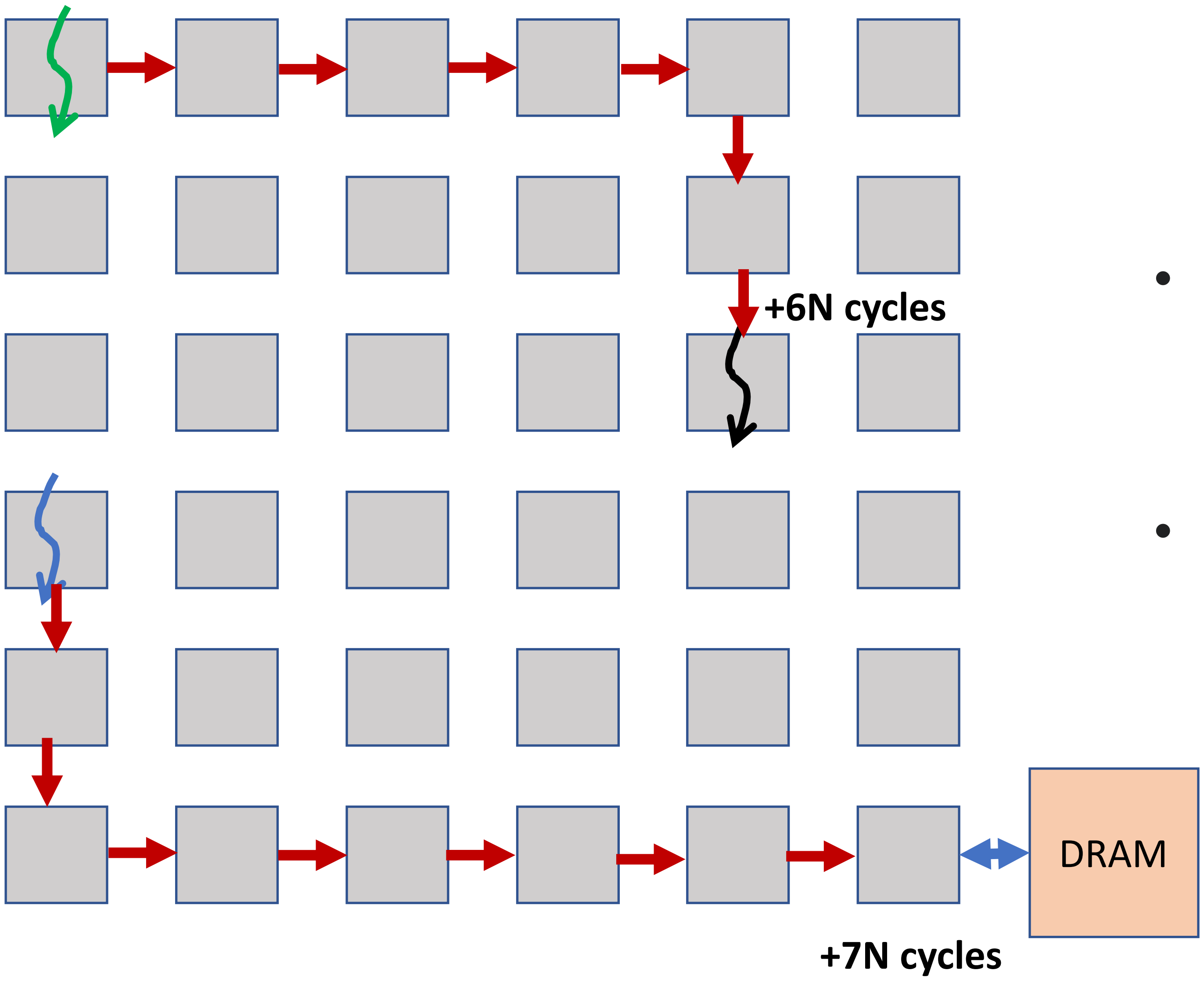
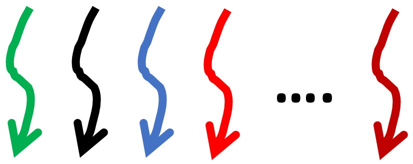
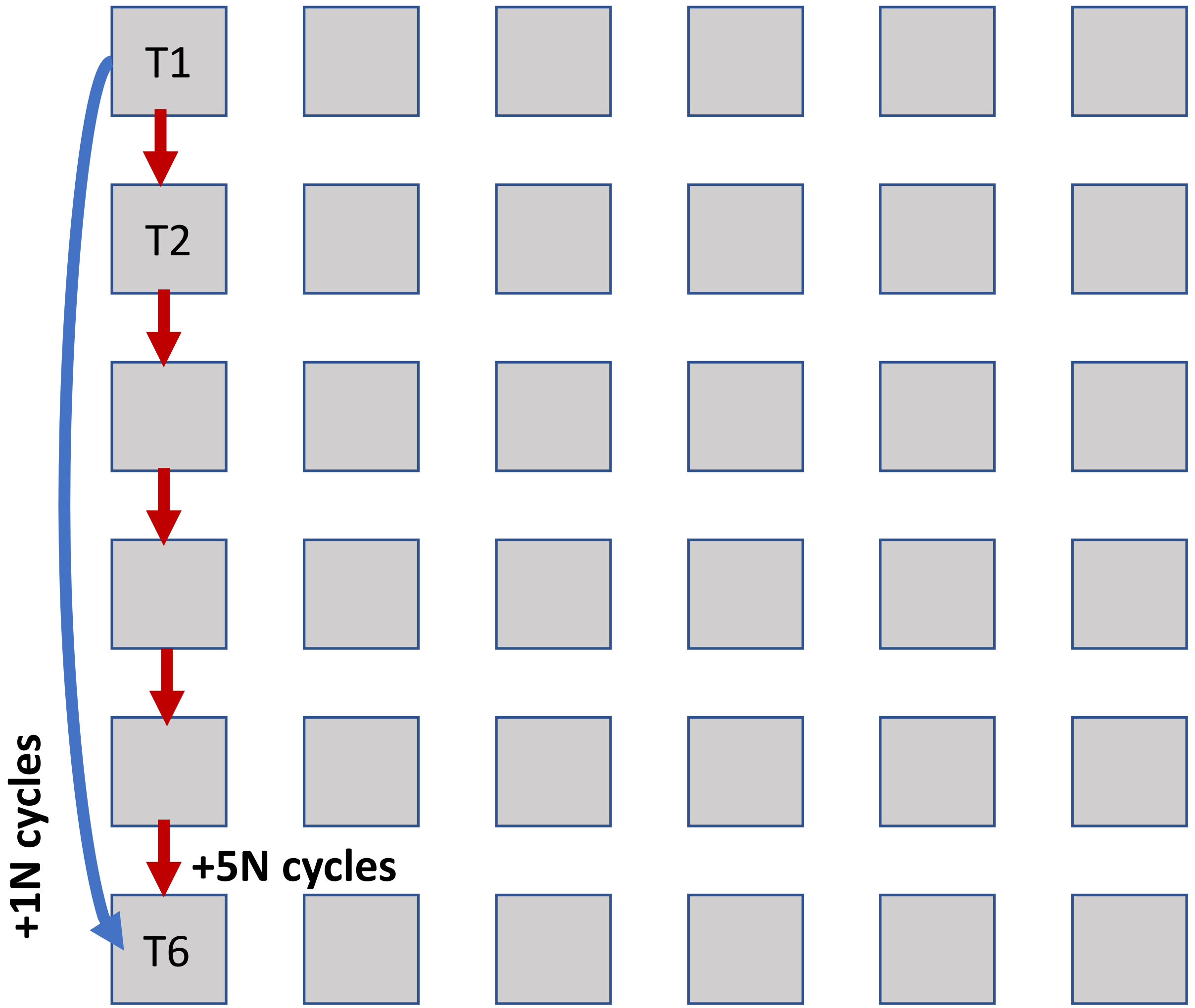
Mapping in manycore





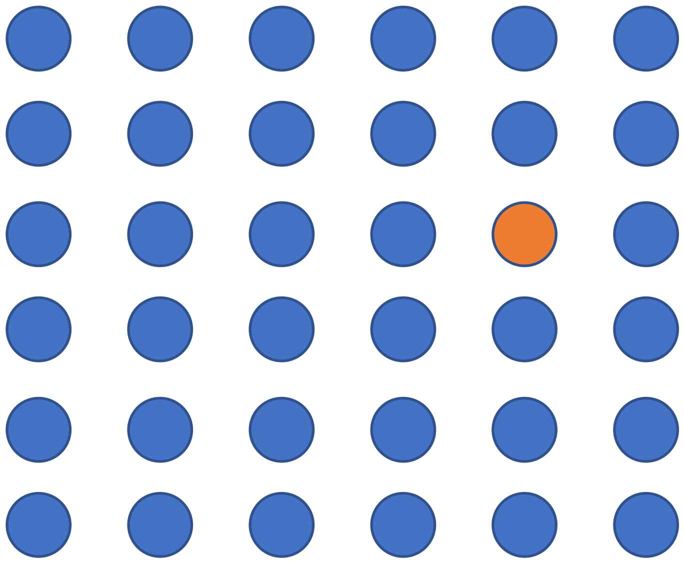
- N tasks → N cores
  - Communication
    - Performance/Execution time
  - Thermal
- N! (N factorial) cases
  - Impossible to explore all possible solutions
  - How to map these tasks?

NoC Design in Manycore



- Application behavior known
  - Case-1: T1 only communicates with T2
  - Case-2: T1 only communicates with T6
- Application-specific NoC design
  - Given L links, N cores
    - C(C(N,2),L) possible ways
  - How to place these links?

Design Space Exploration



- **"Design Space Exploration** (DSE) refers to systematic analysis and pruning of unwanted design points based on parameters of interest." -Wikipedia
- **M possible designs, N valid designs**
  - **M >> N**
- **Single-objective or Multi-objective**

Single-objective optimization

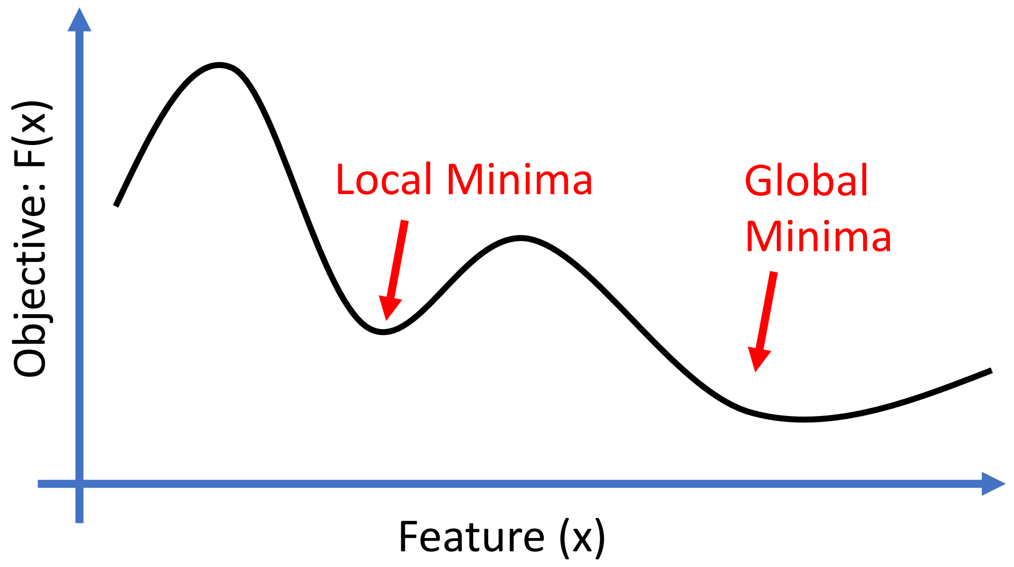
- **Single-objective: Price**
  - **Objective = Price**
  - **Lowest price is the solution**

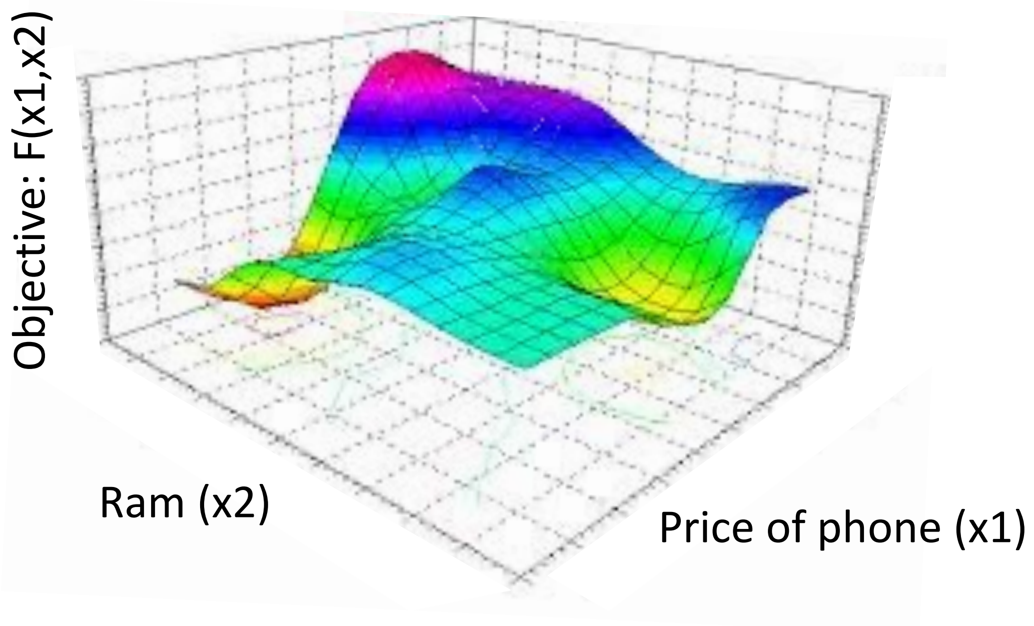
Minima finding problem!

