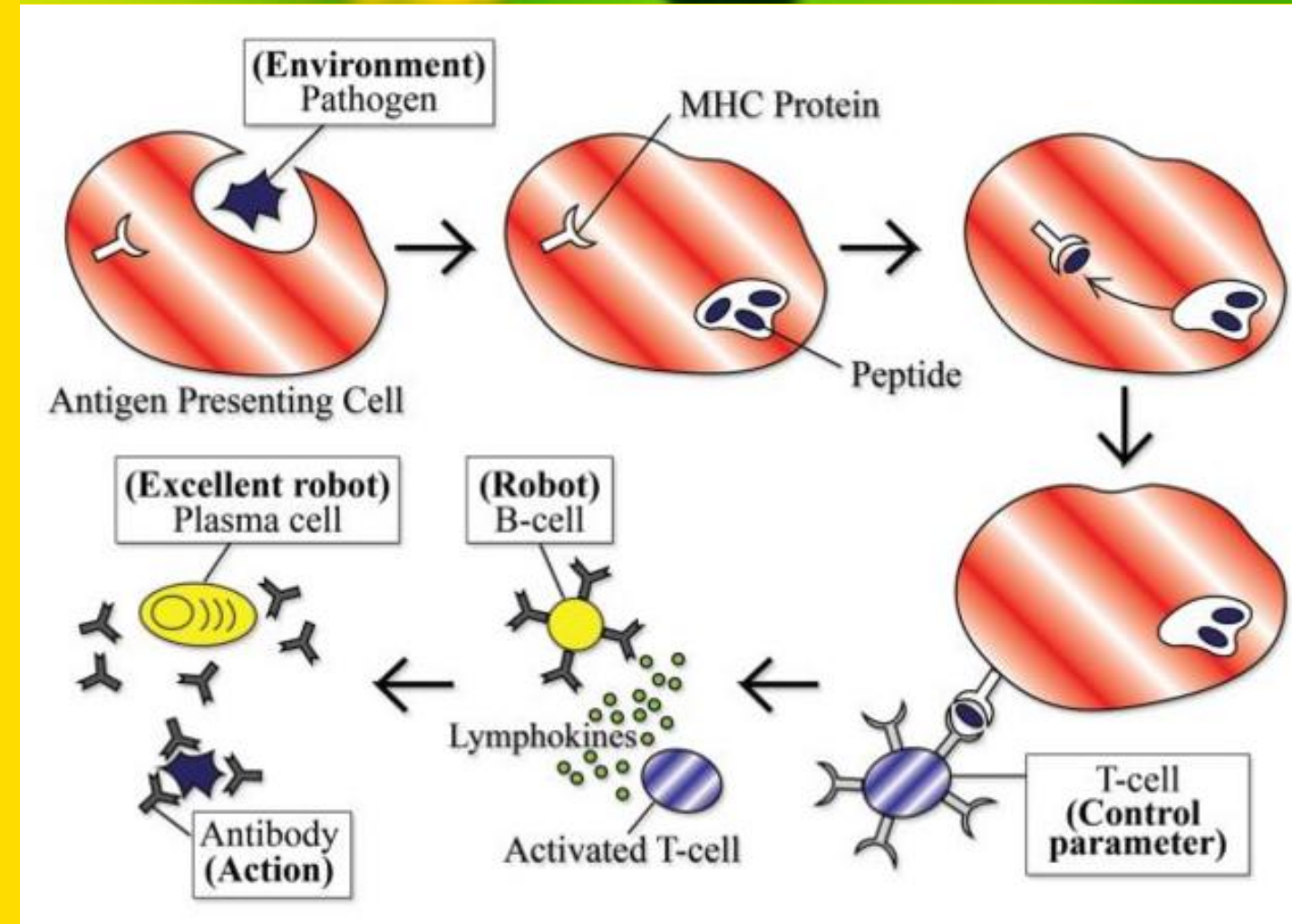


ACIT4610: Evolutionary Artificial Intelligence and Robotics

Lecture – 6: Clever Algorithms

Kazi Shah Nawaz Ripon

karip8799@oslomet.no



Bees Algorithm (BA)

Bees Algorithm (BA)

- Population-based search algorithm
- Mimics the food foraging behavior of honey bees to find the optimal solution
- Developed by Prof. D.T. Pham and his co-workers in 2005.
- Basic version performs a kind of **neighborhood search** combined with **global search**.
- It can be used for both **combinatorial optimization** and **continuous optimization**.



* Pham, D. T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., & Zaidi, M. (2006). The bees algorithm—a novel tool for complex optimisation problems. In *Intelligent production machines and systems* (pp. 454-459). Elsevier Science Ltd.

- Honey bees collect nectar from vast areas around their hive.
- Bee colonies have been observed sending bees to collect nectar from flower patches relative to the amount of food available at each patch.
- Bees communicate with each other at the hive via a **waggle dance** to inform other bees as to the
 - (i) direction,
 - (ii) distance, and
 - (iii) quality rating of food sources.

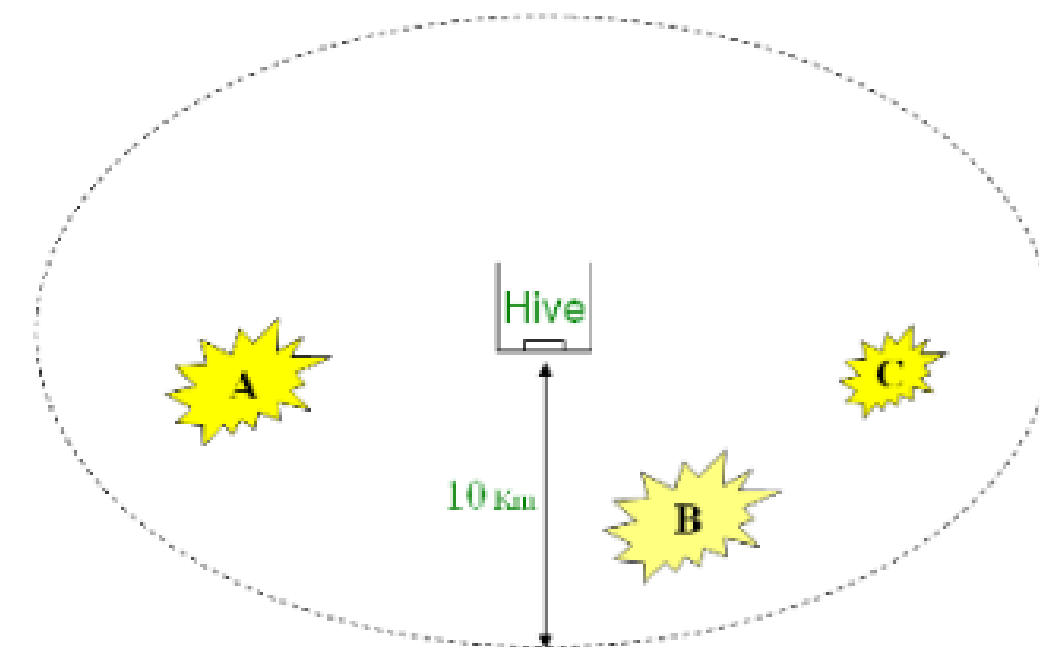


Waggle Dance



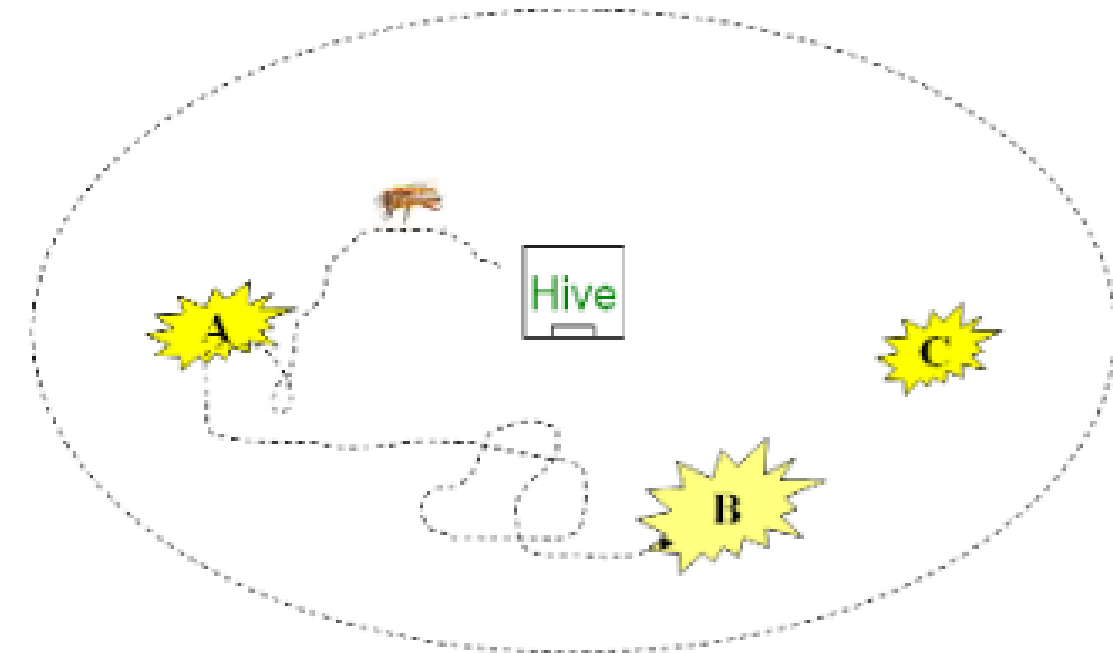
Bees in Nature -- 1

- A colony of honey bees can extend itself over long distances in multiple directions (more than 10 km).
- The colony employs about one-quarter of its members (**scout bees**) as forager bees.



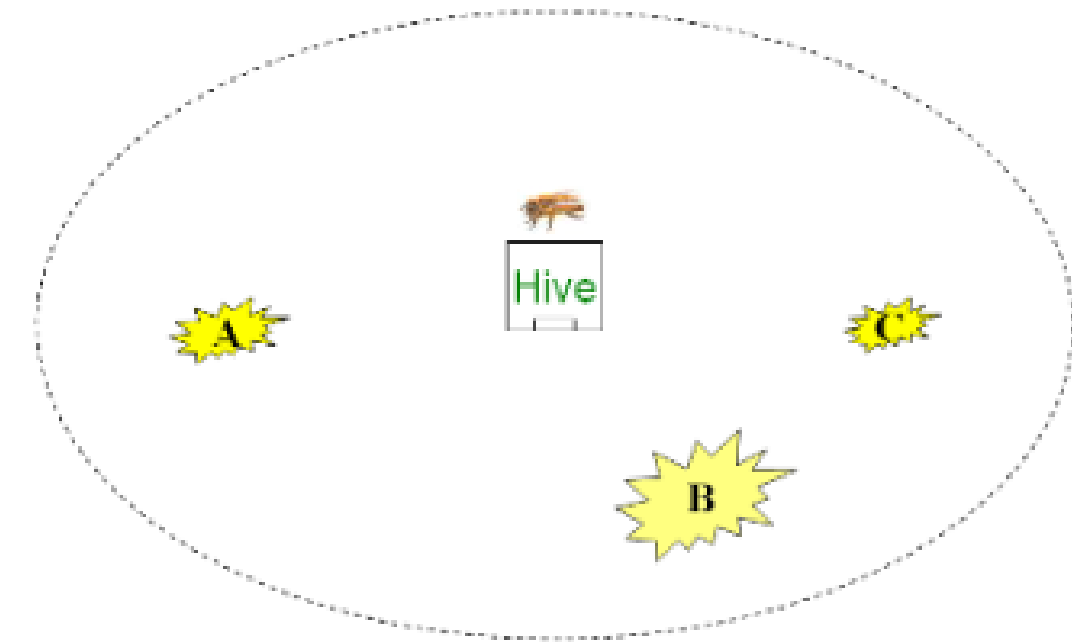
Bees in Nature -- 2

- **Scout bees** search randomly from one patch to another.
- Flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more scout bees,
 - whereas patches with less nectar or pollen should receive fewer scouts.



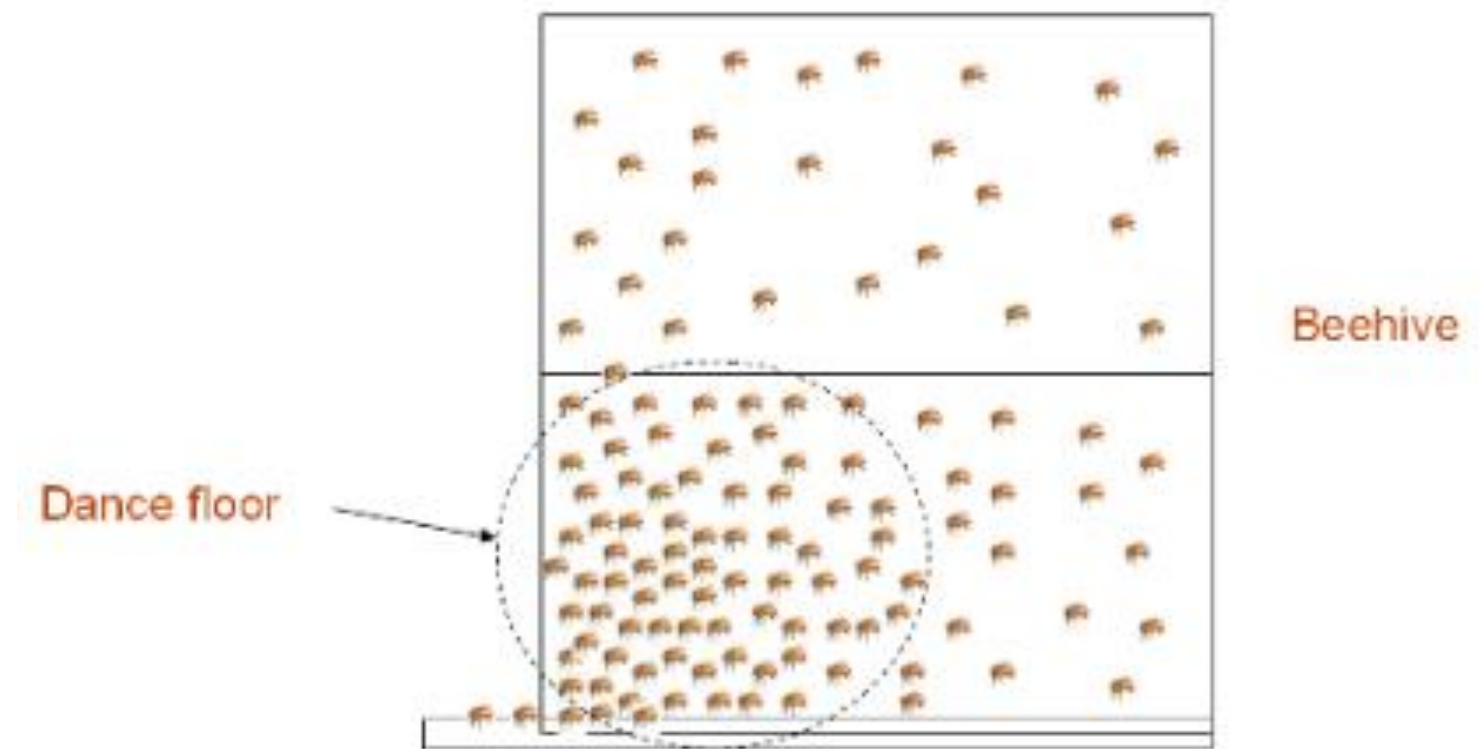
Bees in Nature -- 3

- The scouts who return to the hive, **evaluate** the different patches depending on certain quality threshold (measured as a combination of some elements, such as sugar content).



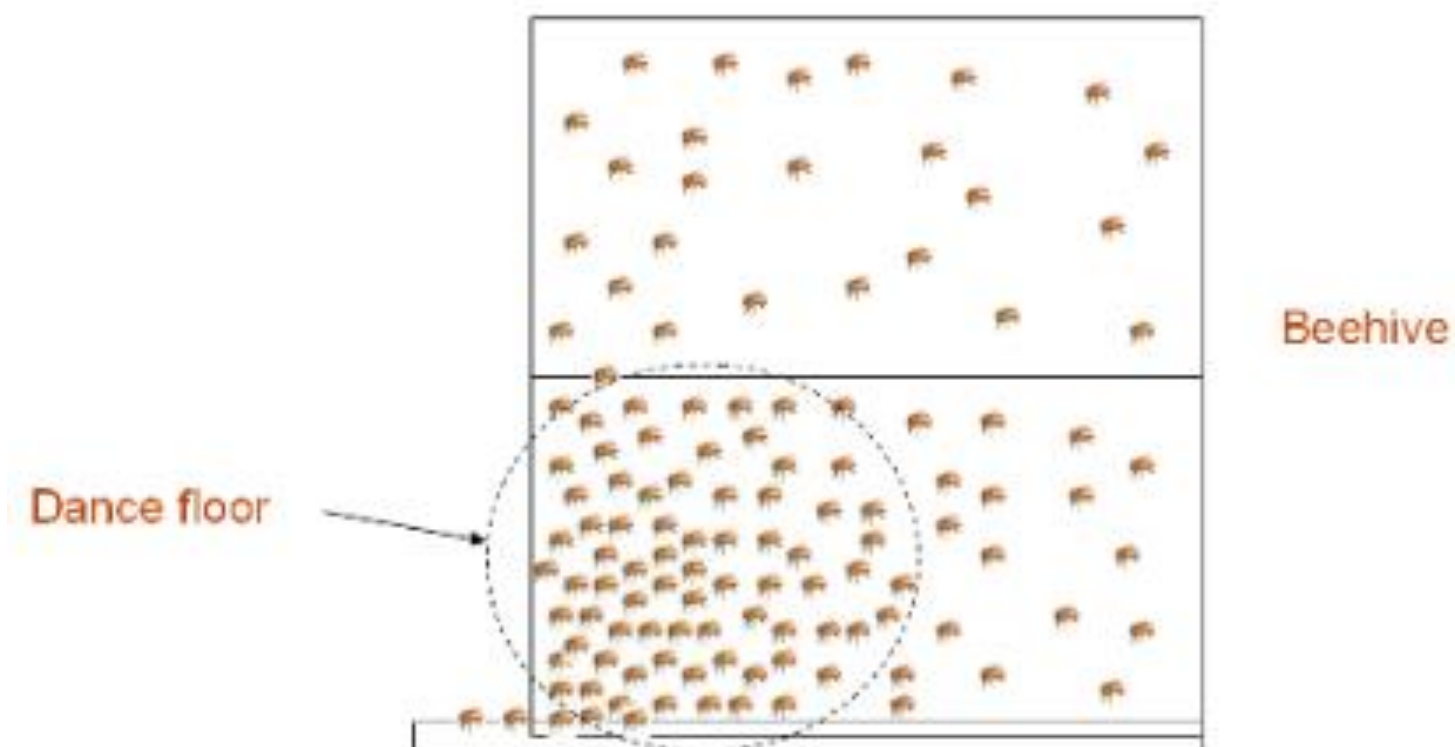
Bees in Nature -- 4

- Scout bees deposit their nectar and go to the dance floor in front of the hive to **communicate** to the other bees by performing “**waggle dance**”.



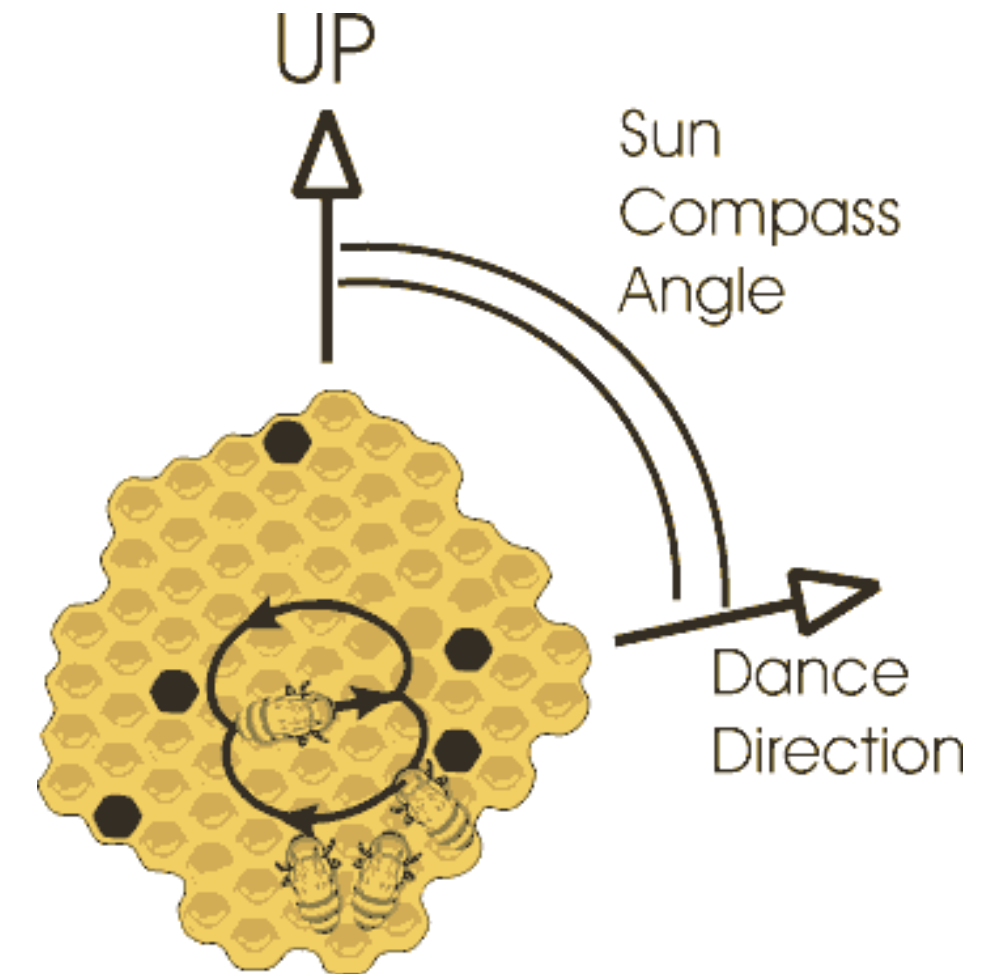
Bees in Nature -- 5

- A small number of scouts continue to search for new patches
 - while bees returning from flower patches continue to communicate the quality of the patch.



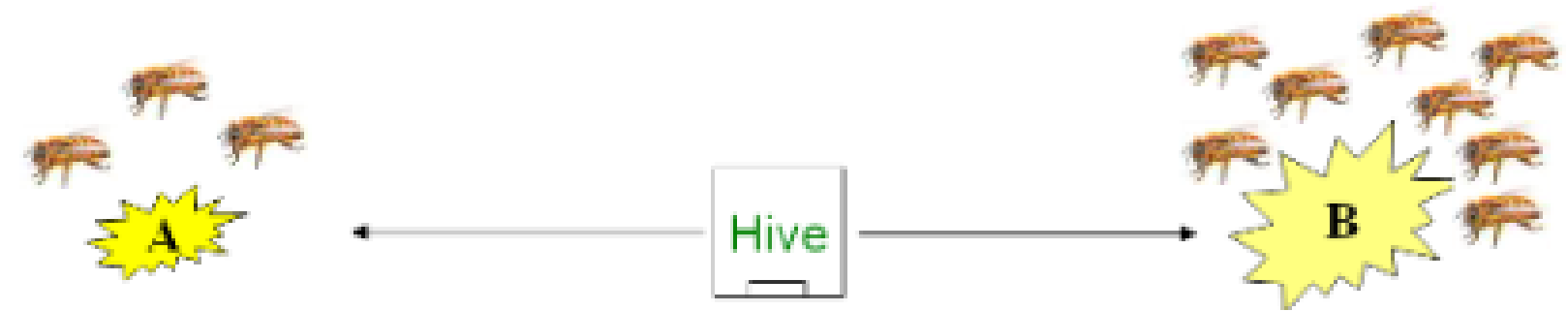
Bees in Nature -- 6

- Scout bees provide the following information by waggle dance:
 1. The **direction** of flower patches (angle between the sun and the patch).
 2. The **distance** from the hive (duration of the dance).
 3. The **quality** rating (fitness) (frequency of the dance).
- These information helps the colony to send its bees precisely.



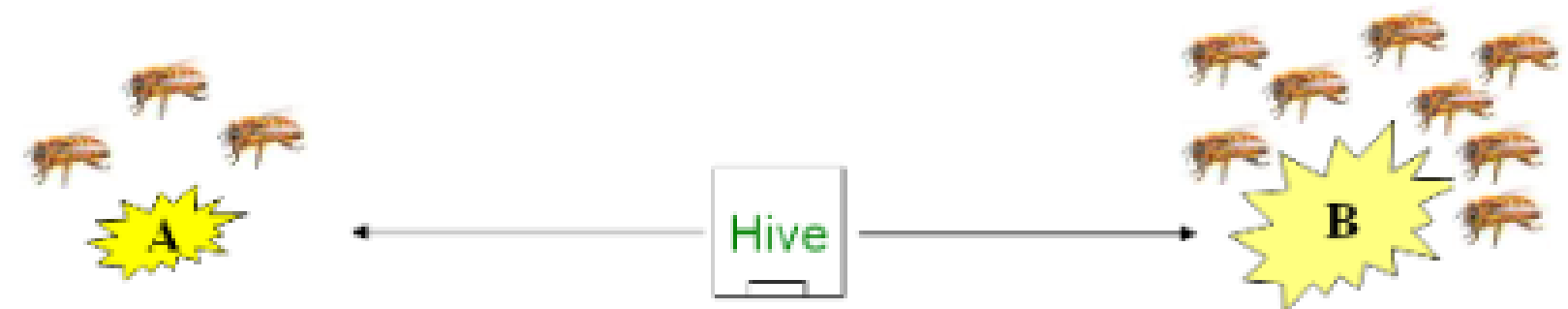
Bees in Nature -- 7

- The scout backs to the flower patch with **follower bees** to gather food efficiently and quickly.



