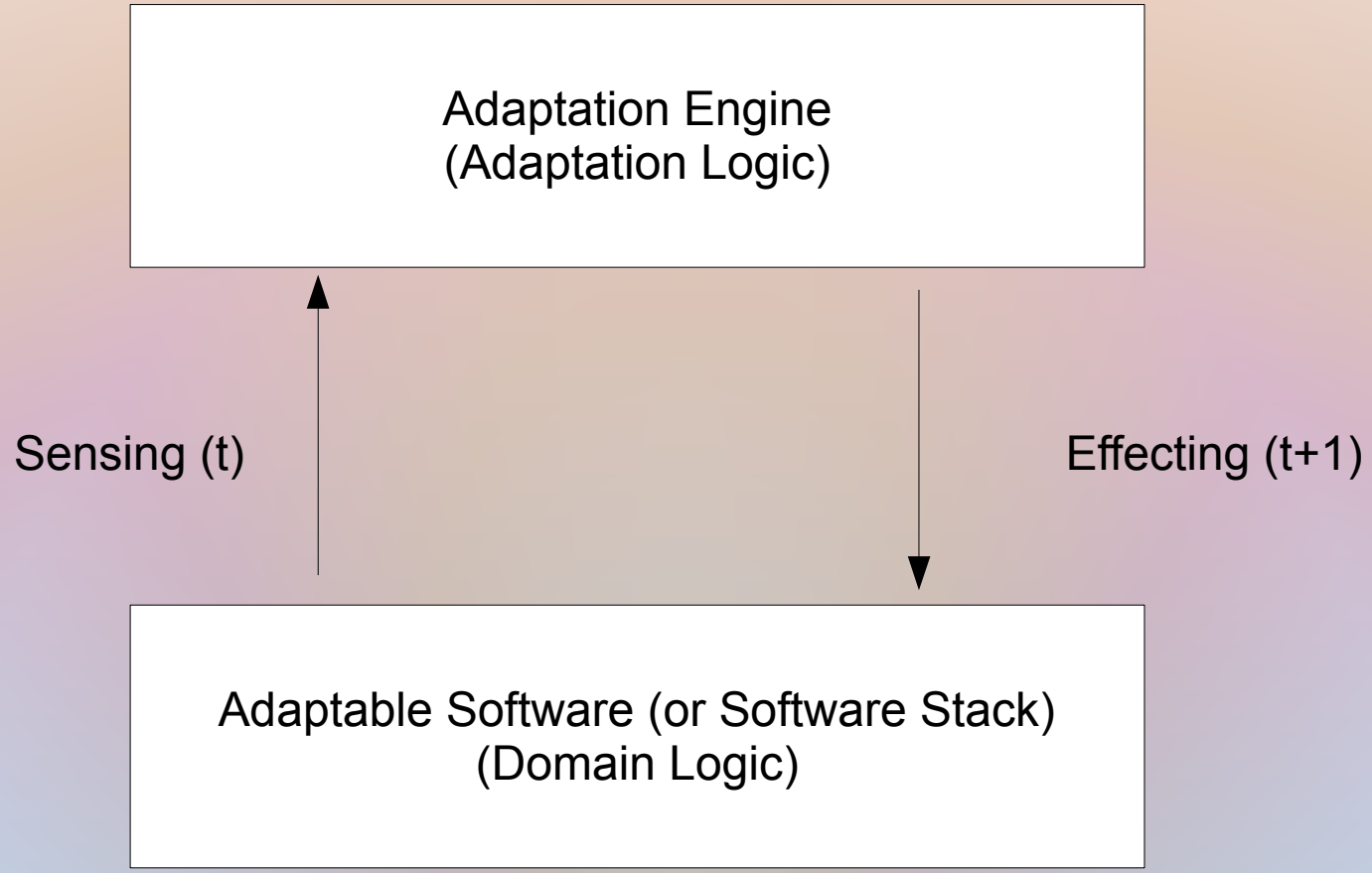
Feature Guided and Knee Driven Multi-Objective Optimization for Self-Adaptive Software at Runtime

Self-Adaptive Software

"Software that are able to modify their behaviours and/or structure in response to their perception of the environment and the system itself, and their goals"

De Lemos, Rogério, et al. Software engineering for self-adaptive systems: A second research roadmap. Springer Berlin Heidelberg, 2013

Self-Adaptive Software



Goal of Runtime Optimization in Self-Adaptive Software System

Optimizing the non-functional attributes (e.g., performance, reliability and cost) of the adaptable software, so that their requirements can be better complied.

Goal of Runtime Optimization in Self-Adaptive Software System

- Objective functions:

$$Response\ Time_k(t) = f_k(F_1(t), F_2(t), \ldots F_n(t), \delta)$$

$$Energy(t) = g(F_1(t), F_2(t), \ldots F_n(t), \delta)$$

- It is an optimization problem with certain number of objectives:

$$Min(Response\ Time(t), Energy(t))$$

Challenges

- Search-based optimization, especially EA, appear to be the neat solution to the problem, but how can we systematically and automatically translate the problem into the context of EA or MOEA?

