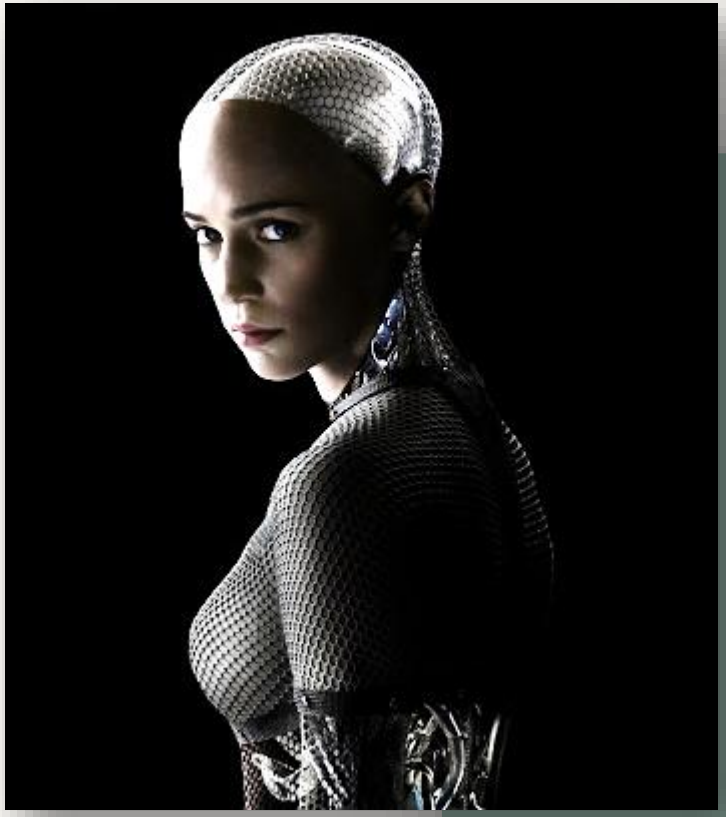
AKADEMIA
GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

AGH

INŻYNIERIA
MECHATRONICZNA

Advanced AI models in engineering

Is this the best you've got?



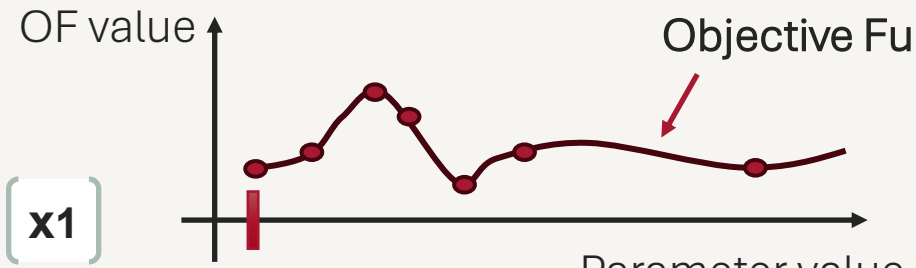
dr hab. inż. Ziemowit Dworakowski

KATEDRA ROBOTYKI I MECHATRONIKI
WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

21/10/26

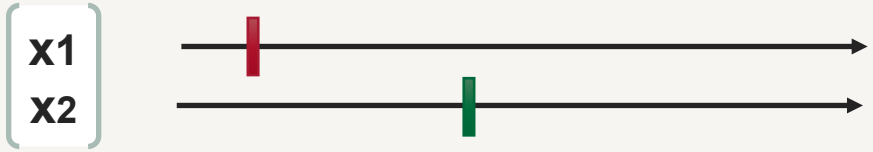
How to find the best solution?

Lets simplify to just one parameter:

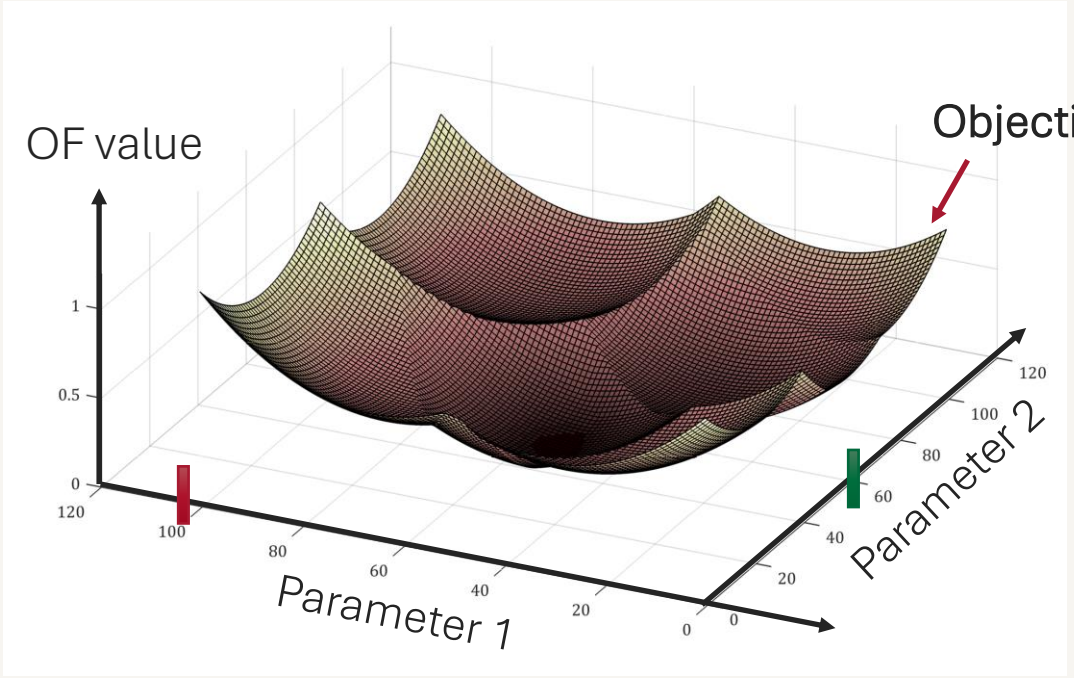


It is like we'd move a slider trying to find the best setup

For two parameters, it works the same:

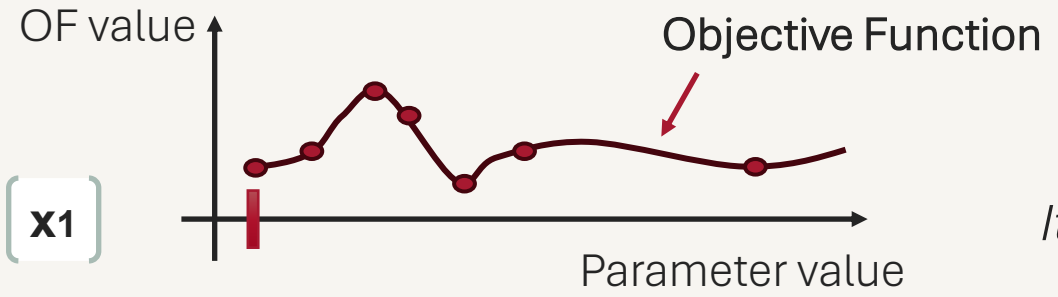


We just have two sliders to work with



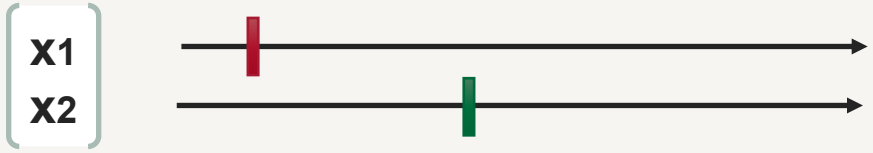
How to find the best solution?

Lets simplify to just one parameter:



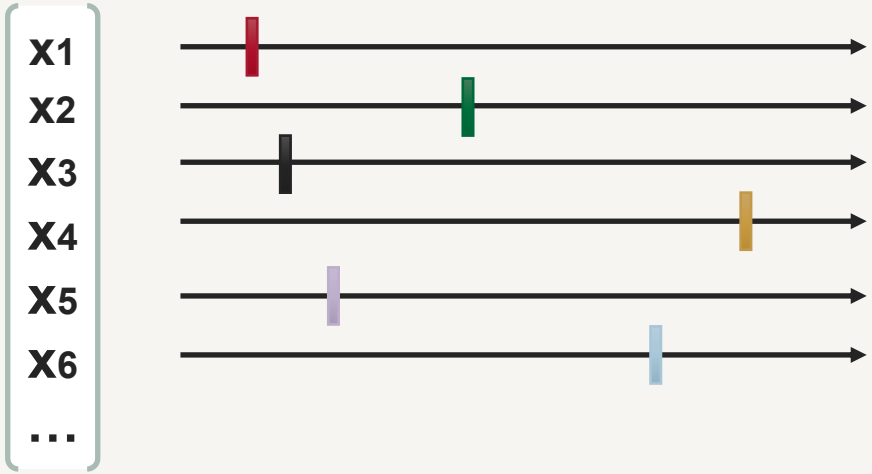
It is like we'd move a slider trying to find the best setup

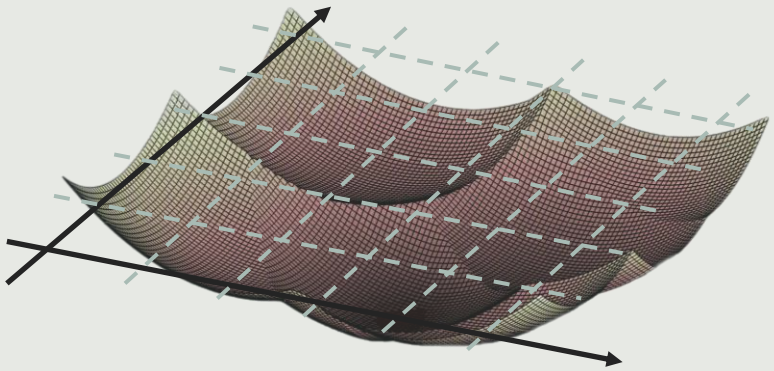
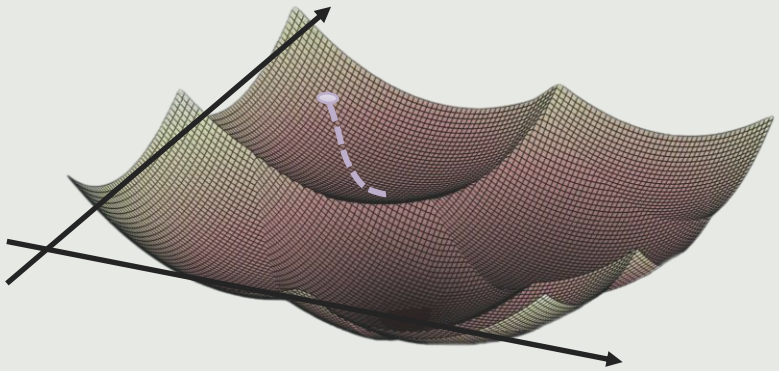
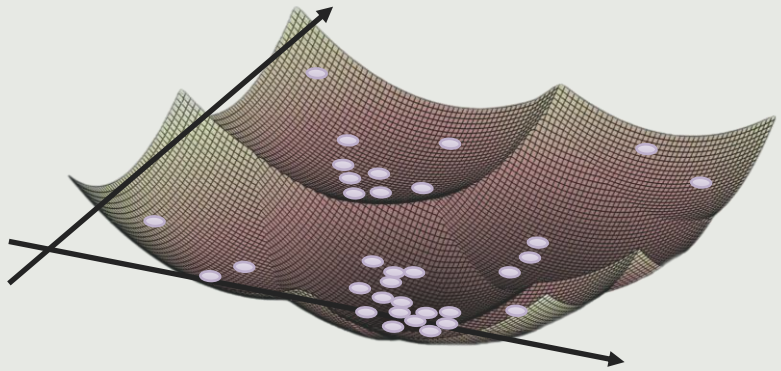
For two parameters, it works the same:



