Steady-State Scheduling on Heterogeneous Platforms

Matthieu Gallet
Advisors: Yves Robert and Frédéric Vivien

École Normale Supérieure de Lyon
GRAAL team, Laboratoire de l'Informatique du Parallélisme

October 20, 2009

Introduction

- Scheduling large applications on complex heterogeneous platforms
- Stream of data to process: video or audio streams, on-the-fly processing of experimental data, ...
- Structured applications: repeatedly apply several filters to each data set
- Several computing resources, of different kinds
- Heterogeneous communication network
- How to organize all processings?

Introduction

- Makespan minimization is a difficult problem $\longrightarrow$ relaxations

Introduction

- Makespan minimization is a difficult problem $\longrightarrow$ relaxations
- Divisible Load scheduling:
  - Presentation of the Divisible Load Theory
  - Scheduling divisible loads on a processor chain
- Steady-state scheduling:
  - Mono-allocation schedules of task graphs on heterogeneous platforms
  - Dynamic bag-of-tasks applications
  - Computing the throughput of replicated workflows
  - Task graph scheduling on the Cell processor

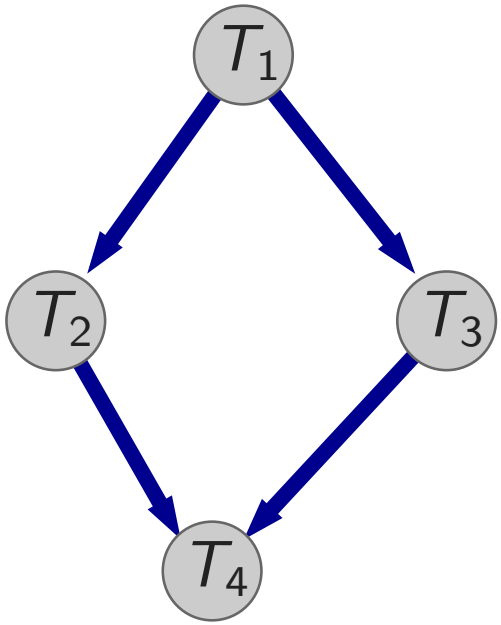
Introduction

- Makespan minimization is a difficult problem $\longrightarrow$ relaxations
- Divisible Load scheduling:
  - Presentation of the Divisible Load Theory
  - Scheduling divisible loads on a processor chain
- Steady-state scheduling:
  - Mono-allocation schedules of task graphs on heterogeneous platforms
  - Dynamic bag-of-tasks applications
  - Computing the throughput of replicated workflows
  - Task graph scheduling on the Cell processor

Outline of this talk

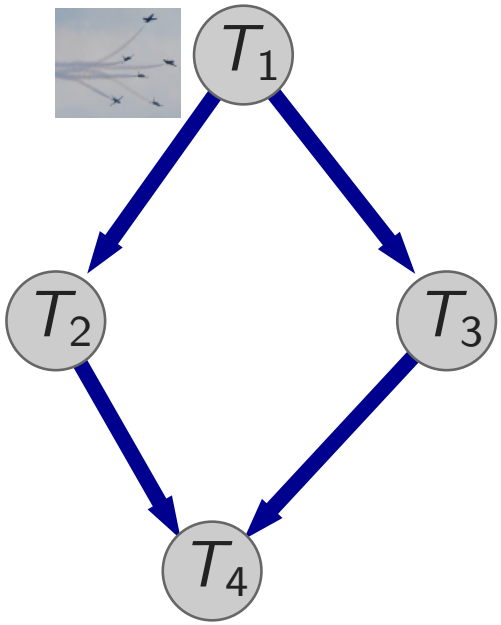
Outline

General presentation



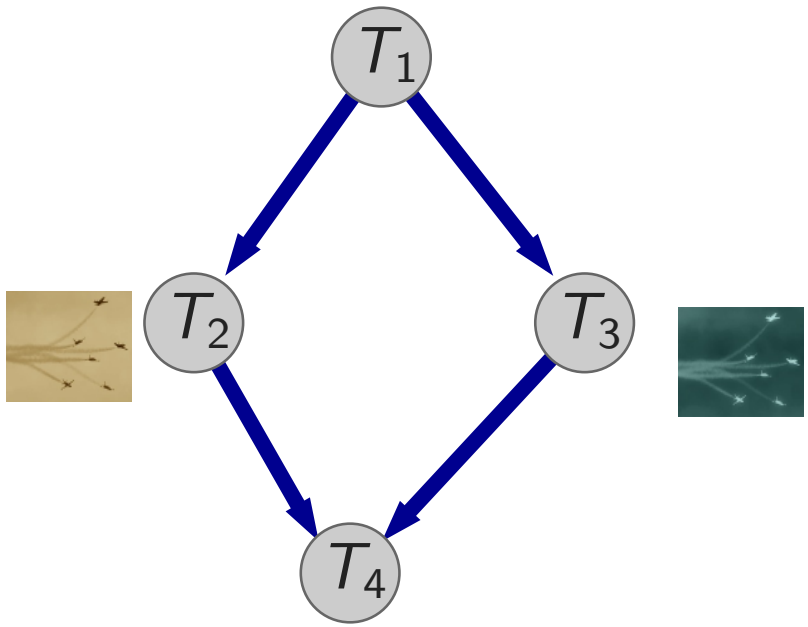
- Structured application: directed acyclic graph
  $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$

General presentation



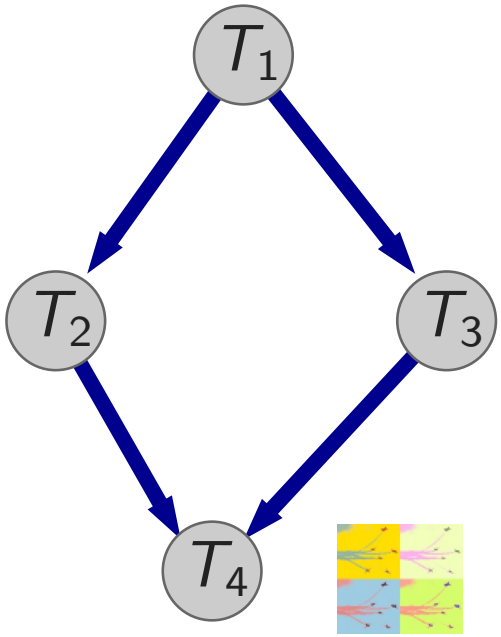
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$

General presentation



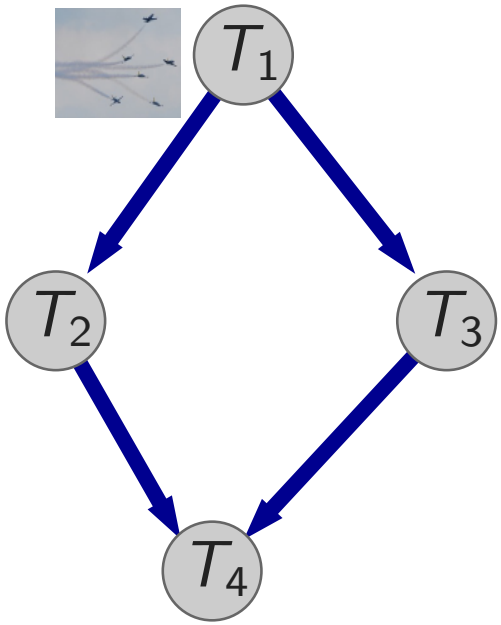
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$

General presentation



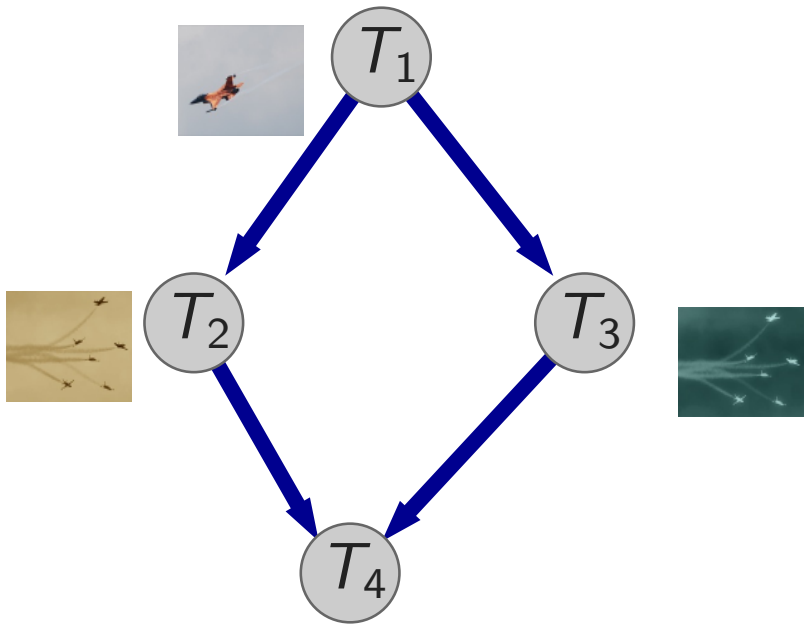
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$

General presentation



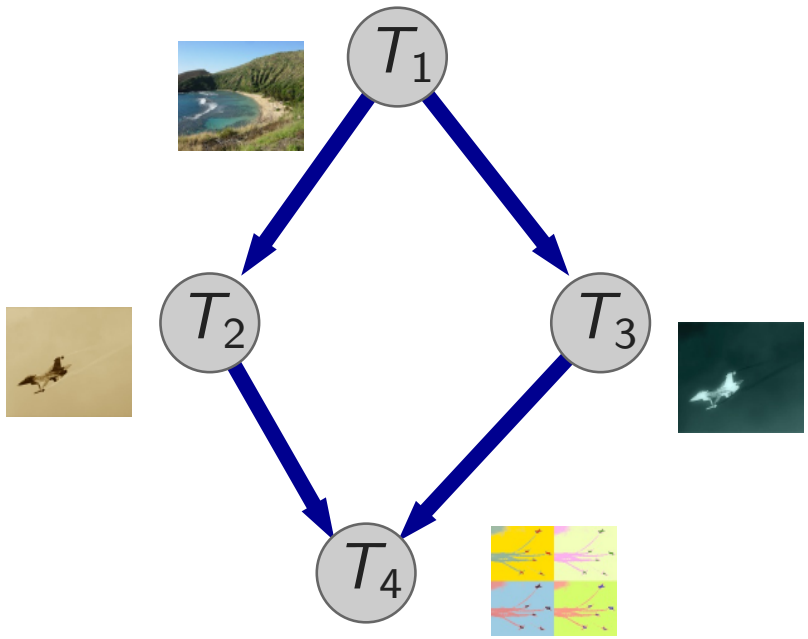
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



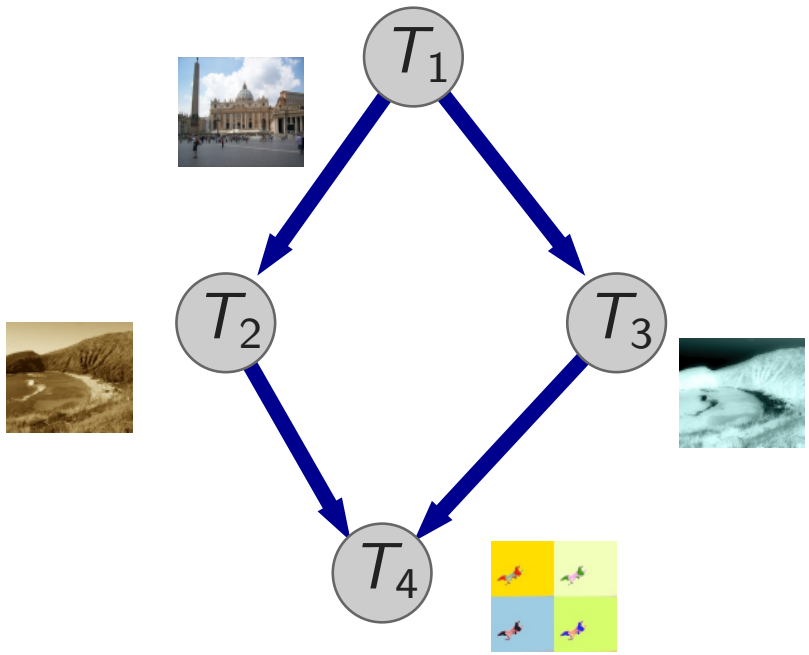
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



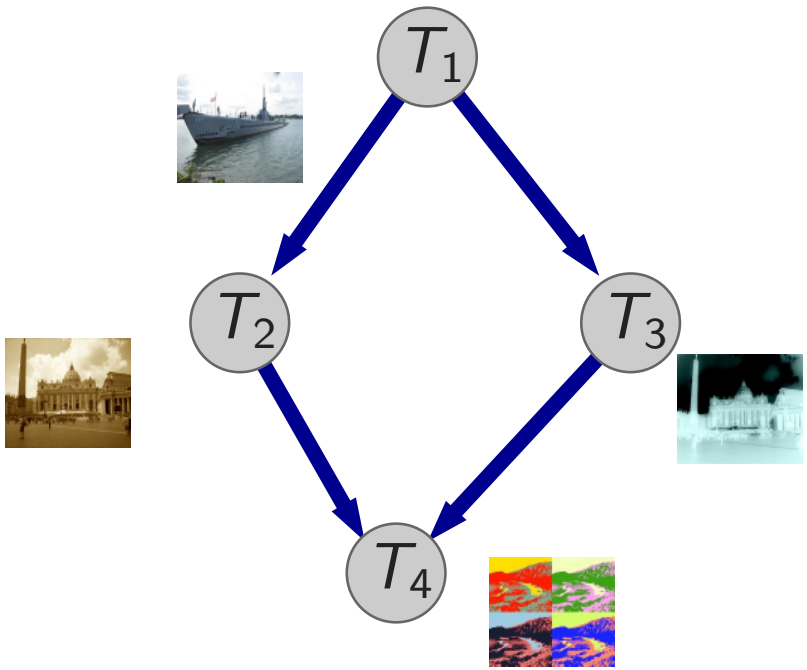
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



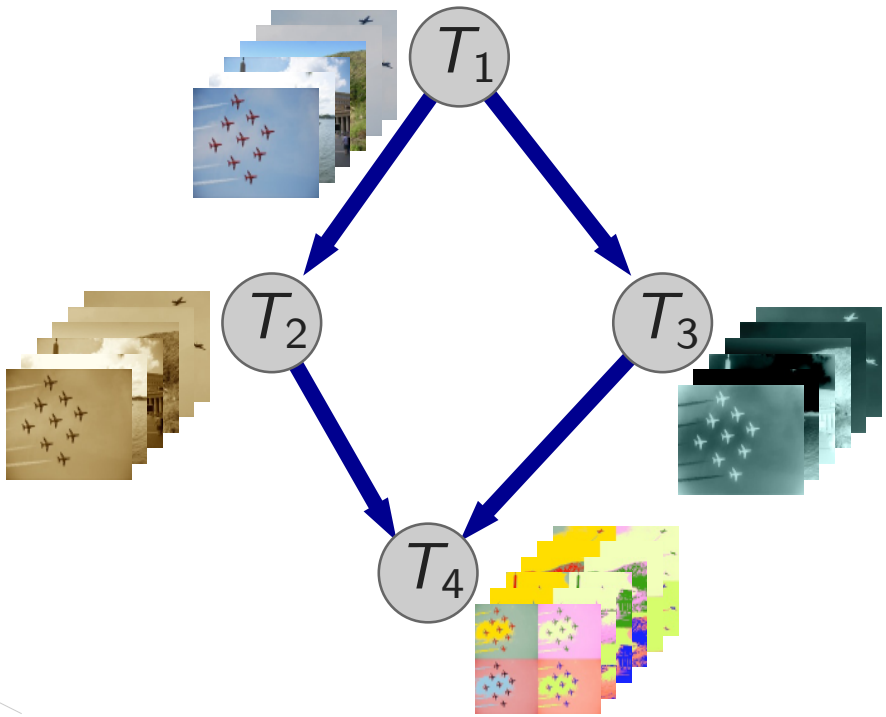
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



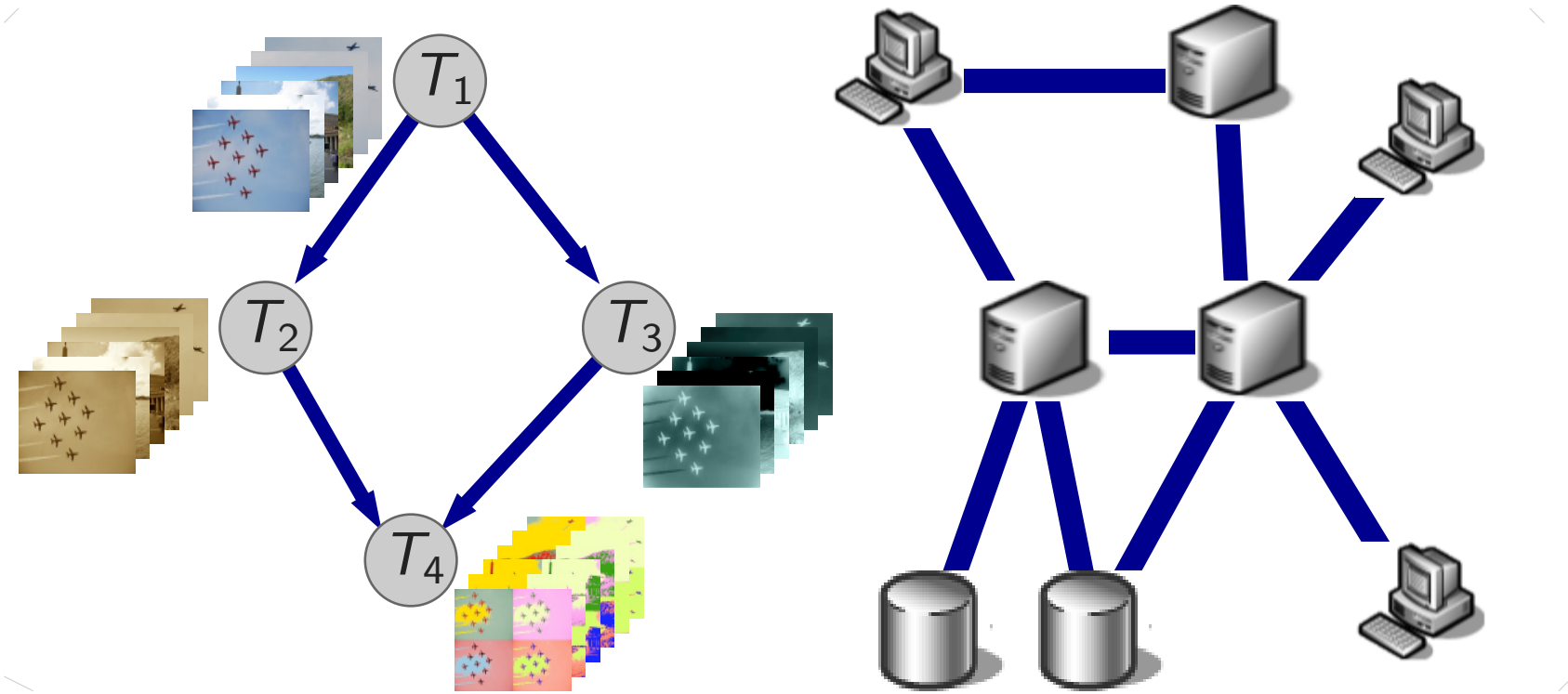
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



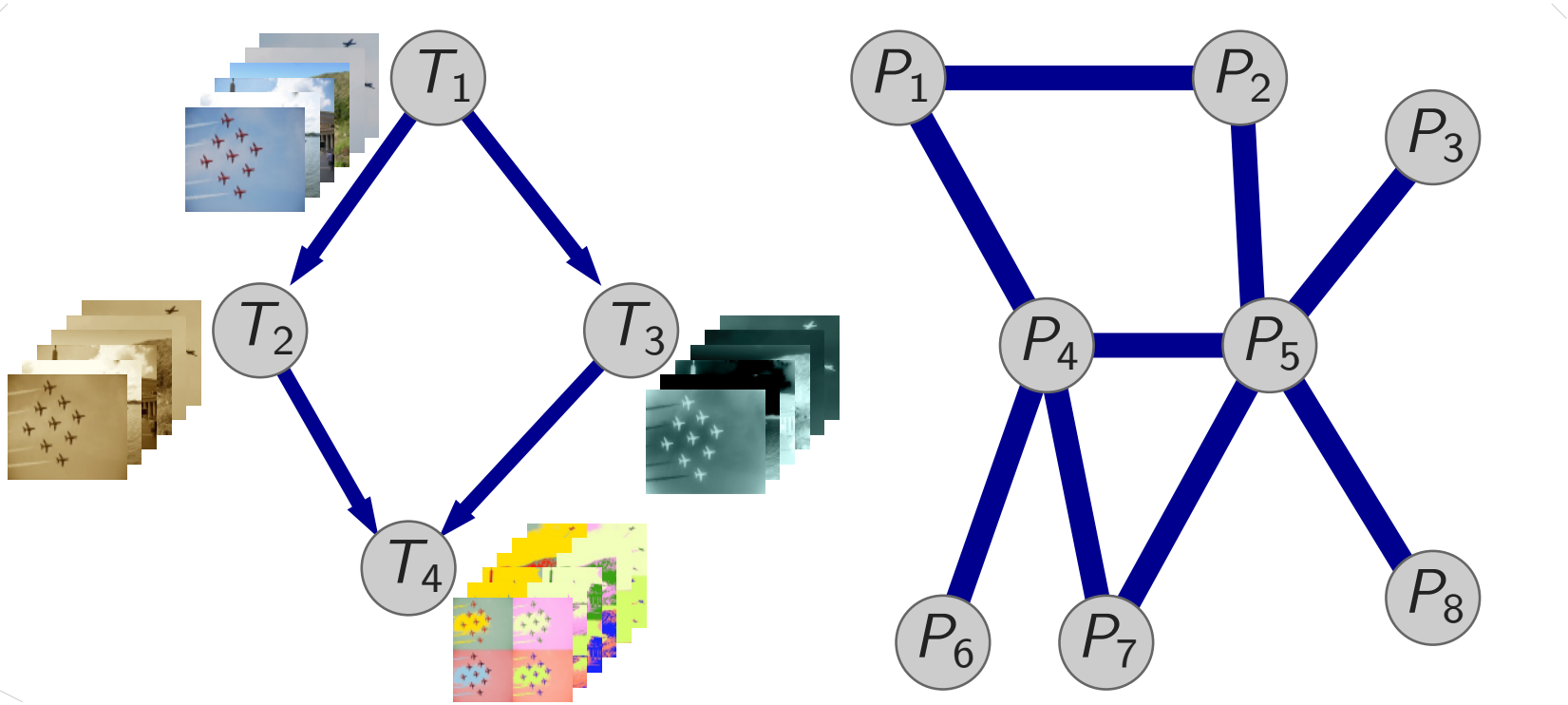
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process

General presentation



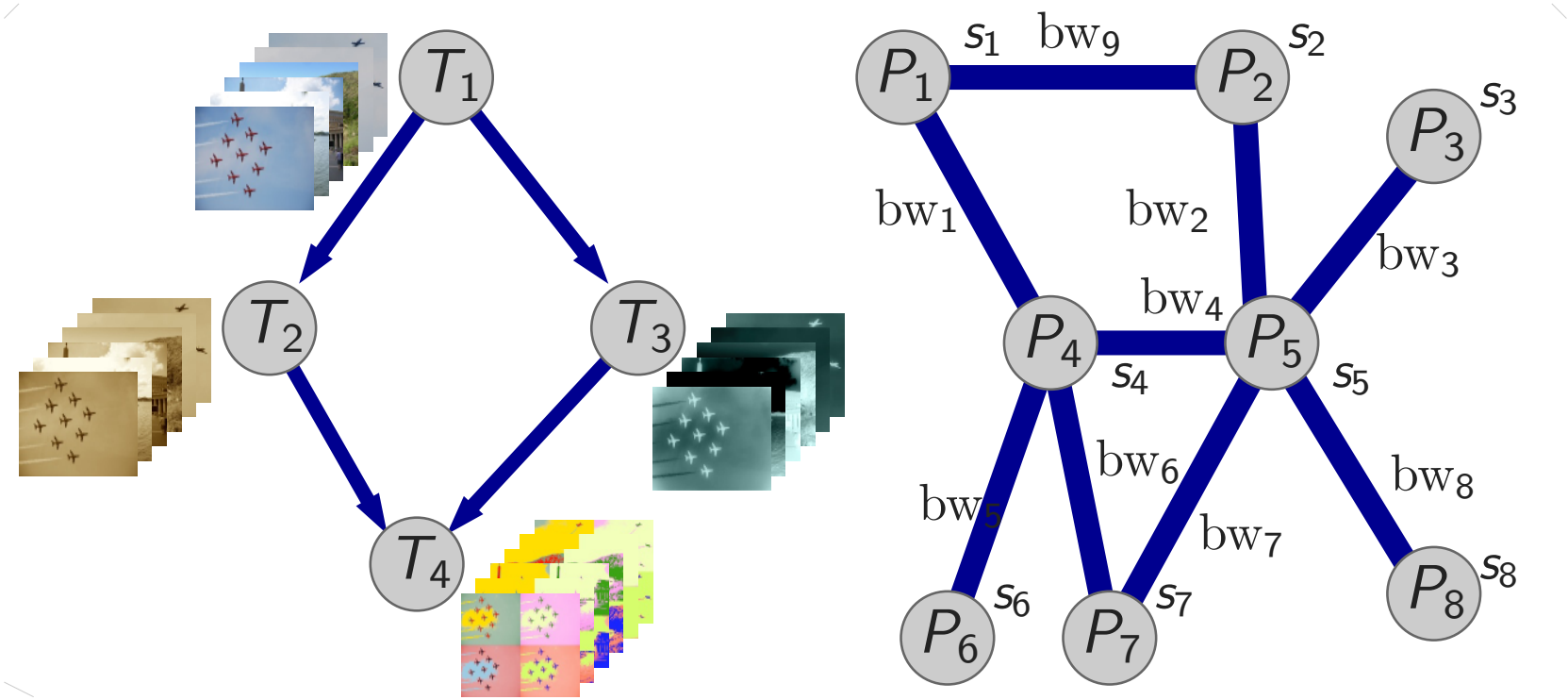
- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process
- Platform

General presentation



- Structured application: directed acyclic graph
  $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process
- Platform, modeled by a graph
  $G_P = (V_P = \{P_1, \cdots, P_p\}, E_P = (P_q \rightarrow P_r))$

General presentation



- Structured application: directed acyclic graph $G_A = (V_A = \{T_1, \ldots, T_n\}, E_A = (F_{k,l})_{k,l})$
- Many data sets to process
- Heterogeneous platform, modeled by a graph $G_P = (V_P = \{P_1, \cdots, P_p\}, E_P = (P_q \rightarrow P_r))$

Objective function

Makespan (maximum completion time)

- The most natural objective function
- Lot of work about its minimization, but difficult problem ☹
- But...