Multi-objective optimization

- **Multi-objective: Price & Ram**
  - **Define custom objective function**
    - **Objective = Price – Ram**
  - **Other functions commonly used**
    - **Weighted sum**
    - **Pareto-Hyper volume, etc**

Finding the minima



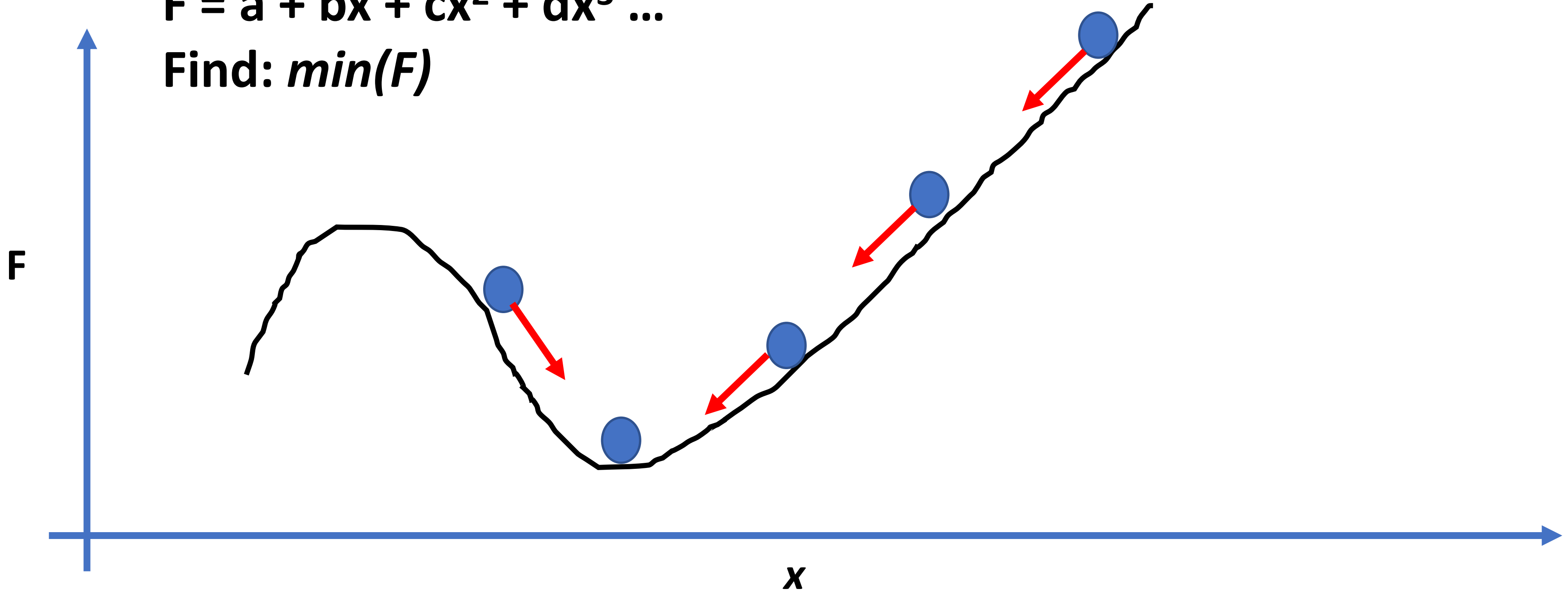


- **Exhaustive exploration**
  - **Guaranteed best solution**
  - **Computing is time consuming**

Solving mathematically

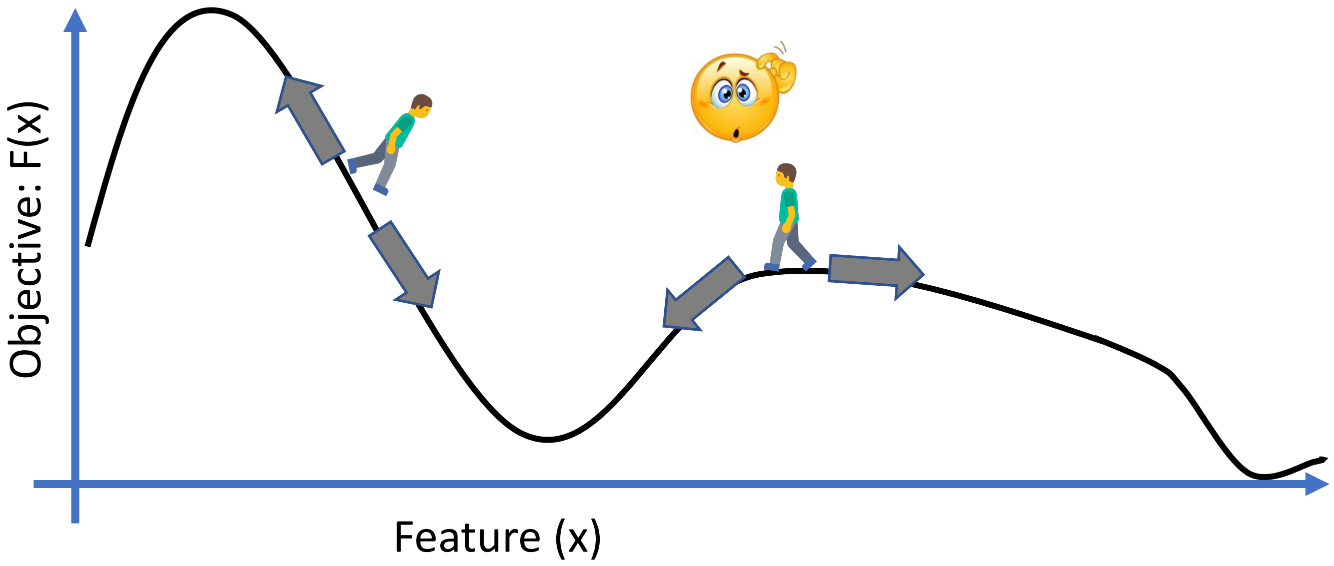
$F = a + bx + cx^2 + dx^3 \ldots$

Find: *min(F)*



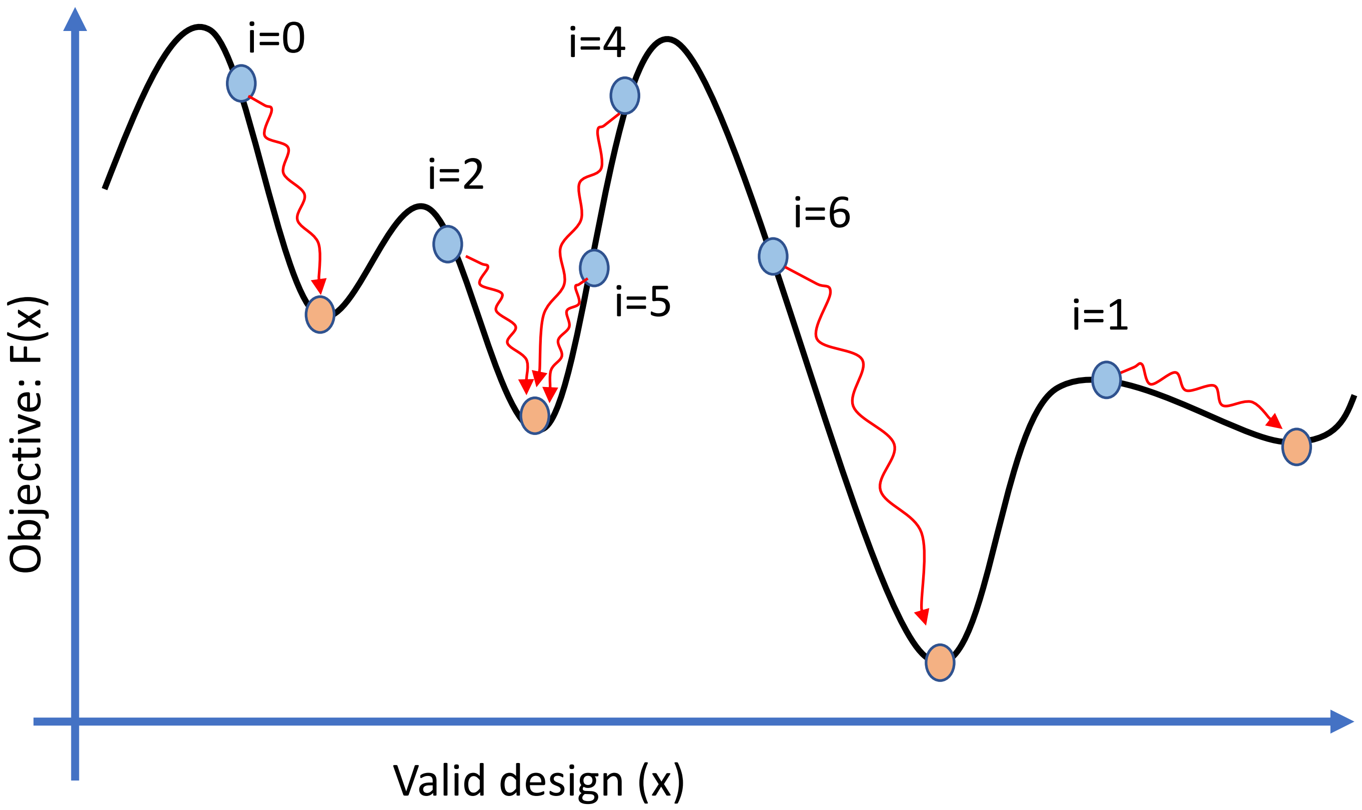
- To find minima, we can use differentiate F:
  - At minima, $\frac{dF}{dx} = 0$
  - Hard to solve
- You may not know F, Solving $\frac{dF}{dx} = 0$ is non-trivial

Hill climbing algorithm



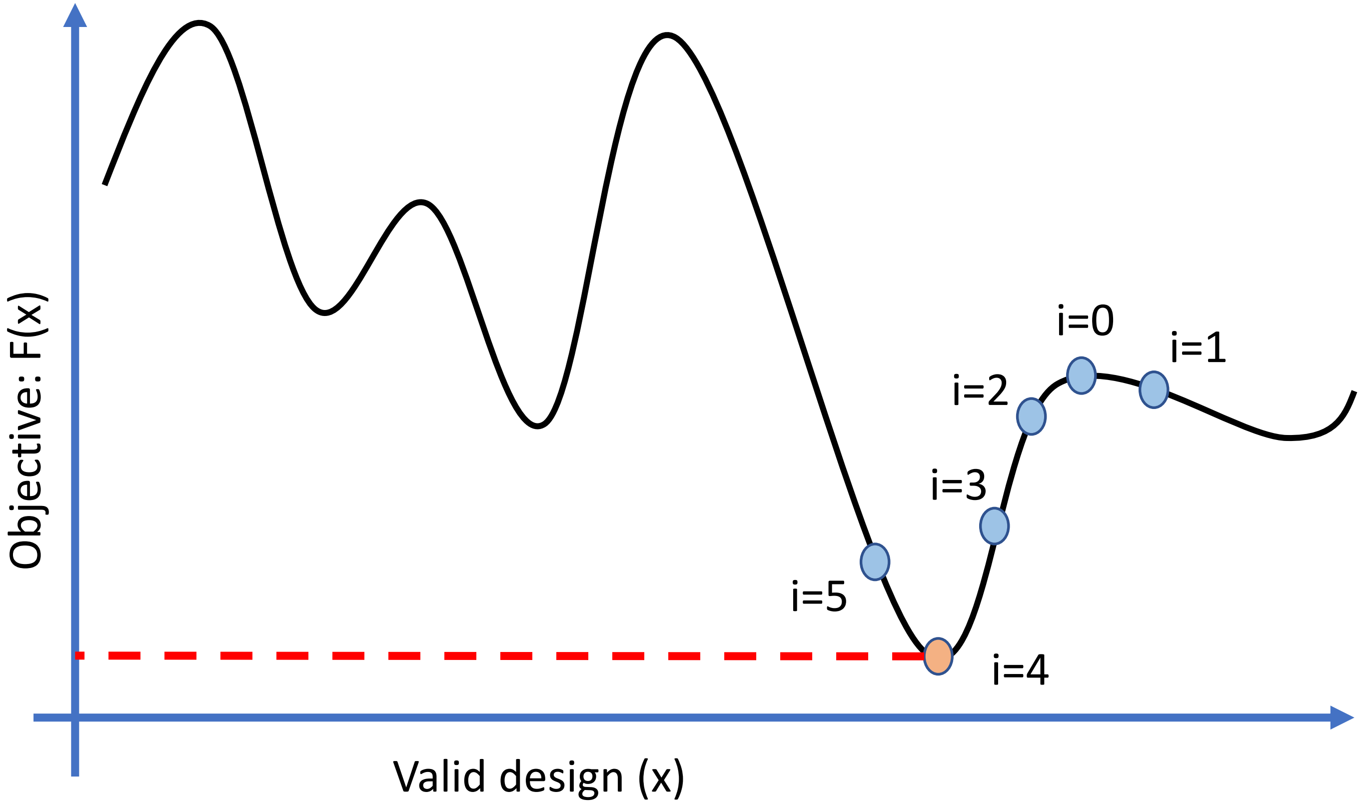
- **Greedy algorithm**
  - **Move towards the most promising direction**
  - **Does not guarantee best solution**
  - **Tends to get stuck at local minima**
- **Gives us "good" solution in short time**
  - **Best solution ⊆ Good solution**

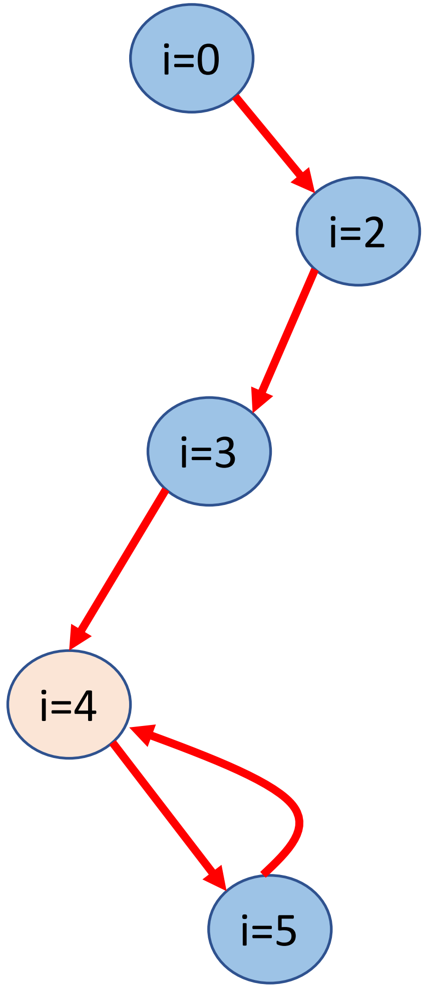
Hill climbing with repeated restarts



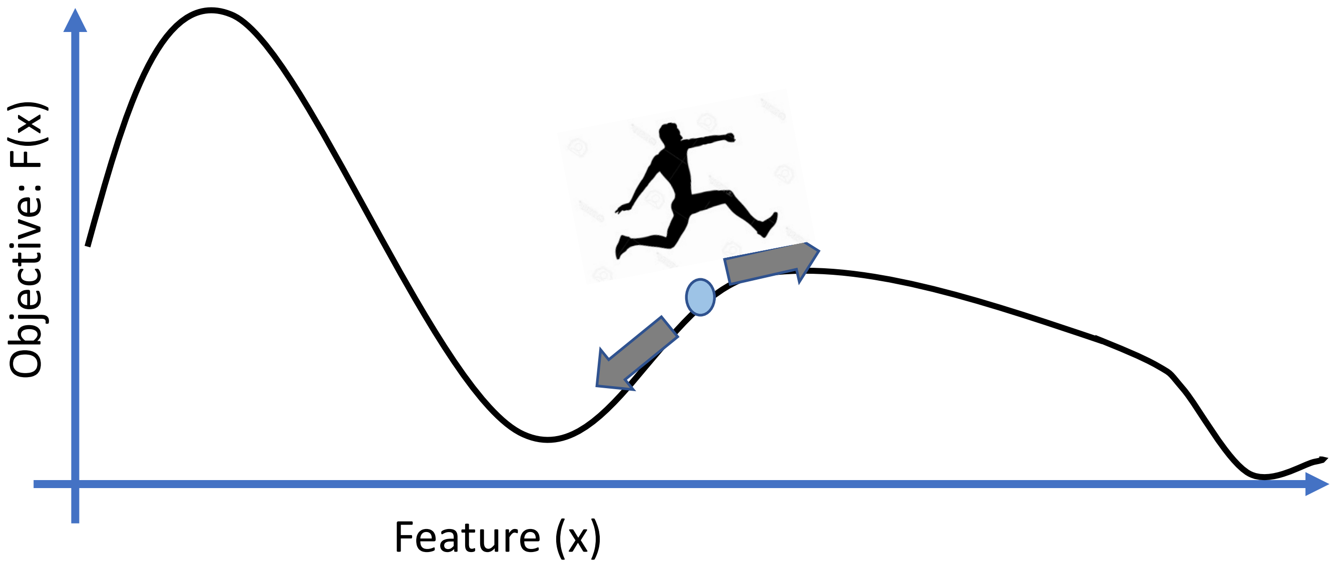
- **Try, try and try again**
- **Start from random locations**
  - **Will end at different minima**
- **Better chance to find global minima**
  - **Time consuming**
  - **Needs some Luck**