We just have two sliders to work with

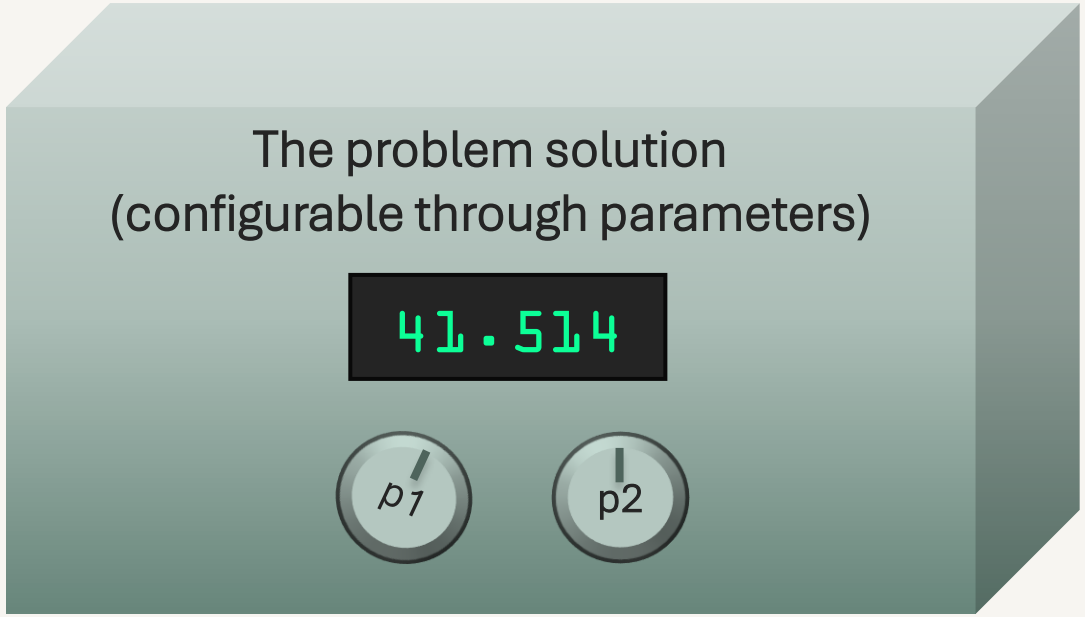
For more parameters, nothing changes, we just can't see the entire function

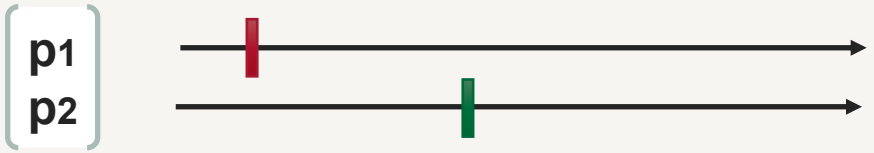


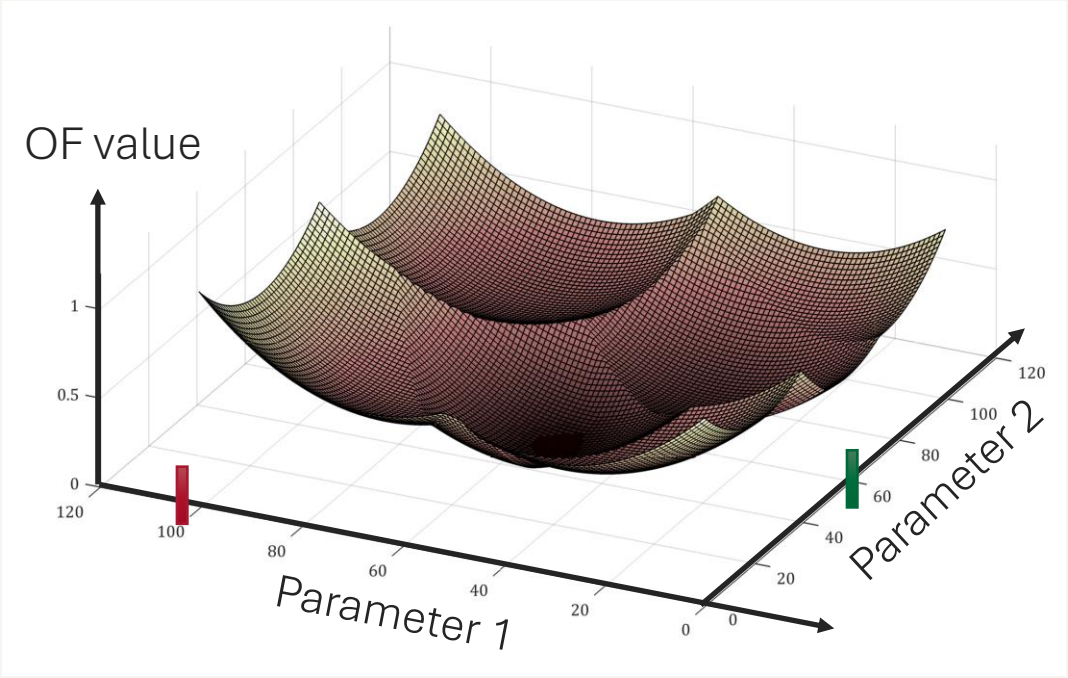
Grid search	Gradient-based	1+1
		
„lets methodically check the function”	*„lets start somewhere and then move down until we can’t improve”*	*„lets test different spots and gradually focus on those that feel right”*
- methodical! - ignores local minima - not scalable - can’t precisely locate minimum	- Fast - Surprisingly scalable - Requires differentiable objective function - May end up in a local minimum	- Does not scale terribly - Does not require differentiable objective function - Is not attracted by local minima (as much)

Lets consider this scalability of gradient-based methods first...

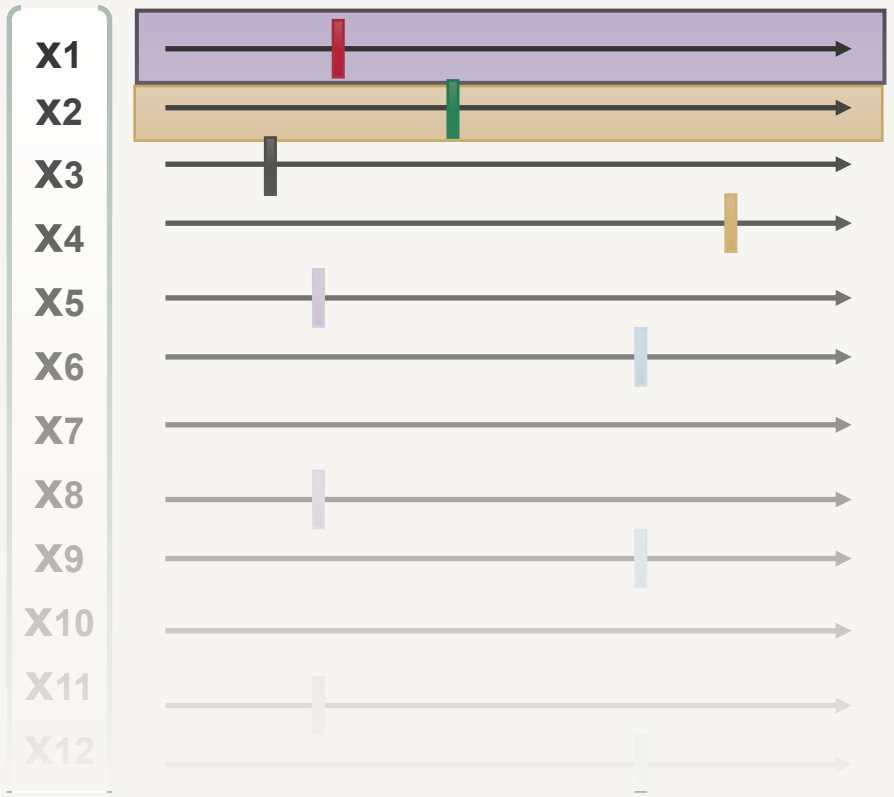
Because – how is it possible that you don't fall into local minimum training ChatGPT?!



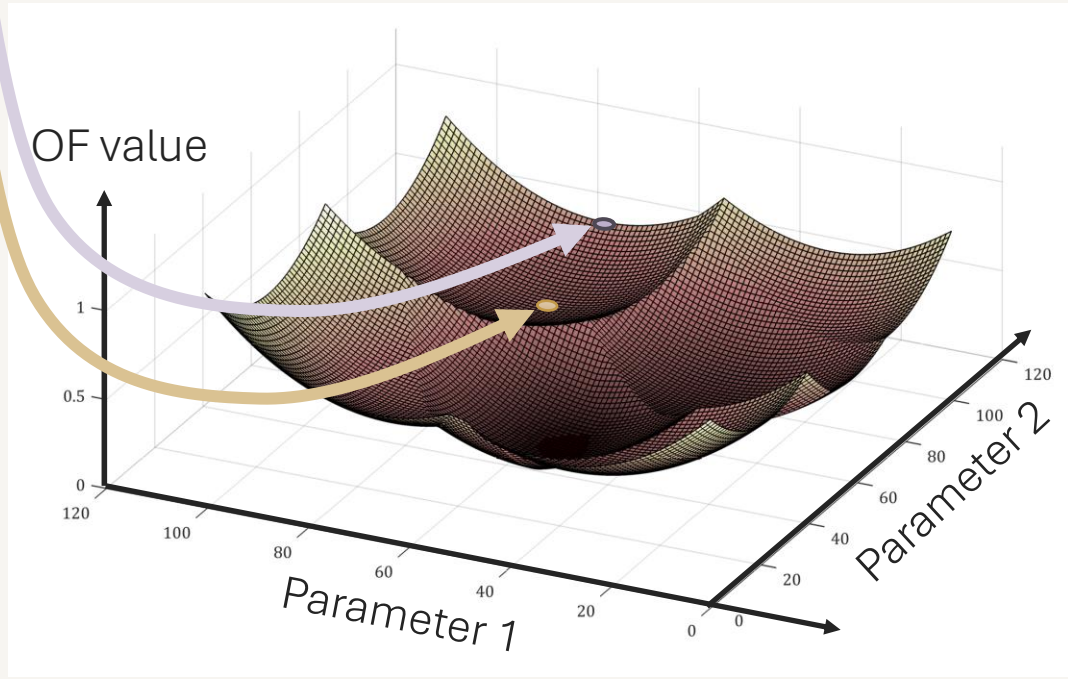




What does it mean „to hit a local minimum" in this context?



If any one of those can be slightly changed for improvement –
we are still NOT in the local minimum!



In high-dimensional parameter spaces „local minima" tend to change to „saddle points"

But there are classes of problems in which we either have these local minima anyway or for variety of reasons we can't really use gradient solutions...

evolutionary optimization

We generate random population of solutions (candidates)

We check how good the candidate solutions are

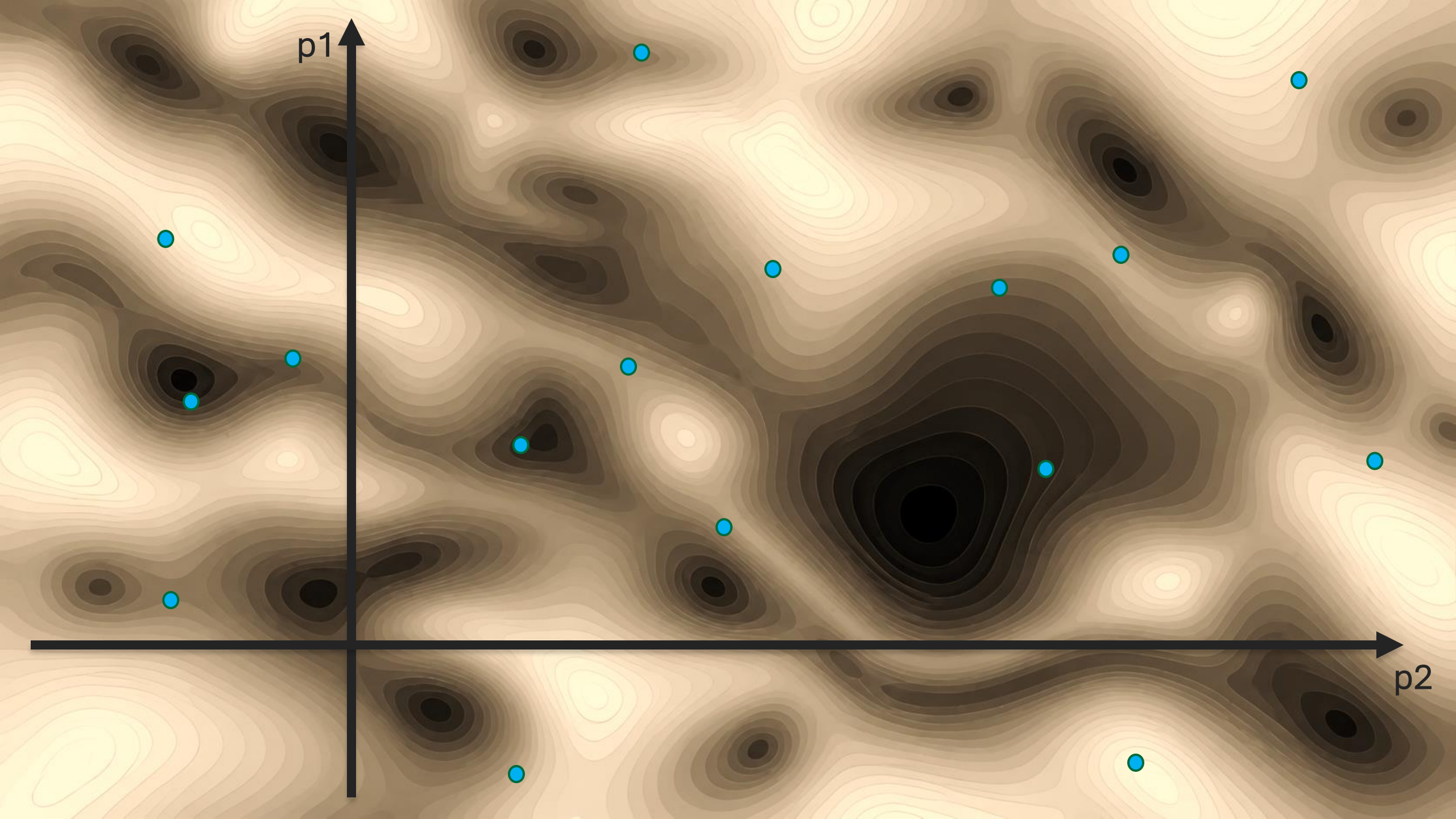
We select better part for the next generation

