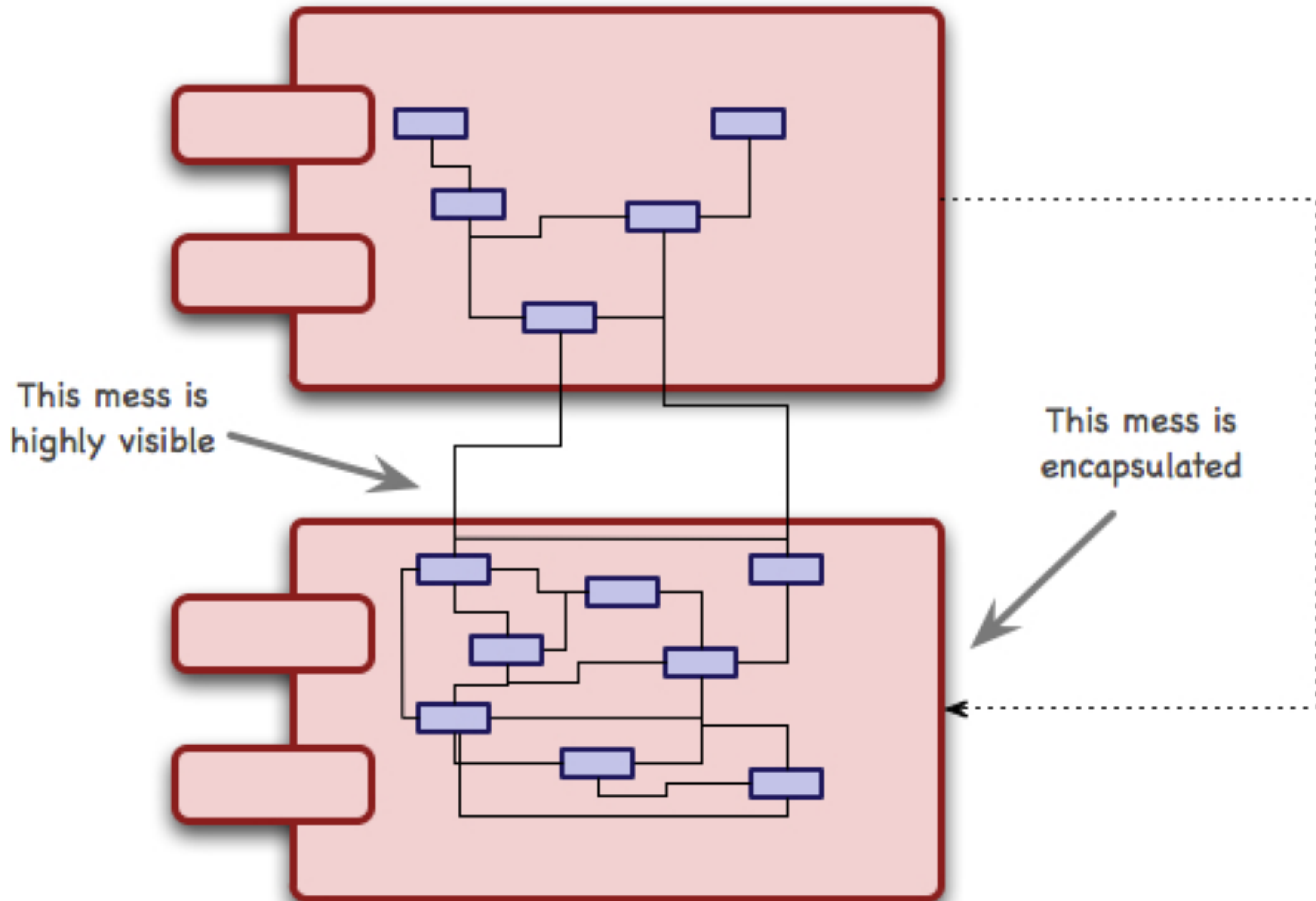


Image from: http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg

Example of Hill Climbing Application: Software Module Clustering (Problem Formulation)

Leandro L. Minku

Hill Climbing Applications

Hill-Climbing is applicable to any optimisation problem, but its success depends on the shape of the objective function for the problem instance in hands.