Objective function

Makespan (maximum completion time)

- The most natural objective function
- Lot of work about its minimization, but difficult problem ☹
- But. . . is the makespan relevant in our case?
- Not really:
  - undefined for a continuous flow of data sets
  - does not benefit from regular problem structure

Objective function

Makespan (maximum completion time)

- The most natural objective function
- Lot of work about its minimization, but difficult problem ☹
- But. . . is the makespan relevant in our case?
- Not really:
  - undefined for a continuous flow of data sets
  - does not benefit from regular problem structure

Throughput

Average number of processed data sets per time unit

- Well-suited to continuous flows of data sets
- Based upon problem regularity

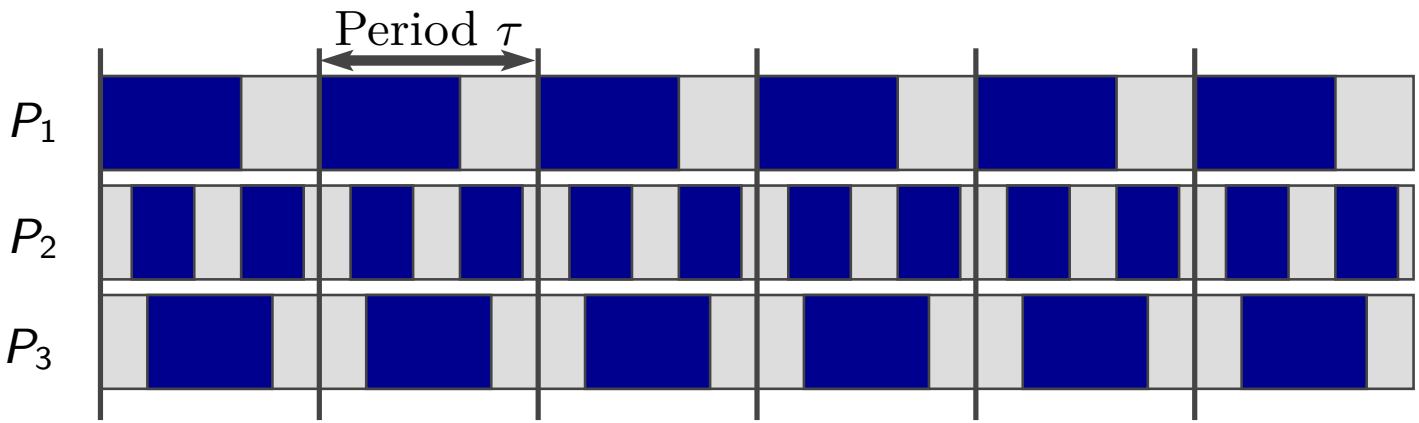
Steady-state scheduling

- Focus on schedule's core
- Neglect initiation and termination phases
- Adapted to throughput maximization

Steady-state scheduling

- Focus on schedule's core
- Neglect initiation and termination phases
- Adapted to throughput maximization

Periodic schedules



- Optimal for throughput $\longrightarrow$ asymptotically optimal for makespan
- Independent of data set count $\longrightarrow$ compact description

Allocation

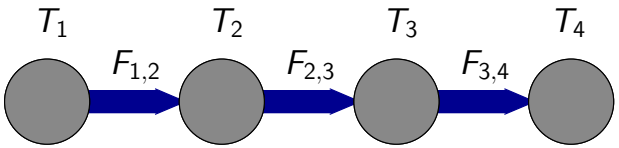
Definition

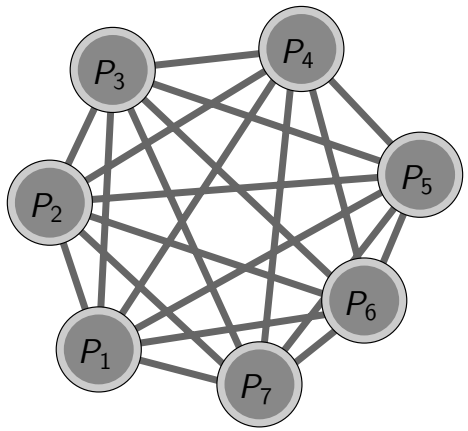
An allocation of the application graph to the platform graph is a function σ associating:

- to each task T_i: a processor $\sigma(T_i)$ that processes all instances of T_i assigned to the allocation
- to each file $F_{i,j}$: a set of communication links $\sigma(F_{i,j})$ that transfers this file from processor $\sigma(T_i)$ to processor $\sigma(T_j)$

Different mapping strategies

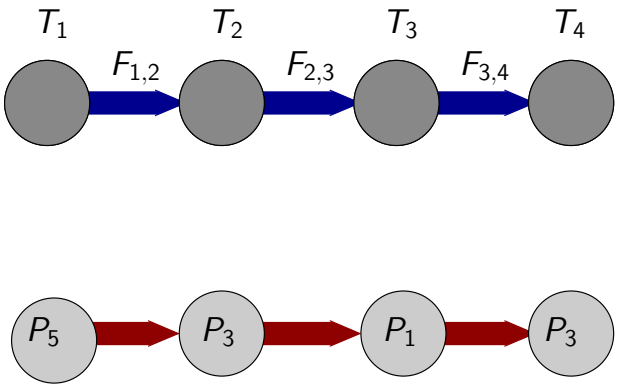
A small example

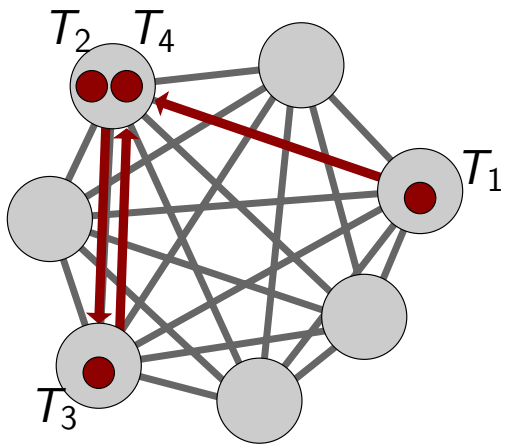




Simple solution, with a single allocation

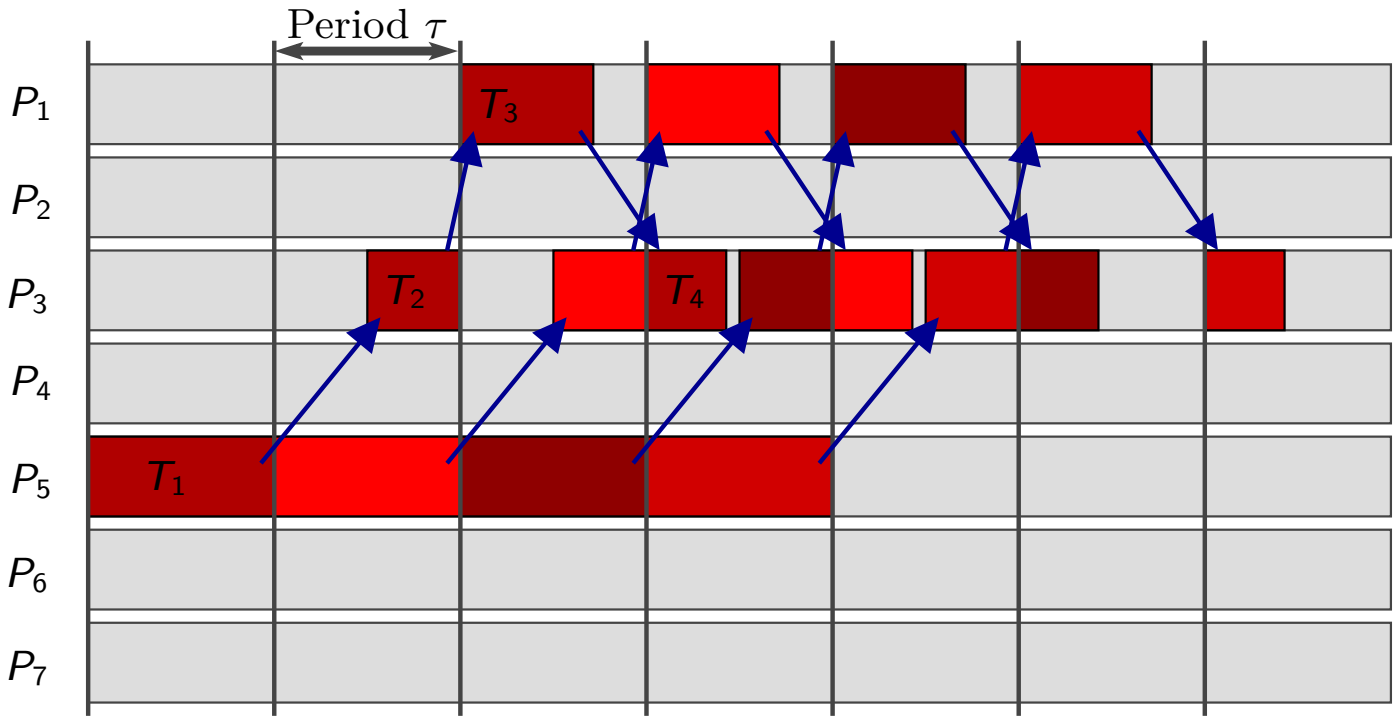
A mono-allocation schedule





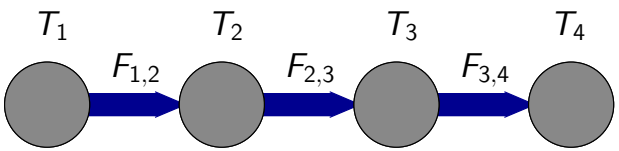
Simple solution, with a single allocation

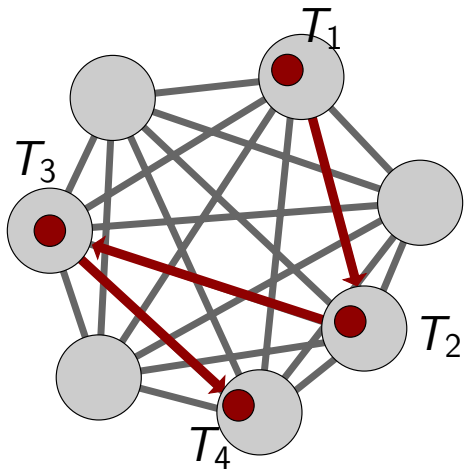
A mono-allocation schedule



Multi-allocation steady-state scheduling

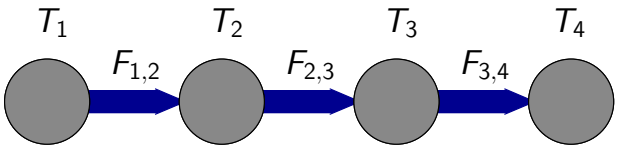
An optimal solution: multi-allocation steady-state scheduling

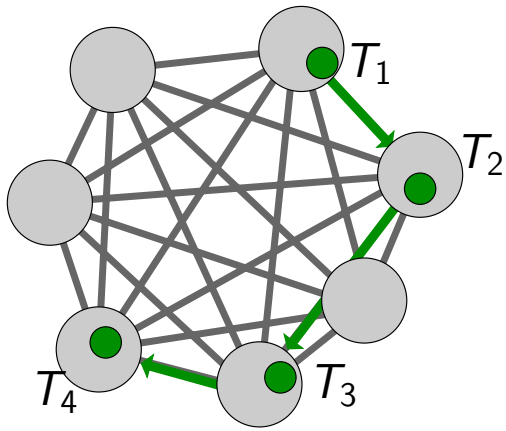




Multi-allocation steady-state scheduling

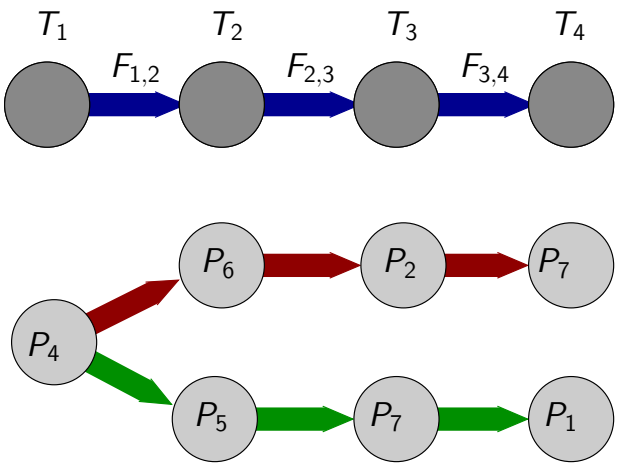
An optimal solution: multi-allocation steady-state scheduling

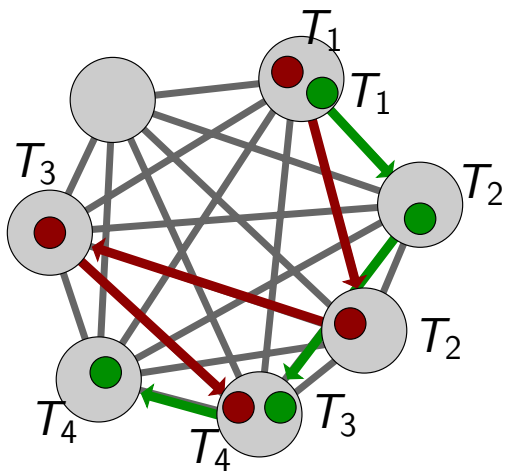




Multi-allocation steady-state scheduling

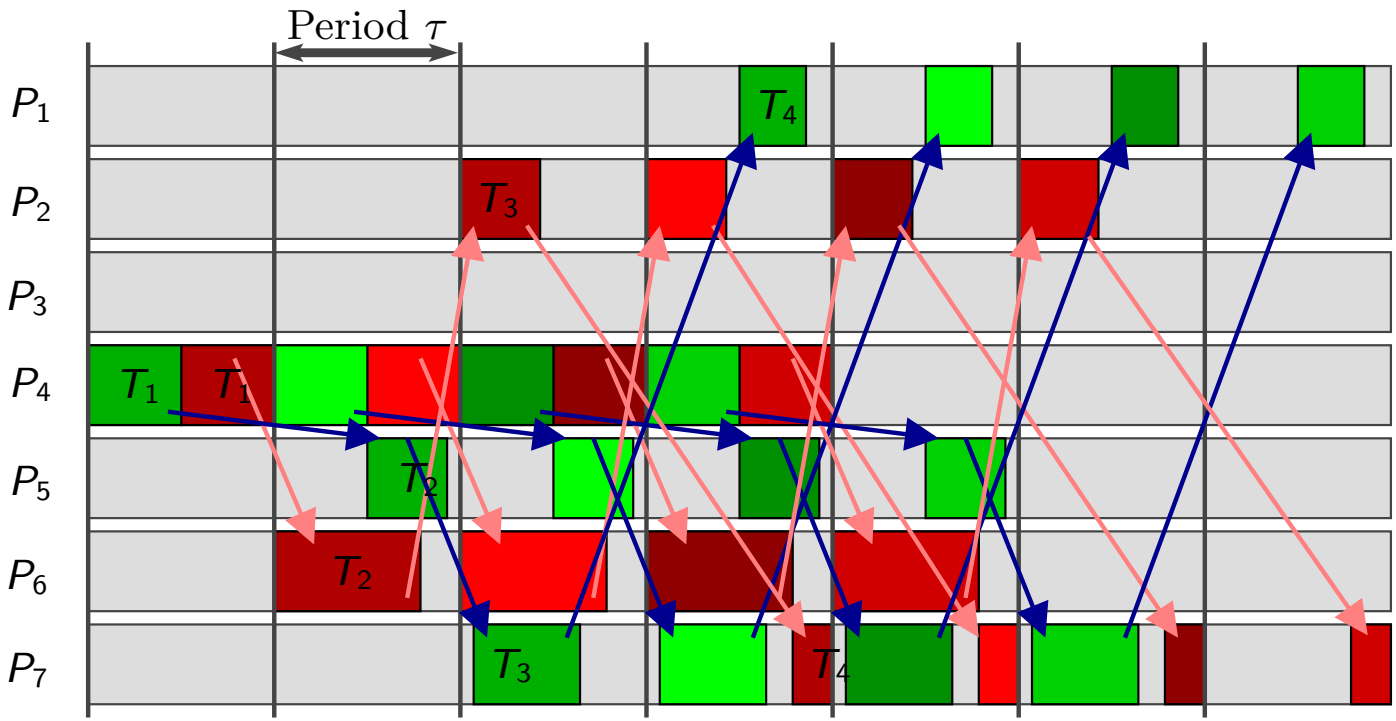
An optimal solution: multi-allocation steady-state scheduling





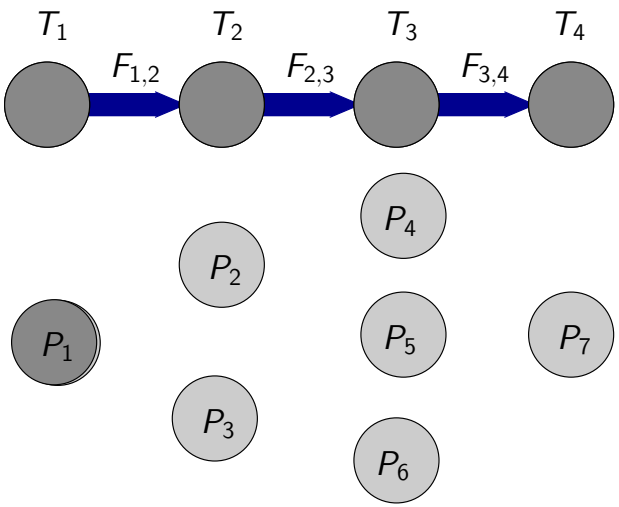
Multi-allocation steady-state scheduling

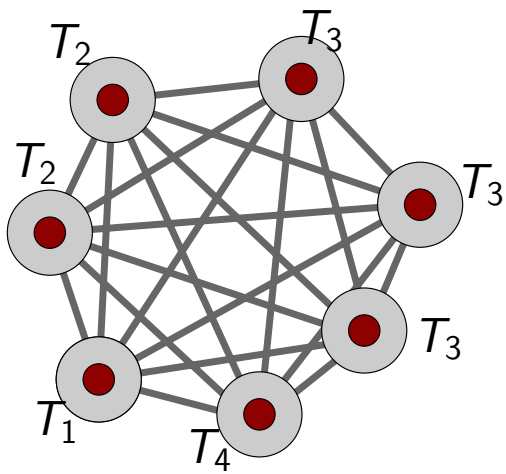
An optimal solution: multi-allocation steady-state scheduling



Mappings with replications

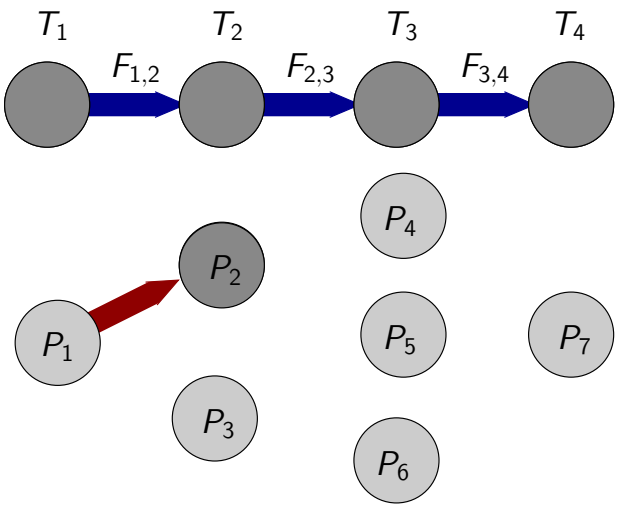
Round-robin distribution of replicated tasks

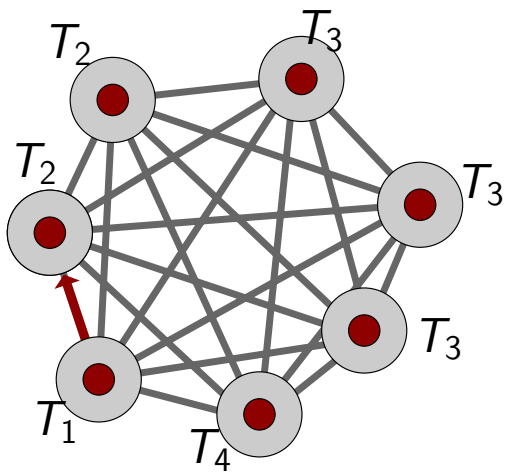




Mappings with replications

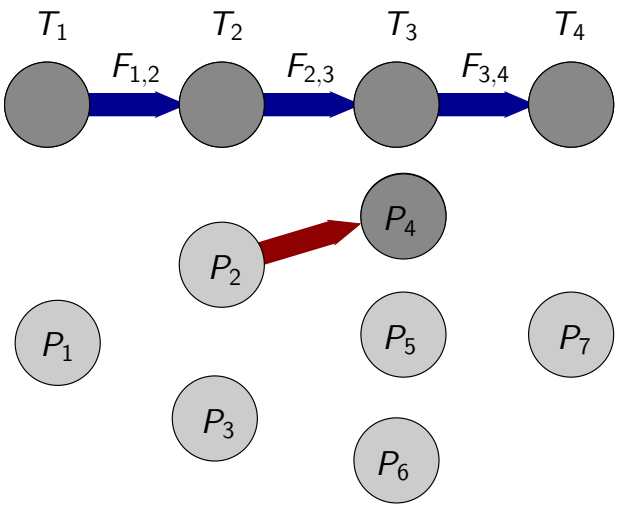
Round-robin distribution of replicated tasks

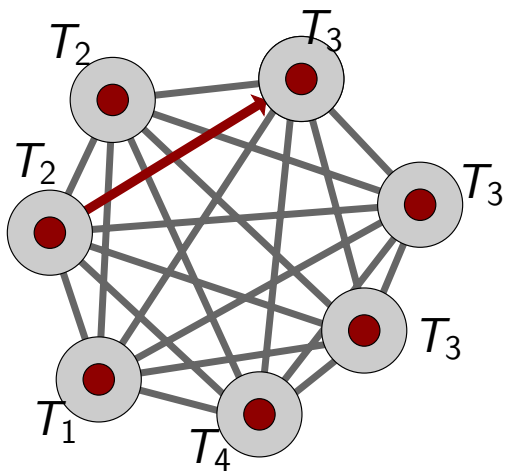




Mappings with replications

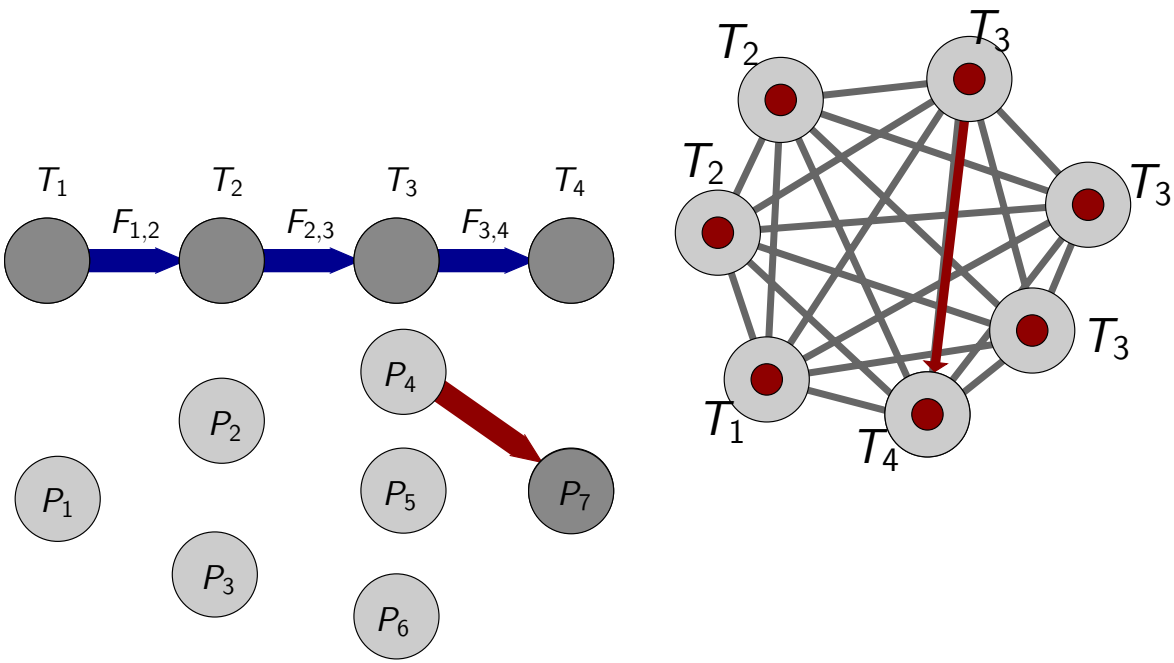
Round-robin distribution of replicated tasks





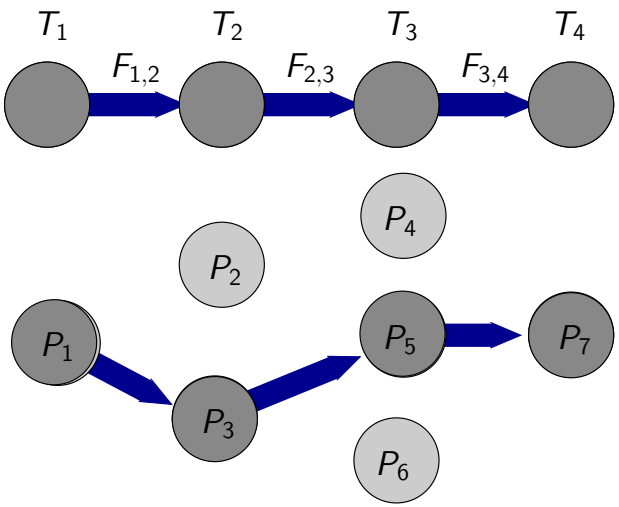
Mappings with replications

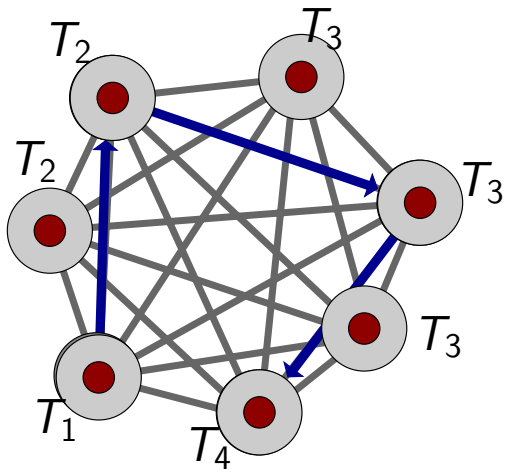
Round-robin distribution of replicated tasks



Mappings with replications

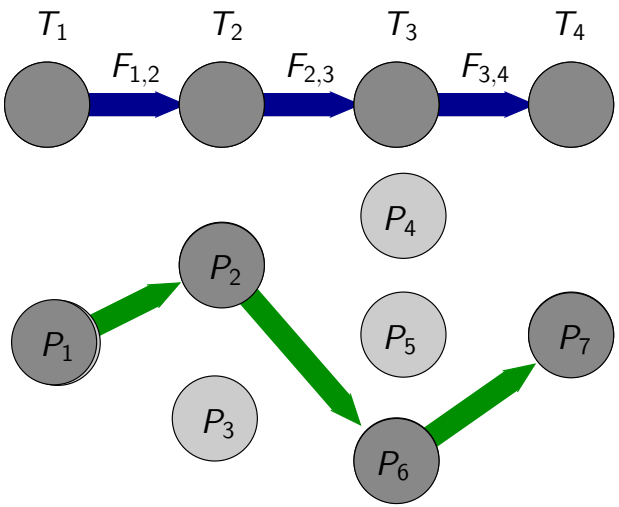
Round-robin distribution of replicated tasks

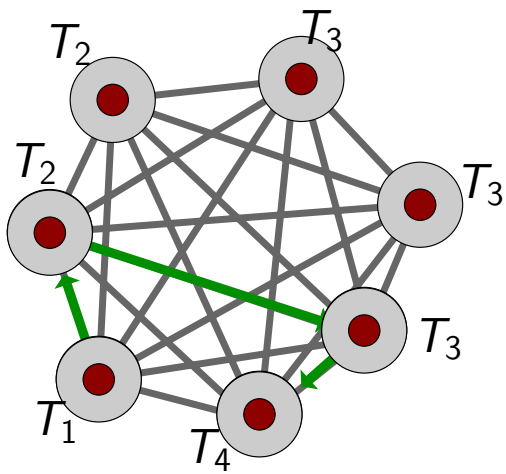




Mappings with replications

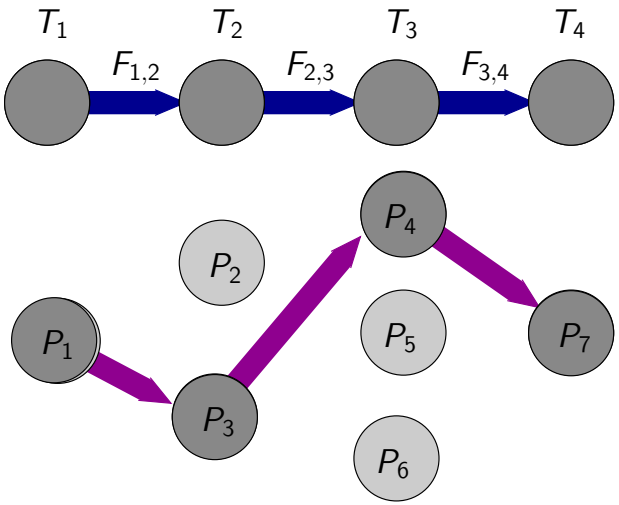
Round-robin distribution of replicated tasks

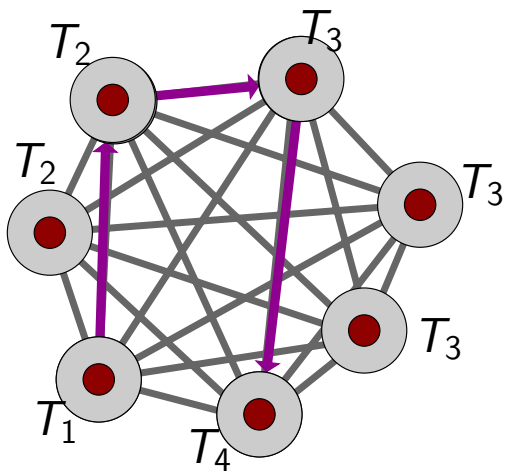




Mappings with replications

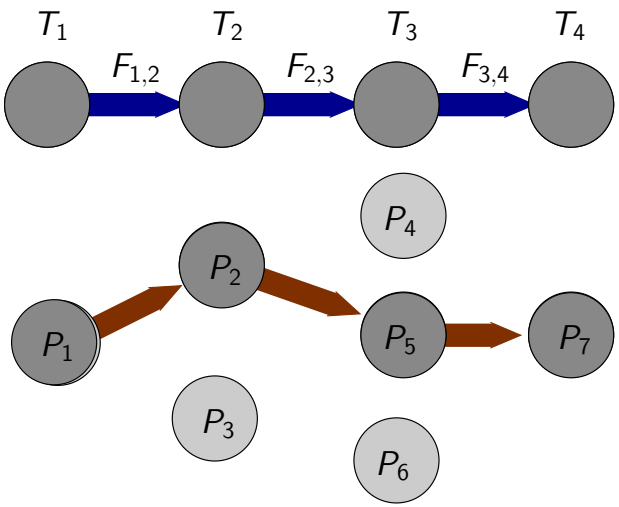
Round-robin distribution of replicated tasks

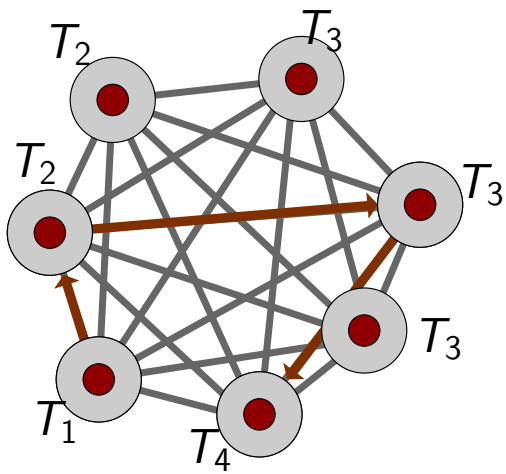




Mappings with replications

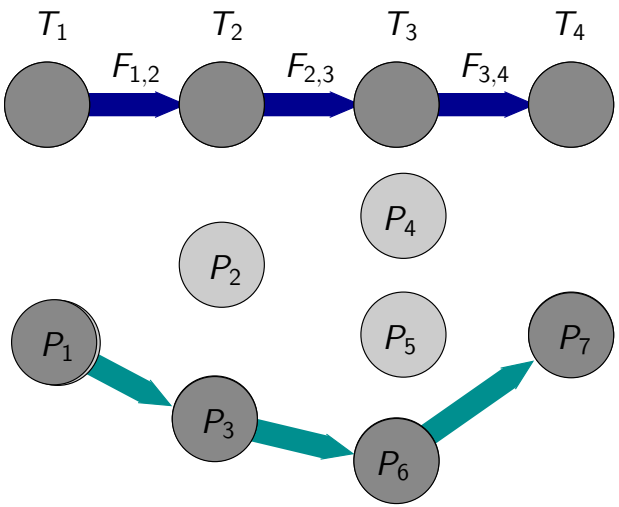
Round-robin distribution of replicated tasks

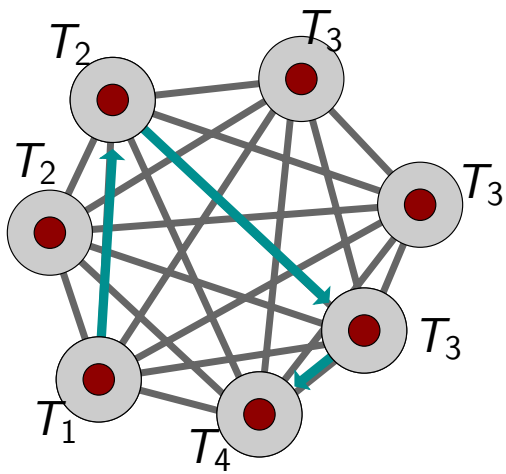




Mappings with replications

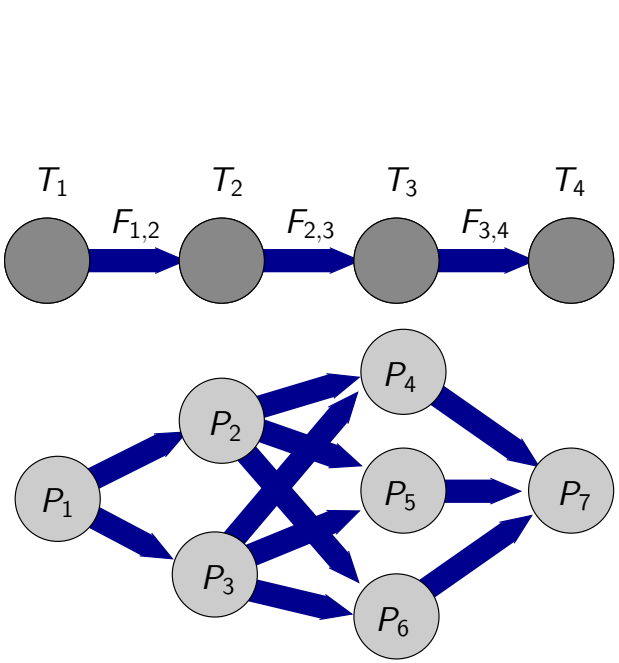
Round-robin distribution of replicated tasks