Hill climbing in a graph way



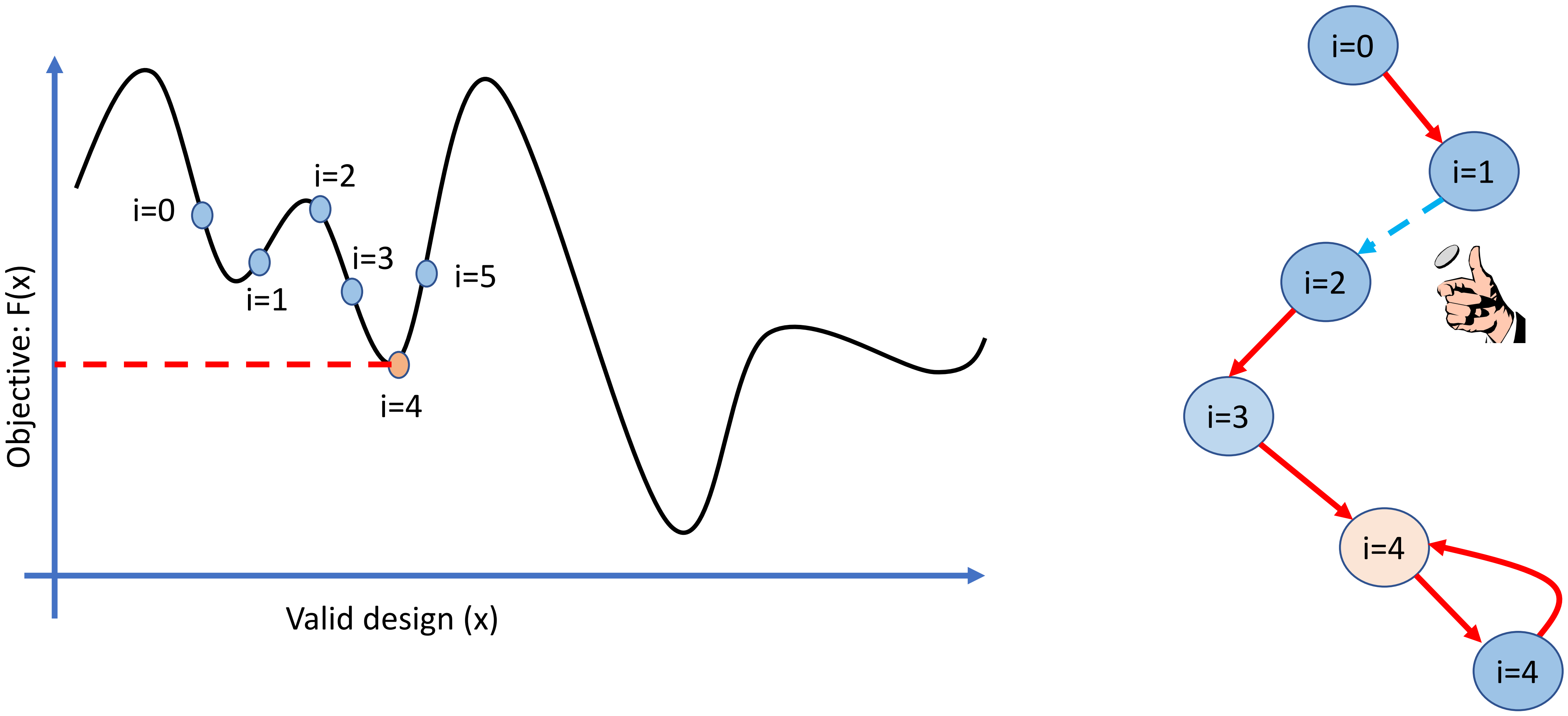


Simulated Annealing algorithm

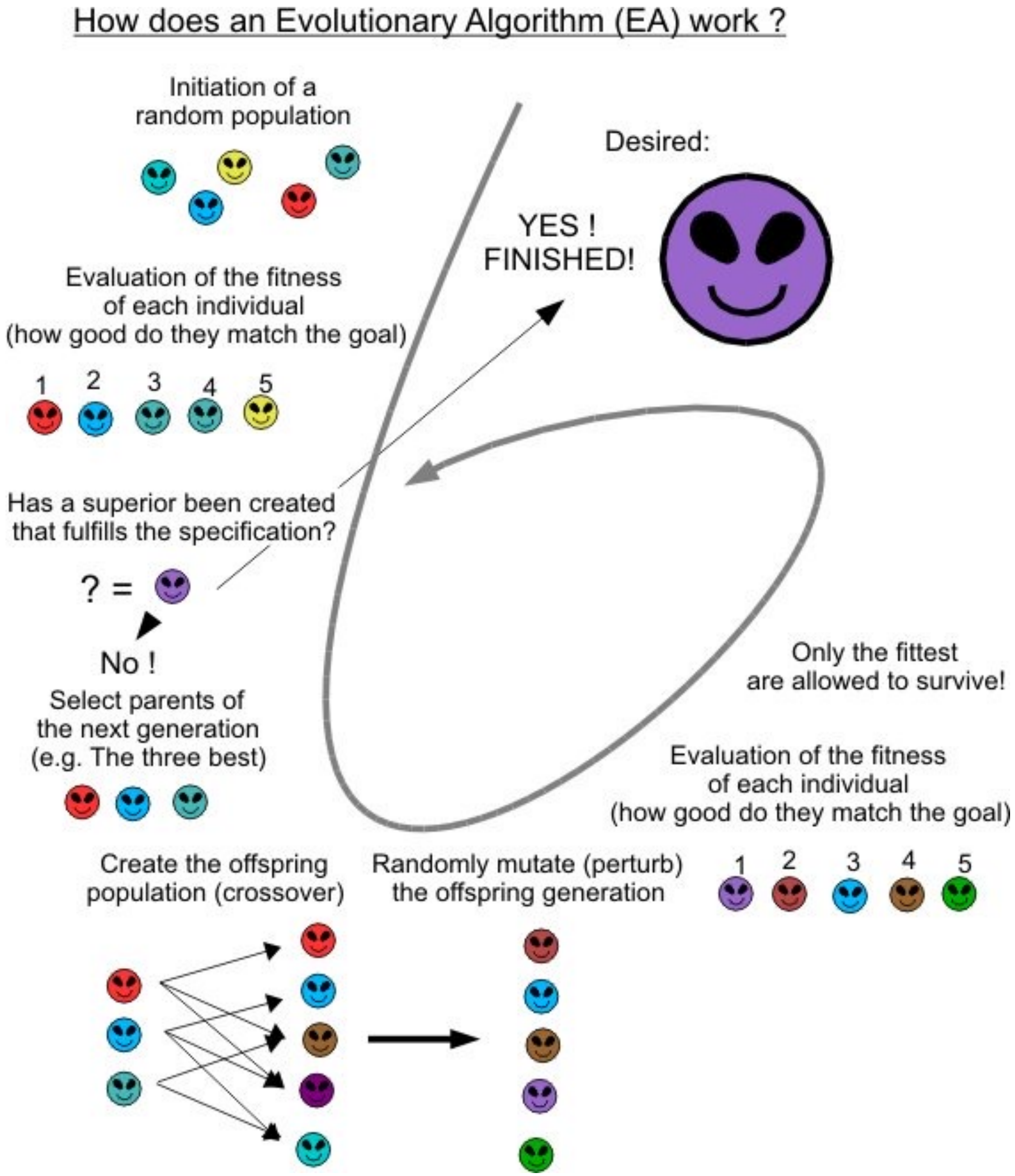


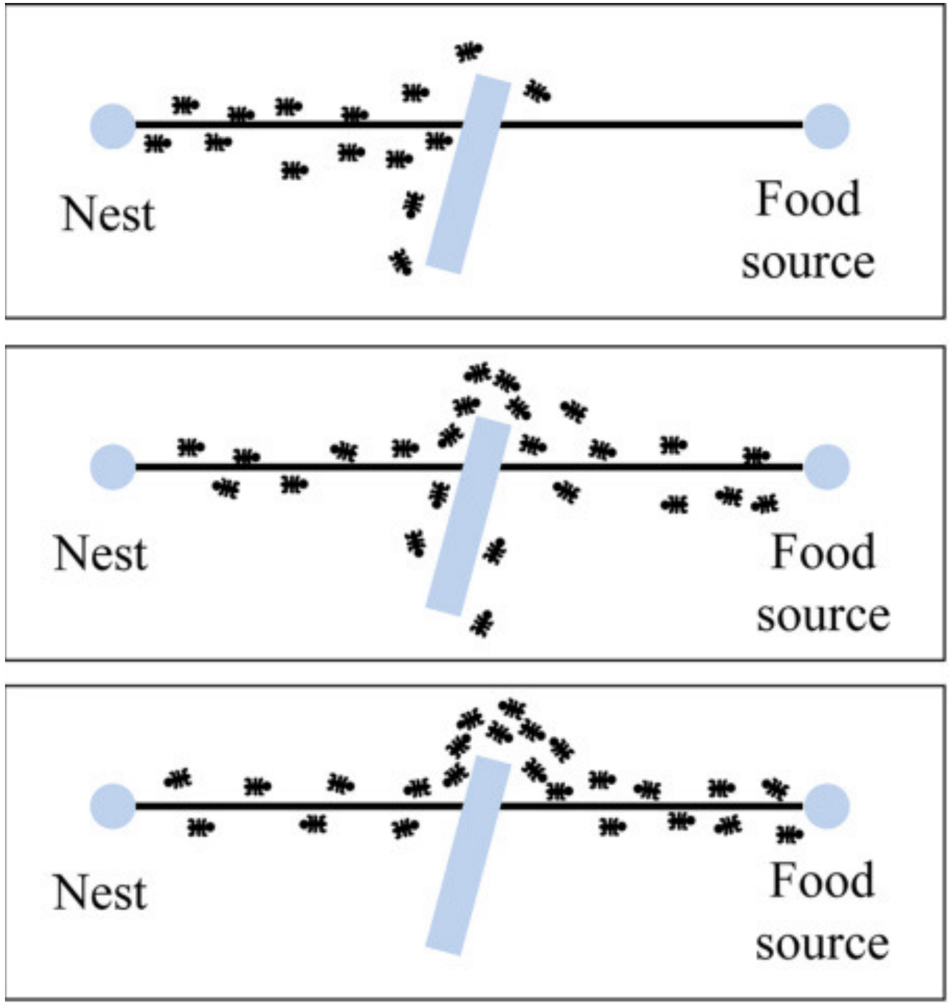
- Make some mistakes
- Willing to accept bad results at first
- Converges to greedy later
- Explore vs exploit
- Works better than greedy algorithm in practice