  - How to encode the problem into chromosome representation.

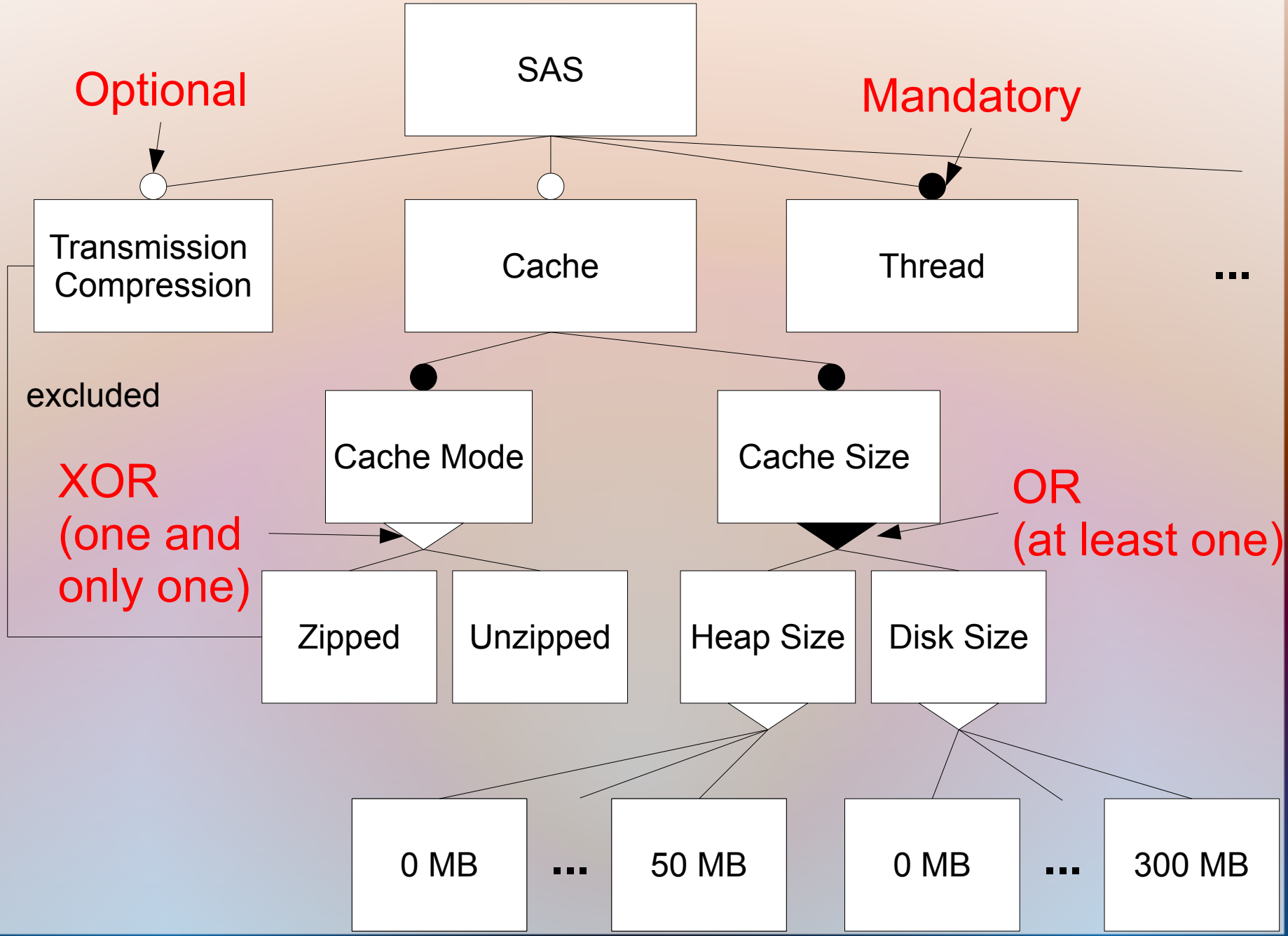
  - How to achieve better dependency handling in the optimization.

- How can we reason about trade-off between objectives and produce a single solution for adaptation at runtime?