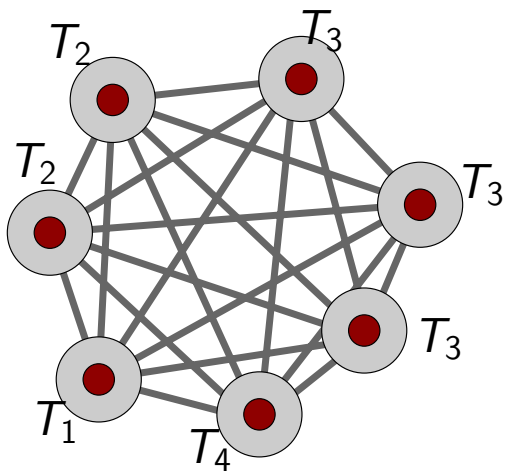




Mappings with replications

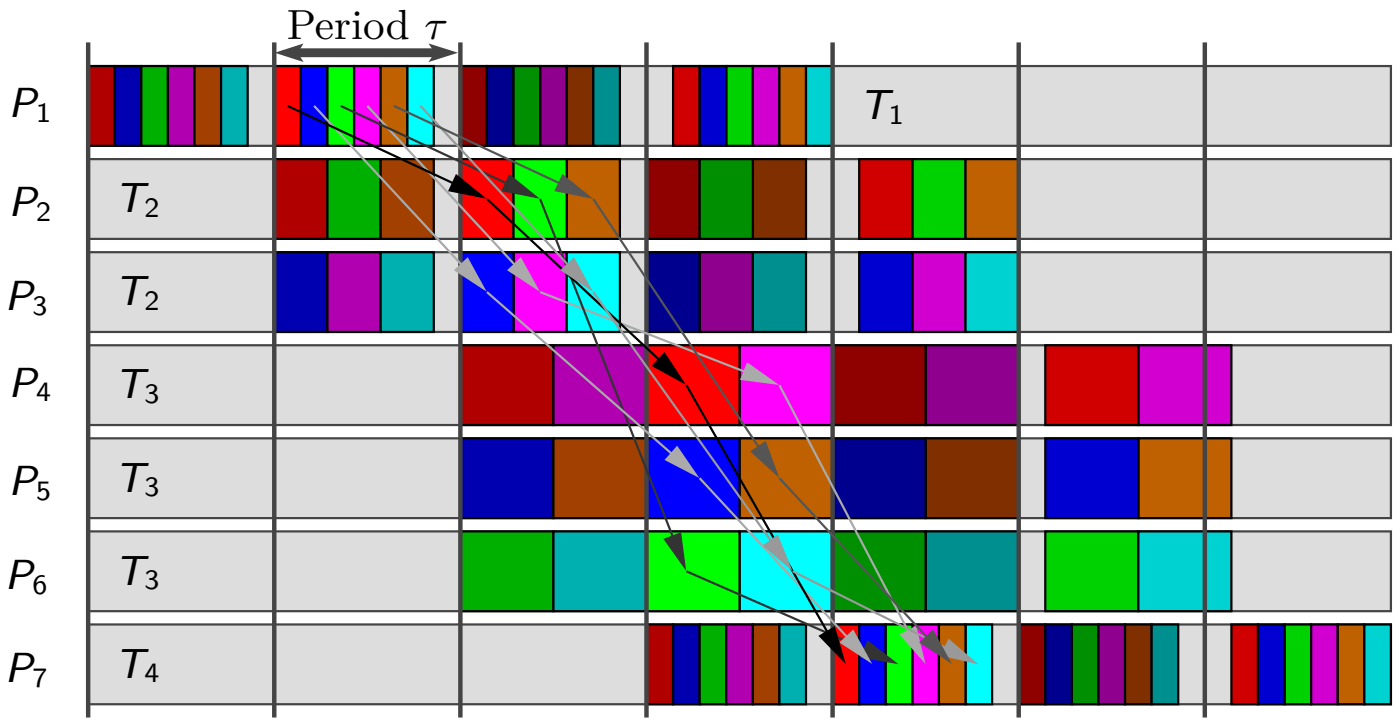
Round-robin distribution of replicated tasks





Mappings with replications

Round-robin distribution of replicated tasks



A short comparison of the three methods 1/3

Mono-allocation steady-state schedules

- Easy to implement ☺
- Handle stateful nodes ☺
- Smaller buffers ☺
- Less efficient schedules (stronger constraints) ☹

A short comparison of the three methods 2/3

General multi-allocation steady-state solution

- Optimal throughput ☺
- Polynomial computation time in almost all cases ☺
- Very long periods ☹
- Huge response time ☹
- Complex allocation schemes ☹
- Never fully implemented ☹

A short comparison of the three methods 3/3

Replication with Round-Round distribution

- Natural extension to a mono-allocation solution ☺
- No buffer required to keep initial order of data sets ☺
- Fully implemented solution (DataCutter) ☺
- Simple control, with closed form to determine processors ☺
- Less efficient schedules (stronger constraints) ☹
- May not fully exploit each resource ☹
- Hard to determine throughput ☹☹

Communication model

- Trade-off between realism and tractability
- Many different models

- *One-Port* model:
  - *Strict One-Port*: a processor can either compute or perform a single communication
  - Full-Duplex *One-Port*: a processor can either compute or simultaneously send and receive data
  - *One-Port* with overlap of computation by communications
- *Bounded Multiport* model: several concurrent communications, respecting resource bandwidths

- Linear cost model: communication time proportional to data size

Communication model

Which model to choose?

- Computing resources (single processors vs. multi-core processors or dedicated co-processors, . . .)
- Network (homogeneous network vs. a server with large bandwidth connected to many light clients)
- Applications and software resources (blocking communications vs. multithreaded libraries)
- Previously studied models
- Algorithmic complexity! (non-trivial results with simpler models vs. realism with complex models)

Outline

Mono-allocation steady-state scheduling

Main idea

All instances of a task are processed by the same resource

- Less efficient schedules (stronger constraints) ☹
- A single allocation ☺

- *Bounded Multiport* model
  - Limited incoming bandwidth $\mathrm{bw}_q^{\mathrm{in}}$
  - Limited outgoing bandwidth $\mathrm{bw}_q^{\mathrm{out}}$
  - Limited bandwidth per link $\mathrm{bw}_{q,r}$
- Period τ, throughput ρ: $\rho = 1/\tau =$ critical resource cycle-time

Complexity

Problem DAG-Single-Alloc

Given a directed acyclic application graph, a platform graph, and a bound B, is there an allocation with throughput $\rho \geq B$?

Theorem

DAG-Single-Alloc is NP-complete

Variables and constraints due to the application

- $\alpha_q^k = 1$ if task T_k is processed on processor P_q, and $\alpha_q^k = 0$ otherwise
- Each task is processed exactly once:

$$\forall T_k, \quad \sum_{P_q} \alpha_q^k = 1$$

- $\beta_{q,r}^{k,l} = 1$ if file $F_{k,l}$ is transferred using path $P_q \rightsquigarrow P_r$, and $\beta_{q,r}^{k,l} = 0$ otherwise
- A file transfer must originate from where the file was produced:

$$\beta_{q,r}^{k,l} \leq \alpha_q^k$$

Constraints on computations

- The processor computing a task must hold all necessary input data, i.e., either it received or it computed all required input data:

$$\alpha_r^k + \sum_{P_q \rightsquigarrow P_r} \beta_{q,r}^{k,l} \geq \alpha_r^l$$

- The computing time of a processor is not larger that τ:

$$\sum_{T_k} \alpha_q^k \times w_{q,k} \leq \tau$$

Constraints on communications

- The amount of data carried by the link $P_q \to P_r$ is:

$$d_{q,r} = \sum_{\substack{P_s \rightsquigarrow P_t \text{ with} \\ P_q \to P_r \in P_s \rightsquigarrow P_t}} \sum_{F_{k,l}} \beta_{s,t}^{k,l} \times \text{data}_{k,l}$$

- The link bandwidth must not be exceeded:

$$\frac{d_{q,r}}{\text{bw}_{q,r}} \leq \tau$$

- The output bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{\text{bw}_q^{\text{out}}} \leq \tau$$

- The input bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{\text{bw}_r^{\text{in}}} \leq \tau$$

Objective

Minimize the maximum time τ spent by all resources

Theorem

An optimal solution of the above linear program describes an allocation with maximal throughput

Summary

- Solutions based on mixed-linear programs
- NP-complete problem ☹
- Need for clever heuristics ☺

Greedy mapping strategies

- Simple mapping:
  - assign "largest" task to best processor
  - continue with second "largest" task, assign it to the processor that decreases the least the throughput
  - ...

- Refined greedy:
  - take communication times into account when sorting tasks
  - when mapping a task, select the processor such that the maximum occupation time of all resources (processors and links) is minimized

Rounding the linear program

1. Solve the linear program over the rationals
2. Based on the rational solution, select an integer variable α_i^k:

   RLP-max:
   - Select the α_i^k with maximum value
   - Set α_i^k to 1

   RLP-rand:
   - Select a task T_k not yet mapped
   - Randomly choose a processor P_i with probability α_i^k
   - Set α_i^k to 1

3. Goto step 1 until all variables are set

Delegating computations

- Start from solution where all tasks are processed by a single processor
- Try to move a (connected) subset of tasks to another processor to increase throughput
- Repeat this process until no more improvement is found

Several issues to overcome:

- Find interesting groups of tasks to move
  - for all tasks, we test all possible immediate neighborhoods, and then try to increase the group along chains
- Hard to find a good evaluation metric: some moves do not directly decrease throughput, but are still interesting
  - for a given mapping, we sort all resource occupation times by lexicographical order, and use the ordered list instead of the throughput in comparisons

Neighborhood-centric strategy

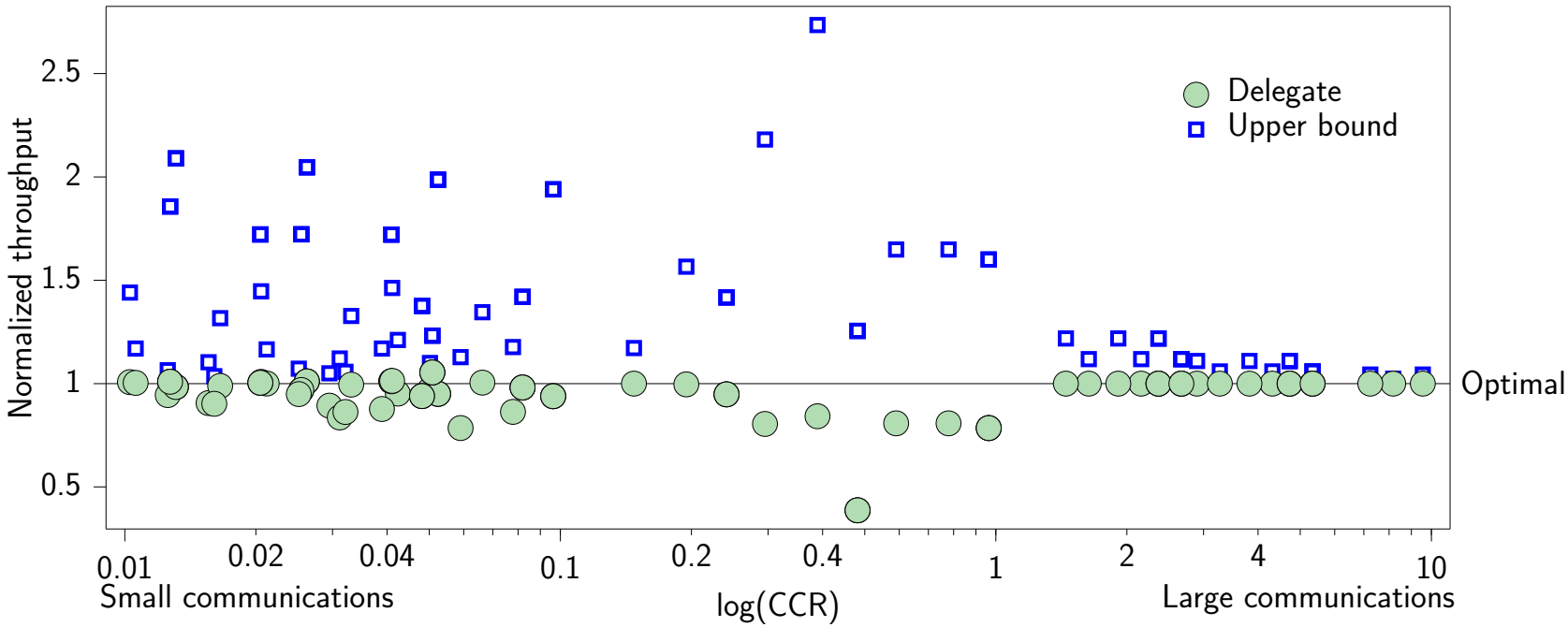
- First, evaluate the cost of any task and its immediate neighbors on an idle platform
- Cost of a task: maximum occupation time over all resources
- Consider each task T_k ordered by non-increasing cost:
  - Evaluate the mapping of T_k and its neighbors on each processor
  - Definitely assign T_k to best processor

- Same evaluation metric as Delegate

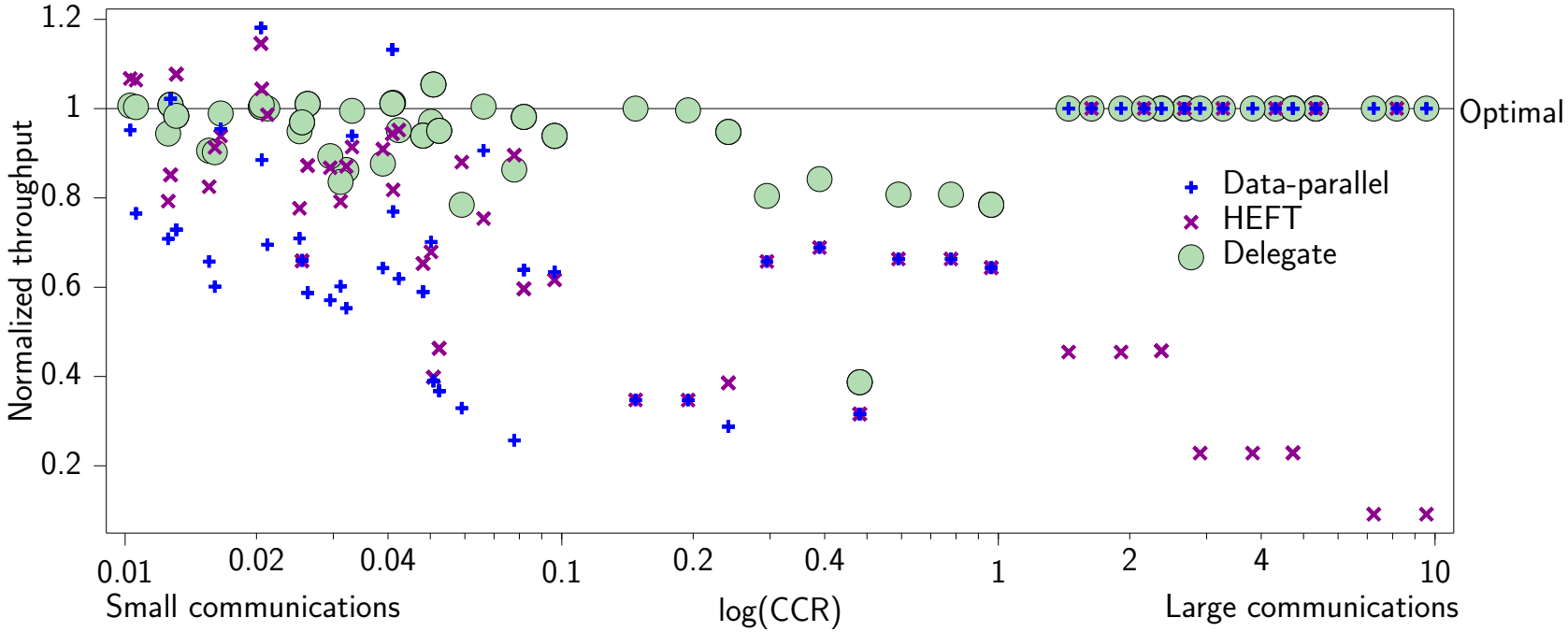
Performance evaluation – methodology

- Reference heuristics: HEFT, Data-Parallel, Clustering
- LP and MLP solved with Cplex 11
- Simulations done using SimGrid
- Platforms: actual Grids, from SimGrid repository
  (only a subset of processors is available for computation)
- Applications: random task graphs + one real application
  - "Small problems": 8–12 tasks
  - "Large problems": up to 47 tasks (MLP not used)
  - for each application, we compute a CCR $= \frac{\text{communications}}{\text{computations}}$
  - we try to cover a large CCR range

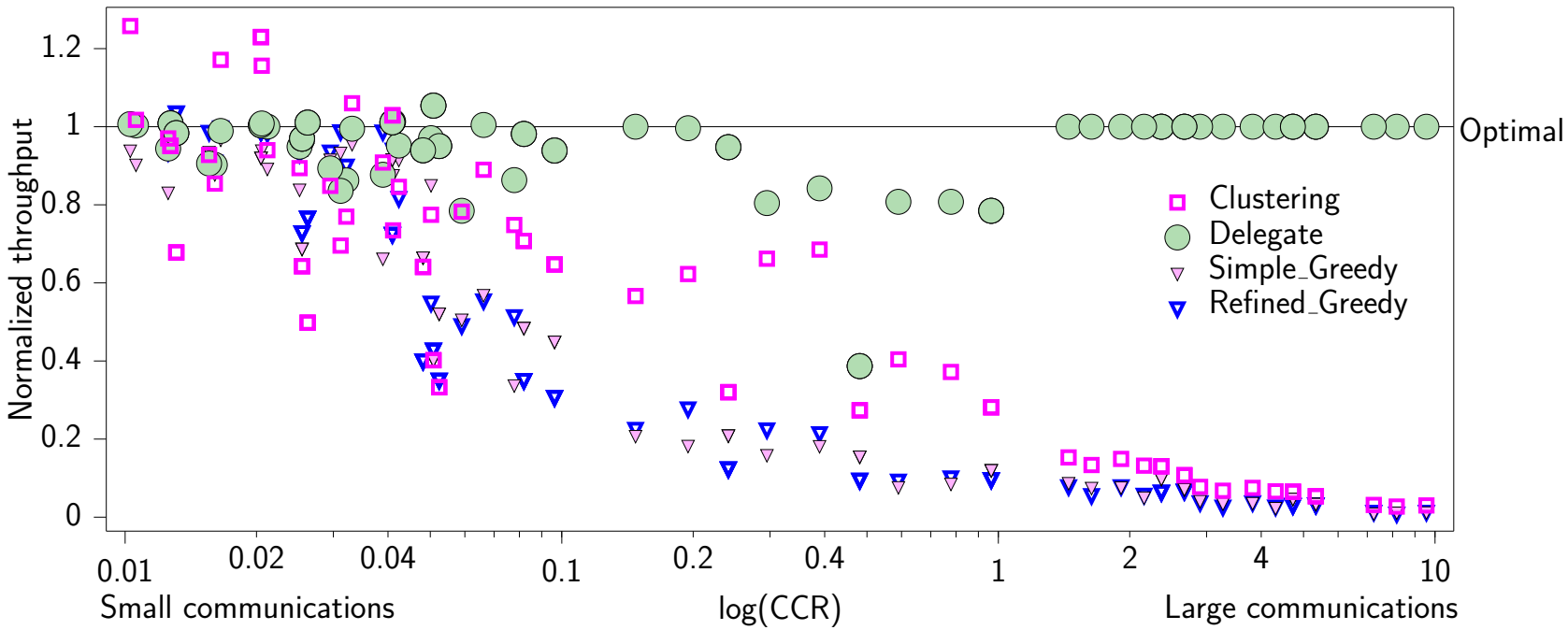
Performance evaluation – results on small problems



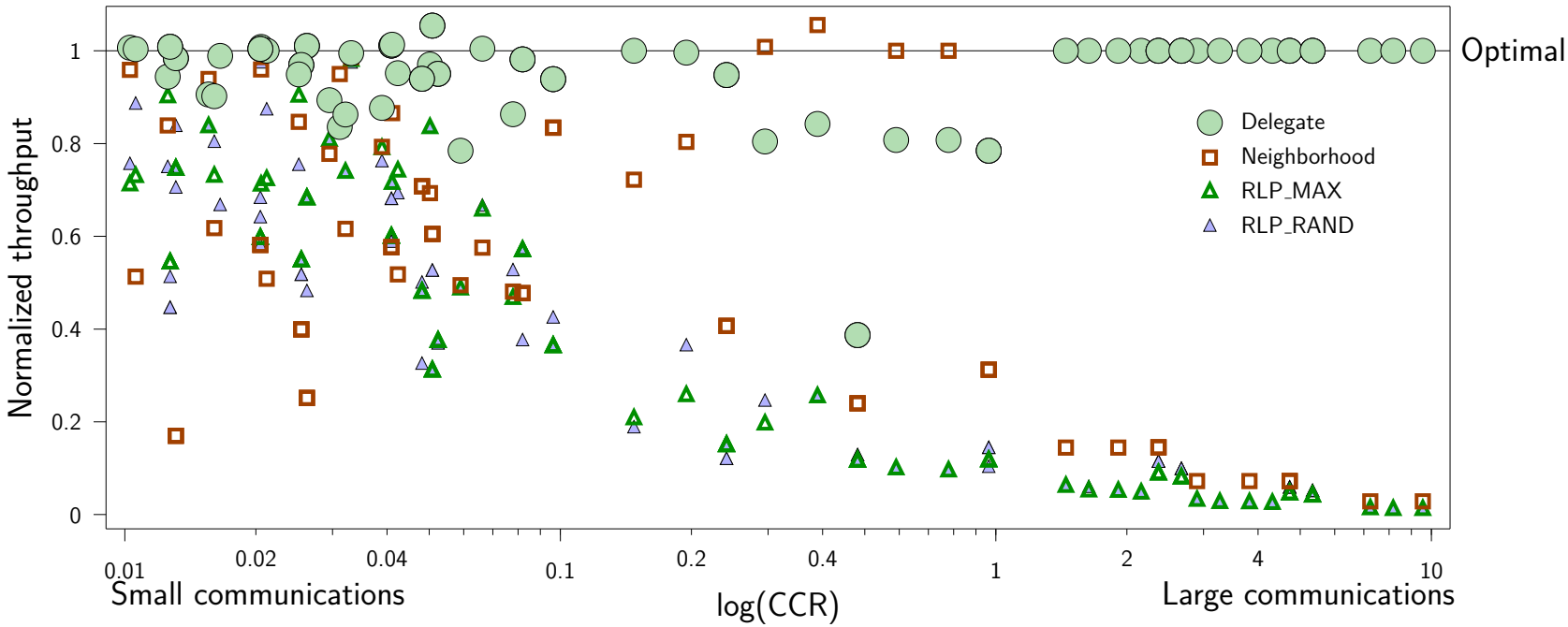
Performance evaluation – results on small problems



Performance evaluation – results on small problems



Performance evaluation – results on small problems



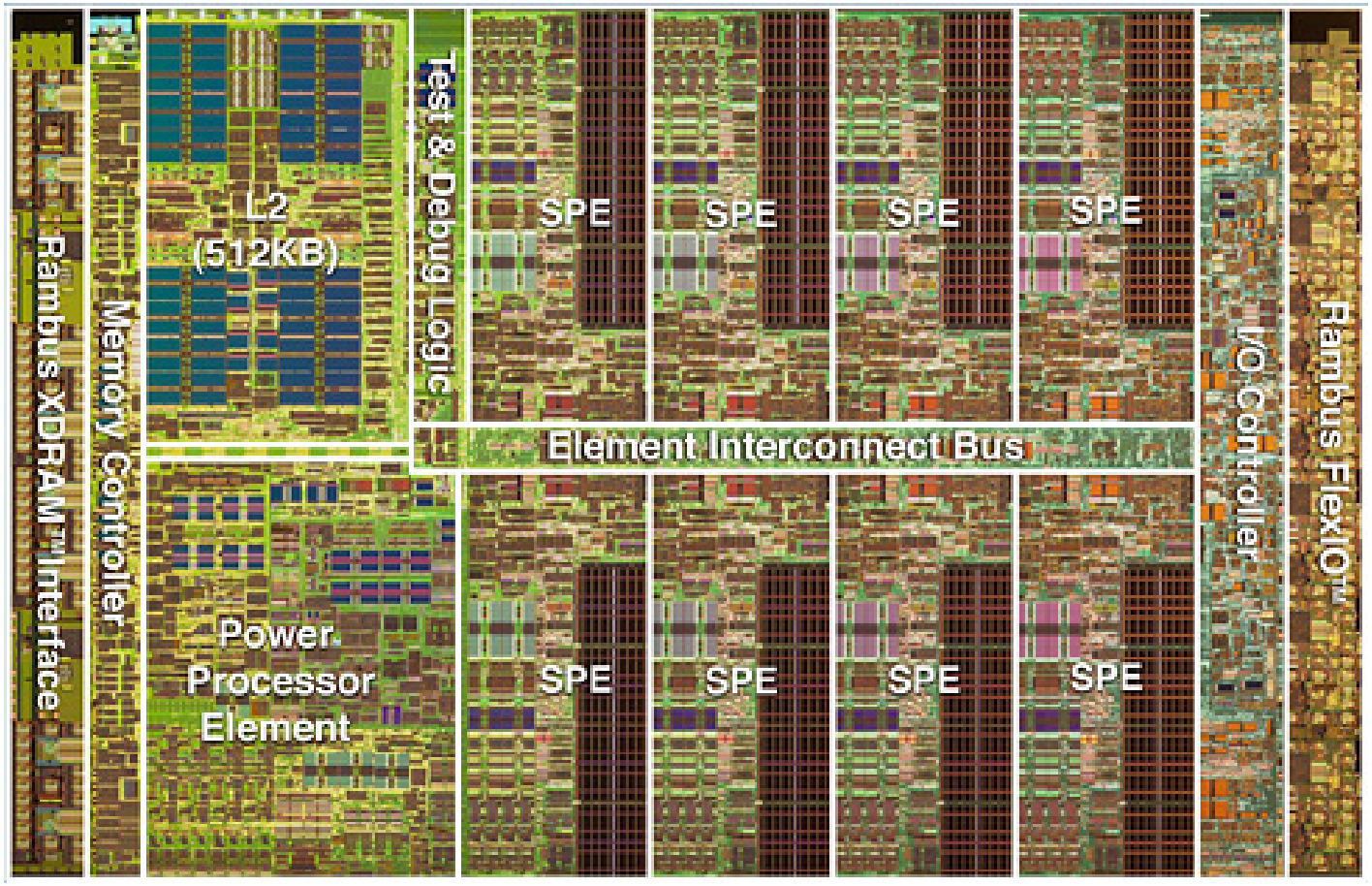
Summary

- Mono-allocation strategies are close to multi-allocation ones
- Outperform HEFT in most cases
- Optimal MLP solution restricted to small problems
- Efficient heuristics handle larger problems

