A GRASP approach for the machine reassignment problem

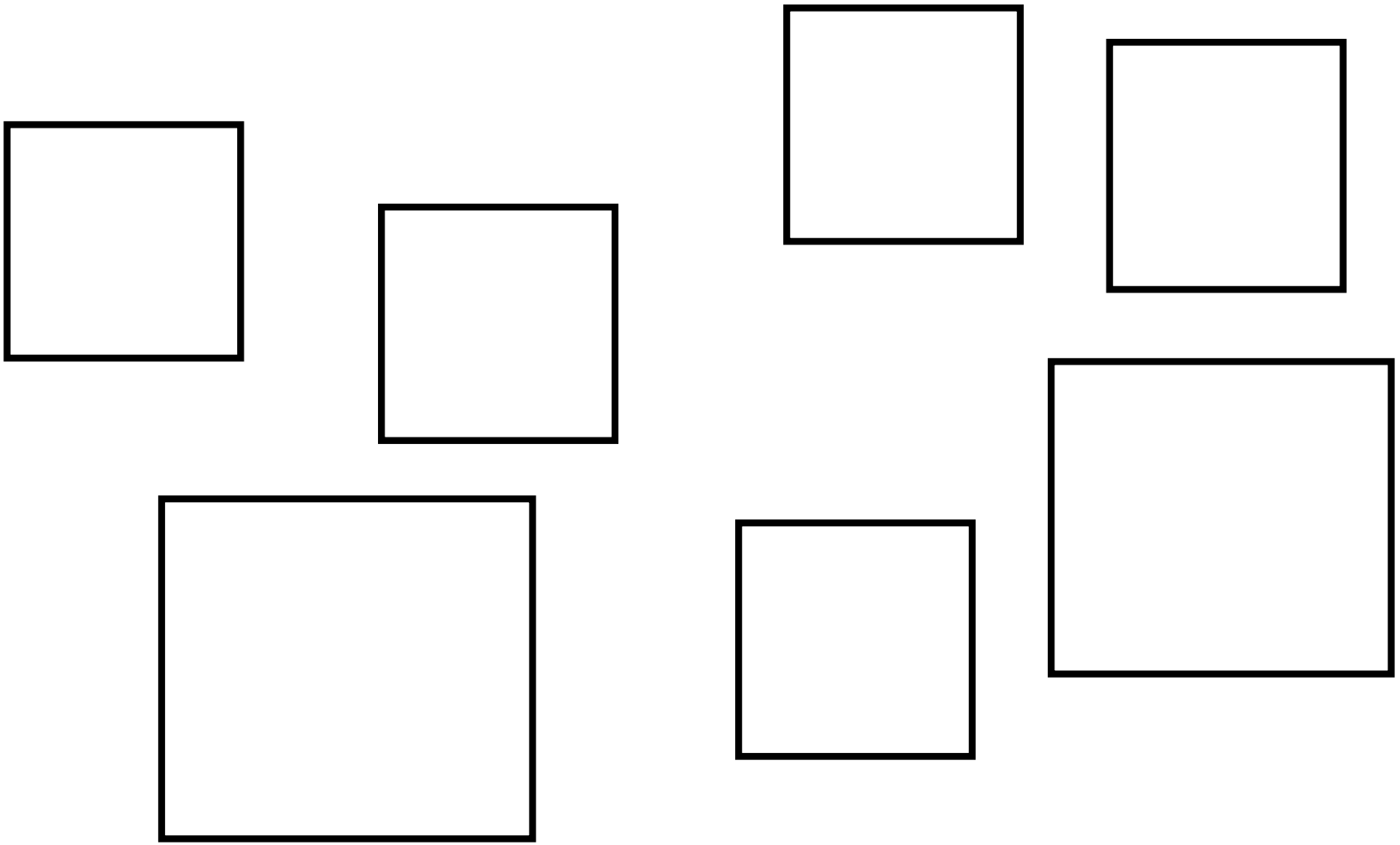
Michaël Gabay
Sofia Zaourar

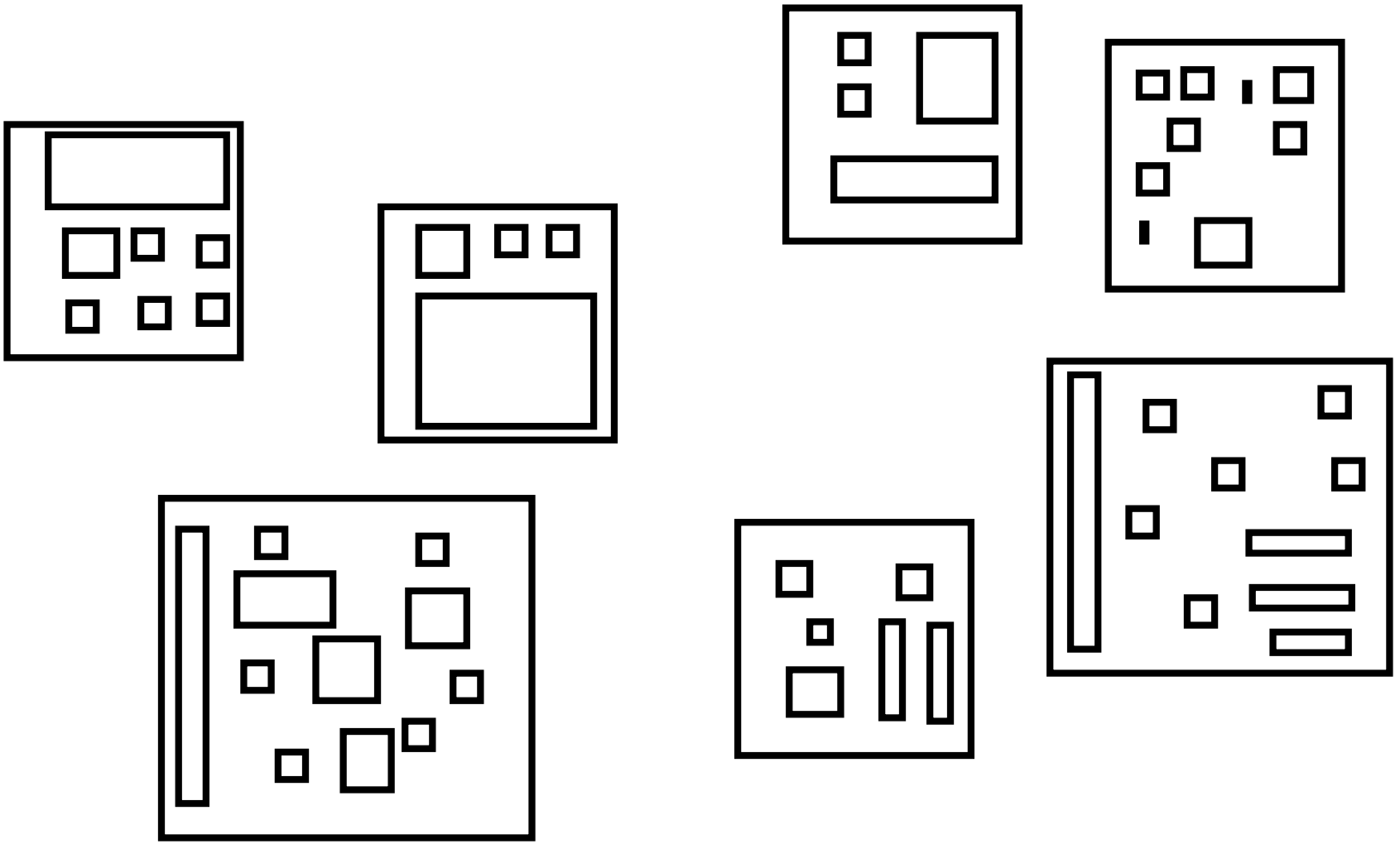
Laboratoire G-SCOP
LJK - Inria Grenoble
France

