum_{k,l}\sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l}\Big) \leq T$$

- All outgoing communications must be completed within T:

$$\forall P_i, \quad \frac{1}{b}\Big(\sum_k \alpha_i^k \times \text{write}_k + \sum_{k,l}\sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l}\Big) \leq T$$

+ constraints on the number of incoming/outgoing communications to respect the DMA requirements
+ constraints on the available memory on SPE

Optimal mapping computation

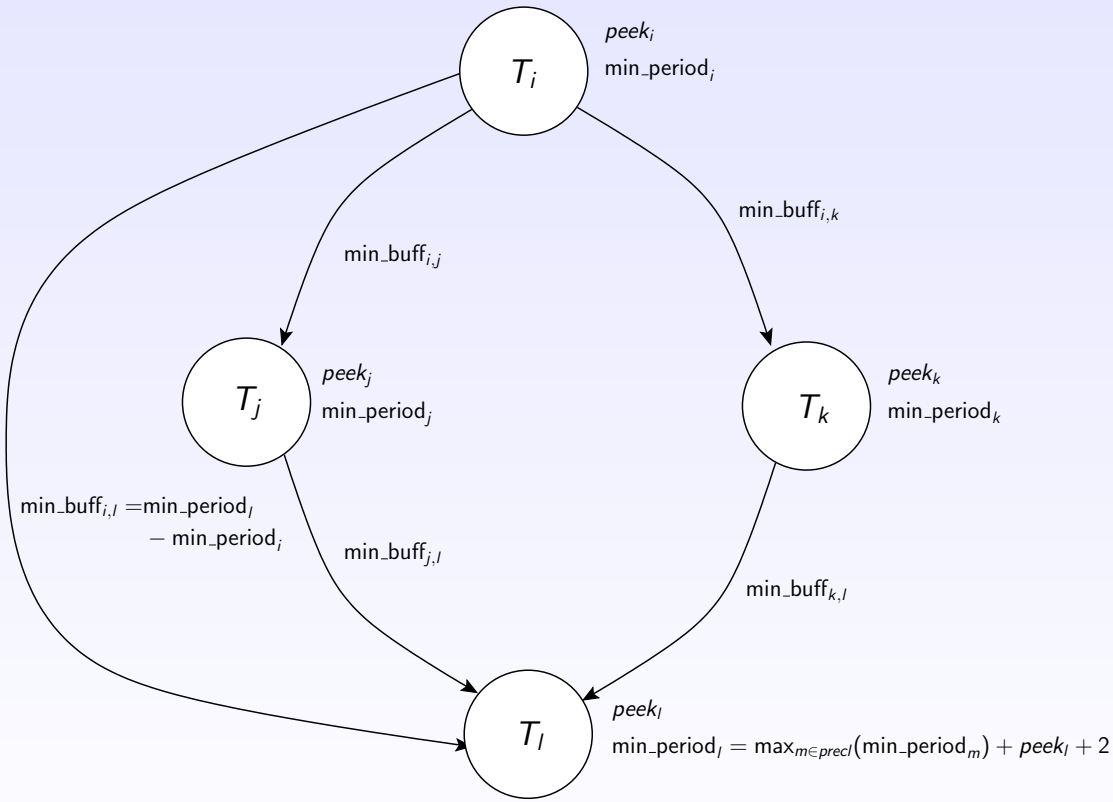
- Linear program with the objective of minimizing T
- Integer (binary) variables: Mixed Integer Programming
- NP-complete problem
- Efficient solvers exist with short running time
  - for small-size problems
  - or when an approximate solution is searched
- We use CPLEX, and look for an approximate solution (5% of the optimal throughput is good enough)

Outline

Preprocessing of the schedule

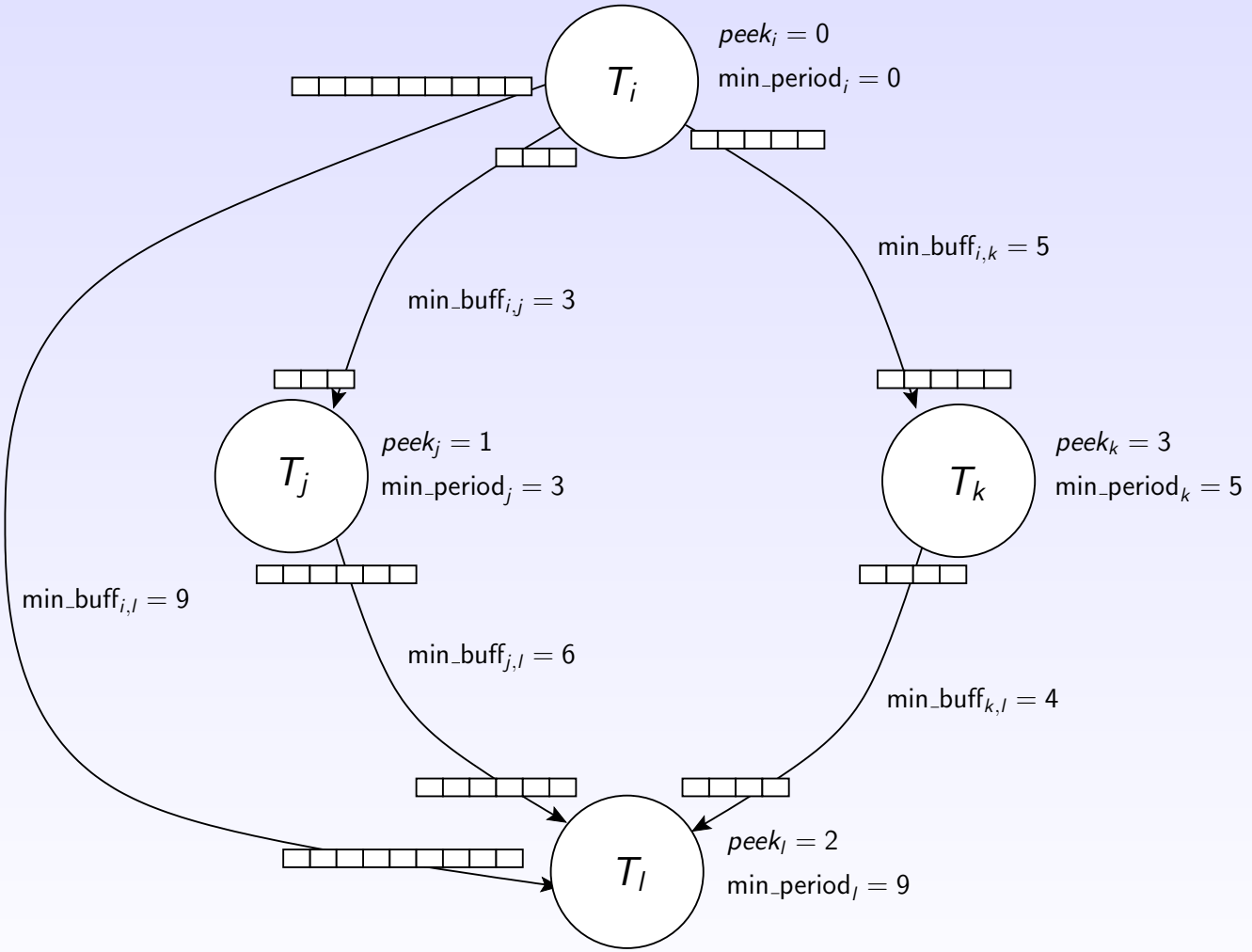
Main Objective: Compute minimal starting period and buffer sizes.

- $\text{min\_period}_l = \max_{m \in \text{prec}l}(\text{min\_period}_m) + \text{peek}_l + 2$
- $\text{min\_buff}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



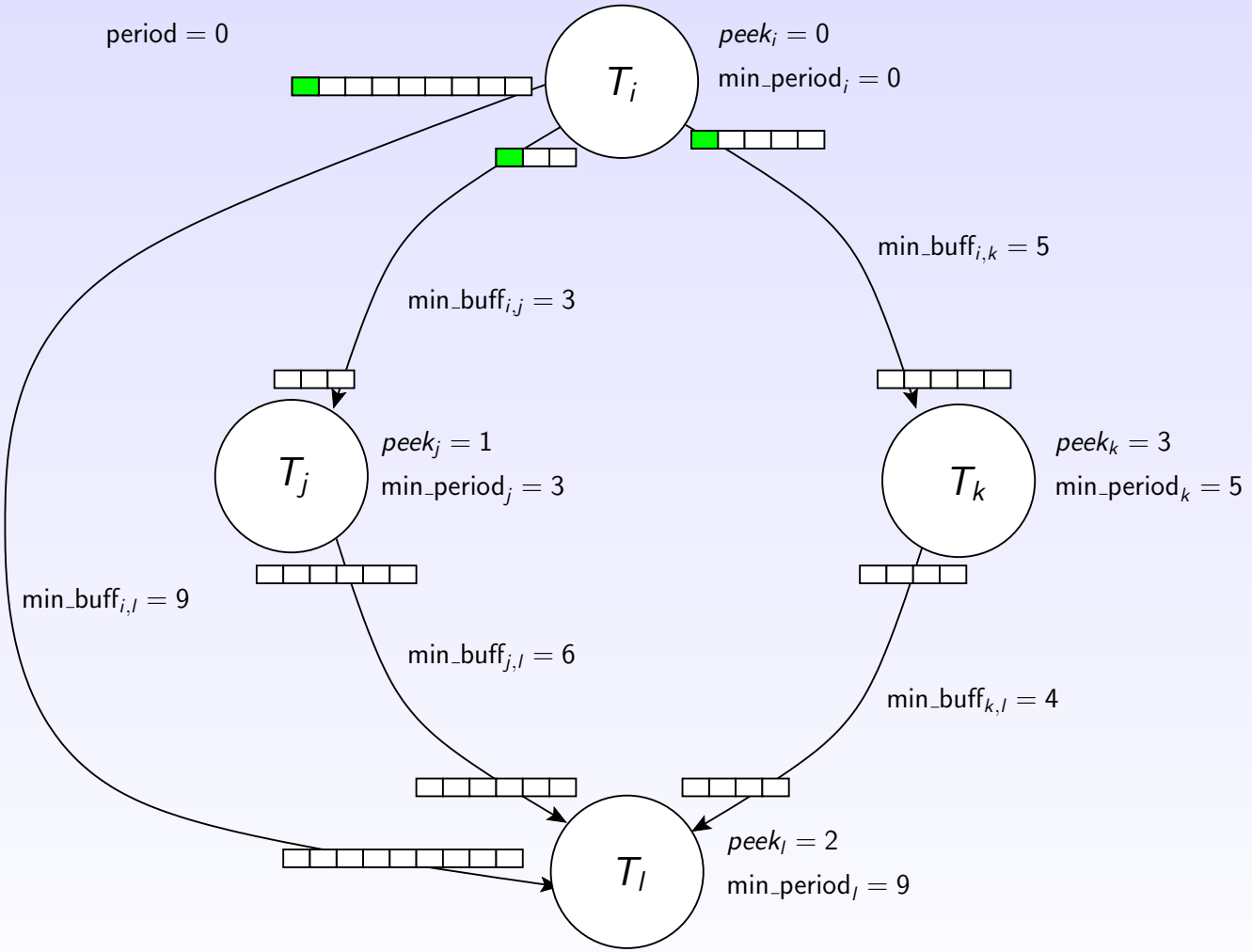
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



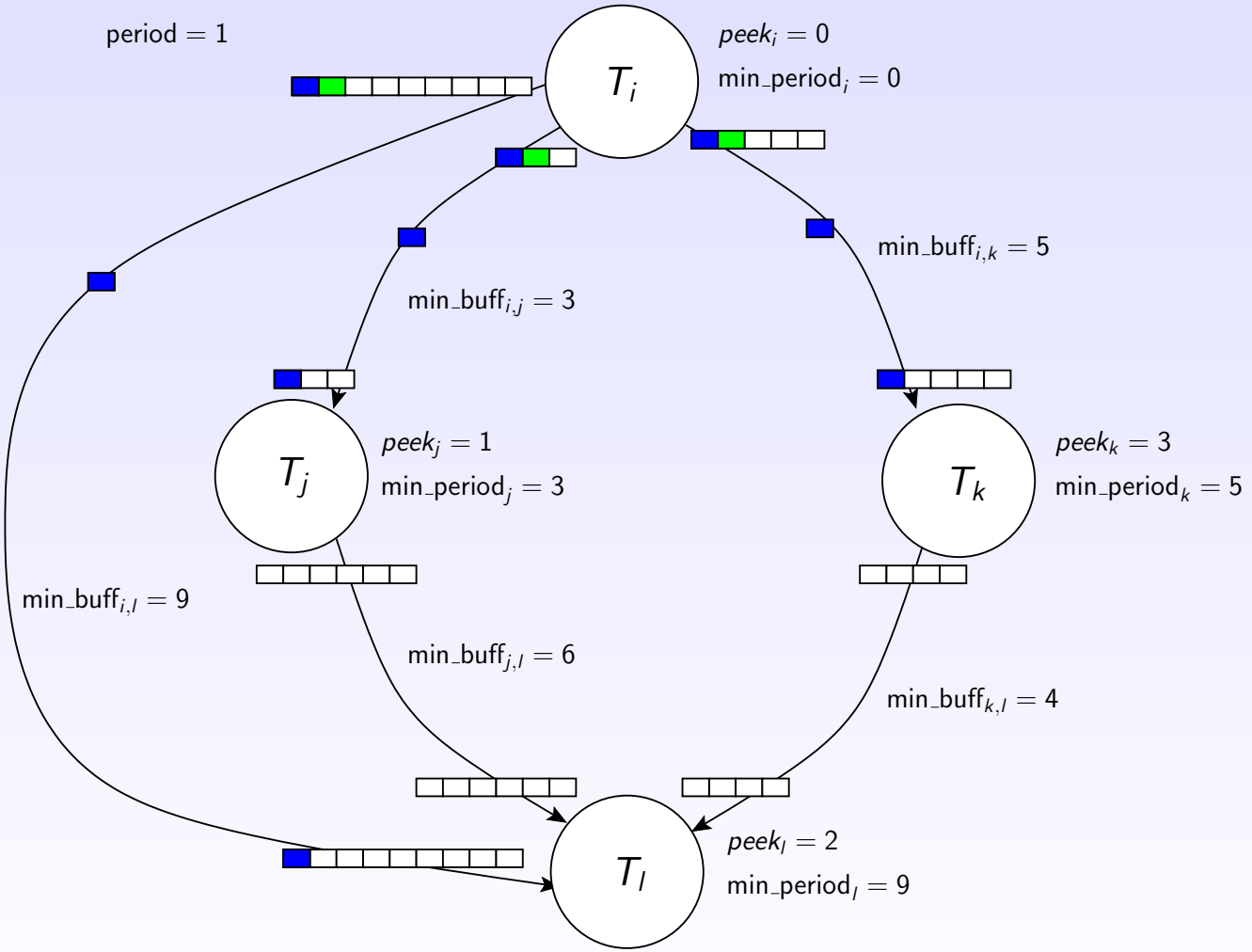
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