Bees in Nature -- 8

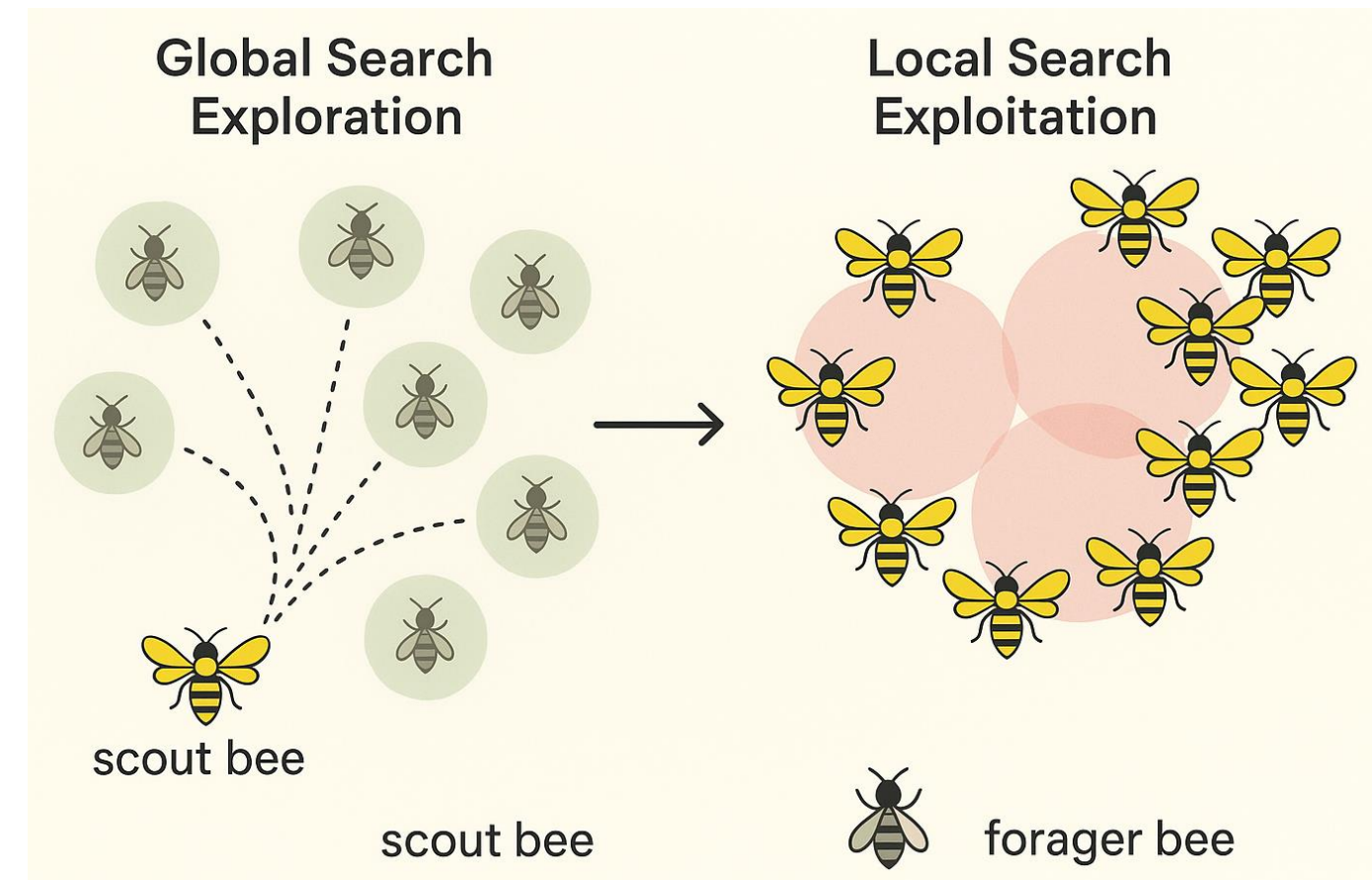
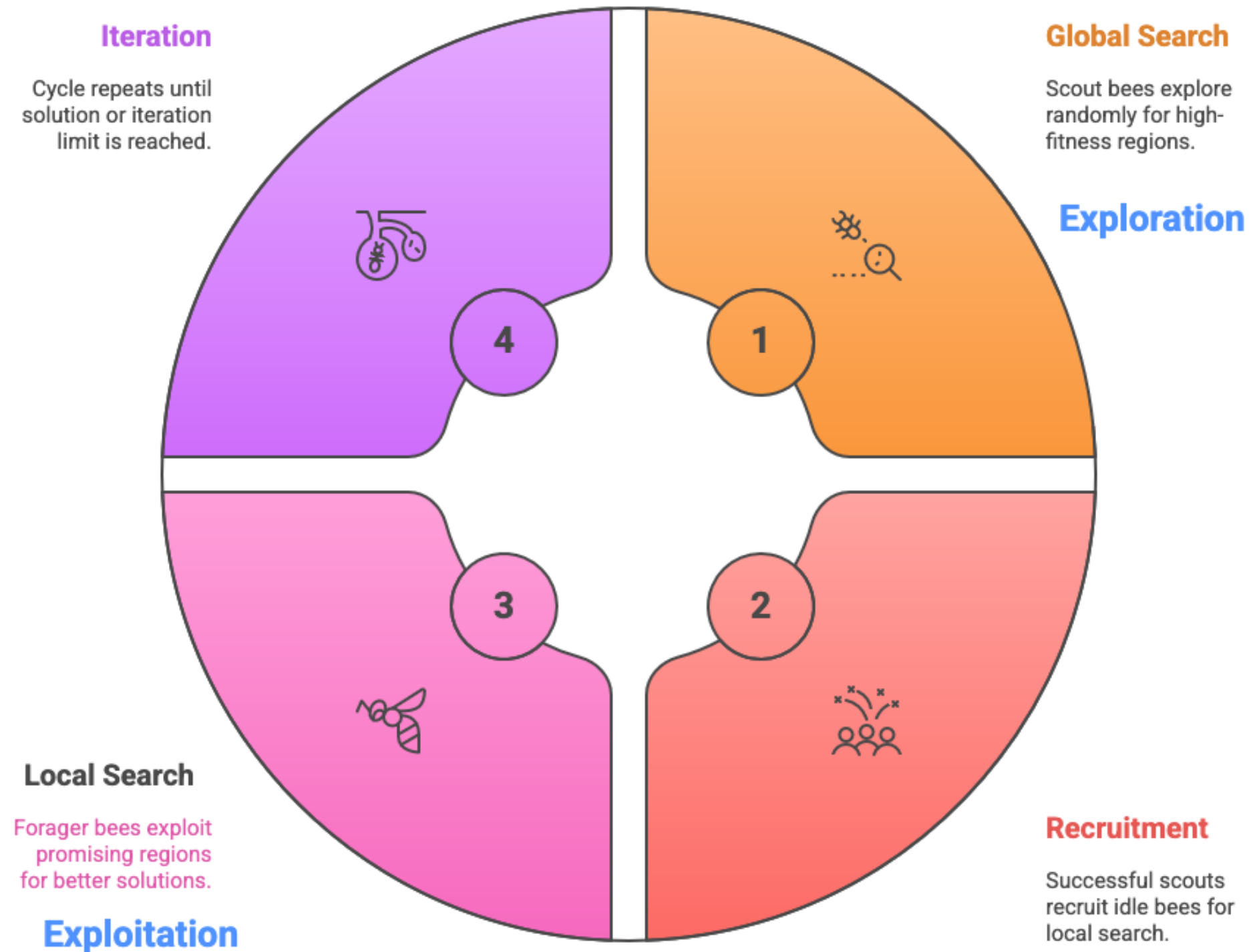
- The same patch will be advertised in the waggle dance again if it is still good enough as a food source and more bees will be recruited to that source.
- Thus, according to the fitness, patches can be visited by more bees or **may be abandoned**.



Bees in Nature: Summary



BA: Local Search vs Global Search

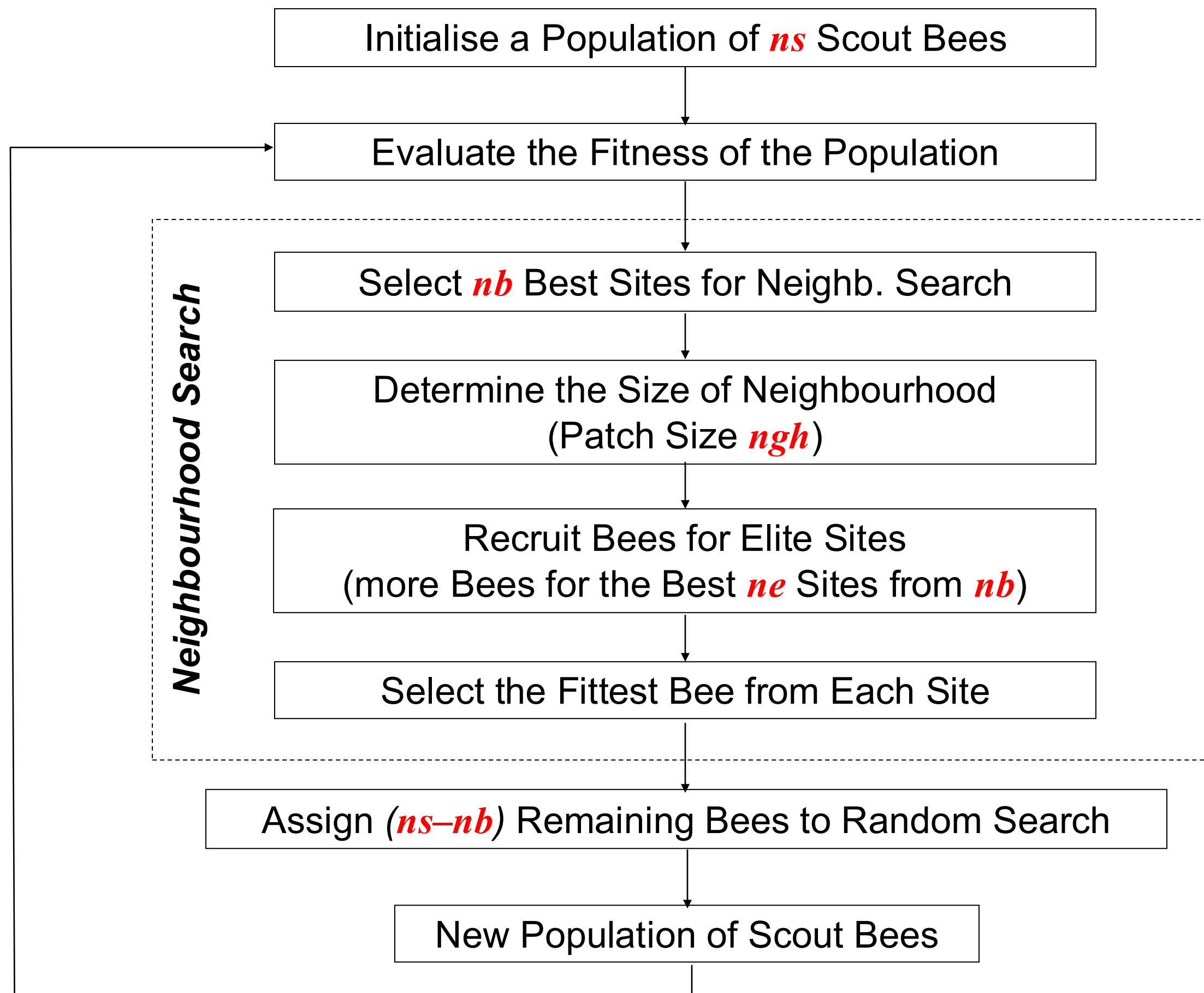


BA: Local Search vs Global Search

Algorithm Implementation:

- The main procedure includes local and global search routines that are executed concurrently during each optimization cycle.
- The good sites are *exploited* via the application of a local search,
 - although many scouts are sent out each iteration always in search of *additional good sites* (*exploration*)

Flowchart of the Basic BA



BA: Main Parameters

ns	Number of scout bees
ne	Number of elite sites
nb	Number of best sites
nre	Recruited bees for elite sites
nrb	Recruited bees for remaining best sites
ngh	Initial size of neighbourhood
$stlim$	Limit of stagnation cycles for site abandonment

- The algorithm uses a population of n artificial bees, divided into ns scouts and $n - ns$ followers.

Pseudocode: Standard BA

1 **for** $i=1, \dots, ns$

i scout[i]=Initialise_scout()

 ii flower_patch[i]=Initialise_flower_patch(scout[i])

- In the **initialisation** routine *ns* scout bees are randomly placed in the search space, and
 - Evaluate the fitness of the solutions where they land.
- For each solution, a **neighbourhood** (**called flower patch**) is delimited.

Pseudocode: Standard BA

1 for $i=1, \dots, ns$

 i scout[i]=Initialise_scout()

 ii flower_patch[i]=Initialise_flower_patch(scout[i])

2 do until stopping_condition=TRUE

 i Recruitment() \longrightarrow **Waggle Dance**

- In the **recruitment** procedure, the scouts that visited the $nb \leq ns$ fittest solutions (best sites) perform the waggle dance.
 - That is, they recruit followers to search further the neighborhoods of the most promising solutions.
- The scouts that located **the very best** $ne \leq nb$ solutions (**elite sites**) recruit nre followers each,
 - whilst the remaining $(nb - ne)$ scouts recruit nrb ($\leq nre$) followers each.
- Thus, the number of followers recruited depends on the profitability of the food source.

Pseudocode: Standard BA

```
1 for i=1,...,ns
    i scout[i]=Initialise_scout()
    ii flower_patch[i]=Initialise_flower_patch(scout[i])
2 do until stopping_condition=TRUE
    i Recruitment()

    ii for i =1,...,nb
        1 flower_patch[i]=Local_search(flower_patch[i])
        2 flower_patch[i]=Site_abandonment(flower_patch[i])
        3 flower_patch[i]=Neighbourhood_shrinking(flower_patch[i])
```

- In the **local search** procedure, the recruited followers are randomly scattered within the flower patches enclosing the solutions visited by the scouts (**local exploitation**).
 - If any of the followers in a flower patch lands on a solution of higher fitness than the solution visited by the scout, that follower becomes the new scout.

Pseudocode: Standard BA

1 for $i=1, \dots, ns$

 i scout[i]=Initialise_scout()

 ii flower_patch[i]=Initialise_flower_patch(scout[i])

2 **do until stopping_condition=TRUE**

 i Recruitment()

ii for $i=1, \dots, nb$

 1 flower_patch[i]=Local_search(flower_patch[i])

 2 flower_patch[i]=Site_abandonment(flower_patch[i])

 3 flower_patch[i]=Neighbourhood_shrinking(flower_patch[i])

} Not in Original BA Version

- If no follower finds a solution of higher fitness, the size of the flower patch is shrunk (**neighbourhood shrinking** procedure).
 - Usually, flower patches are initially defined over a large area, and their size is gradually shrunk by the neighbourhood shrinking procedure.
- If no improvement in fitness is recorded in a given flower patch for a pre-set number of search cycles, the local maximum of fitness is considered found, the patch is abandoned (**site abandonment**), and
 - A new scout is randomly generated.

Pseudocode: Standard BA

```
1 for i=1,...,ns
    i scout[i]=Initialise_scout()
    ii flower_patch[i]=Initialise_flower_patch(scout[i])
2 do until stopping_condition=TRUE
    i Recruitment()
    ii for i =1,...,nb
        1 flower_patch[i]=Local_search(flower_patch[i])
        2 flower_patch[i]=Site_abandonment(flower_patch[i])
        3 flower_patch[i]=Neighbourhood_shrinking(flower_patch[i])
    iii for i = nb,...,ns
        1 flower_patch[i]=Global_search(flower_patch[i])}
```

- As in biological bee colonies, a small number of scouts continue to explore the solution space, looking for new regions of high fitness (**global search**).
- The global search procedure re-initialises the last $ns-nb$ flower patches with randomly generated solutions.

Pseudocode: Standard BA

```
1 for i=1,...,ns
    i scout[i]=Initialise_scout()
    ii flower_patch[i]=Initialise_flower_patch(scout[i])
2 do until stopping_condition=TRUE
    i Recruitment()
    ii for i =1,...,nb
        1 flower_patch[i]=Local_search(flower_patch[i])
        2 flower_patch[i]=Site_abandonment(flower_patch[i])
        3 flower_patch[i]=Neighbourhood_shrinking(flower_patch[i])
    iii for i = nb,...,ns
        1 flower_patch[i]=Global_search(flower_patch[i])}
```

- At the end of one search cycle, the scout population is again composed of ns scouts:
 - nb scouts produced by the local search procedure (some of which may have been re-initialised by the site abandonment procedure), and
 - $ns-nb$ scouts generated by the global search procedure.

BA: Example

- The algorithm starts with the ns scout bees being placed randomly in the search space.
 - (for example $ns=100$)

BA: Example

- Fitnesses of the sites visited by the scout bees after return are evaluated (*fitness function evaluation*).
- The evaluation of the 100 scout bees is stored in array as follow:

1	2	3	4	5	6	99	100
20	50	60	30	80	10	35	72

- Then the array will be reordered based on the evaluation from the higher to the lower value.

BA: Example

- The nb best sites will be selected from ns .
- For example $nb = 10$

1	2	3	4	5	6	7	8	9	10
80	78	75	72	69	66	65	60	59	58

elite sites (ne)

remaining best sites ($nb-ne$)

- Then we choose the best ne site (elite bee) out off nb .
 - For example $ne = 2$

BA: Example

- A neighborhood search sites of size *ngh* is determined which will be used to update the *nb* bees declared in the previous step.
- Recruit Bees for the selected sites and evaluate the fitness of the sites.
 - Number of bees (*nre*) will be selected to be sent to *ne* sites (*nre* = 40).
 - Choosing *nrb* bees which their number is less than *nre*, (*nrb* = 20) to be sent to *nb* - *ne* sites.

BA: Example

- Choosing the best bee (the highest fitness) from each site (*from both elite sites and the remaining best sites*) to form the next bee population.

Example: *(Implementation: creating neighborhood for best sites)*

- The best bee from each of *ne* sites is selected as follow:
 - For the first site from *ne*:
 - An array contains *nre = 40* bees will be constructed,
 - where the value of each bee is equal to the value of the original scout bee with a little modification depending on the neighborhood *ngh*.

Example: (Implementation: creating neighborhood for best sites)

- Based on the fitness evaluation of $nre = 40$ bees:
 - The results will be stored in temporary array.
 - The array will be ordered and the best value will be taken

1	2	3	...	40
82	81.2	79.9	...	79.2

Example: *(Implementation: creating neighborhood for best sites)*

- It is repeated for all *nb* sites.
- At the end we will get the best *nb* = **10** bees which will be stored at the beginning of the array (*ns* = **100**)

1	2	3	4	5	6	...	10	11	...	99	100
82	79	77	73	70	67	...	58.2				

BA: Example

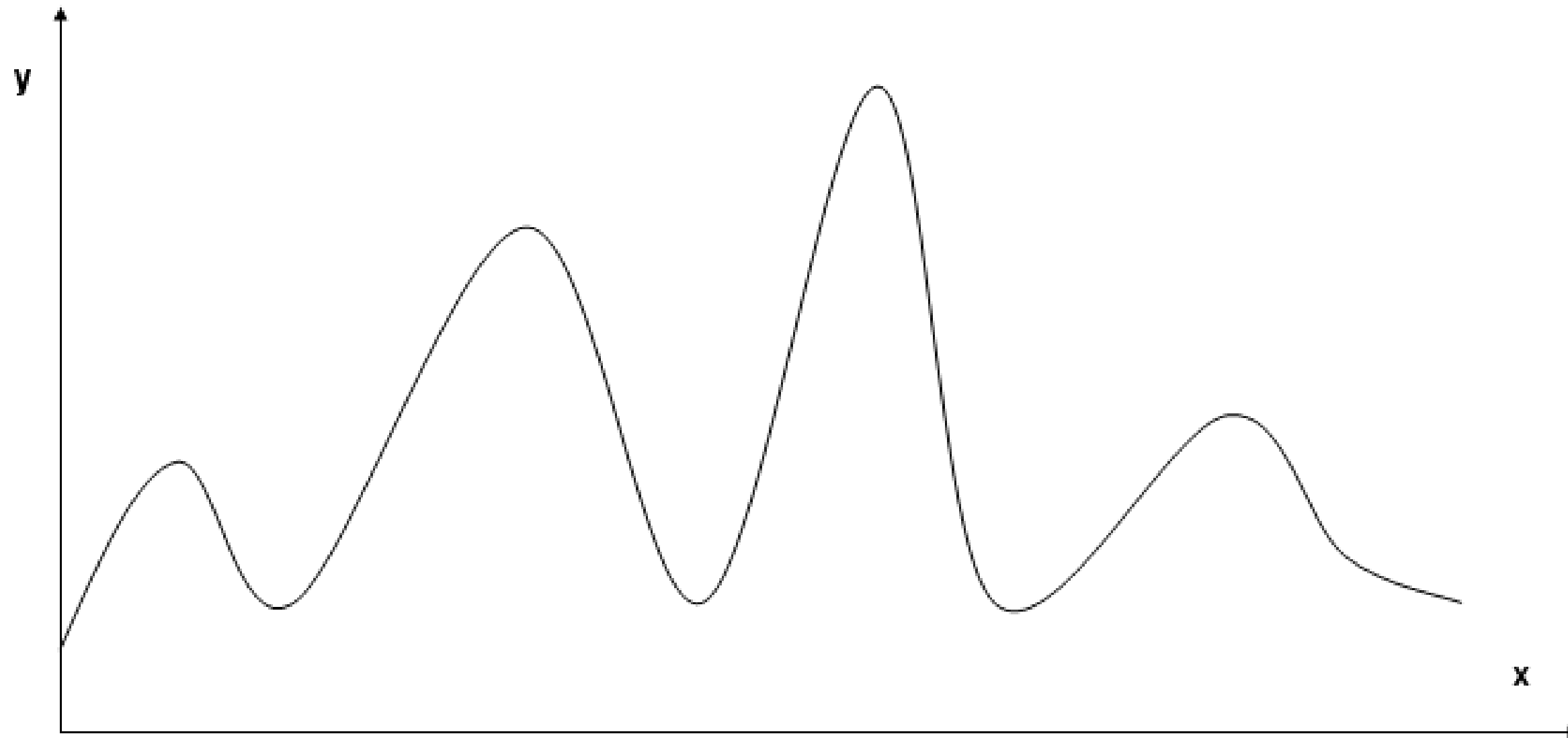
- Initials new population:
 - The remaining bees ($ns - nb$) in the population will be assigned randomly around the search space (values from 11 to 100 in the previous array)
 - The new population becomes as follow:

1	2	3	4	5	6	...	10	11	...	99	100
82	79	77	73	70	67	...	58.2	Random values			

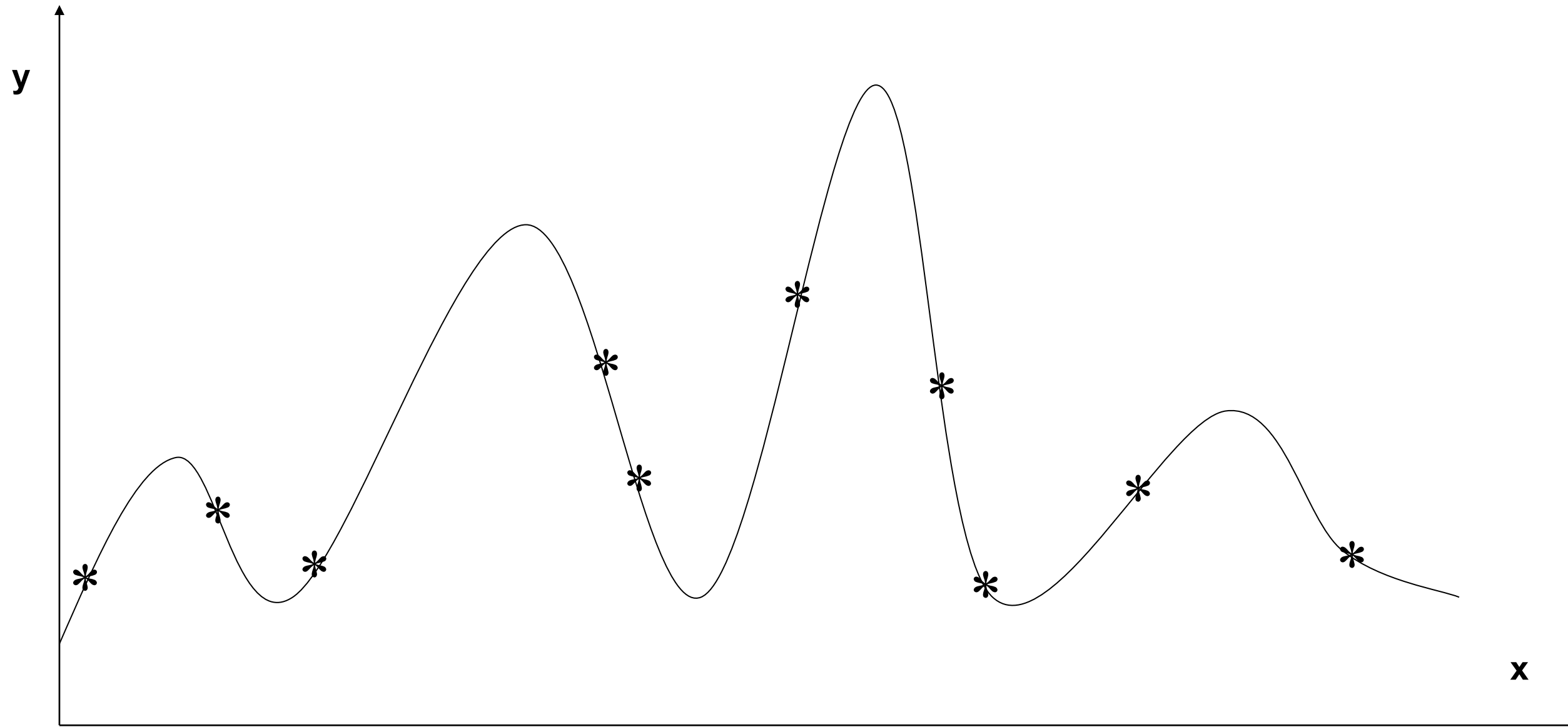
nb $ns - nb$

Example: Function Optimisation

- The following figure shows the mathematical function.