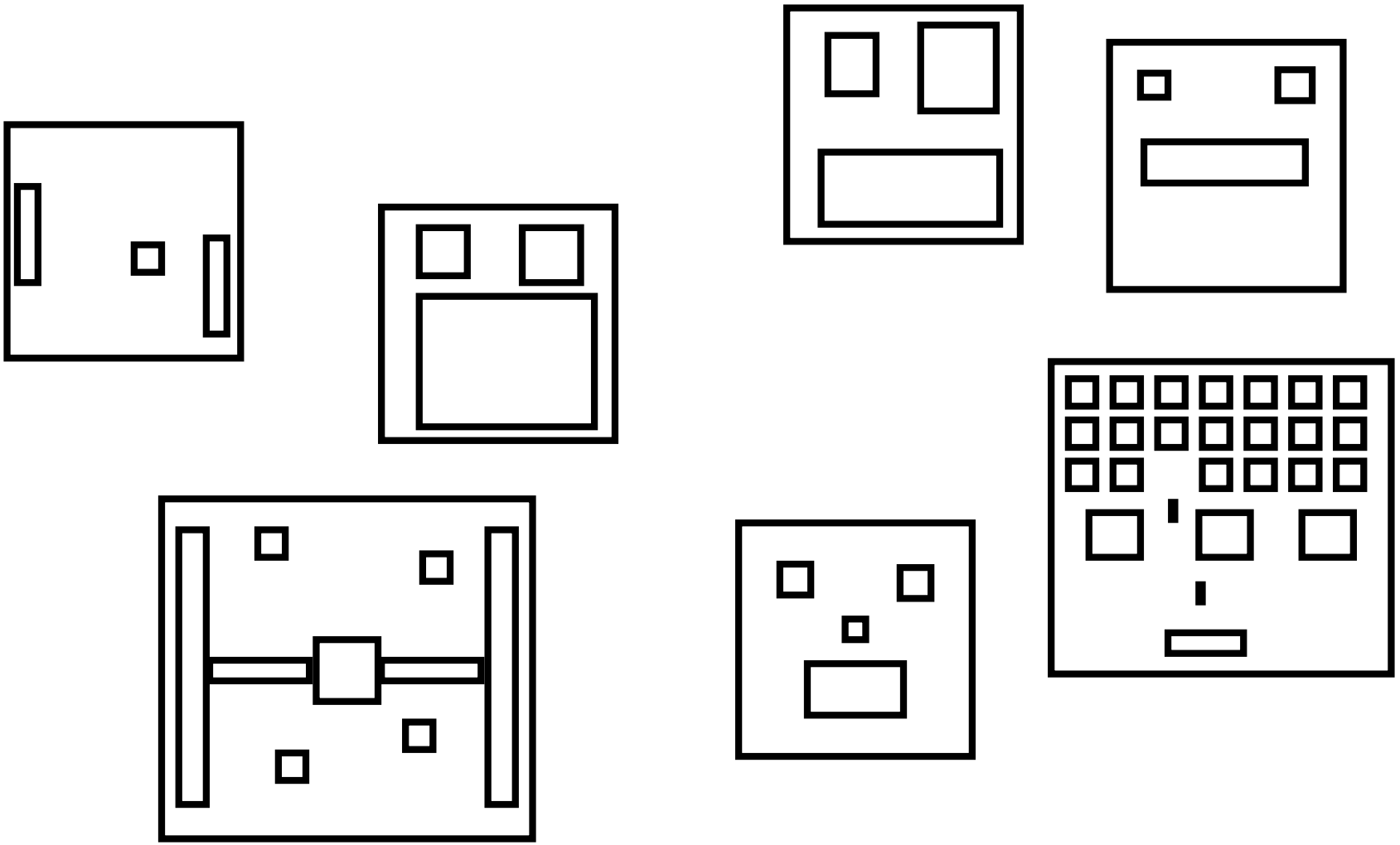
Team J19 - Open source

July 10, 2012

Summary



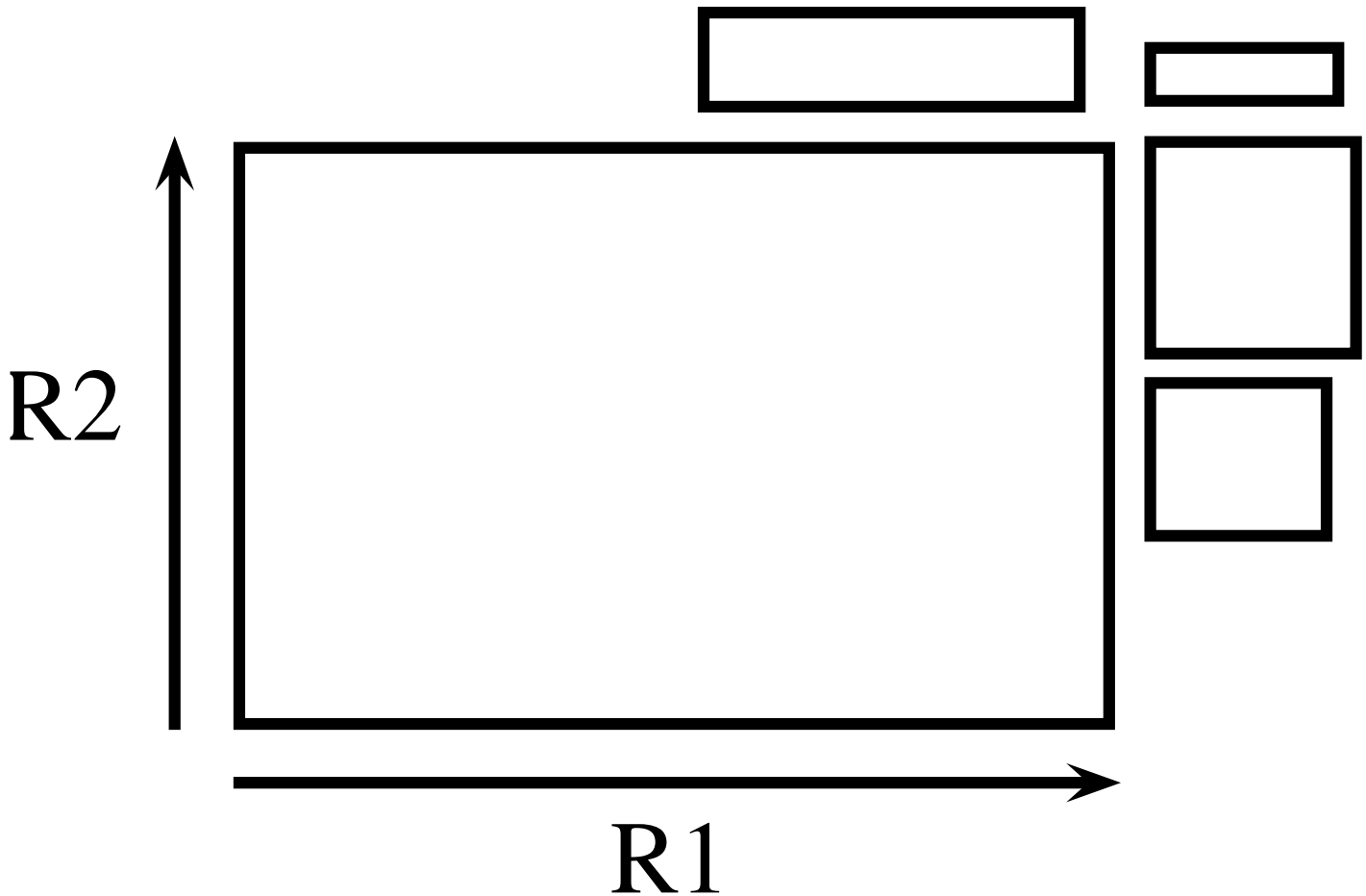




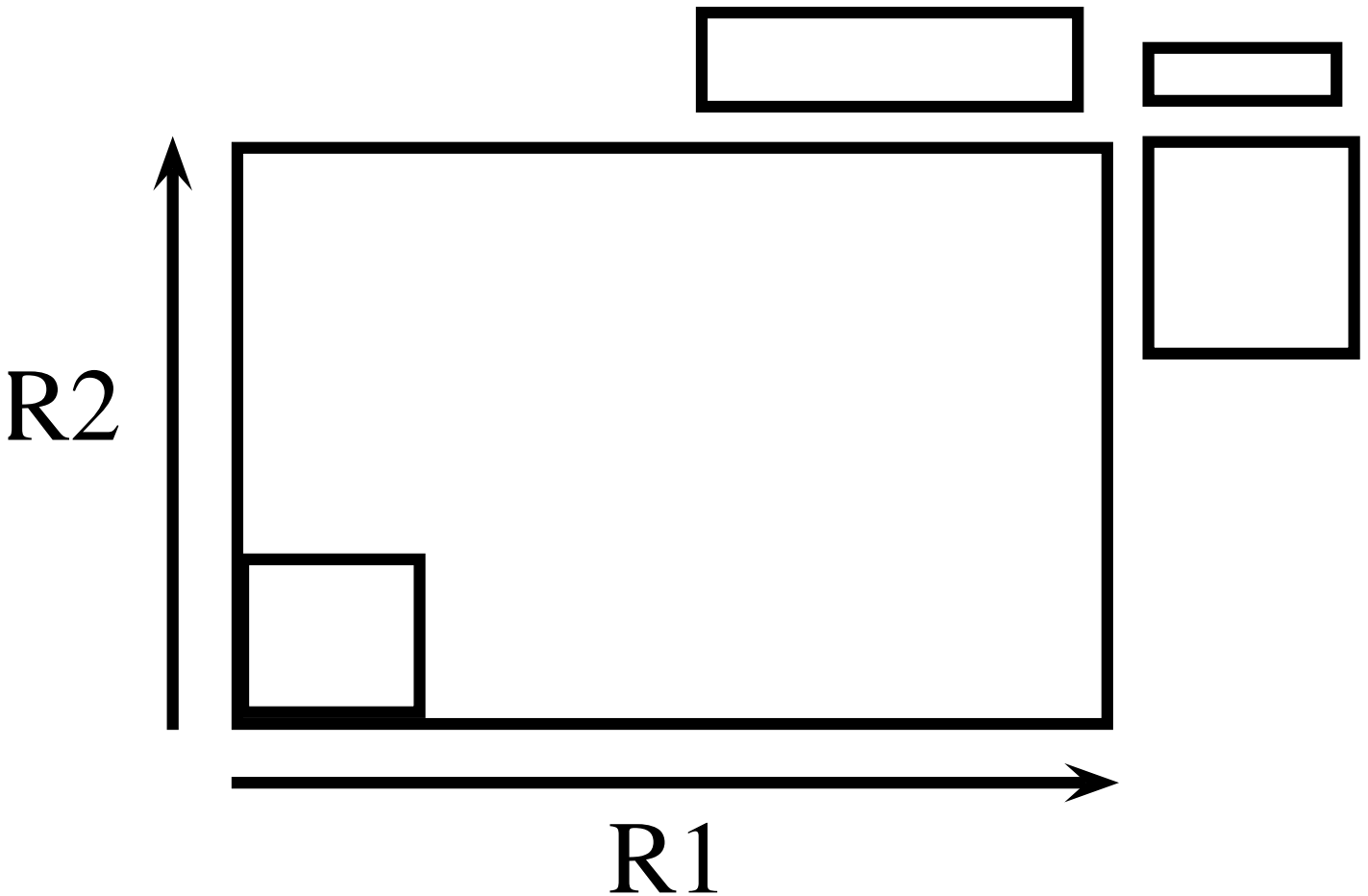
Constraints

- Capacity constraints and transient usage constraints
- Conflict constraints
- Spread constraints
- **Dependency constraints**