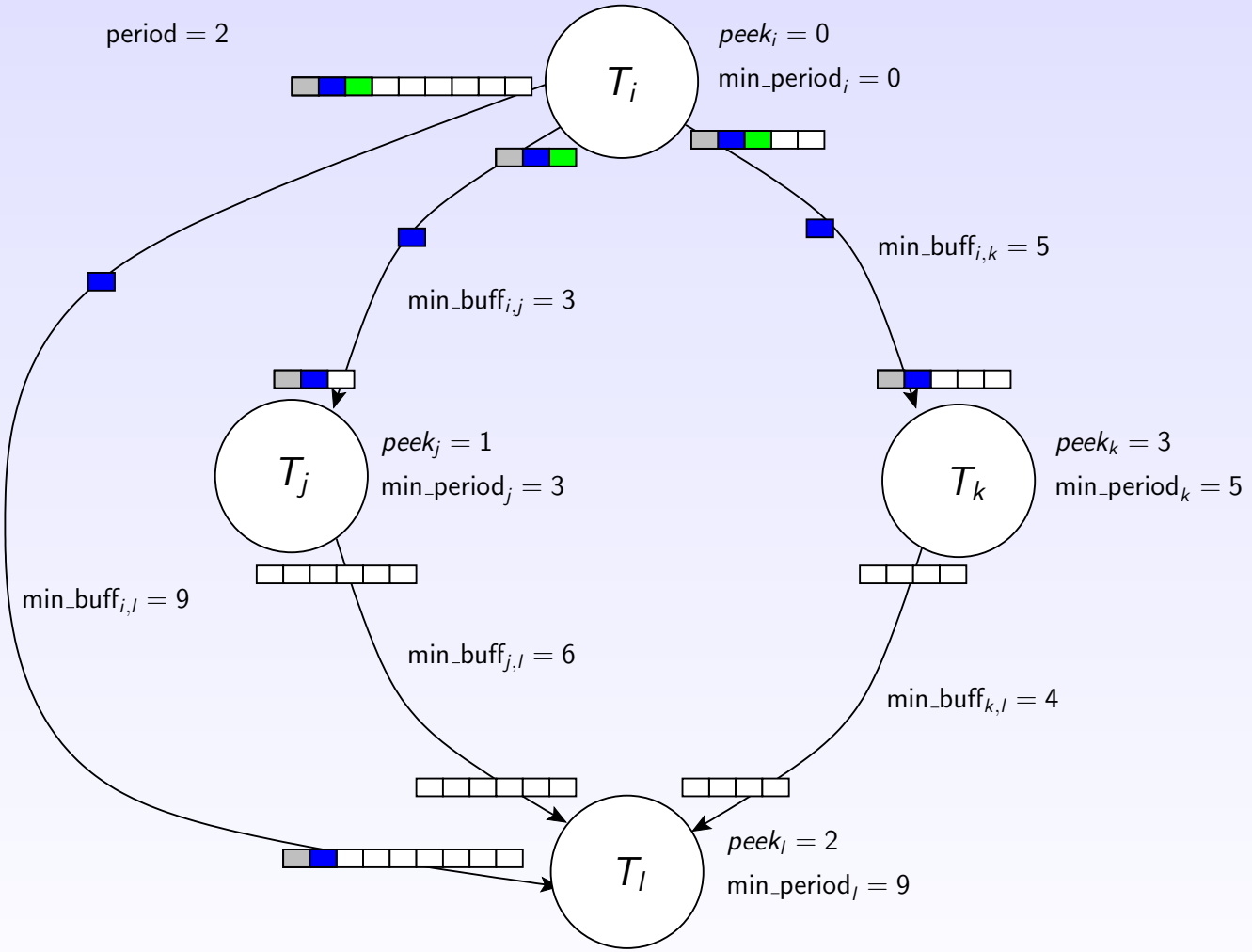
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



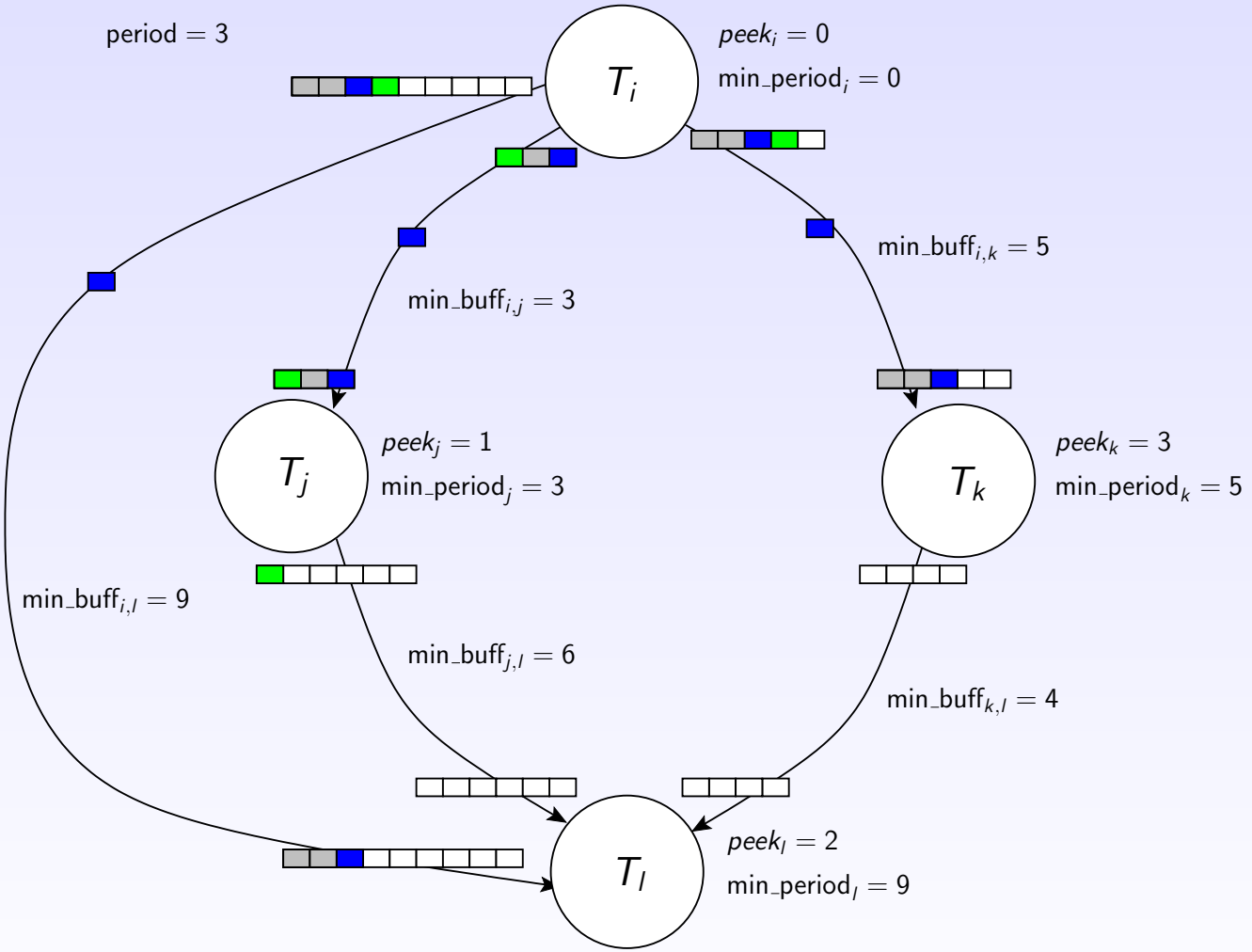
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



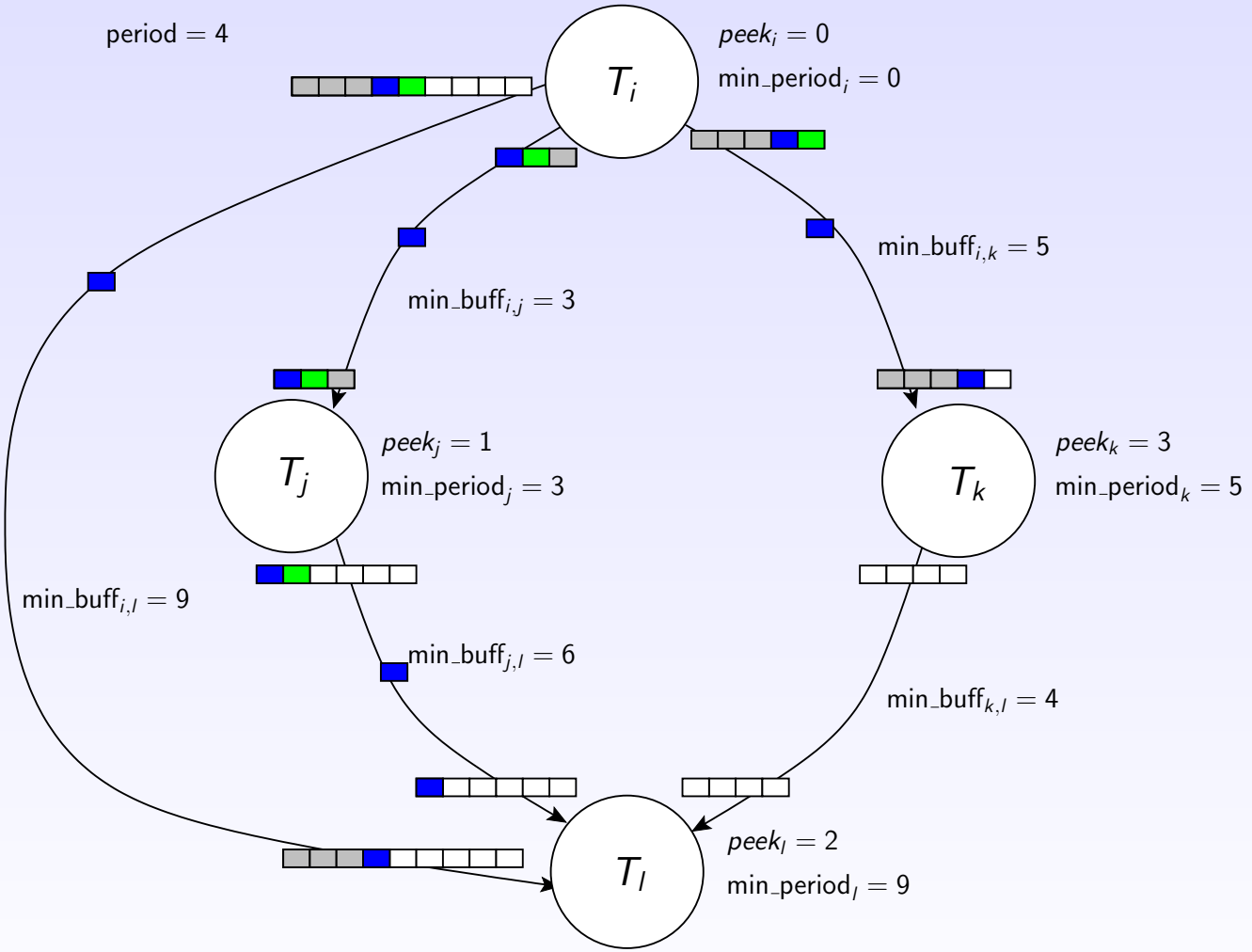
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



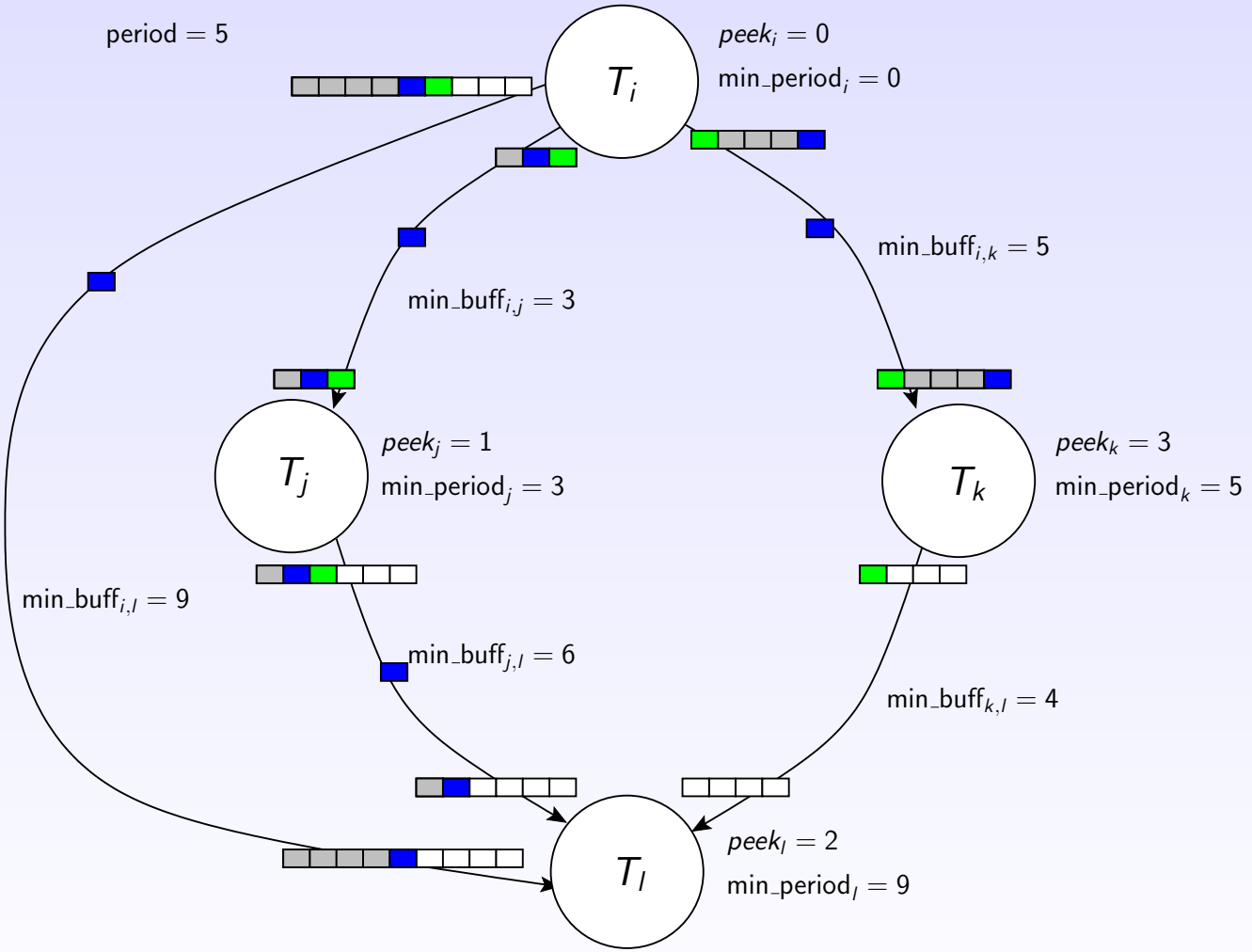
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