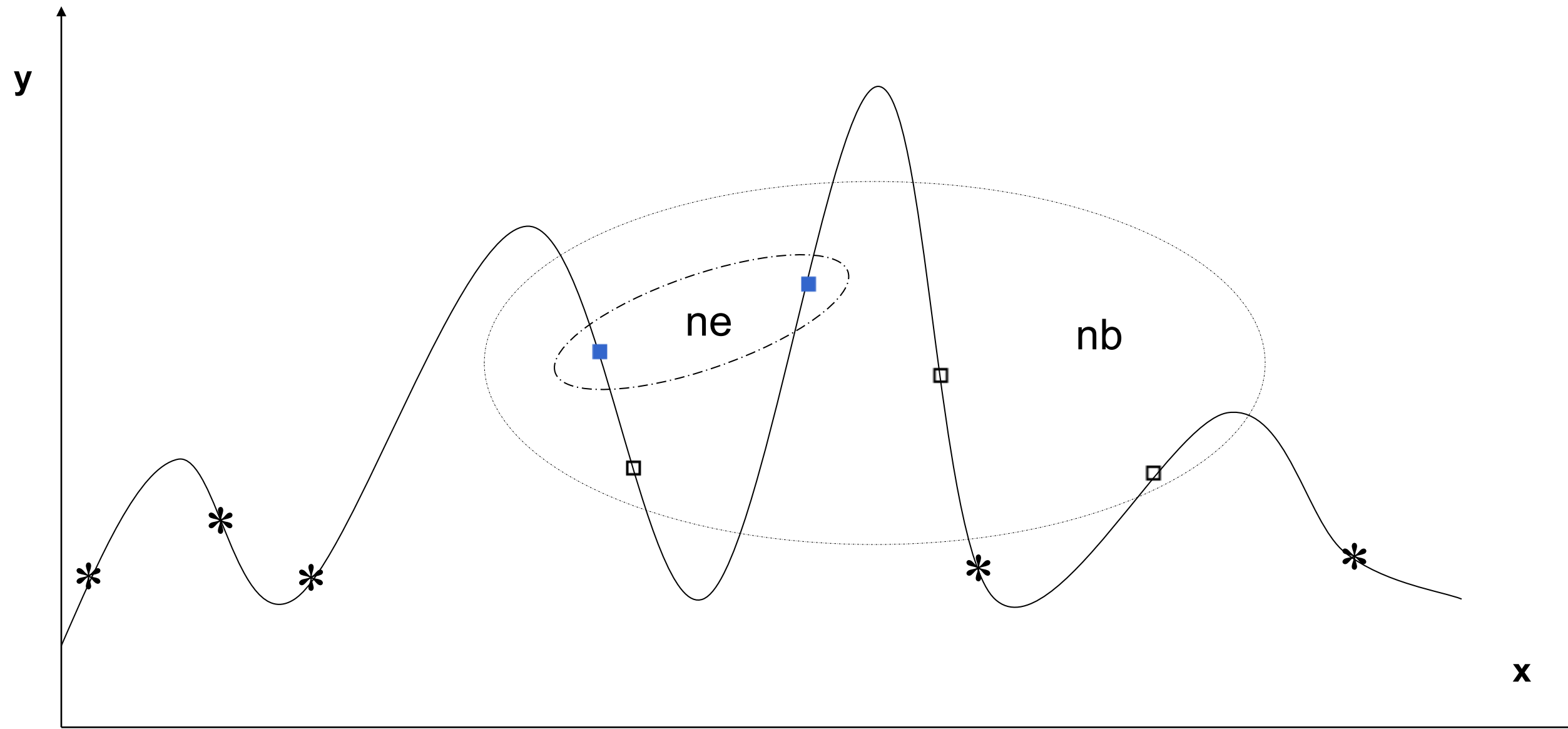


Example: Function Optimisation



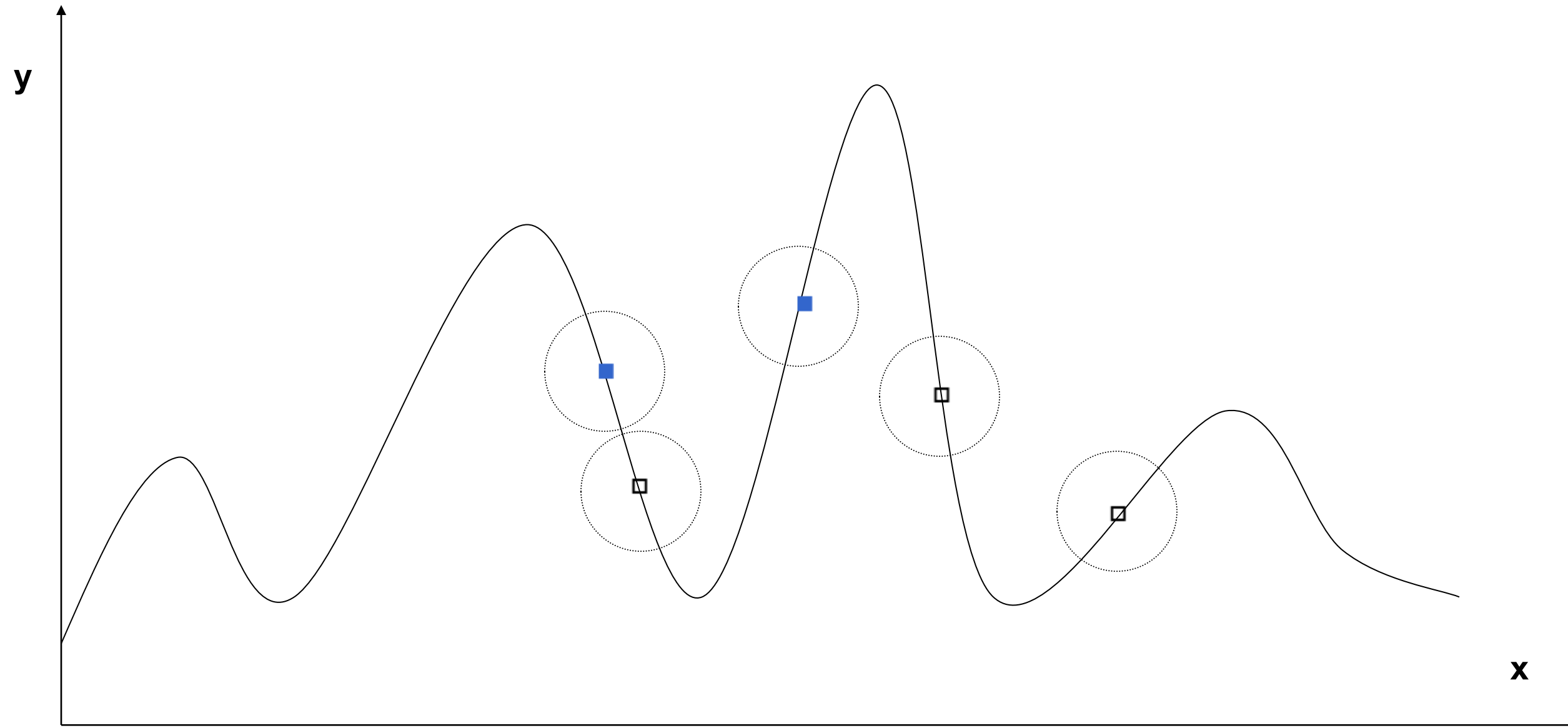
Initialise a Population of ($ns=10$) Scout Bees with random Search and evaluate the fitness.

Example: Function Optimisation



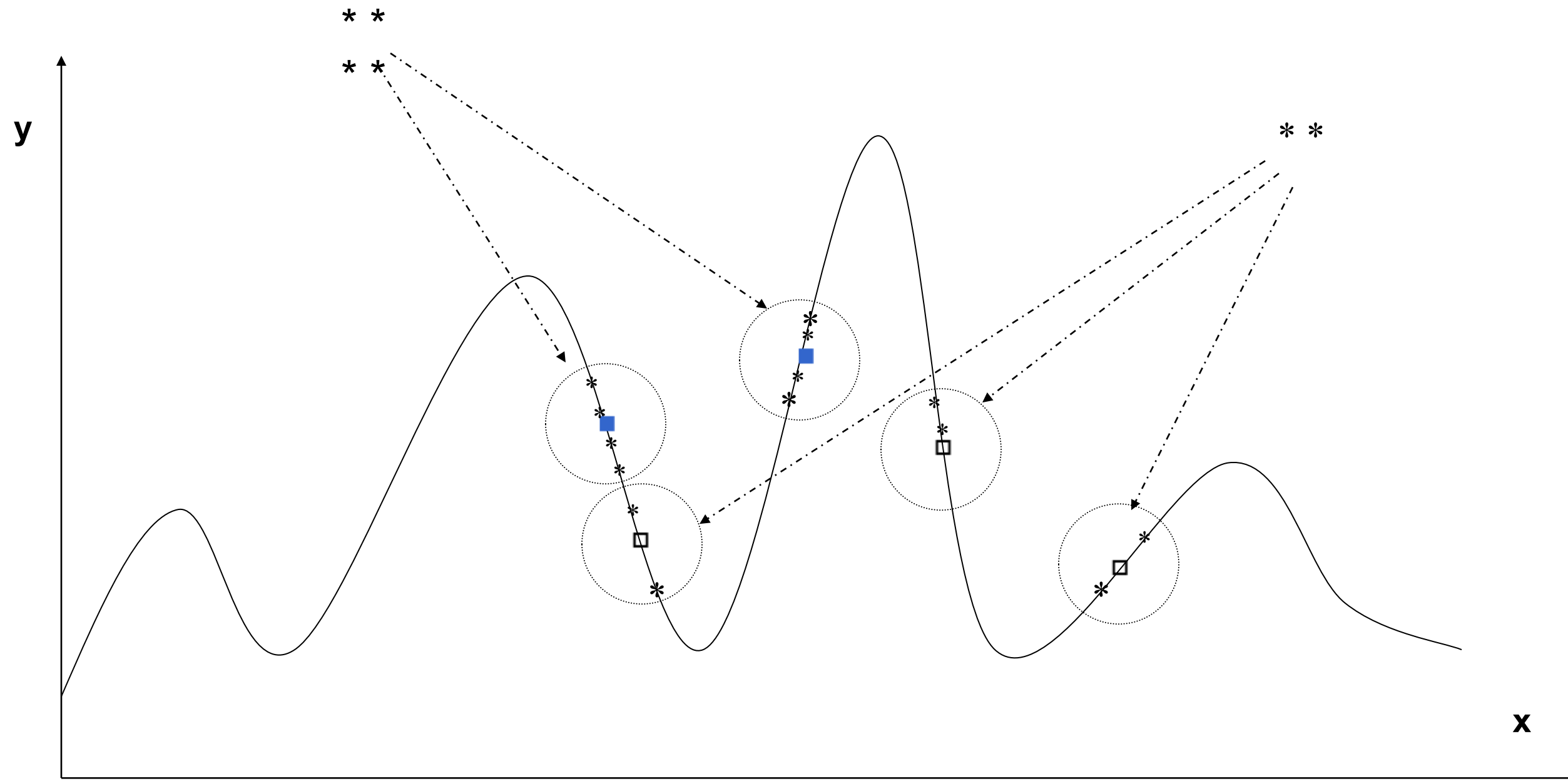
Select best ($nb=5$) Sites for Neighbourhood Search:
($ne=2$) elite bees “■” and ($nb-ne=3$) other selected bees “□”

Example: Function Optimisation



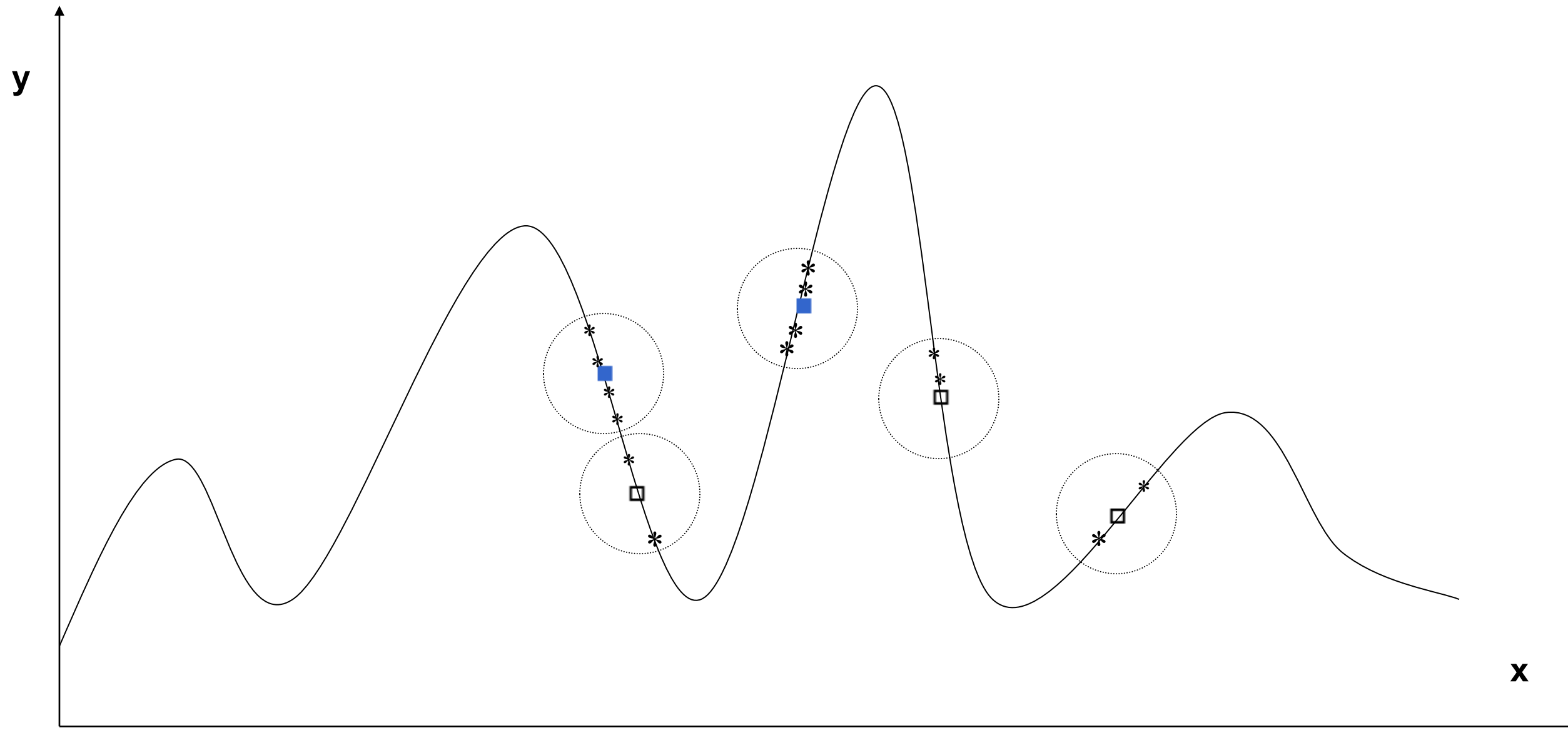
Determine the Size of Neighbourhood (Patch Size *ngh*)

Example: Function Optimisation



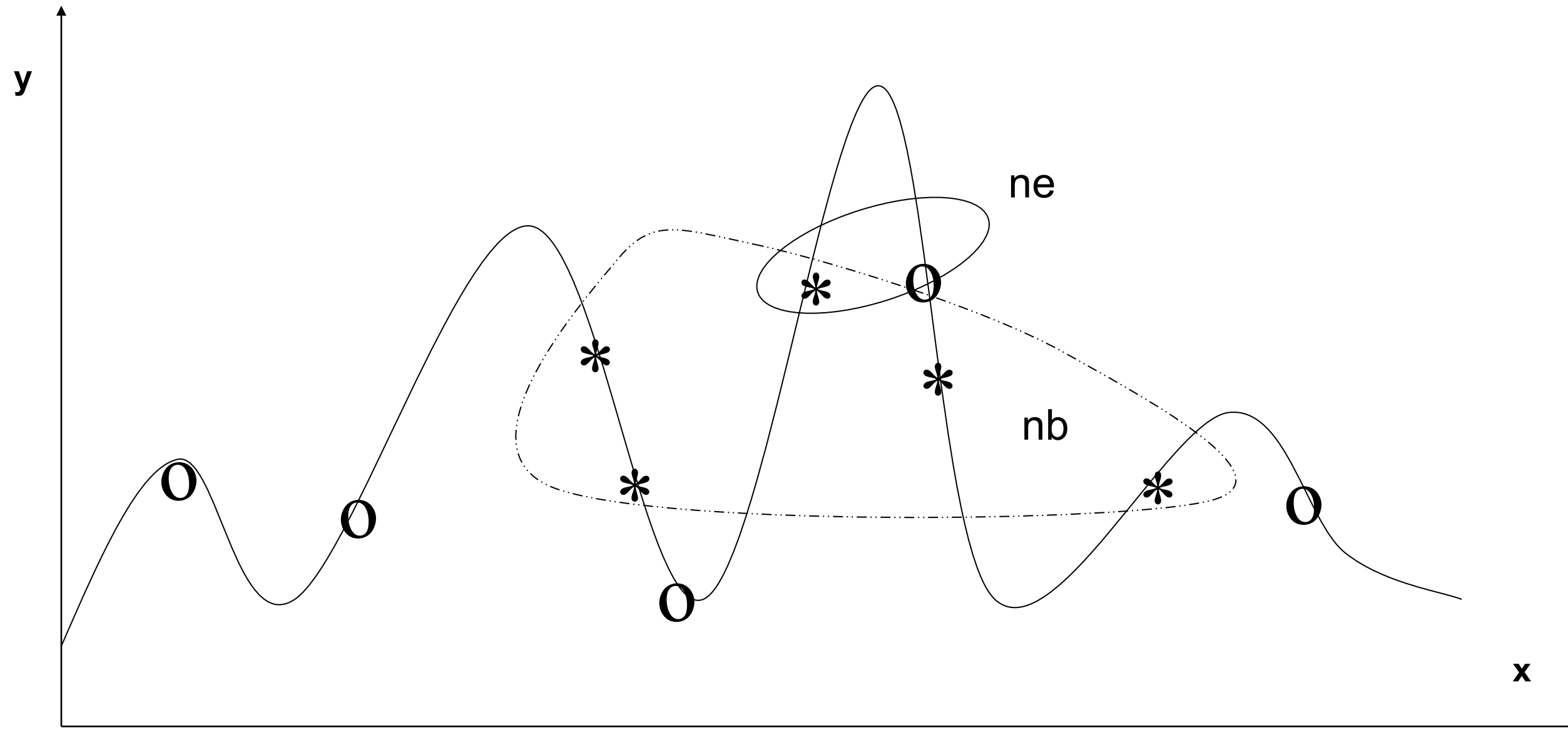
Recruit Bees for Selected Sites
(more Bees for the $ne=2$ Elite Sites)

Example: Function Optimisation



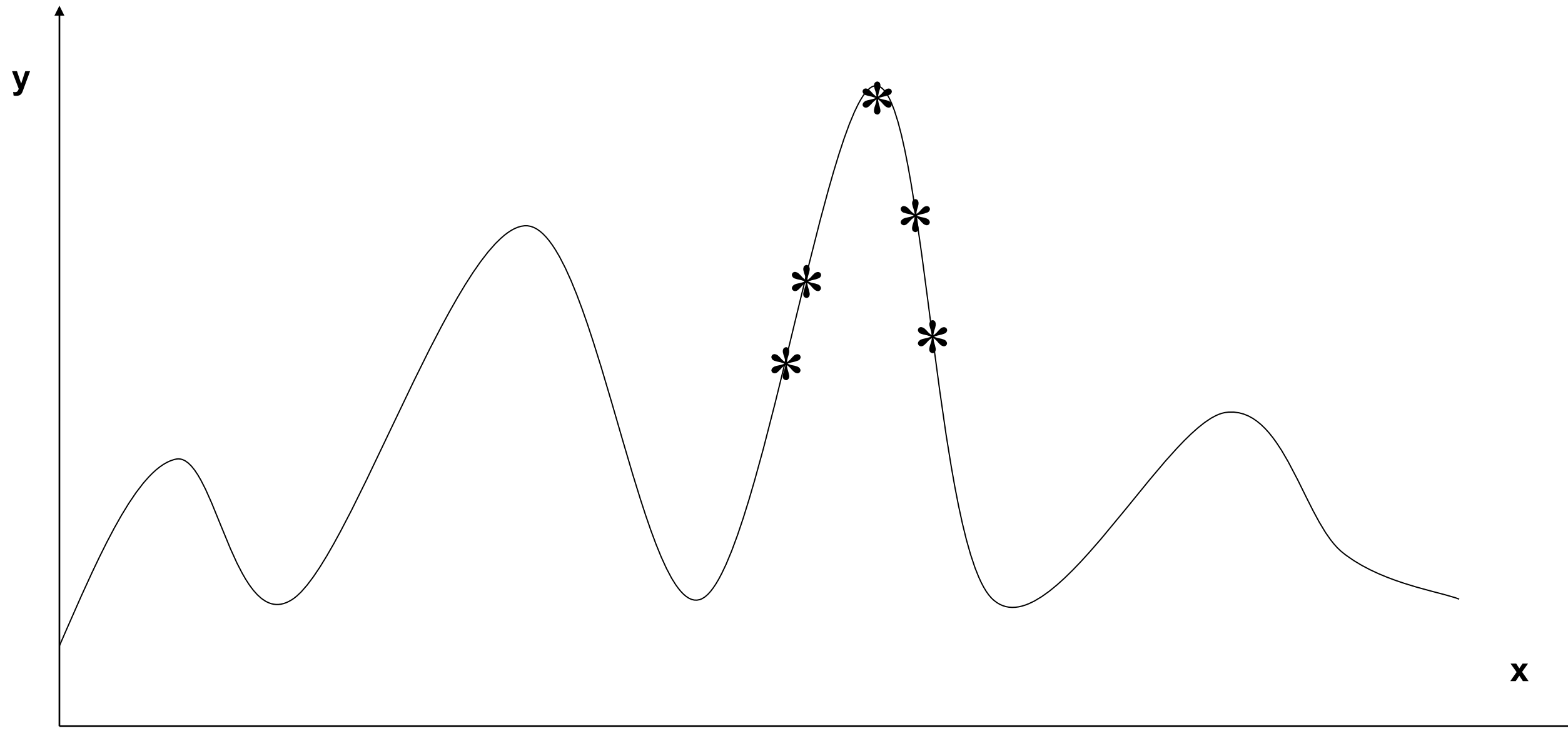
Select the Fittest Bee * from Each Site

Example: Function Optimisation



Assign the $(ns - nb)$ Remaining Bees to Random Search

Example: Function Optimisation



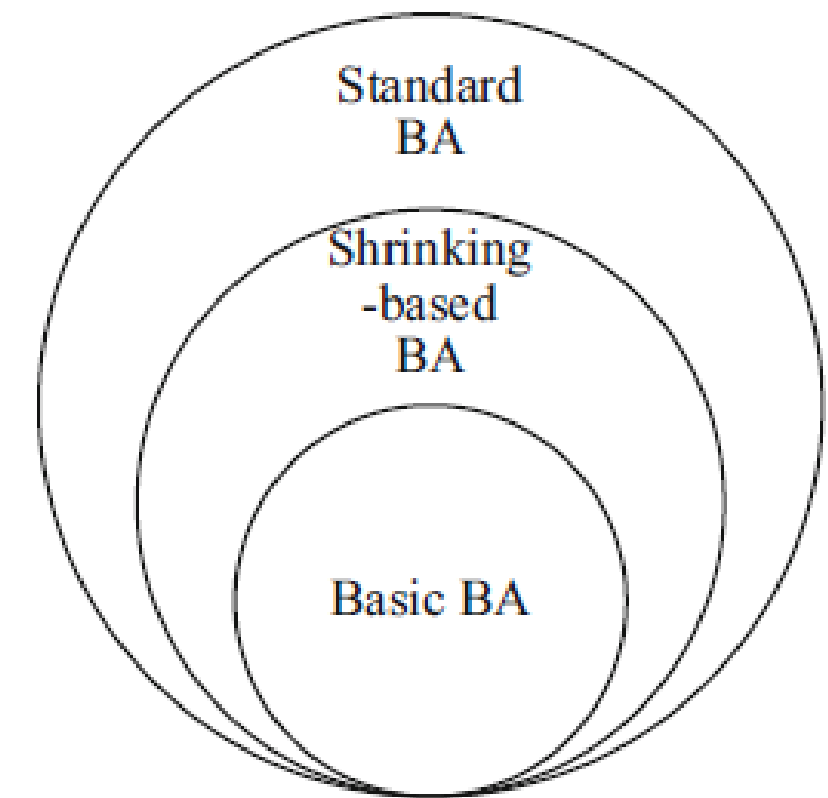
Find The Global Best point

BA: Variants

- BA can be divided into four parts:
 1. parameter tuning
 2. Initialization
 3. local search
 4. global search.
- Several works to enhance the performance of BA by improving some of its parts.
- More than one version of the algorithm has been proposed.

BA: VAriants

- Three main variation (using several different formal names):
 1. Basic BA
 2. Shrinking-based BA
 3. Standard BA (Shrinking + site abandonment)
- Various implementations of the shrinking and site-abandonment procedures are explored and incorporated into BA to constitute different BA implementations.



Neighbourhood Shrinking (*in improved version*)

- If no forager improved the fitness of the solution found by the scout, the size of the neighbourhood (flower patch) is shrunk.
- The neighborhood shrinking mechanism aims to focus progressively the search in a narrow area around the fitness peak, and is akin to the SA.
- At each cycle of stagnation, the size of the flower patch is customarily decreased using the following heuristic formula:

$$a(t+1)=0.8 \cdot a(t)$$

Site Abandonment (in improved version)

- If the local search procedure fails to bring any fitness improvement in a flower patch for $stlim$ consecutive BA cycles,
 - The search is assumed to have found the local fitness optimum.
- In this case, the flower patch is abandoned, and
 - A new scout bee is re-initialized at a random location in the search space.

BA: Pros and Cons

Pros:

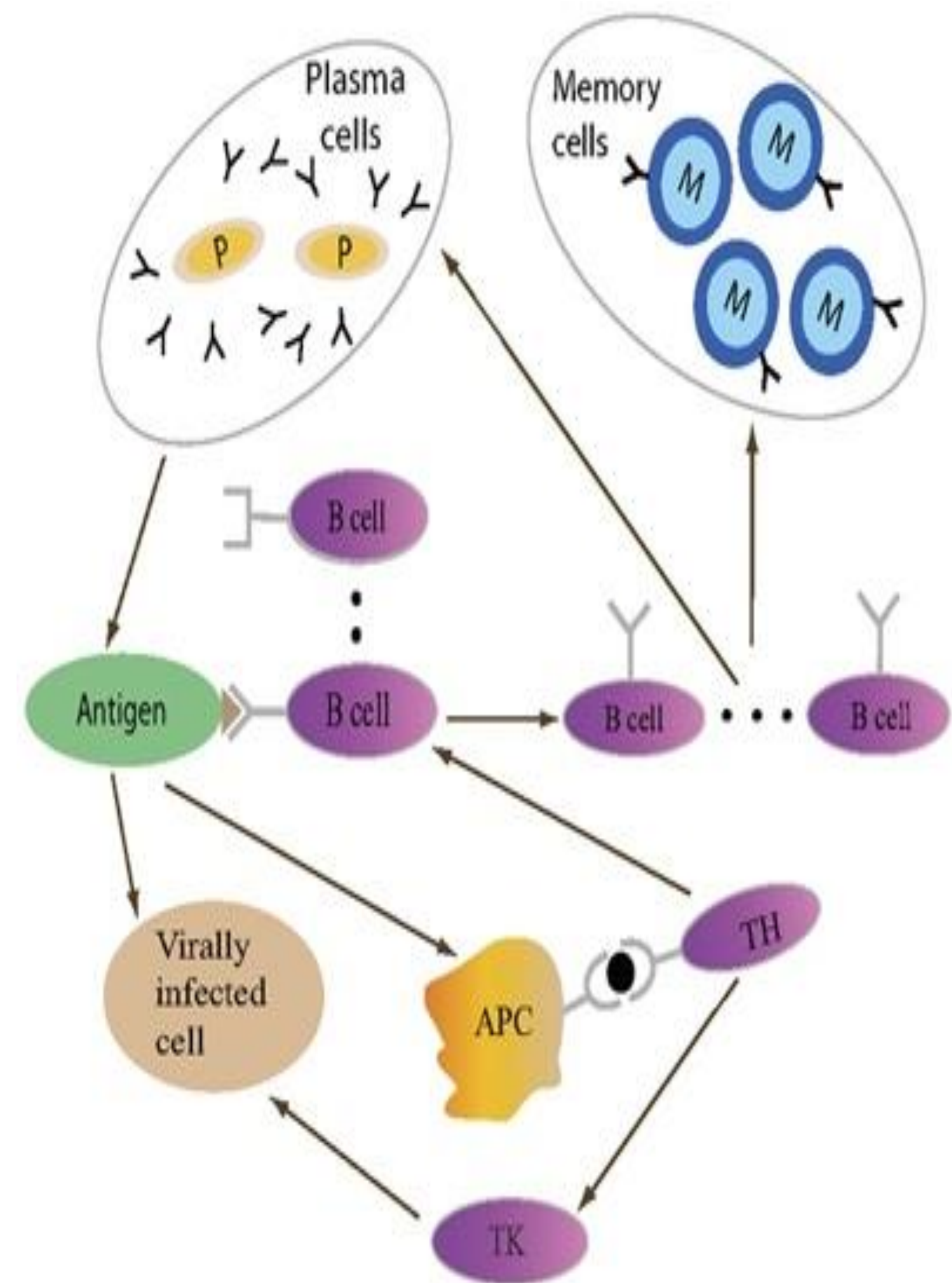
- Very efficient in finding optimal solutions.
- Overcoming the problem of local optima.

Cons:

- Several tunable parameters.