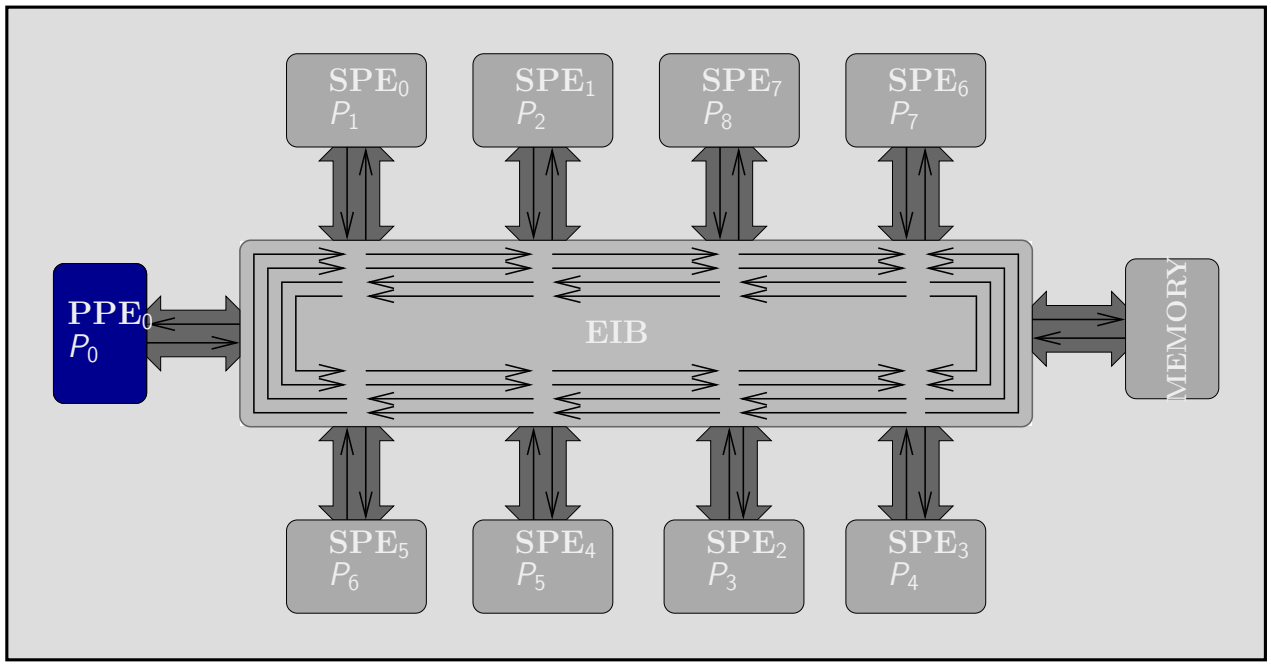
Outline

A short introduction to the Cell

- Joint work of Sony, Toshiba and IBM
- Non-standard architecture

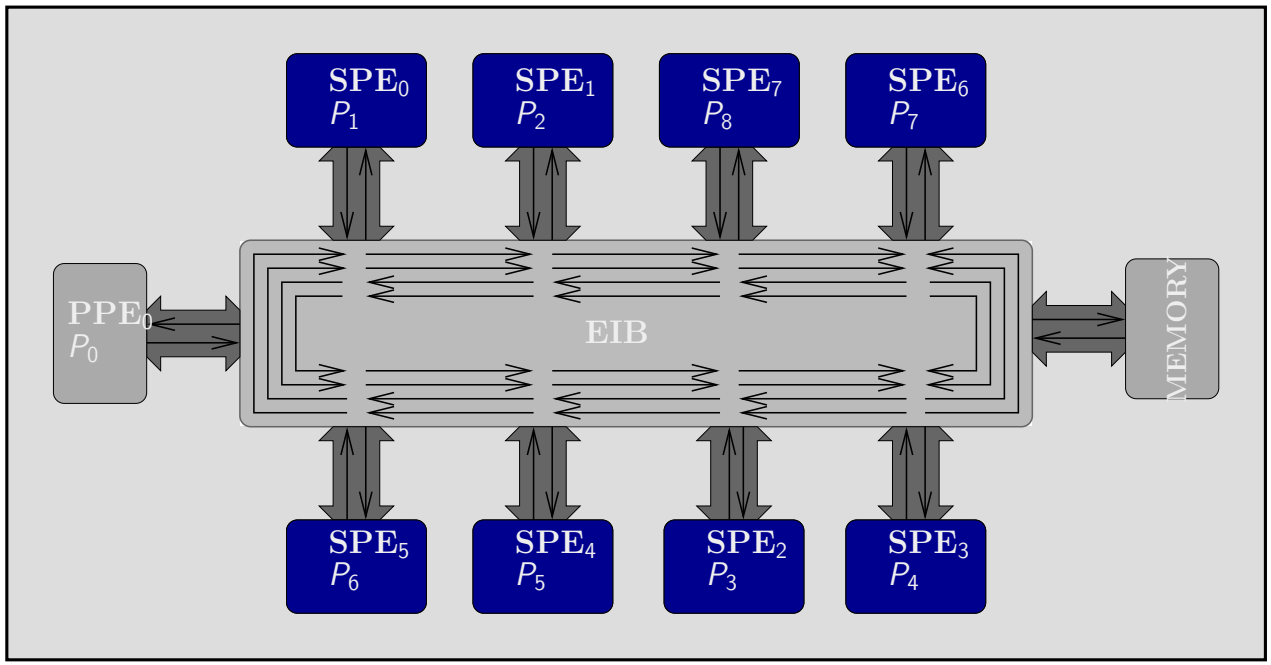


A theoretical vision of the Cell



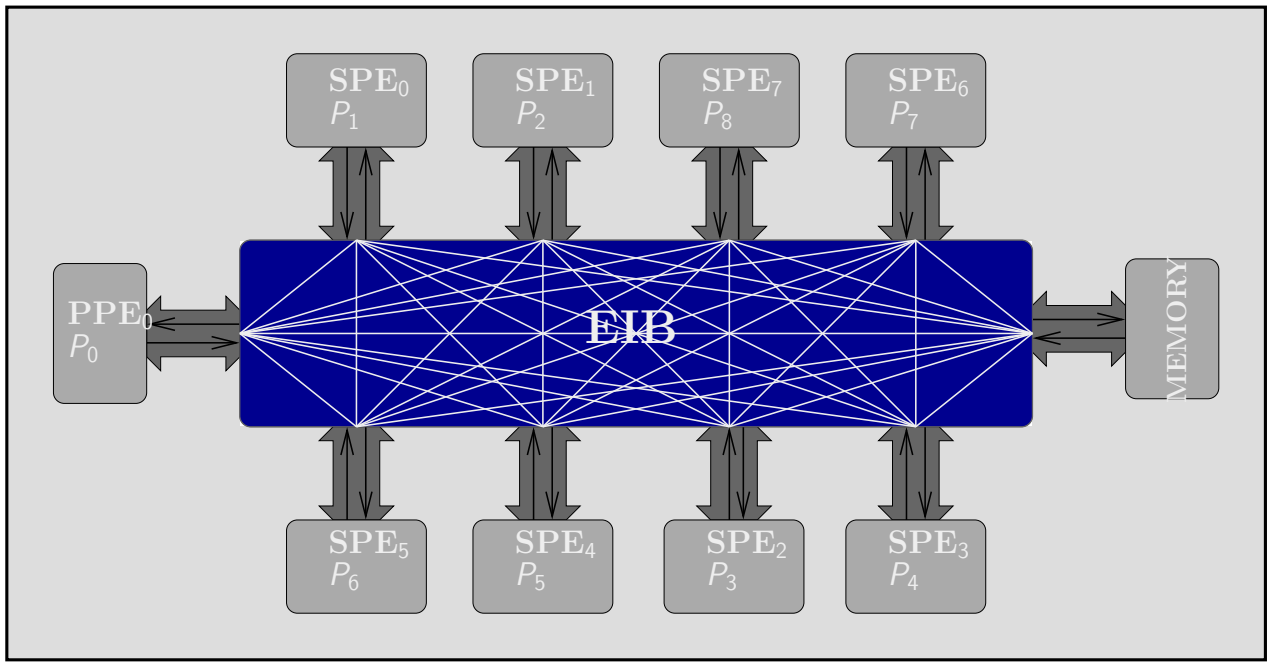
- One Power core (**PPE**) P_0
  standard processor, direct access to memory and L1/L2 cache

A theoretical vision of the Cell



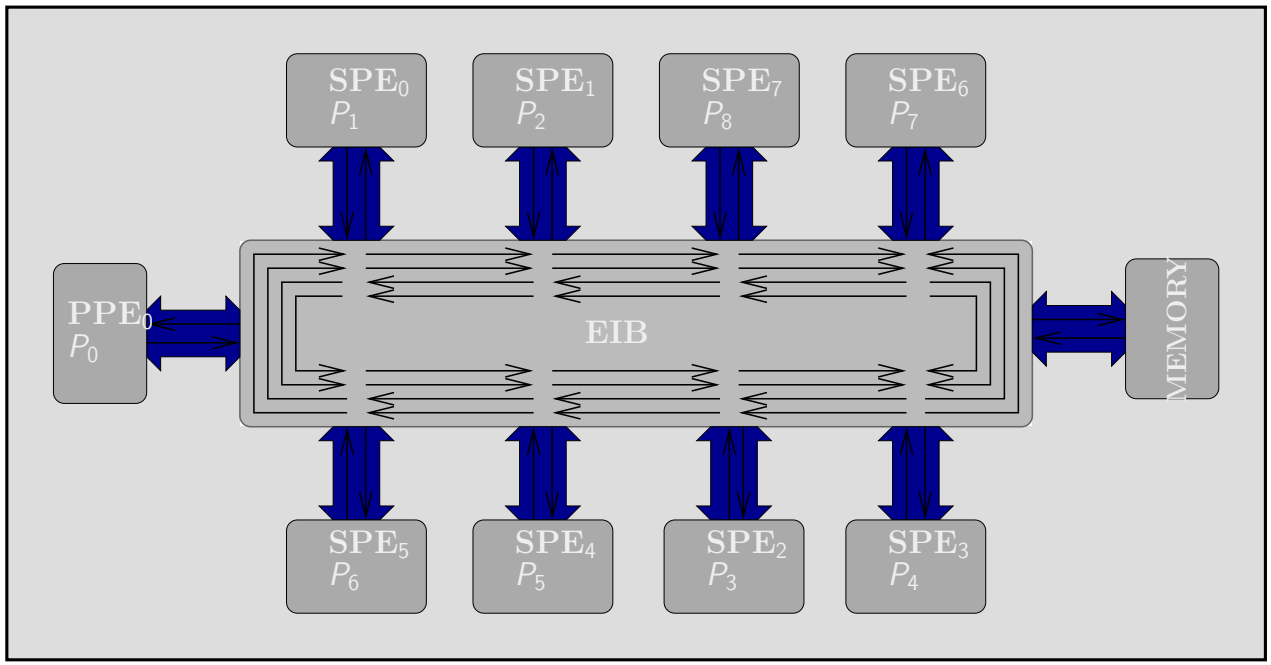
- Eight Synergistic Processing Elements (**SPE**s) $P_1, \ldots, P_8$
  256-kB Local Stores, dedicated asynchronous DMA engine

A theoretical vision of the Cell



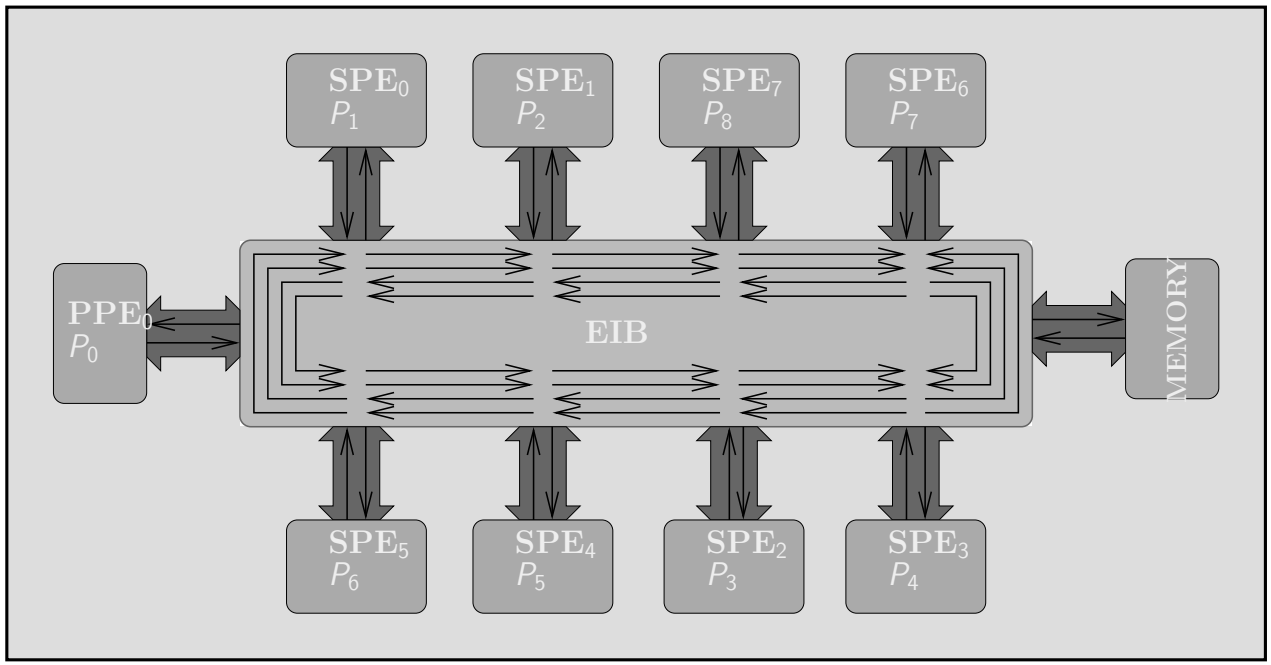
- Element Interconnect Bus (**EIB**)
  200 GB/s $\rightarrow$ no contention

A theoretical vision of the Cell



- Bidirectional communication link between an element and the **EIB** bandwidth $\mathrm{bw} = 25\mathrm{GB/s}$

A theoretical vision of the Cell



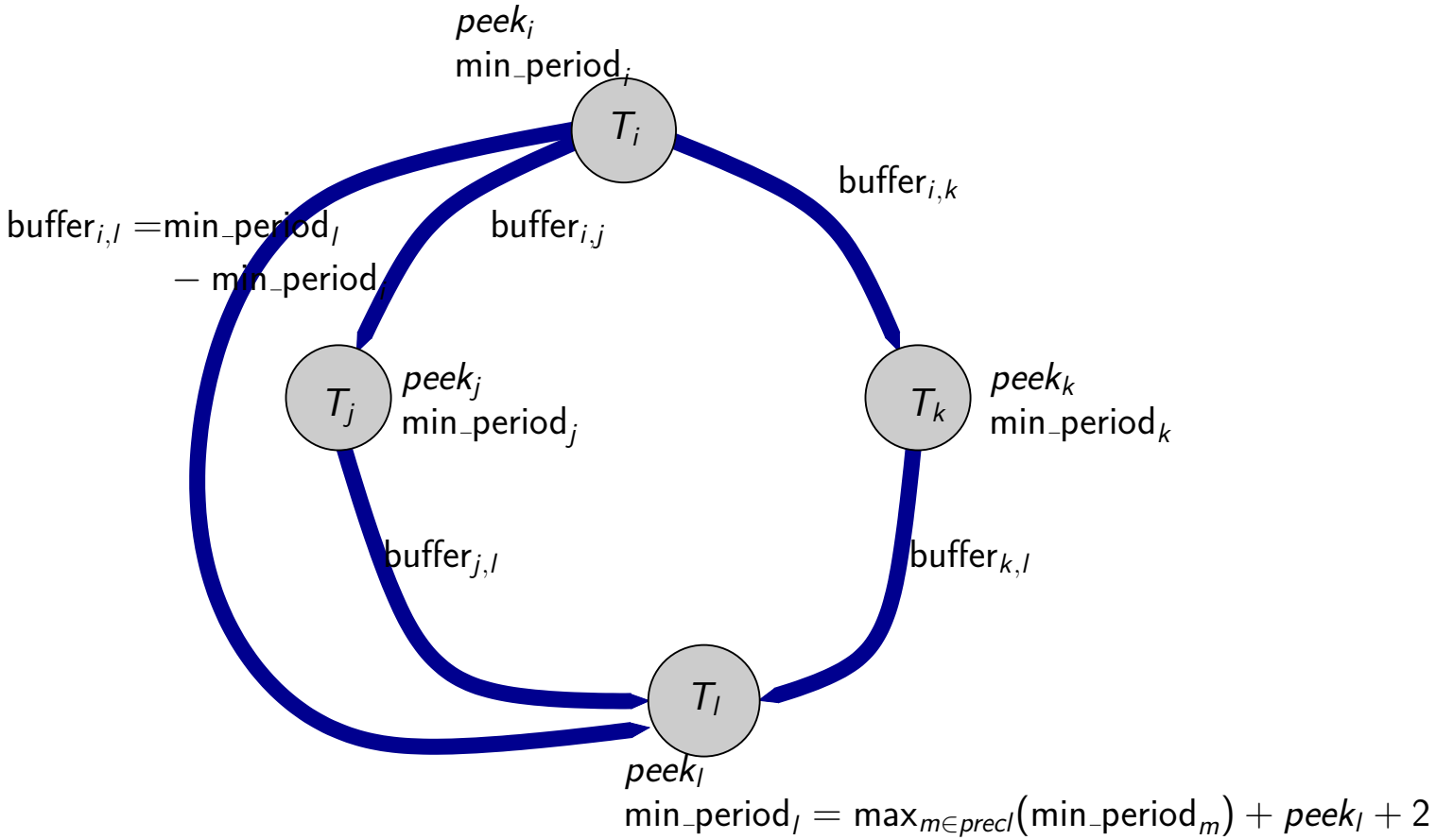
- Limited DMA stack:
  - at most 16 simultaneous incoming communications for each **SPE**
  - at most 8 simultaneous communications between a **SPE** and the **PPE**

Application model

- Previously-used task graph model: $G_A = (V_A, E_A)$
- Many data sets
- Some enhancements:
  - $read_k$: data to read before executing T_k
  - $write_k$: data to write after the execution of T_k
  - $peek_k$: number of next data sets to receive before executing T_k
- Two computations times for T_k: $w_{PPE}(T_k)$ and $w_{SPE}(T_k)$

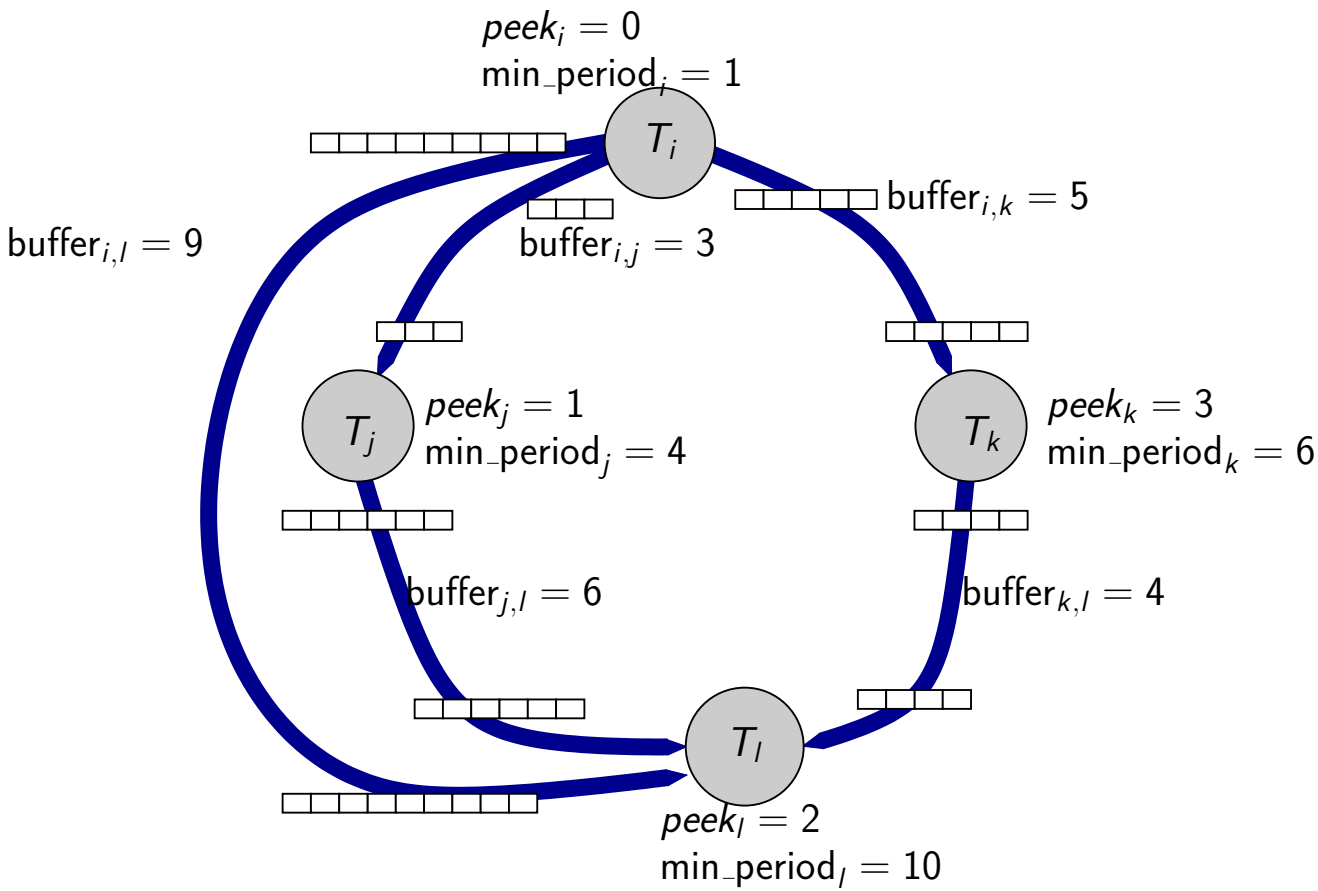
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



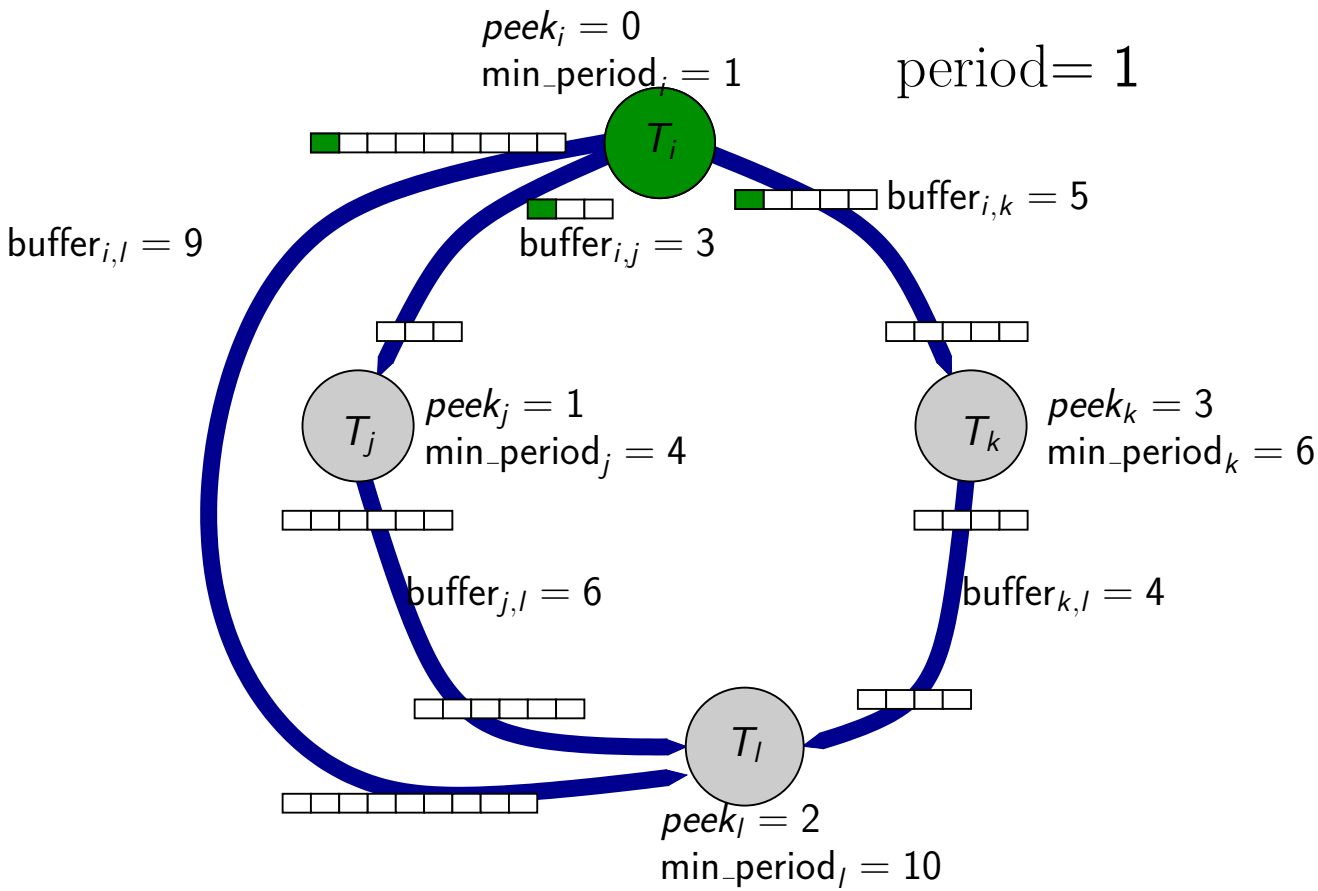
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



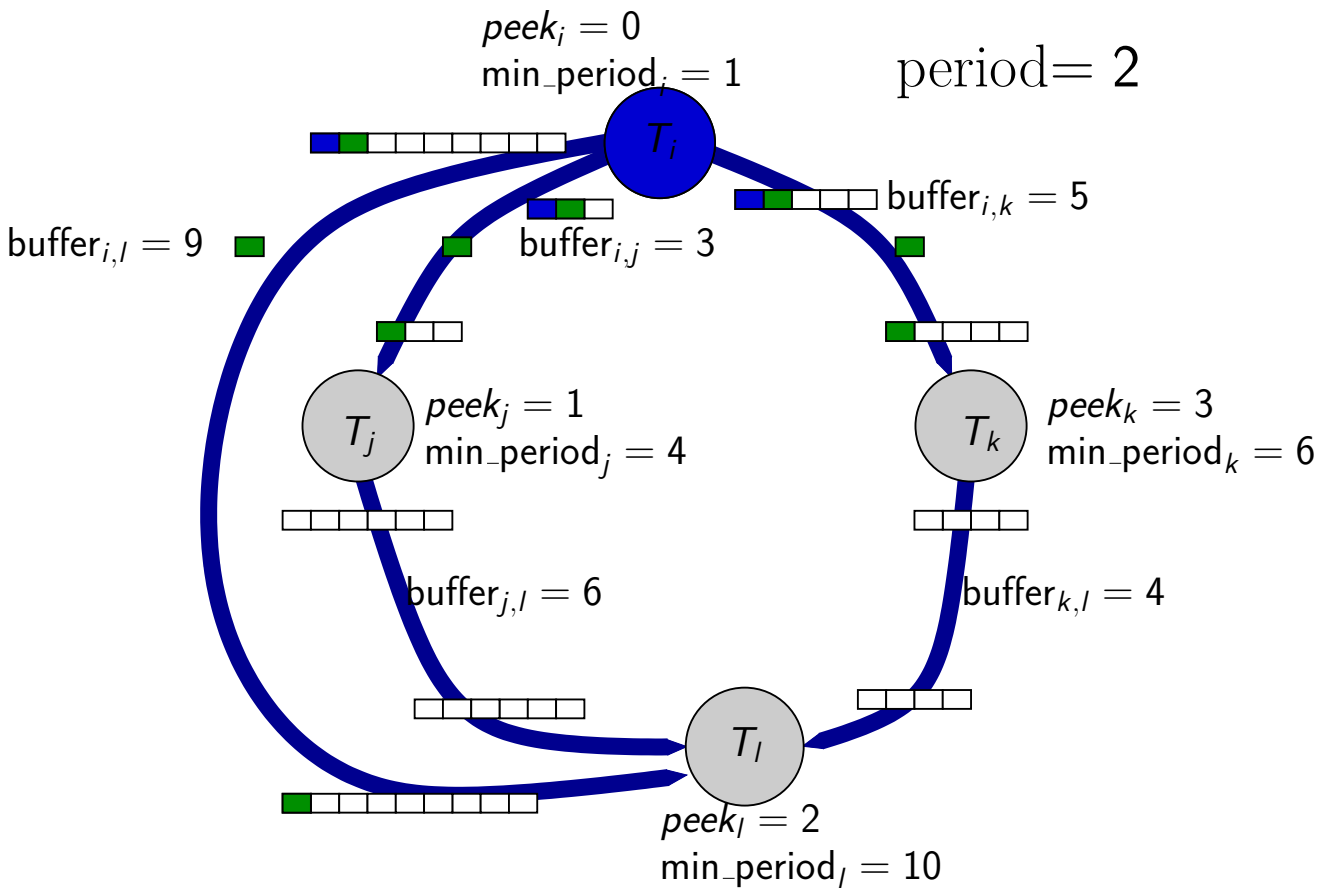
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



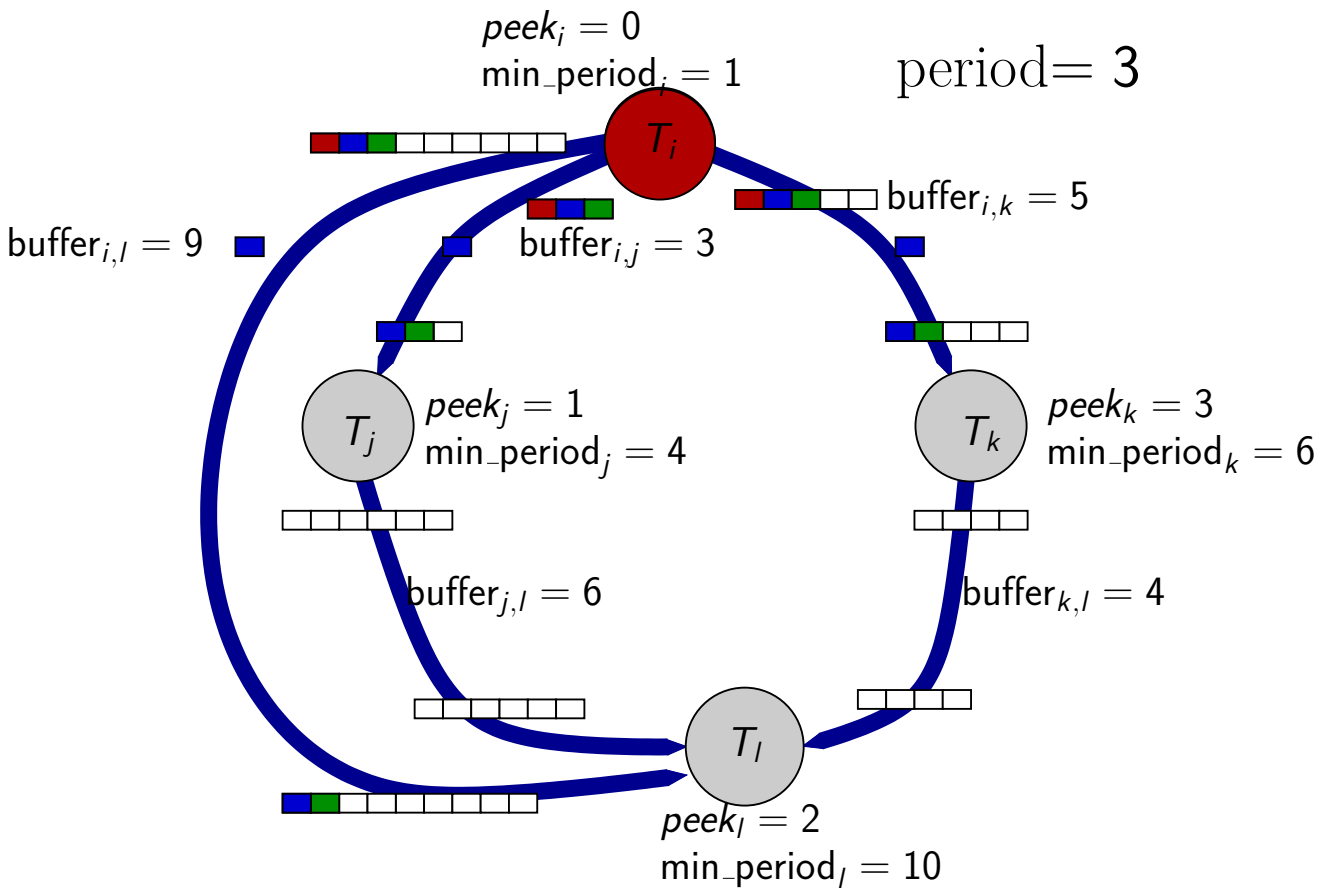
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