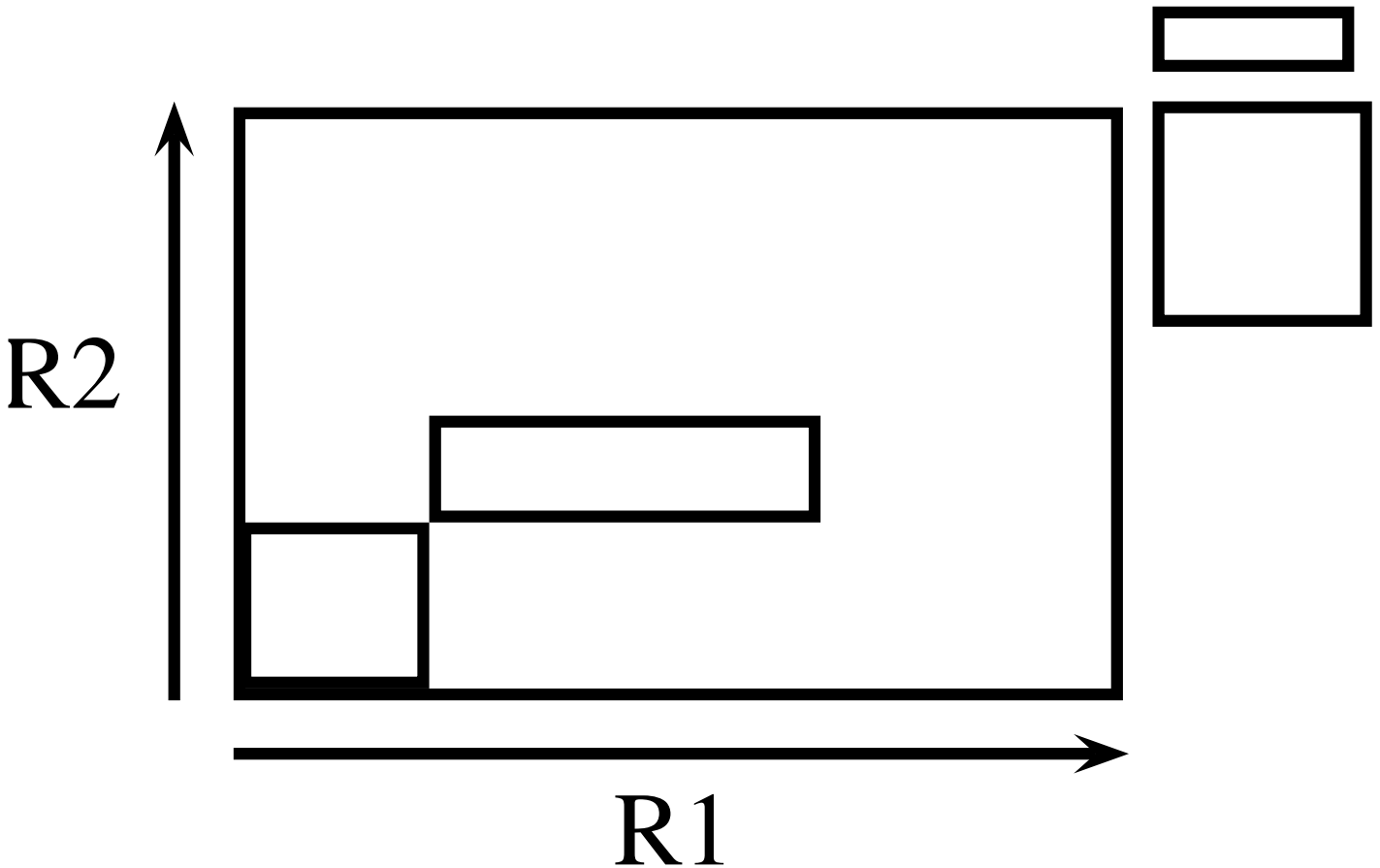
Capacity constraints



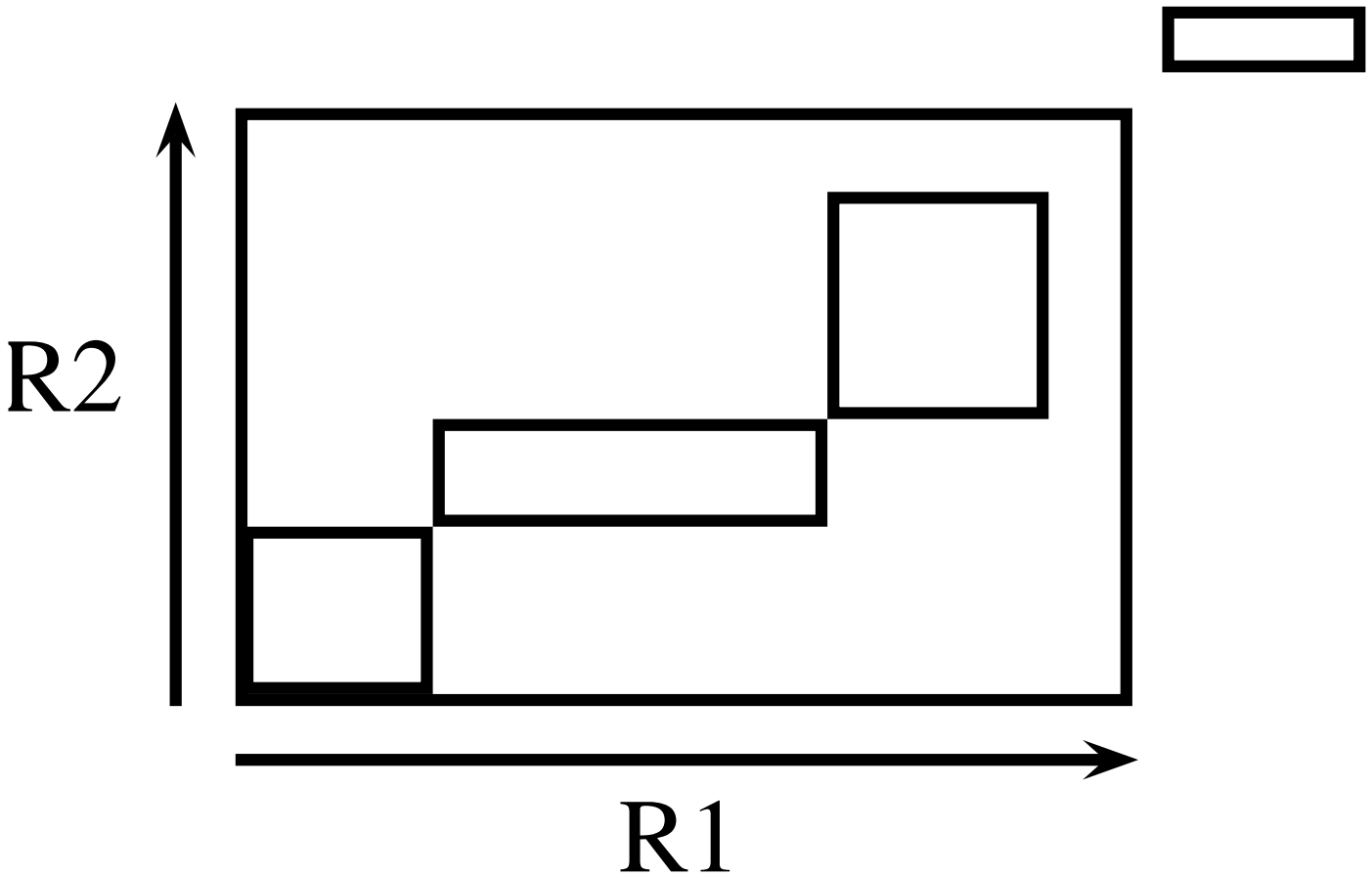
Capacity constraints



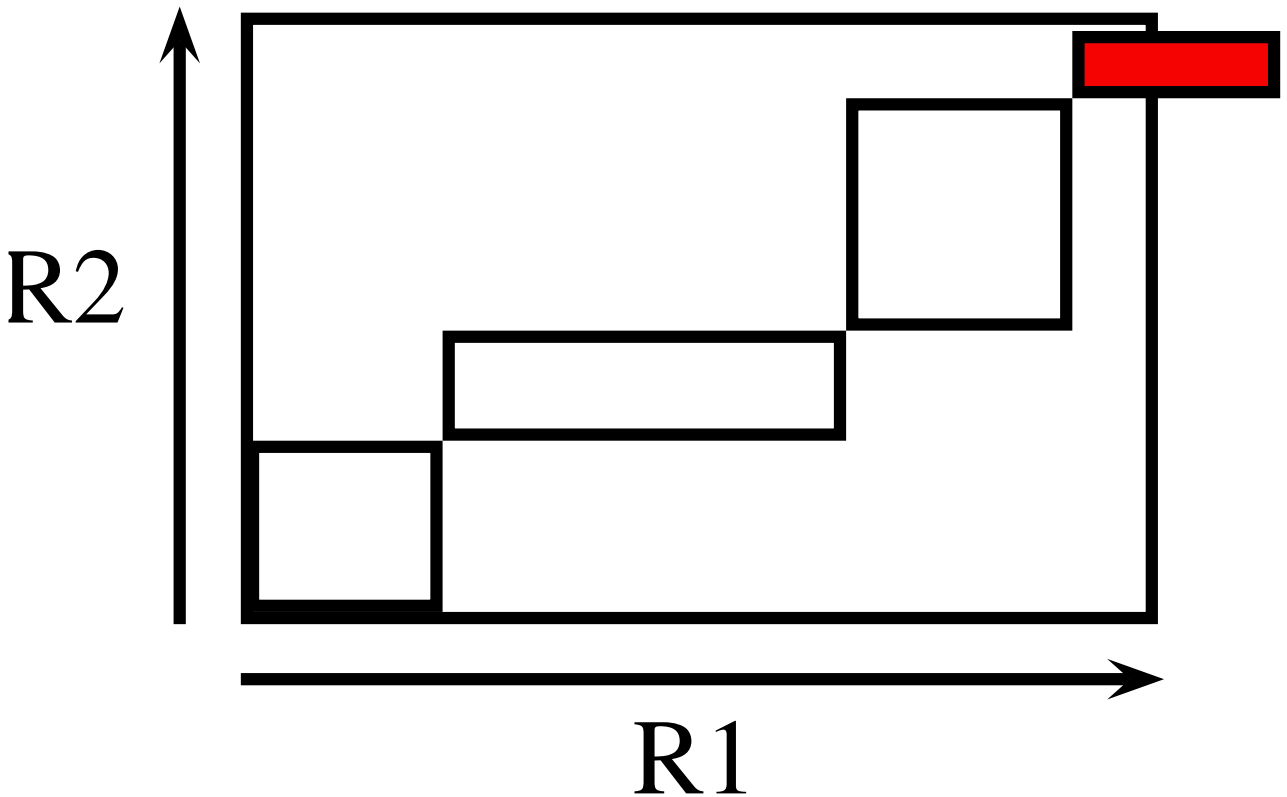
Capacity constraints