Simple algorithm — not difficult to implement.
Could be attempted first to see if the retrieved solutions are good enough, before a more complex algorithm is investigated.

Applications

- Hill-climbing has been successfully applied to software module clustering.

- Software Module Clustering:
    - Software is composed of several units, which can be organised into modules.
    - Well modularised software is easier to develop and maintain.
    - As software evolves, modularisation tends to degrade.

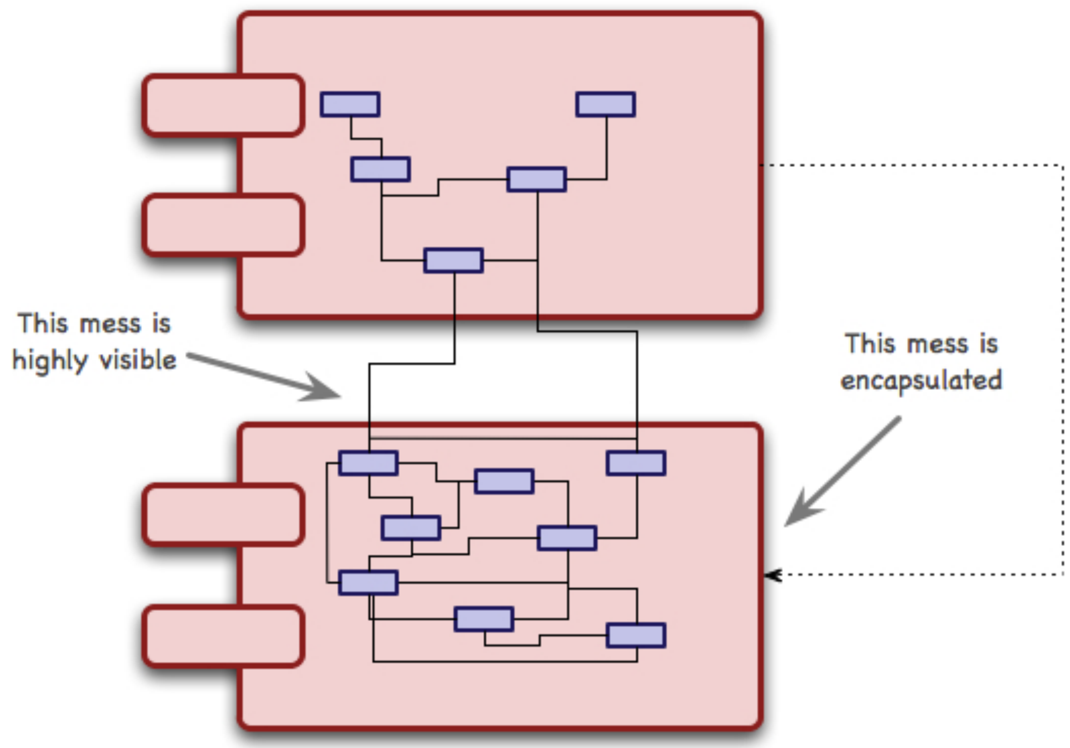


Image from: http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg

Problem: find an allocation of units into modules that maximises the quality of modularisation.

Applying Hill-Climbing (and Simulated Annealing)

- We need to specify:
  - Optimisation problem formulation:
    - Design variable and search space
    - Constraints
    - Objective function
  - Algorithm-specific operators:
    - Representation.
    - Initialisation procedure.
    - Neighbourhood operator.
  - Strategy to deal with constraints, e.g.:
    - Representation, initialisation and neighbourhood operators that ensure only feasible solutions to be generated.
    - Modification in the objective function.

Formulation Optimisation Problems

- Design variables represent a candidate solution.
  - Design variables define the search space of candidate solutions.

- Objective function defines our goal.
  - Can be used to evaluate the quality of solutions.
  - Function to be optimised (maximised or minimised).