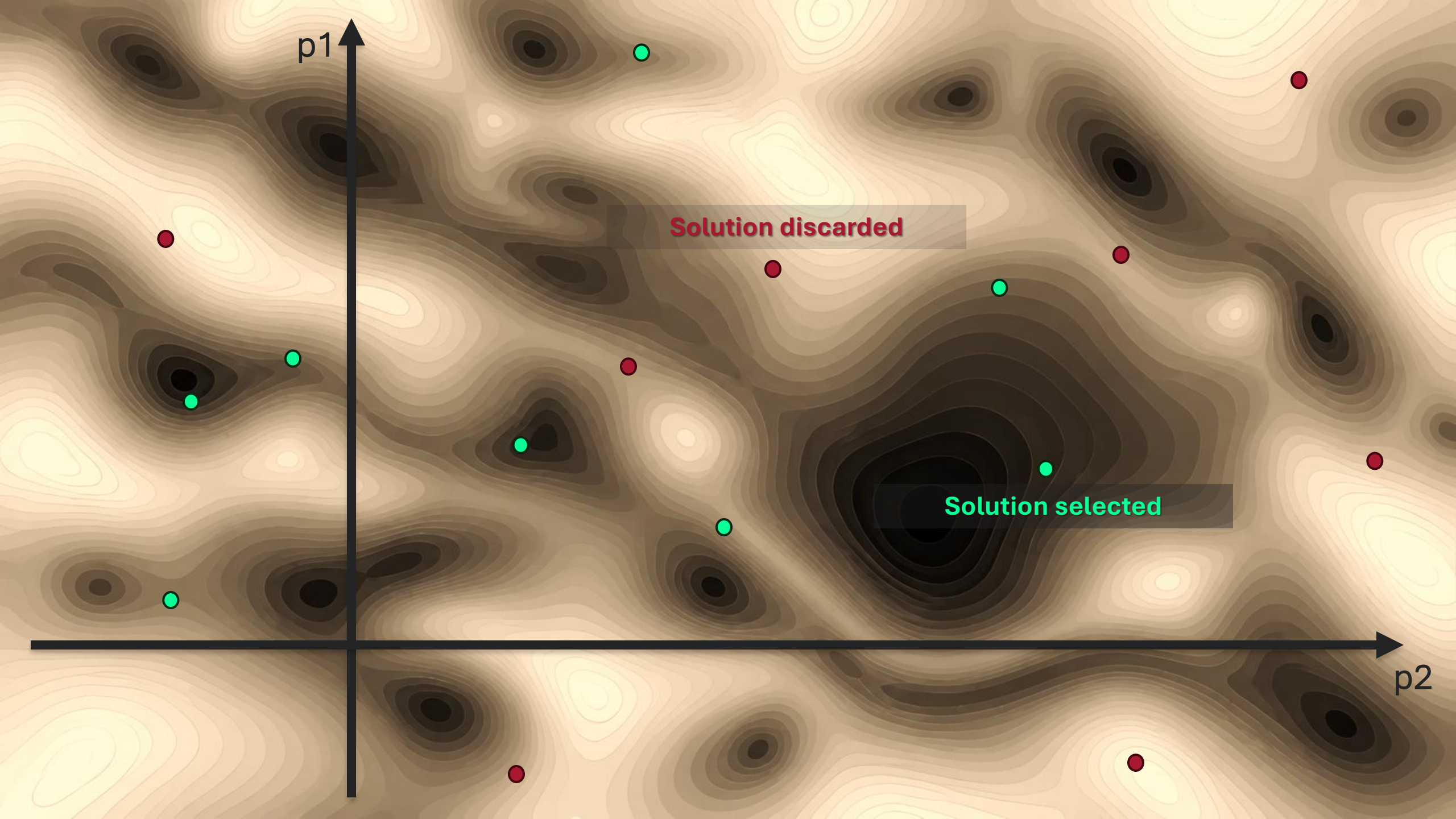
We create new population of candidates from successful individuals

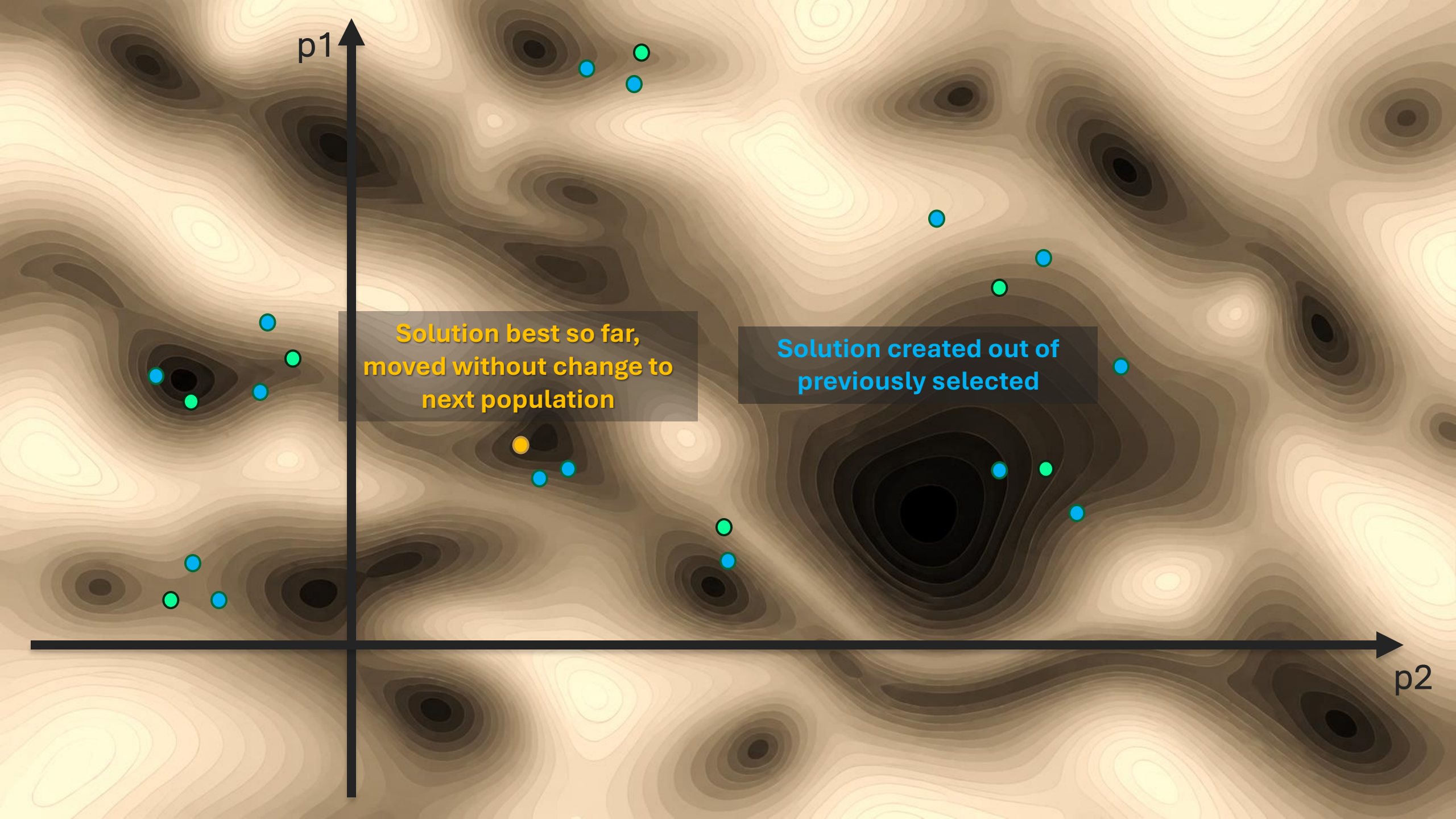


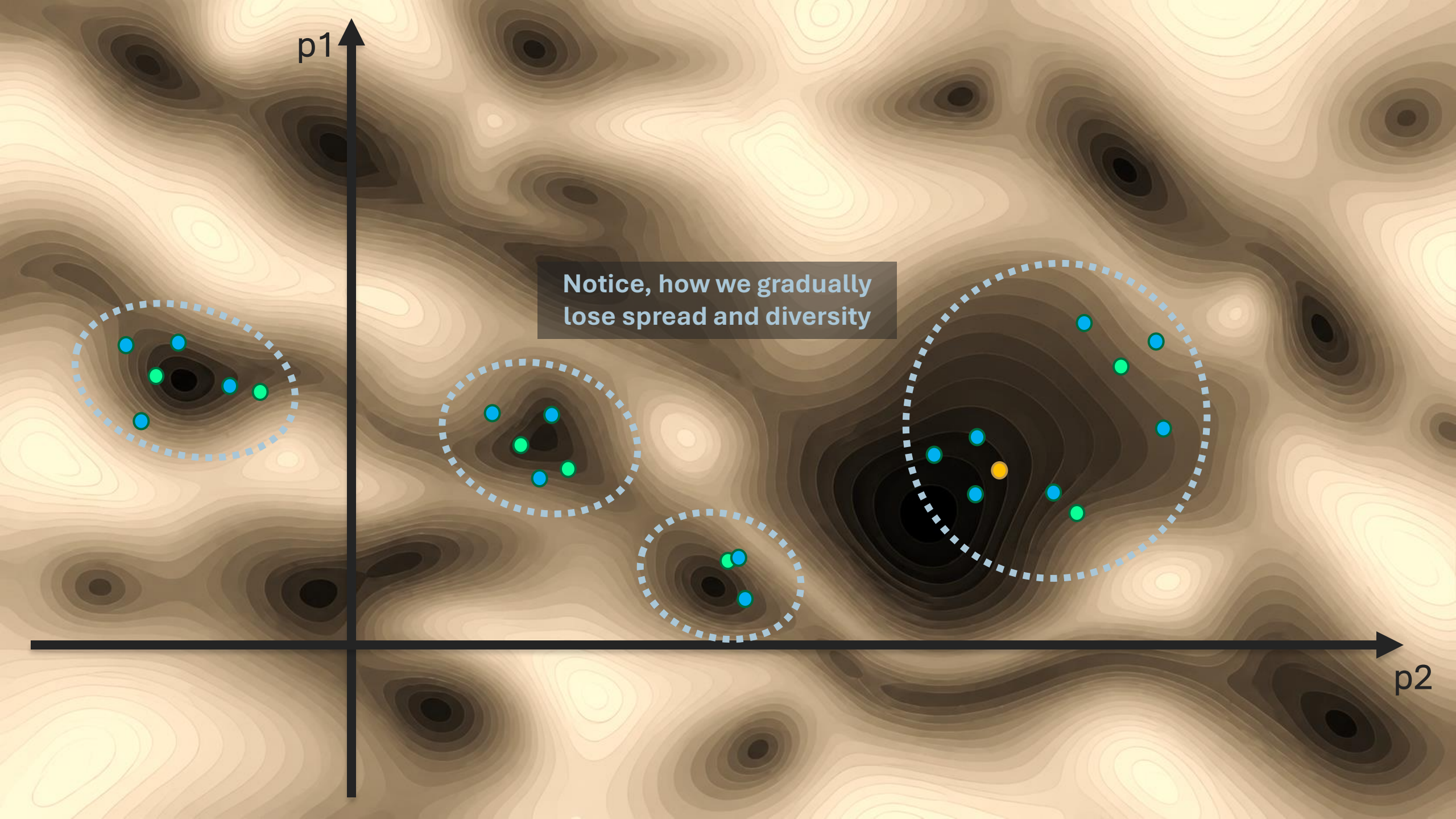
A very advanced solution to a very complex problem...