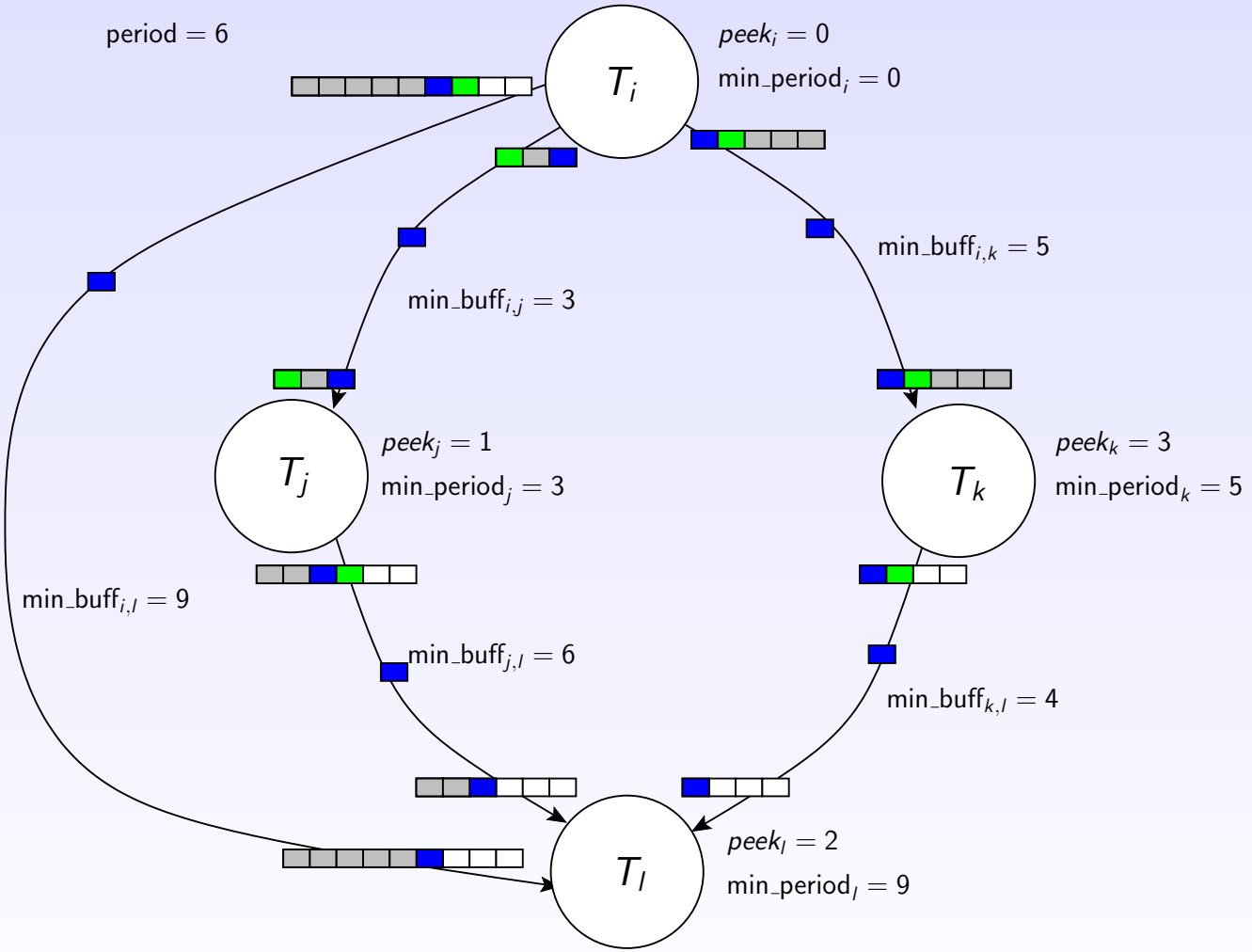
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



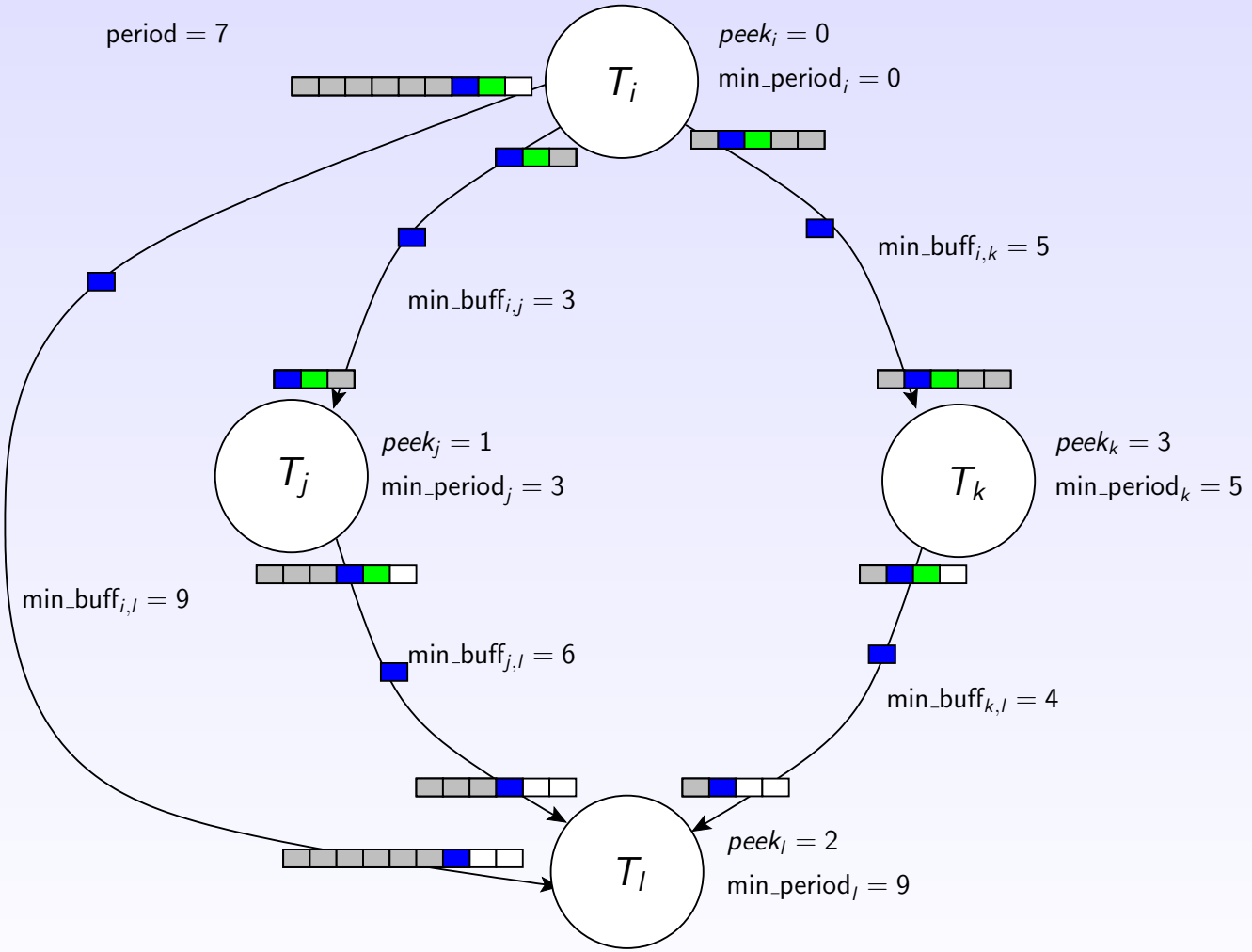
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



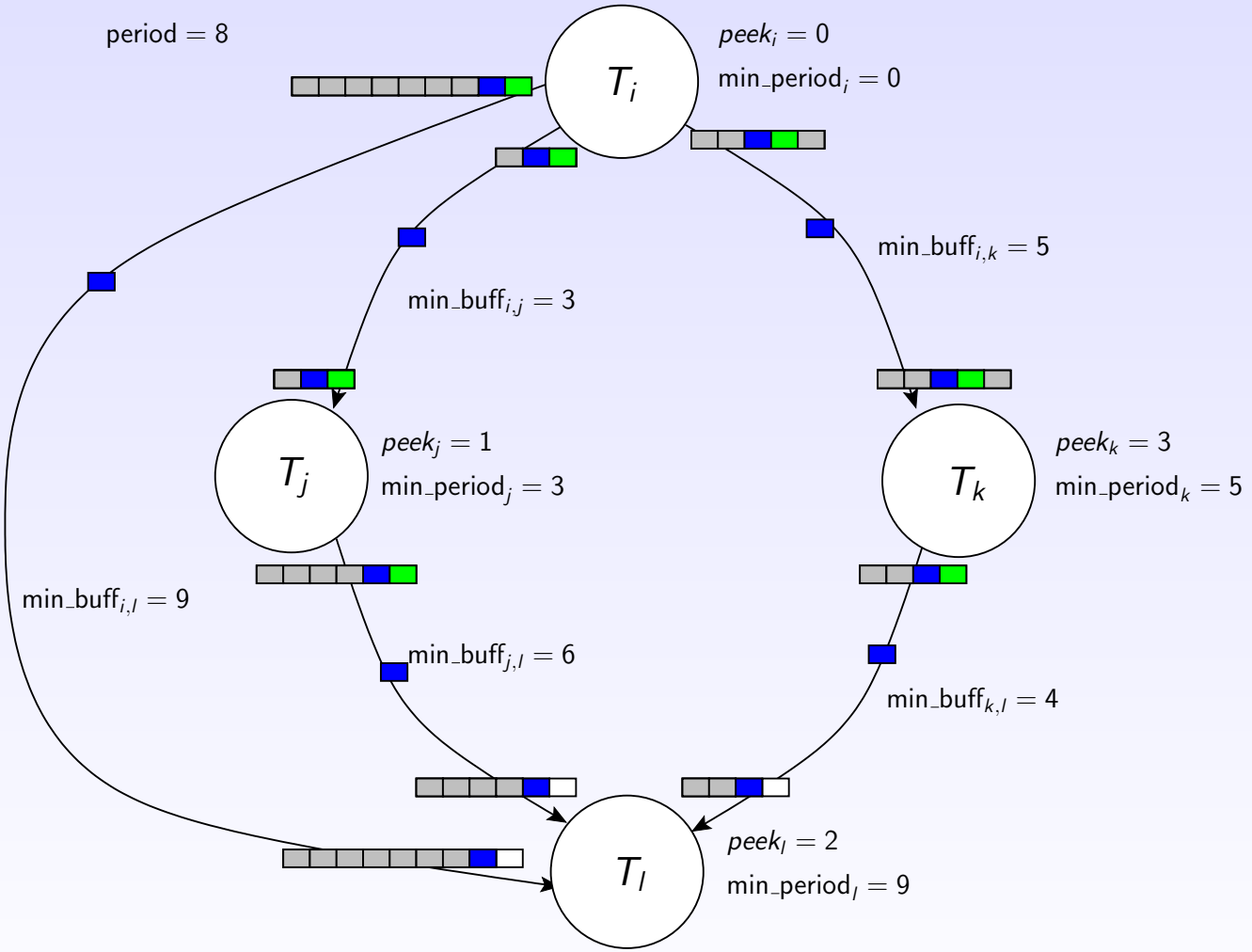
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



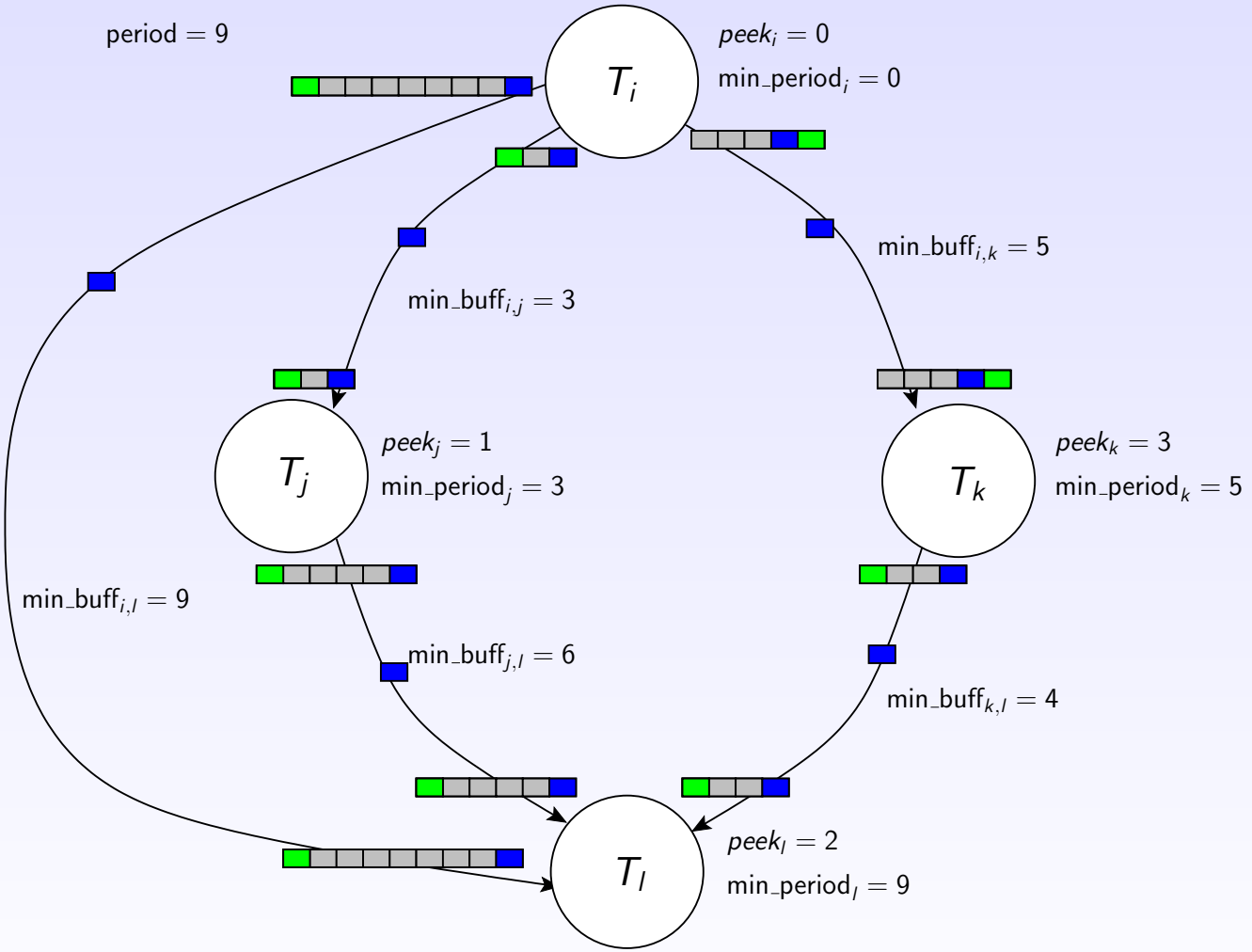
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