Capacity constraints

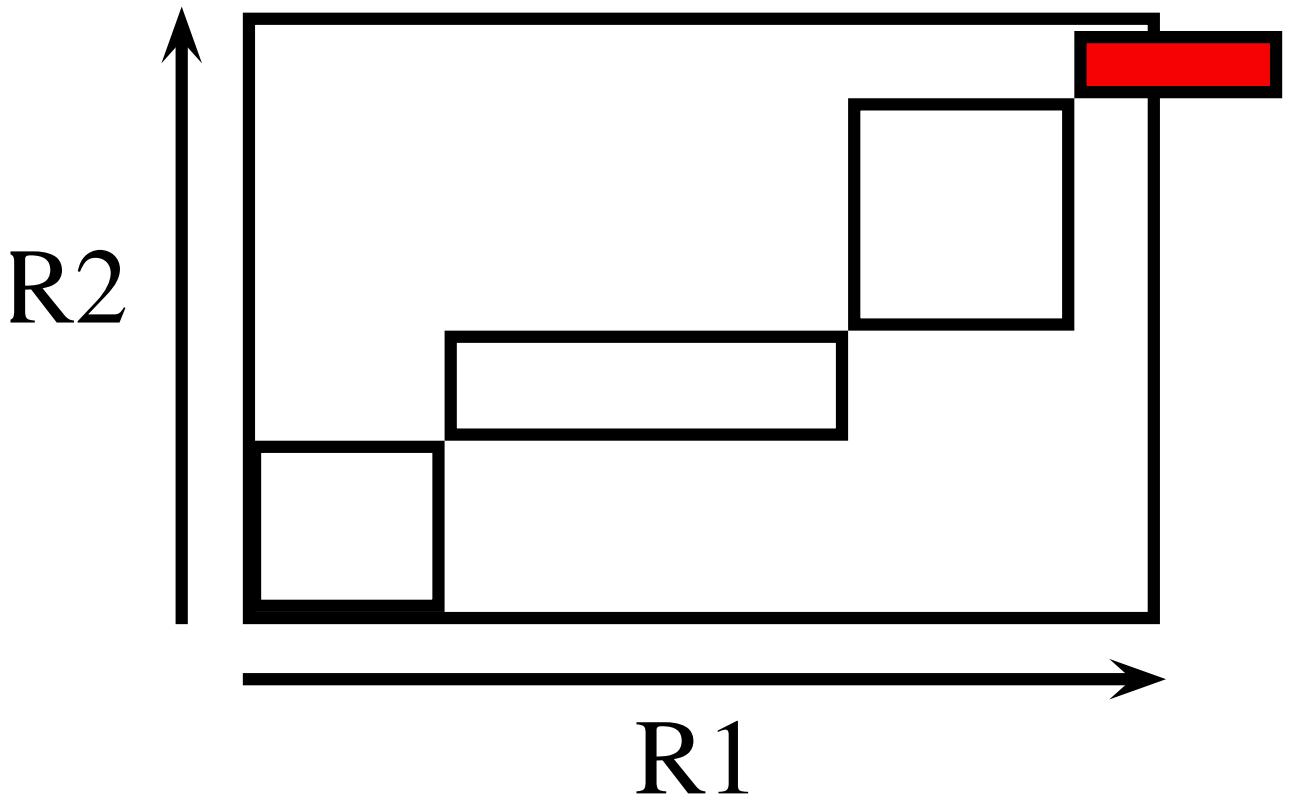


Capacity constraints



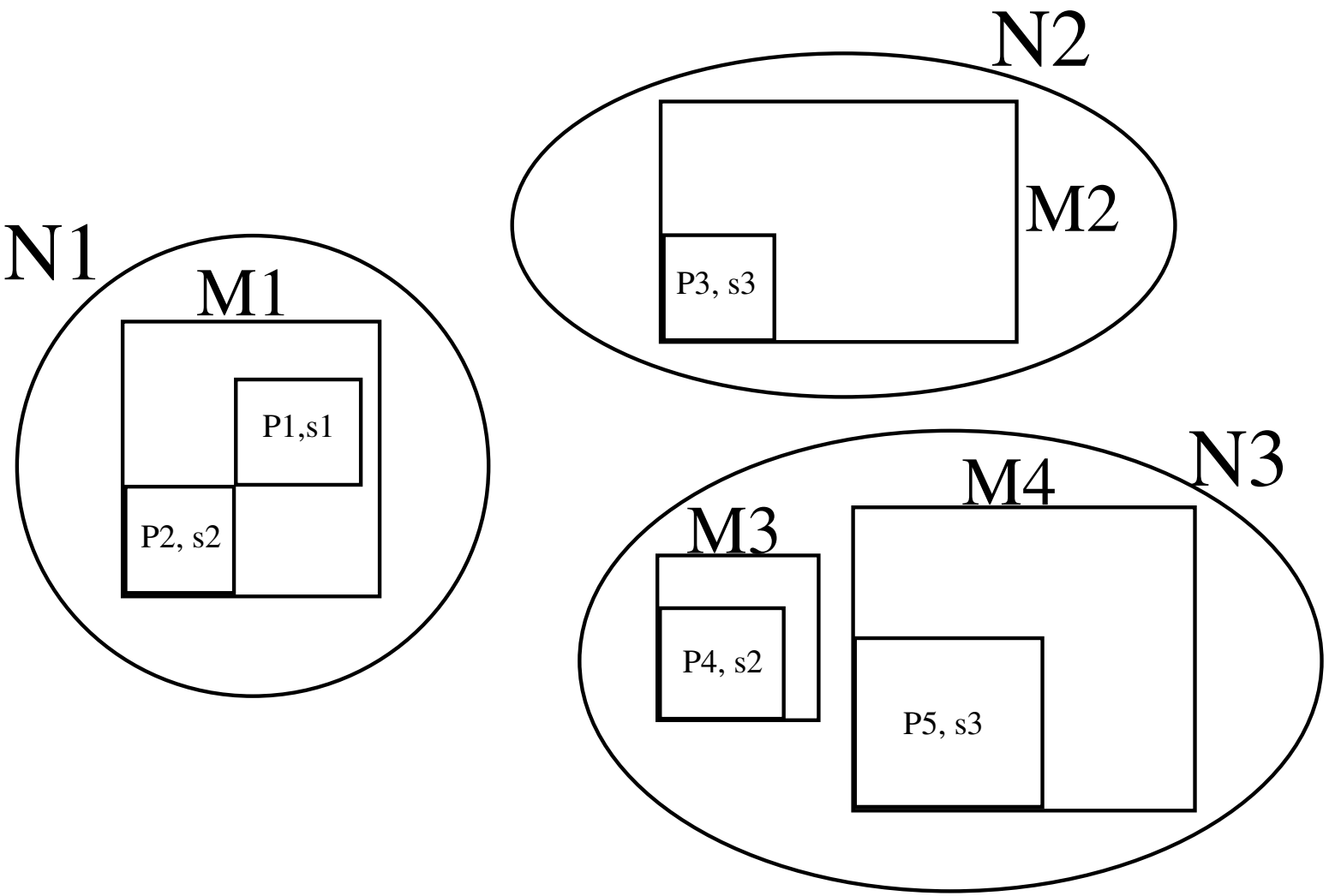
Capacity constraints

Vector bin packing problem, Garey et al., 76