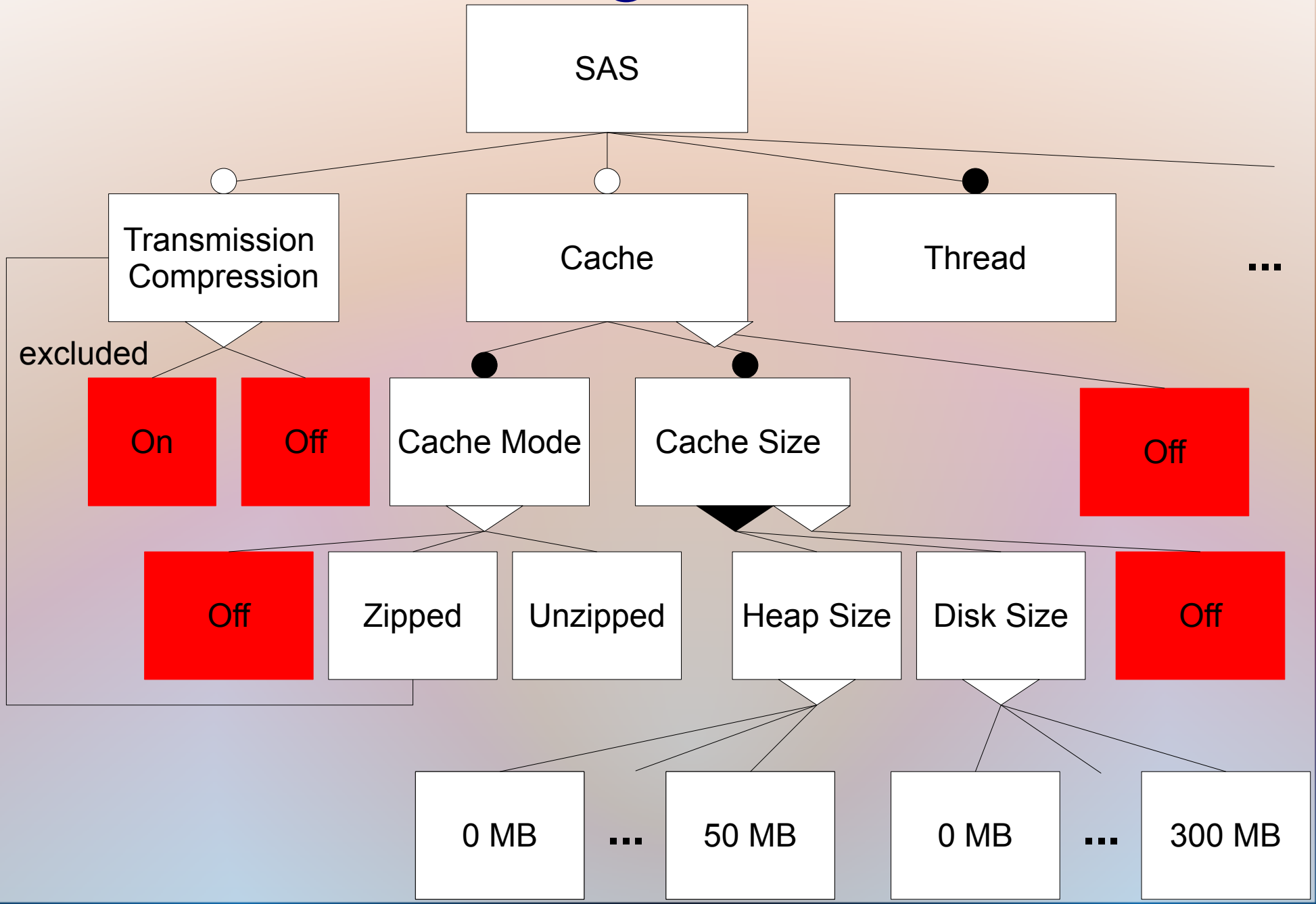
Contributions

- We leverage Feature Model to systematically represent the self-adaptive software system.
- We then automatically *translate* the model into chromosome representation, with much less number of features to express the whole variability of the self-adaptive software.
- Based on such representation, we automatically *extract* the dependencies between features out of the model and *inject* them into the mutation and crossover operators.
- Instead of using the commonly used NSGA-II for the problem, we explore the suitability of an alternative, namely MOEA/D (with Stable Matching Selection), for investigating whether some new results can be concluded.
- We aim to search for the keen solution(s) for the final adaptation decision as they are generally more preferable, especially when it is too difficult for the software engineers to specify relative importance between objectives.