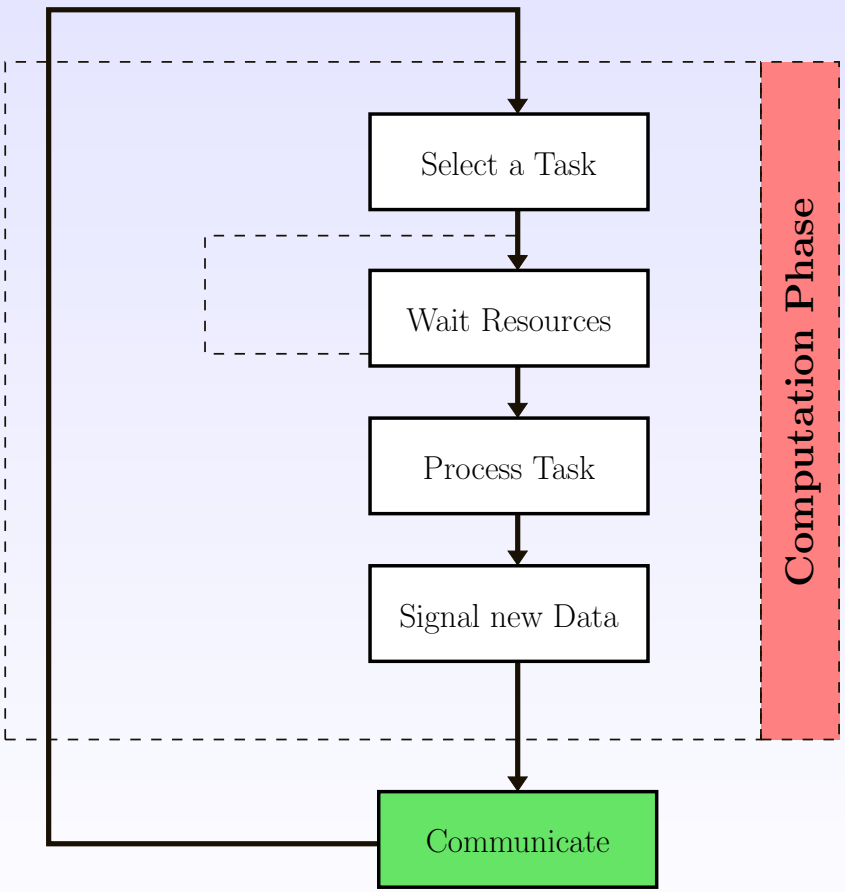
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