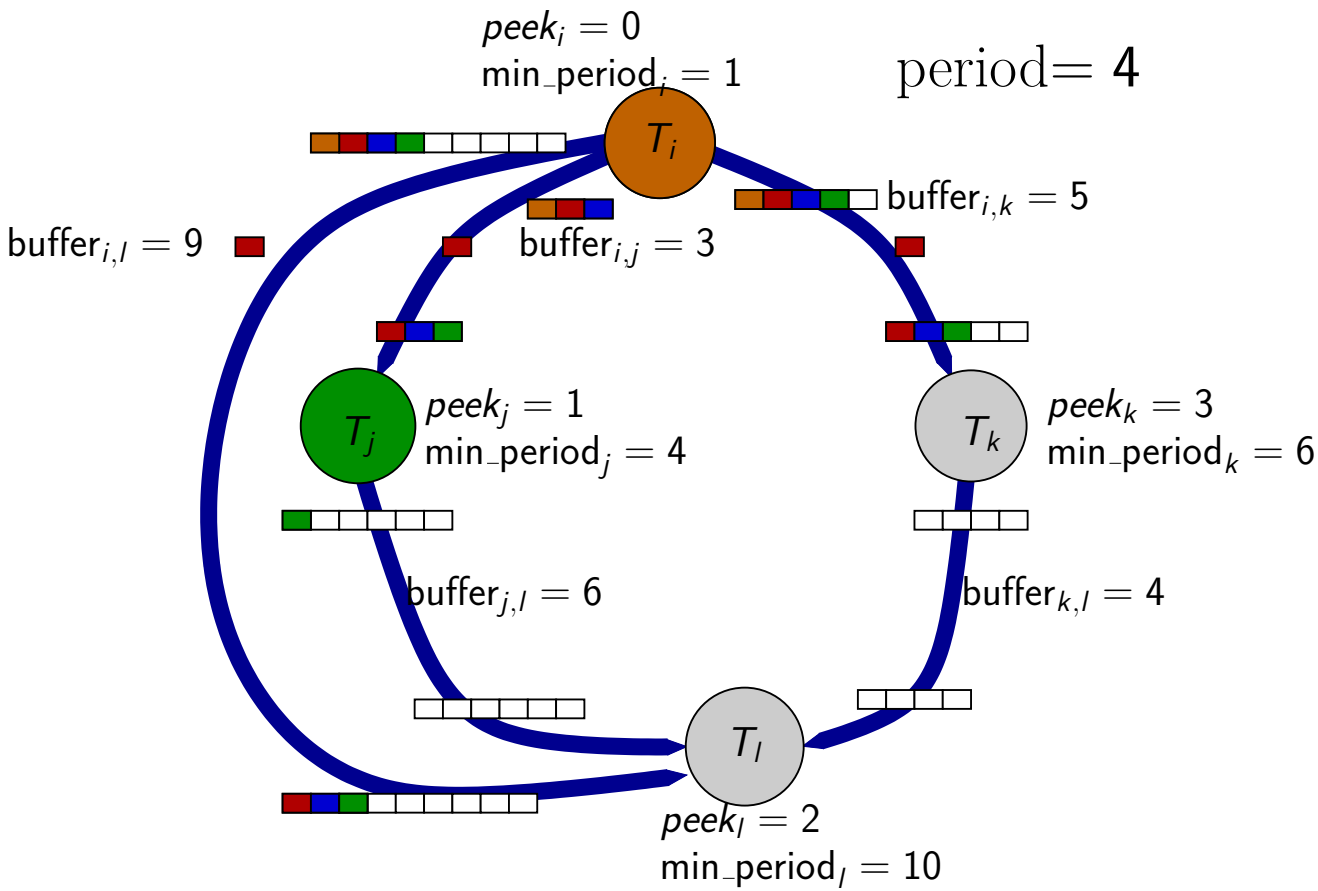
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



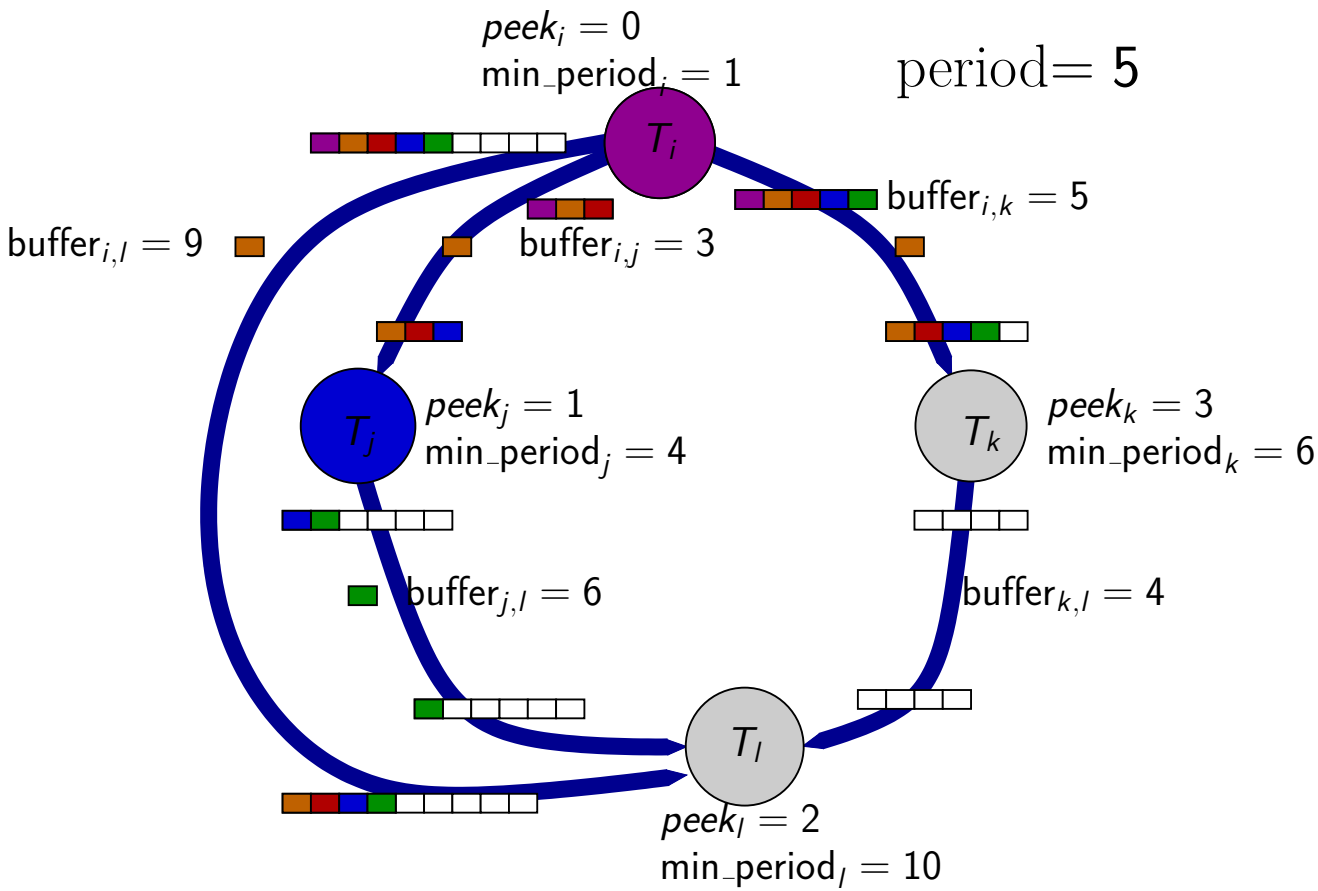
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



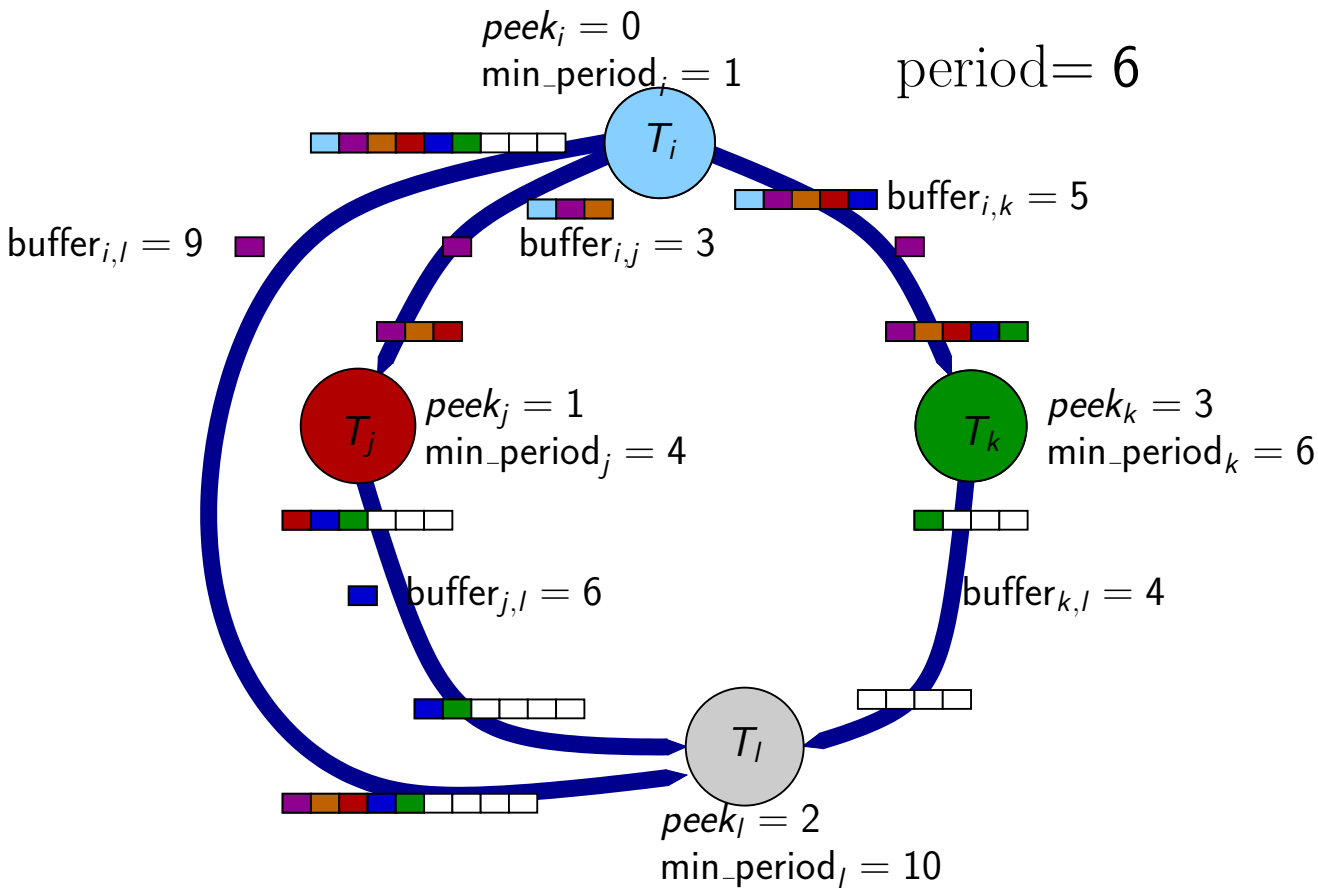
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



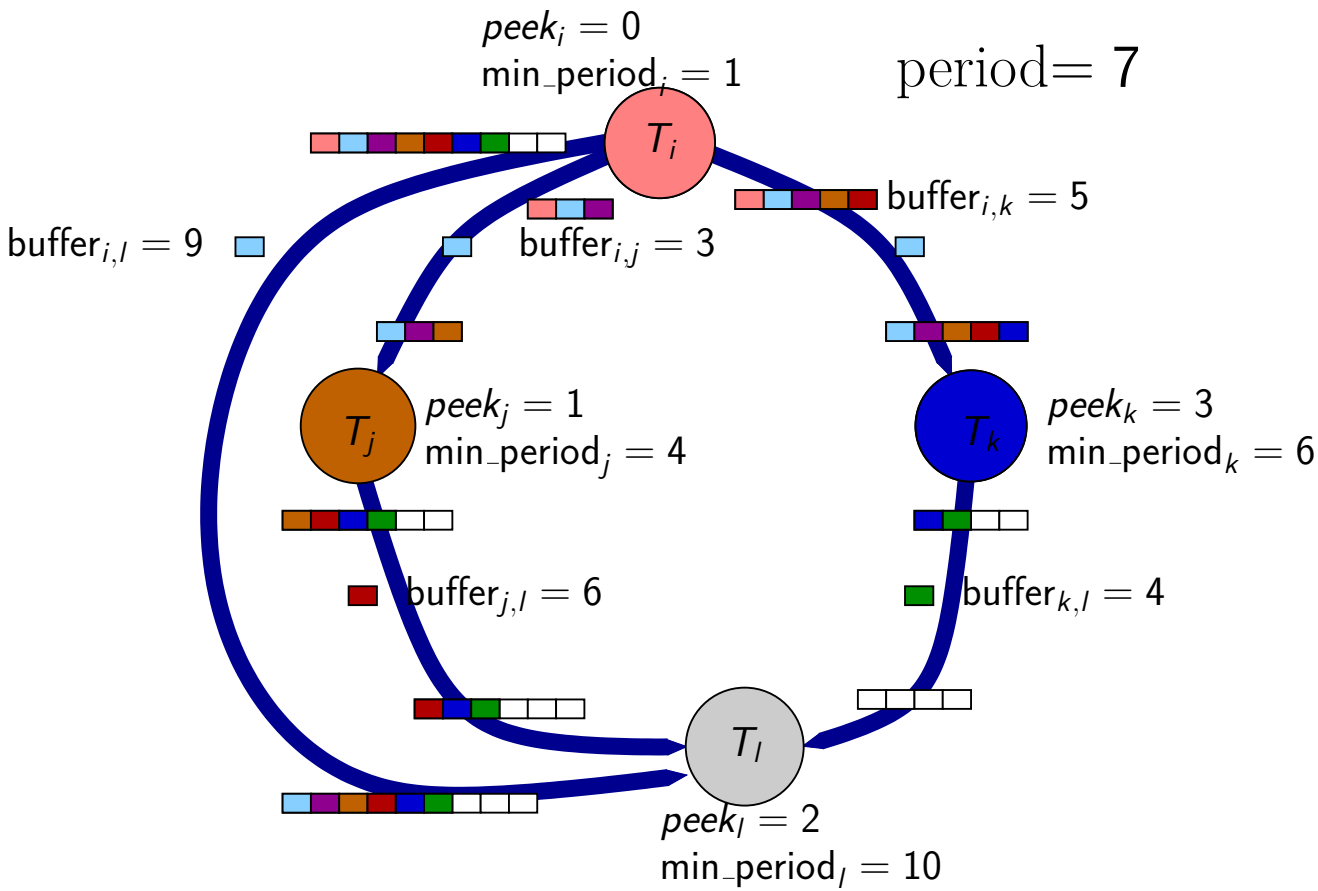
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



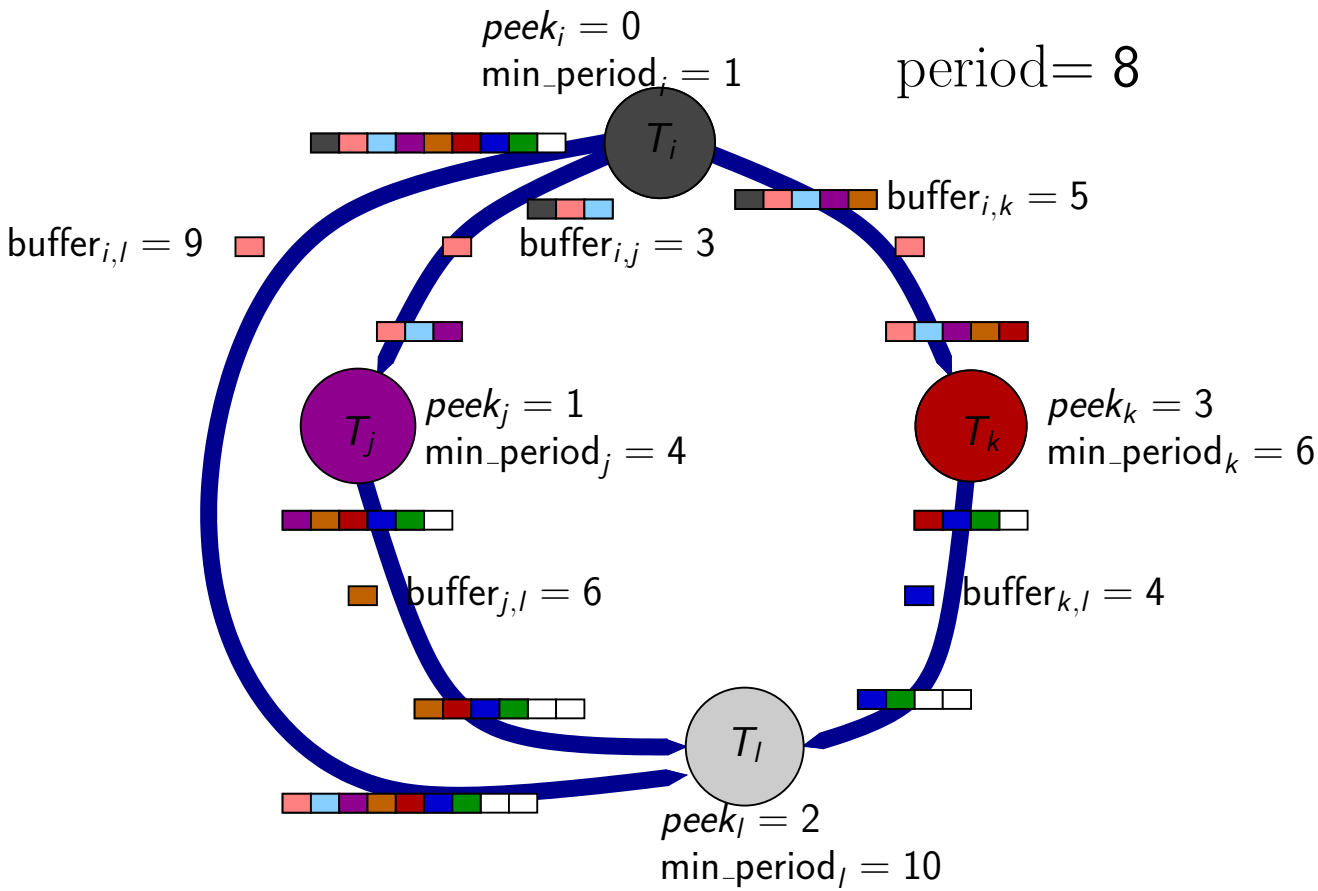
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



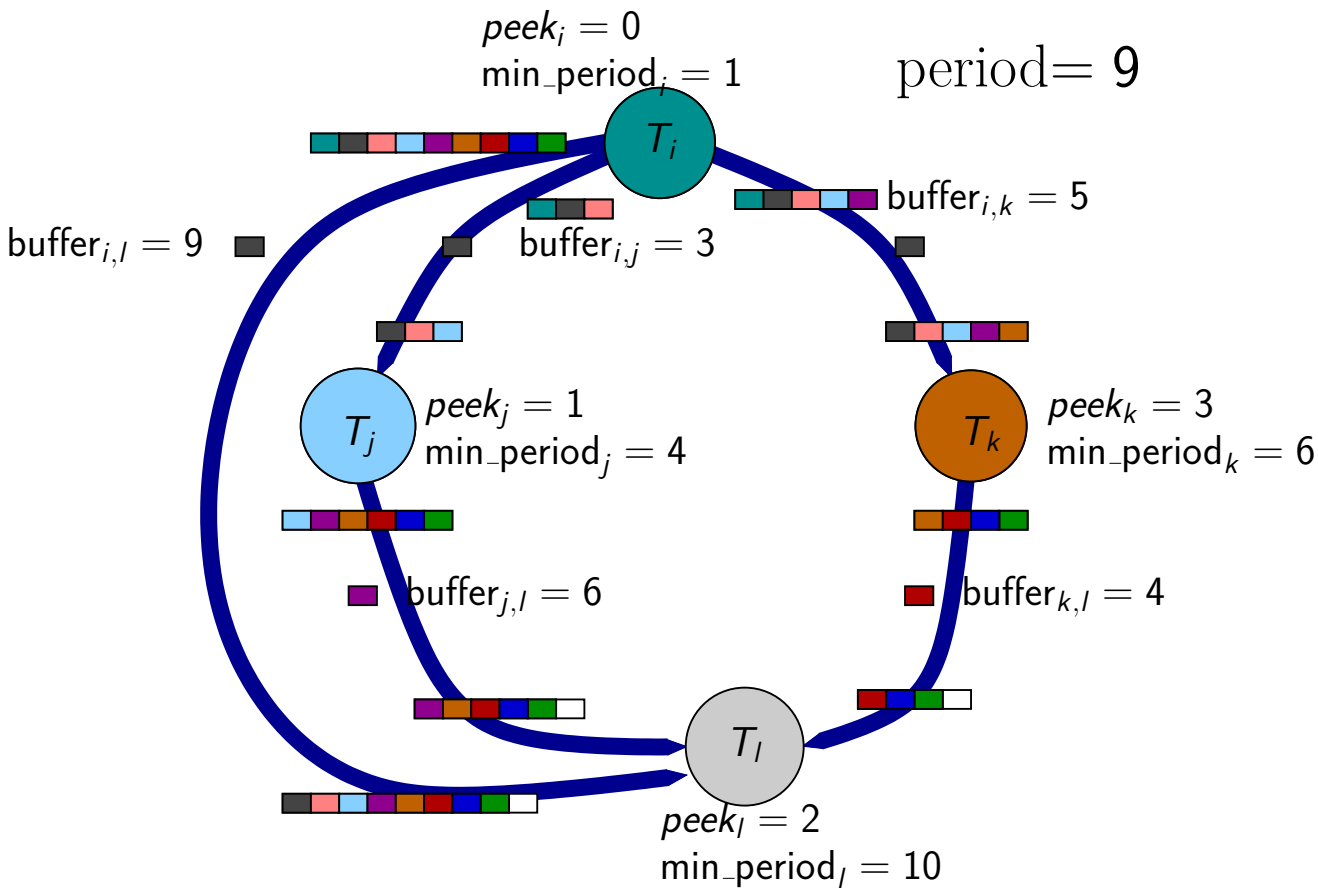
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



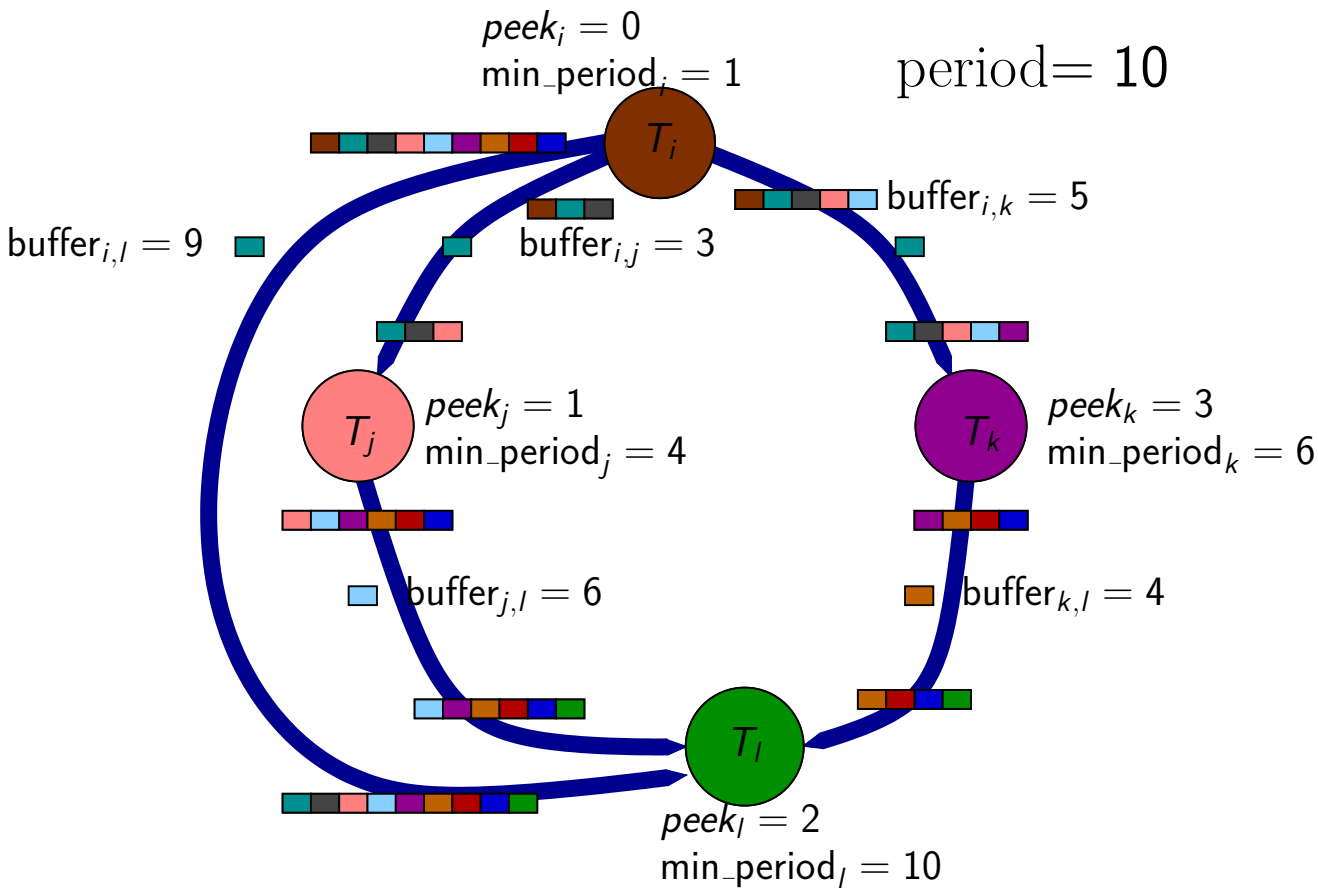
Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



Preprocessing of the schedule

- Objective: compute minimal buffer sizes
- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{buffer}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



Objective

- Maximize throughput
- Obtain a periodic schedule
- Use a single allocation: code size is critical
- Simplification: all communications within a period are simultaneous

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 dependence $T_k \rightarrow T_l$, 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 dependence $T_k \rightarrow T_l$, 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 a given processor, all tasks must be completed within τ:

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

- All incoming communications must be completed within τ:

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

- All outgoing communications must be completed within τ:

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

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

Optimal mapping

- Constraints form a linear program
- Binary variables: exponential solving time ☹
- Can we do better?

Optimal mapping

- Constraints form a linear program
- Binary variables: exponential solving time ☹
- Can we do better?
- NP-complete problem (reduction from 2-Partition) ☹
- Reasonable running times (small number of cores) ☺

Experiments

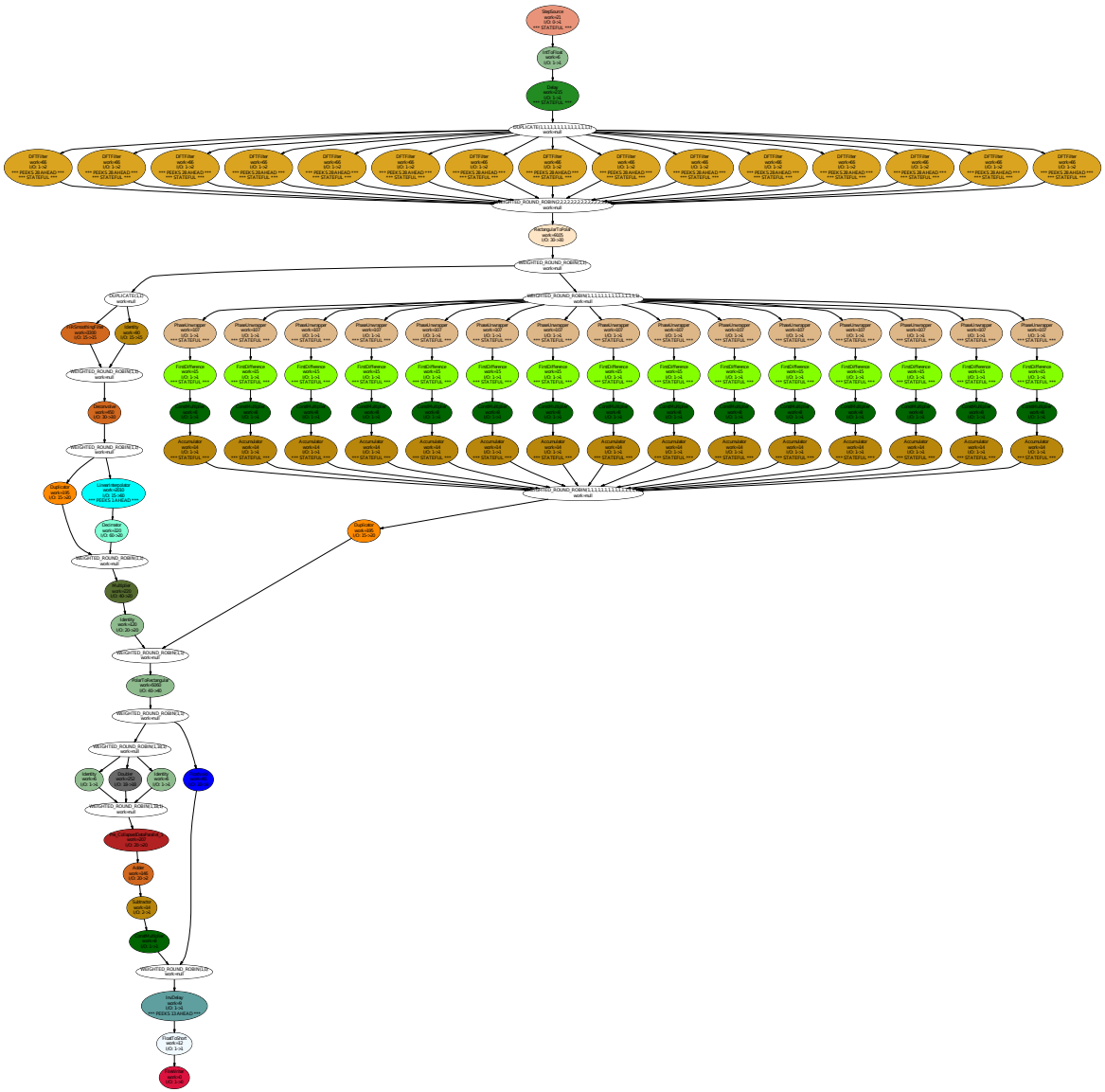
Hardware:

- Sony Playstation 3
- Single Cell processor
- Only 6 available SPEs
- 256-MB memory

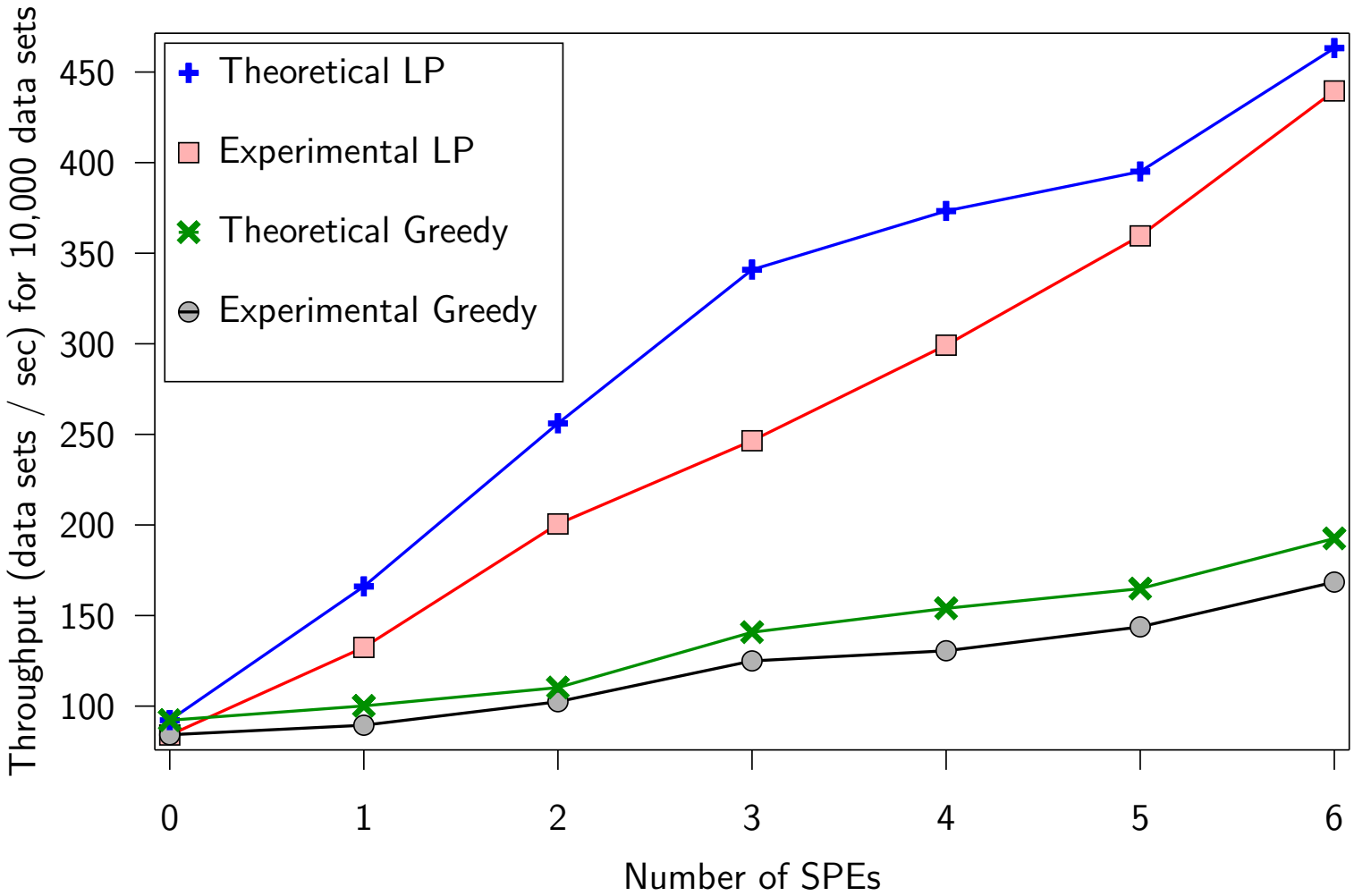
Software:

- New dedicated scheduling framework
- Requires a mono-allocation schedule
- Vocoder application (141 tasks) and random graphs
- Linear programs solved by Cplex (using a 0.05-approximation)
- Greedy memory-aware heuristic as reference

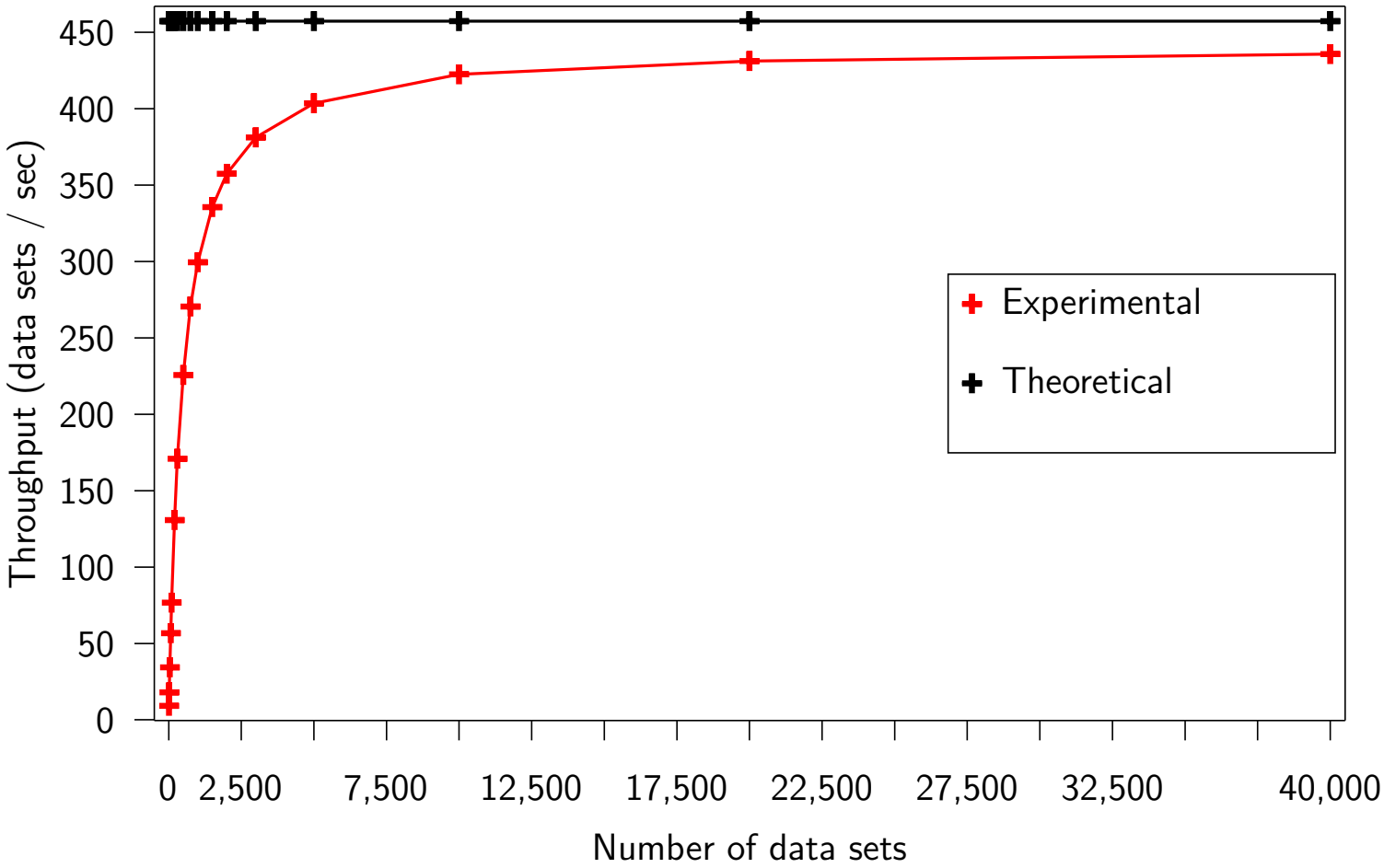
The Vocoder application



Throughput variation according to the number of SPEs



Time to reach steady-state



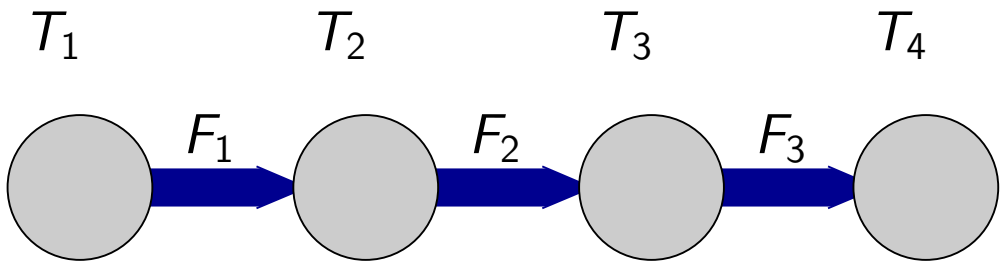
Summary

- Heterogeneity is difficult to handle
- Innovative processor, but with strong hardware constraints
- Optimal solution to steady-state mono-allocation scheduling problem
- New framework dedicated to mono-allocation schedules
- Outperforms greedy memory-aware heuristic

Outline

Application and platform

- A linear workflow with many data sets



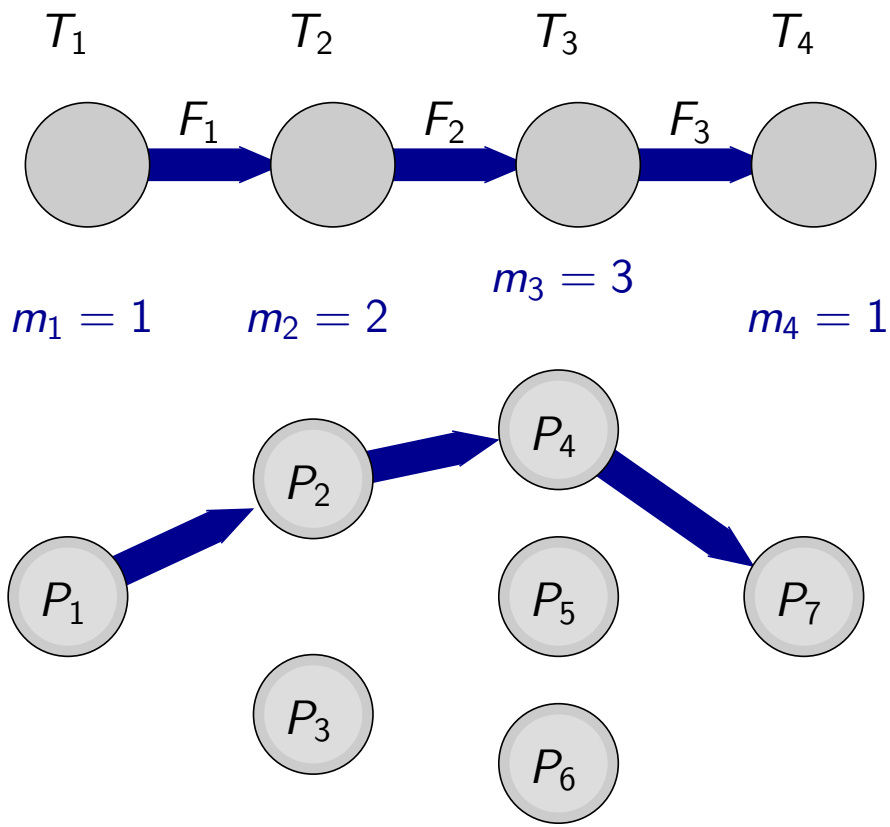
- Fully connected platform
- Heterogeneous processors and communication links
- Mapping is given
- Objective: determine throughput

Communication models

- **Strict One-Port**
- **Overlap One-Port**

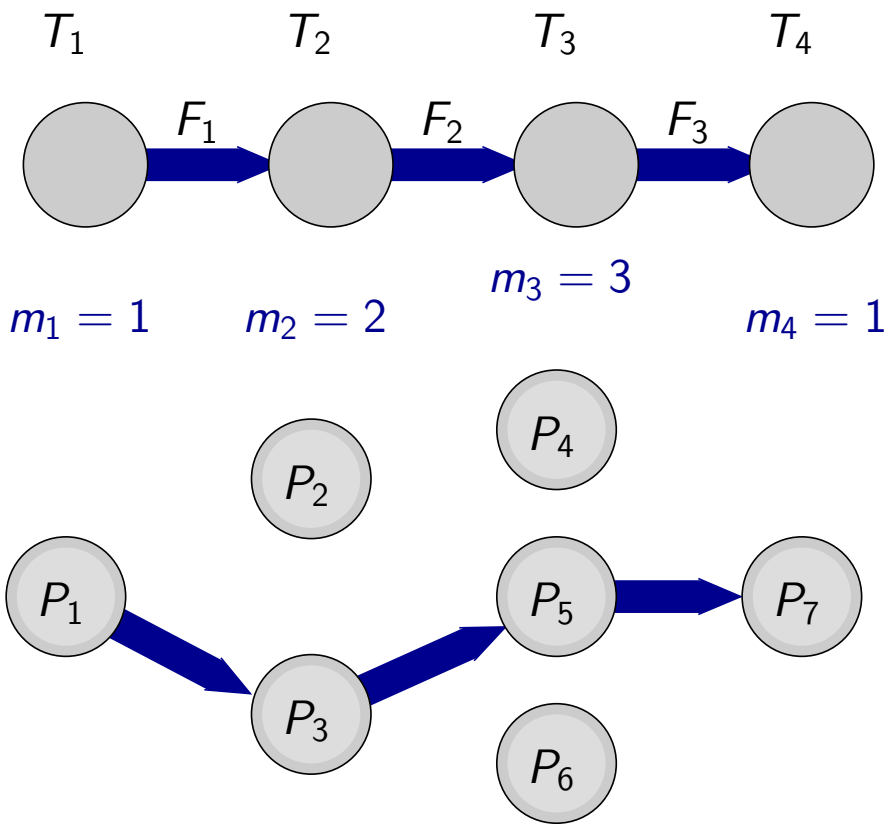
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i



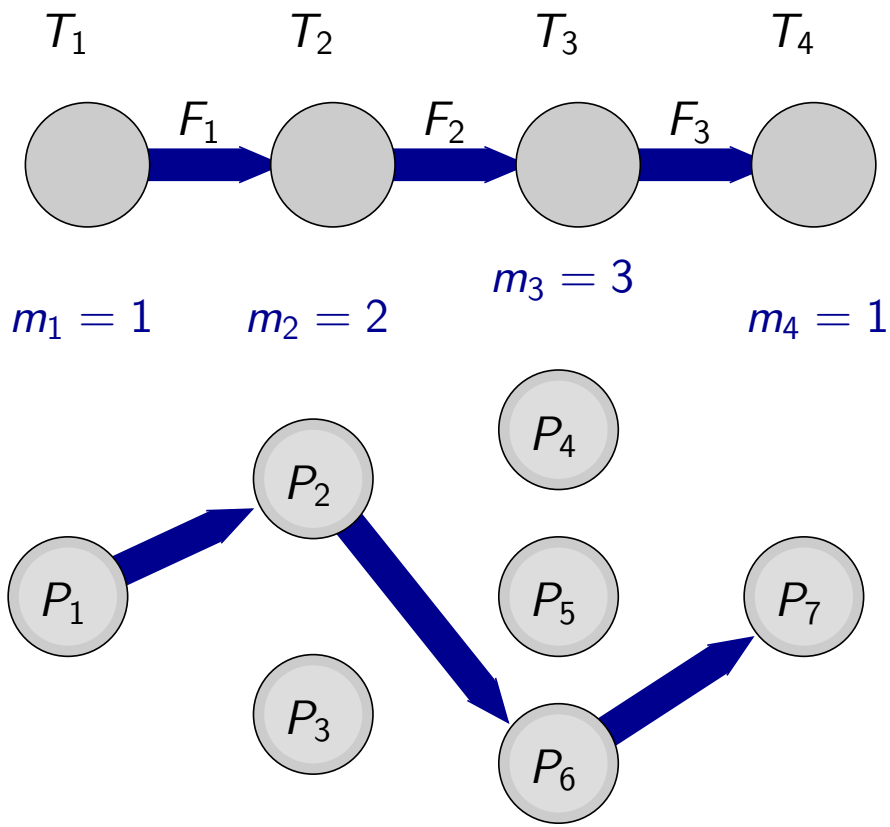
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



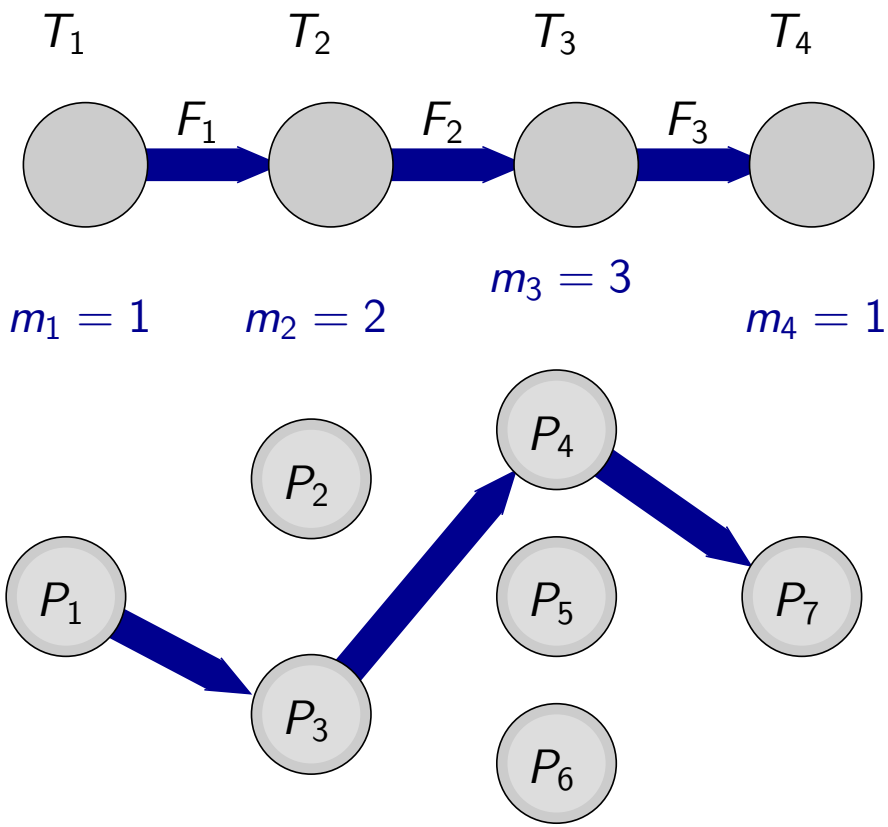
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



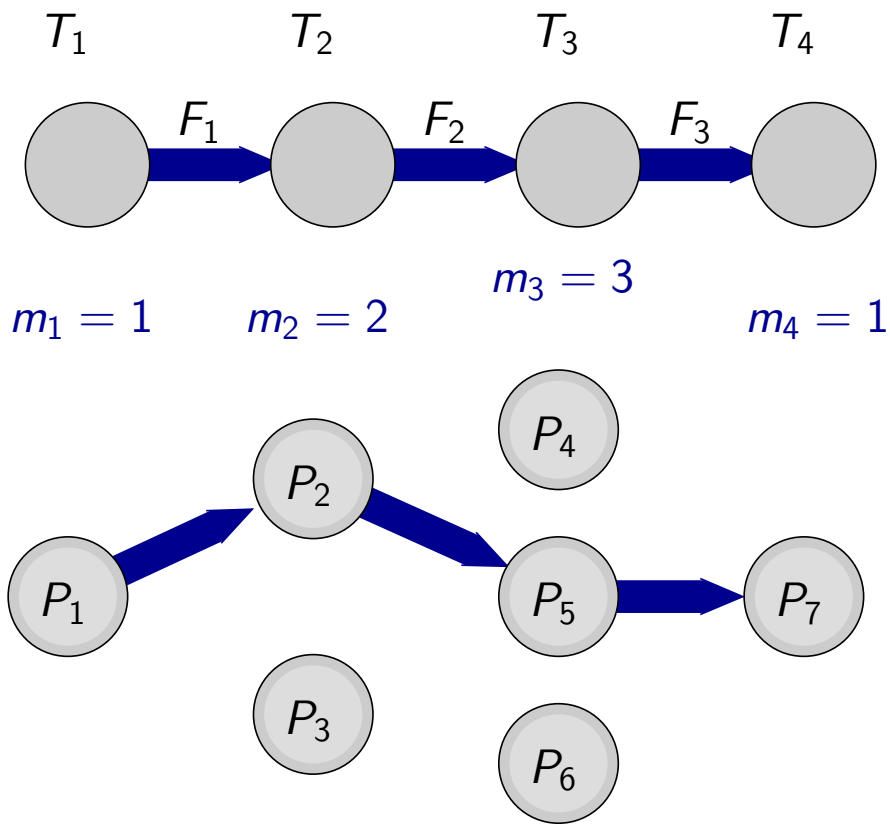
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



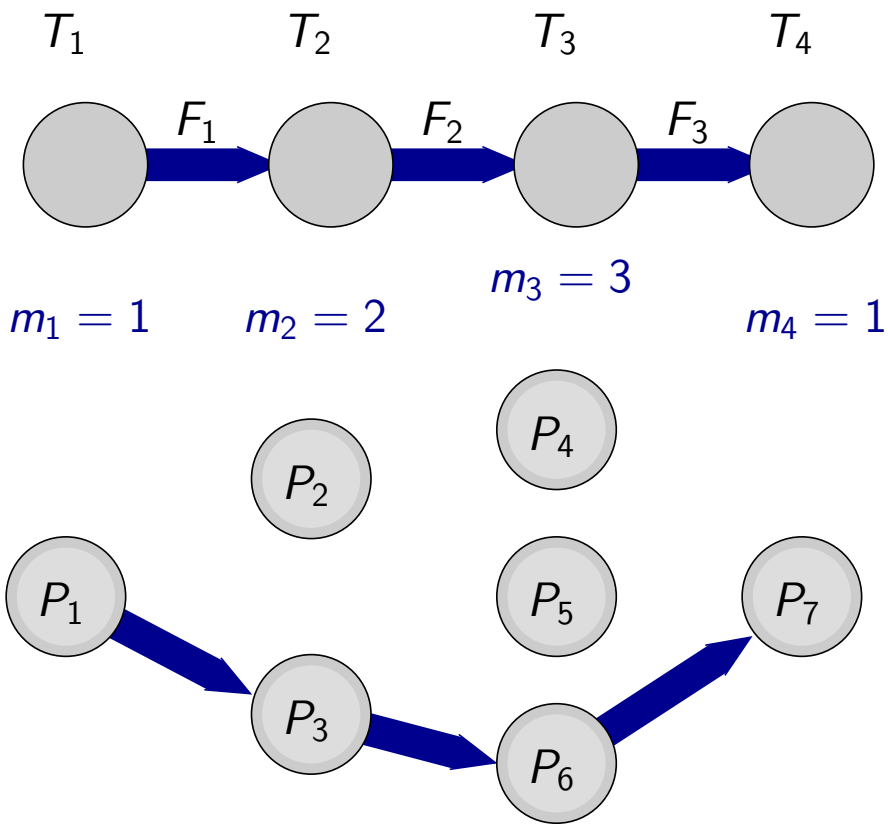
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



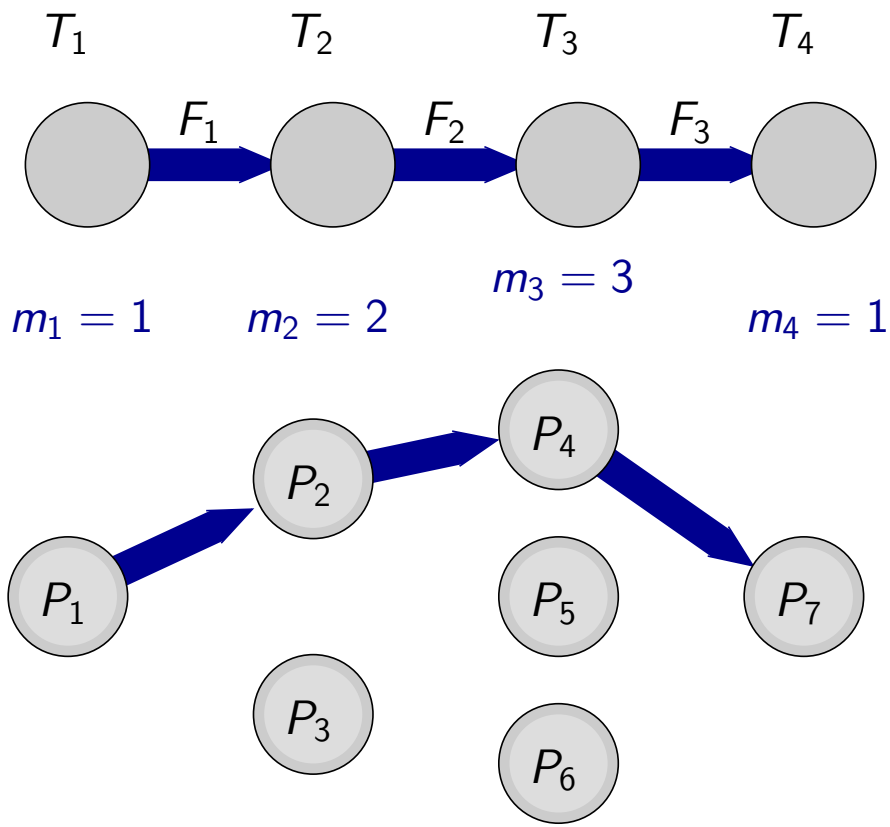
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task



Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task

Input data	Path in the system
1	$P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_7$
2	$P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_7$
3	$P_1 \rightarrow P_2 \rightarrow P_6 \rightarrow P_7$
4	$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_7$
5	$P_1 \rightarrow P_2 \rightarrow P_5 \rightarrow P_7$
6	$P_1 \rightarrow P_3 \rightarrow P_6 \rightarrow P_7$
7	$P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_7$
8	$P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_7$

Mapping

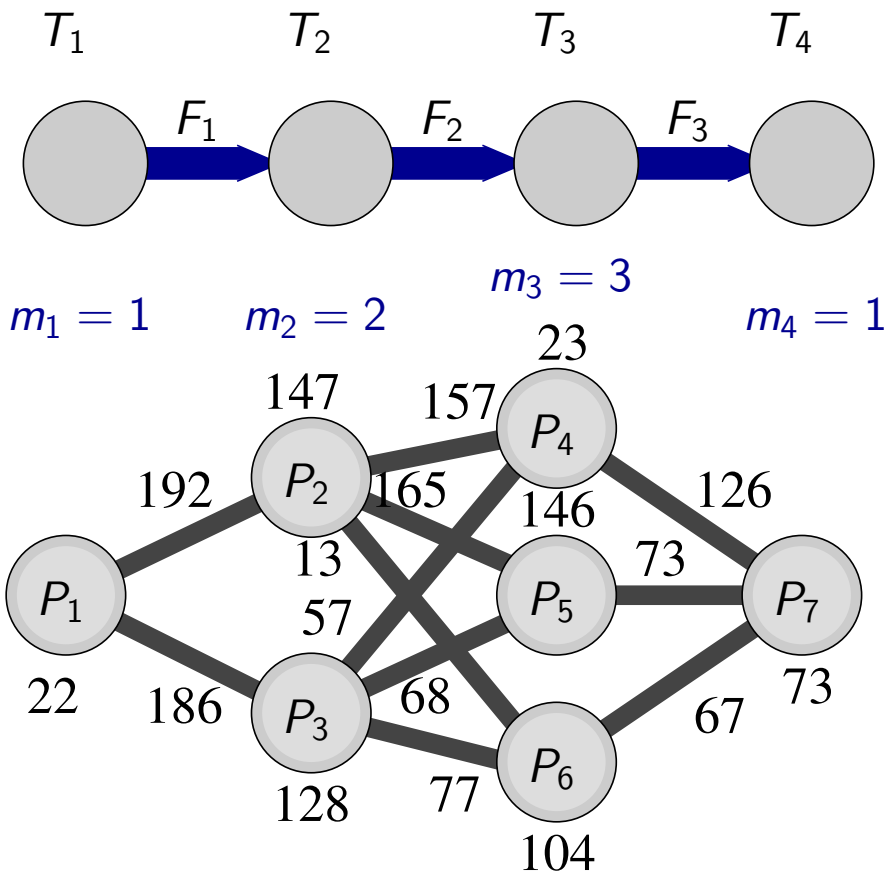
- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Round-Robin distribution of each task

Theorem

Assume that stage T_i is mapped onto m_i distinct processors. Then the number of paths is equal to $m = \operatorname{lcm}(m_1, \ldots, m_n)$.

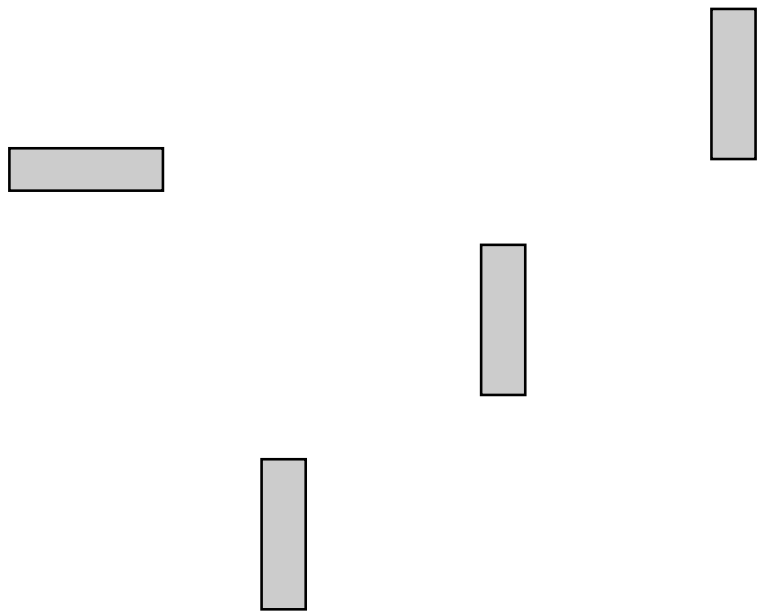
Mapping

- A processor processes at most 1 task
- A task is mapped on possibly many processors
- Replication count of T_i: m_i
- Critical cycle time = 215.8 Period = 230.7



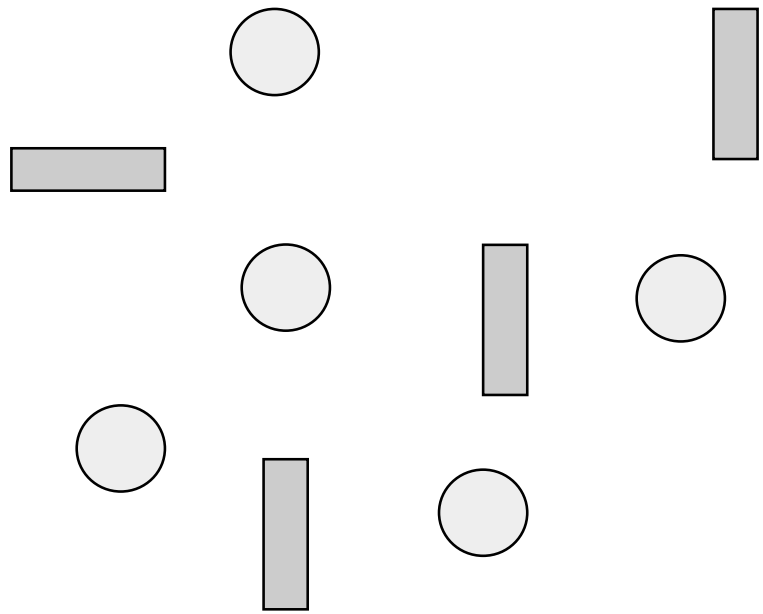
Short presentation of Timed Petri Nets (TPN)

- Some transitions



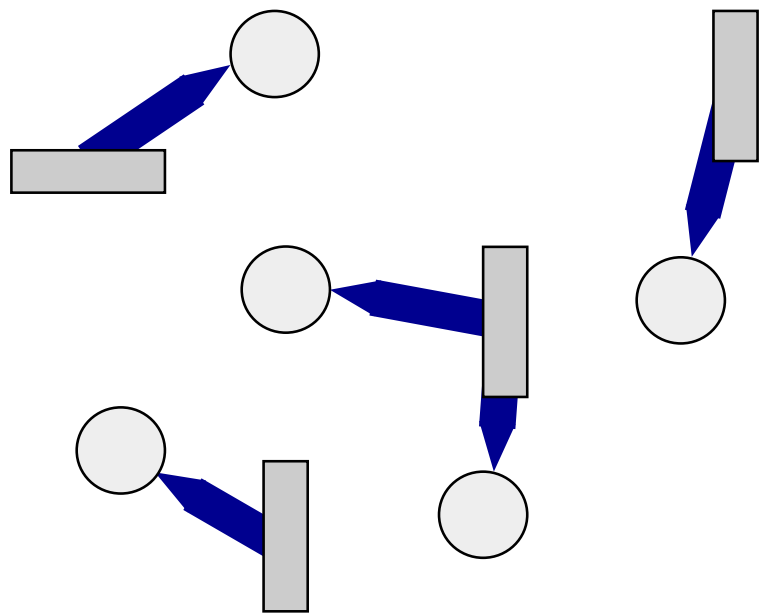
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places



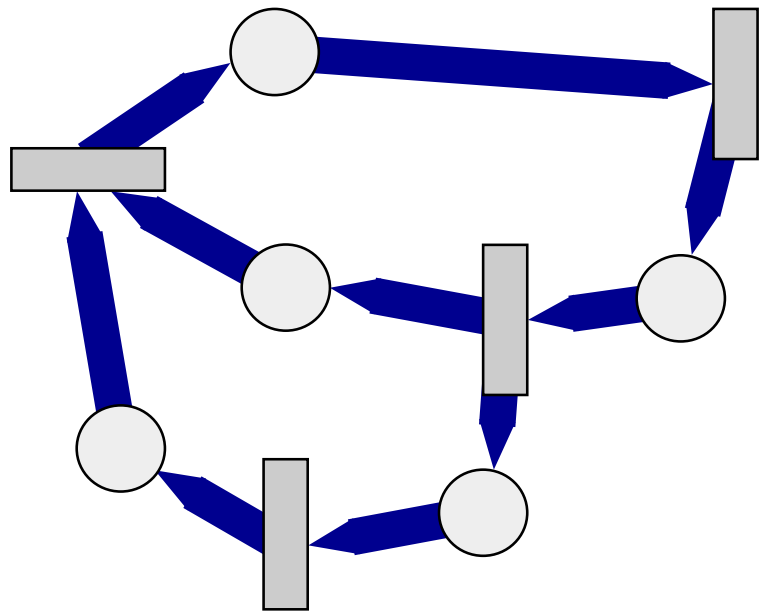
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places...



