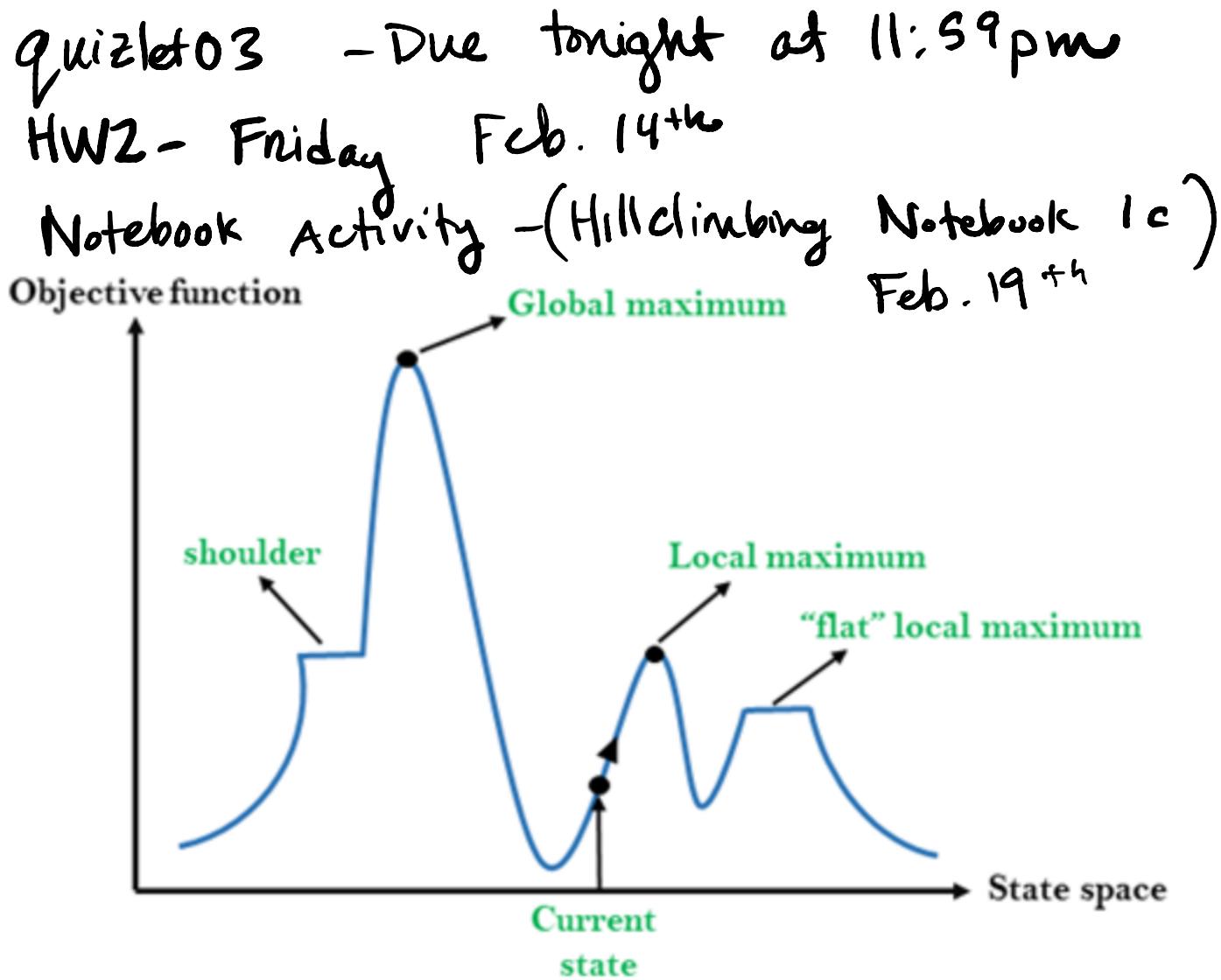


# CSCI 3202: Intro to Artificial Intelligence

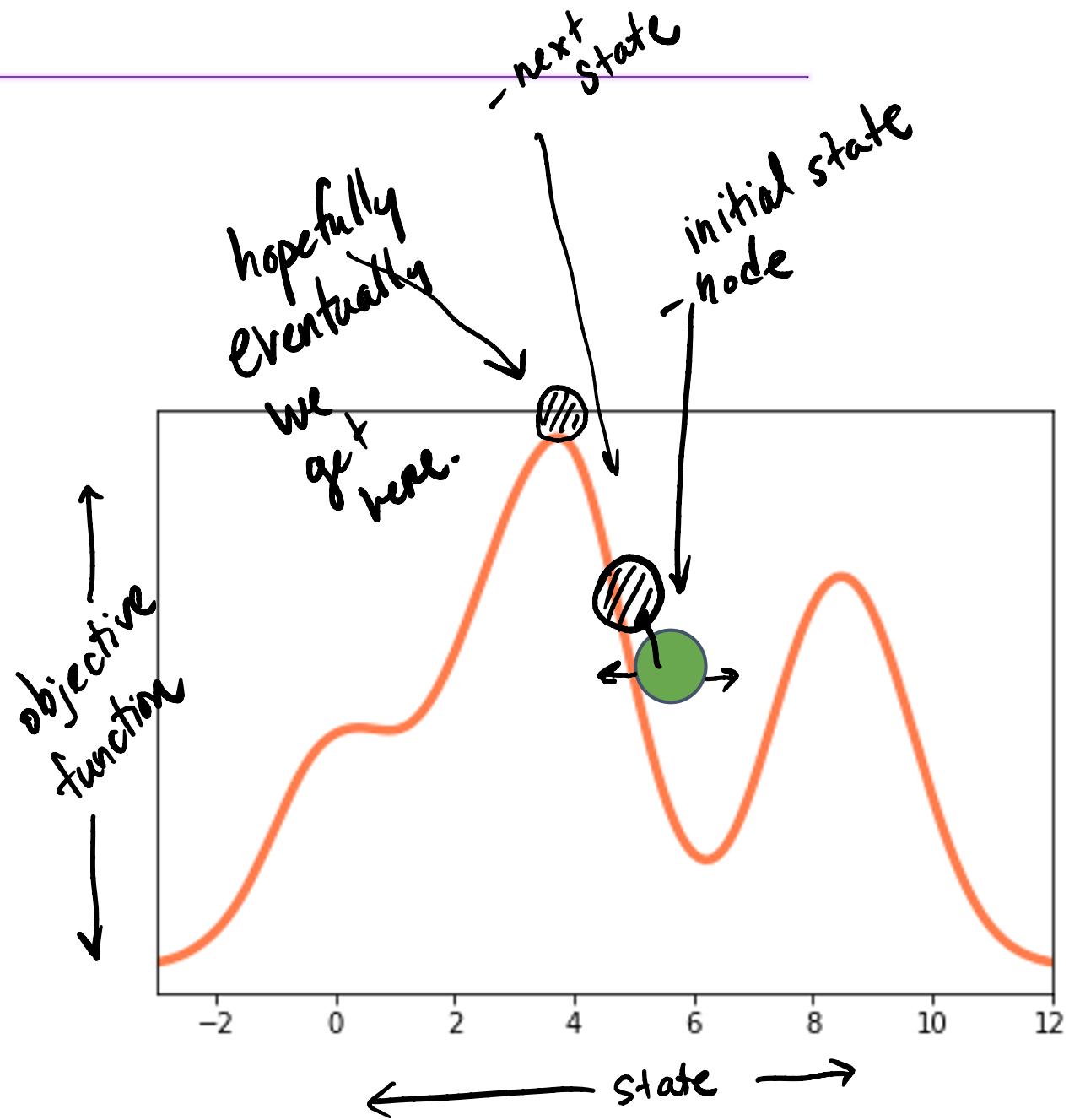
## Lecture 10: Local Search

Rachel Cox  
Department of Computer Science



# Local Search

Suppose the path to the goal doesn't matter to us.