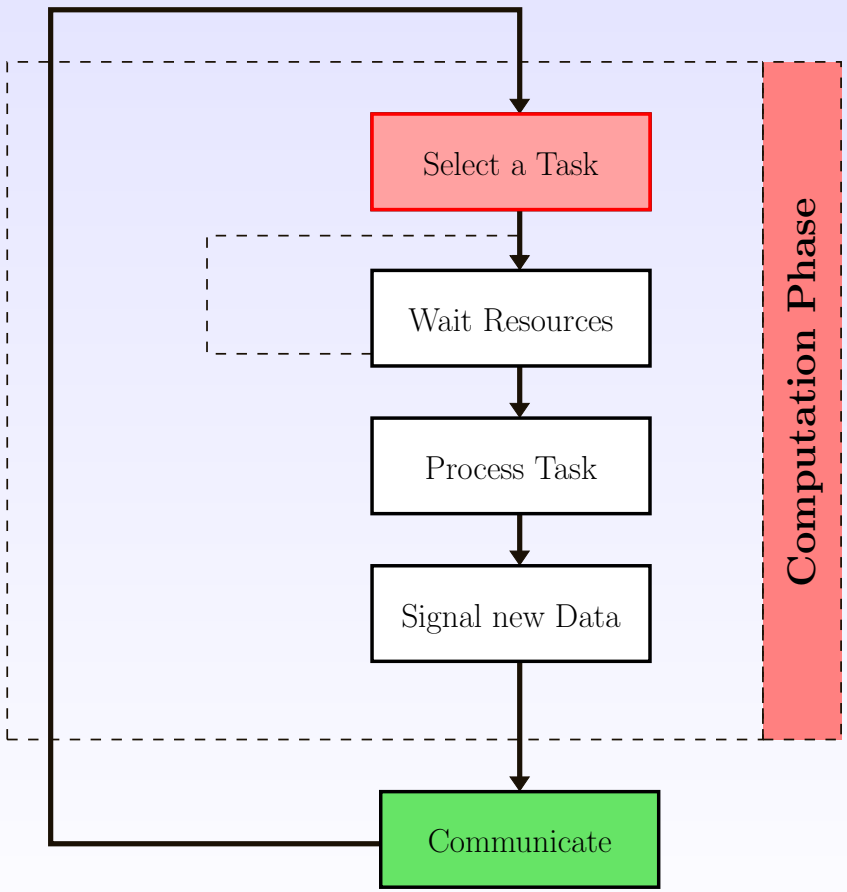
State machine of the framework

Two main phases: Computation and Communication



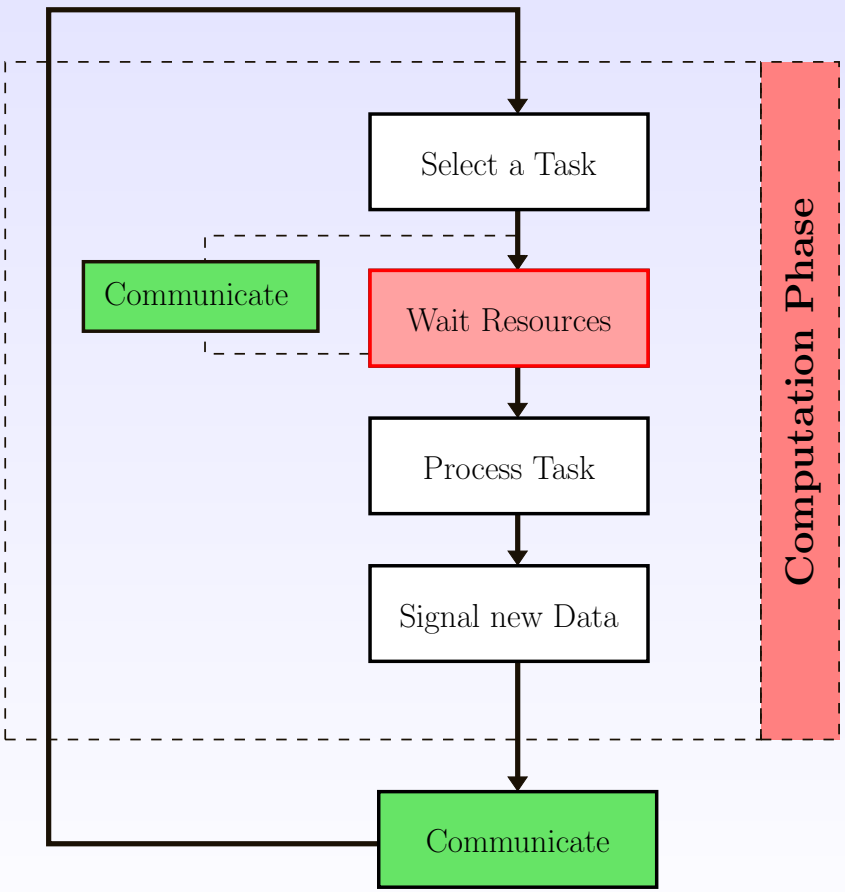
State machine of the framework

Two main phases: Computation and Communication



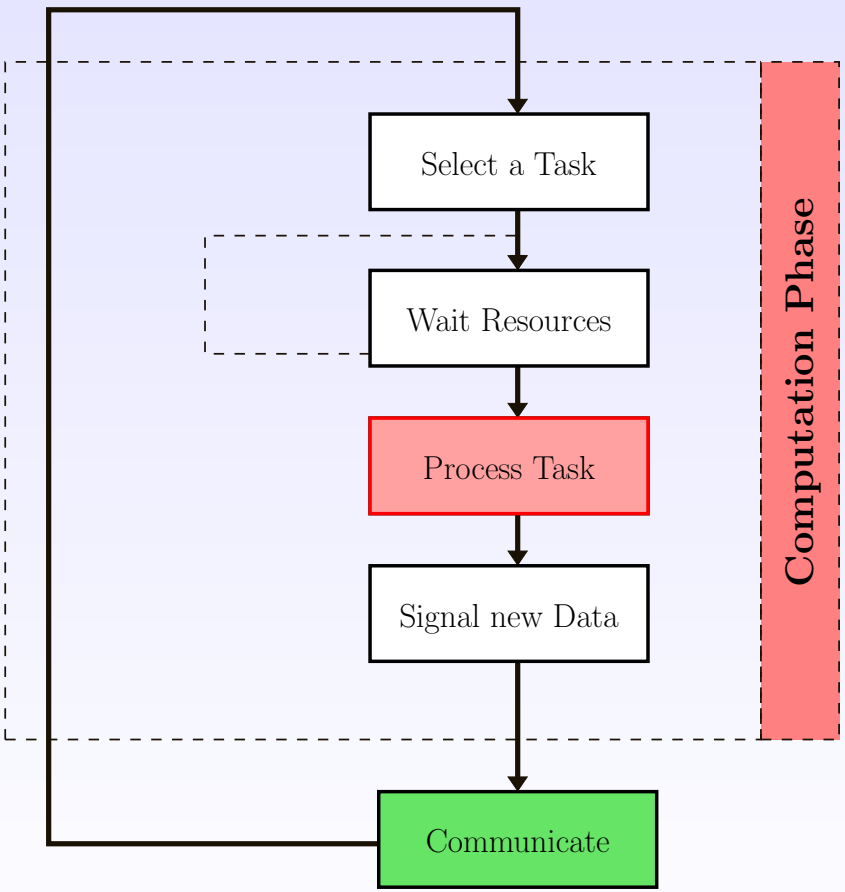
State machine of the framework

Two main phases: Computation and Communication



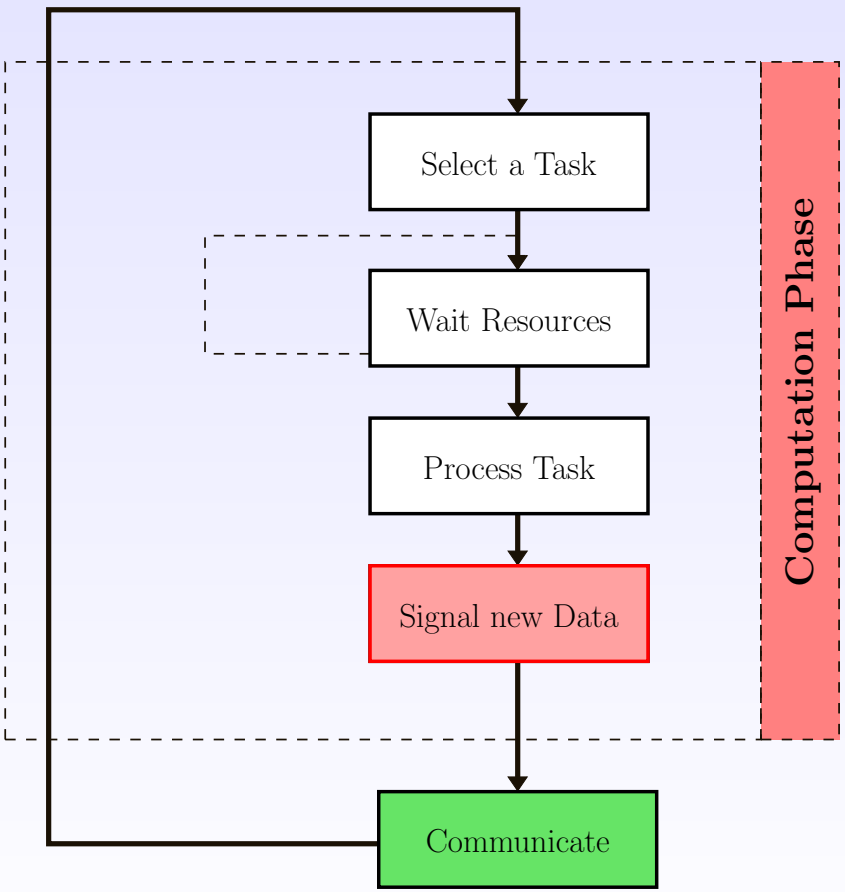
State machine of the framework

Two main phases: Computation and Communication



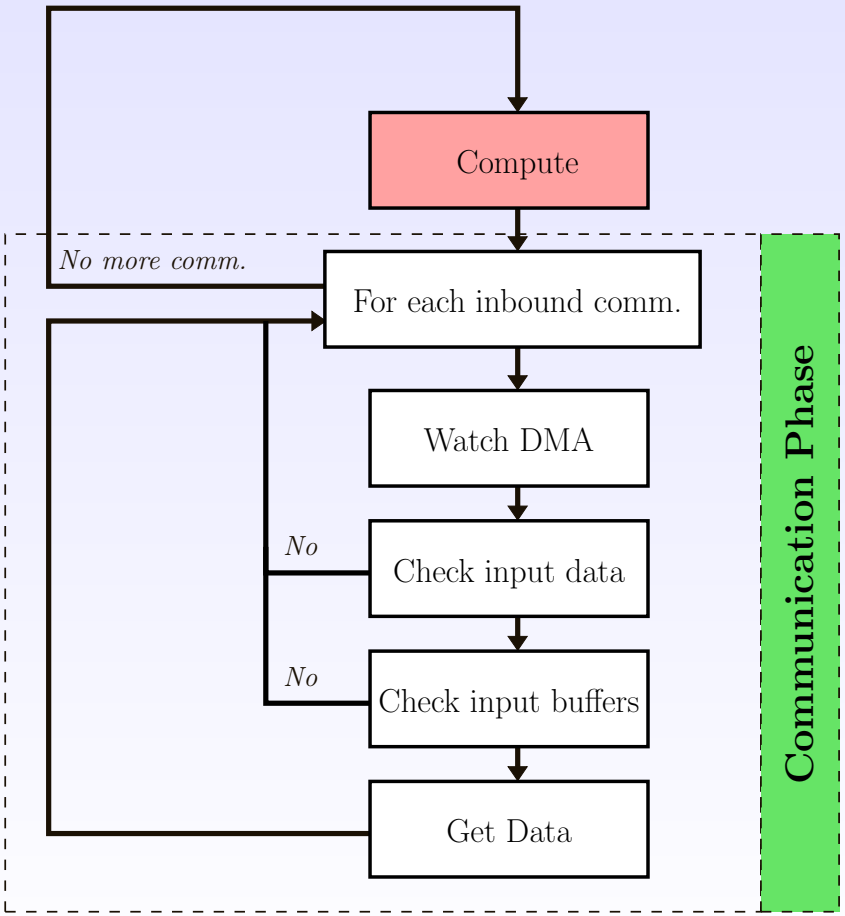
State machine of the framework

Two main phases: Computation and Communication



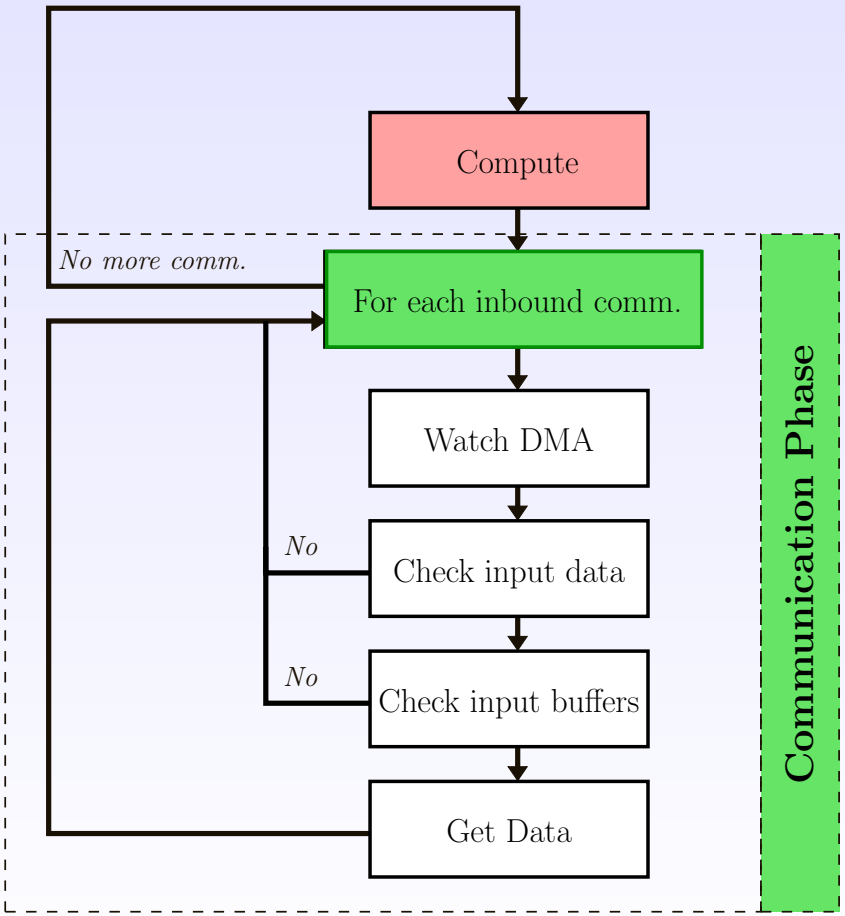
State machine of the framework

Two main phases: Computation and Communication



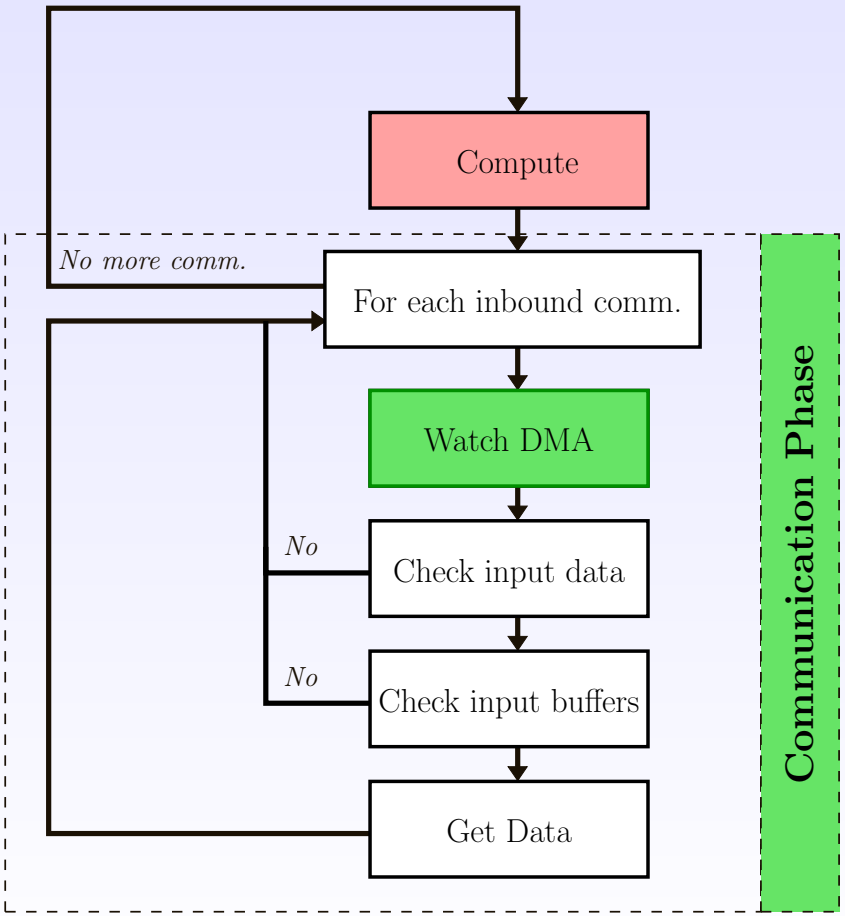
State machine of the framework

Two main phases: Computation and Communication



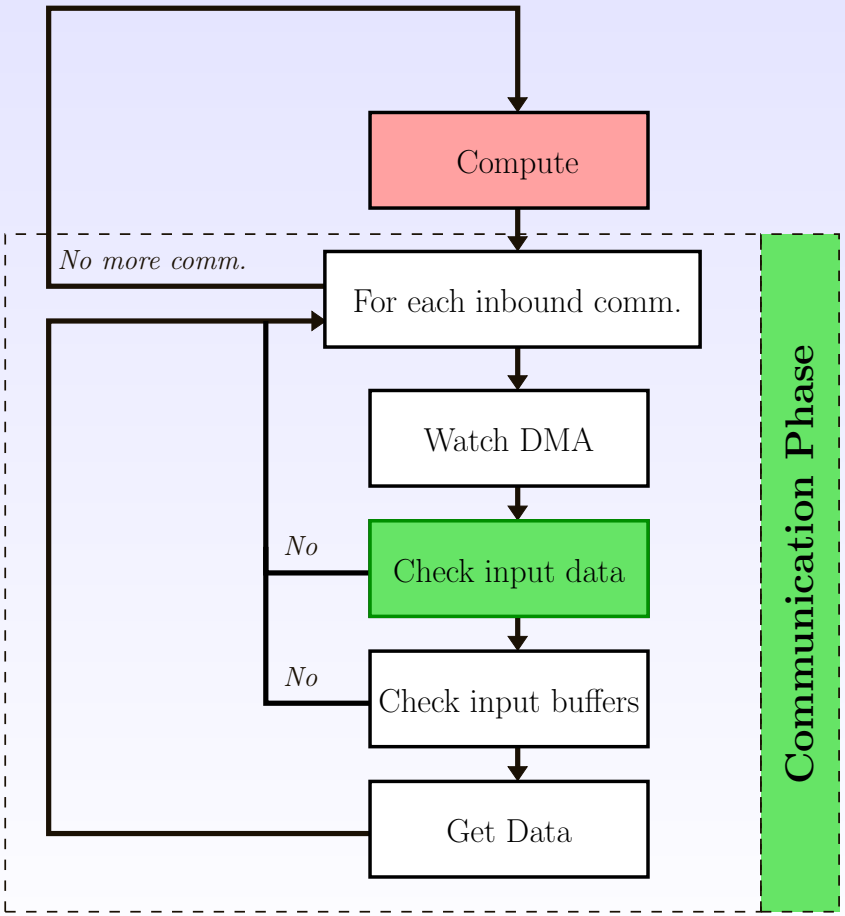
State machine of the framework

Two main phases: Computation and Communication



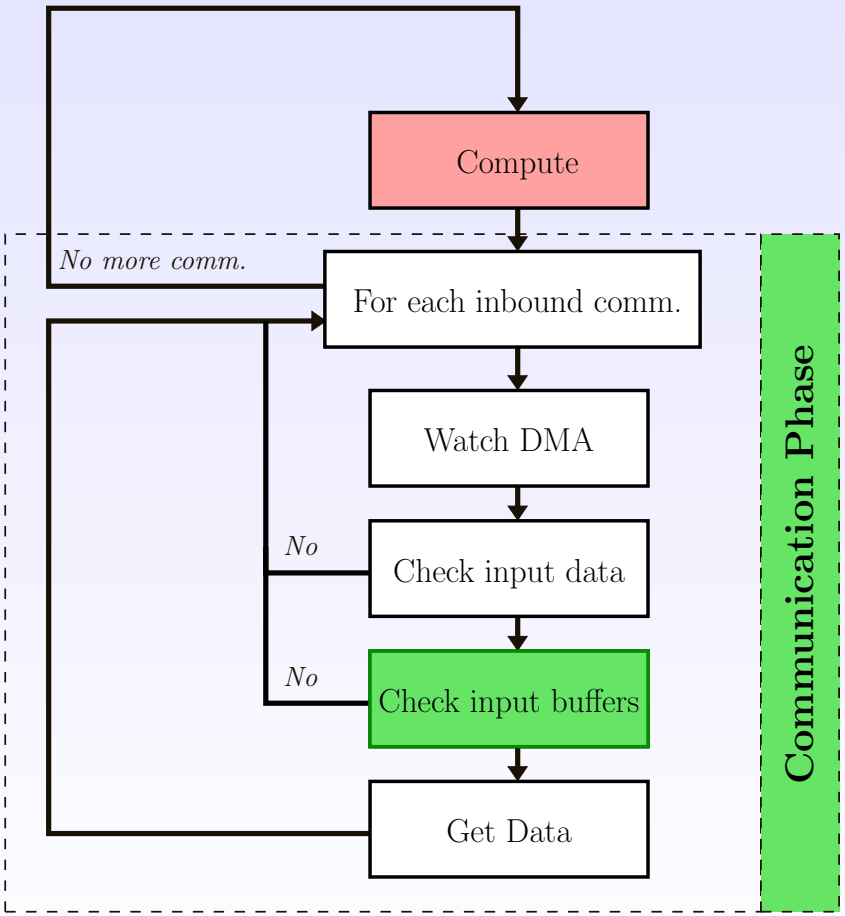
State machine of the framework

Two main phases: Computation and Communication



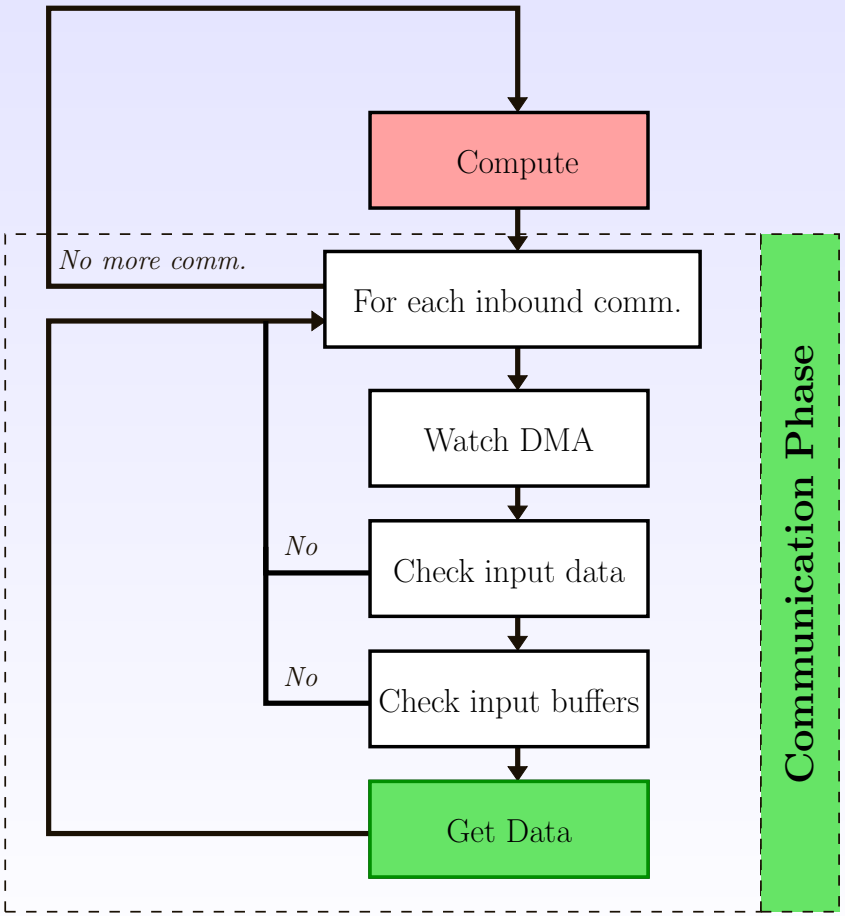
State machine of the framework

Two main phases: Computation and Communication

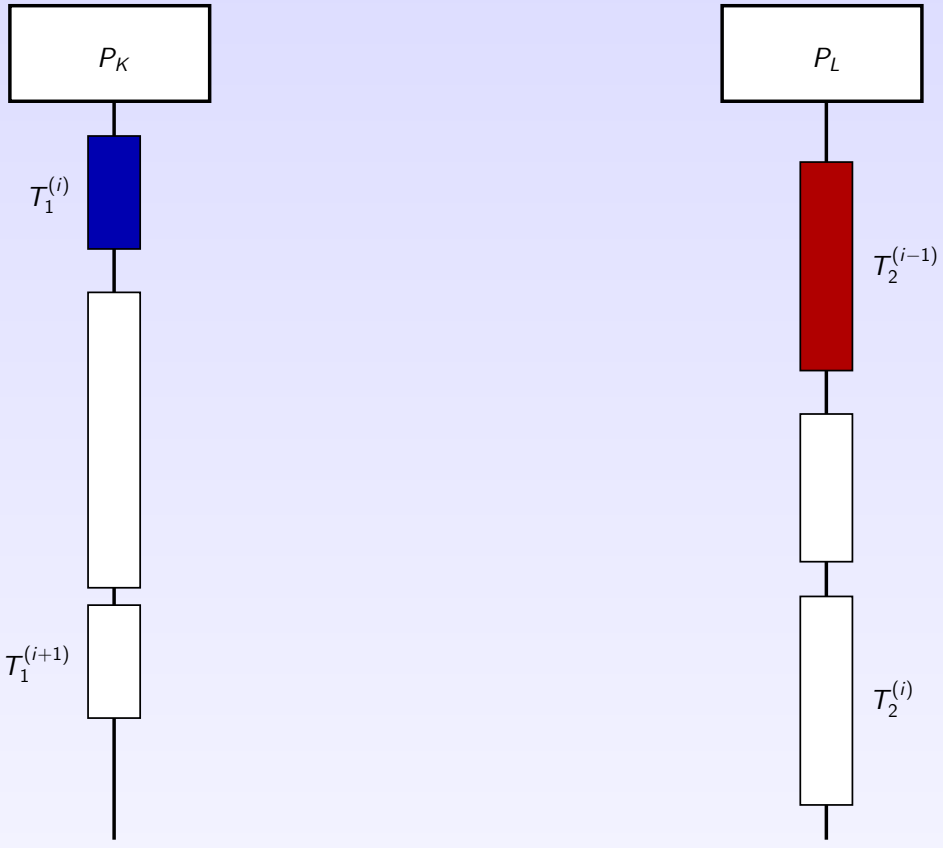


State machine of the framework

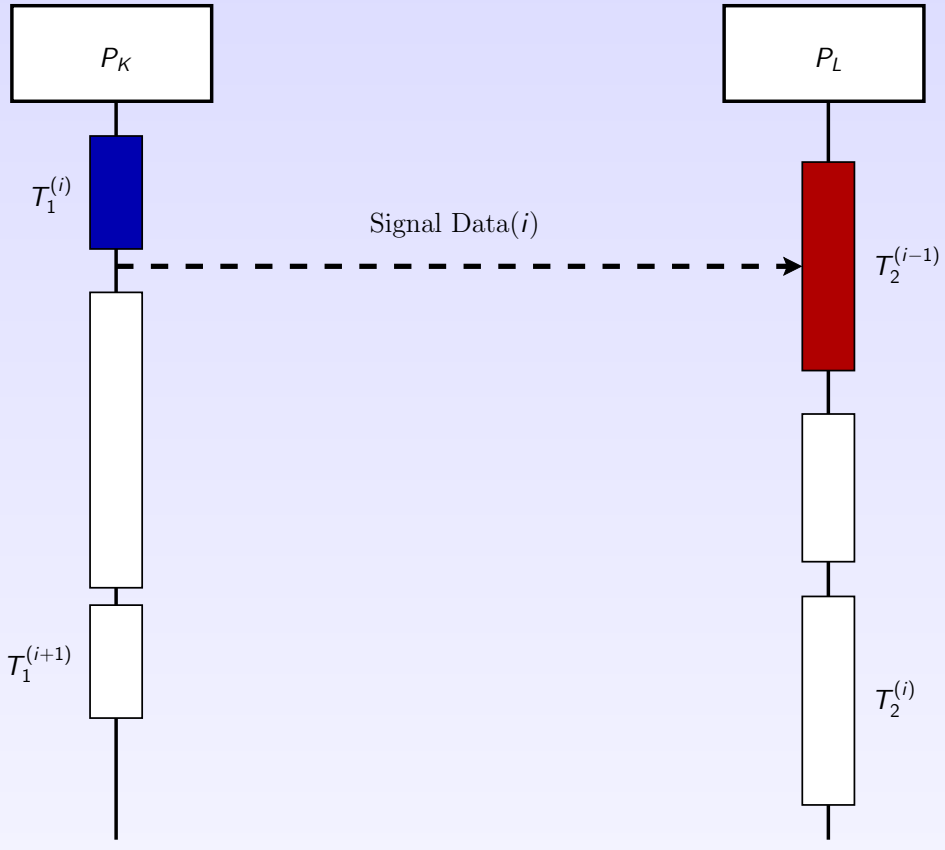
Two main phases: Computation and Communication