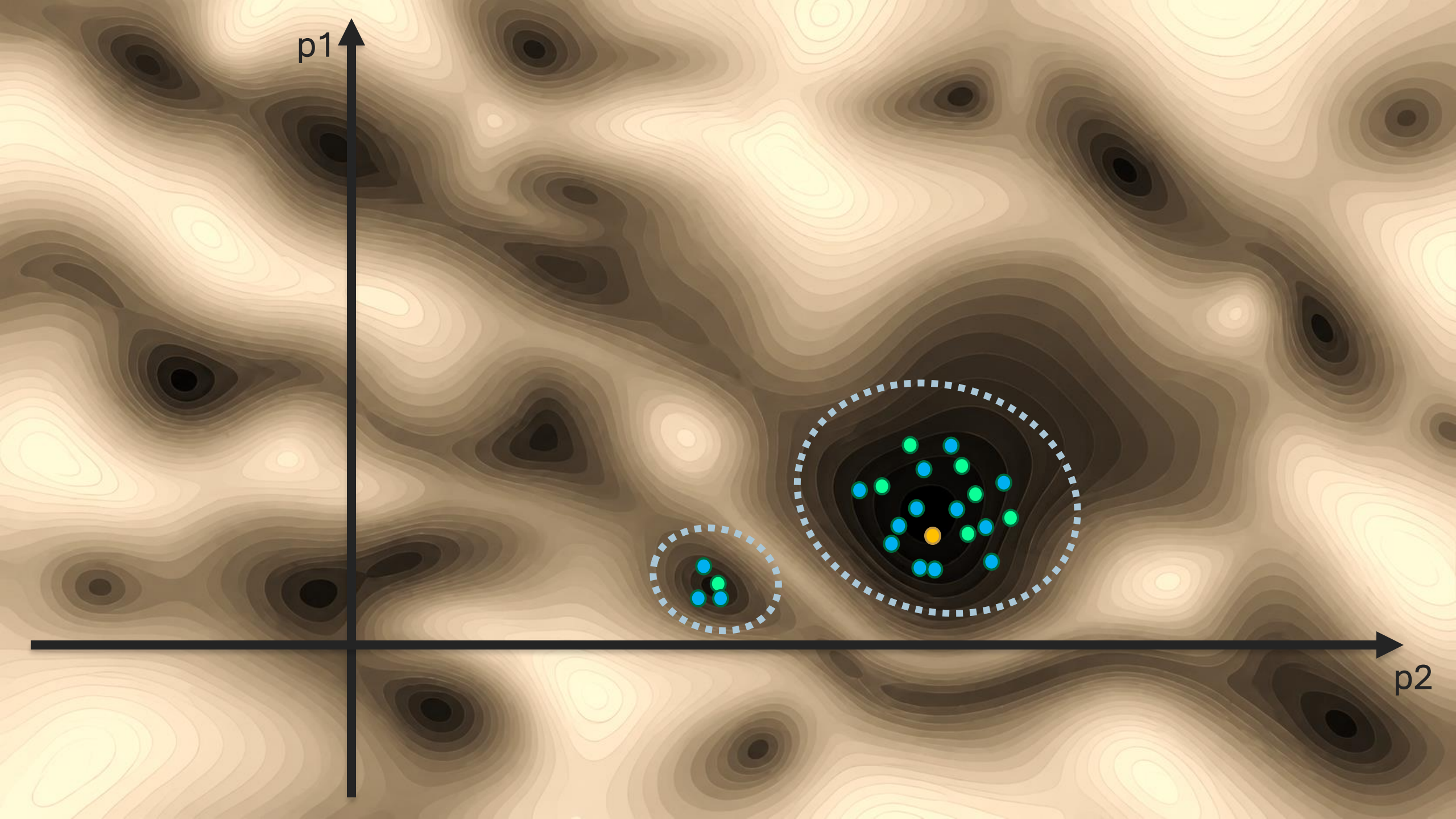
That started from a (relatively) simple premise