# Local Search

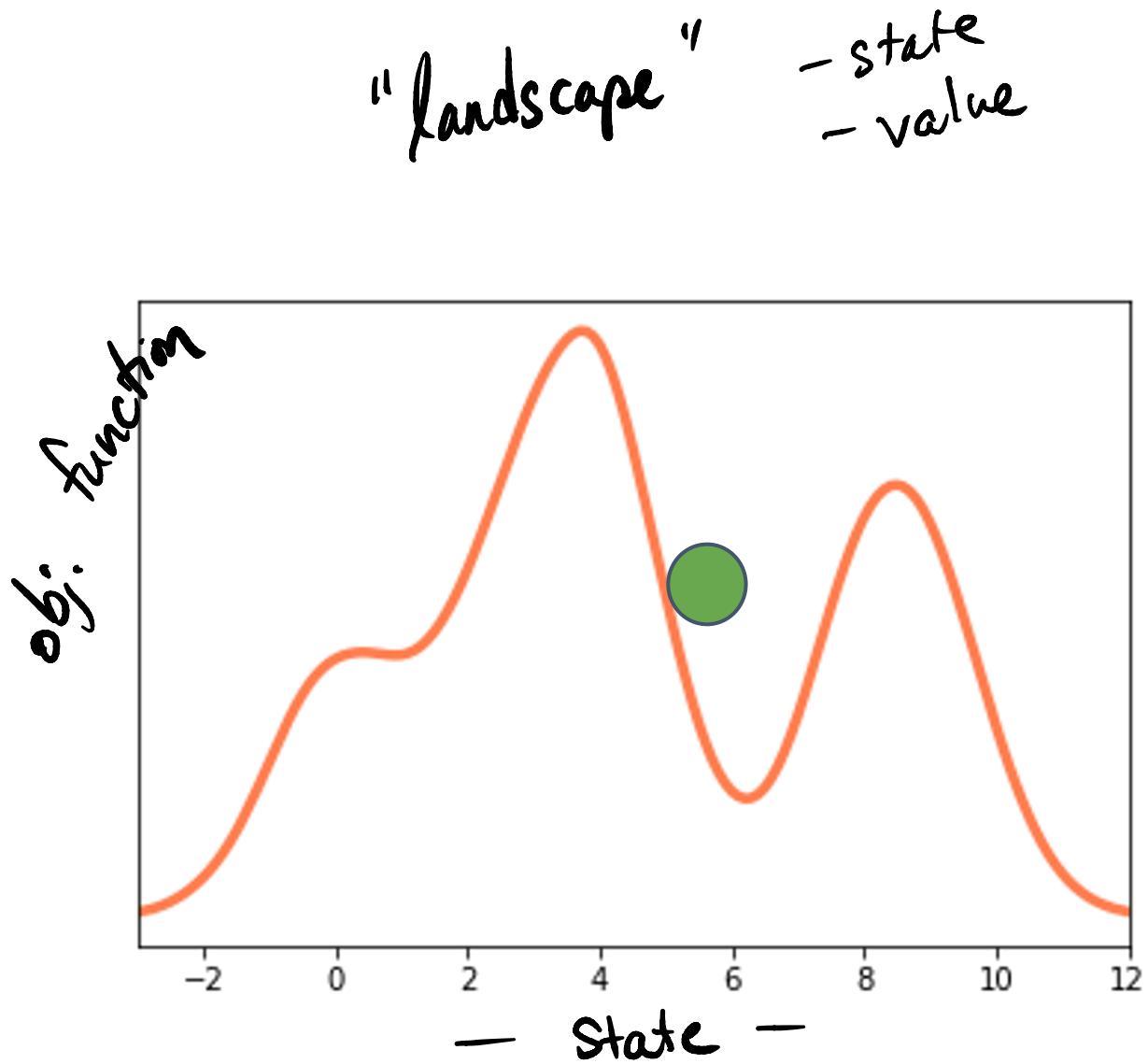
---

Local search:

- Use only the current node/state
- And consider only moves to neighbor states
- Super memory efficient (only care about current state)
- Used widely for **optimization**