Communication between processors



Communication between processors

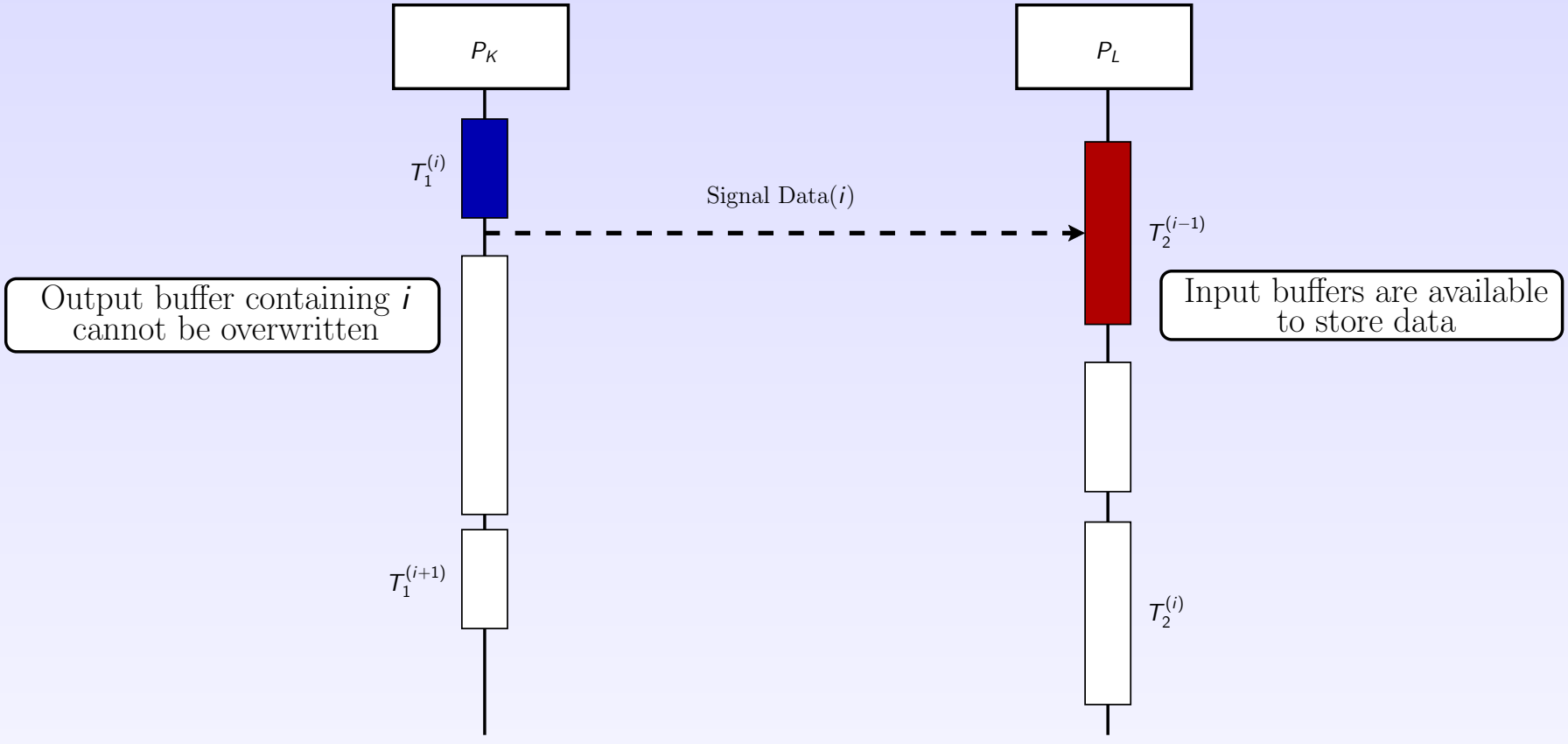


mfc_putb for SPEs' outbound communications.

spe_mfcio_getb for PPEs' outbound communications to SPEs.

memcpy for PPEs' outbound communications to main memory.

Communication between processors

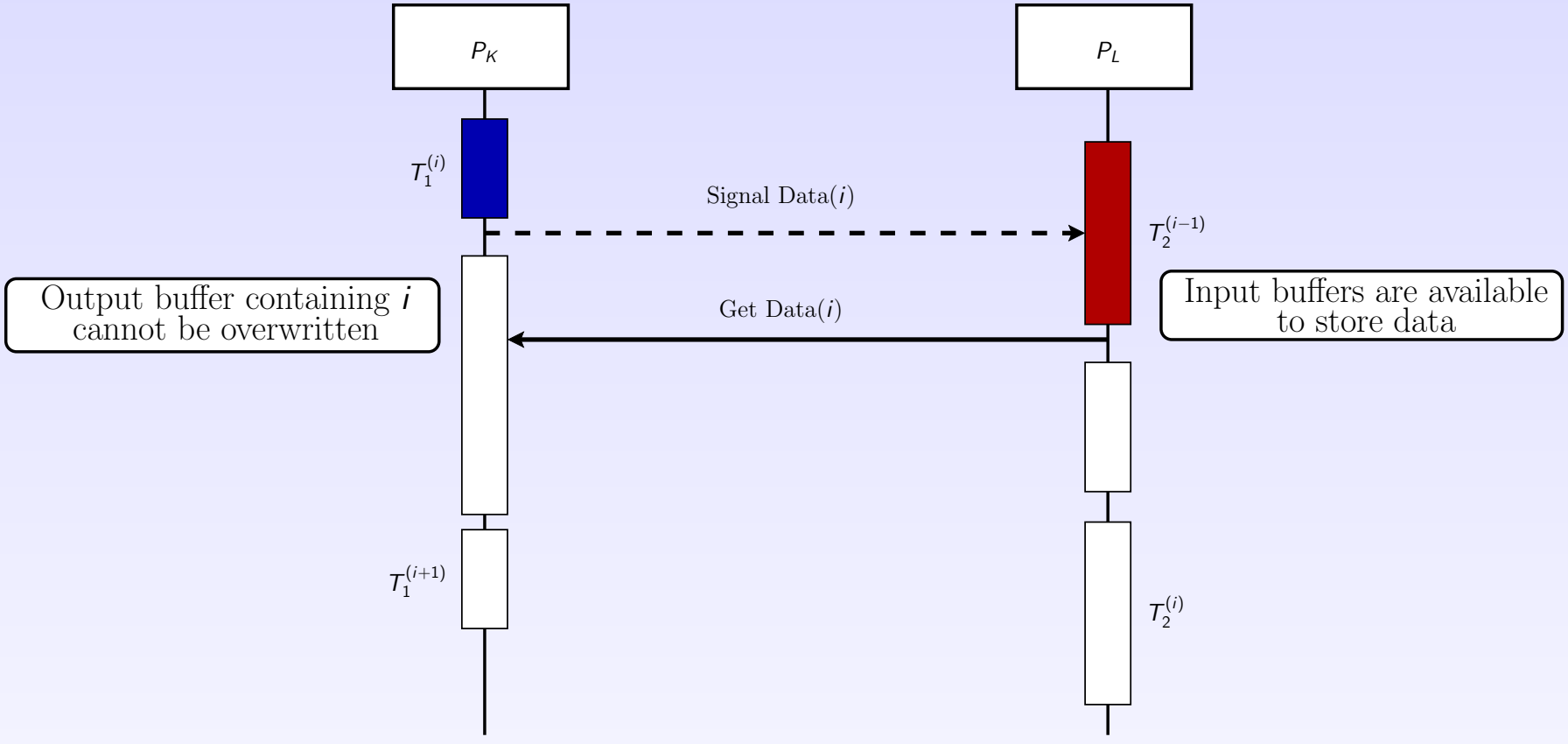


mfc_putb for SPEs' outbound communications.

spe_mfcio_getb for PPEs' outbound communications to SPEs.

memcpy for PPEs' outbound communications to main memory.

Communication between processors

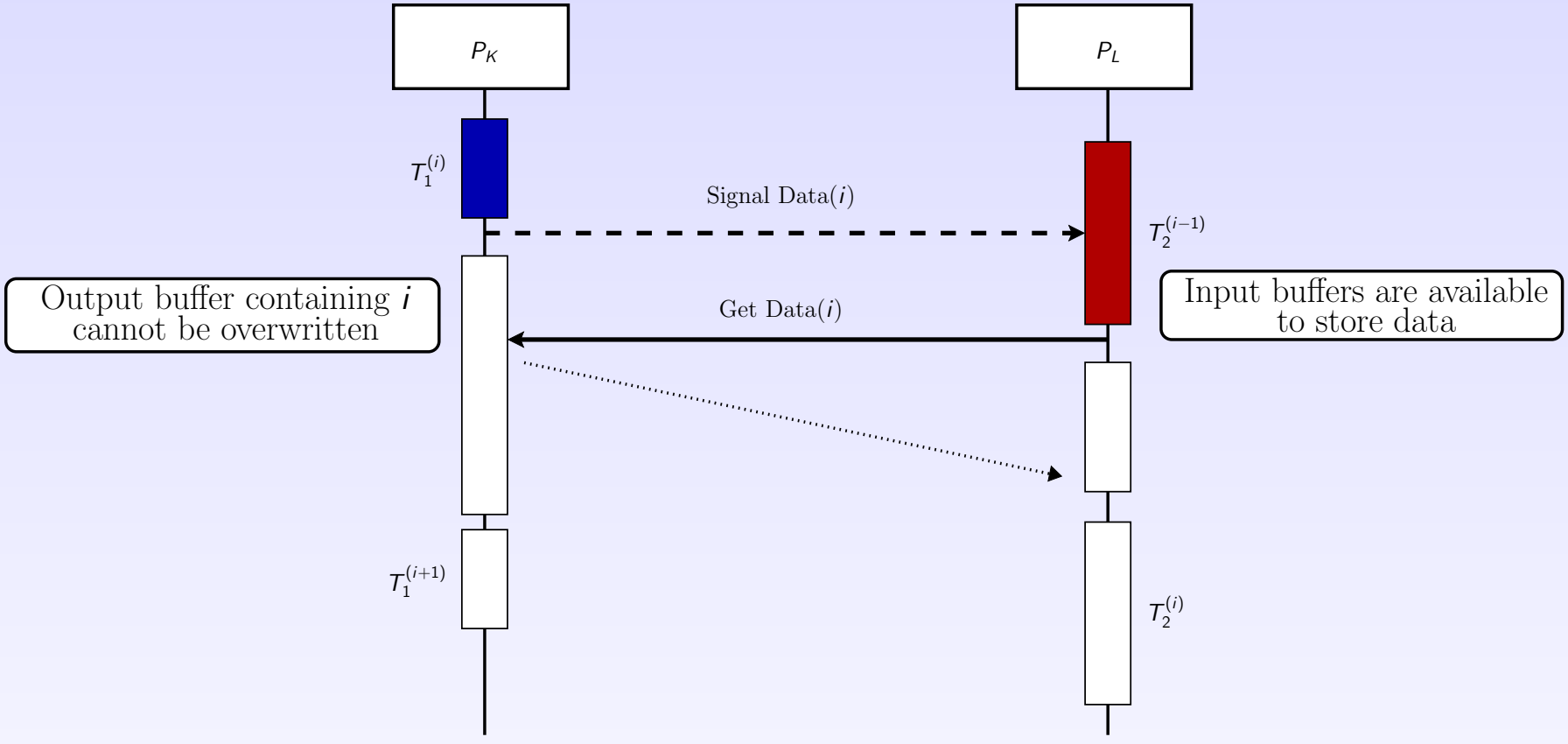


mfc_get for SPEs' inbound communications.

spe_mfcio_put for PPEs' inbound communications from SPEs.

memcpy for PPEs' inbound communications from main memory.