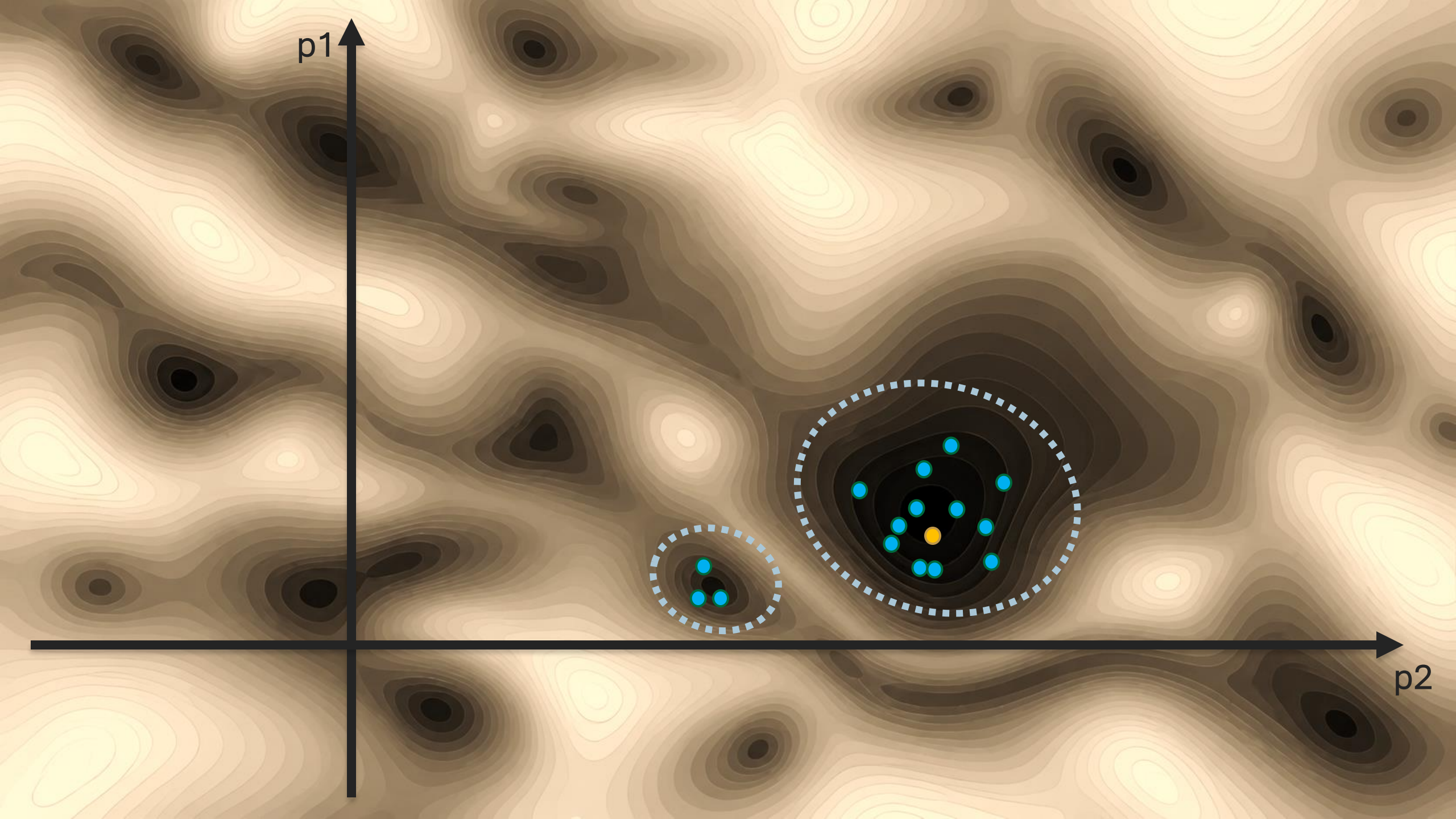

p1
p2

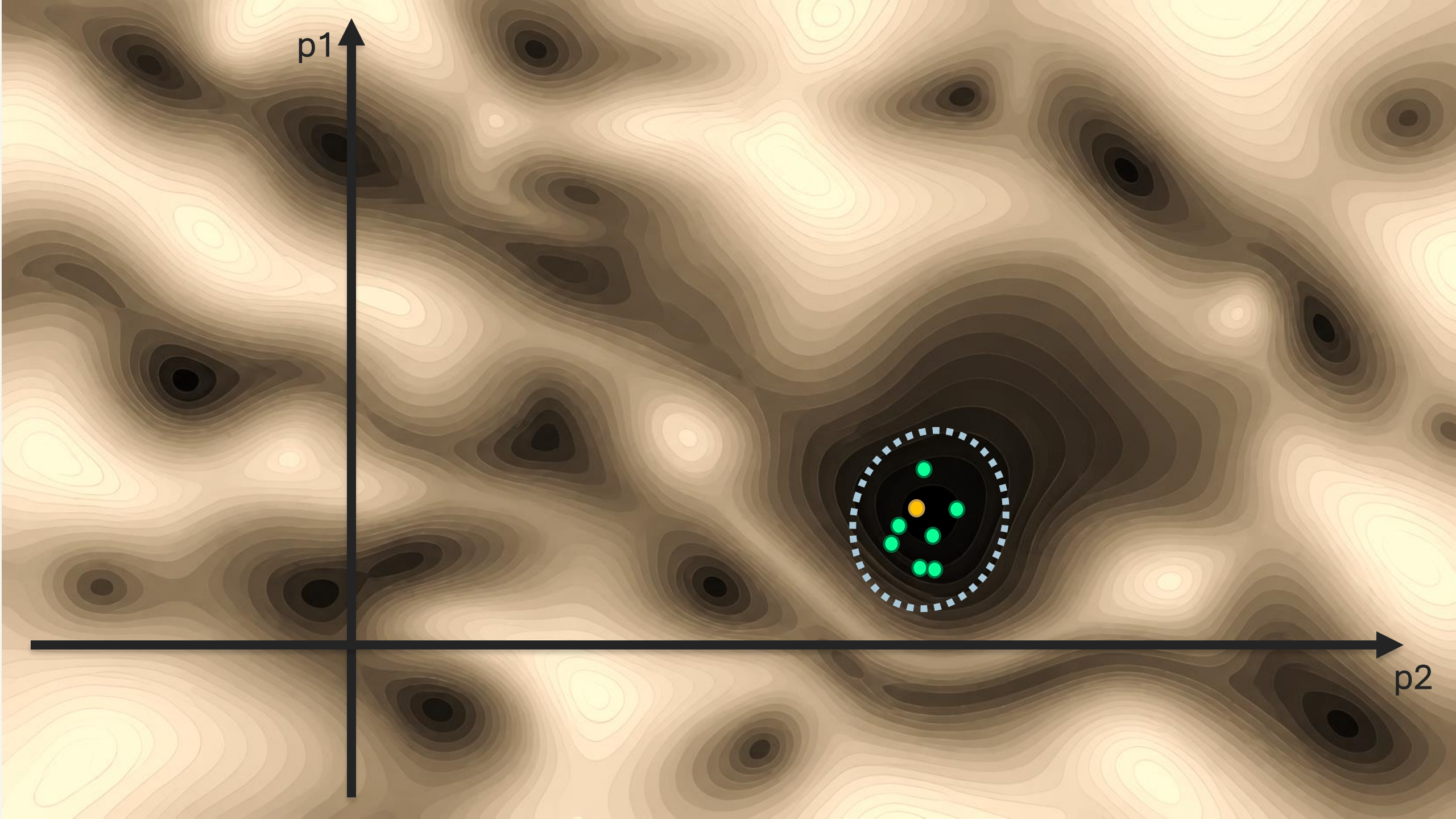

p1
p2
Solution discarded
Solution selected

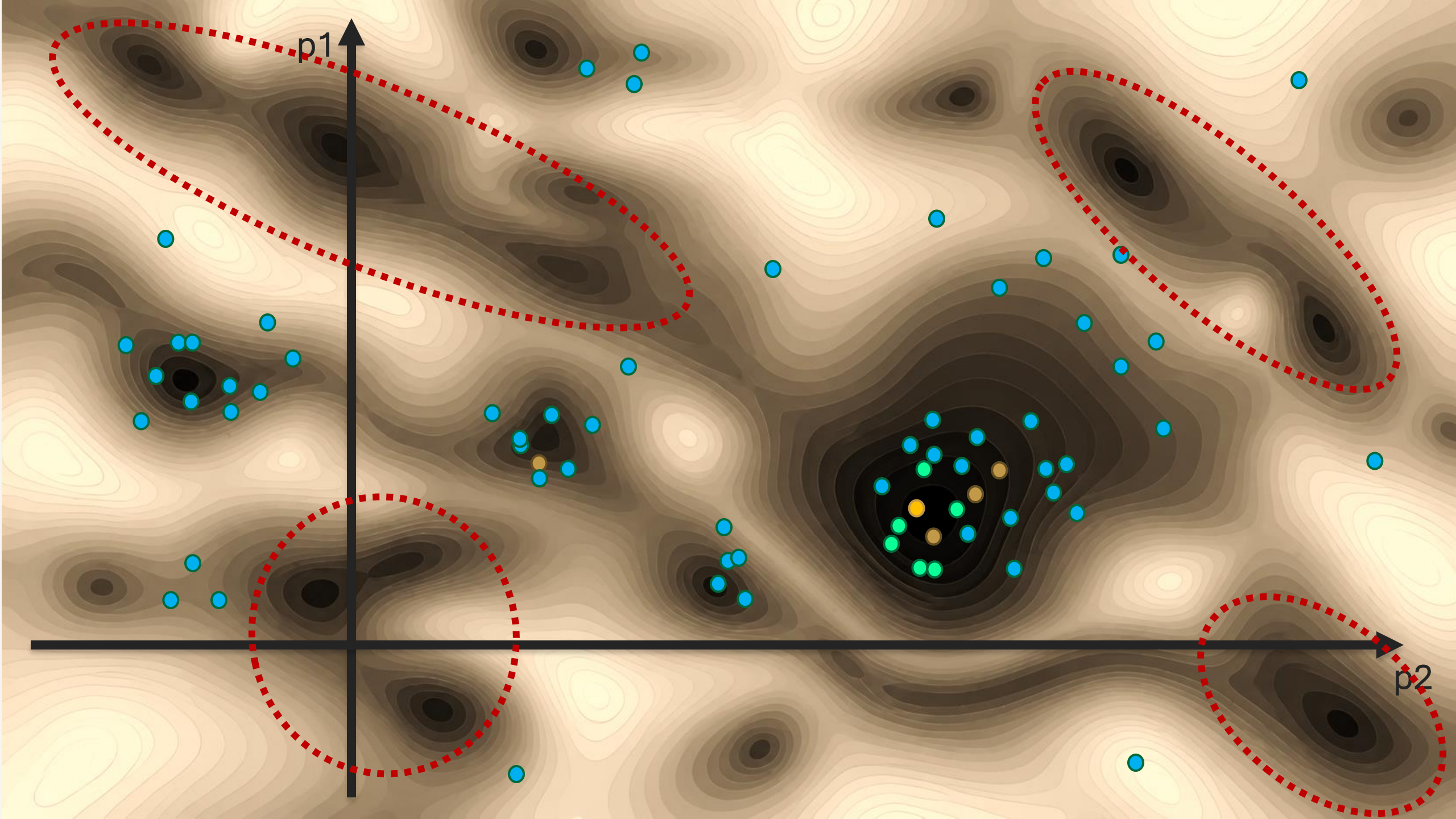

p1
p2
Solution best so far, moved without change to next population
Solution created out of previously selected


p1
p2
Notice, how we gradually
lose spread and diversity


p1
p2


p1
p2


p1
p2


p1
p2

Optimization of genetic operators for the problem type (selection, mutation)

Changes of convergence speed through selective pressure

Exploration/exploitation balance (prevent too fast focus on local minimum)

Cross-minima and cross-solution information exchange (crossover operators)

Diversity-maintenance procedures (prevens early clustering, forces search around various local minima)

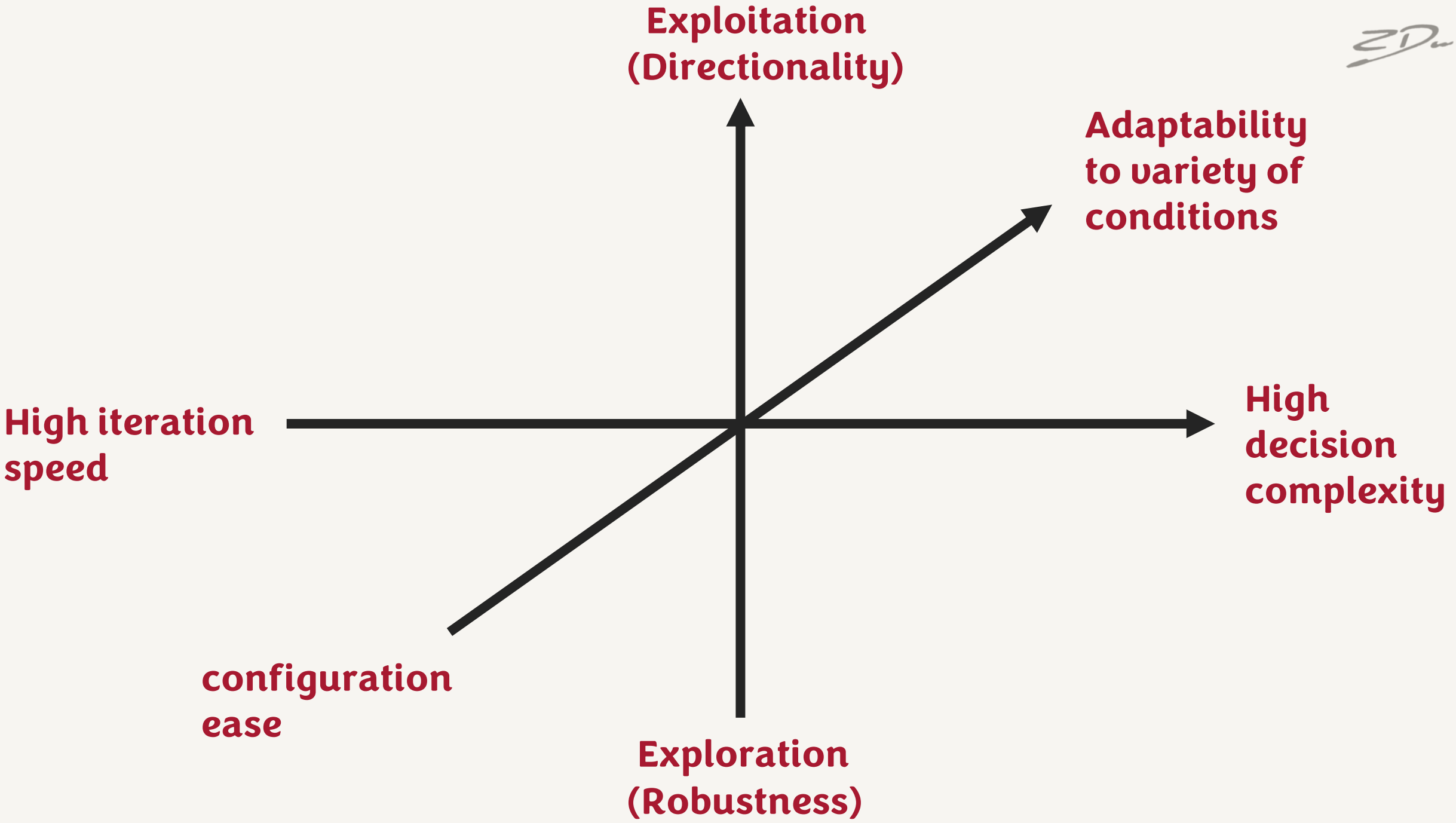
Phenotype adjustments depending on environment
(Ability to generate families of solutions from one individual)

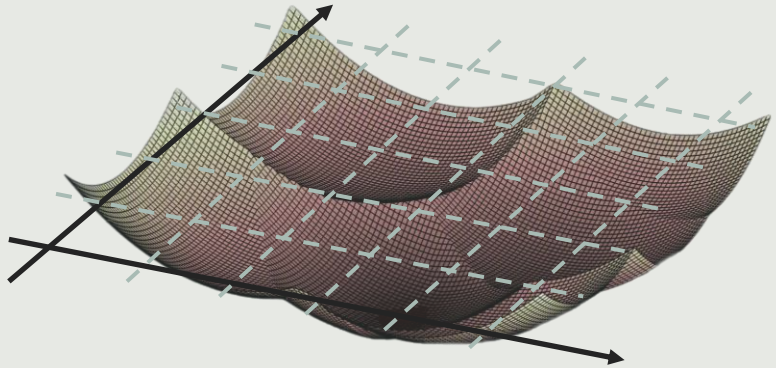
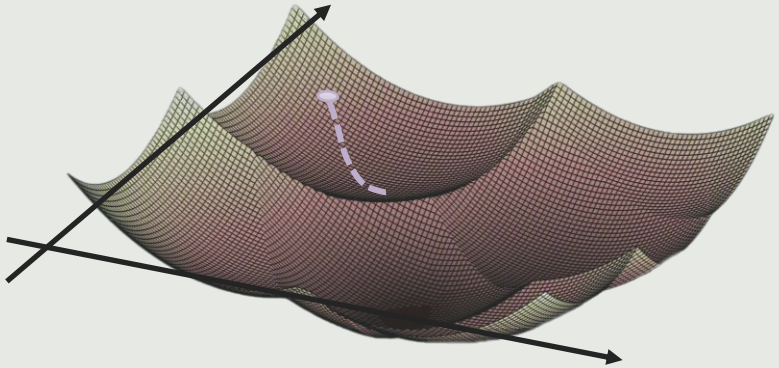
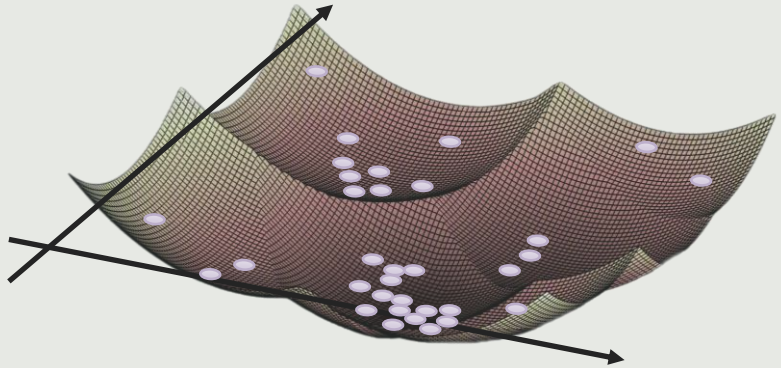
Local search operators (in genotype or phenotype space, searchable under *„memetic algorithm”*)

Its a basic concept. We can develop it further...

p2

Optimization of genetic operators for the problem type (selection, mutation)	Lots of metaparameters, potential for including context knowledge, physics-informed operators, etc... But requires having this knowledge in the first place!
Changes of convergence speed through selective pressure	Faster algorithms tend to be more robust – but get a bit worse results (or are slower)
Exploration/exploitation balance (prevent too fast focus on local minimum)	
Cross-minima and cross-solution information exchange (crossover operators)	Crossover speeds-up convergence IF parts of the genotype code separated components of phenotype
Diversity-maintenance procedures (prevens early clustering, forces search around various local minima)	Requires additional procedure to rank diversity (additional computational step)
Phenotype adjustments depending on environment (Ability to generate families of solutions from one individual)	Again: ability to introduce physical knowledge and on-the-fly adjustment of solutions but increases complexity even more
Local search operators (in genotype or phenotype space, searchable under „*memetic algorithm*”)	Combines strengths of gradient-based algorithms with evolutionary concept. But we risk complicating the solution and getting exactly similar (or worse) results


Exploitation
(Directionality)
Adaptability
to variety of
conditions
High iteration
speed
High
decision
complexity
configuration
ease
Exploration
(Robustness)

Grid search	Gradient-based	~~1+1~~ evolutionary
		
„lets methodically check the function"	*„lets start somewhere and then move down until we can't improve"*	*„lets test different spots and gradually focus on those that feel right"*
- methodical! - ignores local minima - not scalable - can't precisely locate minimum	- Fast - Surprisingly scalable - Requires differentiable objective function - May end up in a local minimum	- Does not scale terribly - Does not require differentiable objective function - Is not attracted by local minima (as much)

But better!

But better!