Simulated Annealing in a graph way



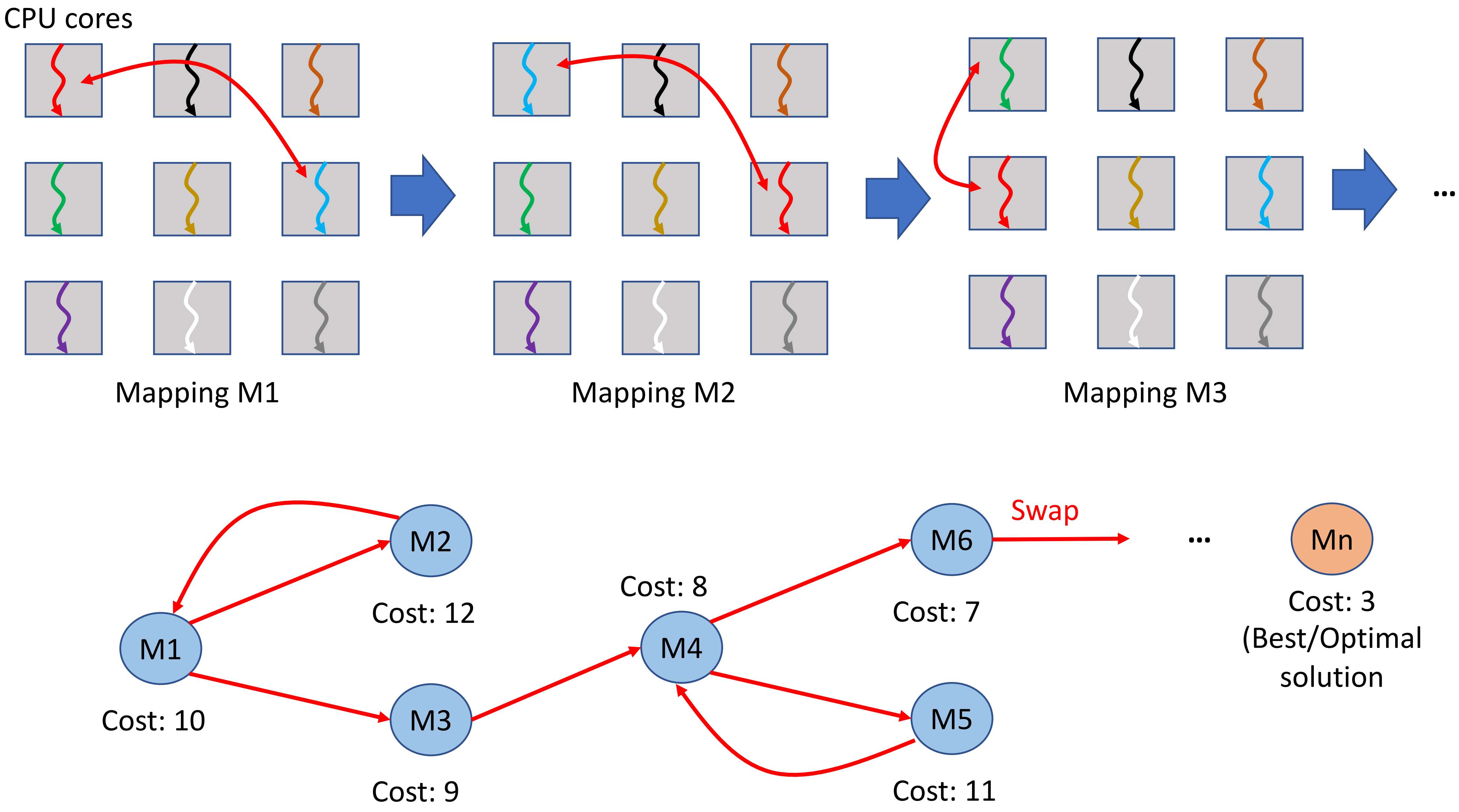
Other algorithms for optimization





Ant colony optimization

Task mapping using Hill climbing

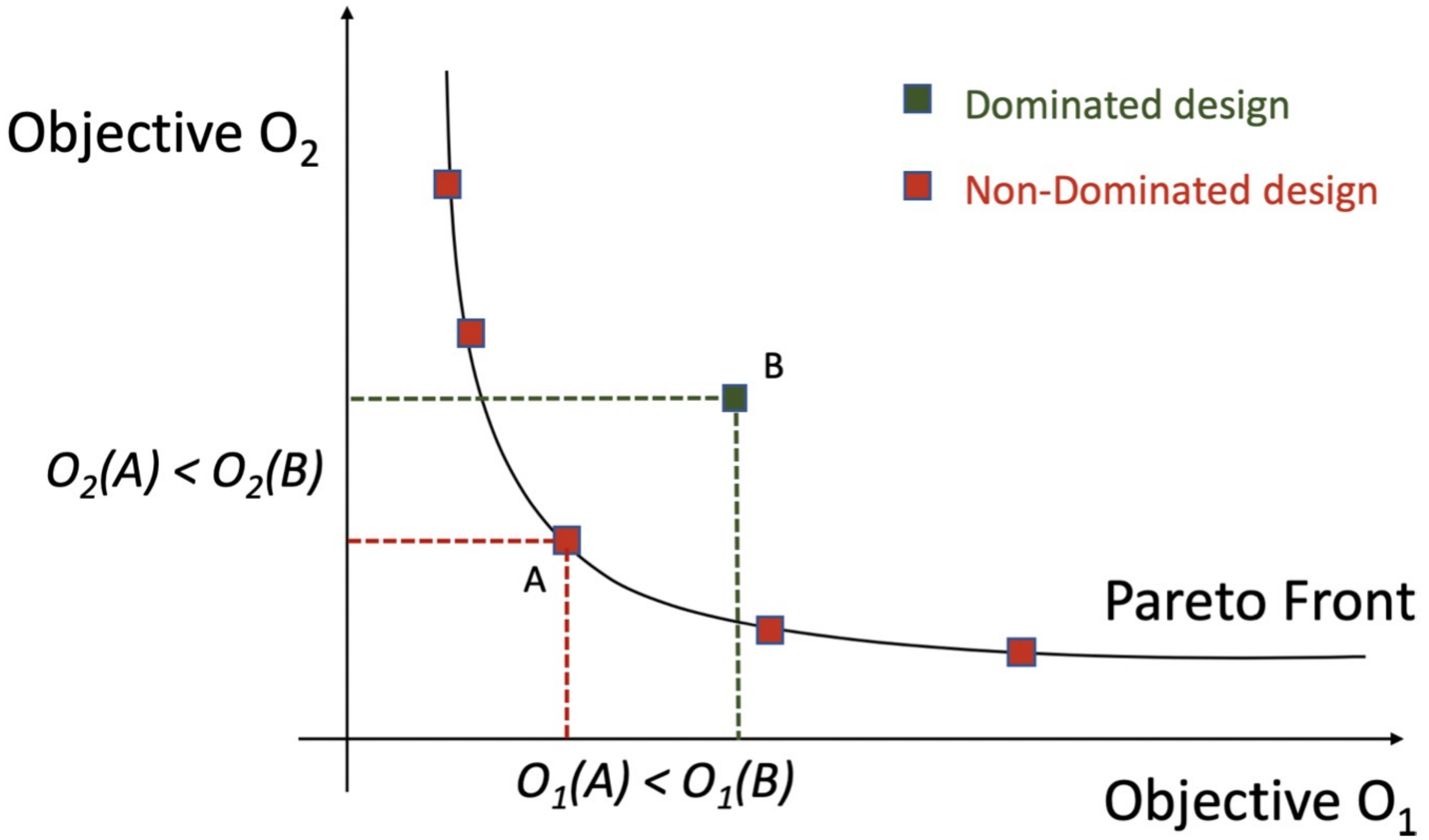


Evaluating a mapping/design

- **GPU objective: Maximize throughput**
  - **Throughput (load balancing)**
    - $\bar{U} = \frac{1}{L}\sum_{k=1}^{L}\left(\sum_{i=1}^{R}\sum_{j=1}^{R} f_{ij} \cdot p_{ijk}\right)$ (Average link utilization)
    - $\sigma = \sqrt{\frac{1}{L}\sum_{k=1}^{L}(U_k - \bar{U})^2}$ (Std. Dev. of link utilization)
- **CPU objective: Low latency**
  - **Latency**
    - Lat = $\frac{1}{C*M}\sum_{i=1}^{C}\sum_{j=1}^{M}\left(r \cdot h_{ij} + d_{ij}\right) \cdot f_{ij}$ (Zero load latency)
- **3D-specific objective: Temperature**
  - **Max. on-chip temperature**
    - $T = \max_{n,k}\left\{\sum_{i=1}^{k}\left(P_{n,i}(t)\sum_{j=1}^{i} R_j\right) + R_b \sum_{i=1}^{k} P_{n,i}(t)\right\} * T_H$

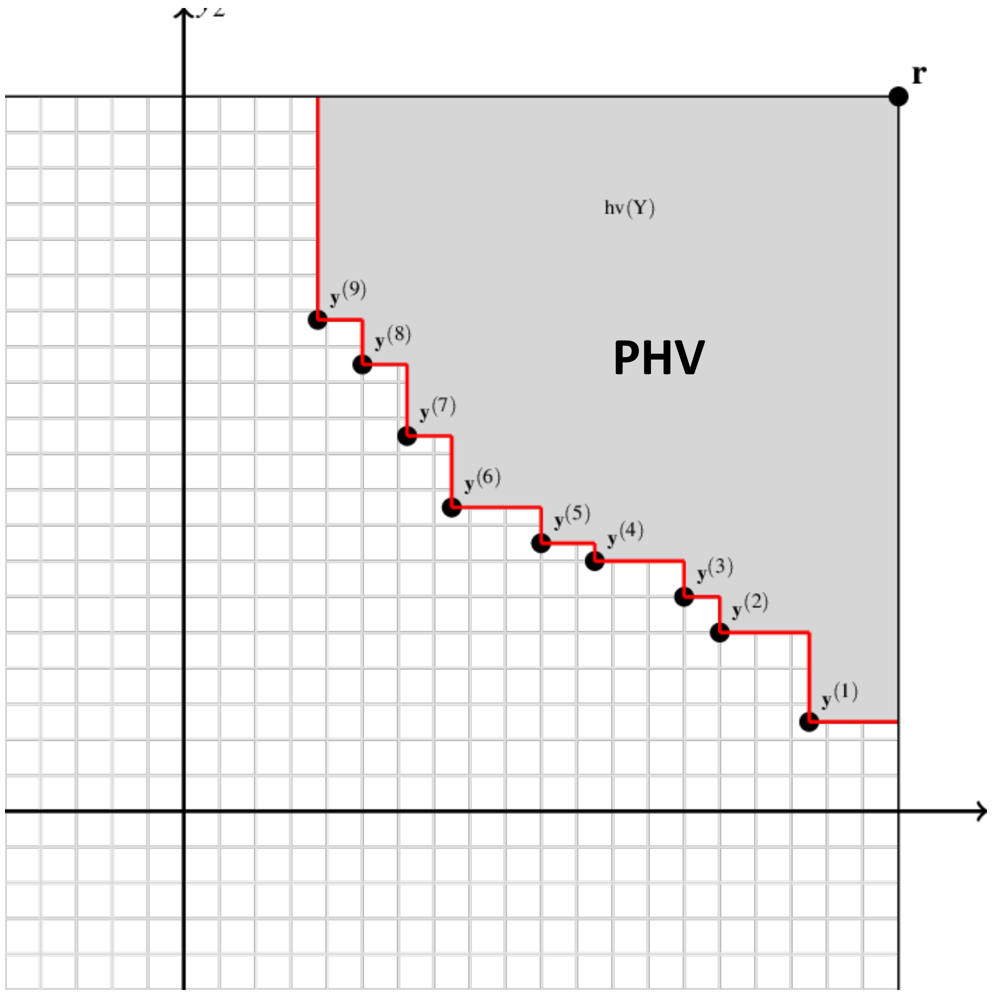
[1] **B. K. Joardar**, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu and R. Marculescu, "Learning-Based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 852-866, 1 June 2019

Pareto dominance for MOO

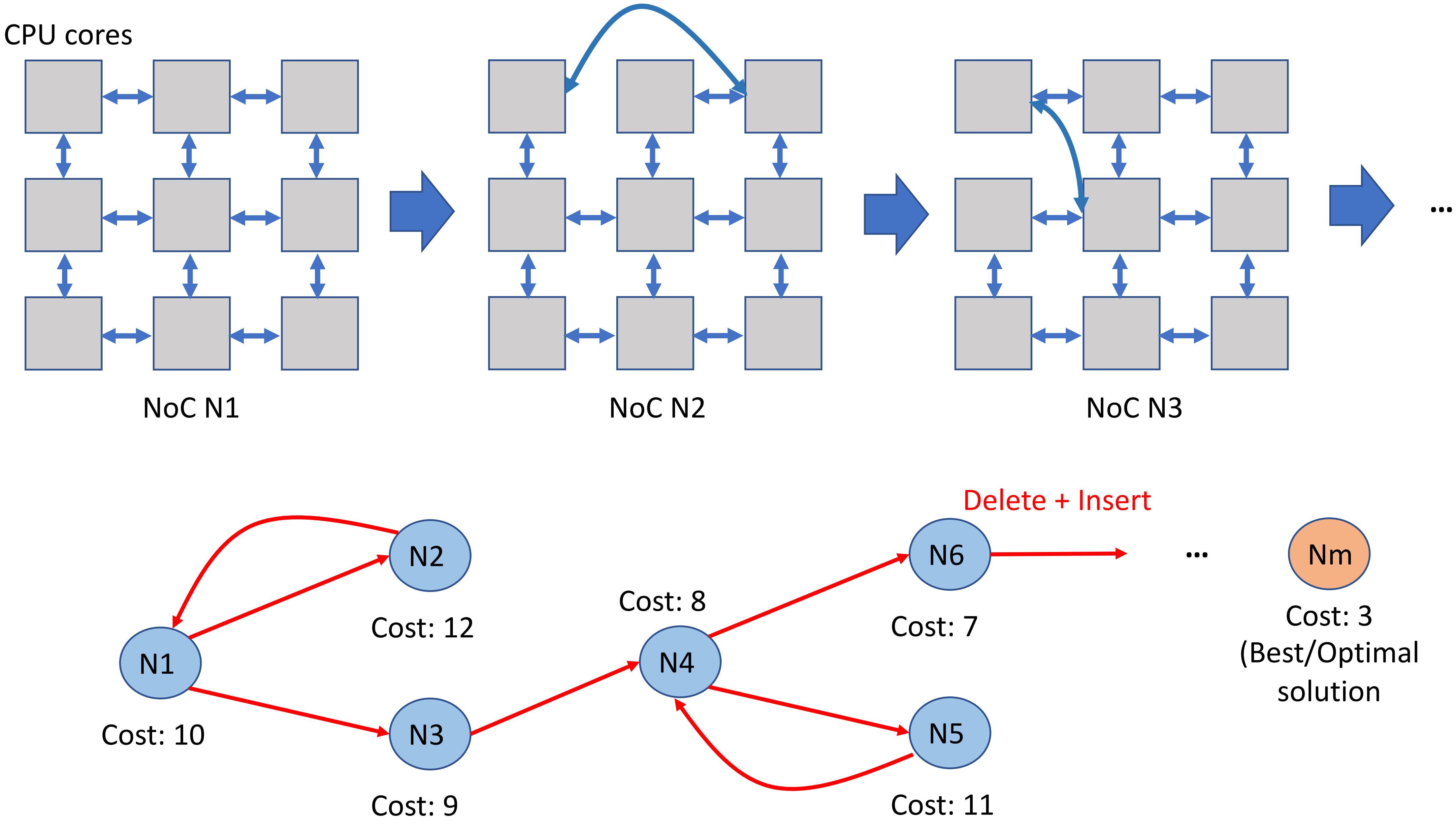


Cost Functions

- **Additive:**
  - **Y=O1+O2**
- **Weighted sum**
  - **Y=a1*O1 + a2*O2**
- **Multiplicative**
  - **Y=O1*O2**
- **Pareto Hyper Volume (PHV)**
  - **Area covered by non-dominated designs**
- **Weighted PHV**
  - **Weighted area covered by non-dominated designs**