Communication between processors

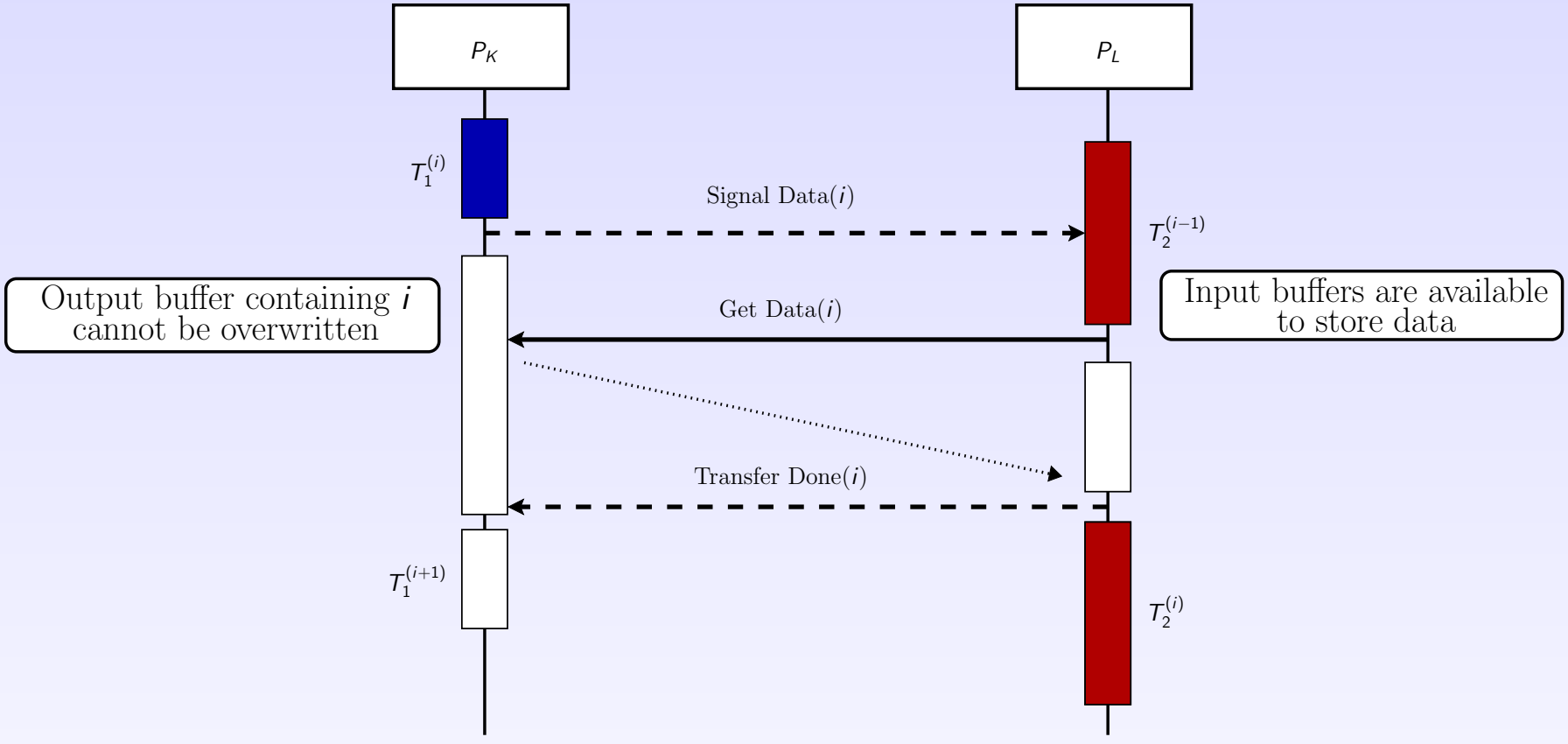


mfc_get for SPEs' inbound communications.

spe_mfcio_put for PPEs' inbound communications from SPEs.

memcpy for PPEs' inbound communications from main memory.

Communication between processors

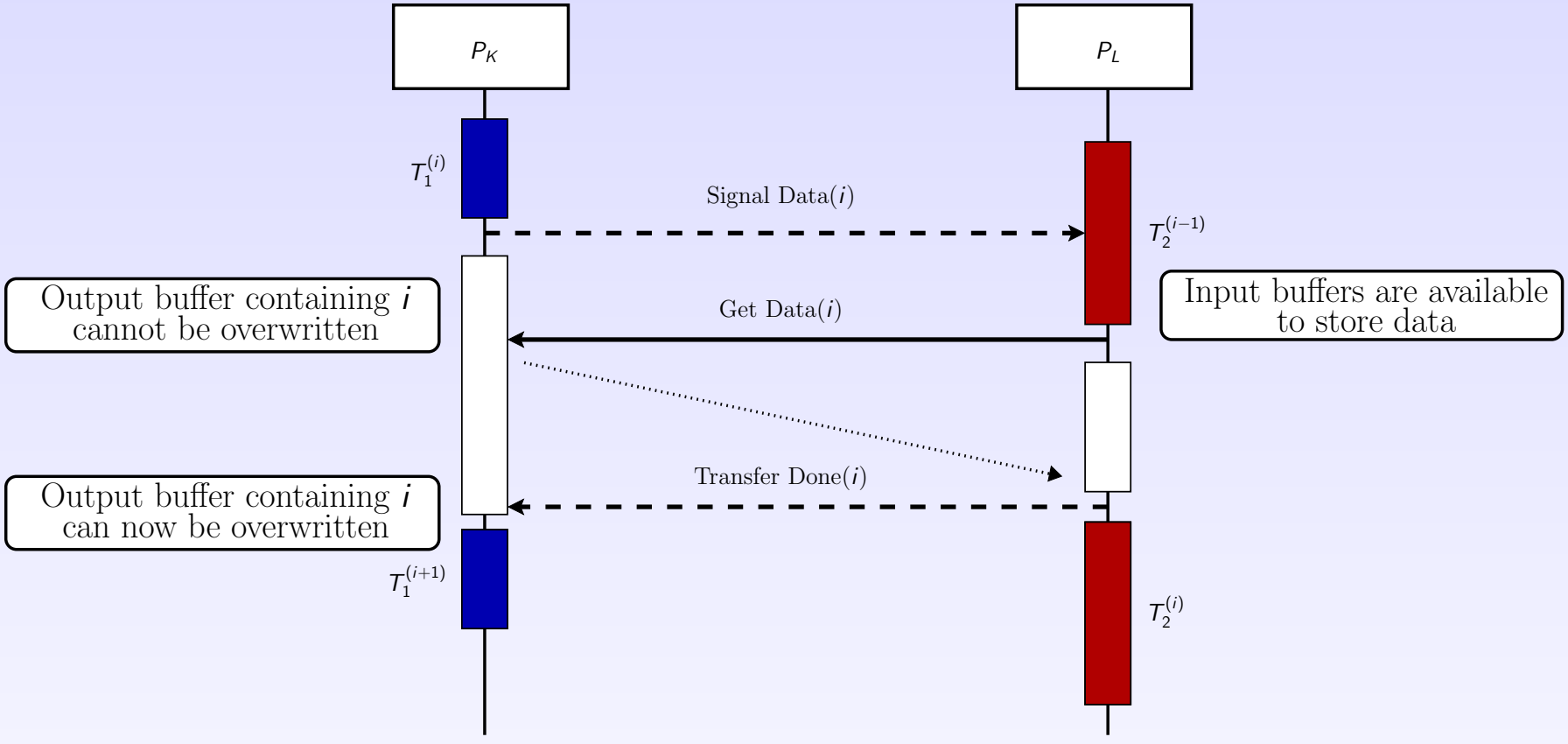


mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Communication between processors

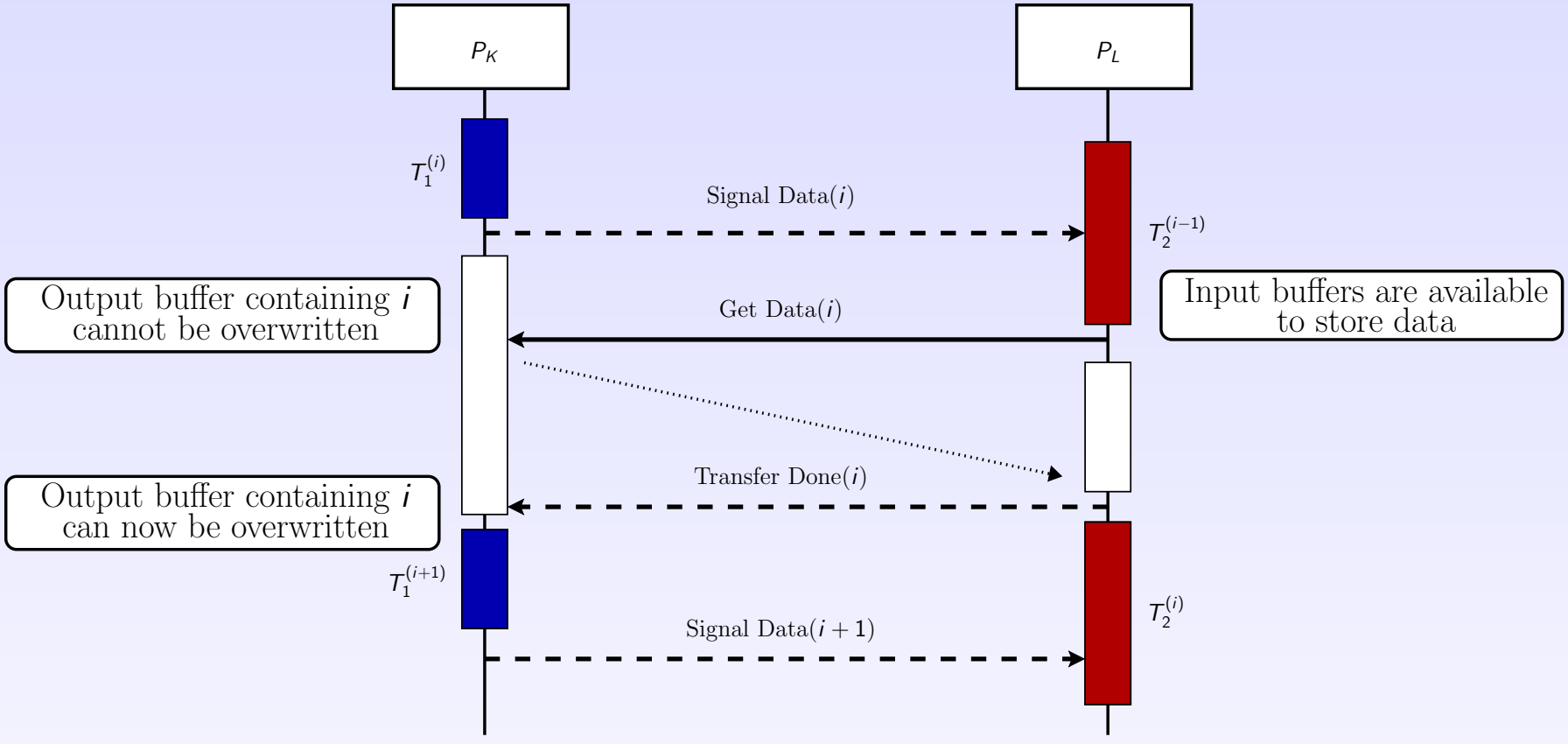


mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Communication between processors



mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time
- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.
- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time

- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.

- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time
- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.
- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time

- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.

- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time
- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.
- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Experimental setup

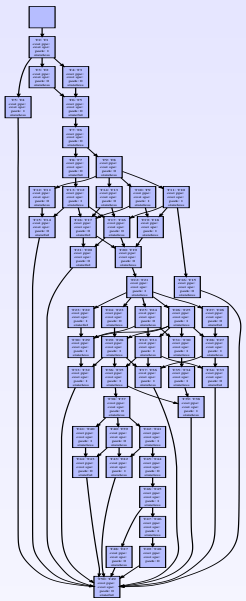
- Linear-Programming: 5% from optimal to reduce compute time
- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.
- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

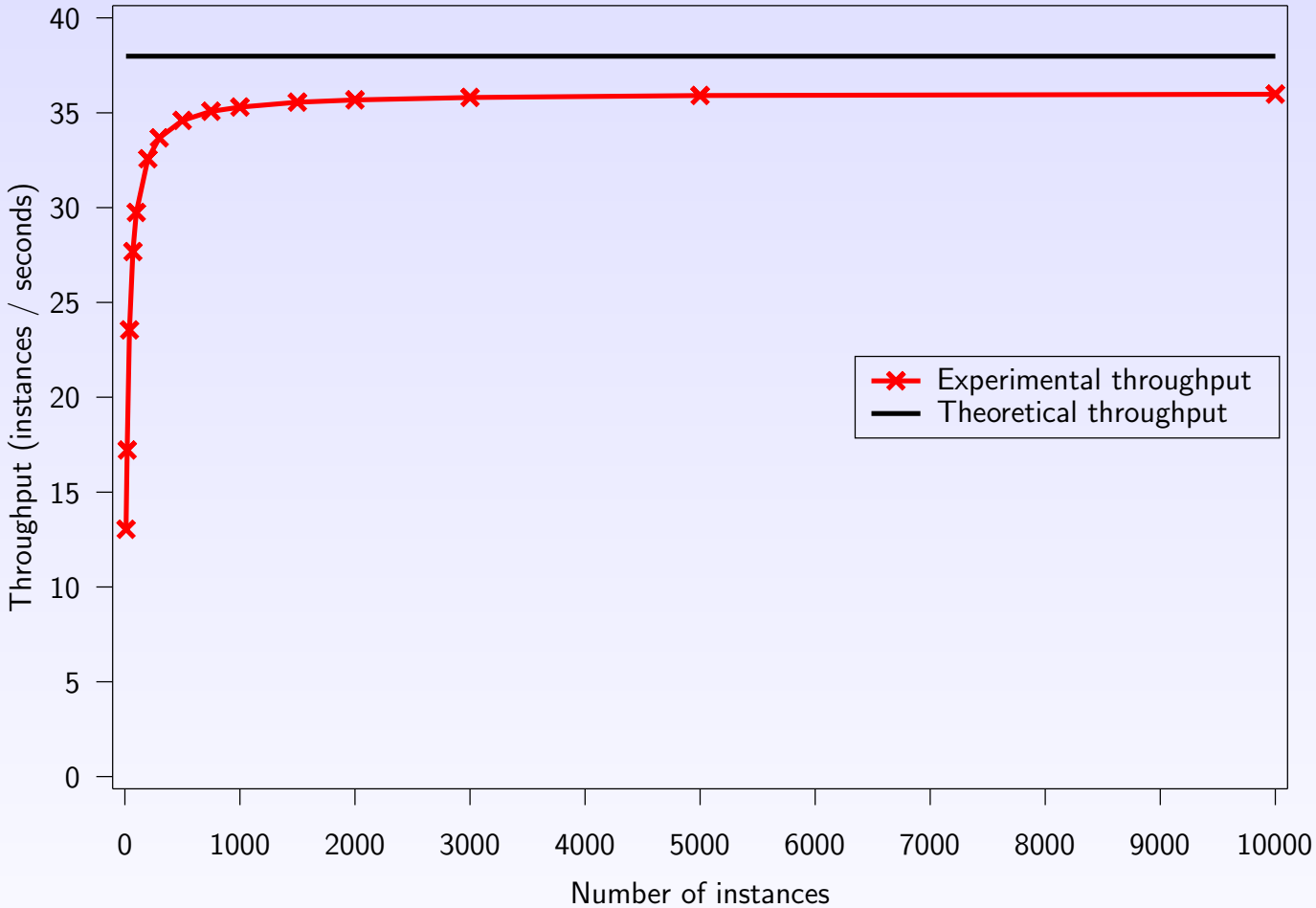
Experimental setup

- Linear-Programming: 5% from optimal to reduce compute time
- GreedyMem: Simple greedy heuristics balancing memory footprint across PEs.
  - Tasks are processed in topological order.
  - Valid SPE with the least loaded memory is selected.
- GreedyCpu: Simple greedy heuristics balancing compute load across PEs.
  - Tasks are processed in topological order.
  - Least loaded SPE is selected, provided that it has enough free memory.