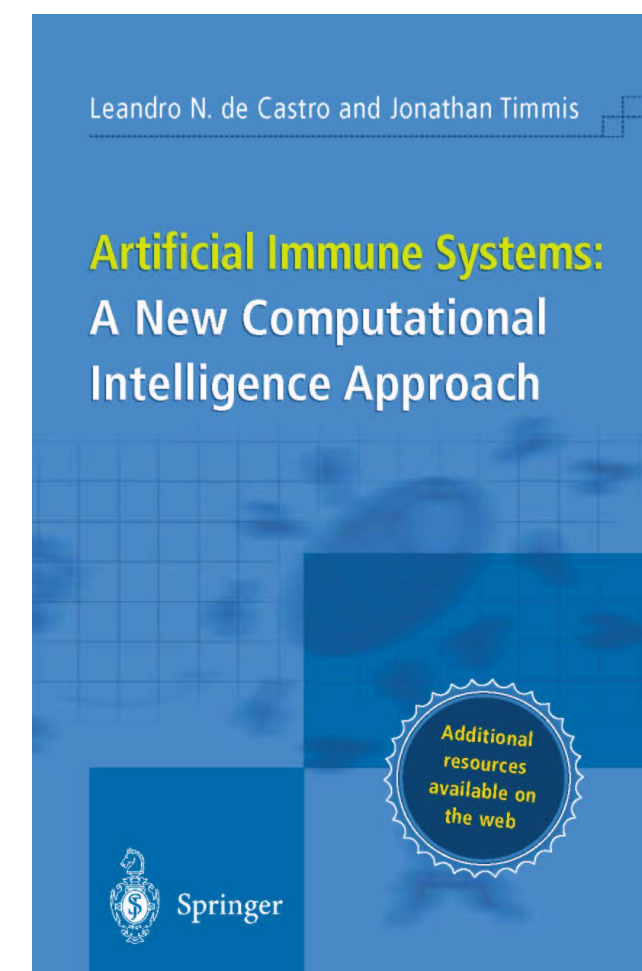
<http://beesalgorithmssite.altervista.org/index.html>

Artificial Immune System (AIS)



Artificial Immune System (AIS)

- Artificial immune systems (AIS) are intelligent and adaptive systems inspired by the immune system toward real-world problem solving. (Dasgupta).
- AIS are adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving. (de Castro and Timmis).
- Relatively new branch of computational intelligence.
- **Not modelling the immune system.**

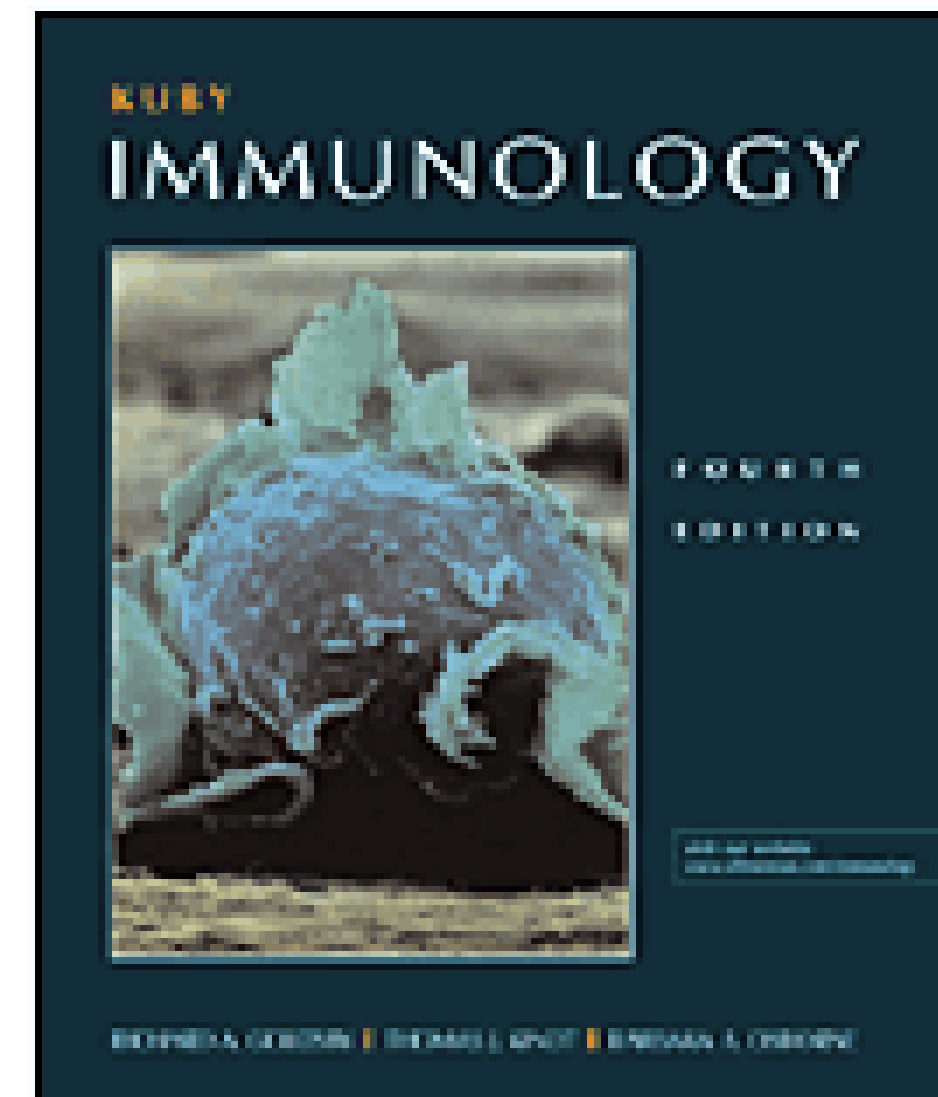


* De Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach*. Springer Science & Business Media.

- Developed from the field of theoretical immunology in the mid 1980's.
- Bersini first use of immune algorithms to solve problems in 1990.
- Forrest et al. (mid 1990's) – Computer Security.
- Hunt et al. (mid 1990's) – Machine learning.

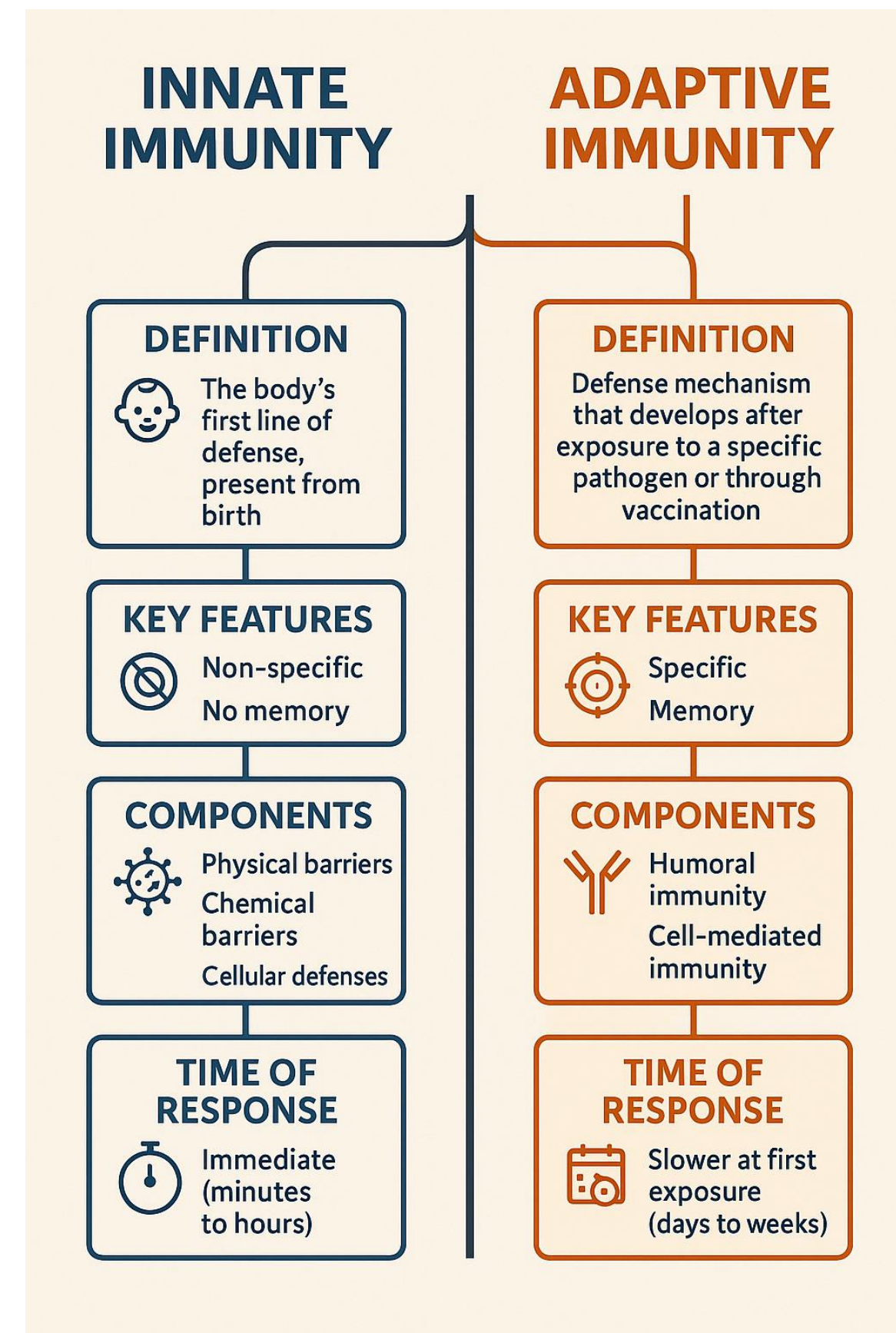
Biological Immune System (BIS)

- **BIS:** The BIS is a robust, complex, highly parallel, distributed, and adaptive system that defends the body from foreign pathogens.
- **Immune system:** A system having the primary function of distinguishing **self from not self** and protects our body **from foreign substances and pathogenic organisms** by producing the immune response.
- **Immunity:** State or quality of being resistant (immune), either by virtue of previous exposure (**adaptive immunity**) or as an inherited trait (**innate immunity**).



Different Immune Systems

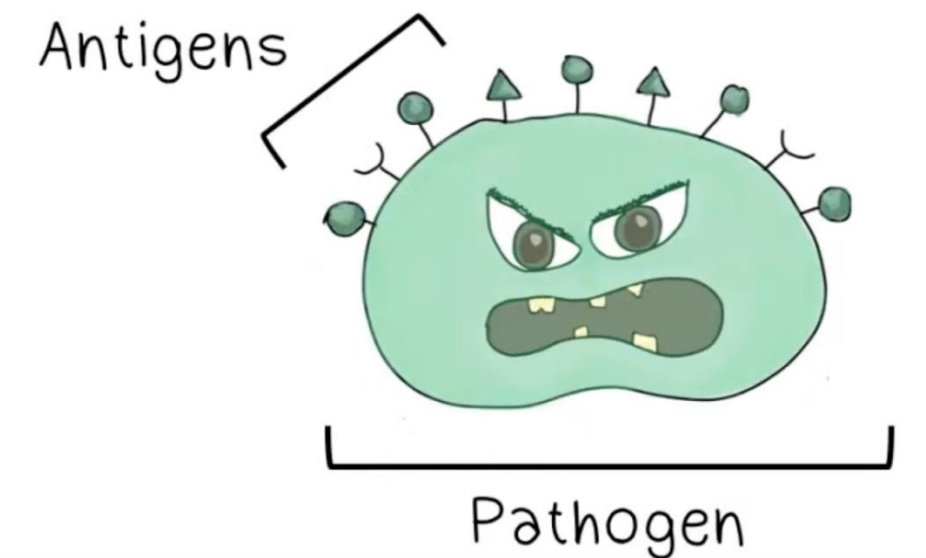
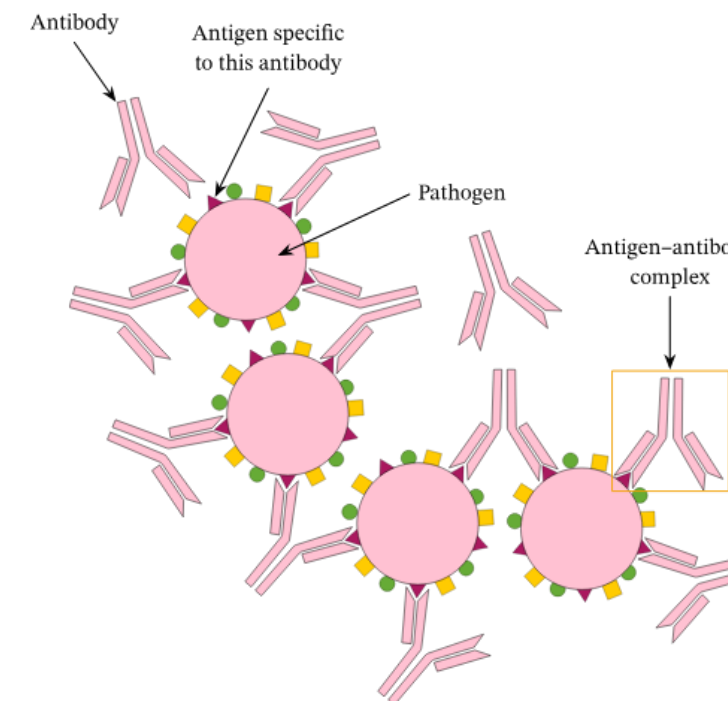
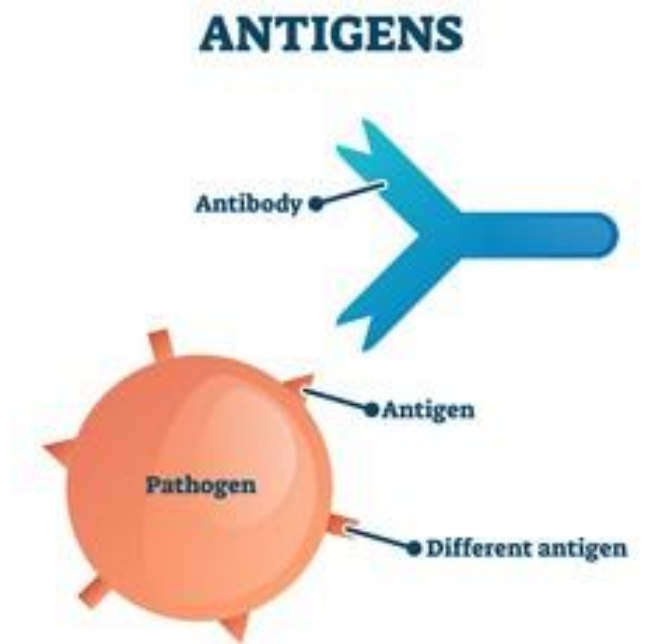
- Primary immune response (**non-specific / innate**)
 - Do not distinguish between one threat and another.
 - Are present at birth.
- Secondary immune response (**specific / adaptive**)
 - Responds to previously unknown foreign cells.
 - Protect against specifically identified threats.
 - Most develop after birth upon exposure to an Antigen.
 - Learning, adaptability, and memory are important characteristics of adaptive immunity (**main focus of interest**).



- BIS is
 - Robust.
 - Complex.
 - Highly parallel.
 - Distributed.
 - Adaptive.
- It uses **learning**, **memory**, and **associative retrieval** to solve recognition and classification tasks.
- In particular, it learns
 - to recognize relevant patterns,
 - remember patterns that have been seen previously, and
 - use combinatorics to construct pattern detectors efficiently.

OSLOMET Pathogen

- A pathogen is an object that can make you sick.
- A harmful organism or agent, such as bacteria, viruses, fungi, or parasites, that can cause disease.
- Pathogens invade the body and trigger an immune response, including the production of **antibodies against antigens** present on the pathogen's surface.
- The surface of a pathogen is covered in foreign **antigens**.



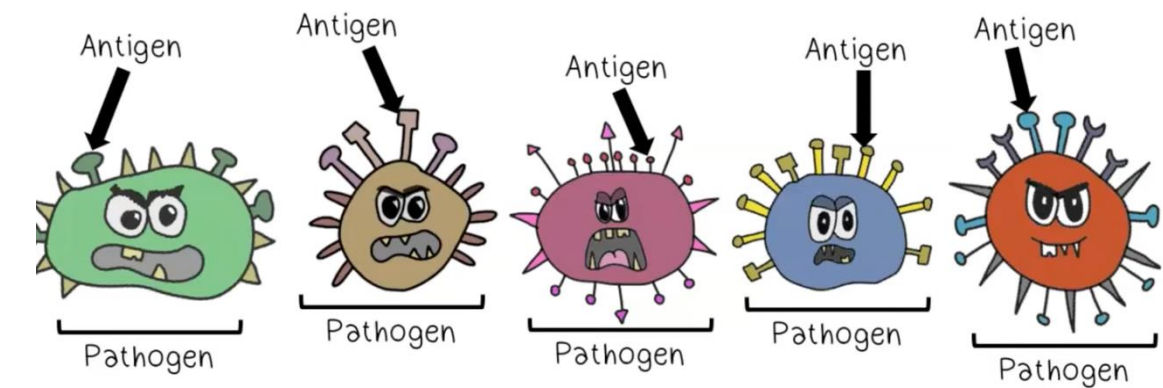
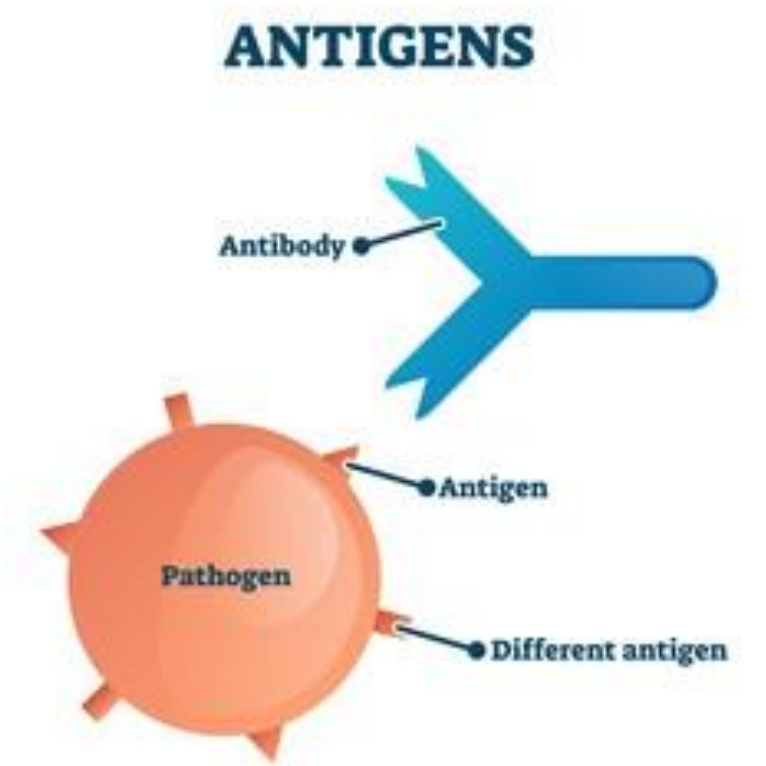
OSLOMET Antigen

- An **antigen** is any molecular feature that the immune system can recognize.
 - **With the right danger signals**, it can trigger a response
- An **antigen** is a recognizable **feature**. It's **often on** a pathogen.
 - But it can also be from a pathogen,
 - or from non-pathogens.
- CS analogy: “Antigen = **feature vector** the classifier can match; only when flagged with ‘**malicious context**’ **metadata** (danger signals) do you launch the defense.



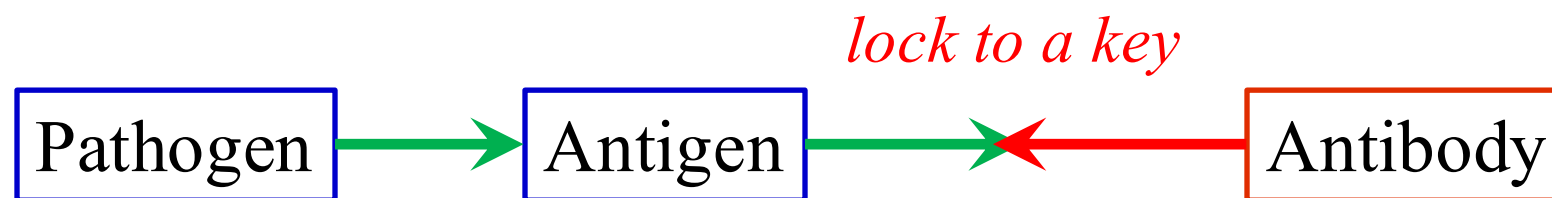
OSLOMET Antigen

- Antigens have very specific shapes.
 - Different types of pathogens have different antigens on their surface.
- A particular antigen stimulates a specific antibody that fights against the particular pathogen with which it is attached.
- One pathogen → many antigens;
 - The same antigen can appear on different pathogens (feature reuse)

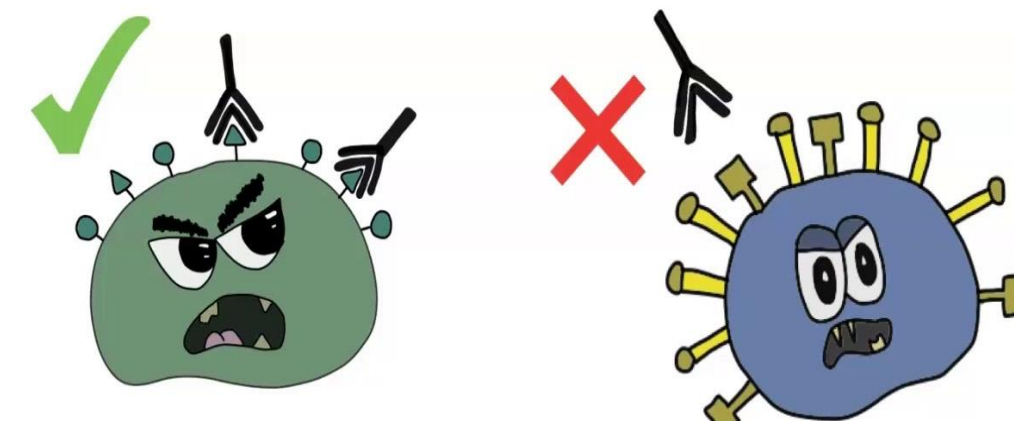
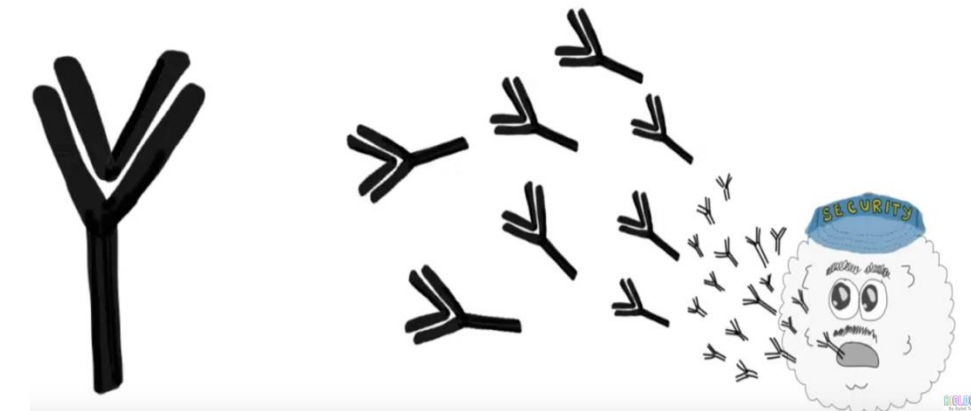


OSLOMET Antibody

- Antibodies are marking flags that latch on to antigens like a lock and key.
- Antibodies are tiny Y-shaped proteins your body (**B cells**/ white blood cell) makes to **recognize and stick to germs** (or their parts).
 - Once they stick, they **block the germ** or **flag it** so other immune cells (antigen) can destroy it.

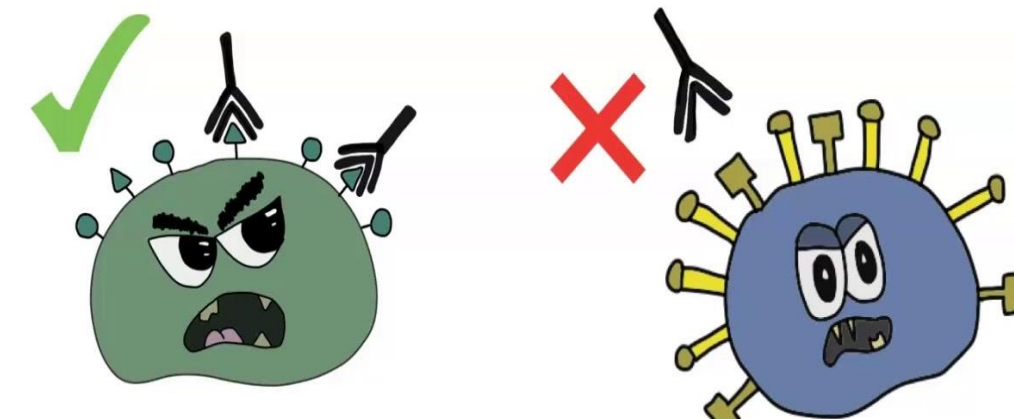
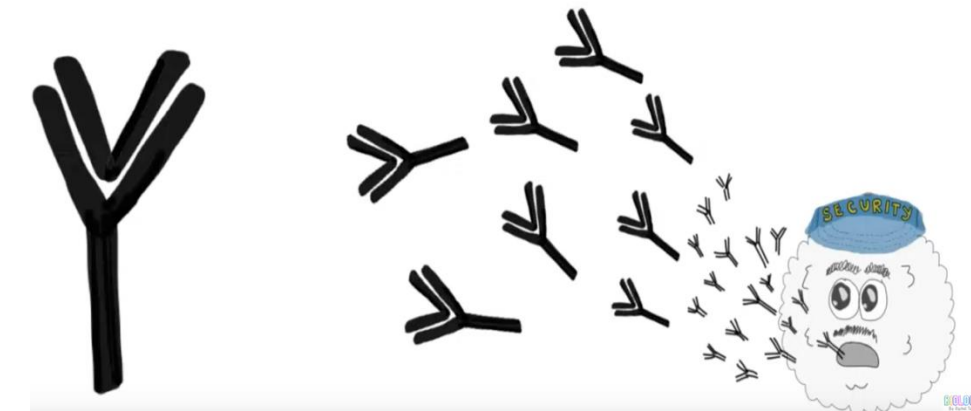


- Antibodies are made with a **SPECIFIC** shape to **MATCH** the antigen detected.



OSLOMET Antibody

- An antibody specifically **binds an antigen's epitope**—like a lock to a key—to help neutralize or mark the threat for removal.
- **Epitope (a.k.a. antigenic determinant):** The **exact part of an antigen** that an antibody or a T-cell receptor **binds**.
- CS analogy:
 - **Pathogen:** The whole folder.
 - **Antigen** = A particular file (whole file).
 - **Epitope** = the **exact signature chunk** the model matches.

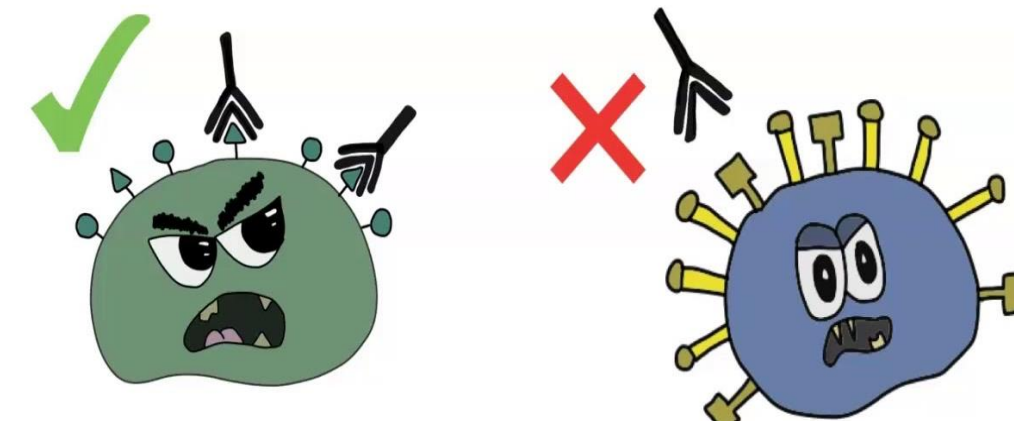
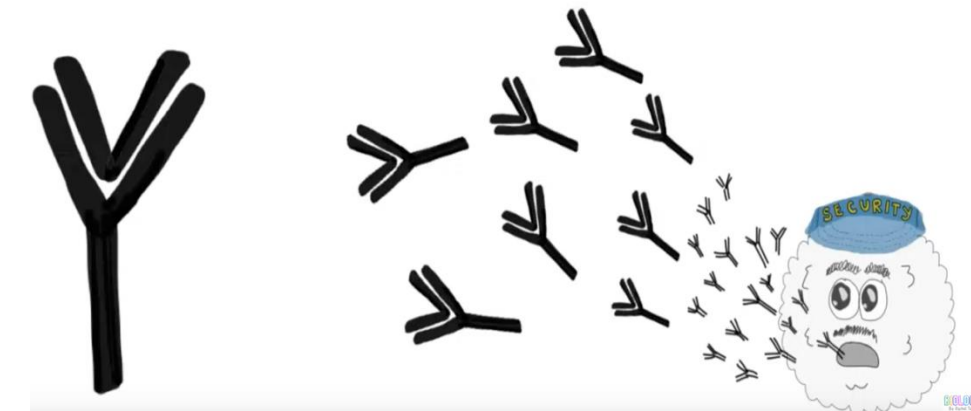
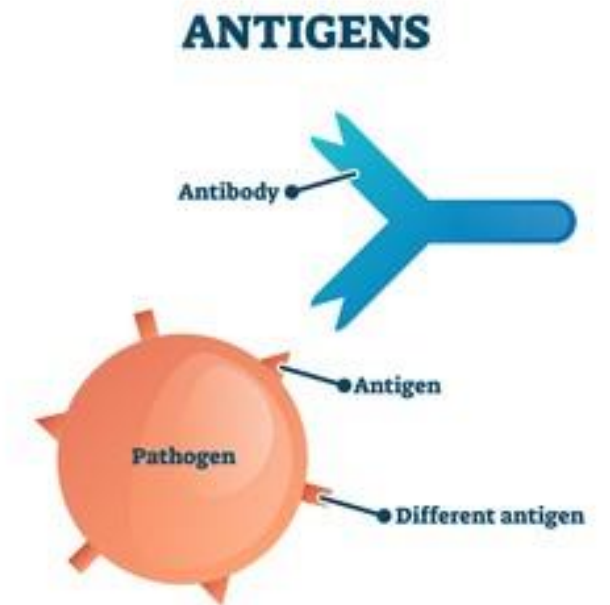


OSLOMET Antibody

- **Role:** neutralize toxins/viruses, tag targets for phagocytes (**opsonization**), and **activate complement**.
- **Specificity & memory:** each antibody is specific; after exposure, the body can make **more, faster** on re-exposure.
- **Where:** found in blood, lymph, and mucosal surfaces.