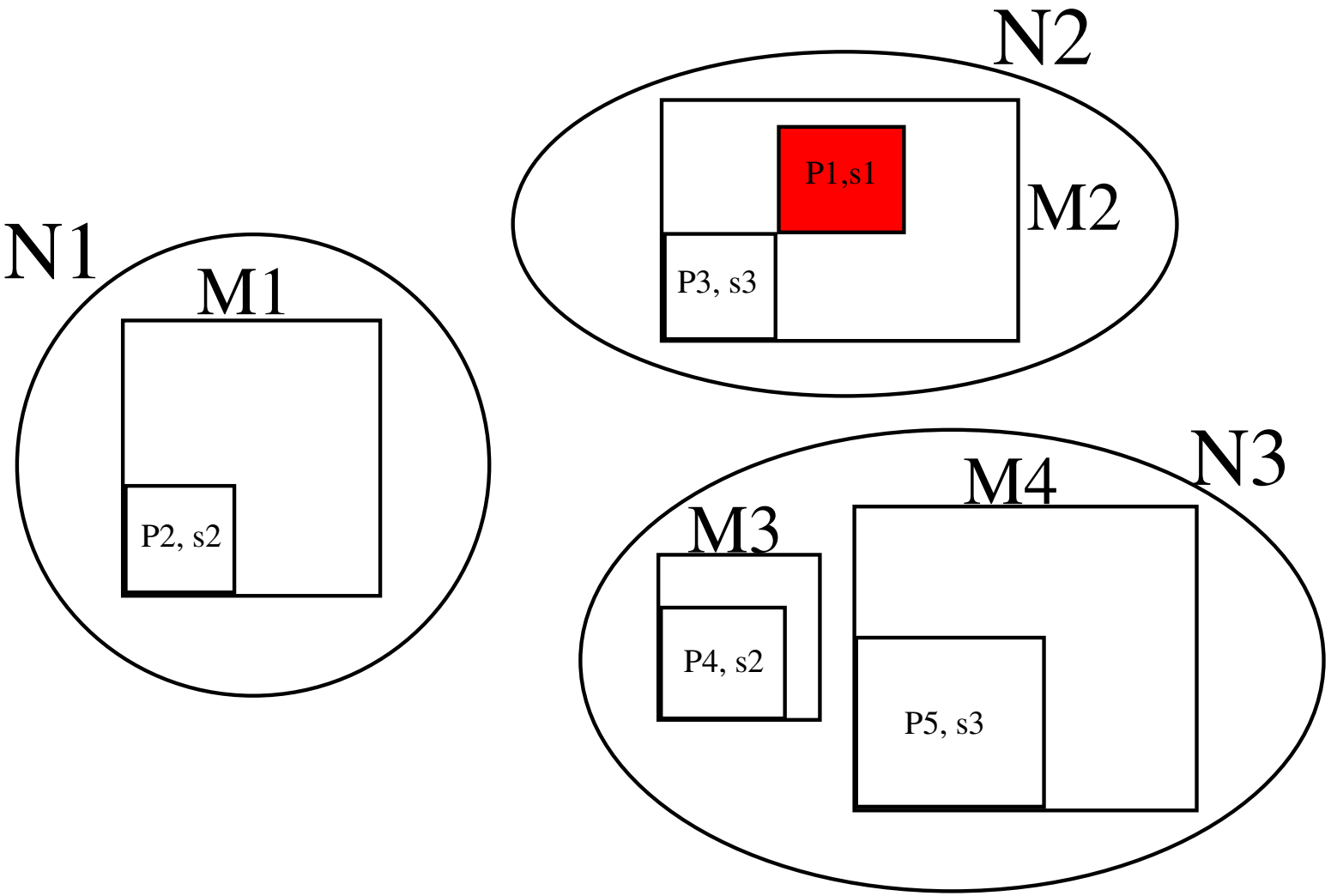
Dependency constraints

s_1 depends on s_2:



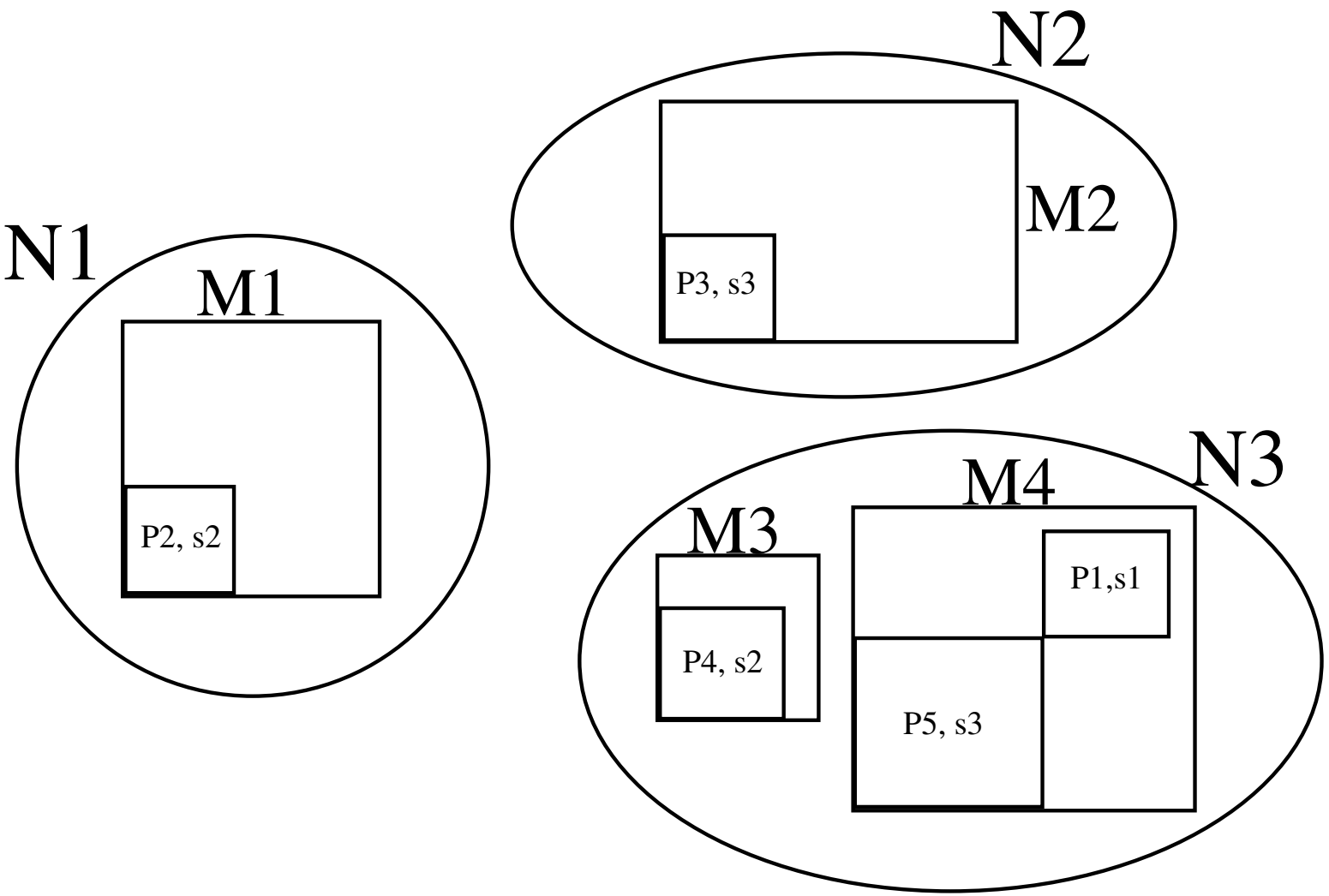
Dependency constraints

s_1 depends on s_2:



Dependency constraints

s_1 depends on s_2:



Dependency constraints

Idea : break up the problem by neighborhoods

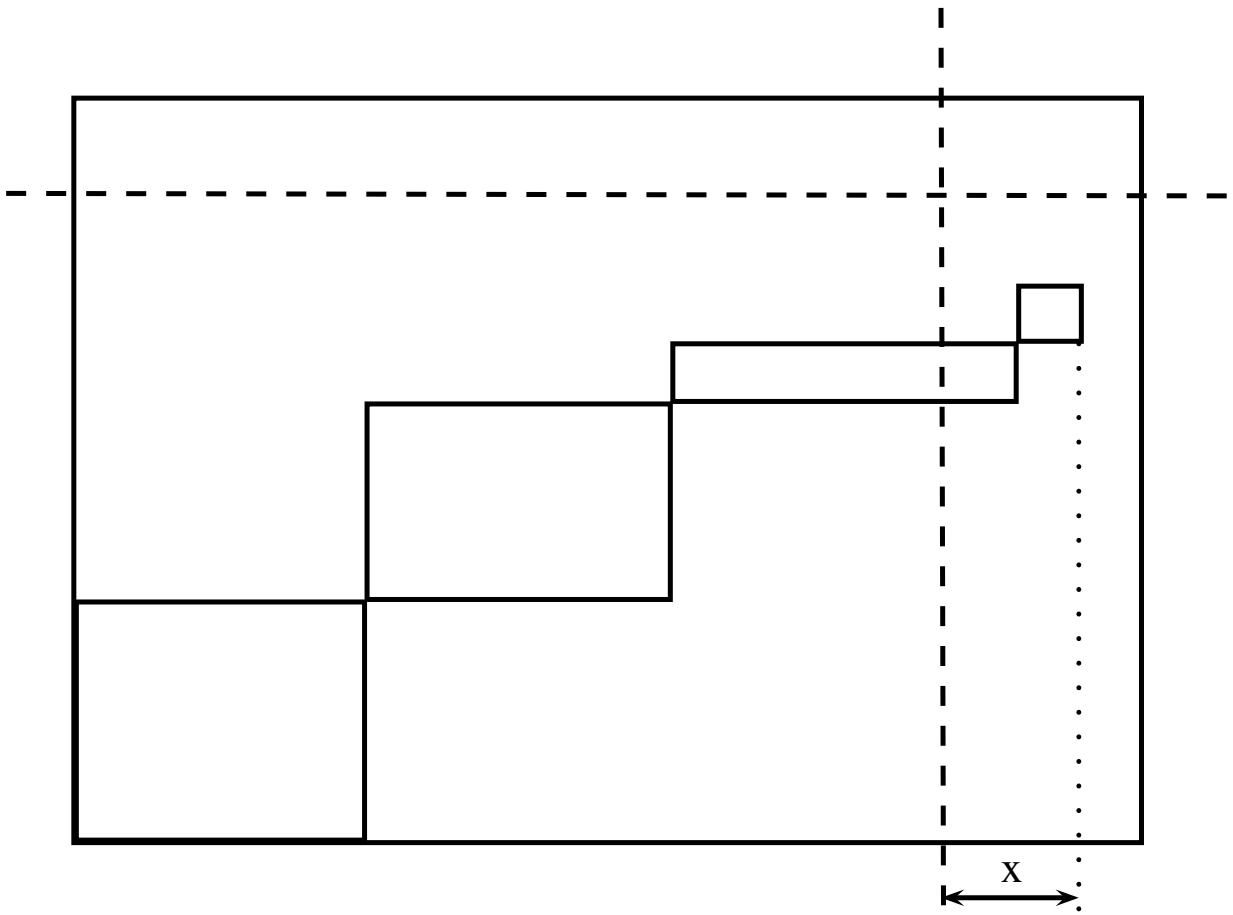
$\Rightarrow$ smaller subproblems with no dependency constraints.

Solvable using integer programming $\Rightarrow$ Matheuristic approach.

Costs

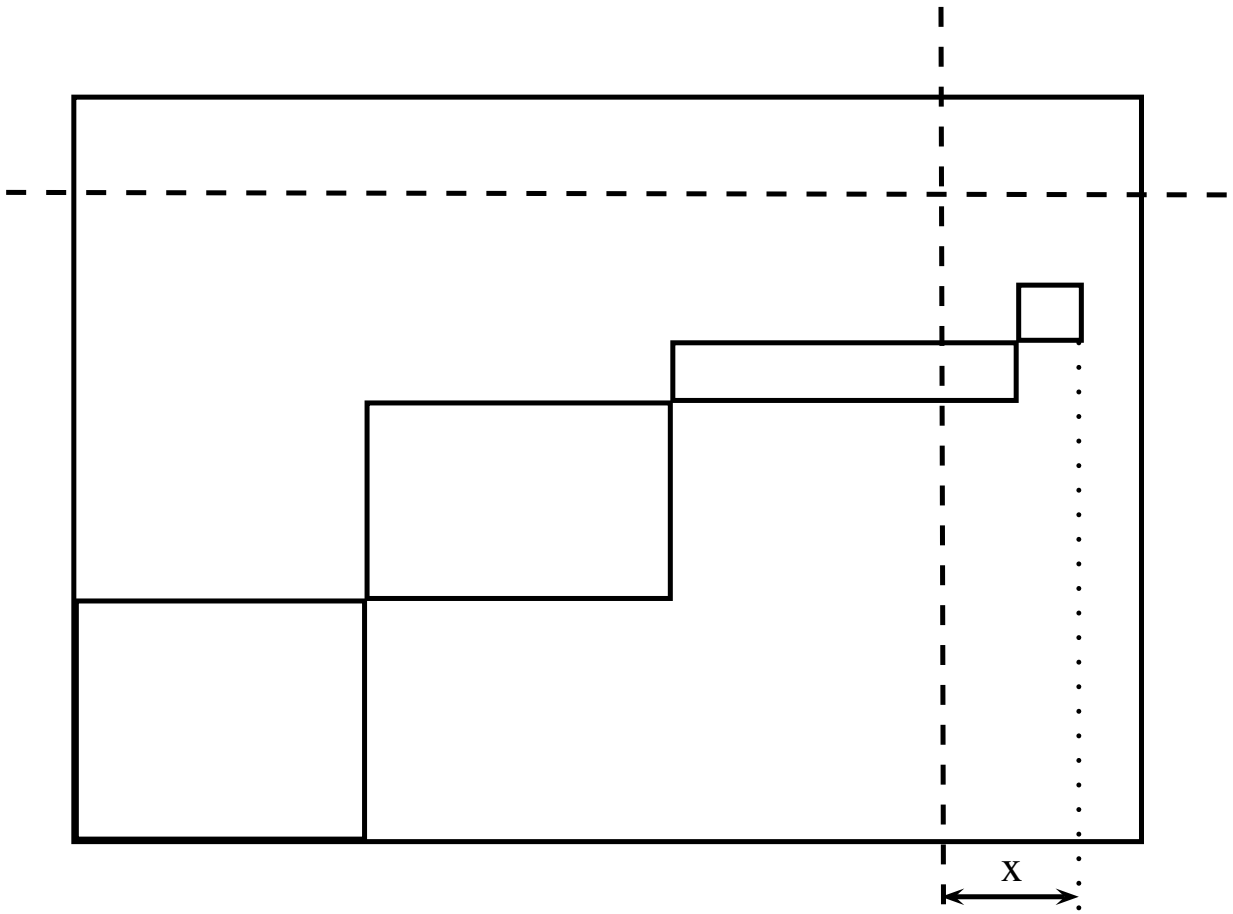
- Process / Service / Machine *move* costs
- **Load costs**
- *Balance costs*

Load costs



Load costs

Use load costs to **guide** search



GRASP ?