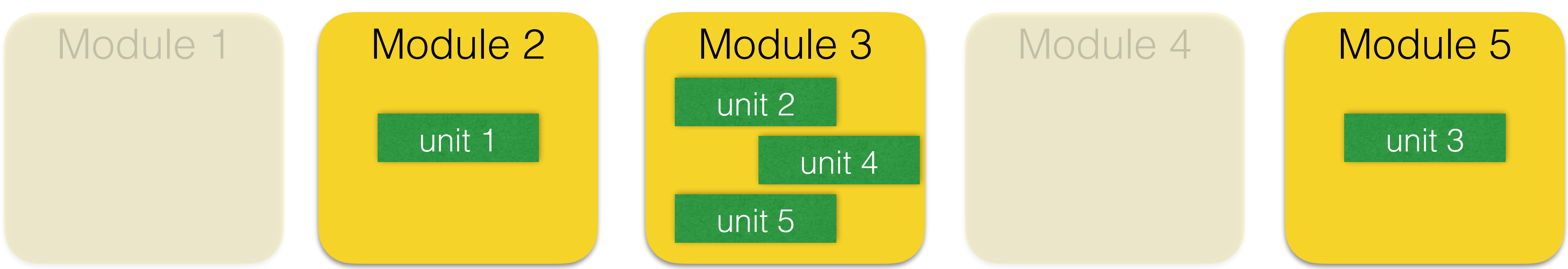
- [Optional] Solutions must satisfy certain constraints.

Design Variable

Design variable: allocation of units into modules.

- Consider that we have N units, identified by natural numbers in $\{1,2,\ldots,N\}$.
- This means that we have at most N modules.
- Our design variable is a list L of N modules, where each module L_i, $i \in \{1,2,\ldots,N\}$, is a set containing a minimum of 0 and a maximum of N units.

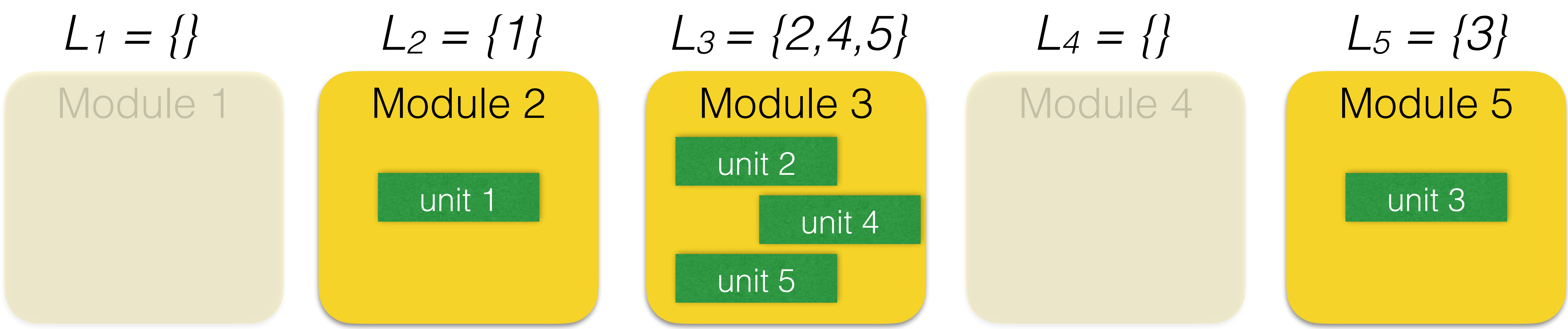
$L_1 = \{\}$ $L_2 = \{1\}$ $L_3 = \{2,4,5\}$ $L_4 = \{\}$ $L_5 = \{3\}$



Design Variable

Design variable: allocation of units into modules.

- Consider that we have N units, identified by natural numbers in $\{1,2,\ldots,N\}$.
- This means that we have at most N modules.
- Our design variable is a list L of N modules, where each module L_i, $i \in \{1,2,\ldots,N\}$, is a set containing a minimum of 0 and a maximum of N units.



Search space: all possible allocations.