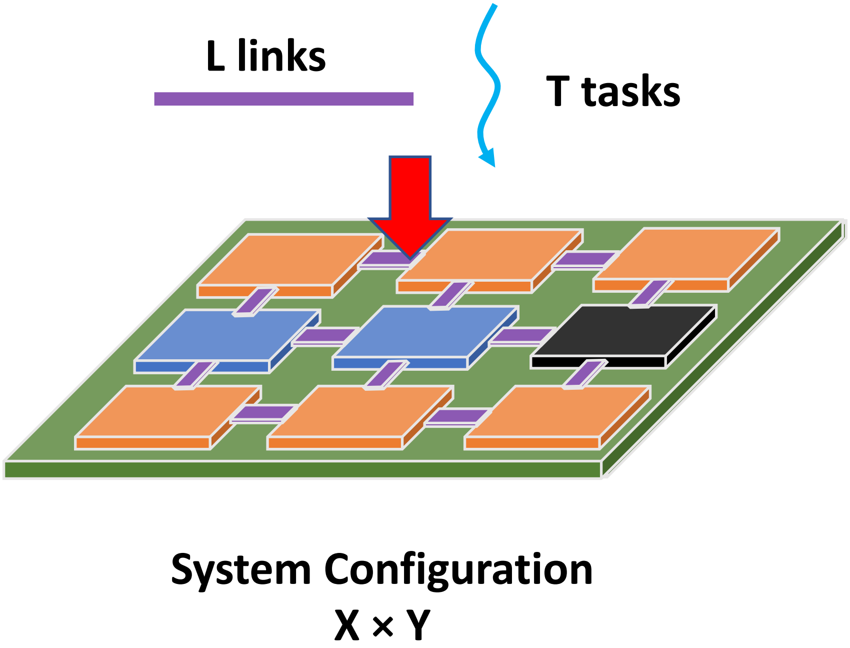


NoC design using Hill climbing

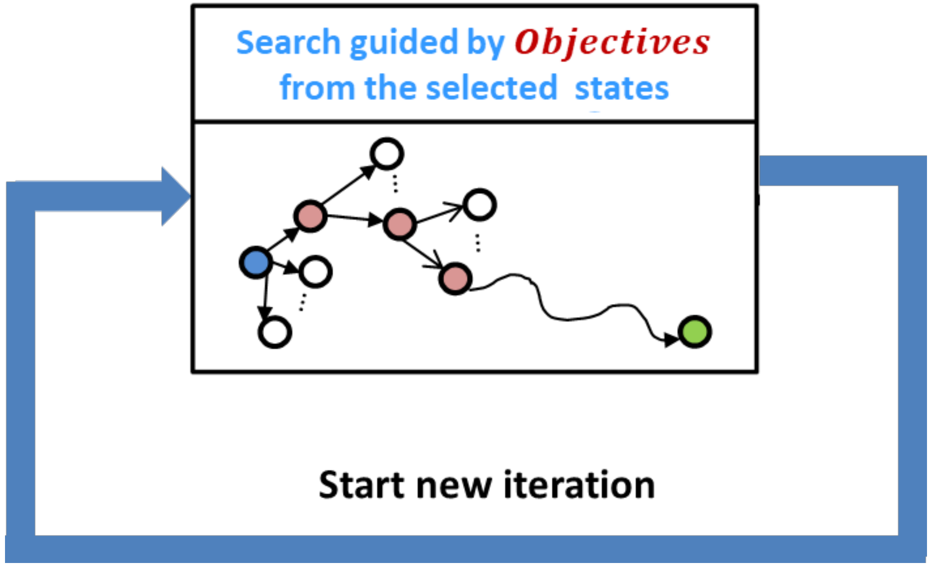


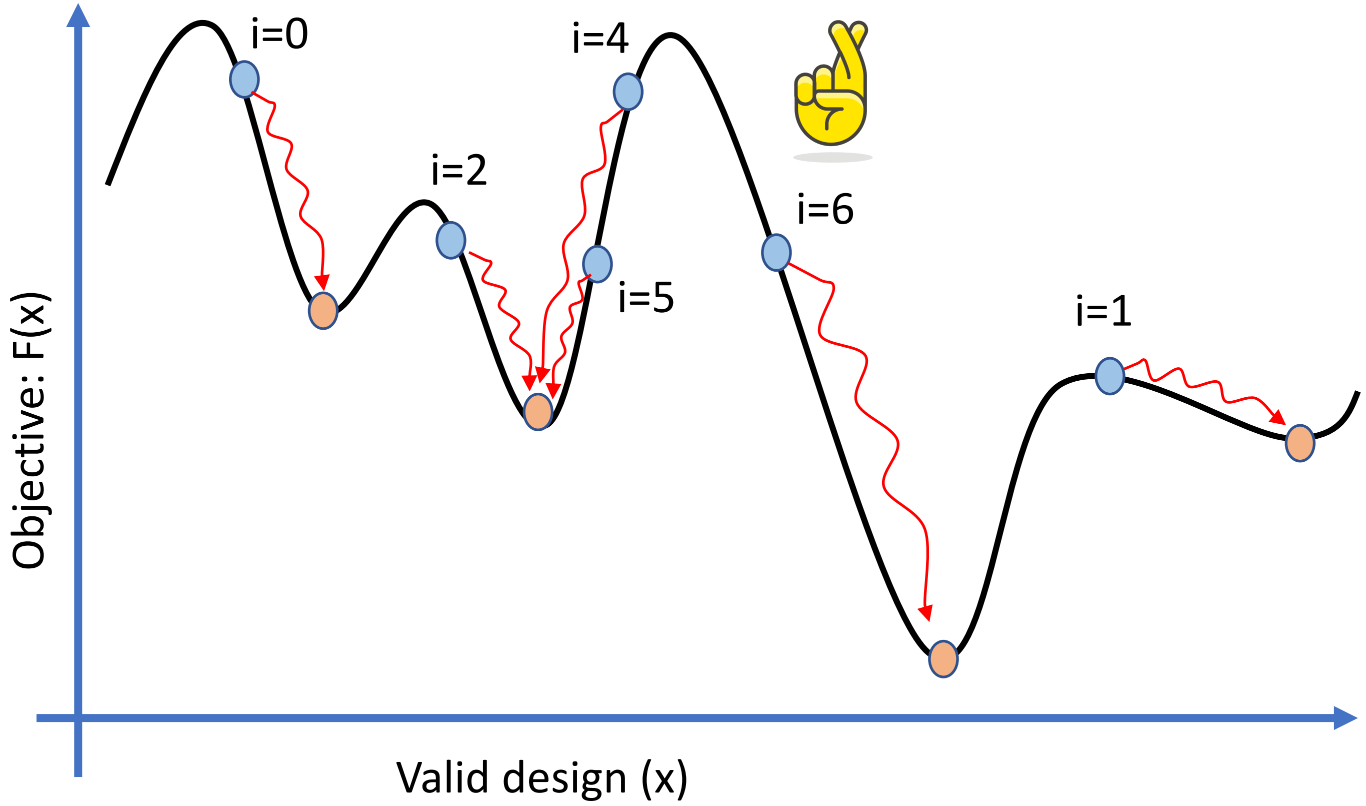
Overall methodology

- **Given:**
  - ***C*** **CPUs,** ***G*** **GPUs,** ***L LLC*****s and** ***P*** **planar links**
  - **System configuration (X × Y)**
- **Goal:**
  - **Place CPU, GPU, LLCs and links**
  - **Satisfy different objectives simultaneously**
    - ***GPU throughput***
    - ***CPU latency***
    - ***Thermal, etc.***



Problems with conventional optimization algorithms

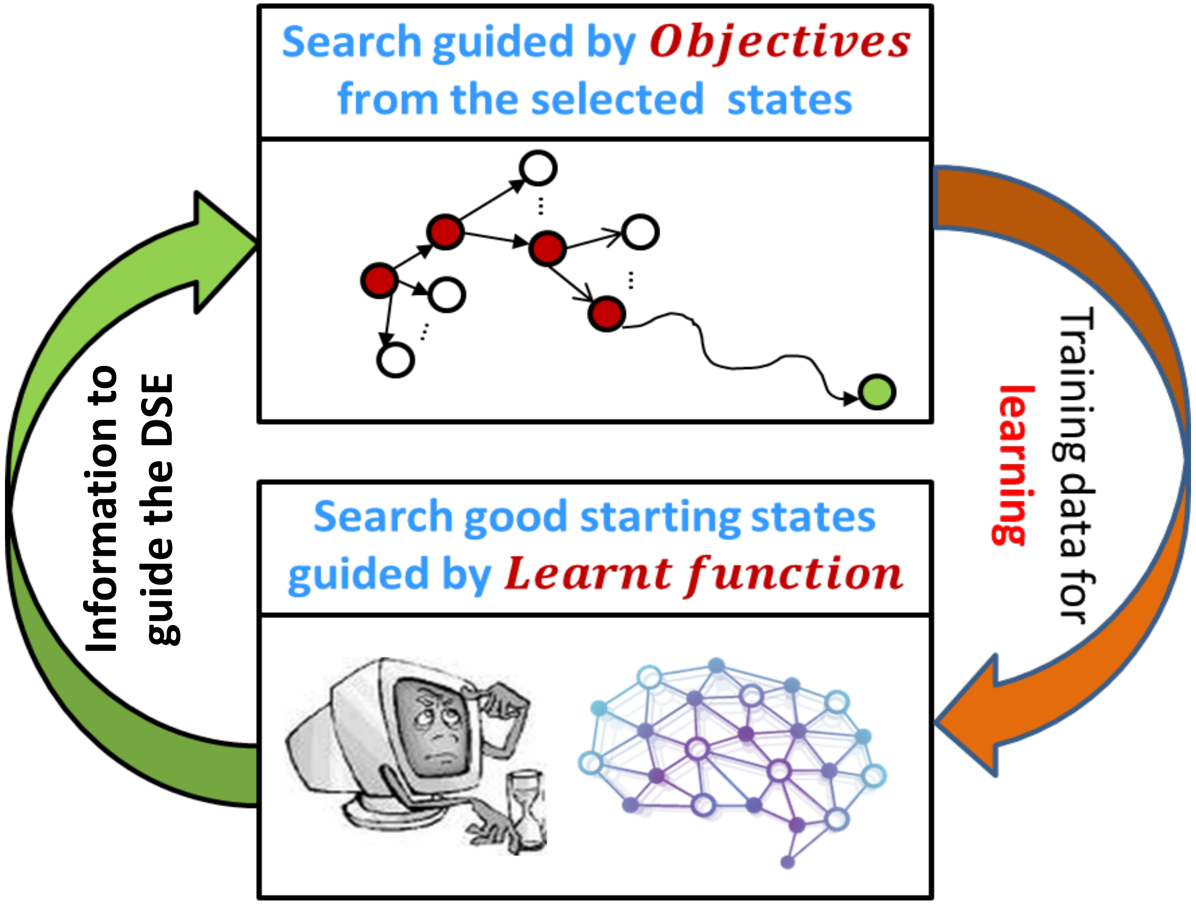




- **Searching blindly**
- **Many iterations before we reach optimal solution**
- **Does not scale with increasing design space**

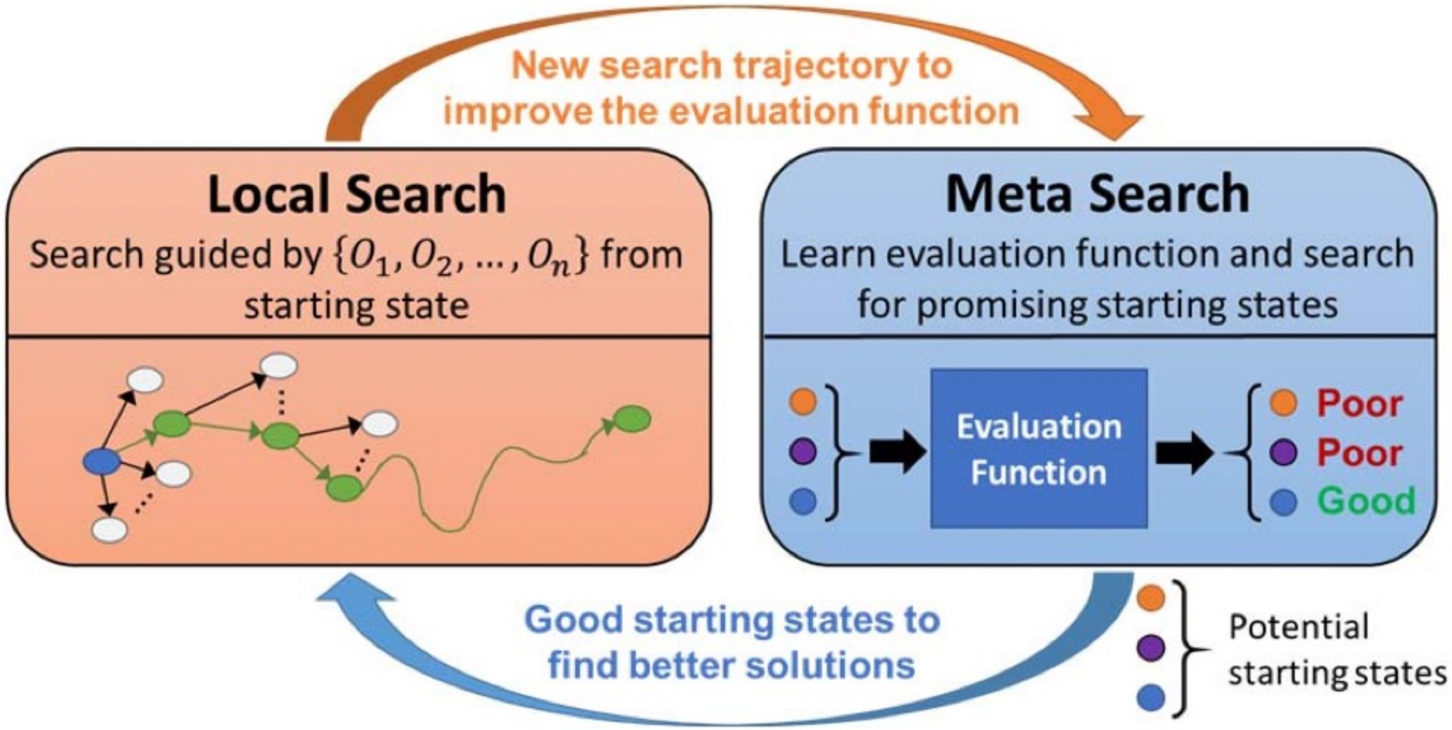
What if someone was there to guide us?

ML based DSE



- Use ML to guide the DSE
- More scalable
- One instance: Use ML to choose better starting points
  - MOO-STAGE [1]
  - Replace "Random restart" by "Guided Restart"

MOO-STAGE: ML-based DSE



- **Replace ”Random restart” with ”Guided restart”**
- **ML model can guide us during exploration**

Local search

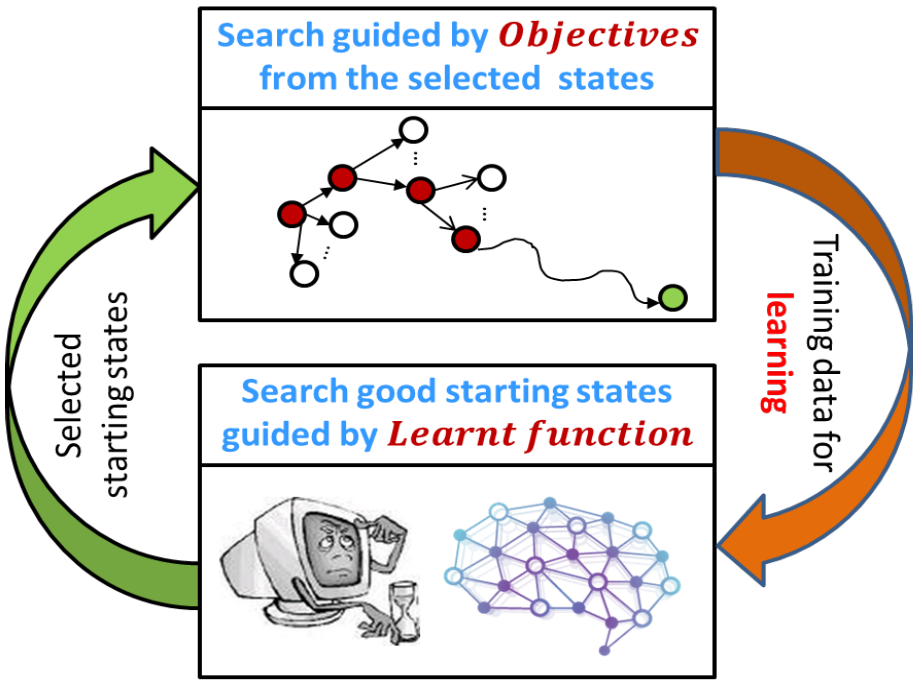
Algorithm 1. Local Search: $local(Obj, d_{start})$