GRASP = **Greedy randomized adaptive search procedure**

Optimization scheme:

1. Create a solution using vector bin packing heuristic
2. Optimize it with a local search
3. Goto 1

Vector Bin Packing

$\forall m \in \mathcal{M}, \forall p \in \mathcal{P}$, volume v:

$$v(m) = \sum_{r \in \mathcal{R}} \frac{C(m,r)}{C(r)}, \quad v(p) = \sum_{r \in \mathcal{R}} \frac{R(p,r)}{R(r)}$$

Vector Bin Packing

$\forall m \in \mathcal{M}, \forall p \in \mathcal{P}$, volume v:

$$v(m) = \sum_{r \in \mathcal{R}} \frac{C(m,r)}{C(r)}, \quad v(p) = \sum_{r \in \mathcal{R}} \frac{R(p,r)}{R(r)}$$

Greedy : First Fit Decreasing (FFD)

FFD Bin-centric heuristic (Panigrahy et al., 2011):

1. Sort machines by increasing $v(m)$, processes by decreasing $v(p)$
2. Pop the smallest remaining machine m
3. While some processes fit into m, place the largest remaining one
4. Goto 2

Vector Bin Packing

$\forall m \in \mathcal{M}, \forall p \in \mathcal{P}$, volume v:

$$v(m) = \sum_{r \in \mathcal{R}} \frac{C(m,r)}{C(r)}, \quad v(p) = \sum_{r \in \mathcal{R}} \frac{R(p,r)}{R(r)}$$

Greedy : First Fit (FF)

FF Bin-Balancing (item-centric) heuristic:

1. Sort machines by increasing $v(m)$, processes by decreasing $v(p)$, $i = 1$
2. Pop the largest remaining process p
3. While some machine can host p, place p on the smallest $j \geq i+1$ (with a cyclic order), $i = j$
4. Goto 2

Vector Bin Packing

$\forall m \in \mathcal{M}, \forall p \in \mathcal{P}$, volume v:

$$v(m) = \sum_{r \in \mathcal{R}} \frac{C(m,r)}{C(r)}, \quad v(p) = \sum_{r \in \mathcal{R}} \frac{R(p,r)}{R(r)}$$

Greedy Random

FF Mixed orderings heuristic:

1. Sort machines by increasing $v(m)$, processes by decreasing $v(p)$, or random sort both
2. Run FFD Bin-centric, run FF Bin-Balancing
3. Goto 1

Vector Bin Packing

Constraints verified ?

- Capacity and transient usage constraints
- Conflict constraints
- Spread constraints
- Dependency constraints

Vector Bin Packing

Constraints verified ?

- Capacity and transient usage constraints $\Rightarrow$ YES
- Conflict constraints $\Rightarrow$ YES
- Spread constraints
- Dependency constraints

Vector Bin Packing

Constraints verified ?

- Capacity and transient usage constraints $\Rightarrow$ YES
- Conflict constraints $\Rightarrow$ YES
- Spread constraints
- Dependency constraints $\Rightarrow$ YES - Assign all processes of a neighborhood to the same neighborhood (possibly not the original one)

Vector Bin Packing

Constraints verified ?

- Capacity and transient usage constraints $\Rightarrow$ YES
- Conflict constraints $\Rightarrow$ YES
- Spread constraints $\Rightarrow$ **REPAIR**
- Dependency constraints $\Rightarrow$ YES - Assign all processes of a neighborhood to the same neighborhood (possibly not the original one)

GRASP

1. Use FF Mixed orderings to get a feasible solution
2. Local search on the solution
3. Goto 1

Problems with VBP

VBP heuristics are very fast but...

- Solutions' costs are too high
- Not so interesting when processes do not change neighborhood,... nor when they all change...
- 10 B instances, 6 feasible,... all violate spread constraints
- Repairing may be too difficult / time consuming

A matheuristic

Idea:

1. For each **neighborhood**, solve the assignment problem using an **IP** (smaller subproblem with no dependency constraints)
2. Optimize using local search
3. Goto 1

(implemented using `Coin-Osi/Clp/Cbc`)

A matheuristic: Problems

A matheuristic: Problems

Subproblems are still too big

A matheuristic: Problems

Subproblems are still too big

Solution:

Divide into smaller subproblems
Randomly / Guided by load cost (most expensive machines with cheapest w.r.t. load costs)

A matheuristic: Problems

Tune size parameters:

- Compromise between time consumption and feasibility

A matheuristic: Problems

Tune size parameters:

Compromise between time consumption and feasibility

Solution:

- Set maximum number of machines and processes to be considered at once
- Set maximum number of nodes

A matheuristic: Problems

Pure local search is better

A matheuristic: Problems

Pure local search is better (and deadline approaches)

A matheuristic: Problems

Pure local search is better (and deadline approaches)

Solution:

Forget about the Matheuristic :-(

Local search

Two simple moves:

- Move p from m_1 to m_2
- Swap p_1 and p_2 on m_1 and m_2

Efficient Structures + randomization

Local search

Two simple moves:

- Move p from m_1 to m_2
- Swap p_1 and p_2 on m_1 and m_2

Efficient Structures + randomization

Hill Climbing: if a move is feasible and decreases the total cost, do it!

Local search

Two simple moves:

- Move p from m_1 to m_2
- Swap p_1 and p_2 on m_1 and m_2

Efficient Structures + randomization

Hill Climbing: if a move is feasible and decreases the total cost, do it!

Guided first step: Move processes from the machines with the highest load costs to cheaper machines

Local search

Two simple moves:

- Move p from m_1 to m_2
- Swap p_1 and p_2 on m_1 and m_2

Efficient Structures + randomization

Hill Climbing: if a move is feasible and decreases the total cost, do it!

Guided first step: Move processes from the machines with the highest load costs to cheaper machines

Blocked ? Restart from a previous solution with a new seed

Final solution

Local search

Two parallel local search with different strategies

Final solution

Local search

Two parallel local search with different strategies

Open source under `LGPL v3` license, available at:

`https://github.com/TeamJ19ROADEF2012`

(Includes VBP and the matheuristic)

Intel Core 2 duo P9400, RAM 4 Go

Instance	Cost	Real time	CPU time
B 1	3 609 228 327	4m58.5s	9m53.4s
B 2	1 017 459 868	4m58.5s	9m54.4s
B 3	170 139 317	4m58.5s	9m53.4s
B 4	4 677 960 720	4m58.5s	9m54.2s
B 5	930 031 137	4m58.5s	9m53.8s
B 6	9 525 886 513	4m58.5s	9m53.7s
B 7	14 911 018 492	4m58.5s	9m52.4s
B 8	1 217 854 951	4m58.5s	9m53.0s
B 9	15 886 884 119	4m58.5s	9m54.1s
B 10	18 391 528 354	4m58.5s	9m51.0s

Questions ?