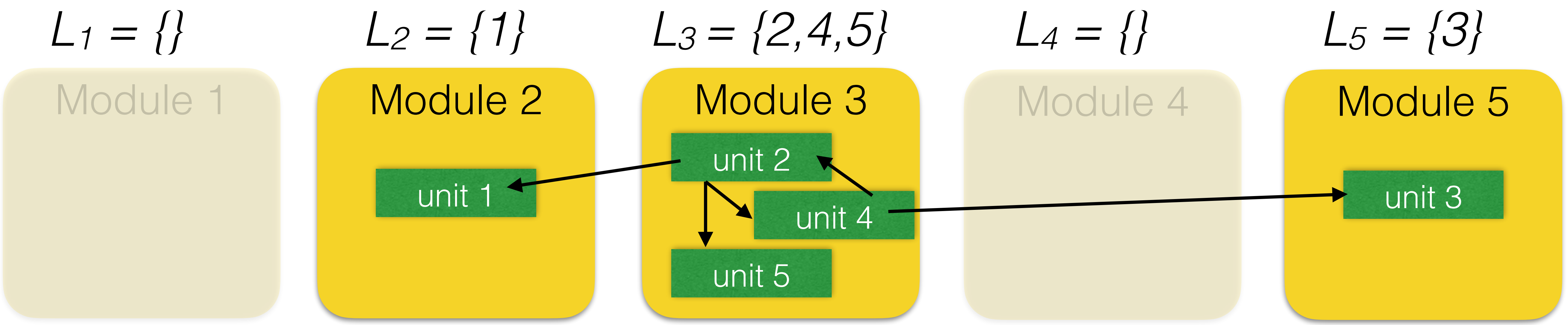
Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?



A unit can make use of (depend on) another unit — this information can be retrieved from the current source code being refactored.

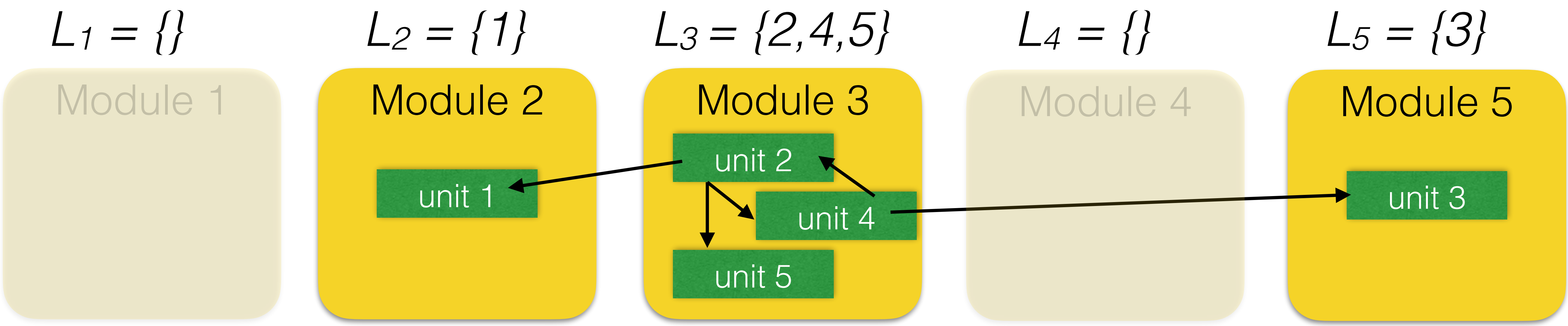
Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?



Lots of connections inside a module (high cohesion) and few connections between modules (low coupling).

Quality of a Module L_i

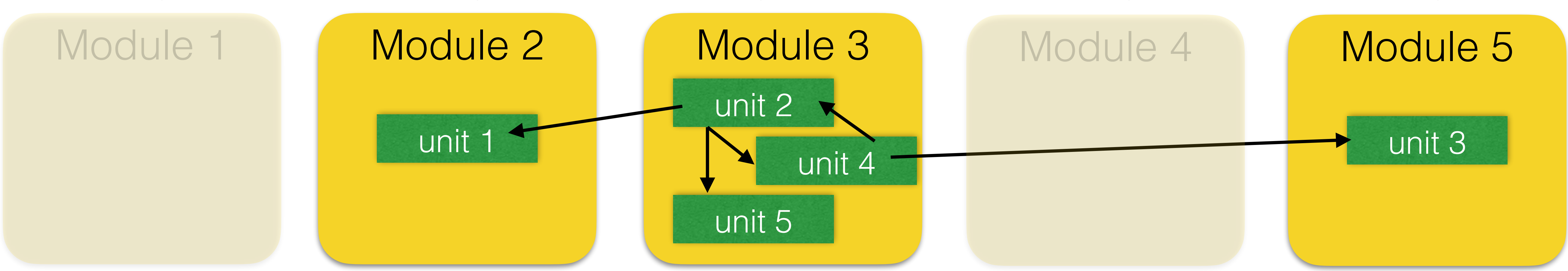
Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$L_1 = \{\}$ $L_2 = \{1\}$ $L_3 = \{2,4,5\}$ $L_4 = \{\}$ $L_5 = \{3\}$