Artificial worlds

(exploratory analysis of solution space)

Artificial worlds

(exploratory analysis of solution space)

We can build models of society or nature and check how environmental parameters alter development of strategies and organisms

Scientific simulations for evolution of complex functions and organisms, e.g. *https://www.nature.com/articles/nature01568*

Primer *https://www.youtube.com/@PrimerBlobs*

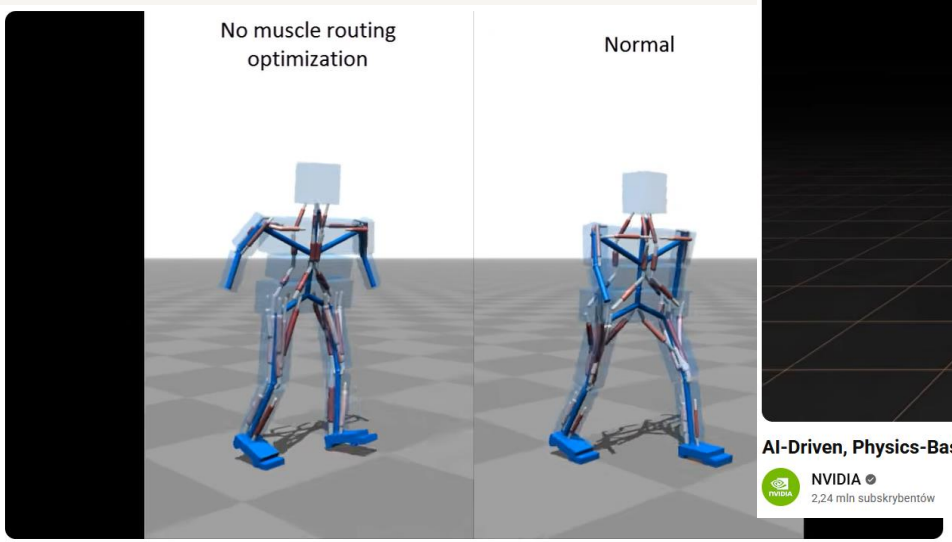
Karl Sims virtual creatures (1994) *https://youtu.be/RZtZia4ZkX8*

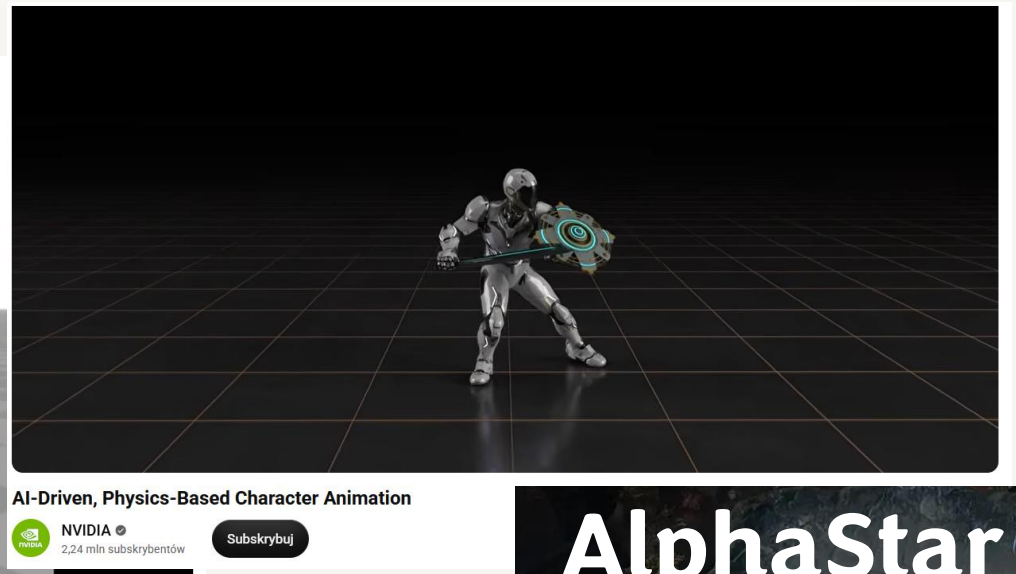
„Predator vs Prey, who wins?" *https://www.youtube.com/watch?v=qwrp3lB-jkQ*

Look how fun it looks!

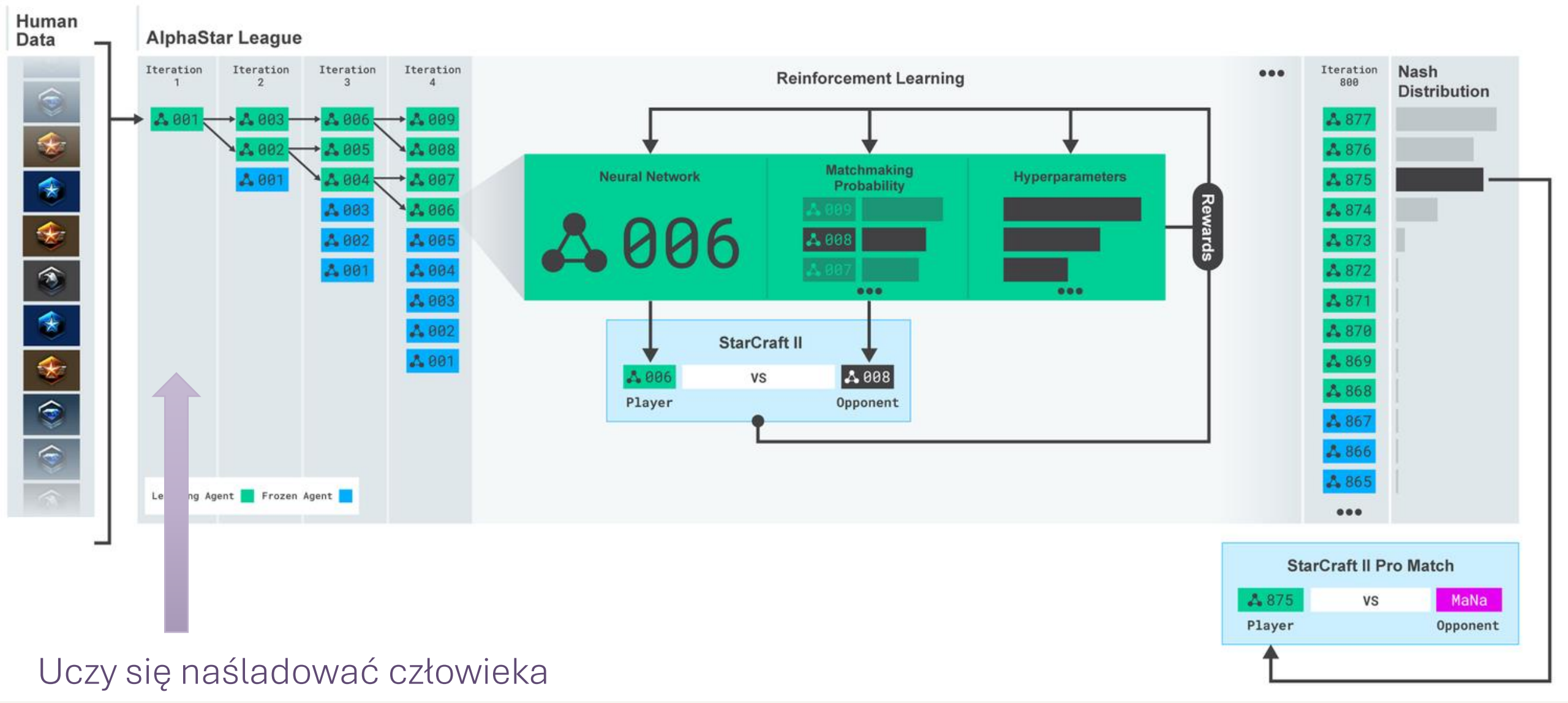
High-complexity optimization

(reinforcement learning, hybrid solutions, etc.)