Rules of thumb:

- One **antigen** can have **many epitopes**.
- Different antibodies can target **different epitopes** on the same antigen.
- Size: antibody epitopes are small surface patches; T-cell epitopes are short peptides.



Pathogen, Antigen, Antibody



Pathogen

A disease-causing organism or agent.
influenza virus,
bacteria.



Antigen

A "tag" on (or from) something that the immune system can recognize
pollen, virus spike protein



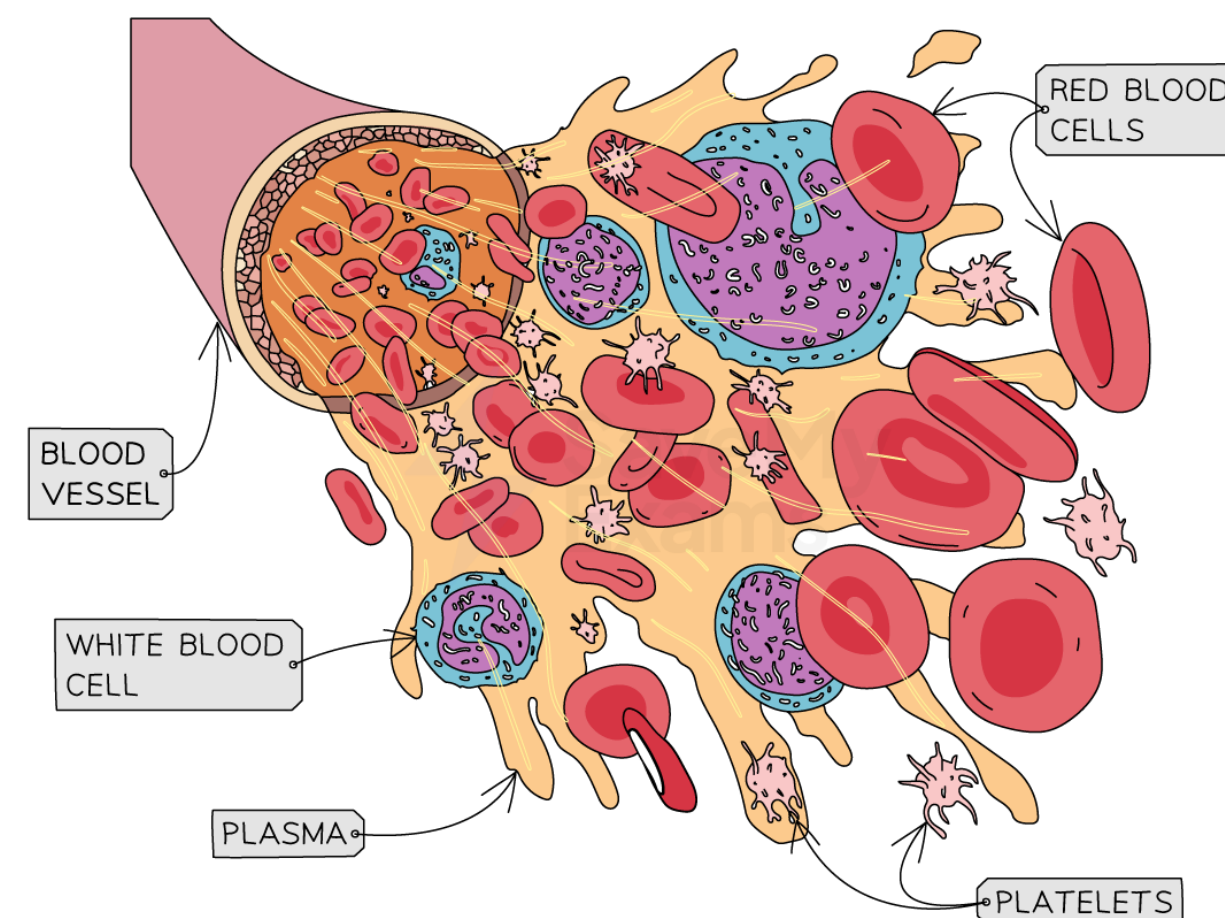
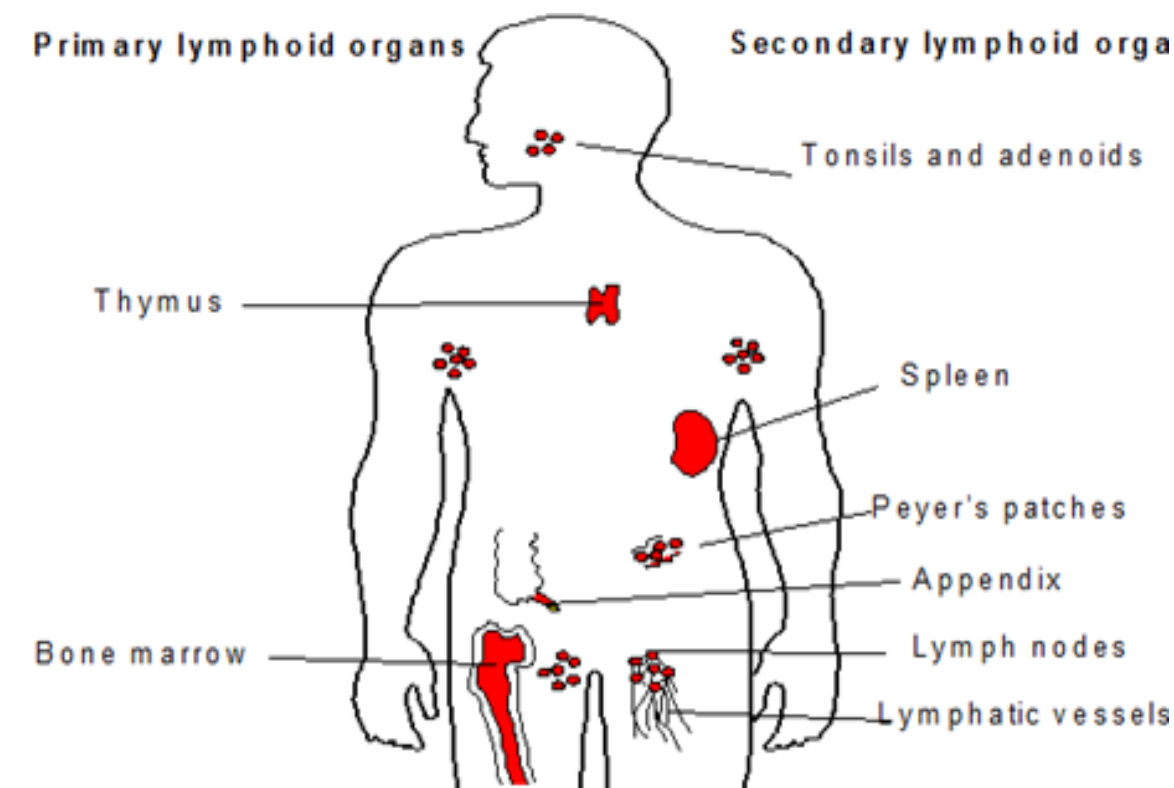
Antibody

A protein that specifically binds an antigen

Pathogen (the whole bug) → displays many antigens (recognition tags) → your immune system makes antibodies that bind those antigens → the pathogen is neutralized/cleared

White Blood Cells (WBCs)

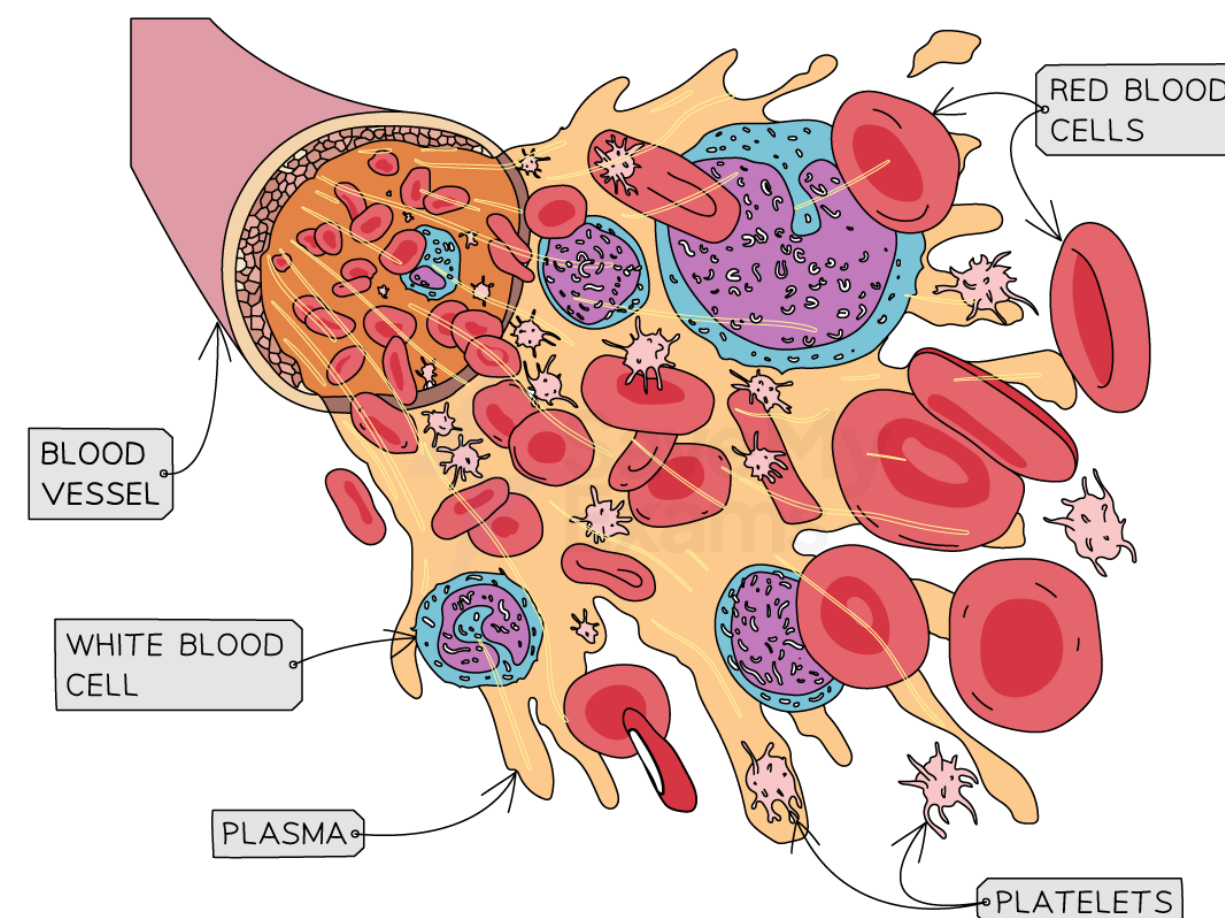
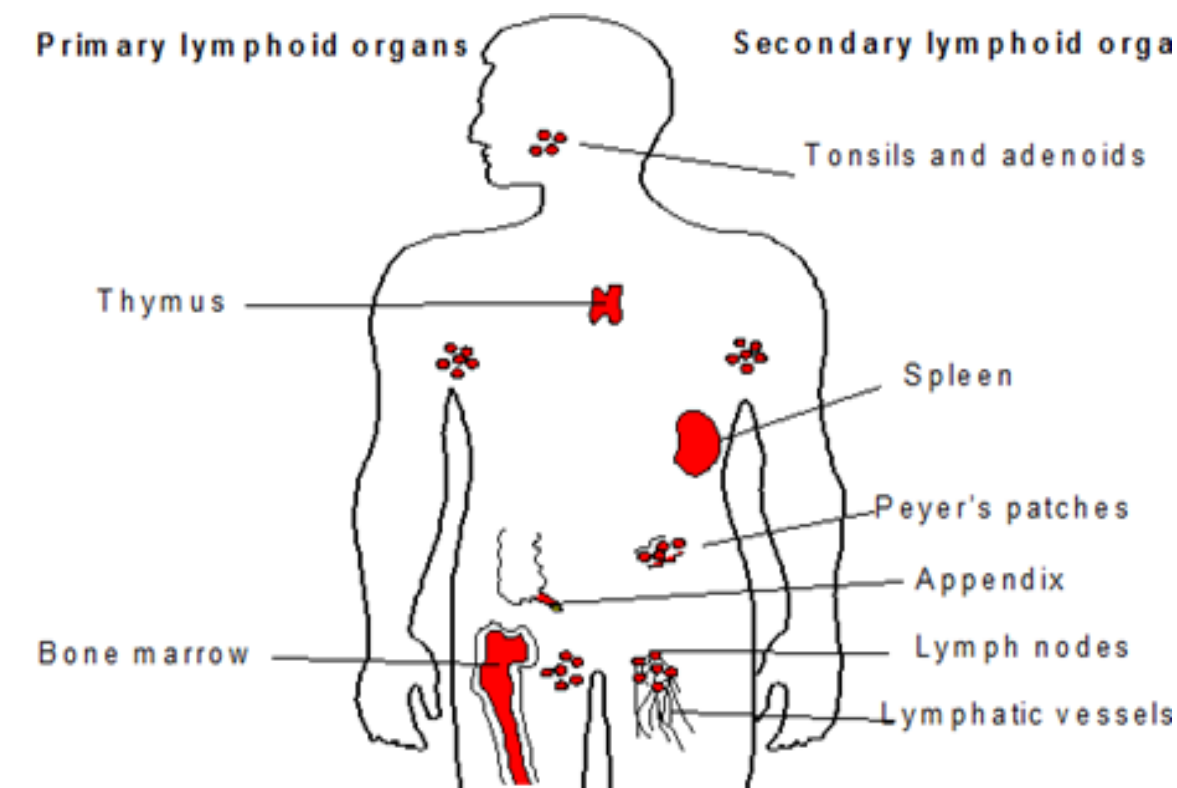
- **WBCs**, also called **leukocytes**, are the body's immune cells.
 - WBCs help the body fight infection and other diseases.
 - They patrol blood and tissues to **detect, fight, and clear** infections, abnormal cells, and debris.
- WBCs are a type of blood cell made in the **bone marrow** and found in the bloodstream and lymph tissue.
- **Where they work:** blood, lymph nodes, spleen, bone marrow, and tissues throughout the body.



Copyright © Save My Exams. All Rights Reserved

WBCs: Core Functions

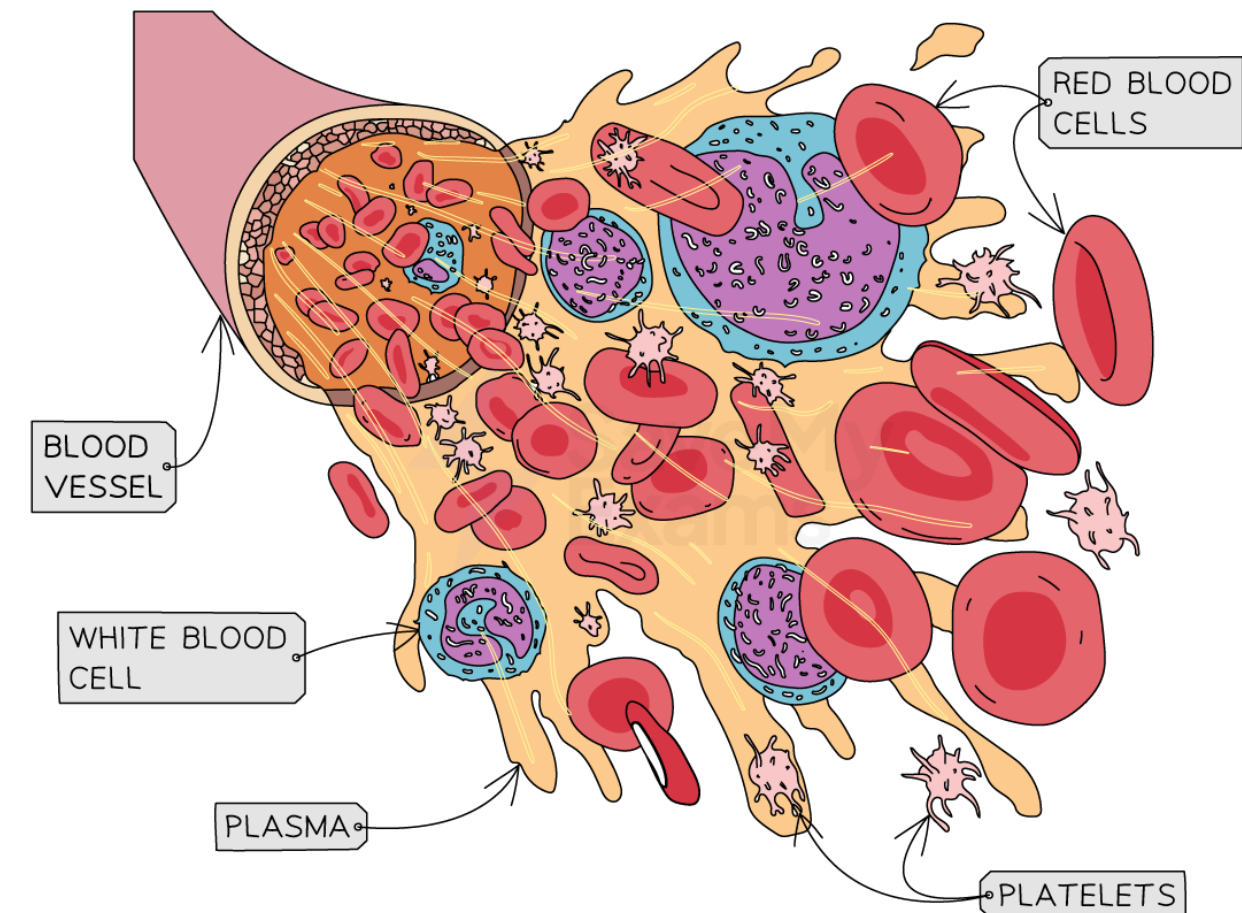
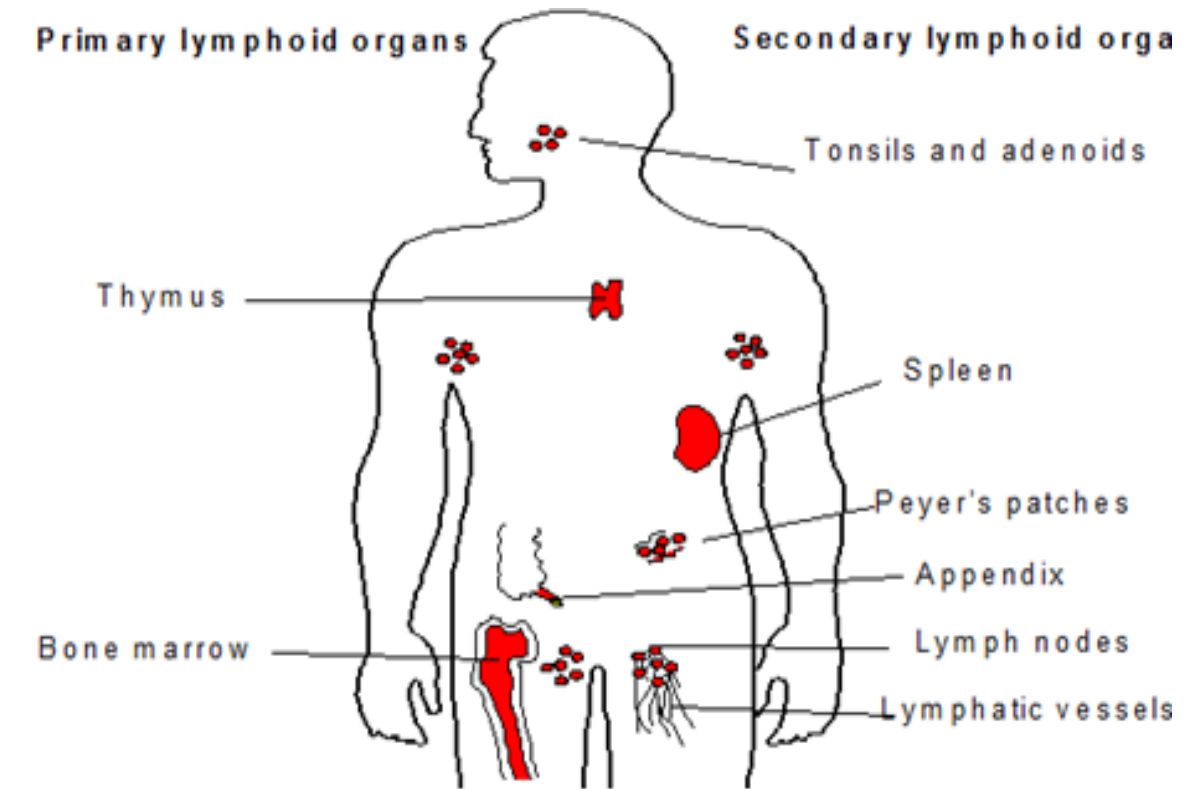
- **Detect & destroy invaders:** recognize bacteria, viruses, fungi, parasites.
- **Coordinate immunity:** send signals (cytokines) to ramp responses up/down.
- **Create memory:** “remember” past threats for faster future defense.
- **Clean up & repair:** remove dead cells and help wound healing.
- **Allergy & parasite defense:** specialized responses to worms and allergens.
- **Cancer surveillance:** identify and eliminate abnormal cells.



Copyright © Save My Exams. All Rights Reserved

WBCs: Main Types

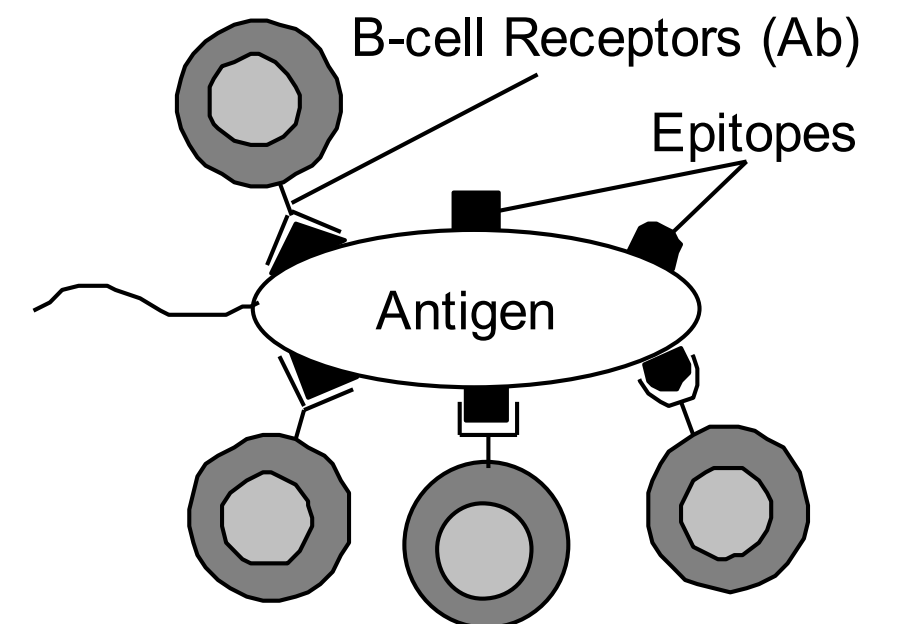
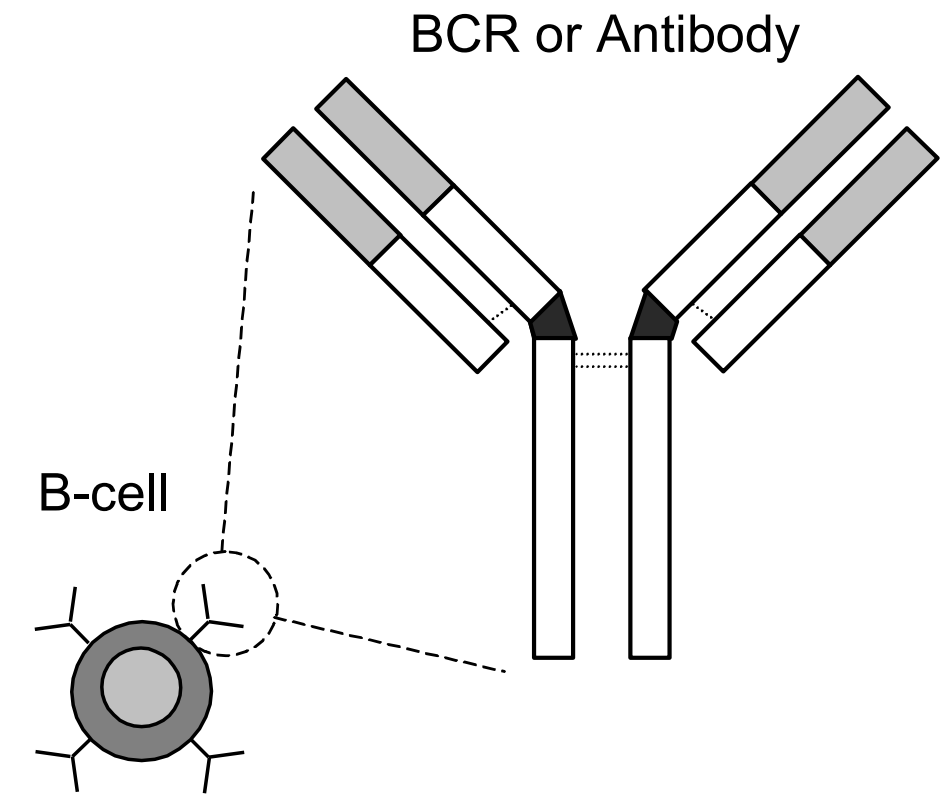
- Neutrophils (innate): first responders; ingest/kill bacteria.
- Lymphocytes (adaptive/innate-like):
 - **B cells:** make antibodies.
 - **T cells:** help coordinate ($CD4^+$) or kill infected/cancer cells ($CD8^+$).
 - NK cells: kill virus-infected and cancer cells without prior “training.”
- Monocytes (innate): big eaters; present antigens to T cells; tissue cleanup/repair.
- Eosinophils (innate): attack parasites; involved in allergy/asthma.
- Basophils (innate): release histamine; drive allergic inflammation.



B-Cells and T-Cells as Lymphocytes

B-Cells (B Lymphocytes):

- Make **antibodies** (humoral immunity).
- Responsible for **producing antibodies** to neutralize pathogens.
- Each B-cell can only produce one particular antibody.
- When B-cells encounter an antigen, they can differentiate into
 1. **Plasma Cells** that produce large amounts of antibodies, enable neutralization, opsonization, complement activation. or
 2. **Memory B-cells** that “remember” the antigen for future immune responses.