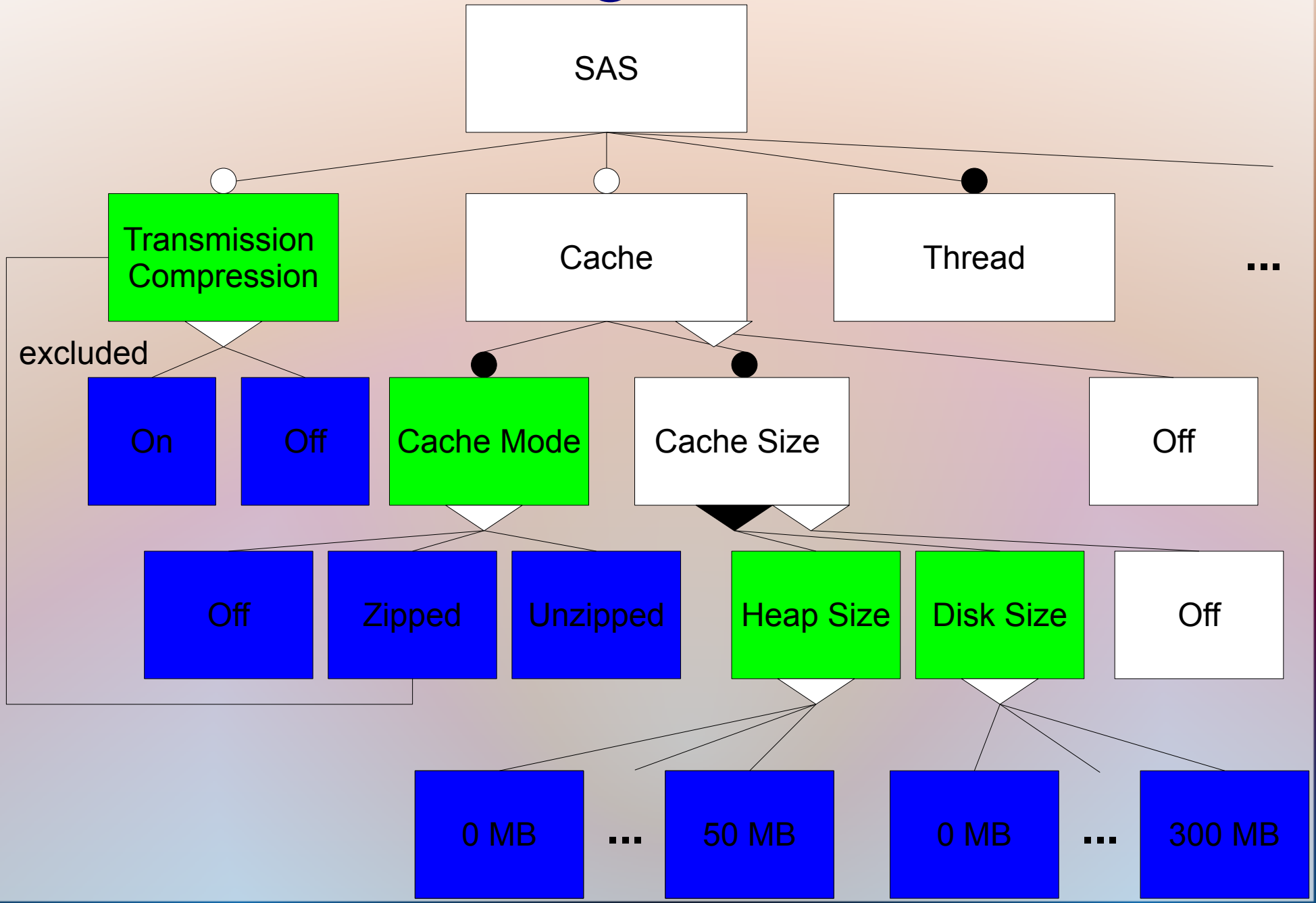
Feature Model of A SAS



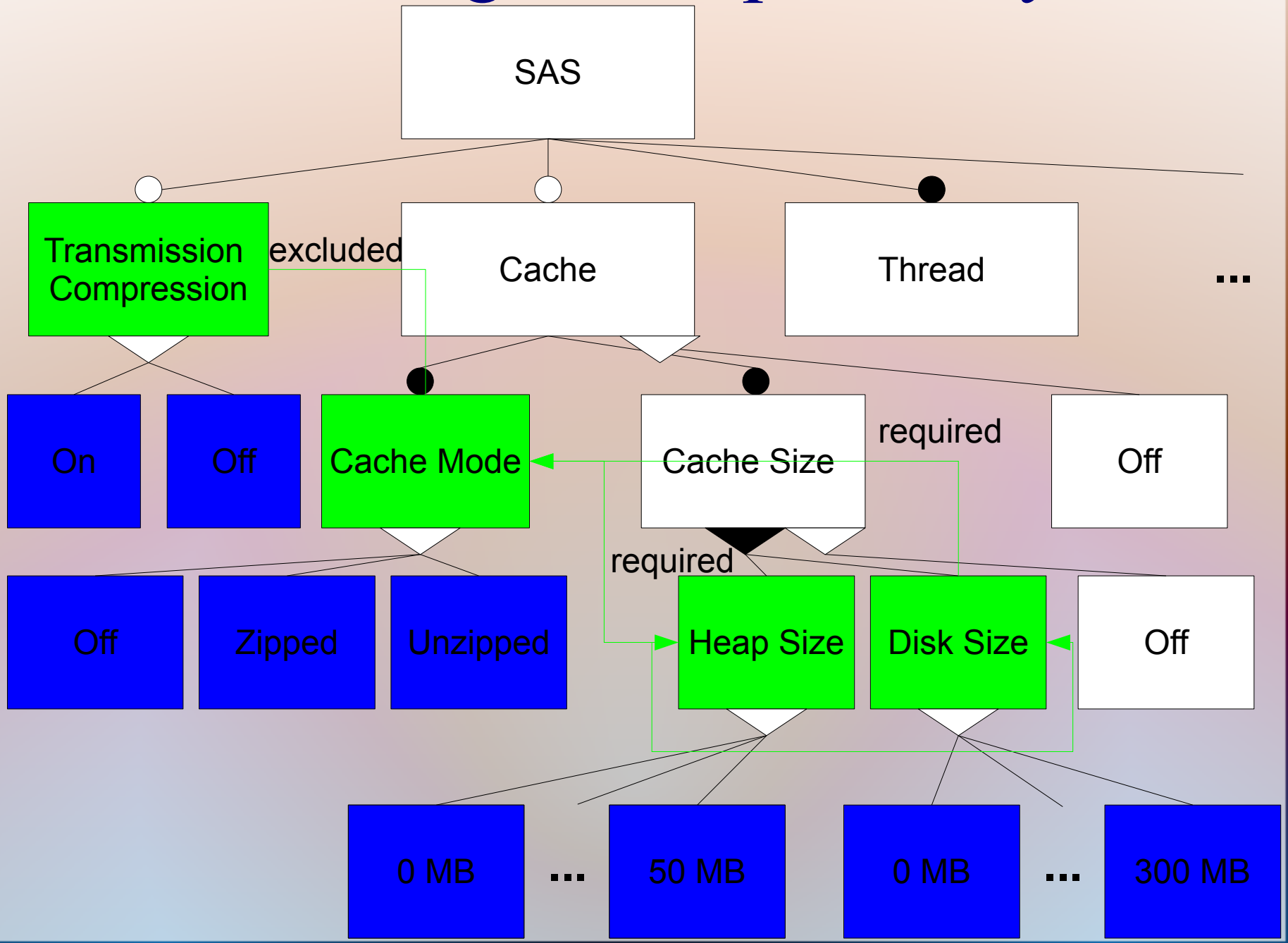
Growing the Tree



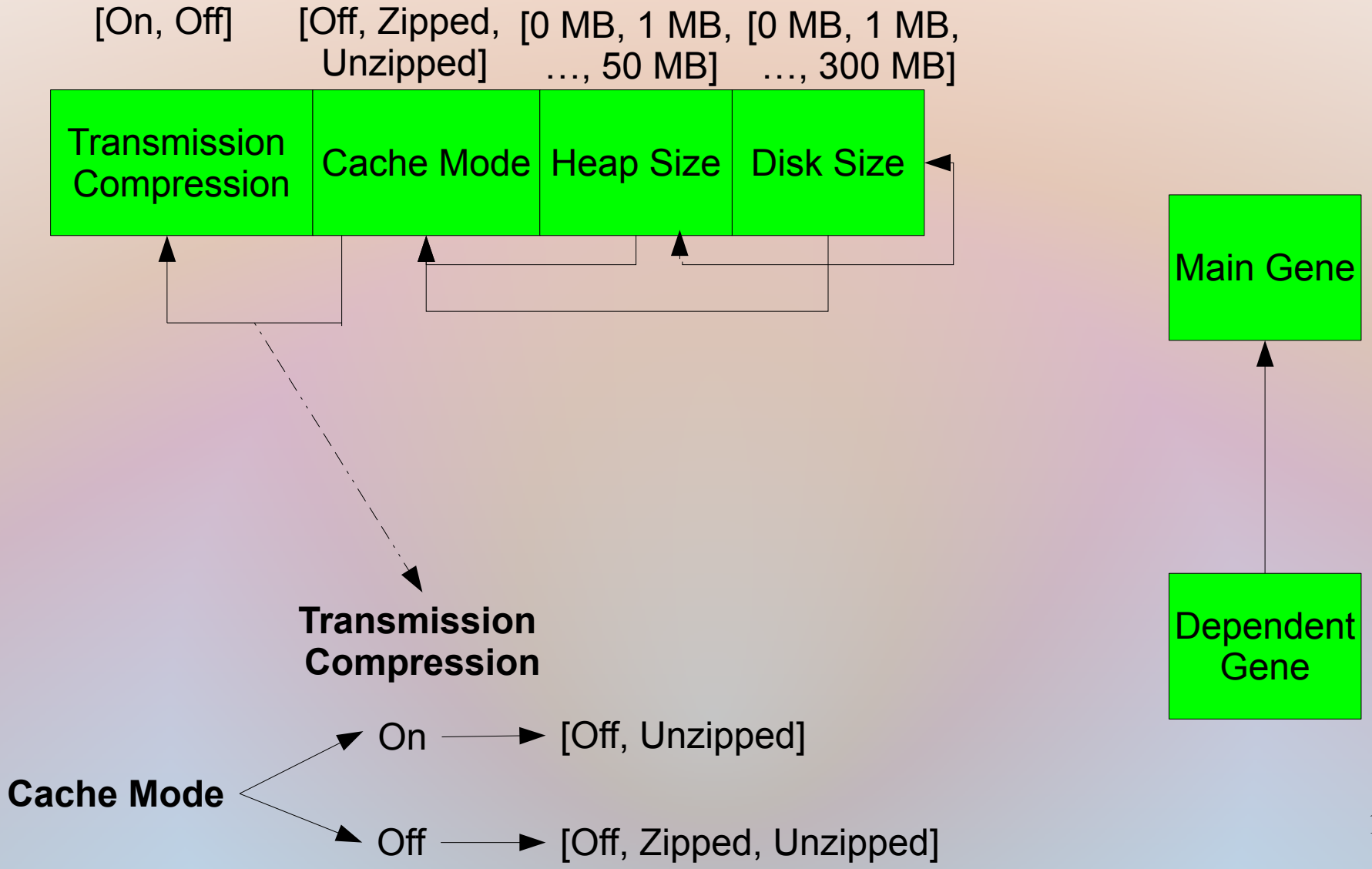
Pruning the Tree



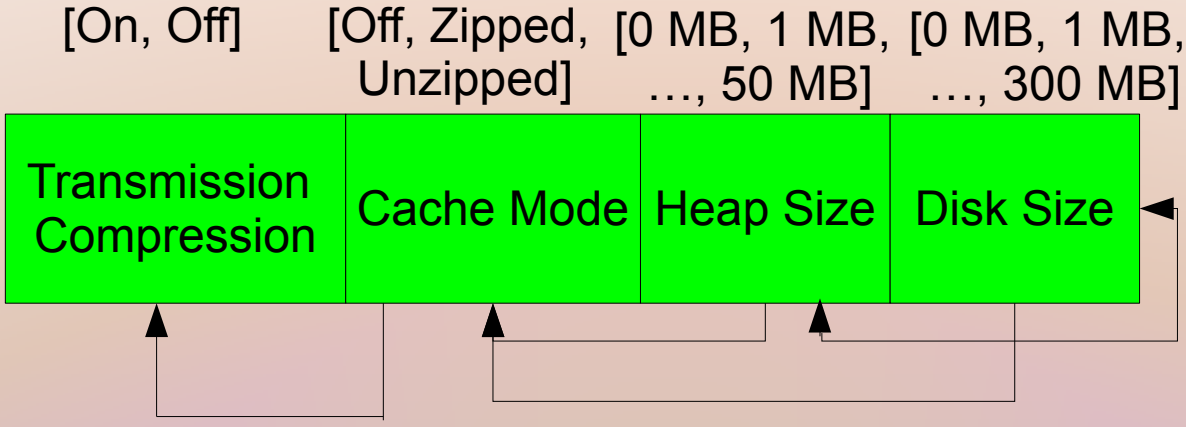
Refactoring the Dependency



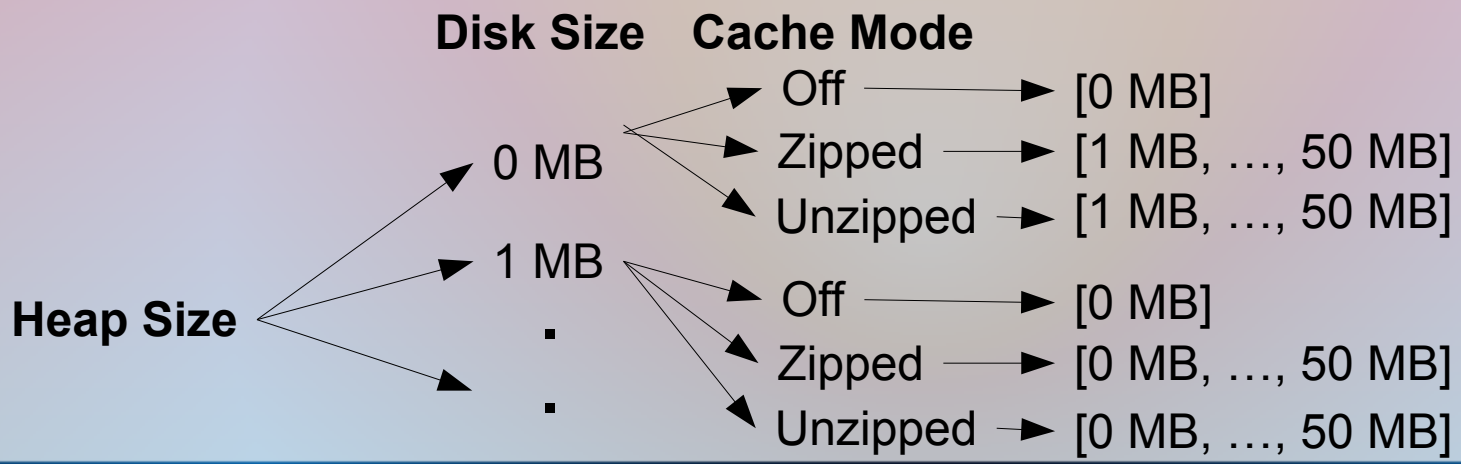
Extracting the Dependency



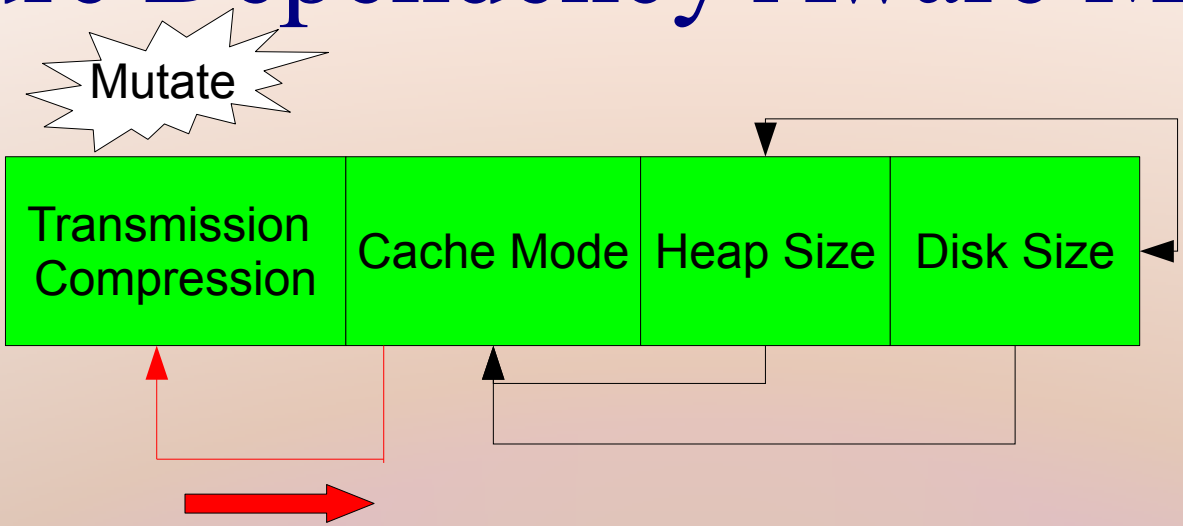
Merging the Dependency



The value tree of Heap Size = Dependency to Disk Size ∩ Dependency to Cache Mode

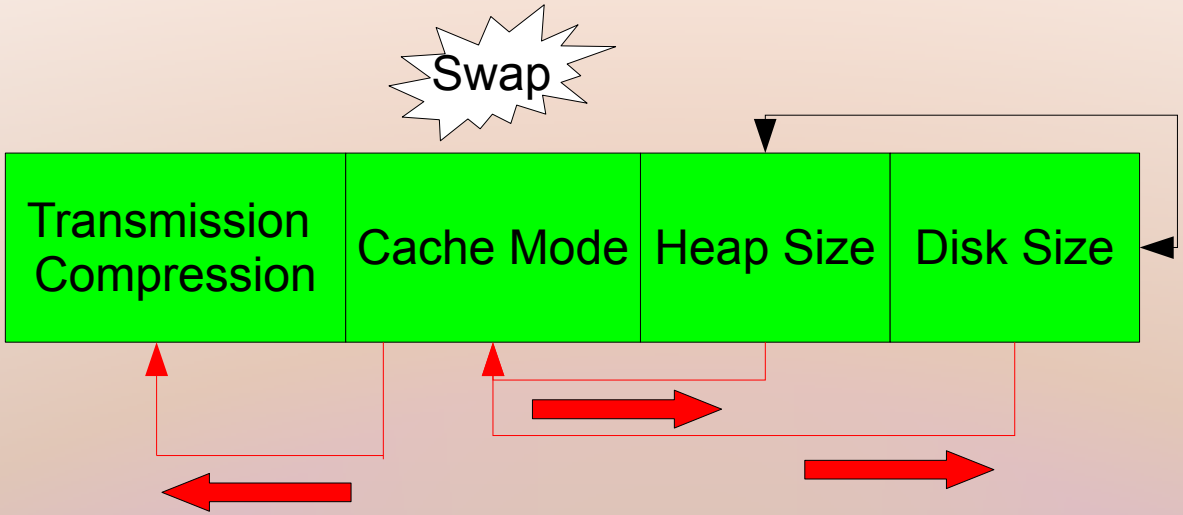