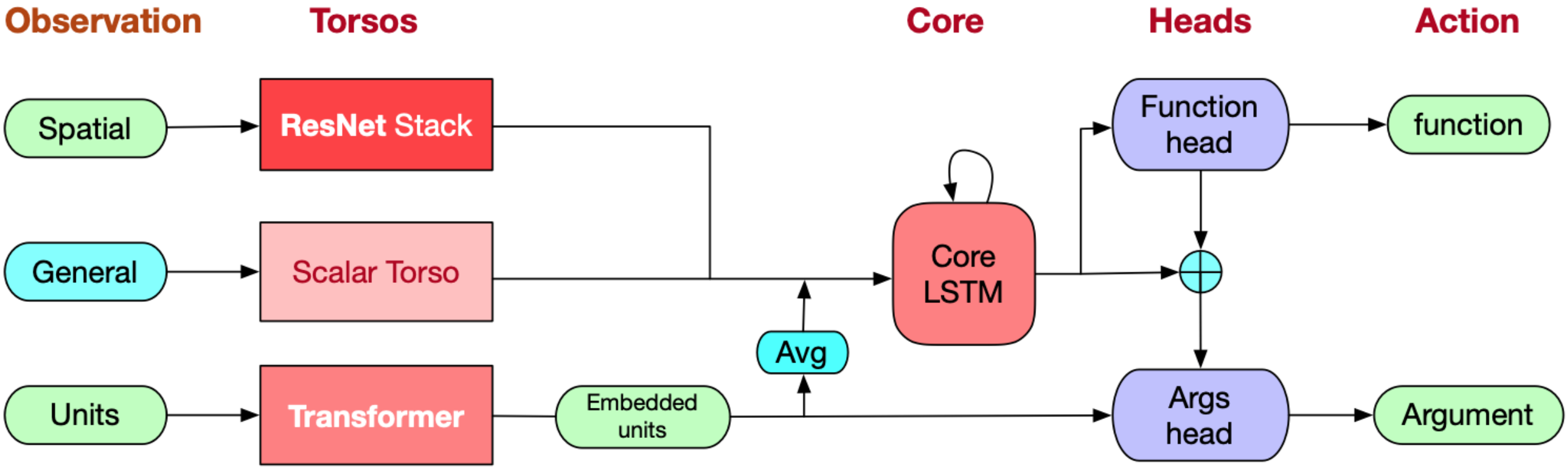




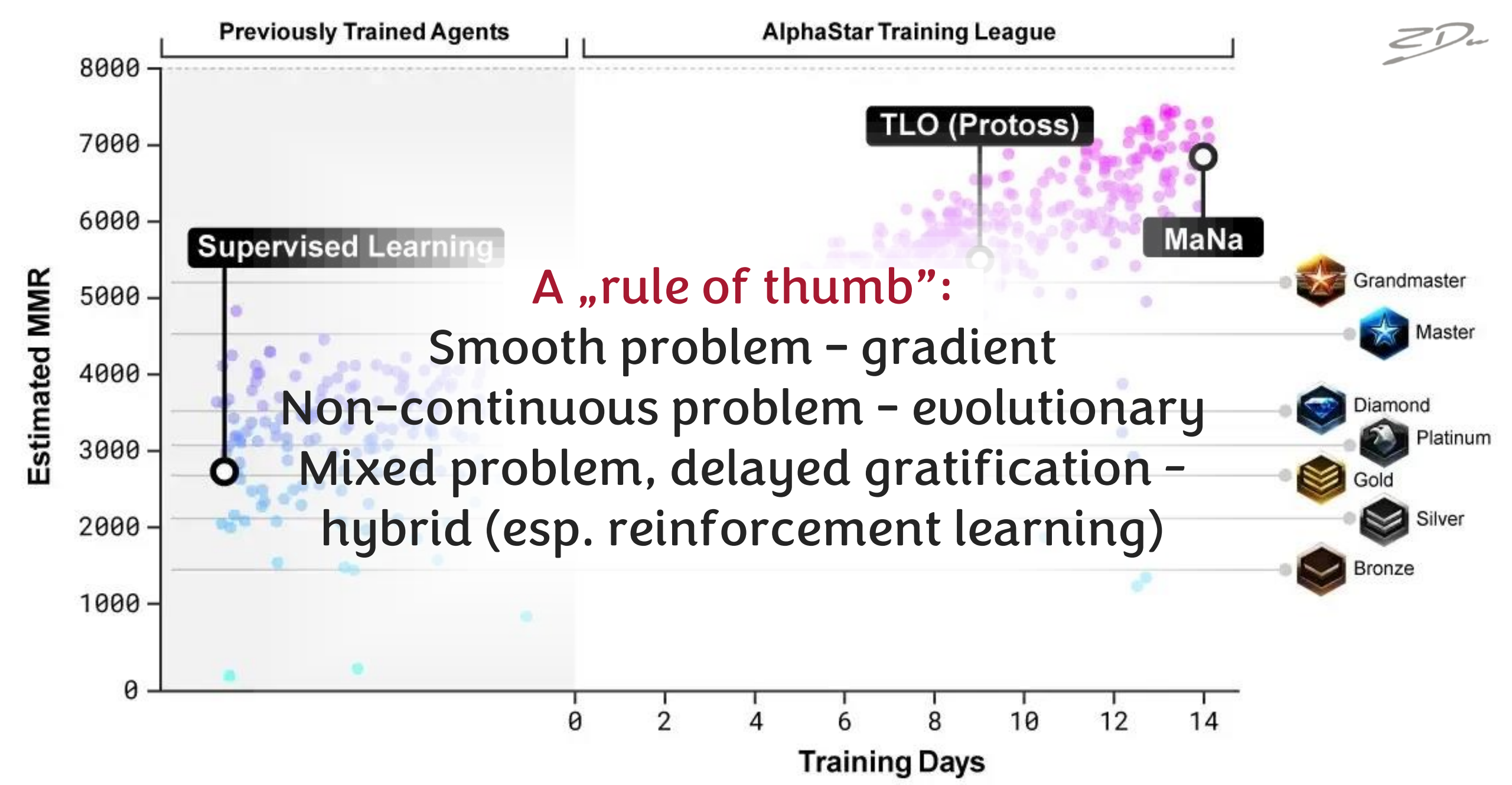


Uczy się naśladować człowieka

Source: https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/



Source: https://paperswithcode.com/method/alphastar



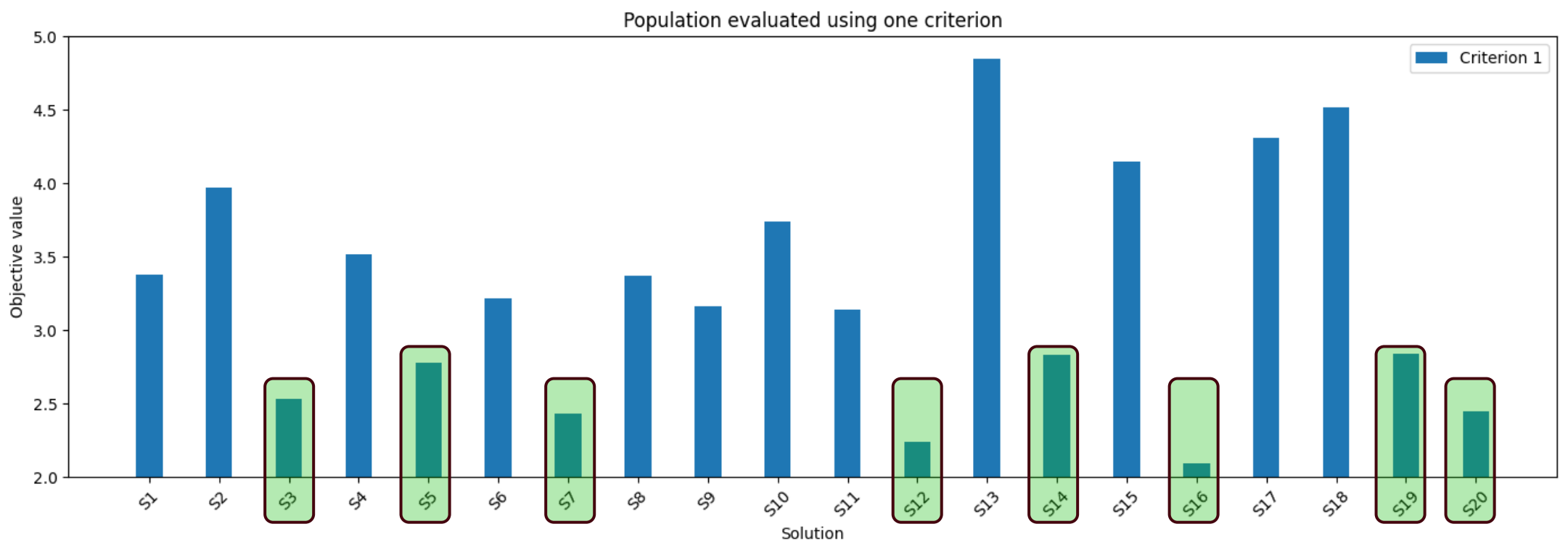
A „rule of thumb”:

Smooth problem – gradient

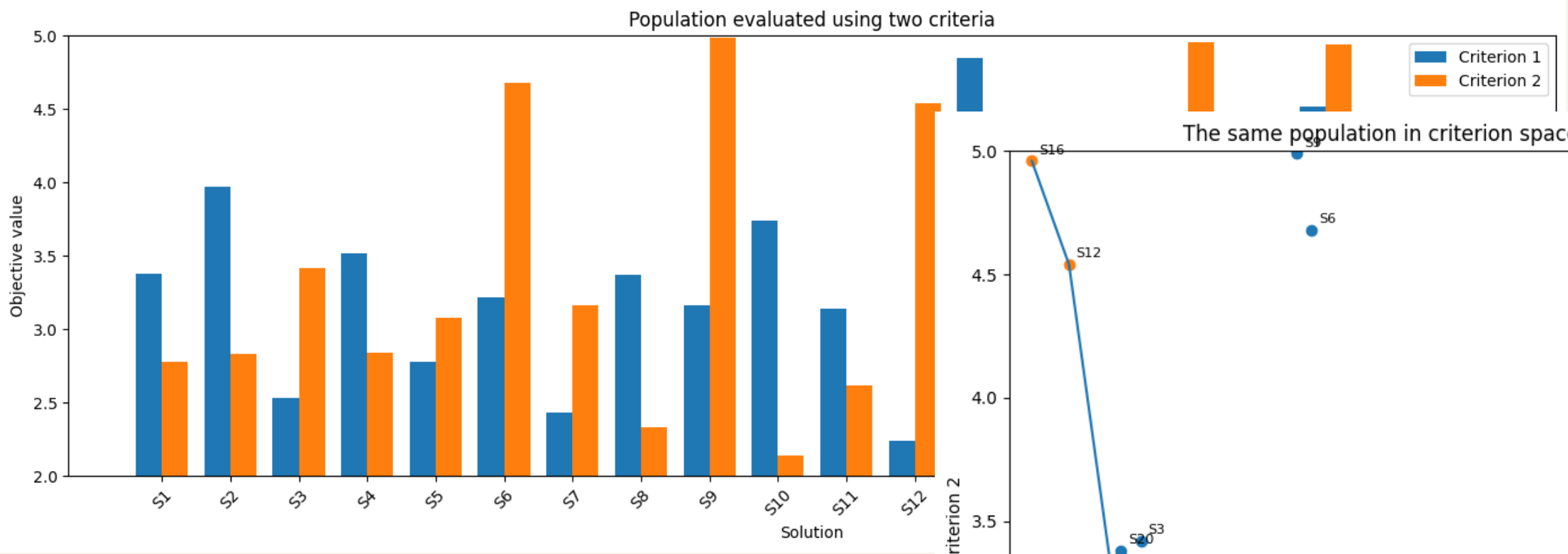
Non-continuous problem – evolutionary

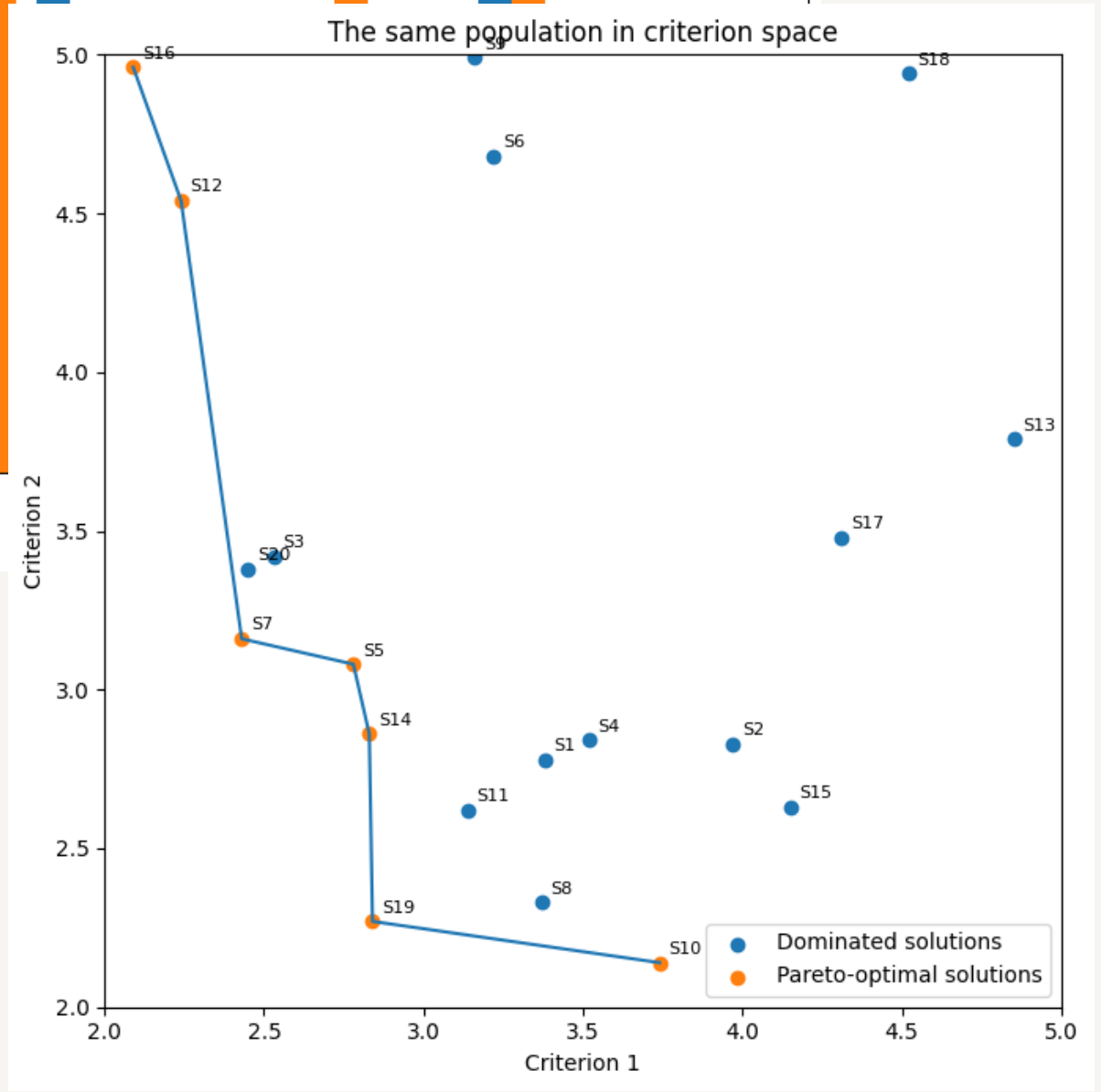
Mixed problem, delayed gratification – hybrid (esp. reinforcement learning)

Source: https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/

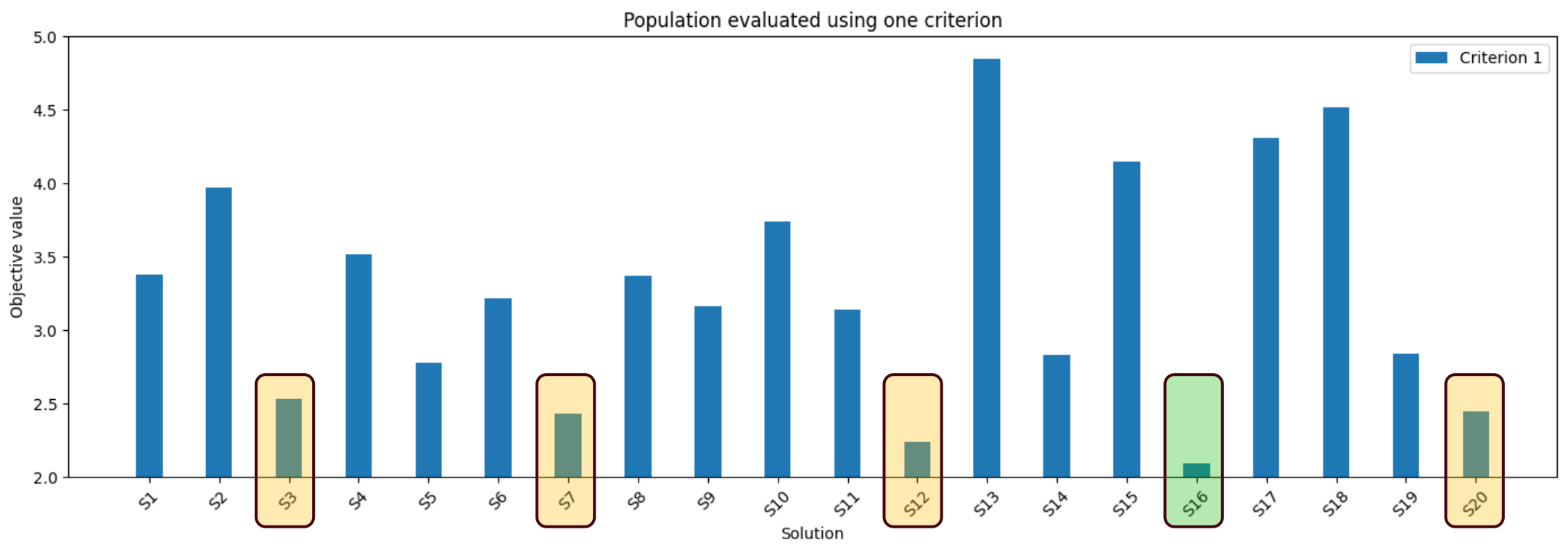


Population-as-a-solution concept





Multi-criterion optimization is a natural candidate for evolutionary solutions...



Evolutionary optimization provides „backup solutions”
if for some reason we can’t fully commit to the „best one”

e.g., if prevalence of best solution
makes it comparatively worse… (?!)

Bio-inspired solutions

(Just for the sake of being bio-inspired)

(I call them „zoo algoritms”)*

*potential biases incoming (probably)...