Reaching steady state

Graph 1

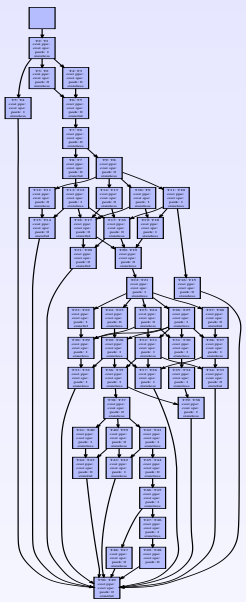


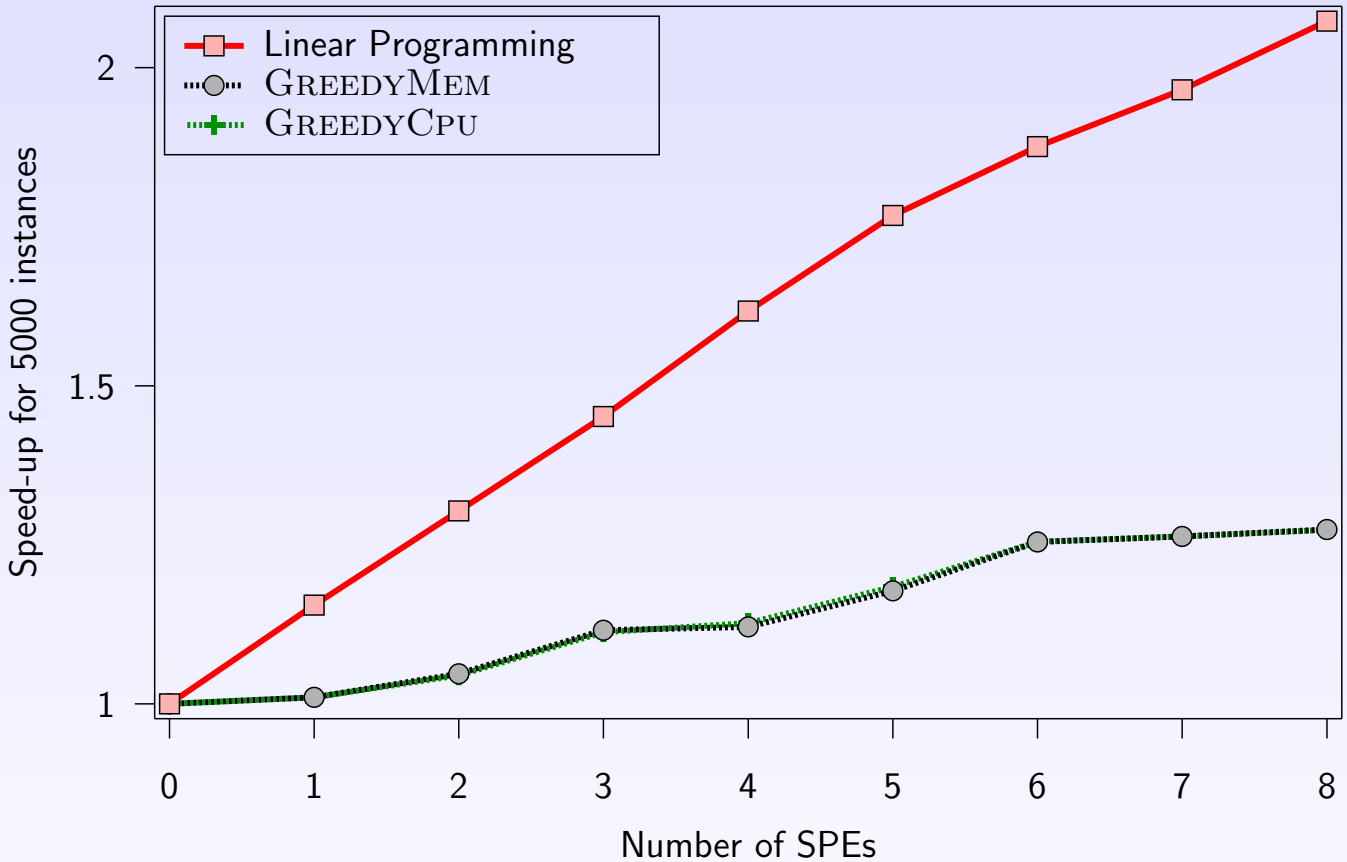


95% of the theoretical throughput is achieved after 1000 periods

Experimental results

Graph 1

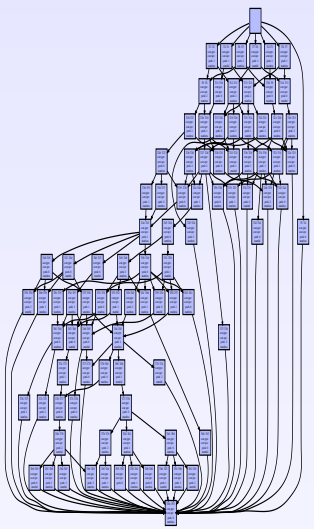


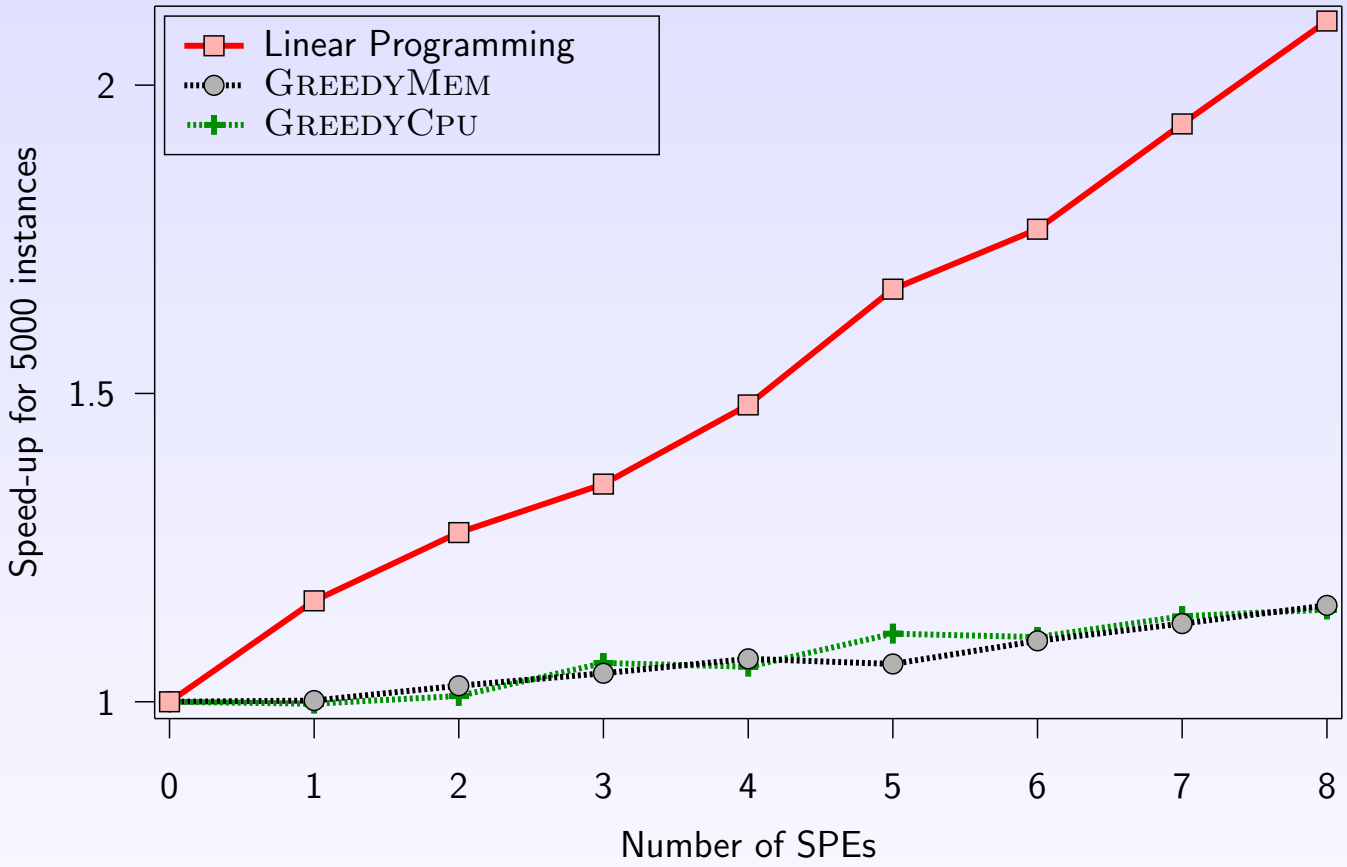


Results are obtained over 5000 periods, 2x speedup using 8 SPEs.

Experimental results

Graph 2

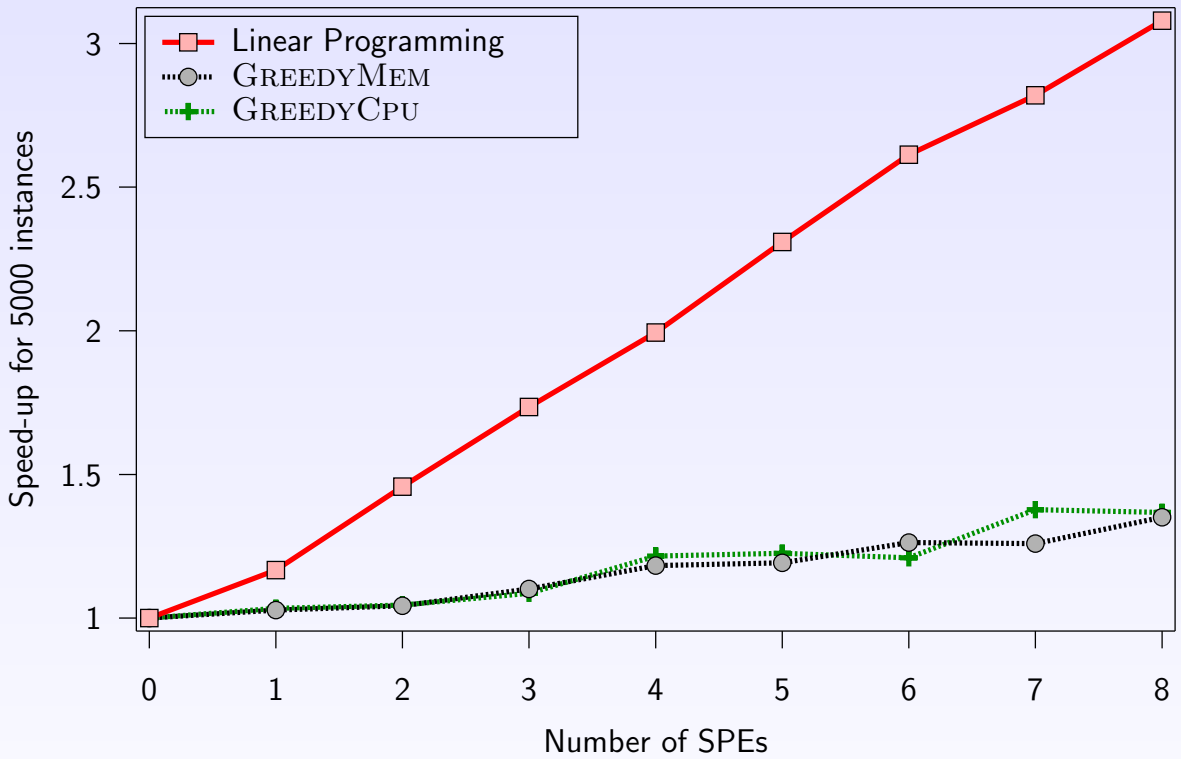




Results are obtained over 5000 periods, 2x speedup using 8 SPEs.

Experimental results

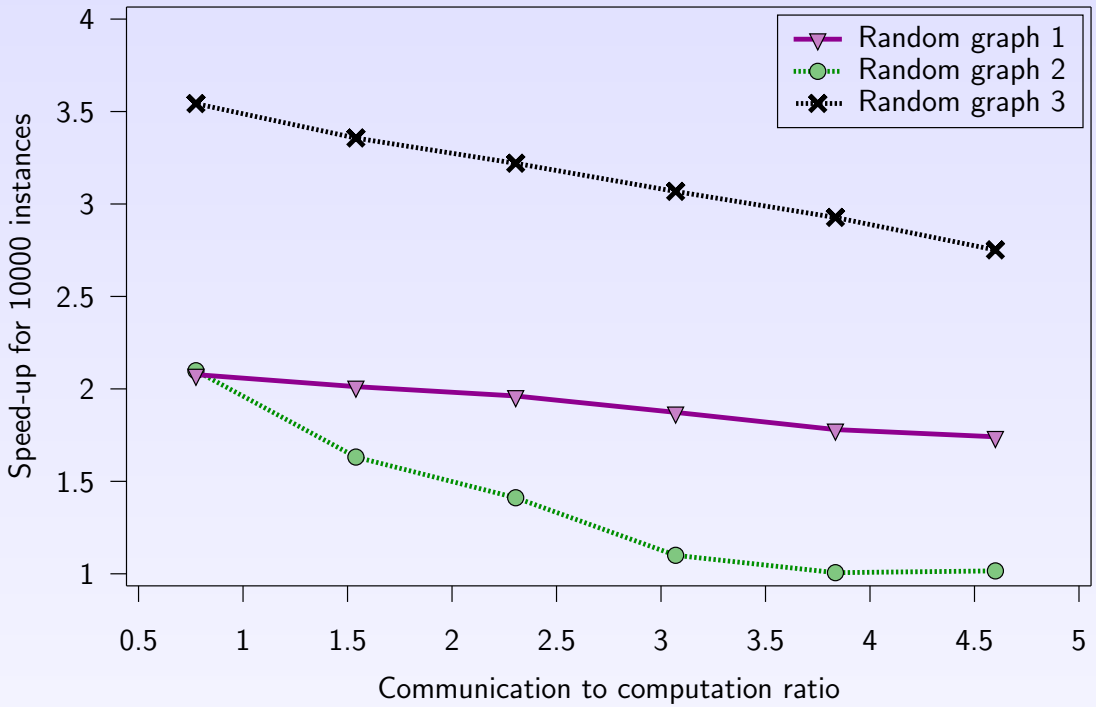
Graph 3: 50 tasks deep chain graph



Results are obtained over 5000 periods, 3x speedup using 8 SPEs.

Experimental results

We let the communication to computation ratio of each graph vary



Results are obtained over 10000 periods.

The heavier communication are, the harder it is to achieve theoretical throughput...

... but increasing the number of periods helps a lot.

Outline

Feedback on our approach

- We designed a realistic and yet tractable model of the Cell processor.
- Our framework allowed us to test our scheduling strategy, and to compare it to simpler heuristic strategies.

- We have shown that :
  - 95% of the throughput predicted by the linear program,
  - Good and scalable speedup when using up to 8 SPEs,
  - Clearly outperforms simple heuristics

Scheduling a complex application on a heterogeneous multicore processor is a challenging task

Scheduling tools can help to achieve good performance.

Feedback on Cell programming

- Multilevel heterogeneity:
  - 32 bits SPEs vs 64 bits PPE architectures
  - Different communication mechanism and constraints
- Non trivial initialization phase
  - Varying data structure sizes (32/64bits)
  - Runtime memory allocation

On-going and Future work

- Better communication modeling
  - Is linear cost model relevant ?
  - Contention on concurrent DMA operations ?
- Larger platforms
  - Using multiple CELL processors
  - CELL + other type of processing units ?
  - Work on communication modeling
- Design scheduling heuristics
  - MIP is costly