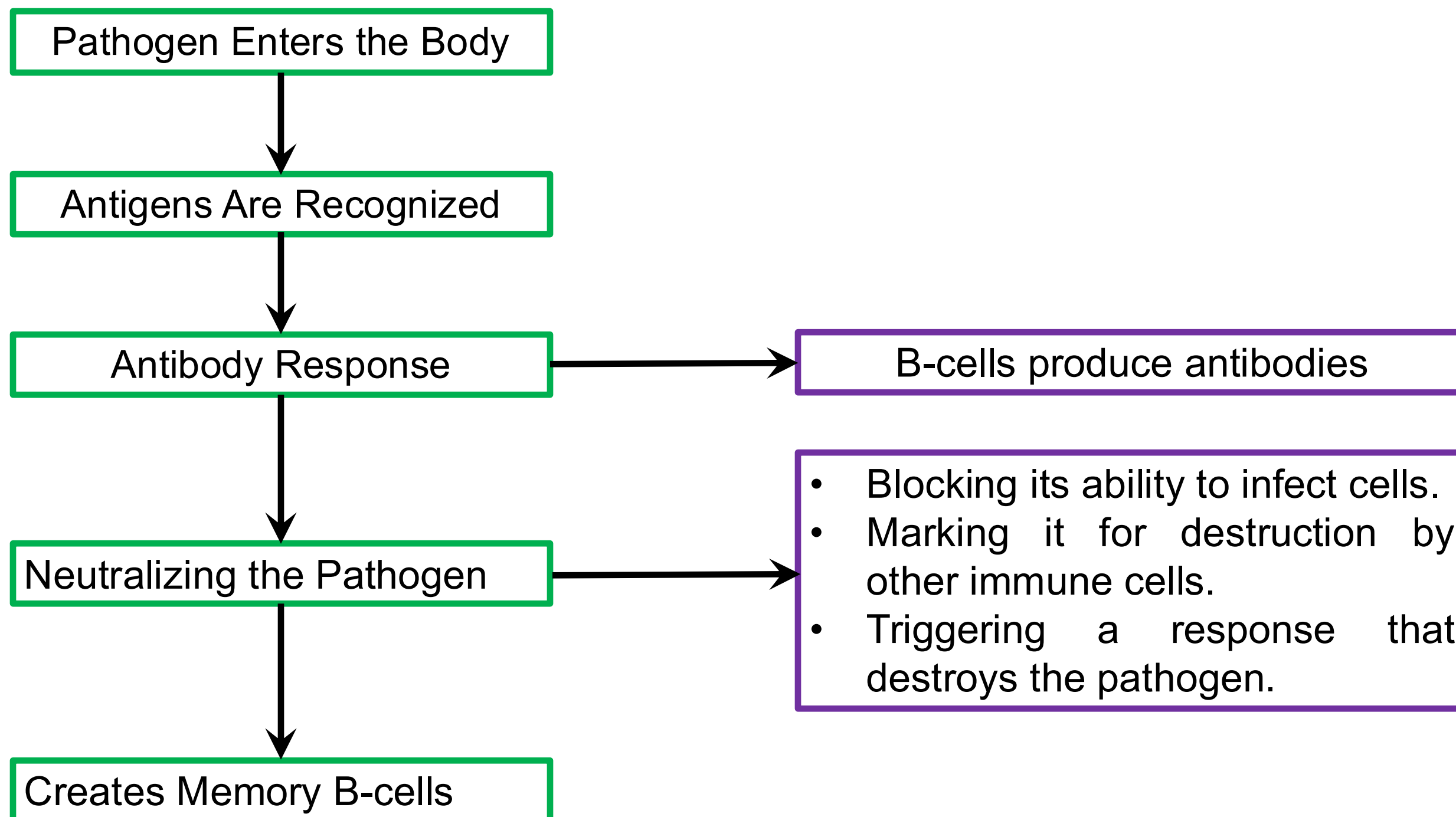


B-Cells and T-Cells as Lymphocytes

T-Cells (T Lymphocytes):

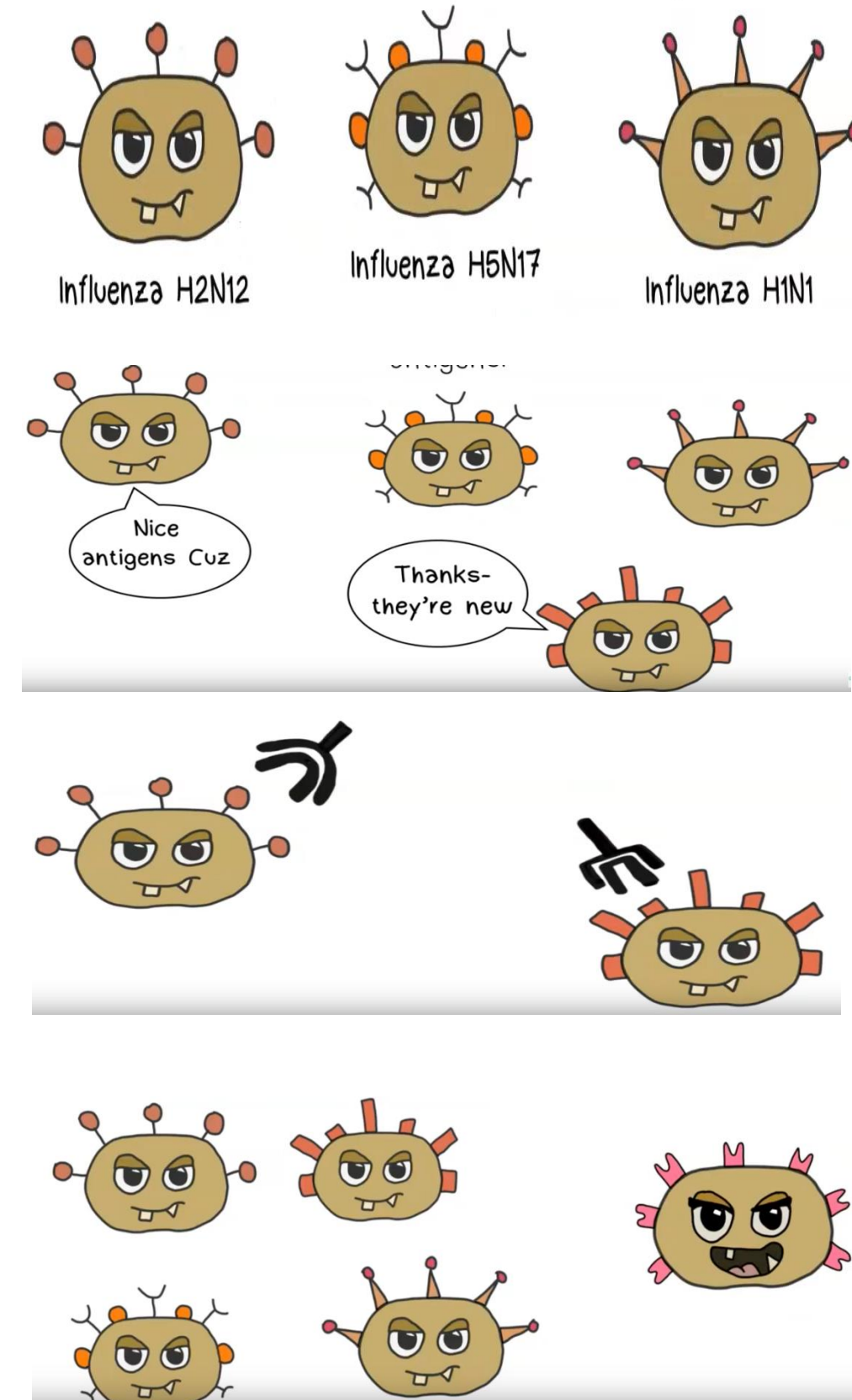
- Directly act on/influence **cells** (cell-mediated immunity): help, kill, regulate.
- T-cells help coordinate the immune response and directly kill infected cells.
- There are several types of T-cells:
 - Helper-T cells: permission giver.
 - Killer T-cells/Cytotoxic T-cells: causing their destruction.
 - Suppressor T-cells/Regulatory T-cells: preventing excessive immune responses and autoimmunity.

How Does IS Work?



How Does IS Work?

- Antigen → activates B cells → produces antibodies.
- Antigen + existing antibody → triggers effector actions.
- **Why do I get the flu when I have already gotten the flu shot?**
- Some viruses that cause the common flu have many different strains.
- Some viruses mutate frequently and develop different antigens.
- Changed antigens require different antibodies to fight against it.
- The flu shot will help you produce antibodies against a few different strains of the flu, but there is always a chance that a different strain could infect you.

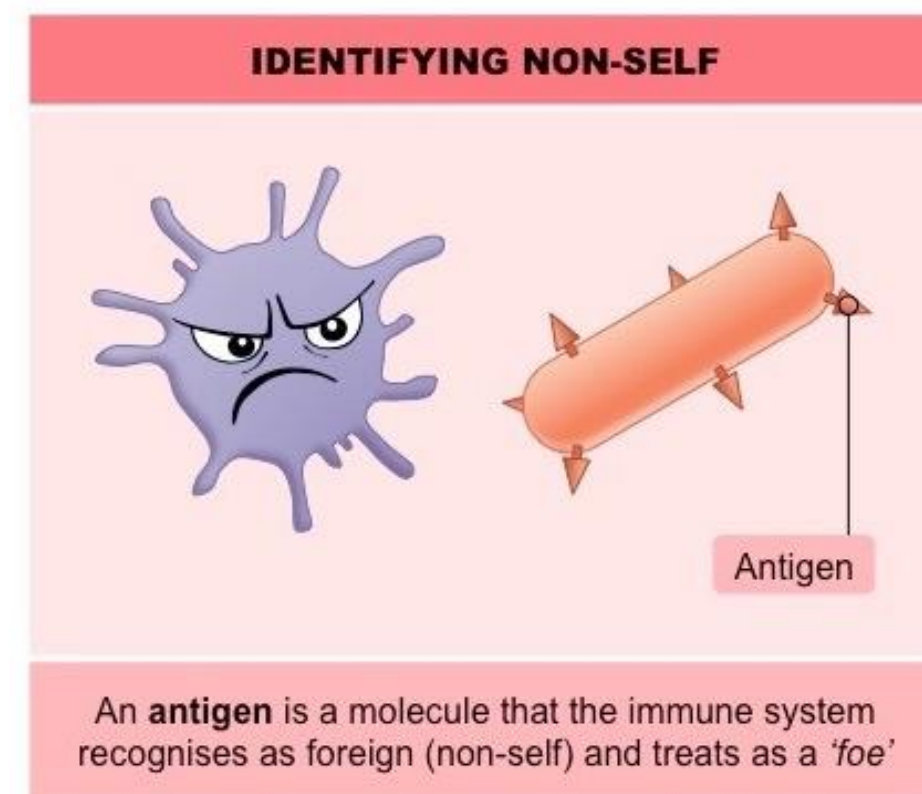
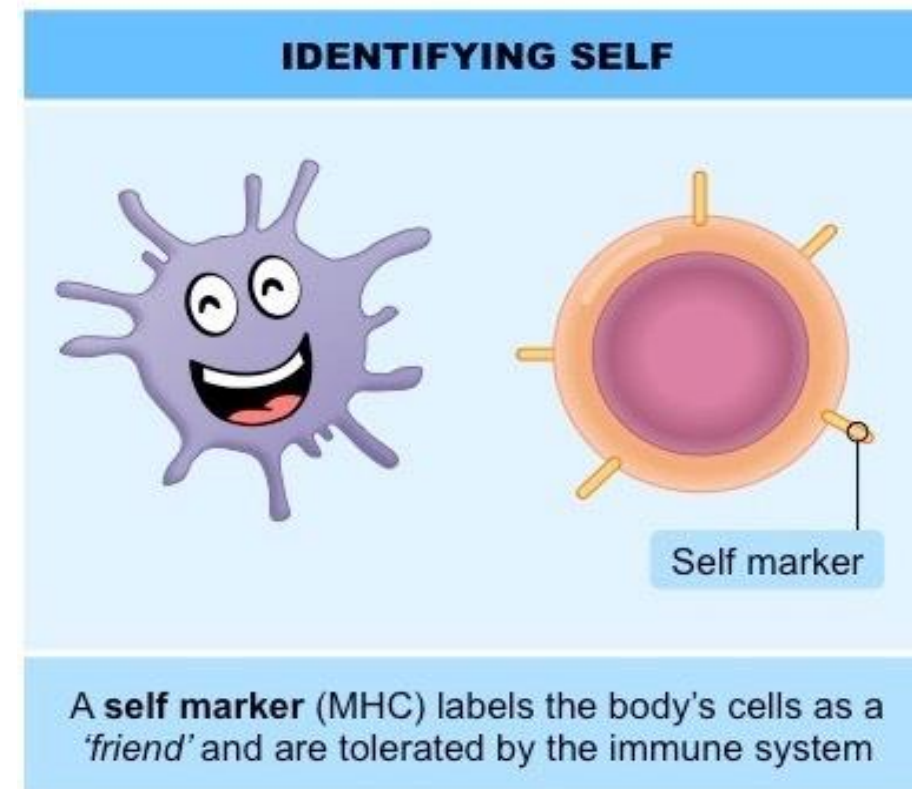


Appealing Features of BIS

- Recognition
 - Anomaly detection
 - Noise tolerance
- Robustness
- Uniqueness
- Autonomy
- Distributed
- Flexible
- Reinforcement learning
- Memory.

Self / Non-Self Recognition

- Immune system needs to be able to differentiate between self and non-self cells.
 - Cells belonging to the body itself are called the “self” cells.
 - External cells are called the “non-self” cells.
- Antigenic encounters may result in cell death, therefore
 - Some kind of **positive selection**.
 - Some element of **negative selection**.
- It is important to keep the “self” cells unharmed in the course of exterminating the “non-self” cells.

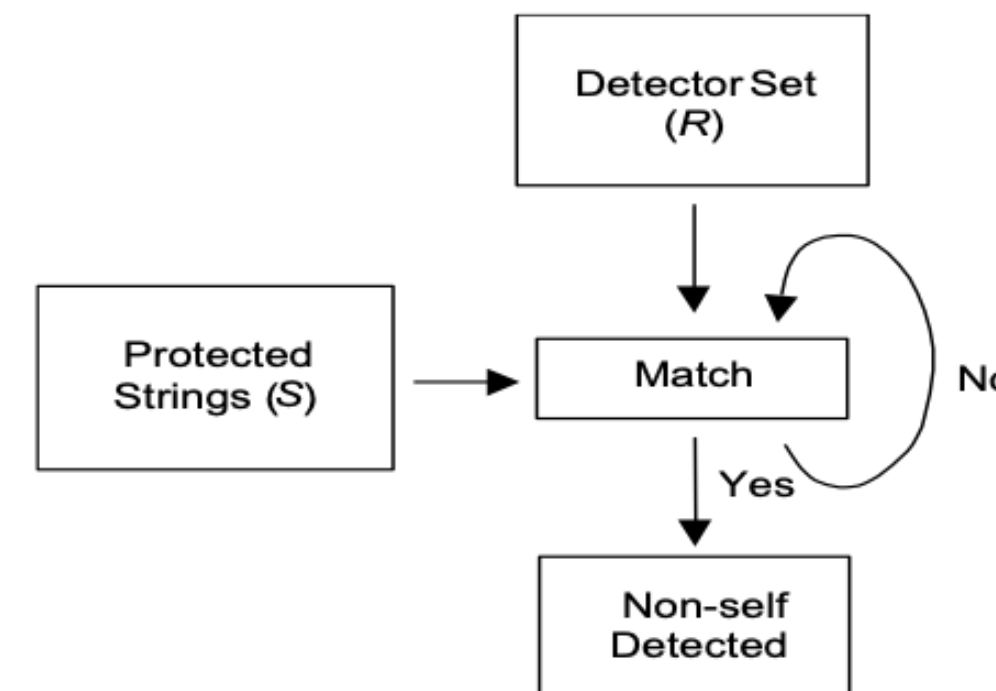
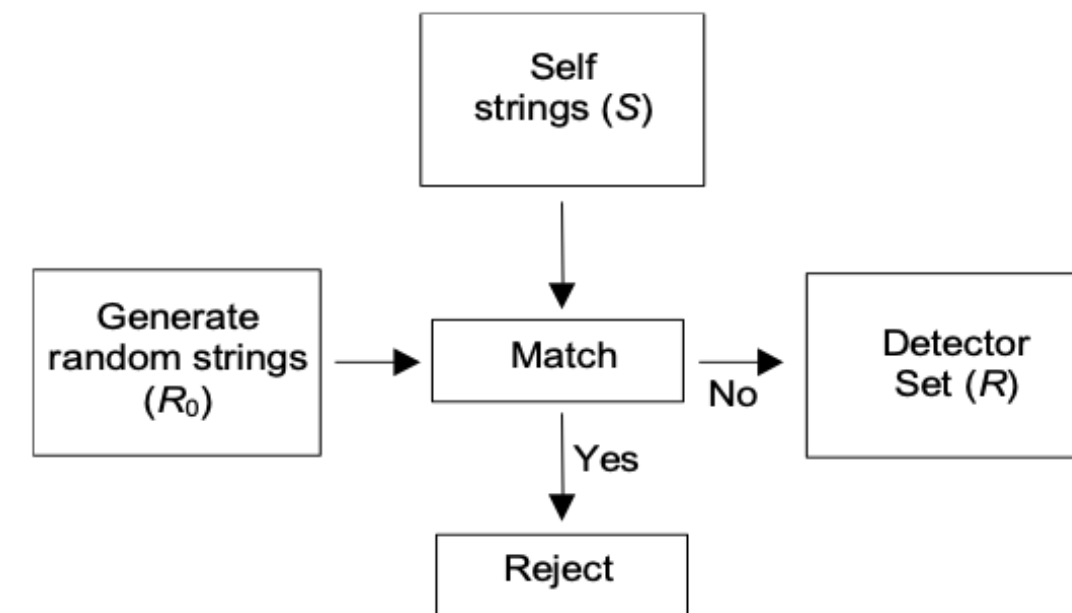


Basic Immune Models and Algorithms

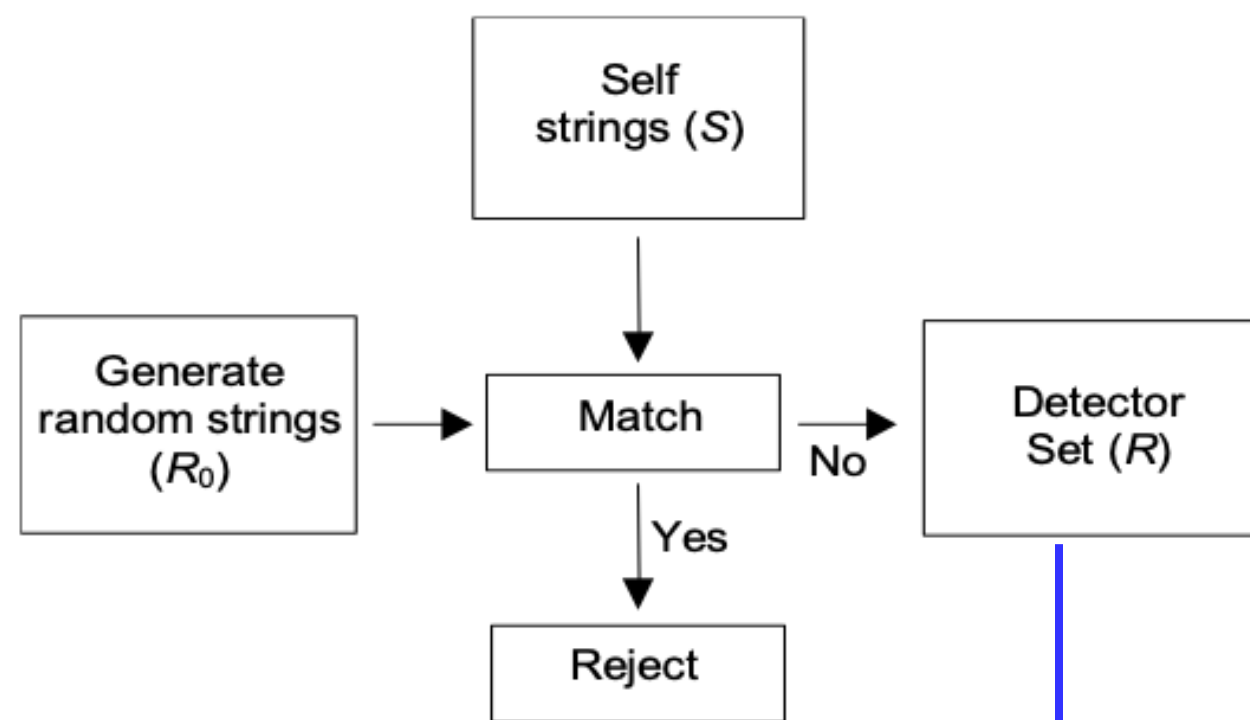
- Negative Selection Algorithms.
- Clonal Selection Algorithm.
- Bone Marrow Models.
- Somatic Hypermutation.
- Immune Network Models.

Negative Selection Algorithm

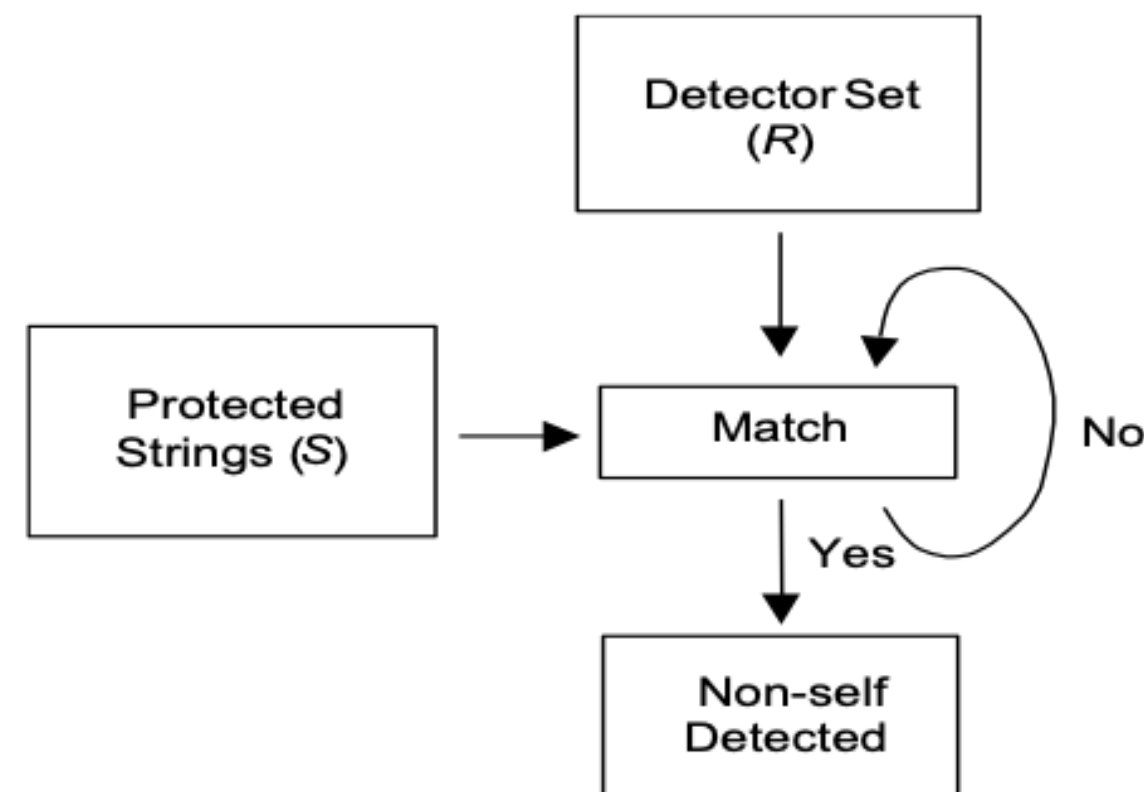
- Concept of **Self** and **Non-Self**.
- Provide tolerance for self cells.
- It deals with the immune system's ability to detect unknown antigens.
 - while not reacting to the self cells.
- Antibodies should not react with body cells.
 - Analogously **detector set** should not detect self cells.