Examples include:

PSO (Particle swarm optimization)

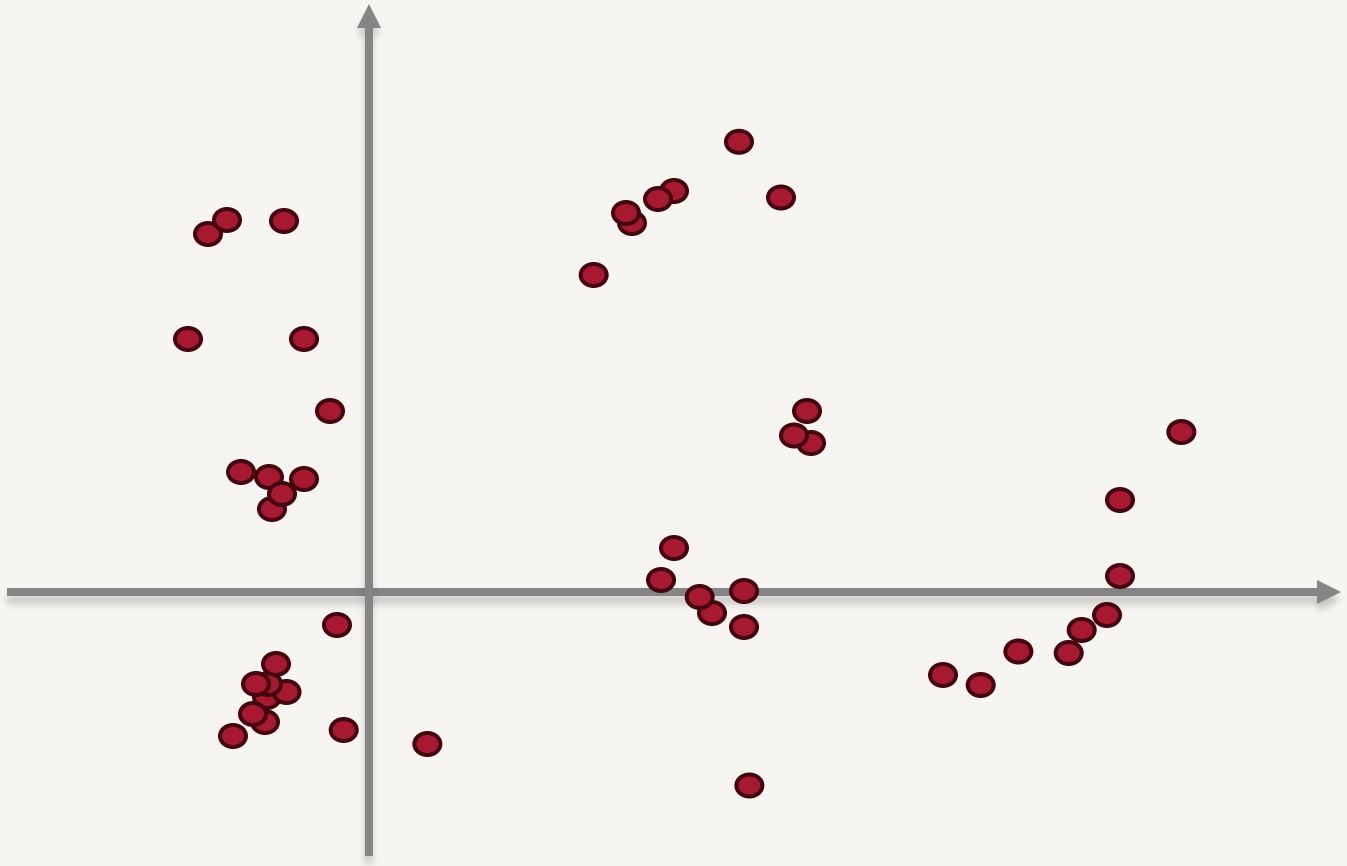
Ant-colony optimization *(pheromone for keeping partial solutions)*
Artificial Bee Colony *(different search roles)*
Bat Algorithm *(echolocation equivalent)*
Whale optimization *(agent cooperation converging on solutions)*
Moth-Flame optimization

...and many, many more...

I don't think any one of these consistently outperforms any „normal" algorithm under any general criteria...
Or at least - I didn't found one

Why intuitions may mislead you?

negative selection

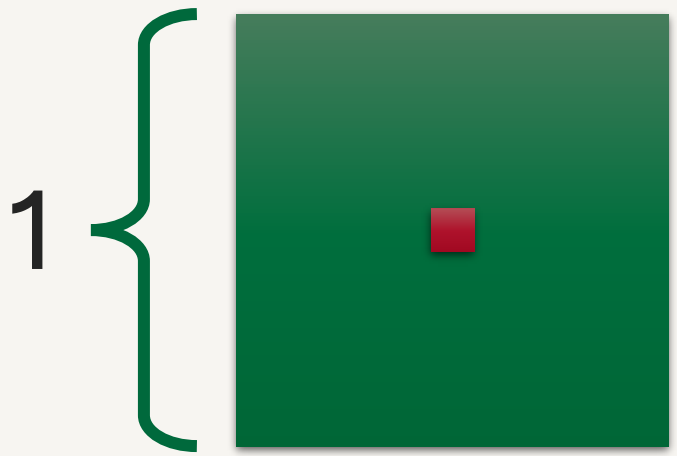


negative selection

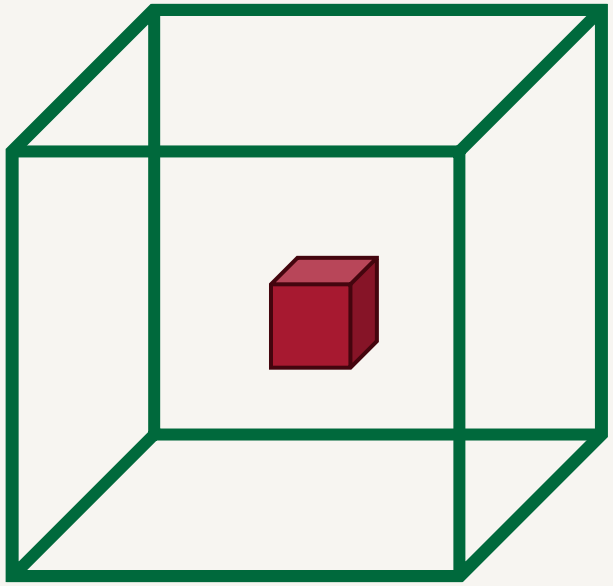


Looks good ...?

Data are uniformly distributed in feature space.
We aim to cover 1% of data (volume). What's the required neighborhood size relative to data range?



a = 10%



a = 21%

10D hypercube

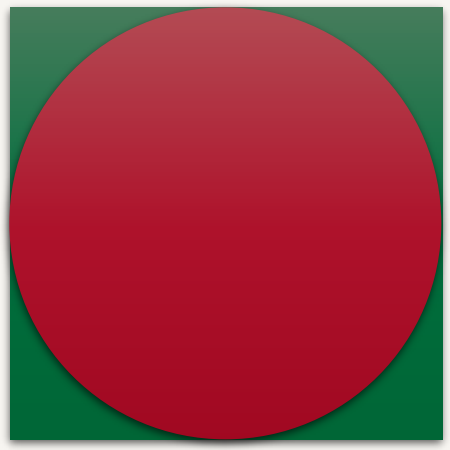
a = 63%

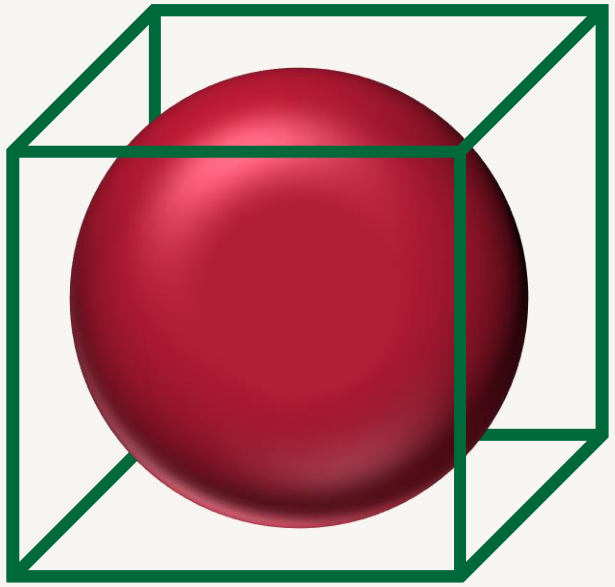
100D hypercube

a = 95%

We are filling the feature space with hypersphere (fitted to size of the data range). How much space is left outside of sphere?

1





10D hypercube/ hypersphere

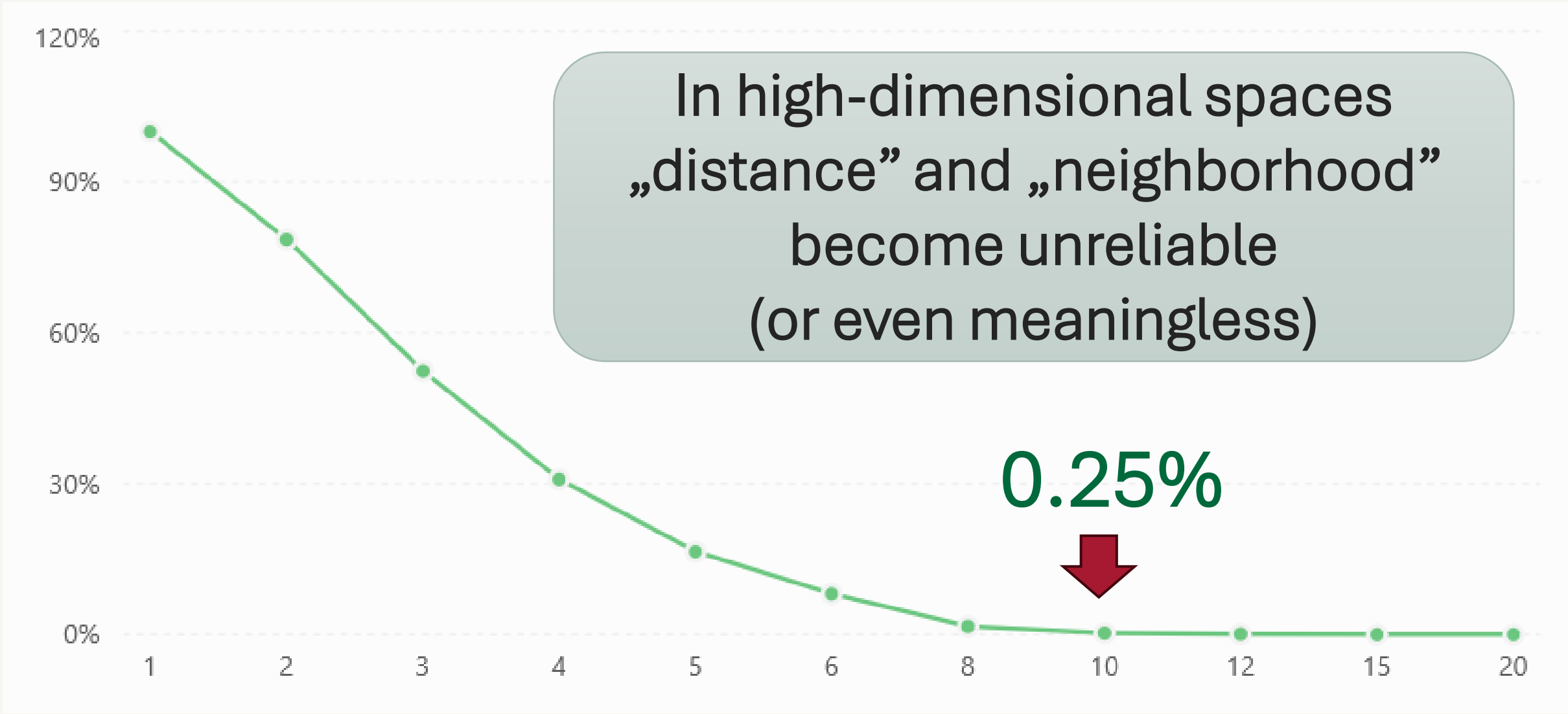
Ag = 1

Ar ~ 0.78 (78%)

Vg = 1

Vr ~ 0.52 (52%)

a = 0.25%


120%
90%
60%
30%
0%
1
2
3
4
5
6
8
10
12
15
20
In high-dimensional spaces
„distance” and „neighborhood”
become unreliable
(or even meaningless)
0.25%

Why you should use evolutionary computation?

You don’t need to worry about gradients, but you CAN use them if you have them

You can search space you have, not the differentiable approximation you wish you had

You can adjust the method to your needs

You can freely add context knowledge, physical operators and local search if necessary

You can be very robust to local minima

You can exploit parallelism, allow solutions to learn from one another and keep options open

Stop forcing the problem to fit the optimizer.
Design optimizer for the problem you actually have...

Why you should NOT use evolutionary computation?

If you can calculate gradient (or its approximate) – its usually much more efficient

High dimensionality punishes blind exploration

Majority of problems are solvable using simpler methods (esp. after representation change) (Occam's razor: we should prefer easier ones)

Its easy to confuse „works" with „works well"

Population size and operators often hide actual computational cost

Its difficult to prove they actually work better (consistently, without tedious setup)

Don't invest more than you have to.
Optimizer should also be optimized...

Optimization methods are not magic search engines. They are different **ways of spending computational effort** to explore a design space. The engineer's task is to understand whether the exploration strategy matches the structure of the problem.