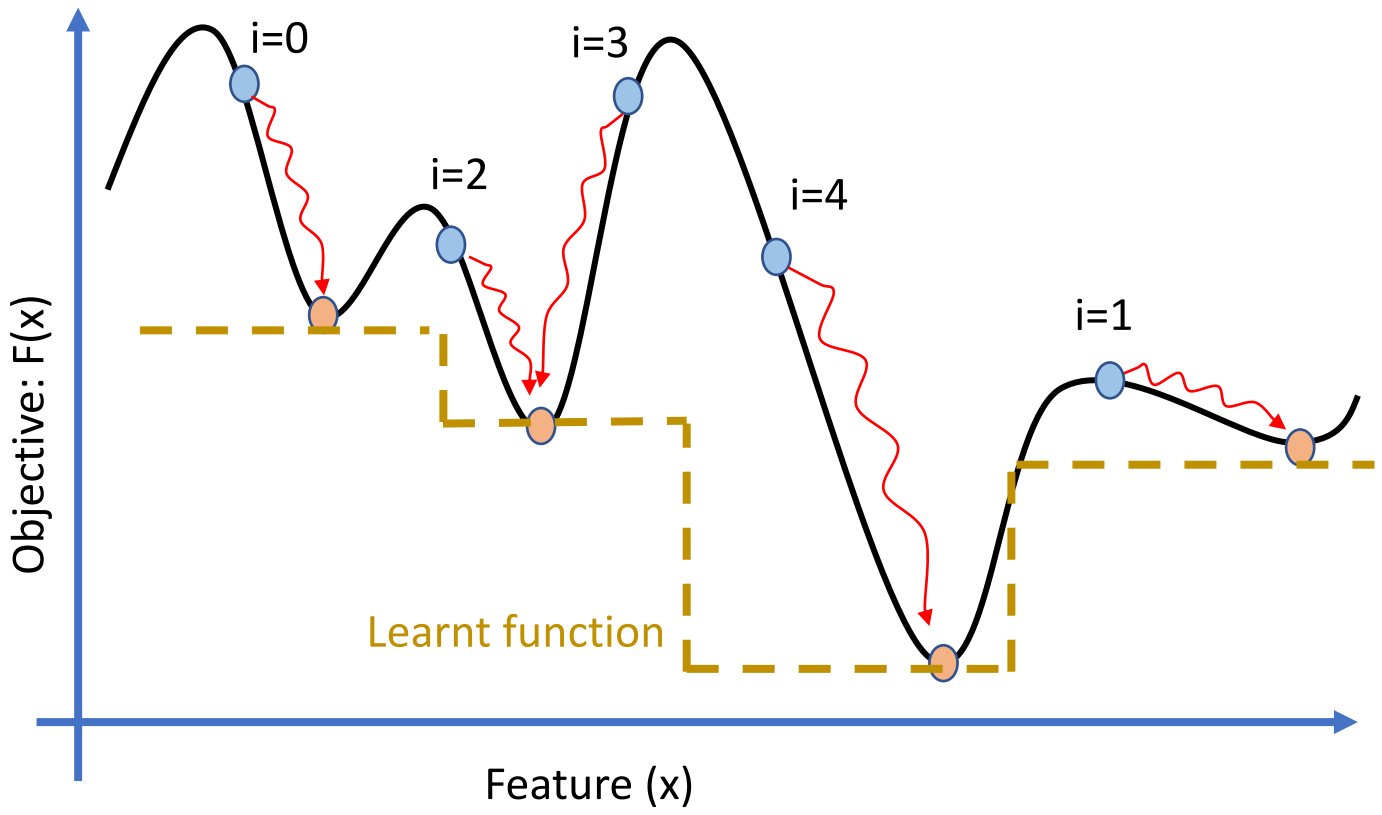
Input: Obj (Set of optimization objectives),
d_{start} (Starting design)
Output: S_{local} (Non-dominated set of designs),
S_{traj} (Trajectory set), d_{last} (Last design)

Initialize: $S_{local} \leftarrow \{d_{start}\}, S_{traj} \leftarrow \{d_{start}\}$,
$d_{curr} \leftarrow d_{start}$
While 1**:**
  $d_{next} \leftarrow \arg \max_{d \in neigh(d_{curr})} PHV_{Obj}(S_{local} \cup \{d\})$
  If $PHV_{Obj}(S_{local} \cup \{d_{next}\}) > PHV_{Obj}(S_{local})$:
    $S_{local} \leftarrow S_{local} \cup \{d_{next}\}$
    $S_{local} \leftarrow \{d \in S_{local} | (\nexists k \in S_{local})[k \prec d]\}$
  Else:
    Return $(S_{local}, S_{traj}, d_{last} \leftarrow d_{curr})$
  $d_{curr} \leftarrow d_{next}$
  $S_{traj} \leftarrow S_{traj} \cup \{d_{curr}\}$

- **Local search step is same as normal greedy search**
- **Uses greedy search here**

Training MOO-STAGE





- **Hill-climbing based search generates the training data**
- **ML algorithm trains to approximate the search space**
- **Self-improving algorithm**
  - **More search → More training data → More accurate Ml model**

Overall algorithm

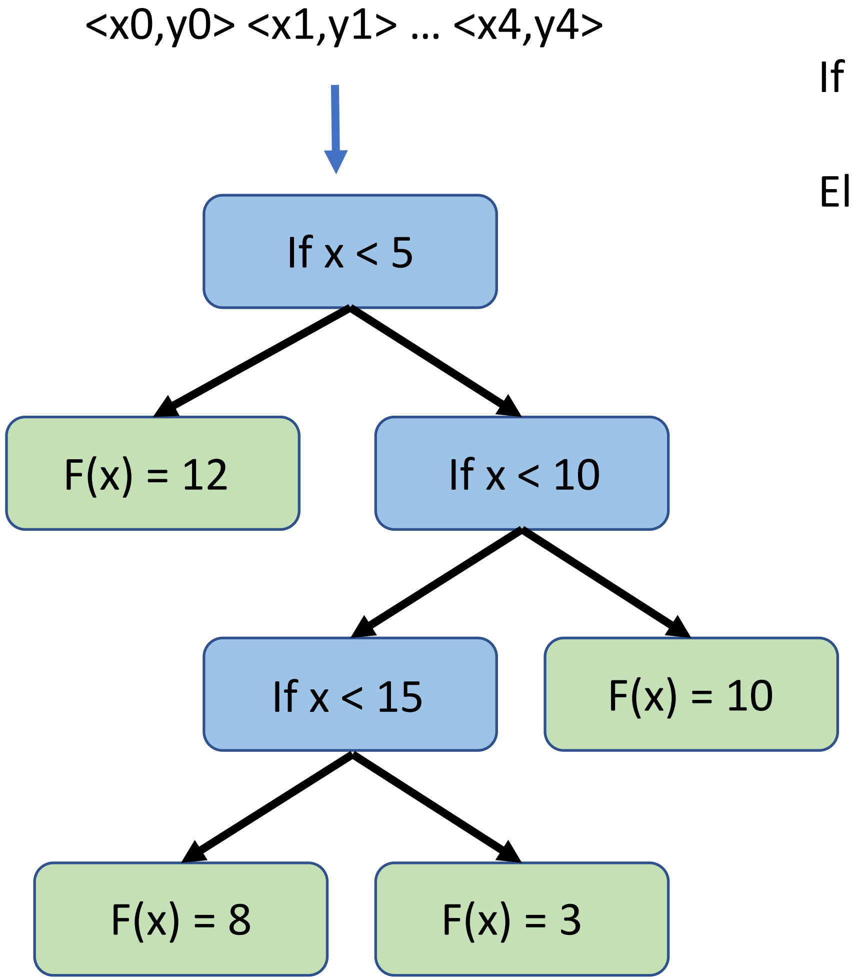
Algorithm 2. MOO-STAGE

Input: Obj (Set of optimization objectives), $iter_{max}$ (Maximum iterations), D (Design space)
Output: S_{global} (Non-dominated set of designs)

Initialize: $S_{global} \leftarrow \emptyset, S_{train} \leftarrow \emptyset, d_{start} \leftarrow rand(D)$
For $i = 0$ to $iter_{max}$:
    $(S_{local}, S_{traj}, d_{last}) \leftarrow local(Obj, d_{start})$
    Maintain non-dominated global set:
    $S_{global} \leftarrow S_{global} \cup S_{local}$
    $S_{global} \leftarrow \{d \in S_{global} \mid (\nexists k \in S_{global})[k \prec d]\}$
    If $S_{global} \cap S_{local} = \emptyset$: [If algorithm converged]
        Return S_{global}
    Add training example for each design $d \in S_{traj}$:
    $S_{train} \leftarrow S_{train} \cup \{(d, PHV_{Obj}(S_{traj}))\}$
    Train evaluation function: $Eval \leftarrow train(S_{train})$
    Greedy Search: $d_{restart} \leftarrow greedy(Eval, d_{last})$
    If $d_{last} = d_{restart}$:
        $d_{start} \leftarrow rand(D)$
    Else
        $d_{start} \leftarrow d_{restart}$
Return S_{global}

- **Uses regression forest in overall optimization**
- **Implements guided restart**

Decision tree algorithm (1)

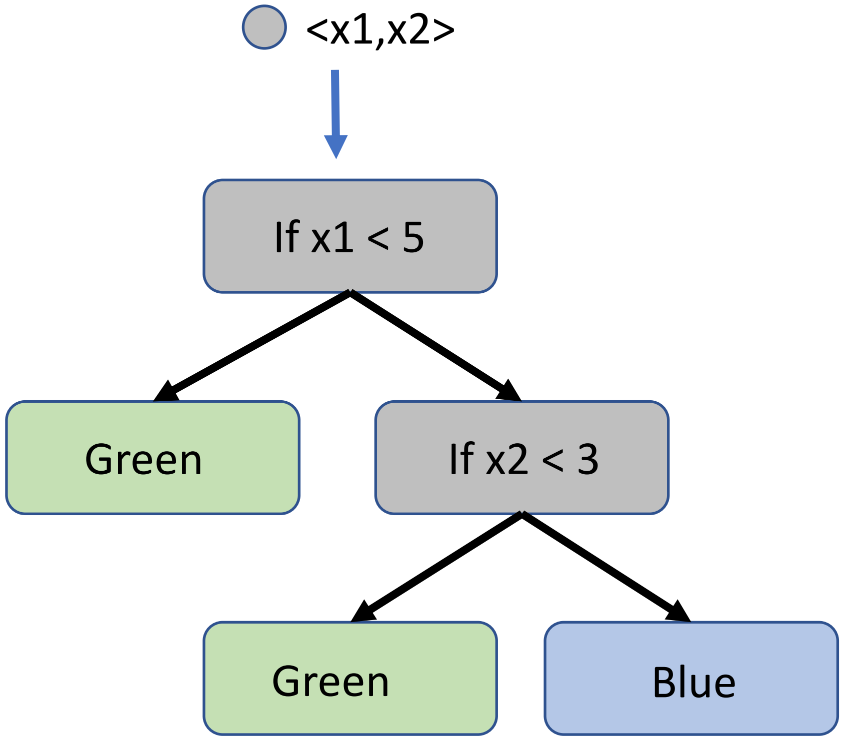


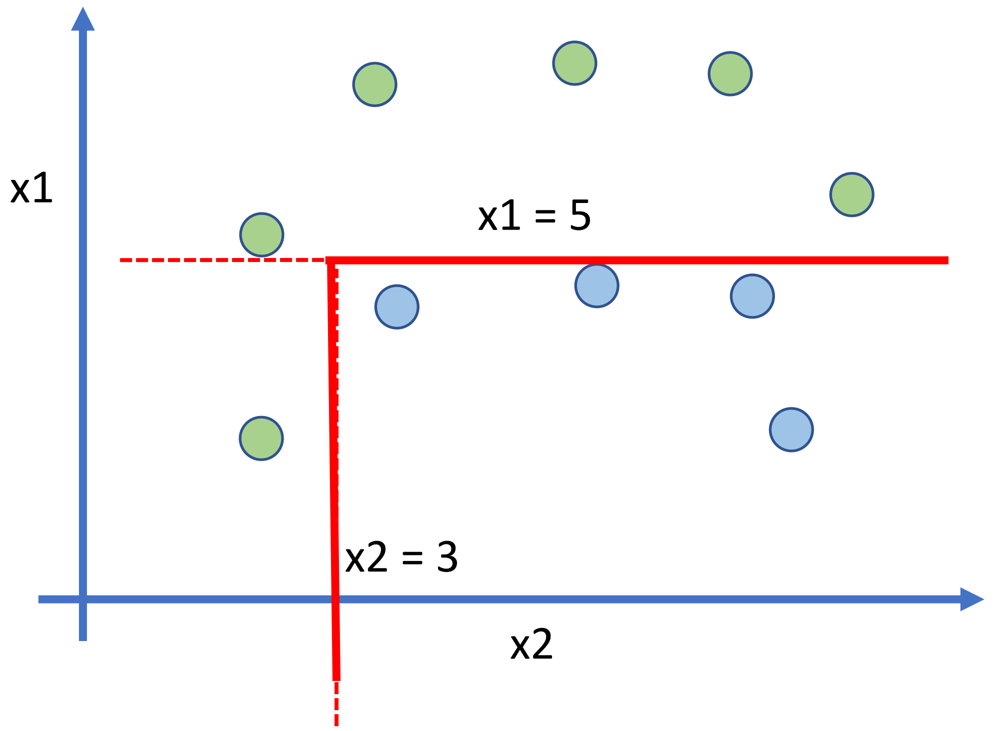

```
If x < 5:
   F(x) = 12
Else:
   if x < 10:
      F(x) = 15
   else:
      if x < 15:
         F(x) = 3
      else:
         F(x) = 8
```

- **A tree made out of if-else**
- **Can be used for both classification and regression**

https://www.youtube.com/watch?v=7VeUPuFGJHk

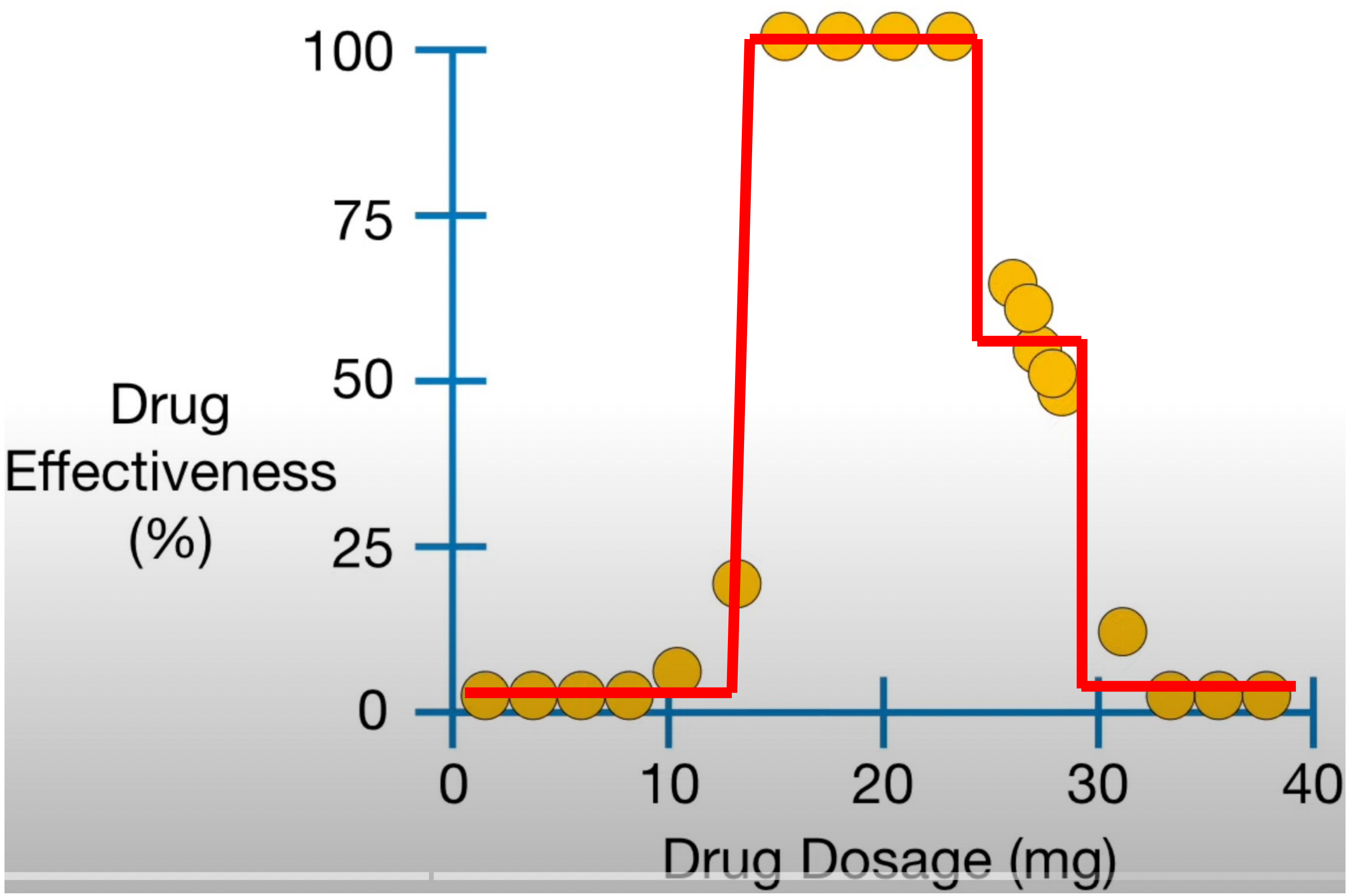
Decision tree algorithm (2)