$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{\#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

(maximise)

1/2 is used to split the penalty of inter-edges across the two modules connected by that edge

Quality of a Module L_i

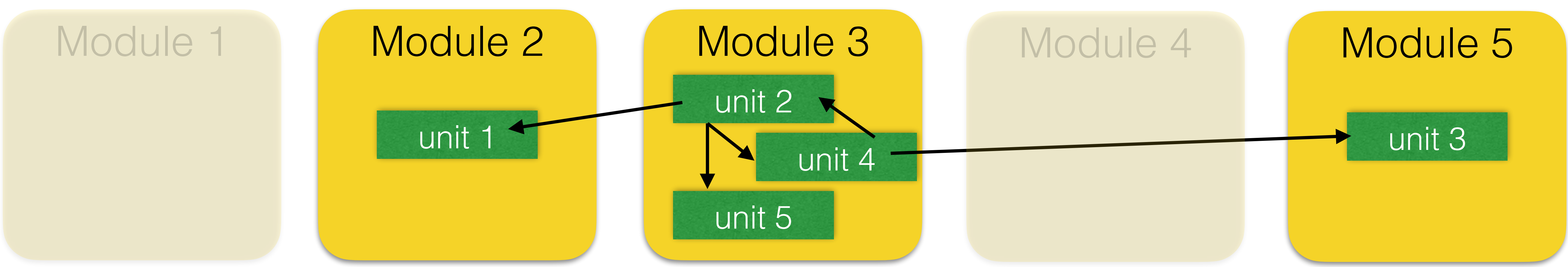
Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$L_1 = \{\}$ $L_2 = \{1\}$ $L_3 = \{2,4,5\}$ $L_4 = \{\}$ $L_5 = \{3\}$



$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{\#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

(maximise)

The #intra_edges_i in the denominator is a normalisation factor.

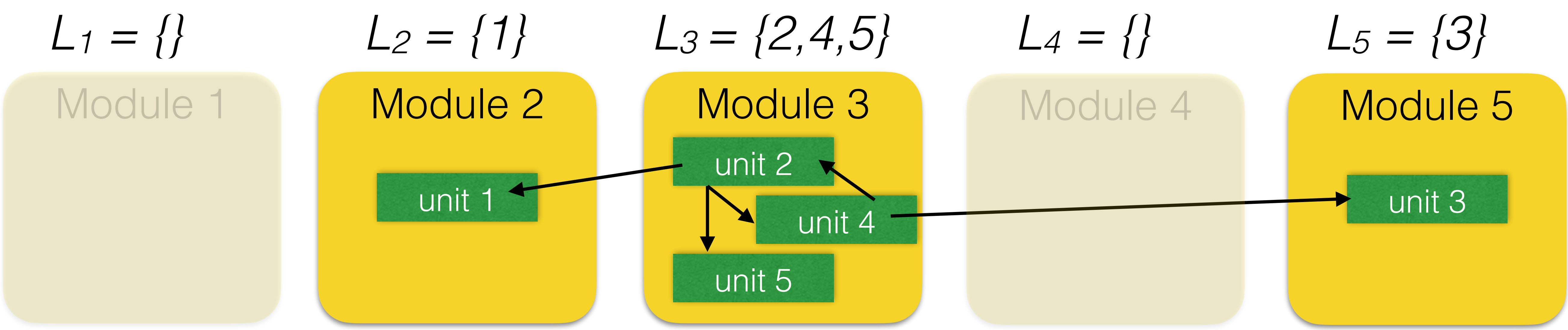
Intra Edges

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?