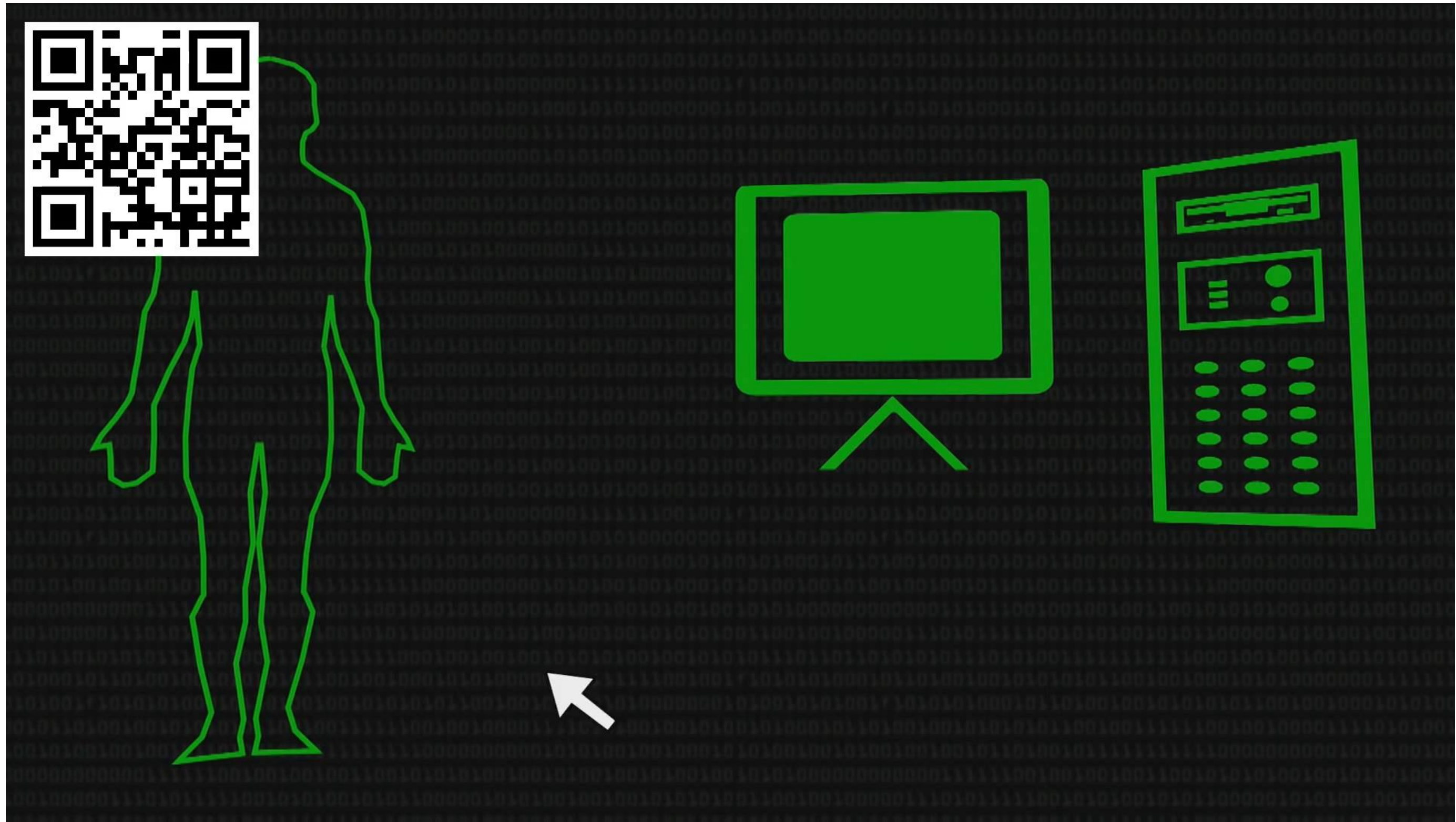
Negative Selection Algorithm



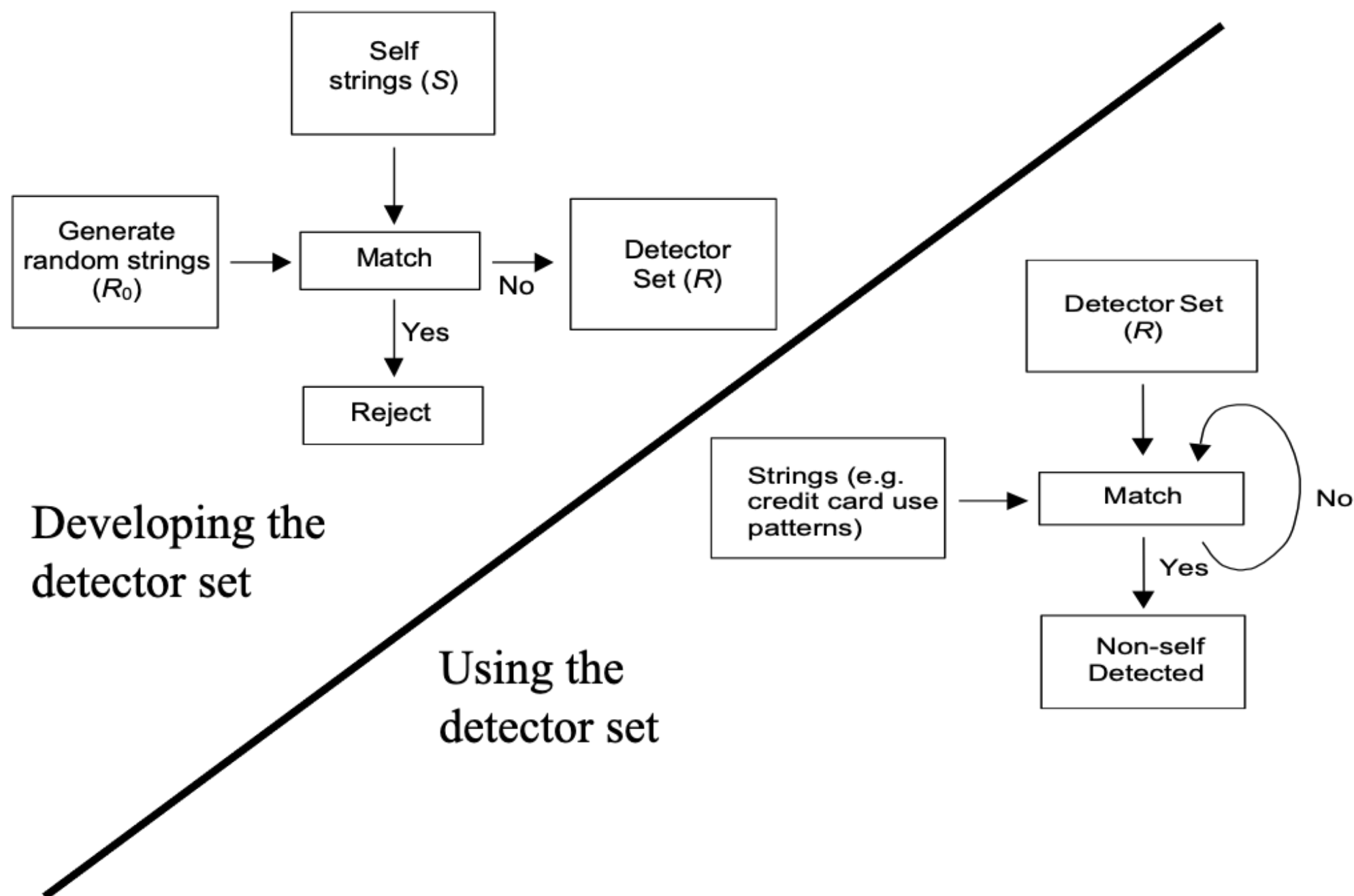
A **self-tolerant** sentinel pattern
used to catch things that look “not
like self.”
(**NOT** “the non-self set”)



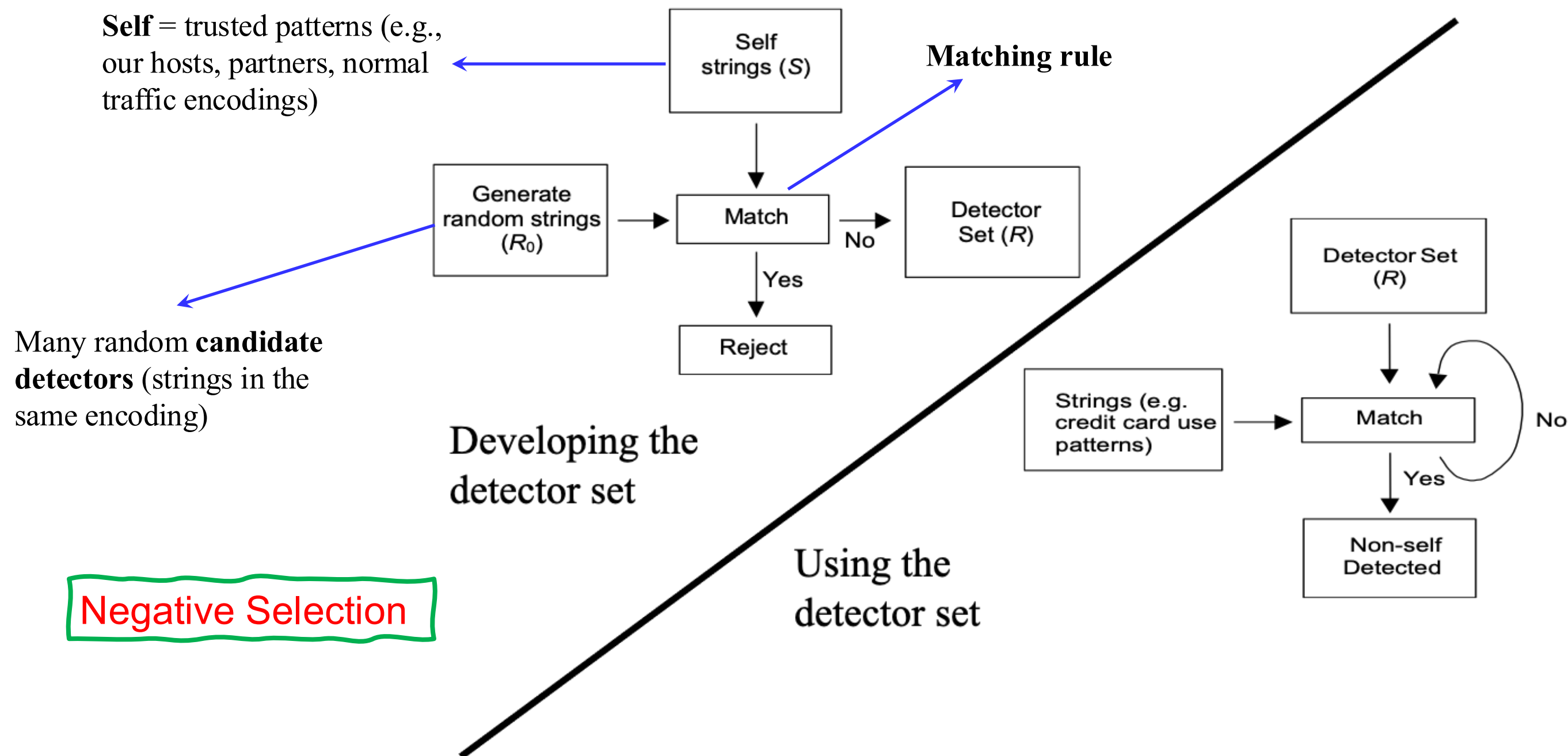
Negative Selection Algorithm



IDS using Negative Selection Algorithms



IDS using Negative Selection Algorithms



Negative Selection (AIS) for Anomaly Detection

- **Goal:** Build detectors that *don't* match **Self** (normal) so they can flag **Non-self** (anomalies).
- **Encoding:** 8-bit strings
- **Match rule:** *r-contiguous bits*, with $r = 3$ (a match if any 3 identical bits align contiguously)
- **Self set S :**

00000000, 00001111, 11110000, 10101010

OSLOMET Negative Selection (AIS) for Anomaly Detection

Training (Negative Selection)

- **Step 1:** Generate candidates (random 8-bit strings)
Example candidates: 00101100, 11101111, 10111010, 01010101, 11000011
- **Step 2:** Filter against Self
For each candidate c , compare to every $s \in S$.
If c **matches** any s by r -contiguous = 3 \rightarrow **discard**.
If c does **not** match any $s \rightarrow$ **keep** as detector.

Quick checks (illustrative):

- 01010101 vs 10101010 \rightarrow many alternating bits but **no 3-bit contiguous match** \rightarrow **KEEP**
- 11101111 vs 11110000 \rightarrow shares 1111 in positions 1–4?
 - (check alignment) if ≥ 3 **contiguous** \rightarrow **DISCARD**
- Suppose survivors (detectors R) after filtering:
$$R = \{01010101, 00101100, 11000011\}$$

OSLOMET Negative Selection (AIS) for Anomaly Detection

Deployment (Monitoring)

- **Step 3:** Monitor new traffic

New items arrive: 10100111, 00000111, 11111111, 00101101

- **Step 4:** Alert if any detector matches

For each new item x , if $\exists r \in R$ s.t. r –contiguous = 3 **matches**, flag as **Non-self**.

- **Example outcomes (illustrative):**

- 10100111 vs 01010101 → no 3-bit contiguous match → **No alert**
- 00000111 vs any $r \in R$ → likely **No alert** (close to self 00001111, **but detectors avoid self**)
- 11111111 vs 11000011 → shares 111 at start? if ≥ 3 **contiguous** → **ALERT**
- 00101101 vs 00101100 → shares 001011 (≥ 3 contiguous) → **ALERT**

OSLOMET Negative Selection (AIS) for Anomaly Detection

Issues to Remember

- Detectors R are self-tolerant sentinels (not “the non-self set”).
- More detectors \rightarrow better coverage, higher runtime.
- Maintenance: Update Self (concept drift) and retrain detectors periodically.

Negative Selection: Example

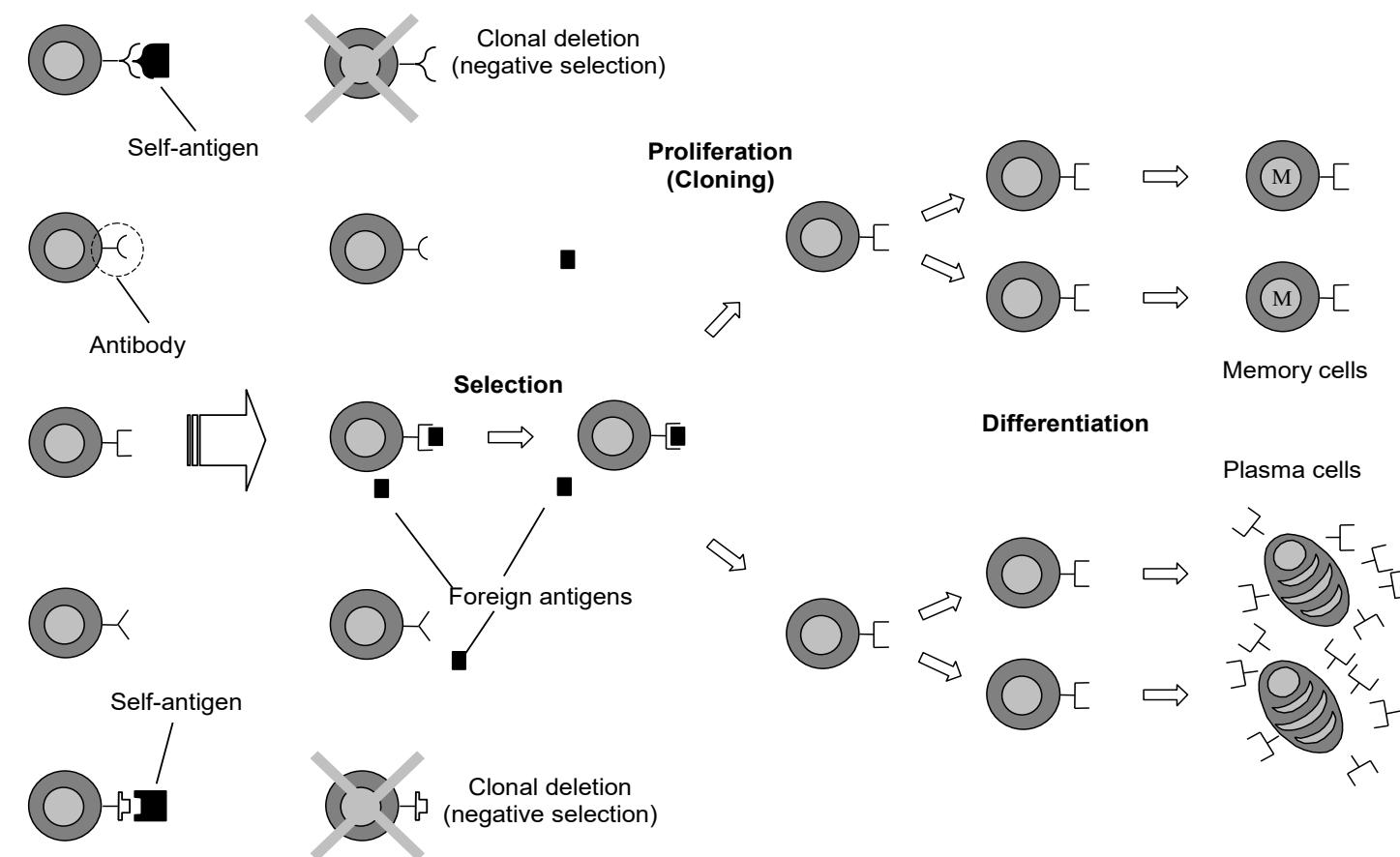
- Under what circumstance would a negative selection algorithm be suitable for an artificial immune system implementation?

Anomaly Detection / Intrusion Detection System

- The normal behavior of a system is often characterized by a series of observations over time.
- The problem of detecting novelties, or anomalies, can be viewed as finding deviations of a characteristic property in the system. (i.e. non-self).

Clonal Selection Algorithm (CSA)

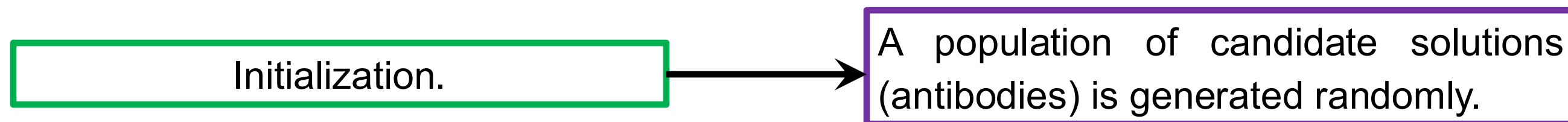
- CSAs are a class of algorithms inspired by the clonal selection theory of acquired immunity that explains:
 - How B and T cells improve their response to antigens over time (affinity maturation).
- Focus on the Darwinian attributes of the theory, where
 - Selection** is inspired by the affinity of antigen-antibody interactions,
 - Reproduction** is inspired by cell division, and
 - Variation** is inspired by **somatic hypermutation**.
- Most commonly applied to optimization and pattern recognition,
 - mimicking how the immune system responds to antigens by **selecting and cloning immune cells** that are best suited for recognizing specific threats.



CSA: Biological Inspiration

- B-cells play a key role in recognizing antigens, like viruses or bacteria, in the human immune system.
- When a B-cell recognizes an antigen, it undergoes a process called **clonal selection**:
 - **Recognition**: The B-cell that has the best match (affinity) to the antigen is selected.
 - **Cloning**: This best B-cell is cloned multiple times to produce more cells that recognize the antigen.
 - **Mutation (Hypermutation)**: Some cloned B-cells undergo mutations to improve their antigen recognition ability.
 - **Memory Cells**: A portion of the best-performing B-cells is stored as memory cells, which allows the immune system to respond more quickly if the same antigen is encountered in the future.

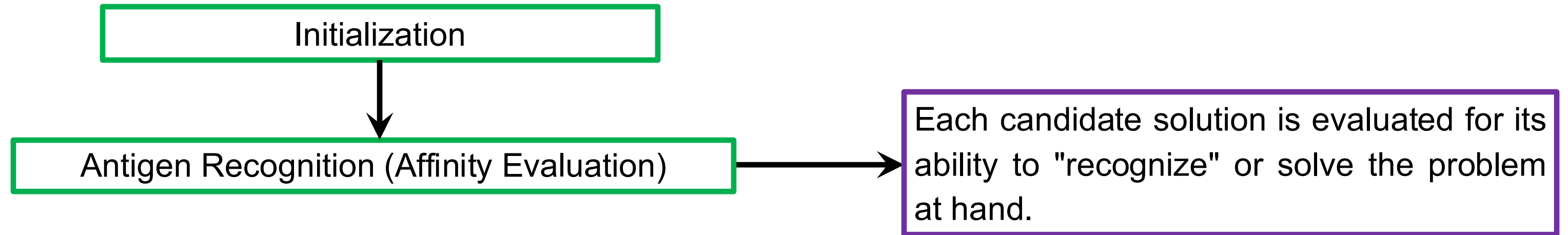
CSA in AIS



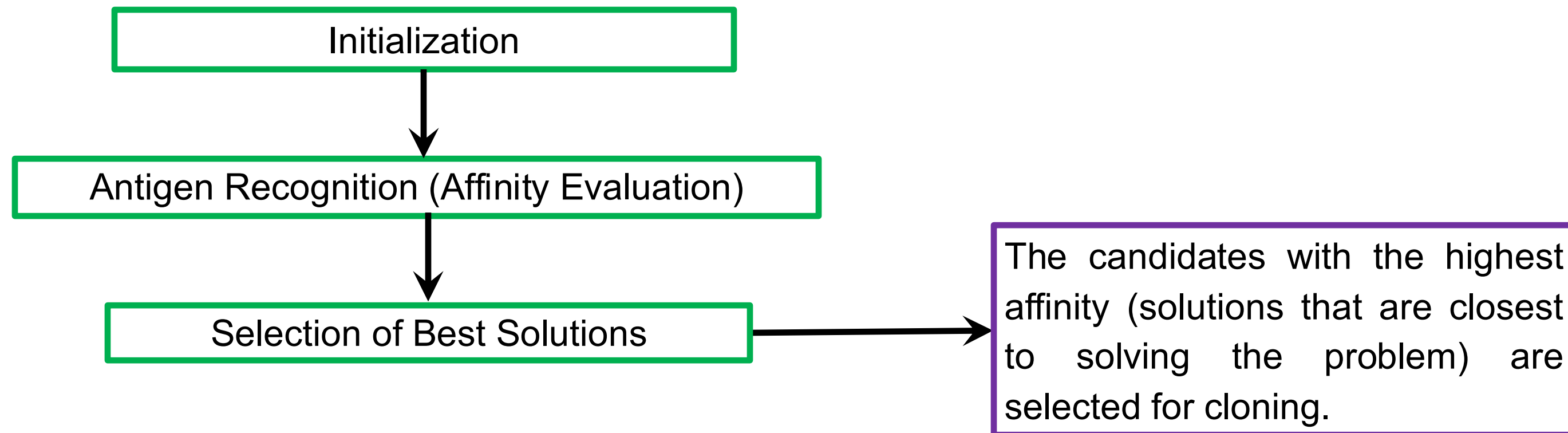
* **Antibody** = a candidate solution (an individual) in the population (initially randomly generated).

** It's the thing you evaluate with a **fitness/affinity** function against the **antigen** (the problem or data pattern).