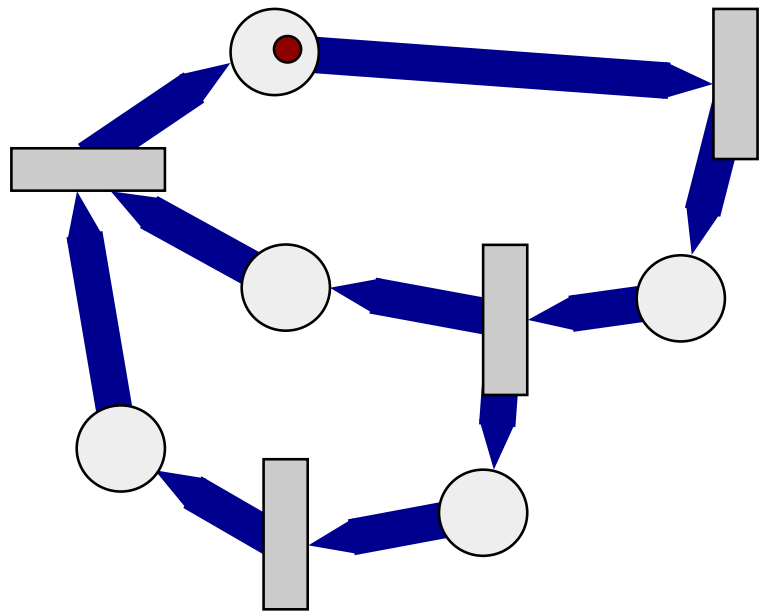
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions



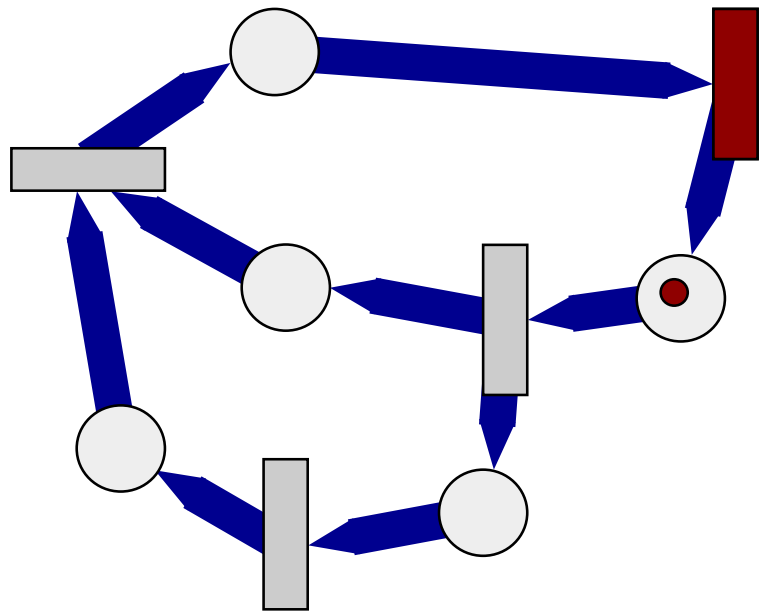
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired



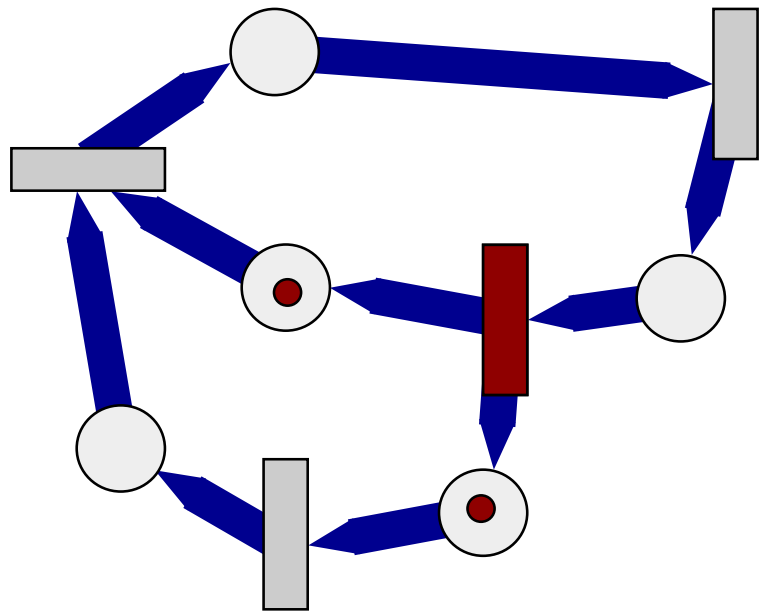
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired



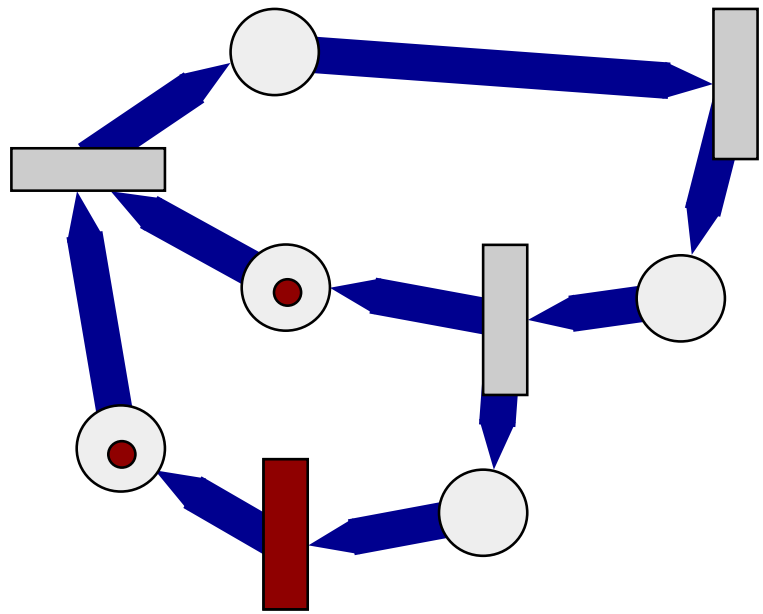
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired



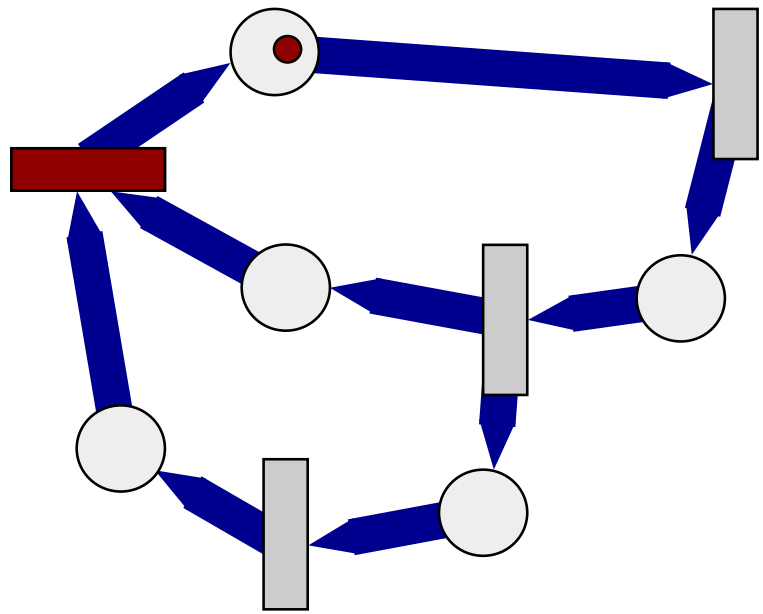
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired



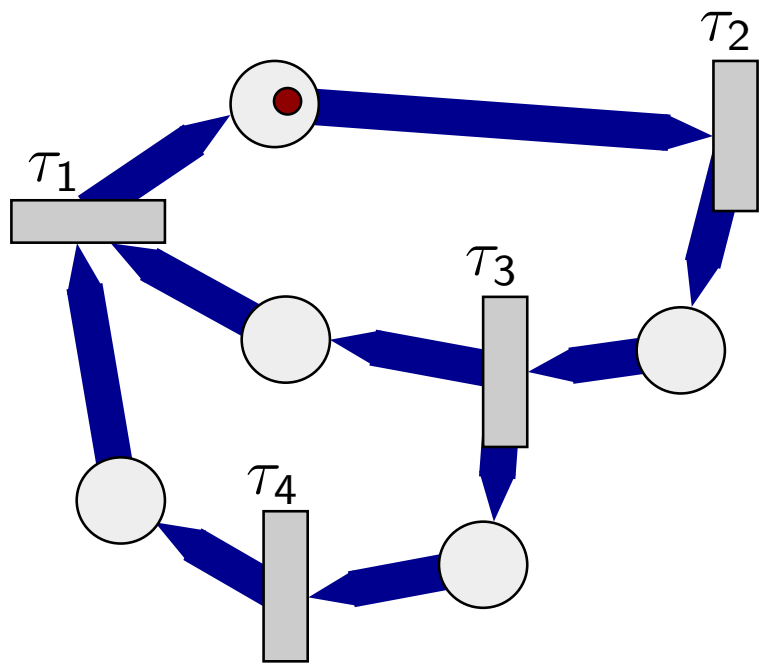
Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired



Short presentation of Timed Petri Nets (TPN)

- Some transitions
- Some places
- Connections between transitions and places... and between places and transitions
- Some tokens allowing transitions to be fired
- Delay between the consumption of input tokens and the creation of output tokens

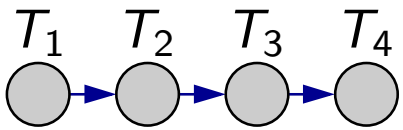


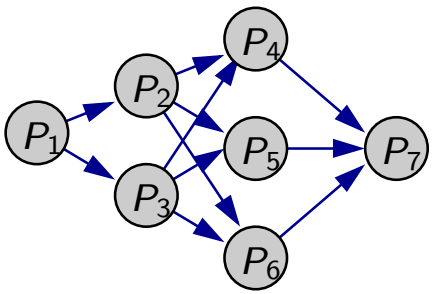
Timed Petri Net model

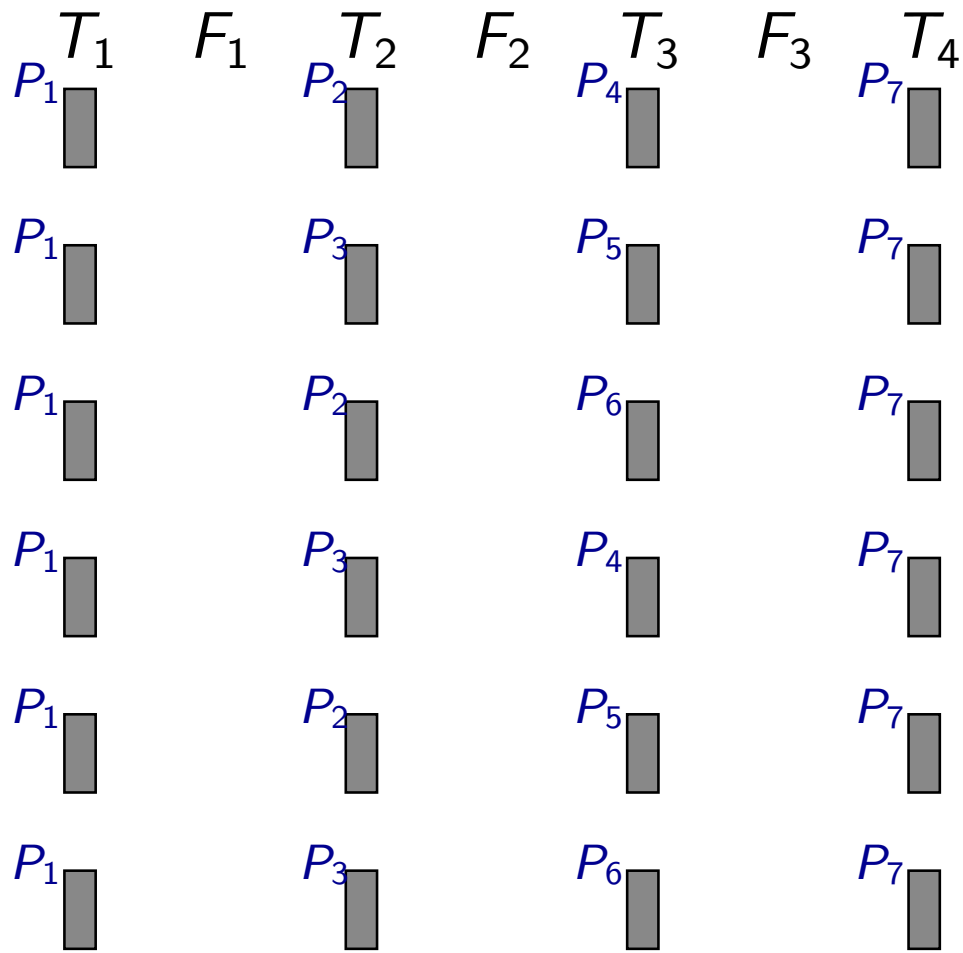
- Transitions: communications and computations
- Places: dependences between two successive operations
- Each path followed by the input data must be fully developed in the TPN

Overlap One-Port model

Computations

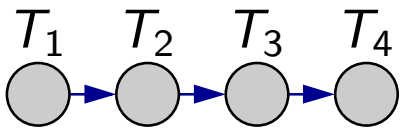


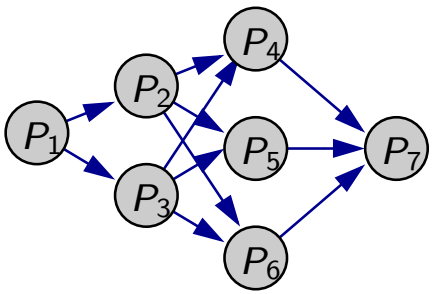


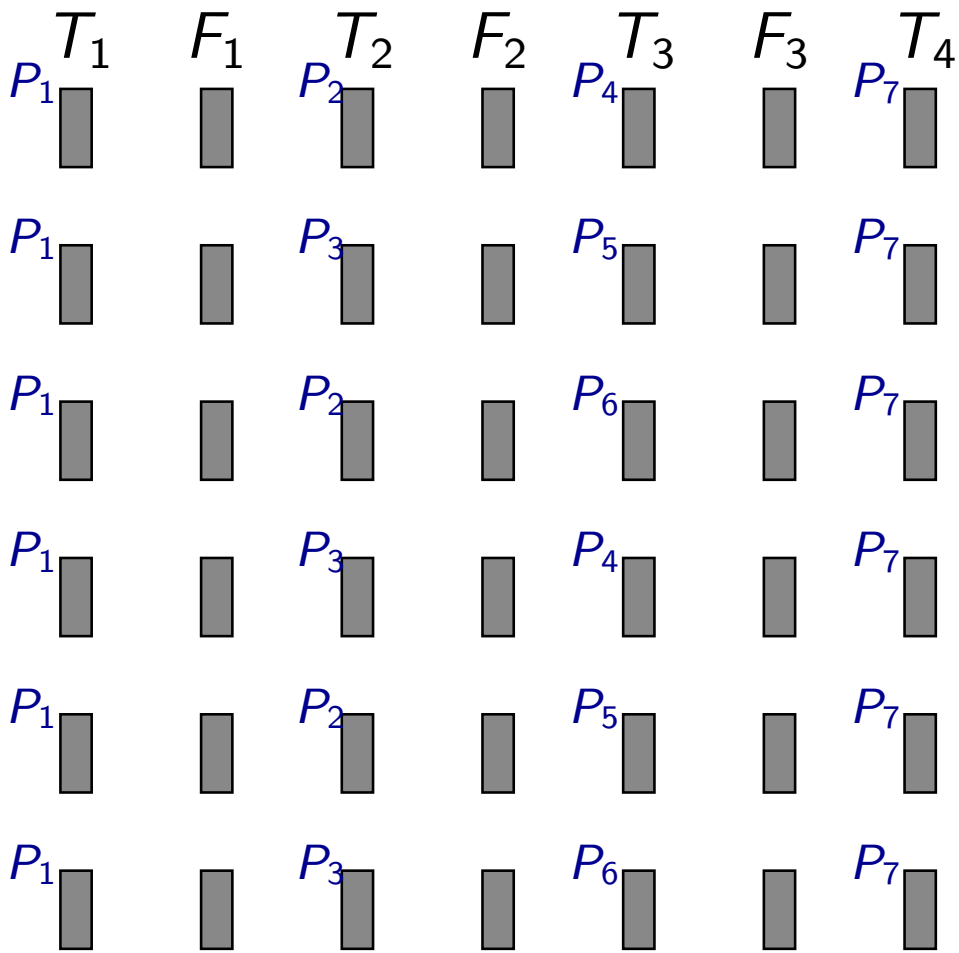


Overlap One-Port model

Computations and commmunications

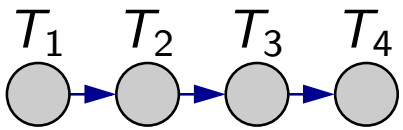


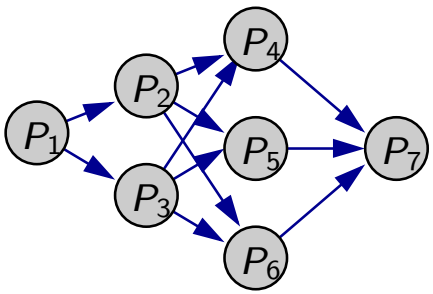


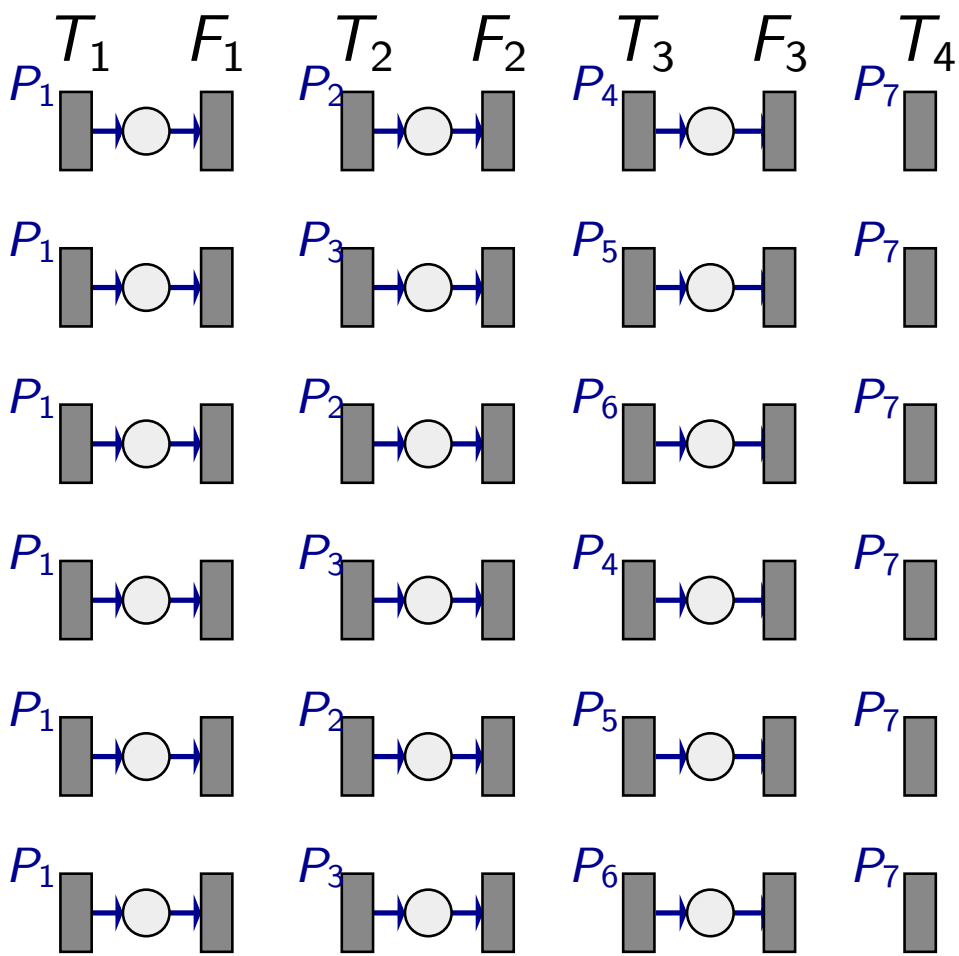


Overlap One-Port model

A communication cannot begin before the end of the computation

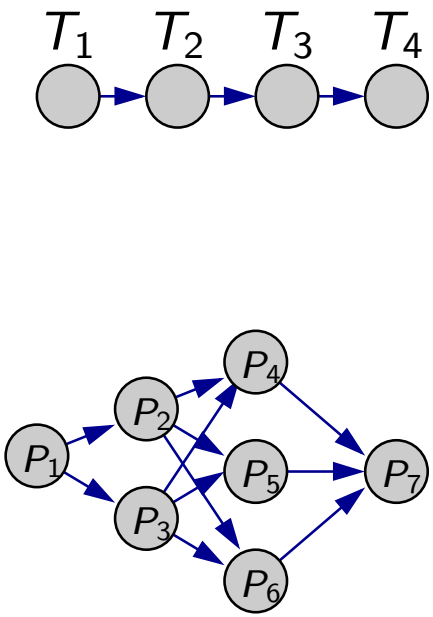


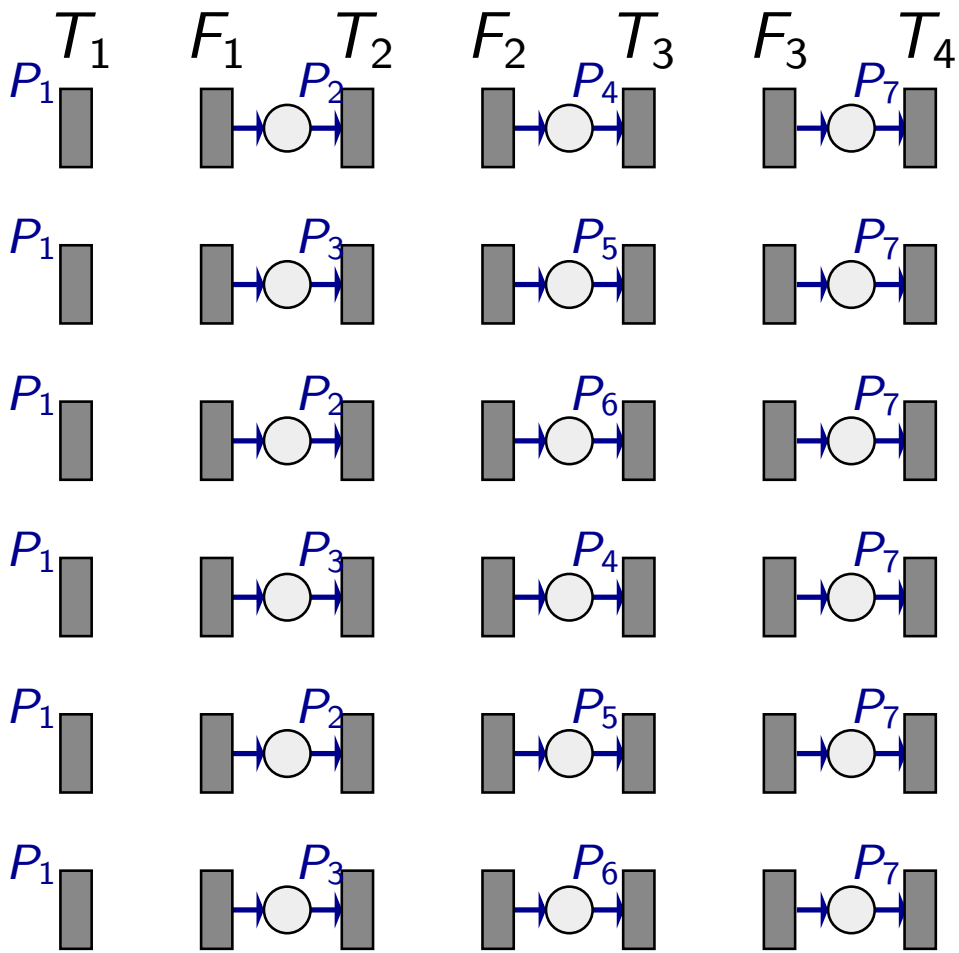




Overlap One-Port model

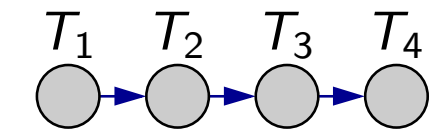
A computation cannot begin before the end of the communication