$$\#\mathrm{IntraEdges}_i = \sum_{j=1}^{size(L_i)} \sum_{j'=1}^{size(L_i)} D_{L_{ij},L_{ij'}} \qquad D_{a,b} = \begin{cases} 1, \text{ if unit } a \text{ depends on unit } b \\ 0, \text{ otherwise (incl. diagonal)} \end{cases}$$

Inter Edges

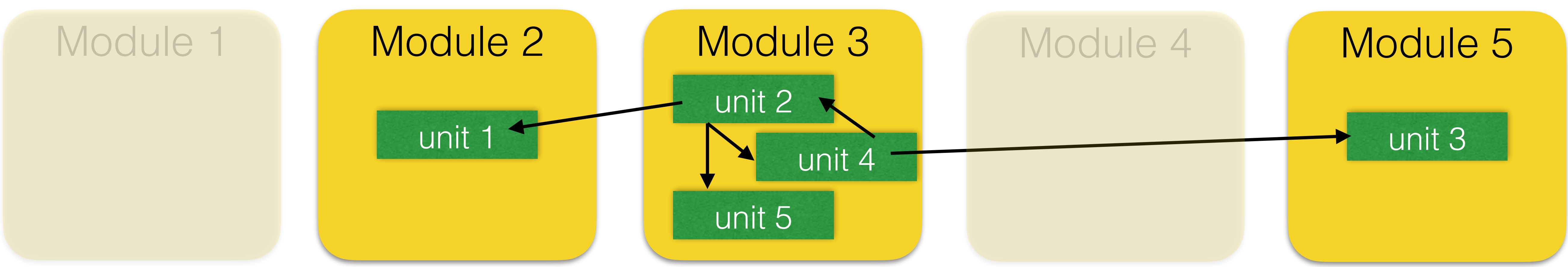
Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$L_1 = \{\}$ $L_2 = \{1\}$ $L_3 = \{2,4,5\}$ $L_4 = \{\}$ $L_5 = \{3\}$



$$\#\text{InterEdges}_i = \sum_{j=1}^{size(L_i)} \sum_{i' \in \{1,2,\cdots,N\} | i' \neq i} \sum_{j'=1}^{size(L_{i'})} (D_{L_{ij},L_{i'j'}} + D_{L_{i'j'},L_{ij}})$$

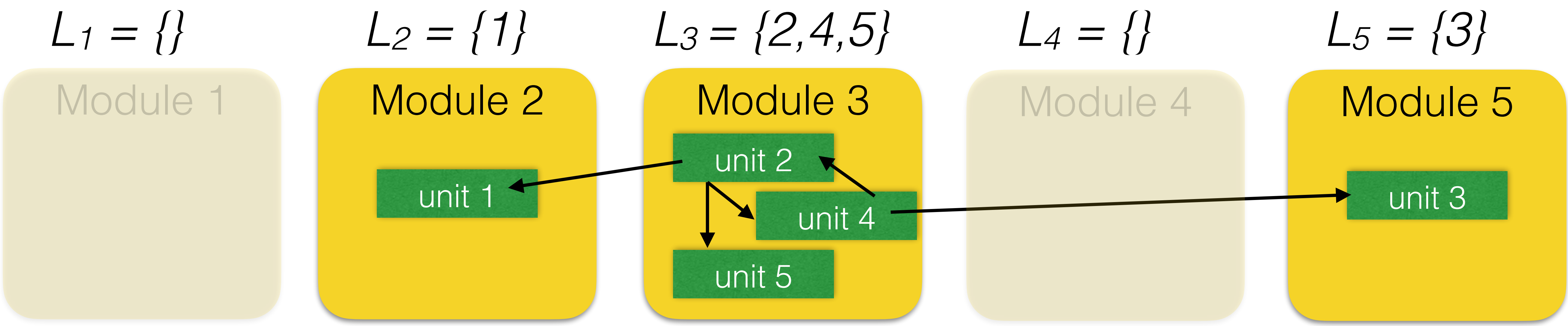
Quality of a Module L_i

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?



$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{\#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

(maximise)

This is the quality of a **single** module.