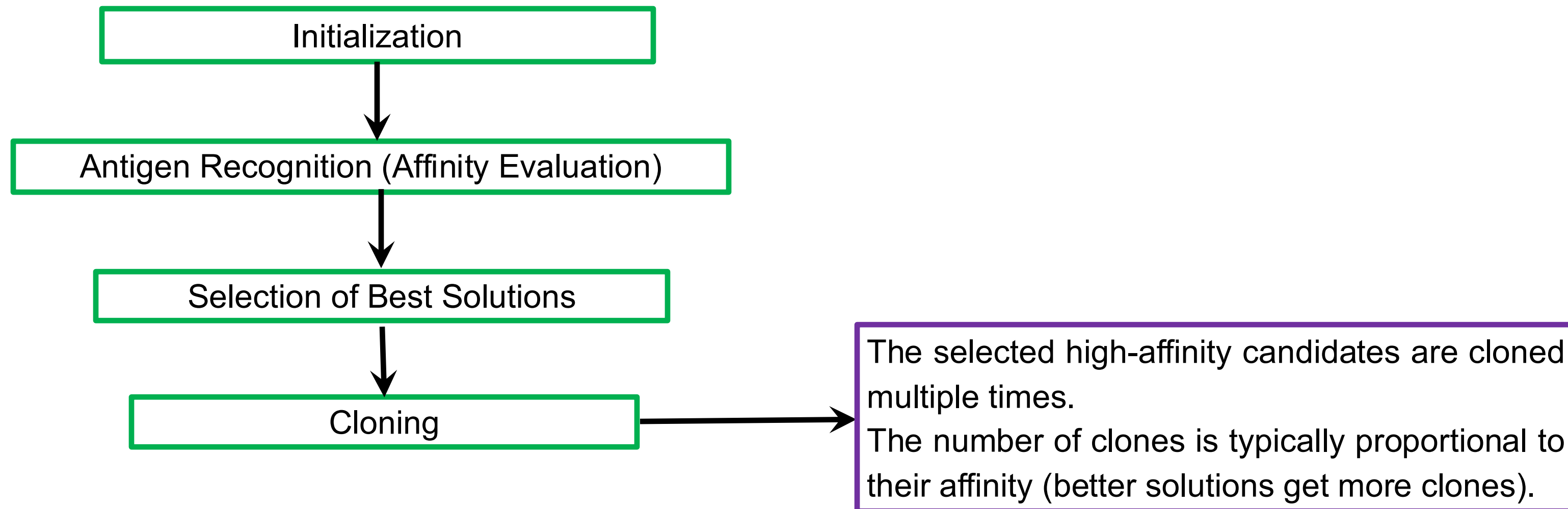
CSA in AIS

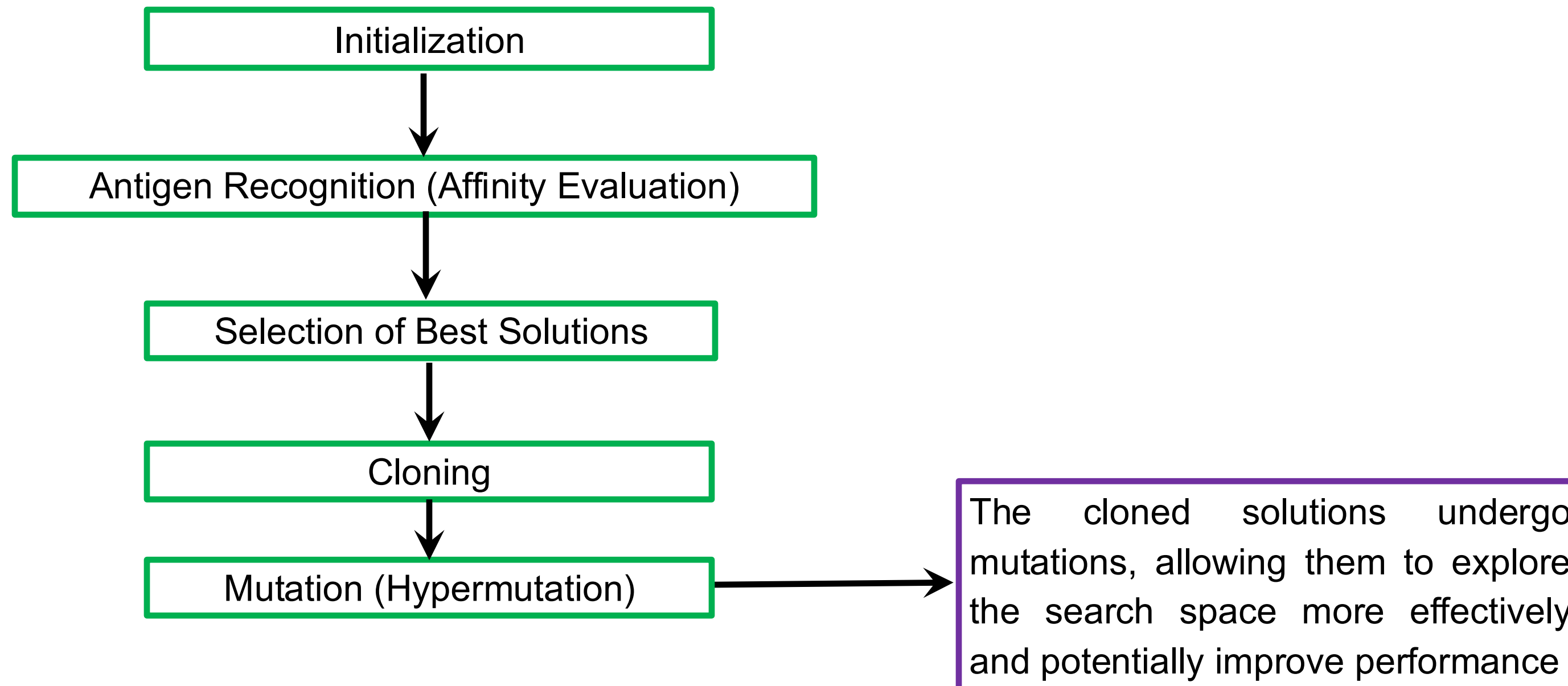


CSA in AIS

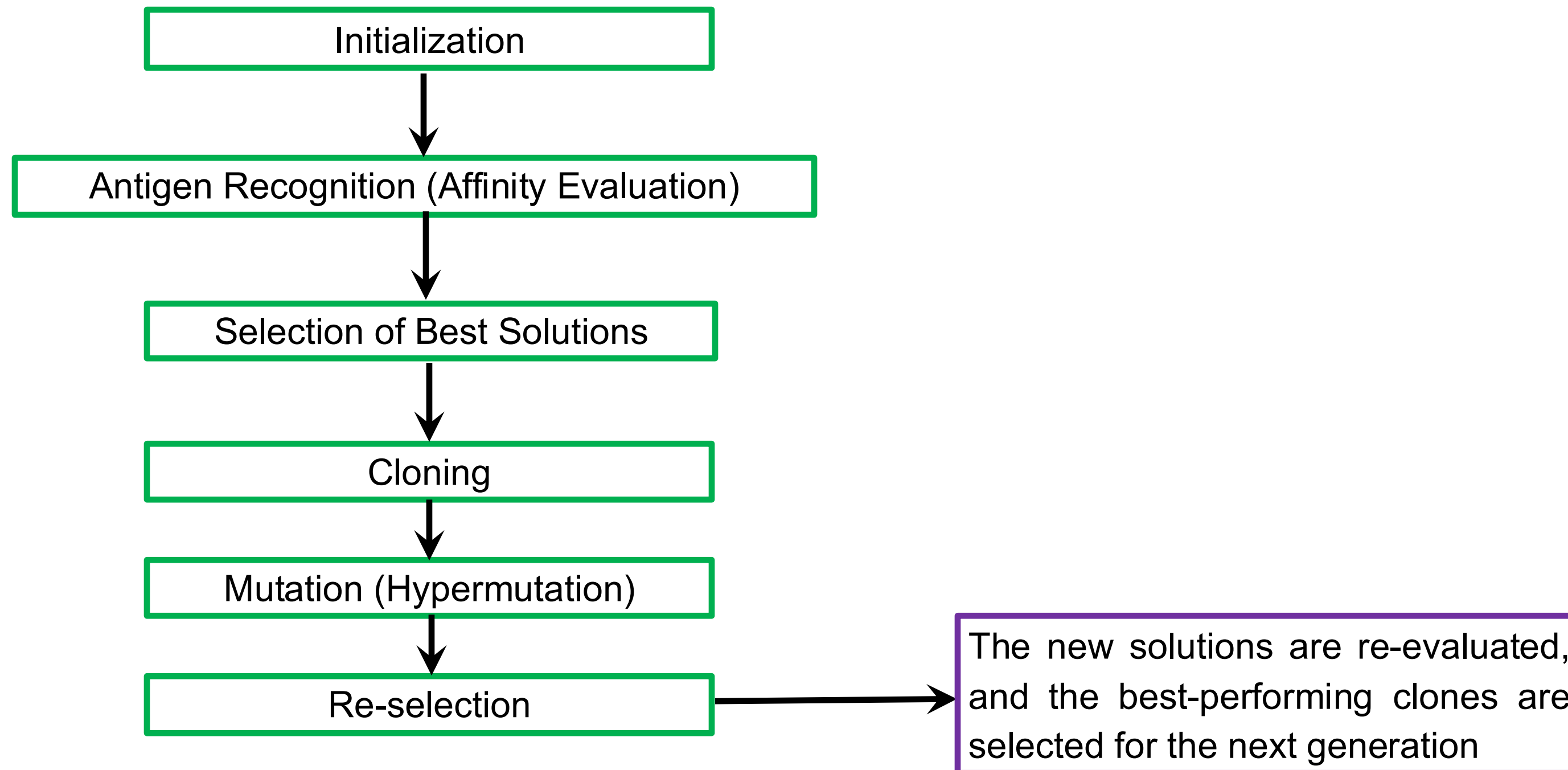


CSA in AIS

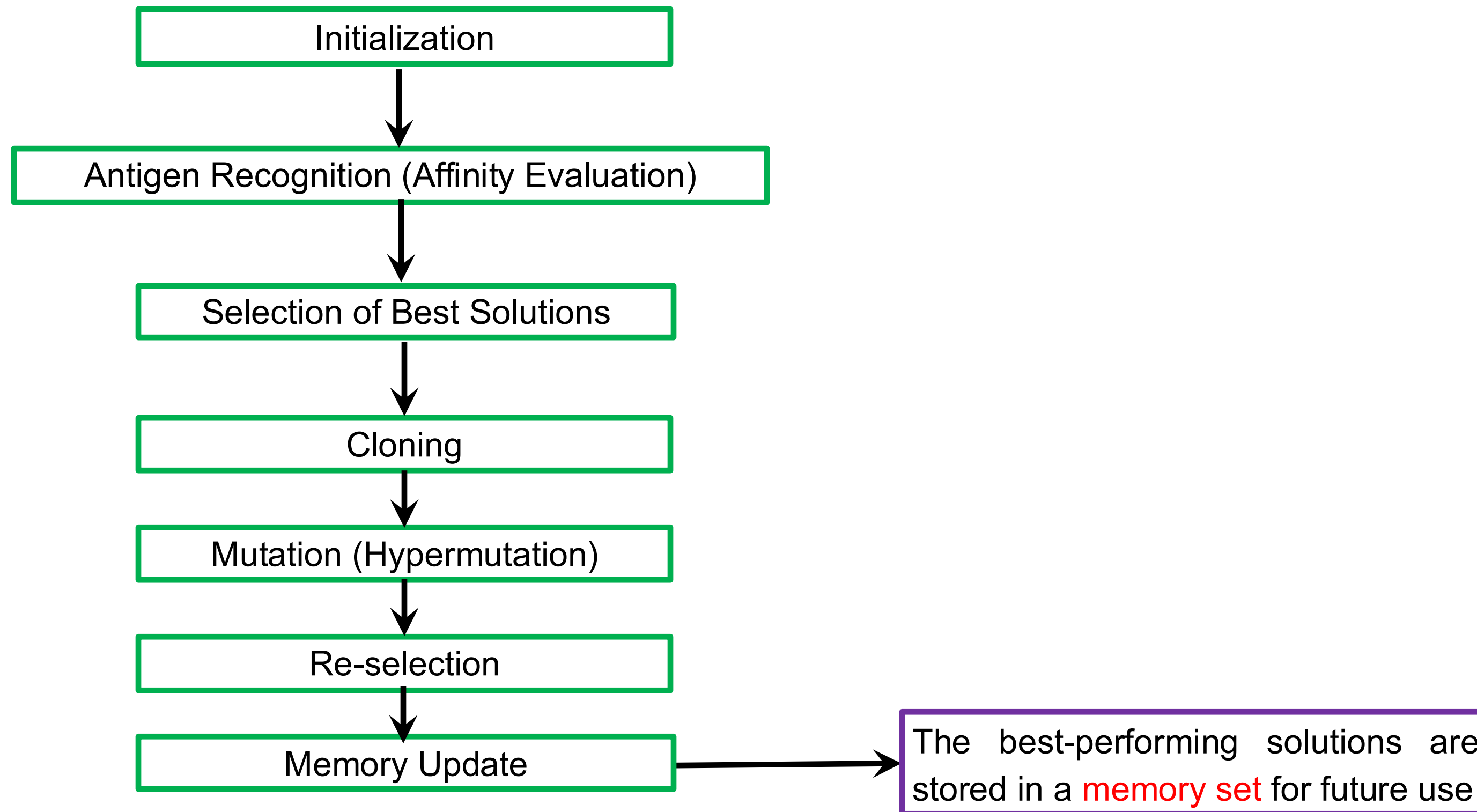




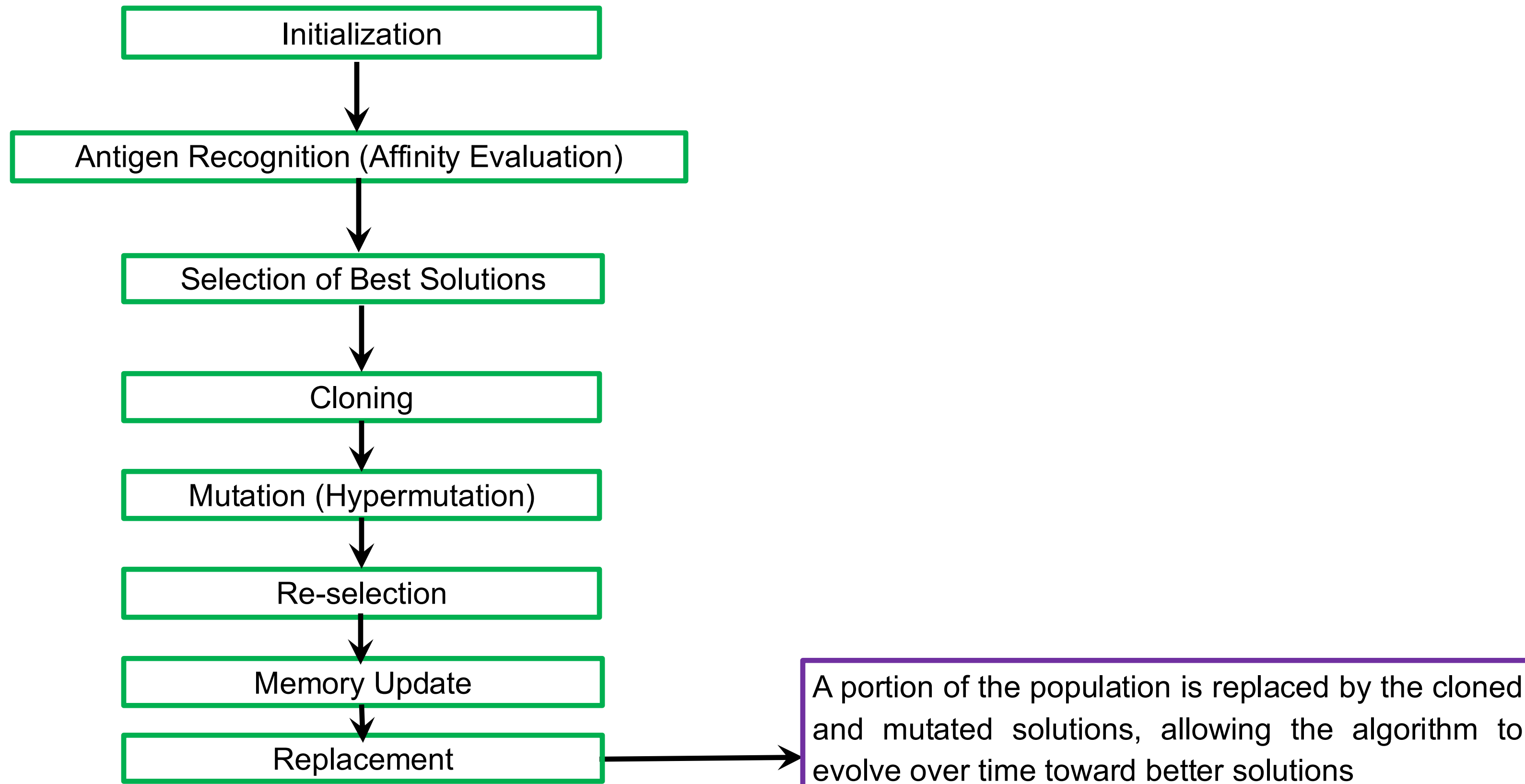
CSA in AIS

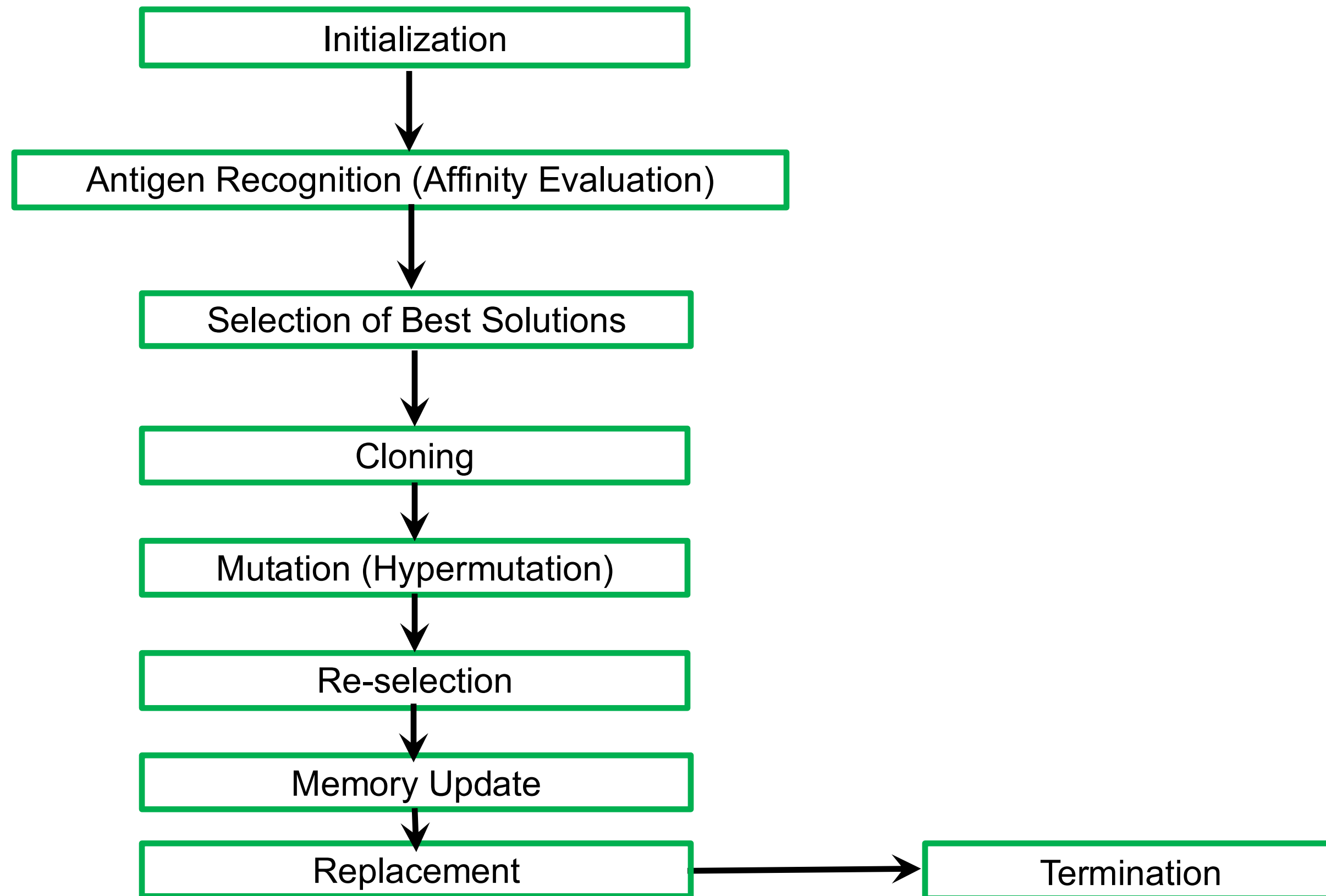


CSA in AIS



CSA in AIS





Hypermutation

- In AIS, hypermutation refers to a process where candidate solutions (B-cells in the BIS) undergo **intensive mutation** at a higher rate compared to standard mutation techniques.
- Hypermutation aims to explore the solution space more thoroughly, allowing for quicker and more diverse adjustments to optimize solutions.
- Although the AIS is inspired by the BIS, the concepts of hypermutation and mutation rates differ between them.

Hypermutation in BIS: Somatic Hypermutation (SHM)

- In BIS, somatic hypermutation occurs in B-cells during the immune response.
 - The purpose is to generate antibodies with diverse affinities for antigens.
- **Mutation rates are adaptive:** Initially, B-cells undergo frequent mutations to explore a wide range of antibody configurations.
 - However, as the immune system identifies B-cells with **higher affinity** antibodies, the **mutation rate decreases**.
- This leads to **affinity maturation**, where antibodies that bind well to antigens are fine-tuned over time, reducing the chance of "disrupting" a good solution.
- **Key point:** In biology, once an effective (high-affinity) antibody is found, further mutations are limited to avoid losing the advantage.
- Classic trade-off between exploration and exploitation.

CAS: Quick mapping (AIS \rightleftharpoons Optimization Terms)

- **Antigen:** the task or data instance you want to solve/classify (e.g., a training pattern).
- **Antibody:** An **encoding of solution** (bitstring, real vector, permutation, NN weights, rule set, etc.).
- **Affinity:** how well the solution solves the task (**fitness score**).
- **Cloning:** copy the **best** antibodies (more copies for higher affinity).
- **Hypermutation:** mutate clones; higher mutation for lower affinity (initial stages), lower mutation for higher affinity (later stages).
- **Memory set:** an elite archive of top solutions.

CAS: Tiny example (Optimization)

- **Encoding (antibody):** 10-dim real vector \mathbf{x} .
- **Antigen:** objective $f(\mathbf{x})$ to minimize.
- **Affinity:** $-f(\mathbf{x})$ (higher is better).

CAS: Example

- Suppose a delivery company needs to visit 6 cities.
- The goal is to find the shortest possible route that visits each city exactly once and returns to the starting point (**TSP**).

Cities and Distances:

City 1	City 2	Distance (km)
New York	Los Angeles	4,500
New York	Chicago	1,300
New York	Houston	2,600
New York	Miami	2,000
New York	Denver	3,200
Los Angeles	Chicago	3,200
Los Angeles	Houston	2,400
Los Angeles	Miami	4,500
Los Angeles	Denver	1,600
Chicago	Houston	1,600
Chicago	Miami	2,300
Chicago	Denver	1,500
Houston	Miami	1,900
Houston	Denver	1,600
Miami	Denver	3,400



CAS: Example: Initial Population

- Generate an initial population of random routes (candidate solutions) → Antibody.
- Each route represents a possible sequence of visiting all cities and returning to the starting city.
- Example of 3 random routes:
 - **Route 1:** New York → Miami → Houston → Denver → Chicago → Los Angeles → New York
 - **Route 2:** New York → Los Angeles → Denver → Chicago → Houston → Miami → New York
 - **Route 3:** New York → Houston → Chicago → Denver → Los Angeles → Miami → New York


CAS: Example: Evaluation (Affinity)

- Calculate the total distance for each route, which corresponds to the affinity or fitness in the CSA.
- The goal is to minimize this distance (maximize affinity)..
- Example of 3 random routes:
 - **Route 1 distance:** $2,000 + 1,900 + 1,600 + 1,500 + 3,200 + 4,500 = 14,700$ km
 - **Route 2 distance:** $4,500 + 1,600 + 1,500 + 1,300 + 2,600 + 2,000 = 13,500$ km
 - **Route 3 distance:** $2,600 + 1,600 + 1,500 + 1,600 + 1,600 + 4,500 = 13,400$ km


CAS: Example: Selection

- Select the best routes based on affinity (the shortest distances).
- The best-performing routes are cloned for the next iteration.
- In this example, let's assume **Route 2** and **Route 3** have better fitness, so they will be cloned.
 - **Route 1 distance:** $2,000 + 1,900 + 1,600 + 1,500 + 3,200 + 4,500 = 14,700$ km
 - **Route 2 distance:** $4,500 + 1,600 + 1,500 + 1,300 + 2,600 + 2,000 = 13,500$ km 
 - **Route 3 distance:** $2,600 + 1,600 + 1,500 + 1,600 + 1,600 + 4,500 = 13,400$ km 

CAS: Example: Cloning

- Create multiple copies (clones) of the selected routes.
 - The number of clones is typically proportional to their affinity.
 - Better routes produce more clones, increasing the chances of improving them further.
-
- **Route 2:** New York → Los Angeles → Denver → Chicago → Houston → Miami → New York
 - **Route 3:** New York → Houston → Chicago → Denver → Los Angeles → Miami → New York
-
- 
-
- **Route 2 clones:** New York → Los Angeles → Denver → Chicago → Houston → Miami → New York
 - **Route 3 clones:** New York → Houston → Chicago → Denver → Los Angeles → Miami → New York

CAS: Example: Hypermutation

- Apply hypermutation to the clones.
 - This involves making small random changes to the order of cities in each clone.
 - The mutation rate is high, and mutations are focused on clones with higher affinity to improve them further.
-
- **Route 2 clones:** New York → Los Angeles → Denver → Chicago → Houston → Miami → New York
 - **Route 3 clones:** New York → Houston → Chicago → Denver → Los Angeles → Miami → New York
-
- 
-
- **Mutated Route 2 clone 1:** New York → Chicago → Denver → Los Angeles → Houston → Miami → New York
 - **Mutated Route 3 clone 1:** New York → Miami → Los Angeles → Denver → Chicago → Houston → New York

CAS: Example: Reevaluation

- Reevaluate the new mutated routes by calculating their distances (affinity).
- **Mutated Route 2 clone 1** has the shortest distance (10,700 km) compared to the original routes.
 - **Mutated Route 2 clone 1 distance:** $1,300 + 1,500 + 1,600 + 2,400 + 1,900 + 2,000 = 10,700$ km
 - **Mutated Route 3 clone 1 distance:** $2,000 + 4,500 + 1,600 + 1,500 + 1,600 + 1,900 = 13,100$ km

CAS: Example: Selection of Best Solutions

- Select the best-performing (shortest) routes from the newly mutated population.
- In this case, **Mutated Route 2 clone 1** is the best.
 - **Mutated Route 2 clone 1 distance:** $1,300 + 1,500 + 1,600 + 2,400 + 1,900 + 2,000 = 10,700$ km
 - **Mutated Route 3 clone 1 distance:** $2,000 + 4,500 + 1,600 + 1,500 + 1,600 + 1,900 = 13,100$ km

CAS: Example: Repeat

- Repeat the **cloning**, **hypermutation**, and **selection** process iteratively until the algorithm converges on an optimal or near-optimal solution (route).
- Over several iterations, the algorithm refines the routes to minimize the travel distance.
- After several iterations, the algorithm might converge on an optimal route such as:

Implementation of A Basic AIS

Fixed the encoding



Choose a suitable similarity measure

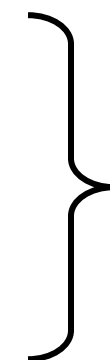


Selection



Cloning

Mutation



Based on the
similarity measure



Until stopping criteria are met.

- Four decisions have to be made:
 - Encoding
 - Similarity Measure
 - Selection
 - Mutation.

Choice of Representation

- Antigens and antibodies are represented or encoded in the same way.

- Assume the general case:

$$\mathbf{Ab} = \langle Ab_1, Ab_2, \dots, Ab_L \rangle$$

$$\mathbf{Ag} = \langle Ag_1, Ag_2, \dots, Ag_L \rangle$$

- Binary representation
 - Matching by bits
- Continuous (numeric)
 - Real or Integer, typically Euclidian
- Categorical (nominal)
 - E.g female or male of the attribute Gender. Typically no notion of order

Representation (Example)

- For the movie recommendation, a possible encoding is:

$$User = \{\{id_1, score_1\}, \{id_2, score_2\}, \dots, \{id_n, score_n\}\}$$

- For intrusion detection, the encoding may be to encapsulate the essence of each data packet transferred,

$$[<protocol> <source\ ip> <source\ port> <destination\ ip> <destination\ port>]$$

$$[<tcp> <113.112.255.254> <108.200.111.12> <25>]$$

Choice of Affinity Functions

- Affinity: Closeness between Antibody and Antigen.
- Closely coupled to the encoding scheme.

Affinity: Binary Coding

- Hamming distance

0	0	1	1	0	0	1	1
1	1	1	0	1	1	0	1

XOR : 1 1 0 1 1 1 1 0
Affinity: 6

- r -contiguous bits rule

0	0	1	1	0	0	1	1
1	1	1	0	1	1	0	1

XOR : 1 1 0 1 1 1 1 0
Affinity: 4

Affinity: Not Binary

- If the encoding is non-binary, e.g. real variables, there are even more possibilities to compute the “distance” between the two strings:

- Euclidean $D = \sqrt{\sum_{i=1}^L (Ab_i - Ag_i)^2}$

- Manhattan $D = \sum_{i=1}^L |Ab_i - Ag_i|$

- Hamming $D = \sum_{i=1}^L \delta$, where $\delta = \begin{cases} 1 & \text{if } Ab_i \neq Ag_i \\ 0 & \text{otherwise} \end{cases}$

- For some problems (e.g. data mining), similarity often means “**correlation**”.

$$r = \frac{\sum_{i=1}^n (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_{i=1}^n (u_i - \bar{u})^2 \sum_{i=1}^n (v_i - \bar{v})^2}}$$

Pearson correlation coefficient

Mutation

- Very similar to that found in Genetic Algorithms.
 - for binary strings bits are flipped.
 - for real value strings one value is changed at random.
 - for others the order of elements is swapped.

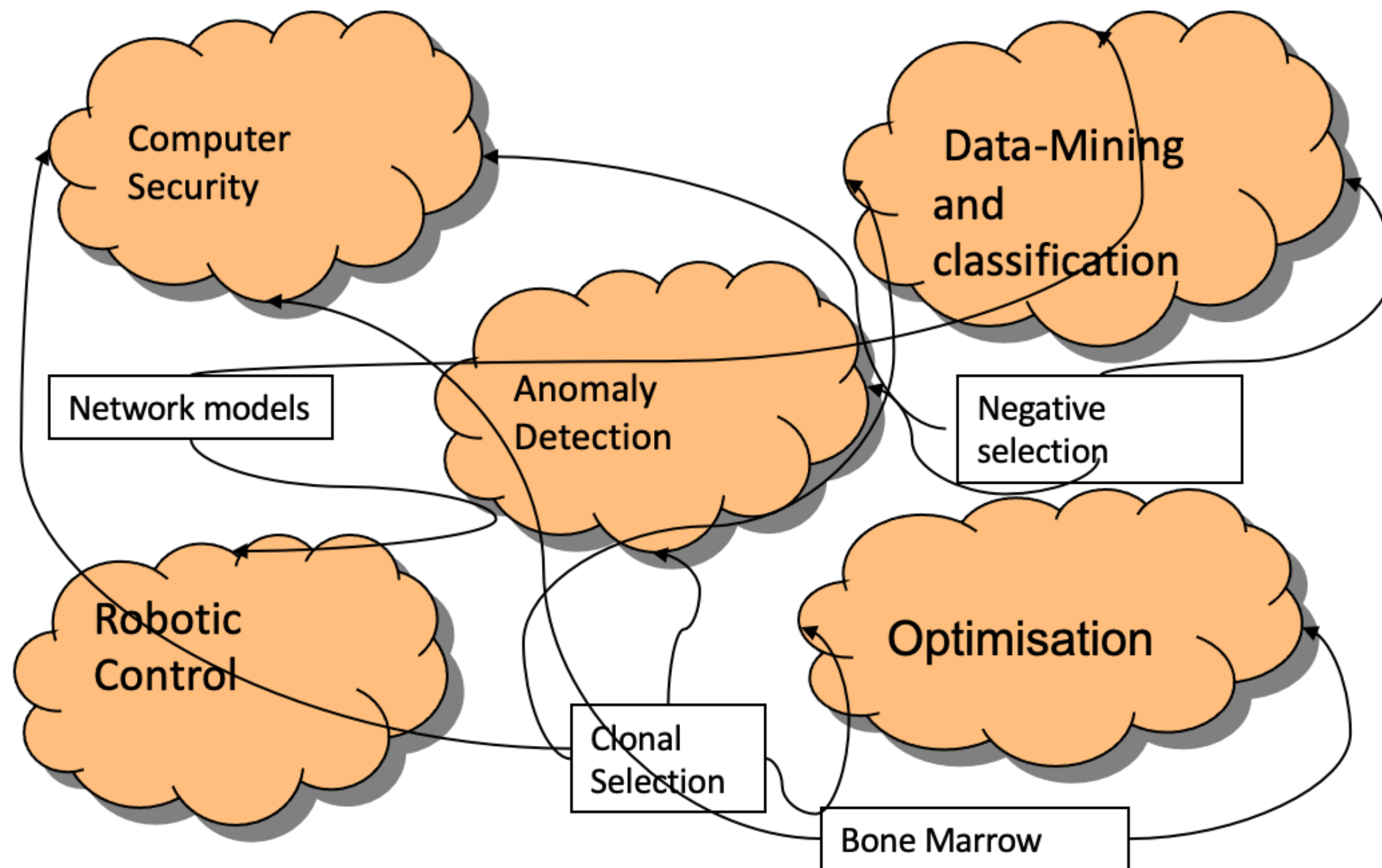
Differences Between GA and CSA

- **Selection:**
 - GA: Pick parents by fitness (roulette/tournament).
 - CSA: Pick **top-affinity antibodies** to **clone** (more clones if better).
- **Variation:**
 - GA: **Crossover** (core) + small, usually fixed **mutation**.
 - CSA: **No crossover** (classic). **Hypermutation**.
- **Memory/Elitism:**
 - GA: Optional **elitism** (carry a few best).
 - CSA: Explicit **memory set** (long-term archive) that seeds/guards top solutions.
- **Exploitation vs Exploration:**
 - **GA:** Exploit via crossover; explore via mutation.
 - CSA: Exploit via **clone neighborhoods** around elites (low hypermutation at later stage); explore via high **hypermutation during initial stages + random inserts**.

Differences Between GA and CSA

- **Big Idea:**
 - GA: Evolve by recombining solutions (crossover) + light mutation.
 - **CSA:** Evolve by **cloning the best** + **hypermutation** (mutation rate tied to affinity) + **memory**.
- **One-line mental model:**
 - GA: “breed good parents to mix their genes.”
 - **CSA:** “copy the best, mutate their clones aggressively (bad → more), keep a memory of champions.”

Example Application Areas



Suggested Reading

- Yuce, B., Packianather, M. S., Mastrocinque, E., Pham, D. T., & Lambiase, A. (2013). Honey bees inspired optimization method: the bees algorithm. *Insects*, 4(4), 646-662.
- Aickelin, U., Dasgupta, D., & Gu, F. (2014). Artificial immune systems. In *Search methodologies* (pp. 187-211). Springer, Boston, MA.

Questions?