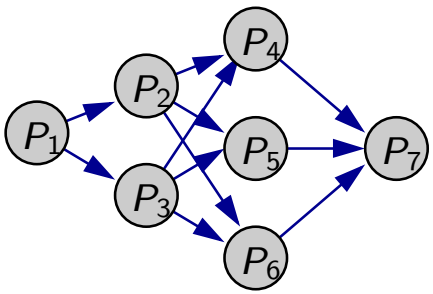


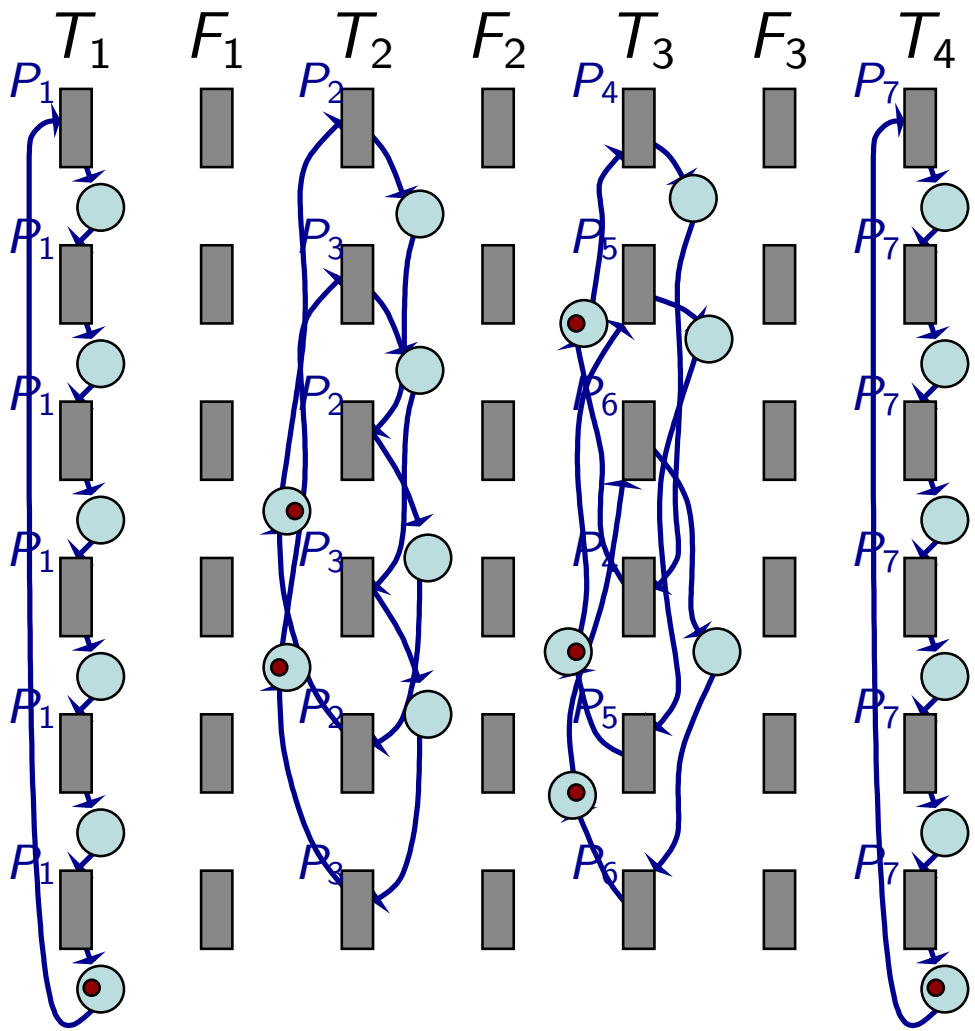


Overlap One-Port model

Dependences due to the round-robin distribution of computations

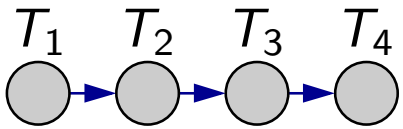


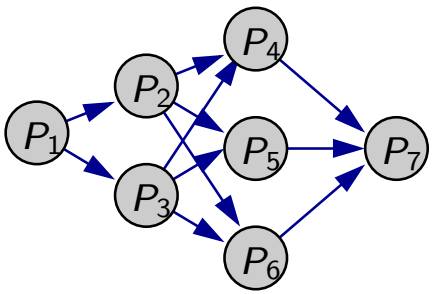


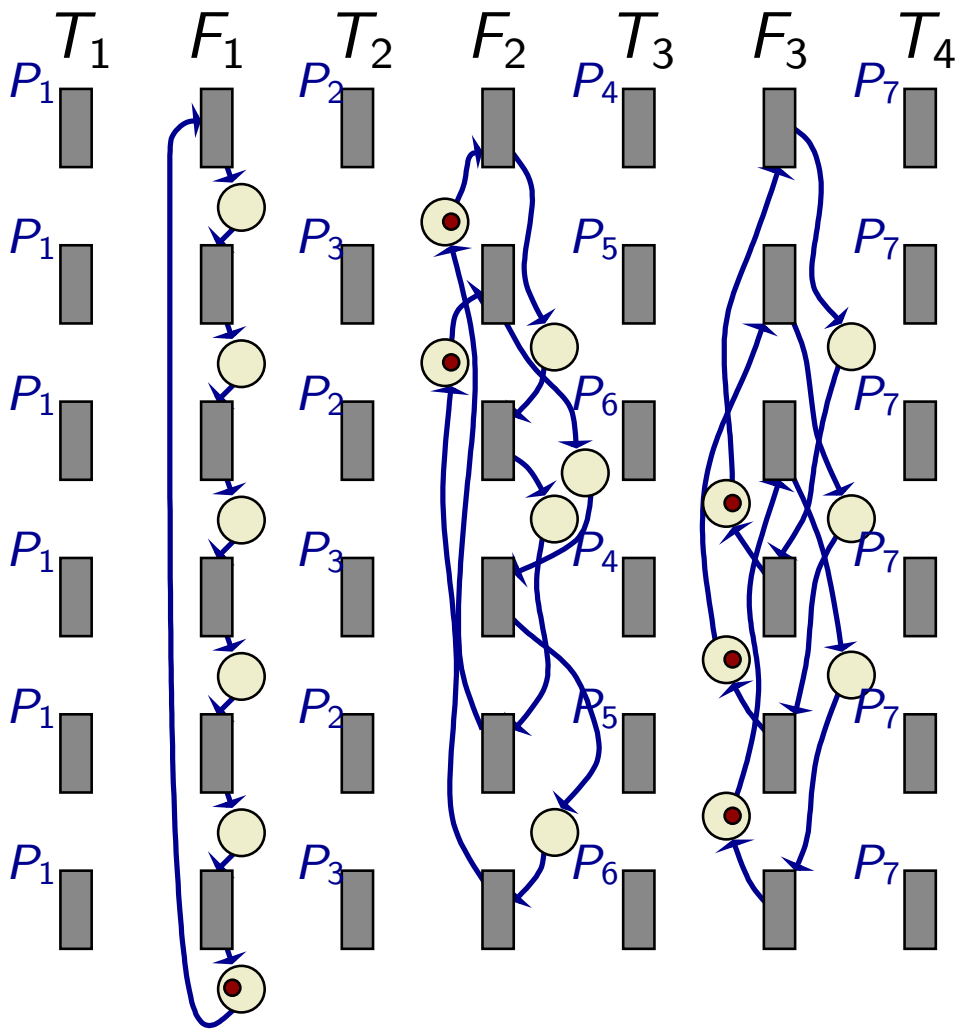


Overlap One-Port model

Dependences due to the round-robin distribution of outgoing communications

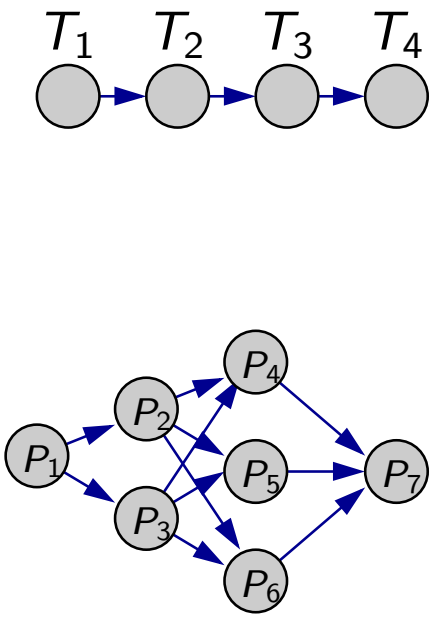


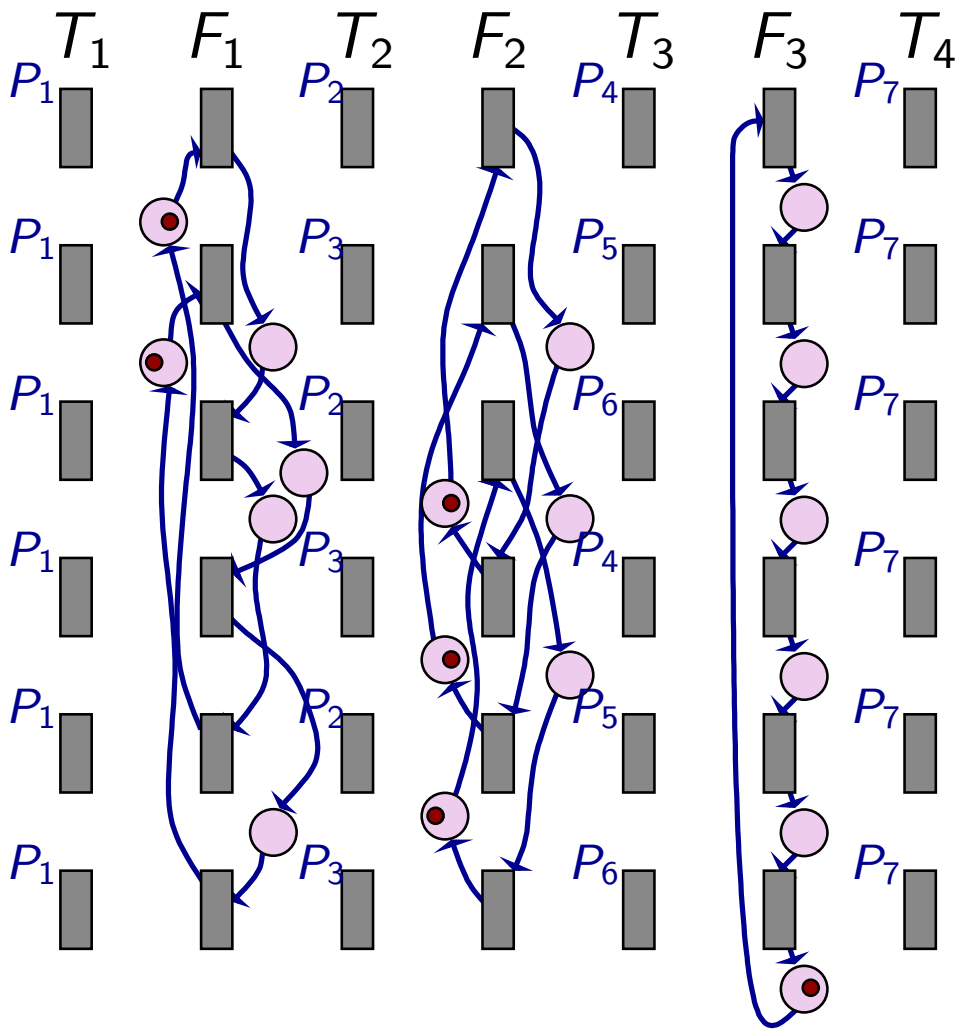




Overlap One-Port model

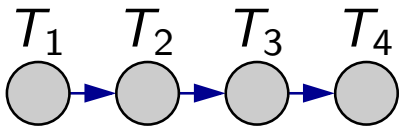
Dependences due to the round-robin distribution of incoming communications

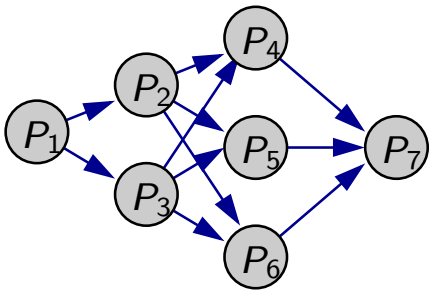


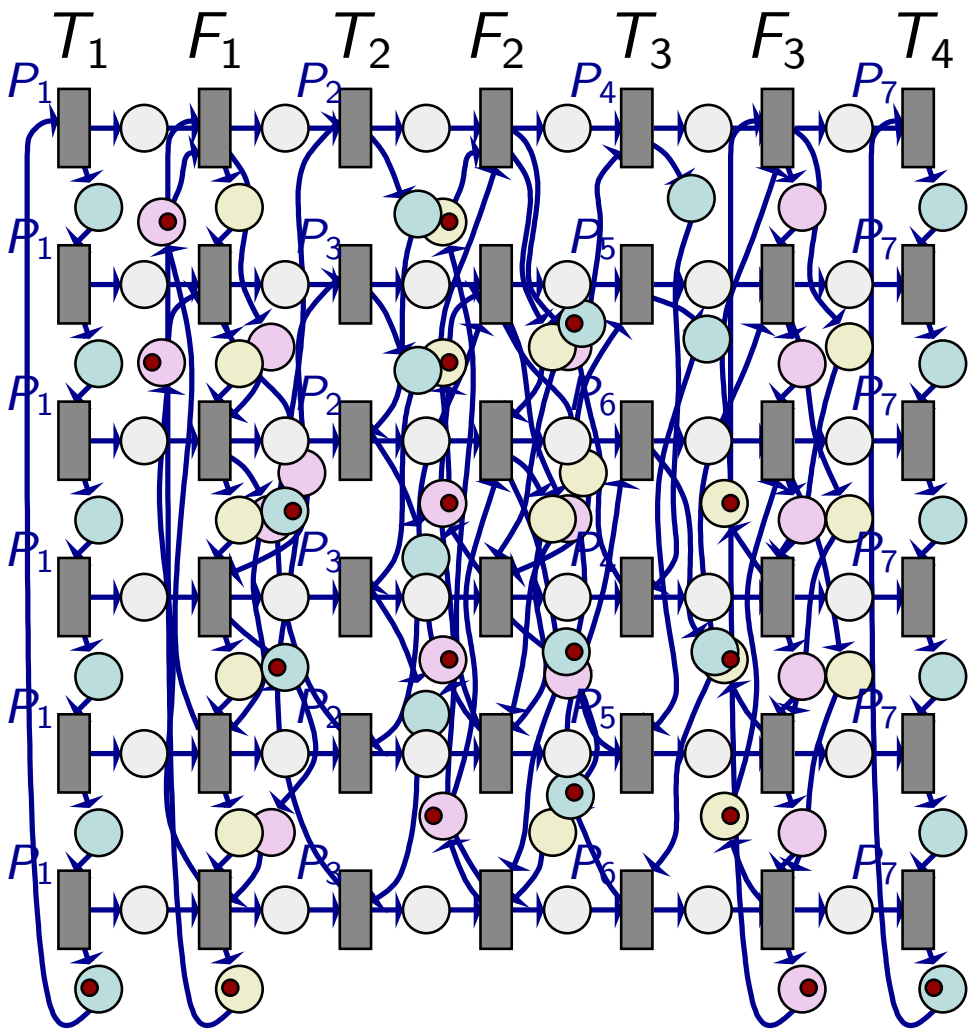


Overlap One-Port model

All dependences!

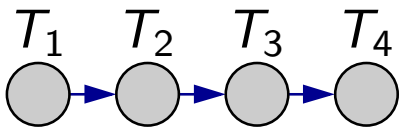


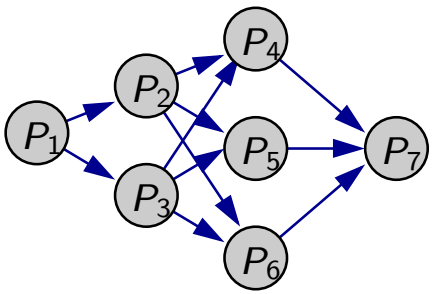


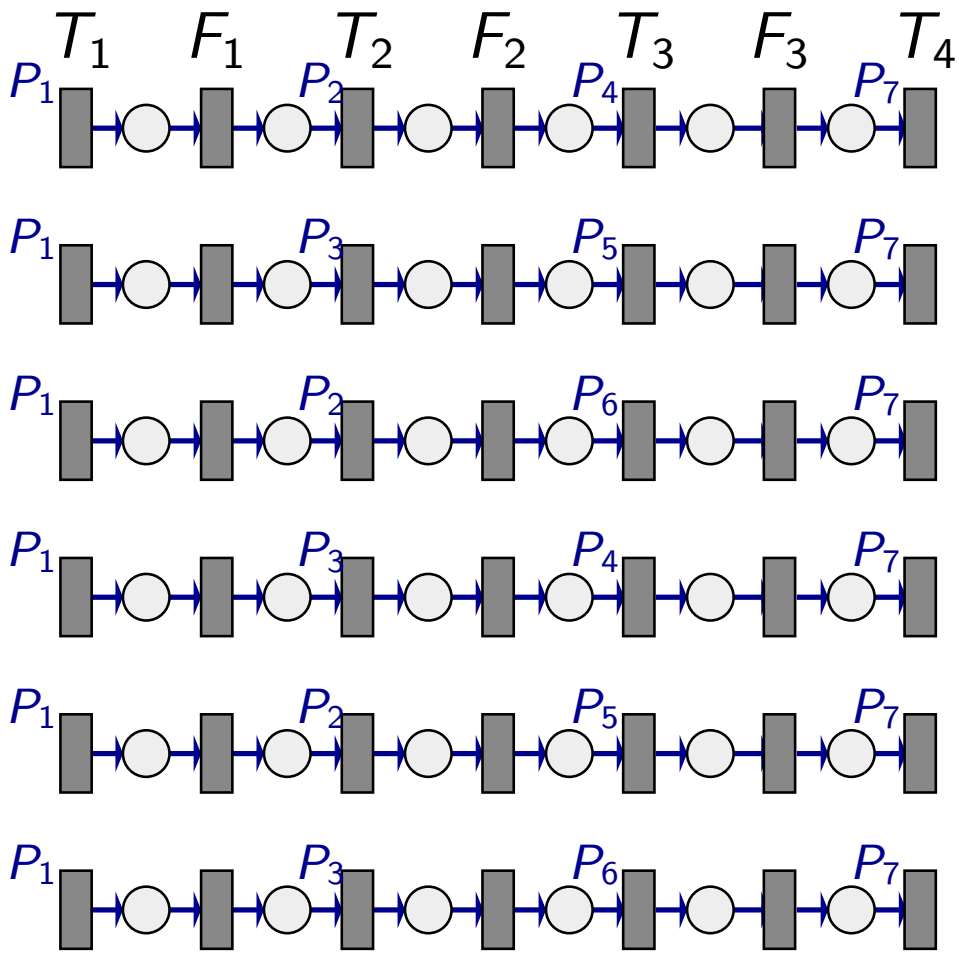


Strict One-Port model

Dependences between communications and computations

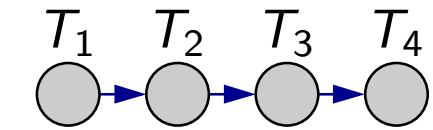


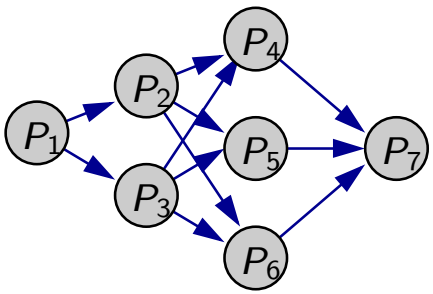


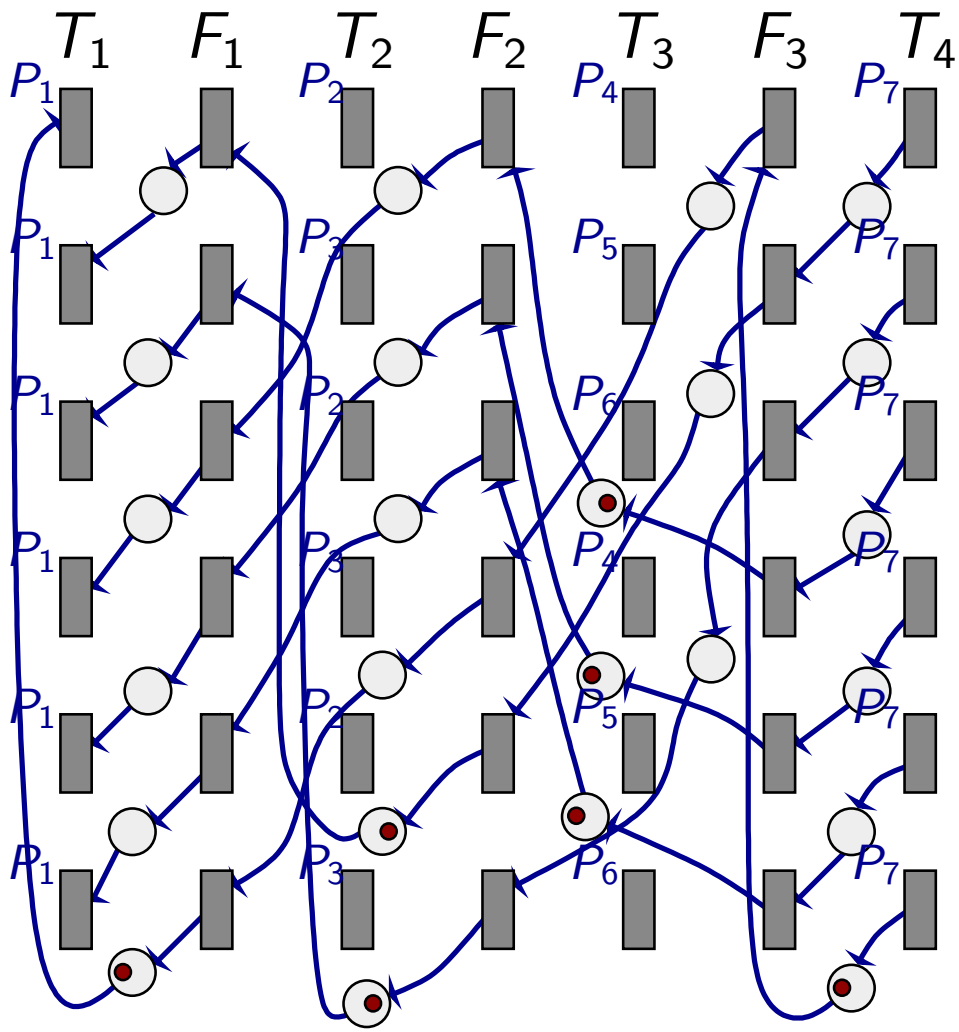


Strict One-Port model

Dependences due to the **Strict One-Port** model

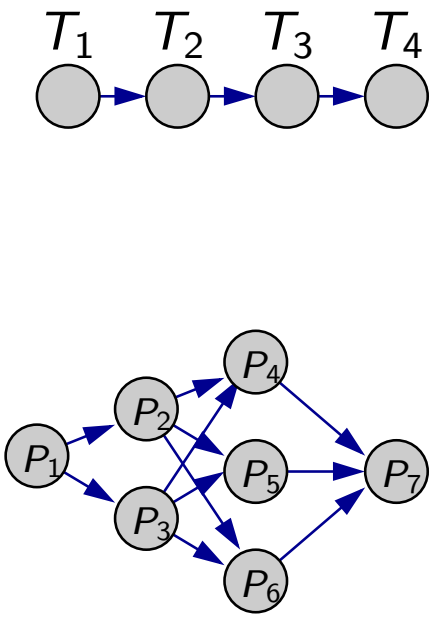


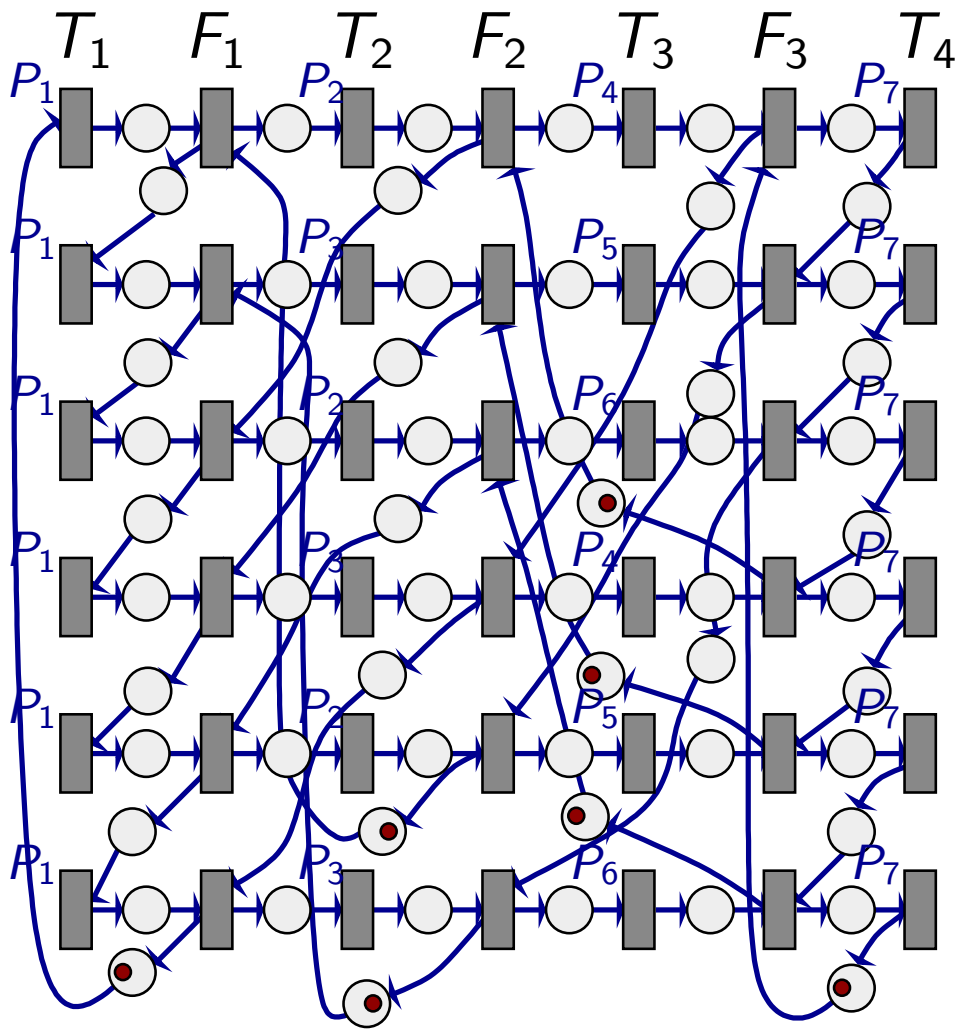




Strict One-Port model

All dependences!





Computing the throughput

- Equivalent to find critical cycles
- $\mathcal{C}$ is a cycle of the TPN
- $\mathcal{L}(\mathcal{C})$ is its length (total time of transitions)
- $t(\mathcal{C})$ is the total number of tokens in places traversed by $\mathcal{C}$
- A critical cycle achieves the largest ratio $\max_{\mathcal{C}\text{cycle}} \frac{\mathcal{L}(\mathcal{C})}{t(\mathcal{C})}$
- This ratio gives the period τ of the system
- Can be computed in time $\mathcal{O}(n^3 m^3)$ $(m = \operatorname{lcm}(m_1, \ldots, m_n))$

Computing the throughput

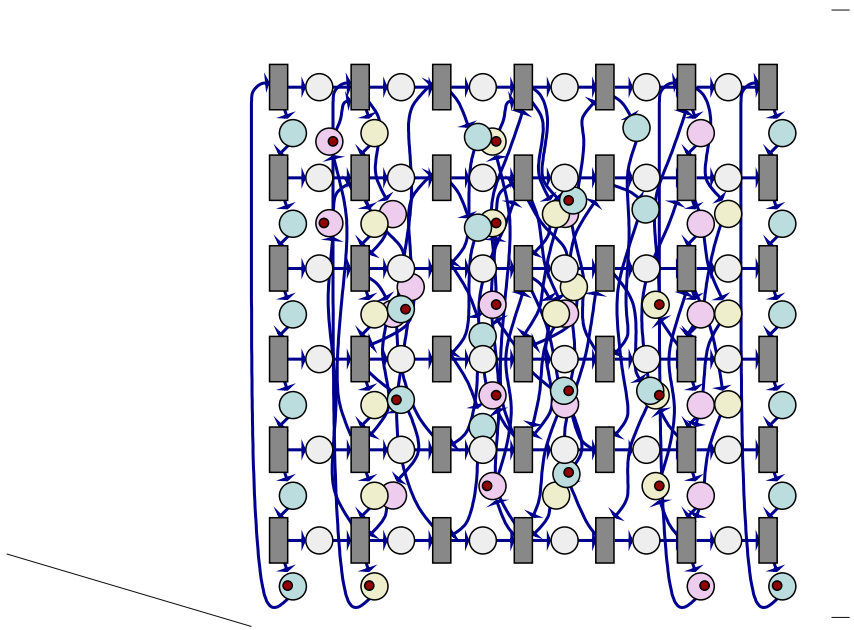
- TPN has an exponential size ☹
- **Overlap One-Port** model:

Theorem

Consider a pipeline of n tasks $T_1, \ldots, T_{n-1}$, such that stage T_i is mapped onto m_i distinct processors. Then the average throughput of this system can be computed in time $\mathcal{O}\left(\sum_{i=1}^{n-1} (m_i m_{i+1})^3\right)$.

- **Strict One-Port** model: problem remains open

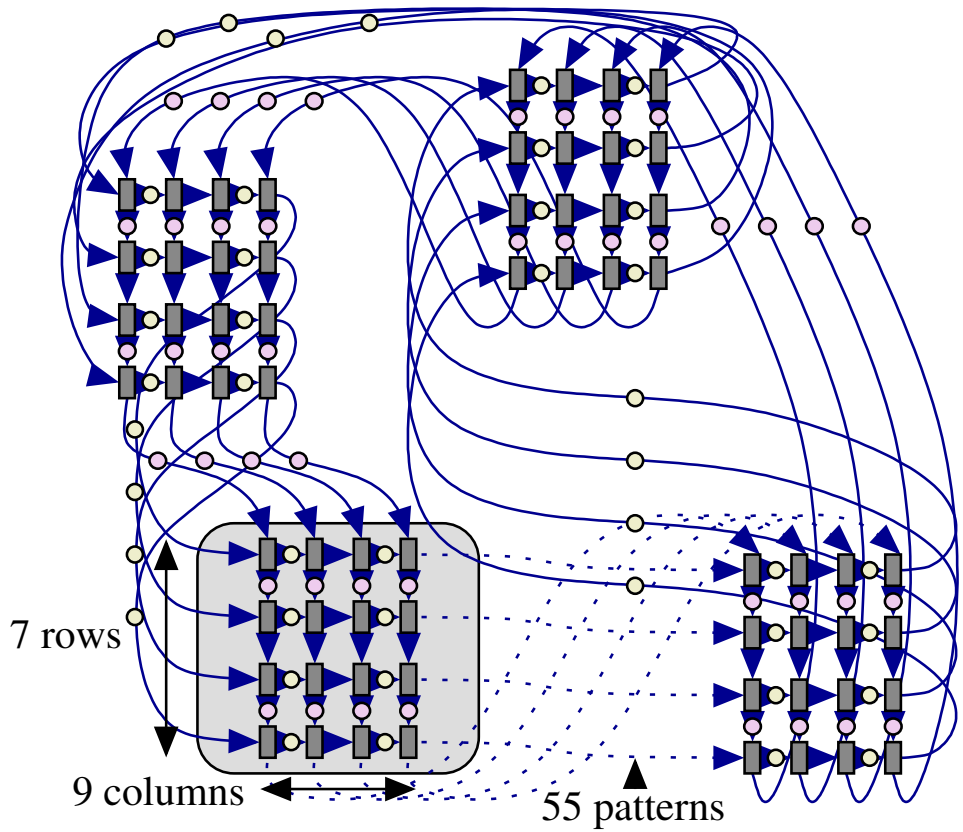
Key ideas of proof



- Split the TPN into $2n - 1$ columns
- Computation columns: simple problem
- Communication columns: reduction to smaller TPNs with critical cycles of same weight

The case of a communication column

- Several connected components.
- Example of connected component:



$$m_1 = 5, m_2 = 21, m_3 = 27, m_4 = 11$$

Summary

- Even when mapping is given, the throughput is hard to determine
- Examples without critical resource for both communication models
- Such examples remain seldom

Outline

Conclusion

Presentation of three steady-state scheduling problems

- Mono-allocation task graph scheduling on large heterogeneous platforms
- Task graph scheduling on the Cell processor
- Computing throughput of replicated workflows

Algorithms and methods:

- NP-completeness proofs
- Optimal algorithms, mainly using linear programs
- Heuristics
- Use of Timed Petri Nets to model a complete system

Conclusion

Presentation of three steady-state scheduling problems

- Mono-allocation task graph scheduling on large heterogeneous platforms
- Task graph scheduling on the Cell processor
- Computing throughput of replicated workflows

Experimental study:

- Simple numerical simulations
- Simulation using the SimGrid framework
- Cell implementation

Ongoing Work and Perspectives

Mono-allocation steady-state scheduling

- Introduce duplication to increase the throughput

Scheduling task graphs on Cell processors

- Write new models to capture multi-Cell platforms
- Clever heuristics to handle very large task graphs
- Need to correctly model new heterogeneous architectures

Computing throughput of replicated workflows

- Determine complexity of the *Strict One-Port* case
- Work with stochastic computation and communication times
- Efficient heuristics to find good mappings

Publications

Journal and book chapter

Comments on "Design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks"
Matthieu Gallet, Yves Robert, Frédéric Vivien, *Journal of Parallel and Distributed Computing, 68(2)*, 2008

Divisible Load Scheduling
Matthieu Gallet, Yves Robert, Frédéric Vivien, *Introduction to Scheduling* , 2009

International conferences

Scheduling communication requests traversing a switch: complexity and algorithms
Matthieu Gallet, Yves Robert, Frédéric Vivien, *Proceedings of the 15th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP'2007)* , 2007

Scheduling multiple divisible loads on a linear processor network
Matthieu Gallet, Yves Robert, Frédéric Vivien, *Proceedings of the 13rd IEEE International Conference on Parallel and Distributed Systems (ICPADS'07)* , 2007

Efficient Scheduling of Task Graph Collections on Heterogeneous Resources
Matthieu Gallet, Loris Marchal, Frédéric Vivien, *Proceedings of the 23rd International Parallel and Distributed Processing Symposium (IPDPS'09)* , 2009

Allocating Series of Workflows on Computing Grids
Matthieu Gallet, Loris Marchal, Frédéric Vivien, *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)* , 2008

Computing the throughput of replicated workflows on heterogeneous platforms
Anne Benoit, Matthieu Gallet, Bruno Gaujal, Yves Robert, *Proceedings of the 38th International Conference on Parallel Processing (ICPP'09)* , 2009

Performance evaluation – running times

Average running times in seconds to schedule 1000 data sets:

	small task graphs	large task graphs
HEFT *	14.30	83.36
MLP	49.45	n/a
Delegate	16.74	40.49
Simple-Greedy	0.11	0.61
Refined-Greedy	0.12	0.81
RLP-max	166.38	1301.80
RLP-rand	16.78	812.30

*: HEFT running time grows with the number of data sets

Difference between experimental and theoretical values

Algorithm	Average error	Standard deviation
MLP	3%	3%
Simple-Greedy	8%	11%
Refined-Greedy	5%	6%
RLP-max	8%	12%
RLP-rand	16%	28%
Delegate	2%	2%
Neighborhood	6%	12%