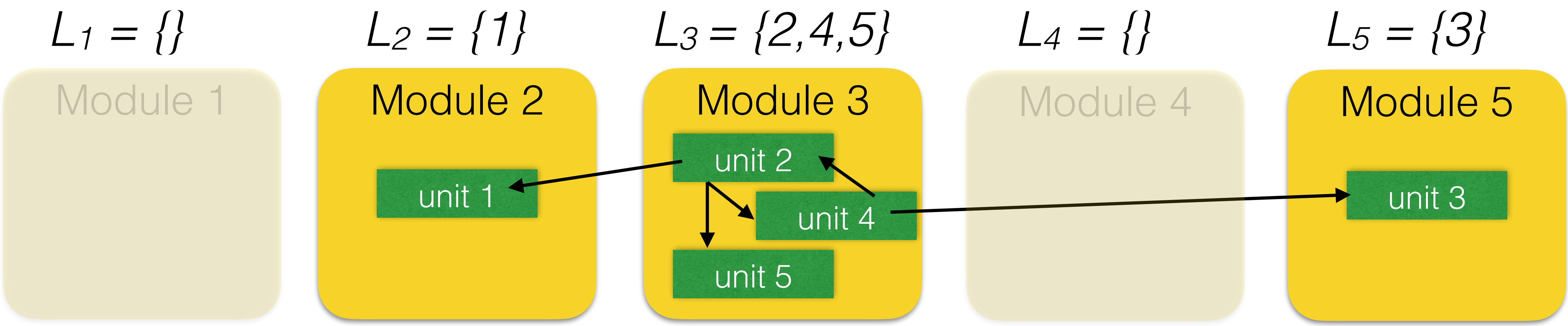
Quality of a Solution *L*

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?



Quality(L) = sum of the qualities of the non-empty modules (maximise)

Quality of a Solution *L*

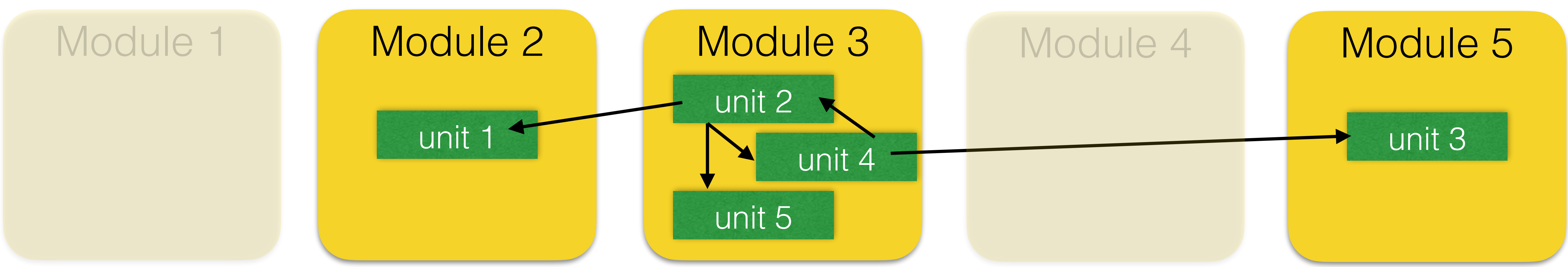
Constraints: N/A

Objective function: quality of modularisation (to be maximised).

How to compute quality?

What does good quality mean?

$L_1 = \{\}$ $L_2 = \{1\}$ $L_3 = \{2,4,5\}$ $L_4 = \{\}$ $L_5 = \{3\}$



$$\text{Quality}(L) = \sum_{i \in \{1,2,\ldots,N\} \mid L_i \neq \{\}} \text{Quality}(L_i)$$

(maximise)

Problem Formulation

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

Design variable —>
what is a candidate solution for us?

Problem Formulation

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

Design variable —>
what is a candidate solution for us?

Objective —>
what is quality for us?

Are there any constraints that need to be satisfied?

Simulated Annealing would also require a problem formulation to be able to solve a problem.

Summary

- Software Module Clustering problem formulation.

Next

- Representation, initialisation and neighbourhood operators.