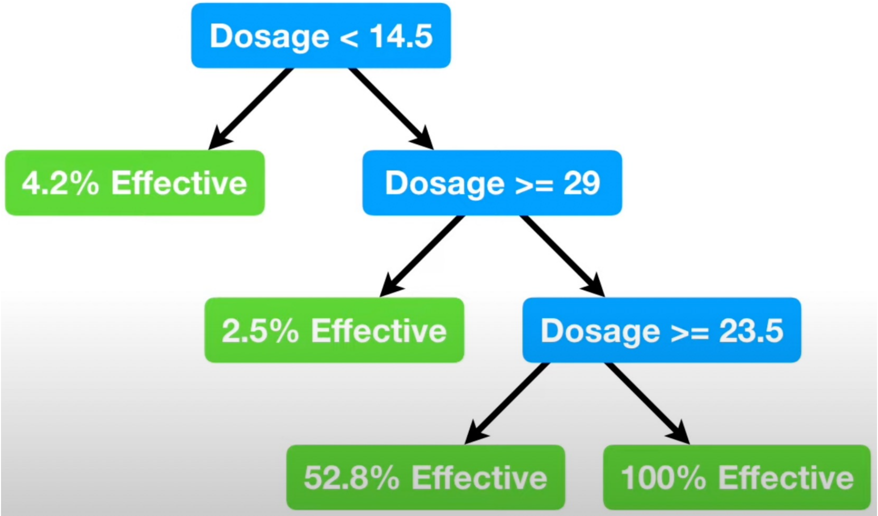




- **Color classification problem**
- **Two linear lines make a non-linear boundary**

Regression tree

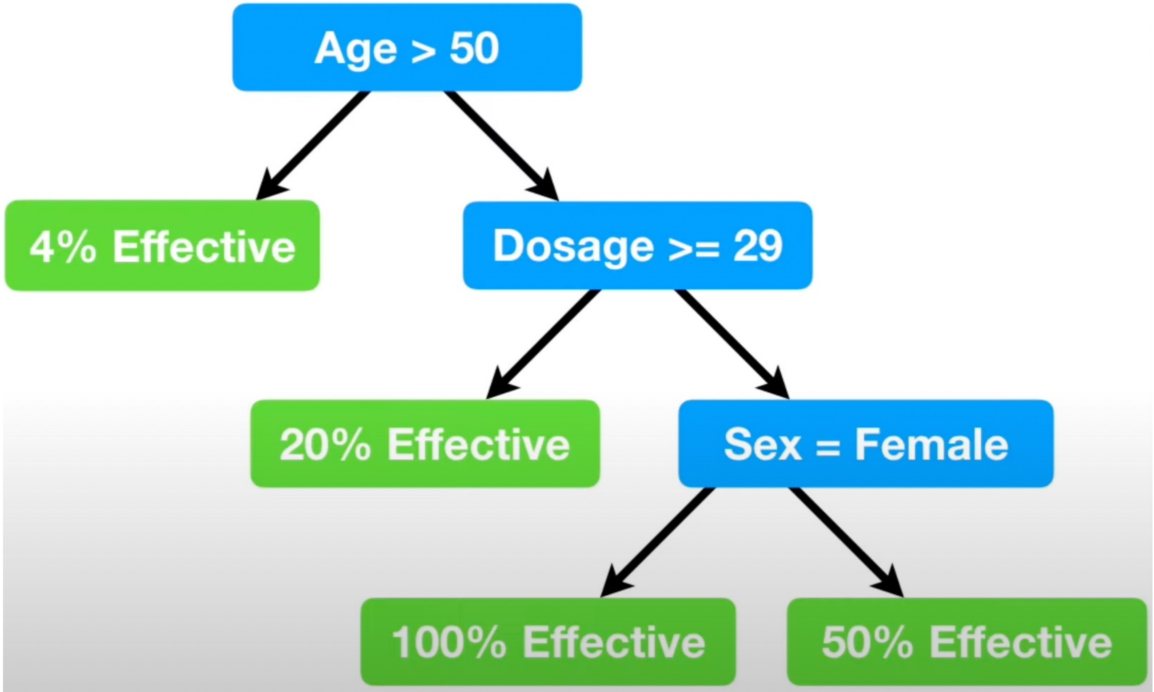




https://www.youtube.com/watch?v=g9c66TUylZ4

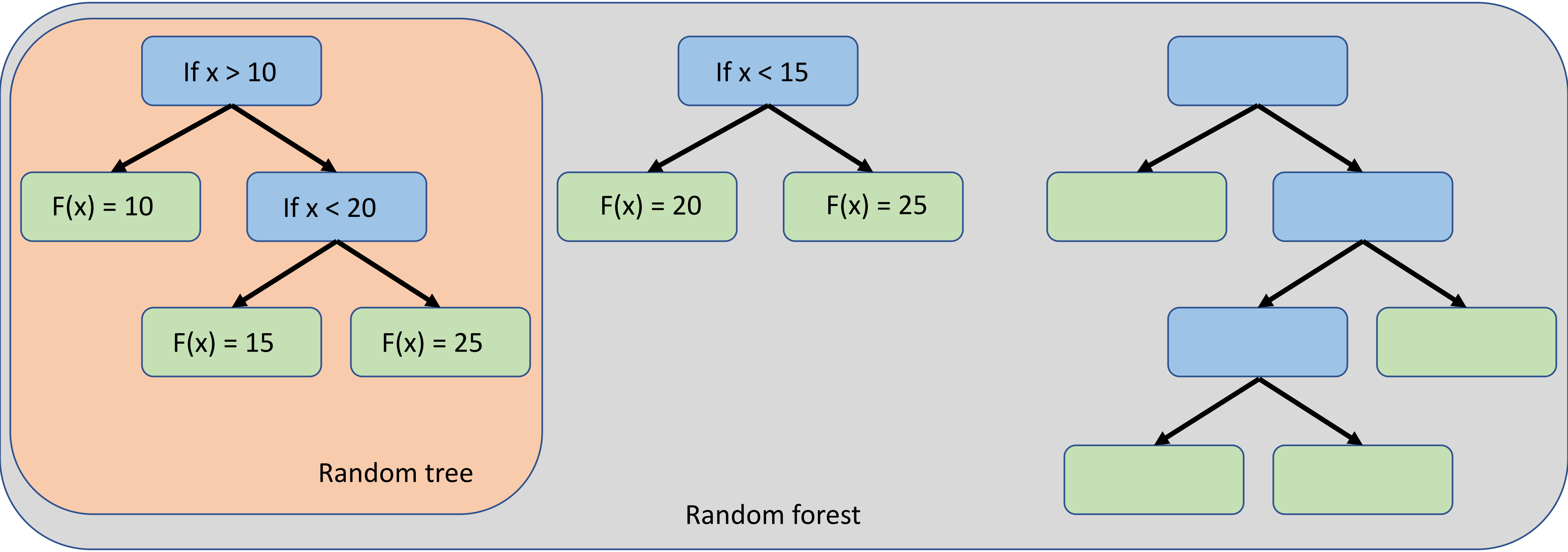
Regression tree (2)

Dosage	Age	Sex	Etc.	Drug Effect.
10	25	Female	...	98
20	73	Male	...	0
35	54	Female	...	100
5	12	Male	...	44
etc...	etc...	etc...	etc...	etc...



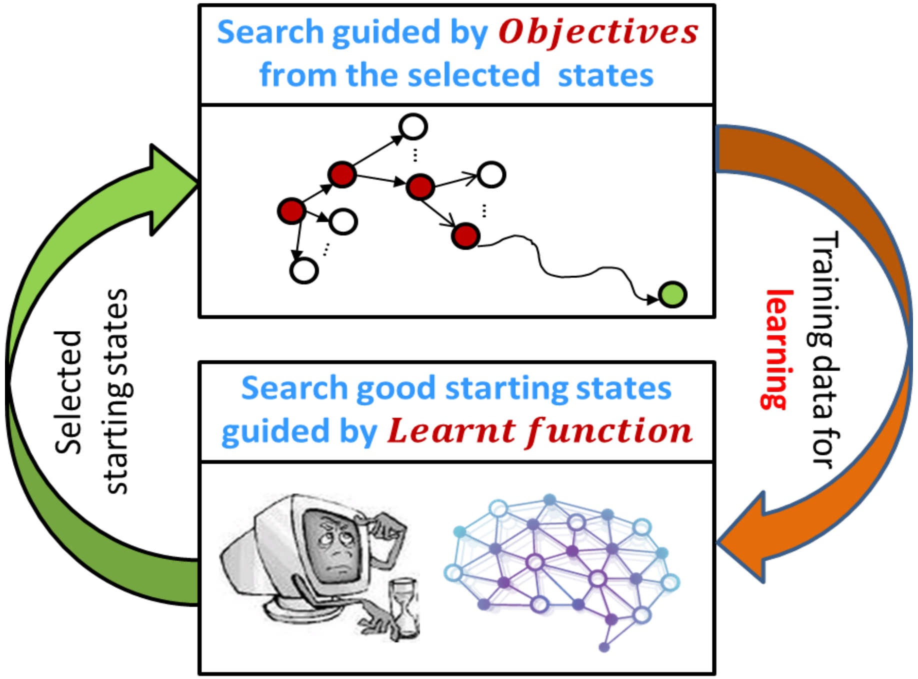
https://www.youtube.com/watch?v=g9c66TUylZ4

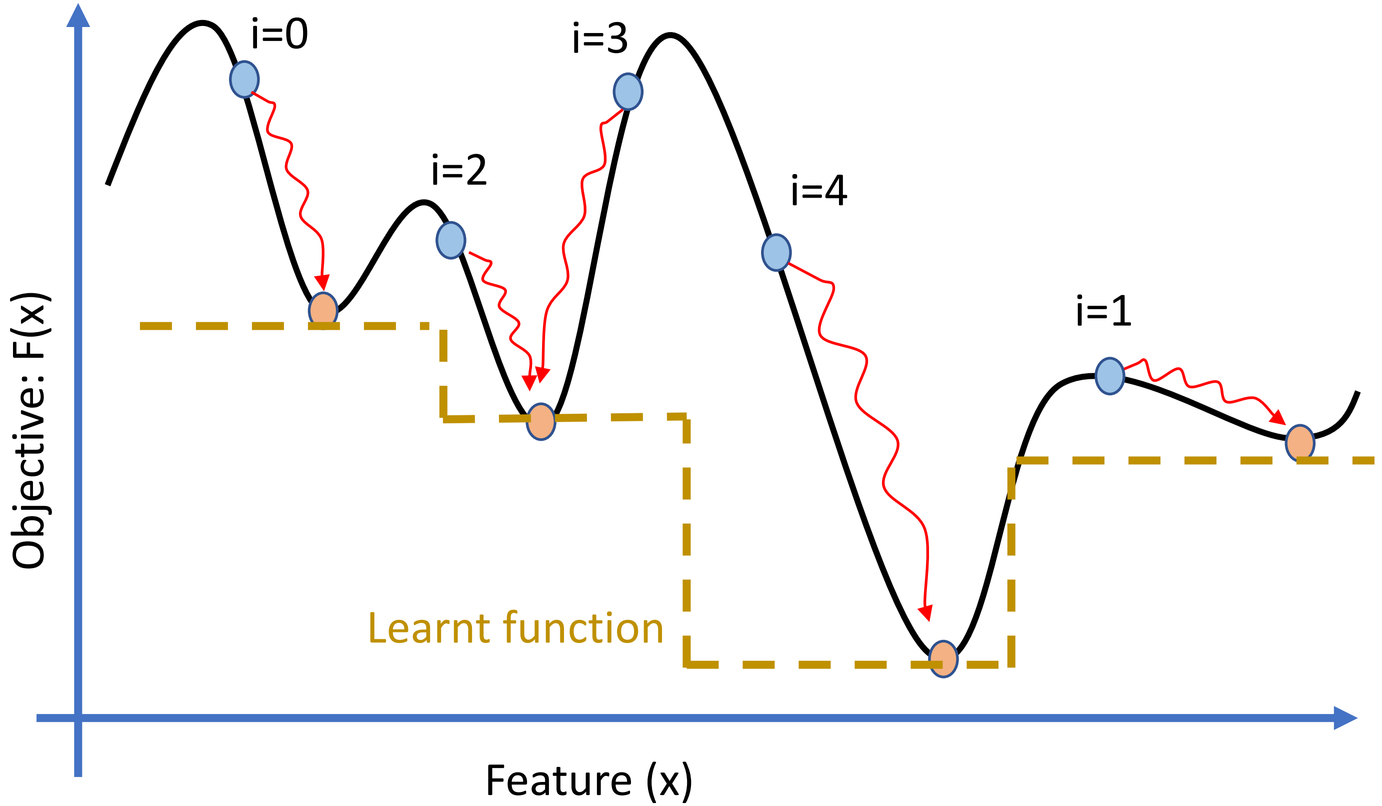
Random Forest



- **Random forest regressor**
  - **Forest = Many trees**
  - **Each 'tree' is similar to a 'if-else' tree**
  - **https://www.youtube.com/watch?v=J4Wdy0Wc_xQ**