Feature Dependency Aware Mutation



- When a gene *Transmission Compression* is mutated, the value is selected using its value tree.
- It then propagates, according to the dependency chain, to any dependent genes of *Transmission Compression* for checking if their dependencies are violated. If this happen, the corresponding gene is mutated using its value tree.
- The process stops when there is no dependency violation.

Feature Dependency Aware Crossover



- When a gene *Cache Mode* is swapped, it then propagates, according to the dependency chain, to any dependent genes of *Cache Mode* for checking if their dependencies are violated, unless the corresponding gene has been swapped already. If violation found, the process swaps the gene accordingly.
- *Cache Mode* also check itself to see if it violates its own dependency to its main genes, if it does, then all the main genes of *Cache Mode* would be swapped, unless they have been swapped already.
- The process stops when there is no dependency violation or all genes have been swapped.

MOEA/D

Step 1) Initialization:

Step 1.1) Set $\mathrm{EP} = \emptyset$.

Step 1.2) Compute the Euclidean distances between any two weight vectors and then work out the T closest weight vectors to each weight vector. For each $i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.

Step 1.3) Generate an initial population $x^1, \ldots, x^N$ randomly or by a problem-specific method. Set $FV^i = F(x^i)$.

Step 1.4) Initialize $z = (z_1, \ldots, z_m)^T$ by a problem-specific method.