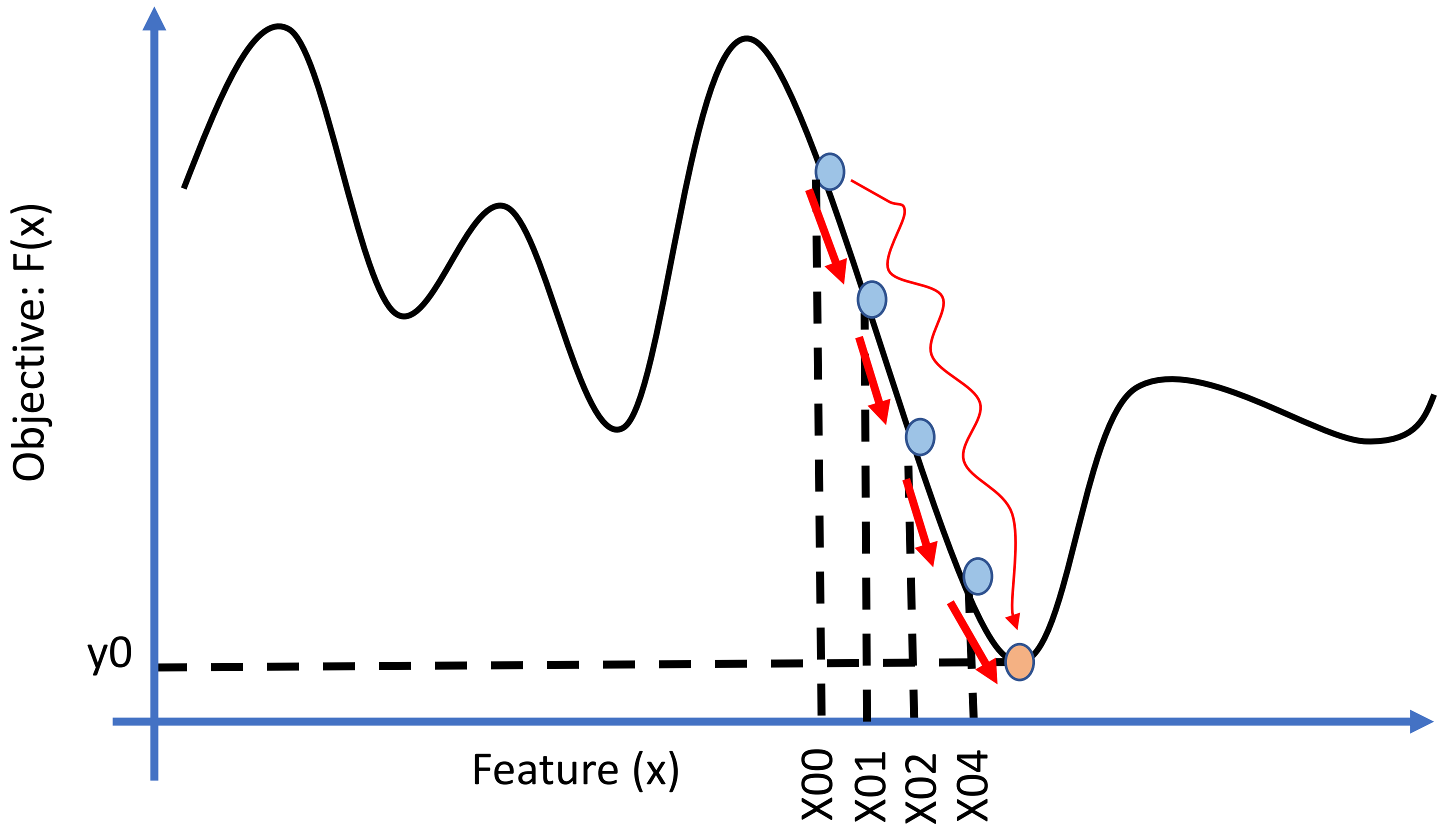
MOO-STAGE: Recap





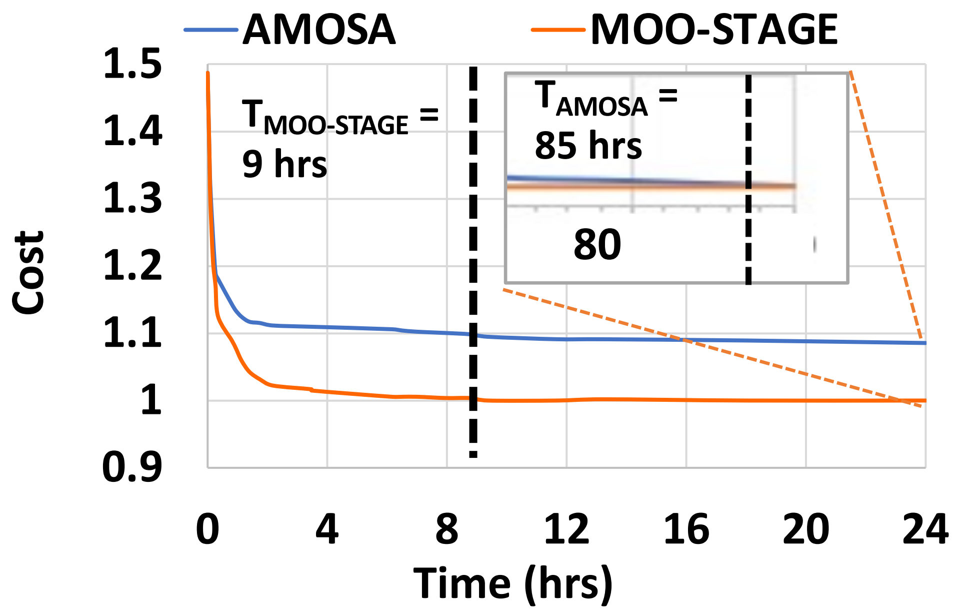
- **Hill-climbing based search generates the training data**
- **ML algorithm trains to approximate the search space**
- **Self-improving algorithm**
  - **More search → More training data → More accurate Ml model**

Creating the training data



- **Training Data: <Point in design space, BestCost that can be reached from this point>**
  - **{ <x00,y0> <x01,y0> <x02,y0> <x03,y0> }**

MOO-STAGE performance



- Trained model acts as a filter
- Prevents searching in bad locations of design space
  - Without ML, the algorithm would waste time exploring bad regions
  - Up to 10X faster than conventional algorithms

Summarizing the DSE problem

- Task mapping
  - Given: T tasks and N cores
  - Output: Map T tasks to N cores such that goal is reached
  - Goal: Lower execution time, Better communication, etc.
  - Design objectives: Low latency, High throughput, Low energy, etc.
- NoC design
  - Given: L links and N cores
  - Output: Place L links between pairs of cores to reach objective
  - Goal: Lower execution time, Better communication, etc.
  - Design objectives: Low latency, High throughput, Low energy, etc.

MOOS algorithm

ALGORITHM 2: MOOS Algorithm

Input: $\mathcal{D}$ = design space; $\mathcal{O}$ = set of optimization objectives; C = physical constraints; MAX = maximum number of iterations

Initialize $\mathcal{P}_{global}$ with some initial design
Initialize the tree-based model $\mathcal{M}$ over the space of λ values
while convergence or MAX iterations **do**
    Select the best leaf node n^* from $\mathcal{M}$ for expansion
    Create three child nodes n_{left}, n_{center}, and n_{right} corresponding to equal partitions of the hyper-rectangle represented by node n^* along the longest dimension
    Suppose λ_{left} and λ_{right} be the centers of hyper-rectangles represented by n_{left} and n_{right}
    for all $\lambda \in \{\lambda_{left}, \lambda_{right}\}$ **do**
        $d_{start} \leftarrow$ SELECT-START-DESIGN$(\mathcal{O}, \lambda, \mathcal{P}_{global})$
        $\mathcal{P}_{local} \leftarrow$ LOCAL-SEARCH$(\mathcal{O}, d_{start}, C, MAX)$
        $\mathcal{P}_{global} \leftarrow$ non-dominating designs from $\mathcal{P}_{global} \cup \mathcal{P}_{local}$
        $PHV_{global} \leftarrow PHV(\mathcal{O}, \mathcal{P}_{global})$
        Update tree-based model $\mathcal{M}$ using the new evaluation data (λ, PHV_{global})
    end for
end while
return global Pareto set $\mathcal{P}_{global}$

MOOS flowchart

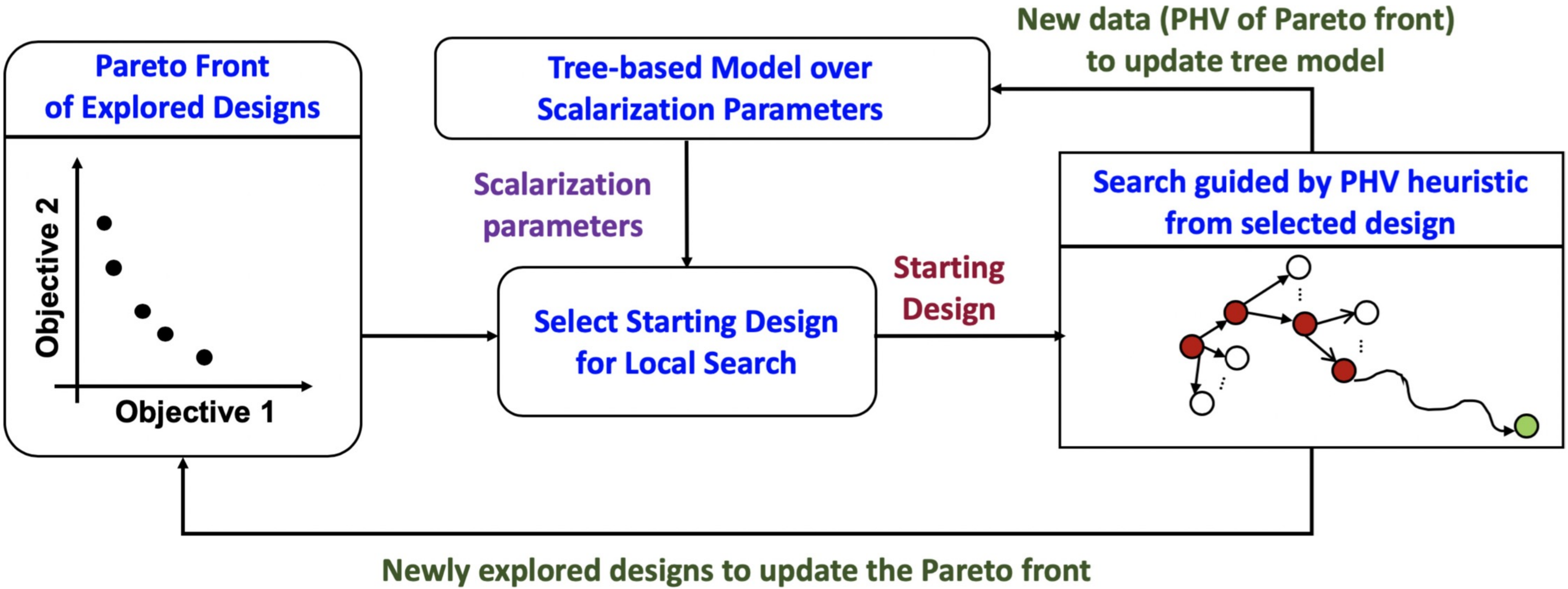


Fig. 4. Overview of MOOS algorithm.

Search hyperparameter space using trees

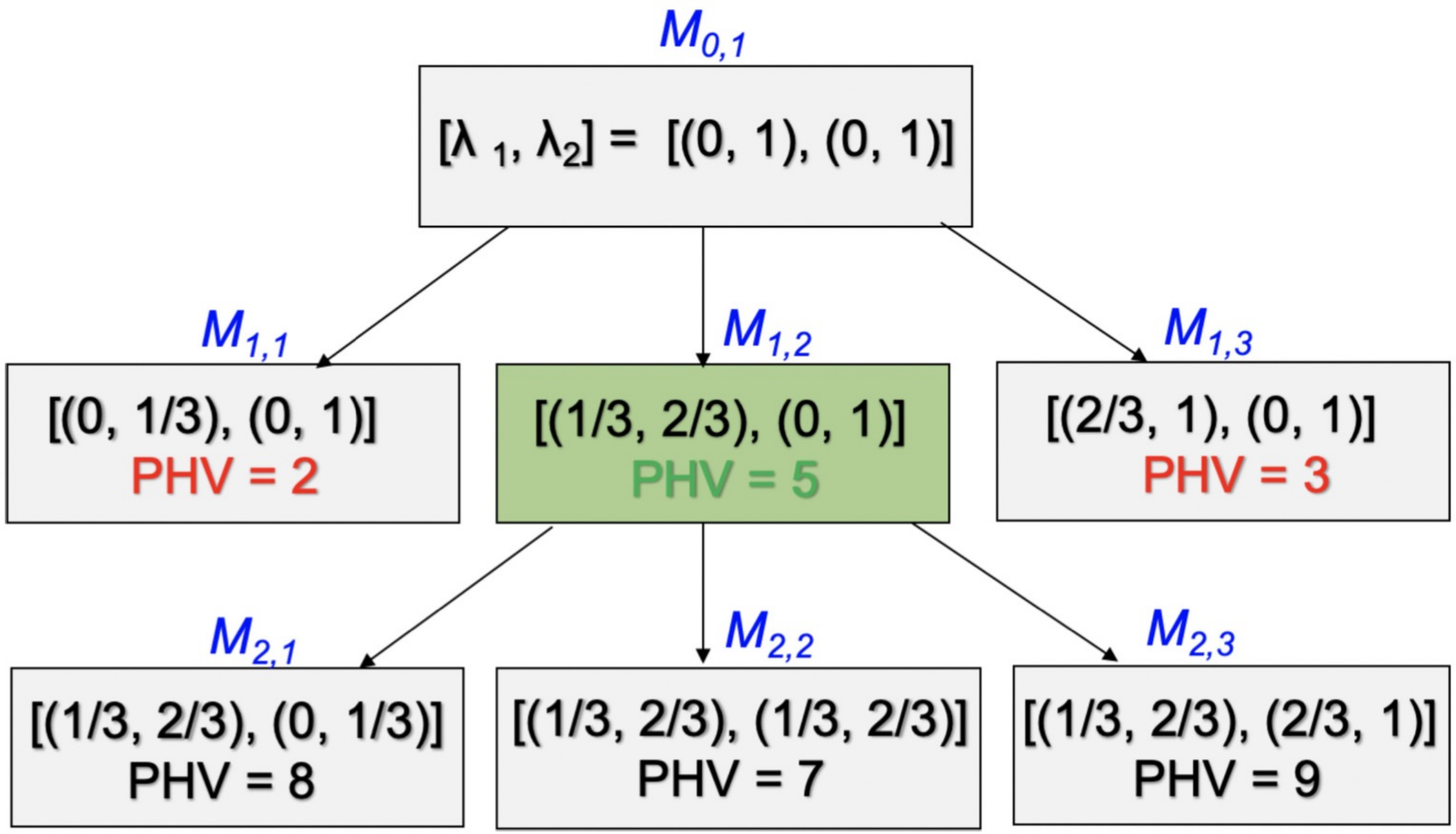


Fig. 5. Illustration of optimistic search via tree based model. Each cell corresponds to a partition of the space of λ parameters. After initial partition, the leaf cell with the best PHV denoted in green (higher PHV is better when minimizing all the k=2 objectives) is partitioned into $W = 3$ children cells for evaluation. This process is repeated till convergence.

MOOS local search

ALGORITHM 4: Local Search Procedure

Input: $\mathcal{O}$ = set of optimization objectives; d_{start} = starting design; C = physical constraints; MAX = maximum iterations

$\mathcal{P}_{local} \leftarrow \{d_{start}\}$; and $d_{curr} \leftarrow d_{start}$
while convergence or MAX iterations **do**
    $d_{next} = \text{argmax}_{d \in \mathcal{N}(d_{curr}) \cup \{d_{curr}\}} PHV(\mathcal{O}, \mathcal{P}_{local} \cup \{d\})$
    if $PHV(\mathcal{O}, \mathcal{P}_{local} \cup \{d_{next}\}) > PHV(\mathcal{O}, \mathcal{P}_{local})$ **then**
        $\mathcal{P}_{local} \leftarrow$ non-dominating designs from $\mathcal{P}_{local} \cup \{d_{next}\}$
    end if
    $d_{curr} \leftarrow d_{next}$
end while
return local Pareto set $\mathcal{P}_{local}$

MOOS vs MOO-STAGE