Step 2) Update:

For $i = 1, \ldots, N$, do

Step 2.1) Reproduction: Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.

Step 2.2) Improvement: Apply a problem-specific repair/ improvement heuristic on y to produce y'.

Step 2.3) Update of z: For each $j = 1, \ldots, m$, if $z_j < f_j(y')$, then set $z_j = f_j(y')$.

Step 2.4) Update of Neighboring Solutions: For each index $j \in B(i)$, if $g^{te}(y'|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z)$, then set $x^j = y'$ and $FV^j = F(y')$.

Step 2.5) Update of EP:

Remove from EP all the vectors dominated by $F(y')$.

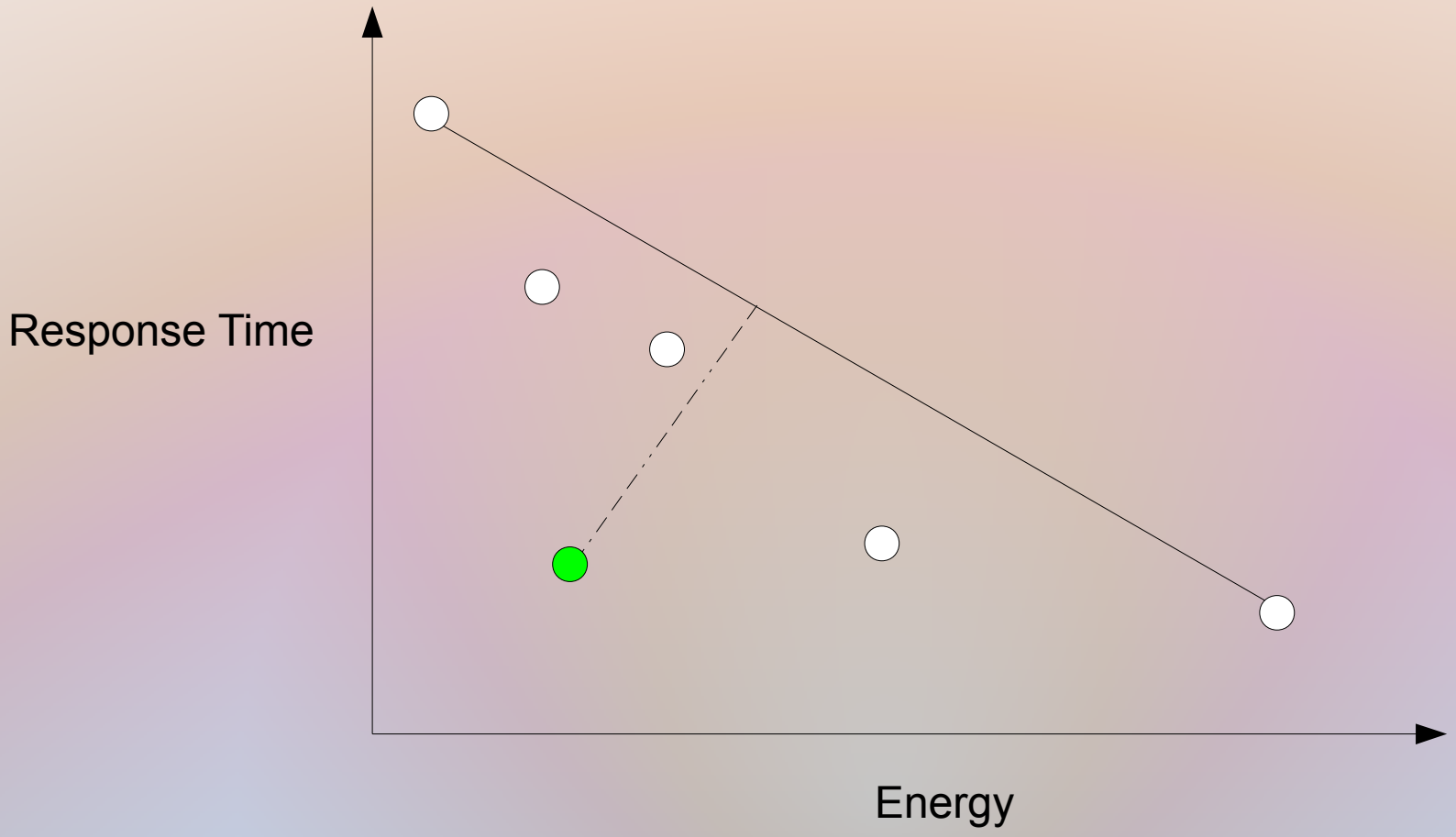
Add $F(y')$ to EP if no vectors in EP dominate $F(y')$.

Step 3) Stopping Criteria: If stopping criteria is satisfied, then stop and output EP. Otherwise, go to **Step 2**.

Inject feature dependency aware mutation and crossover operators into Step 2.1 and 2.2

Using a Stable-matching based selection in Step 2.5

Knee Solution for Adaptation



Progress So Far

- Finalising the experiment results.
- Writing a paper for ICSE 2017 (deadline by the end of Aug 2016).