Find the “best” state according to some **objective function**

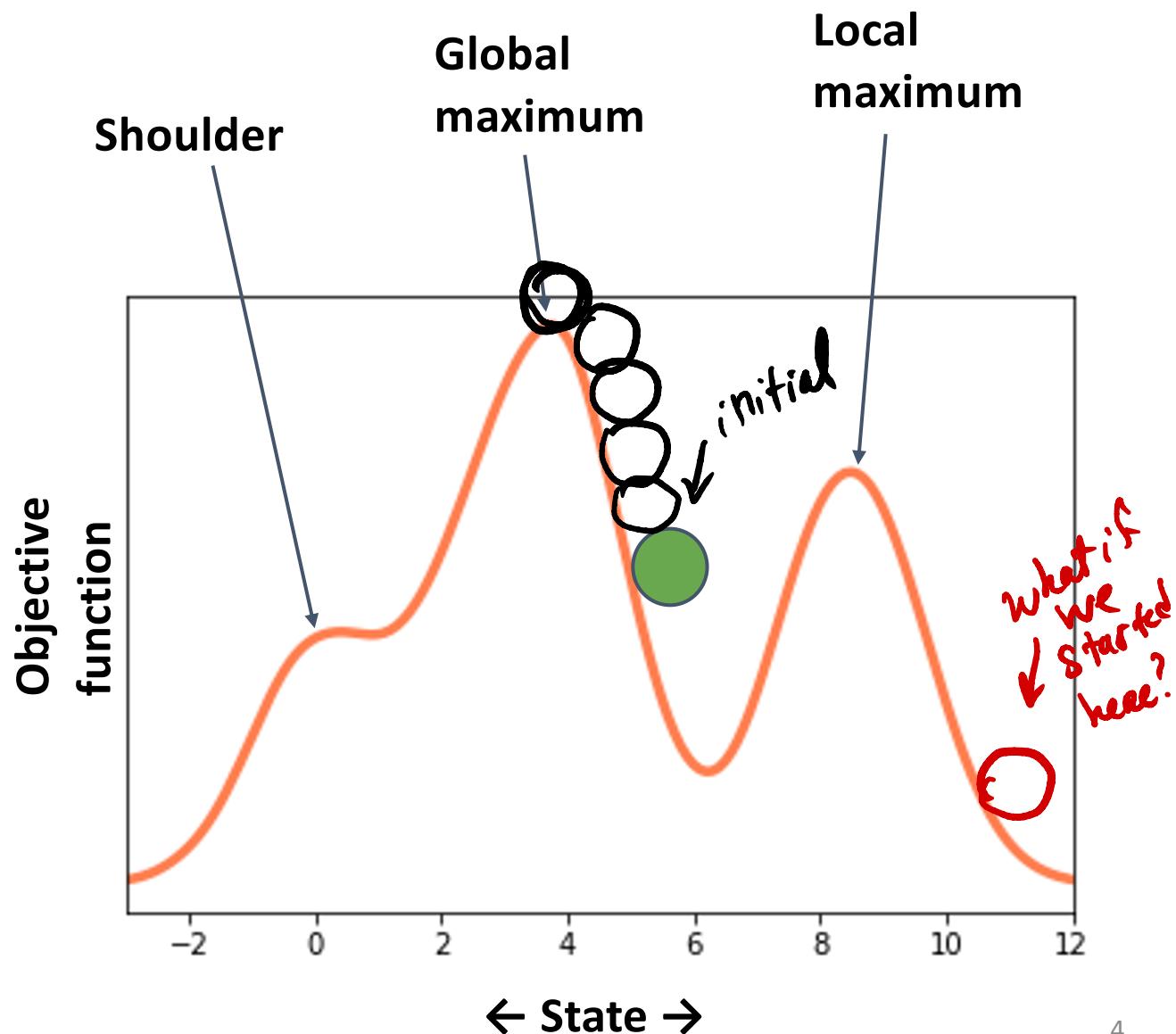
(think: minimizing a loss, or maximizing a utility)



# Local Search

Local search:

- Some terminology...



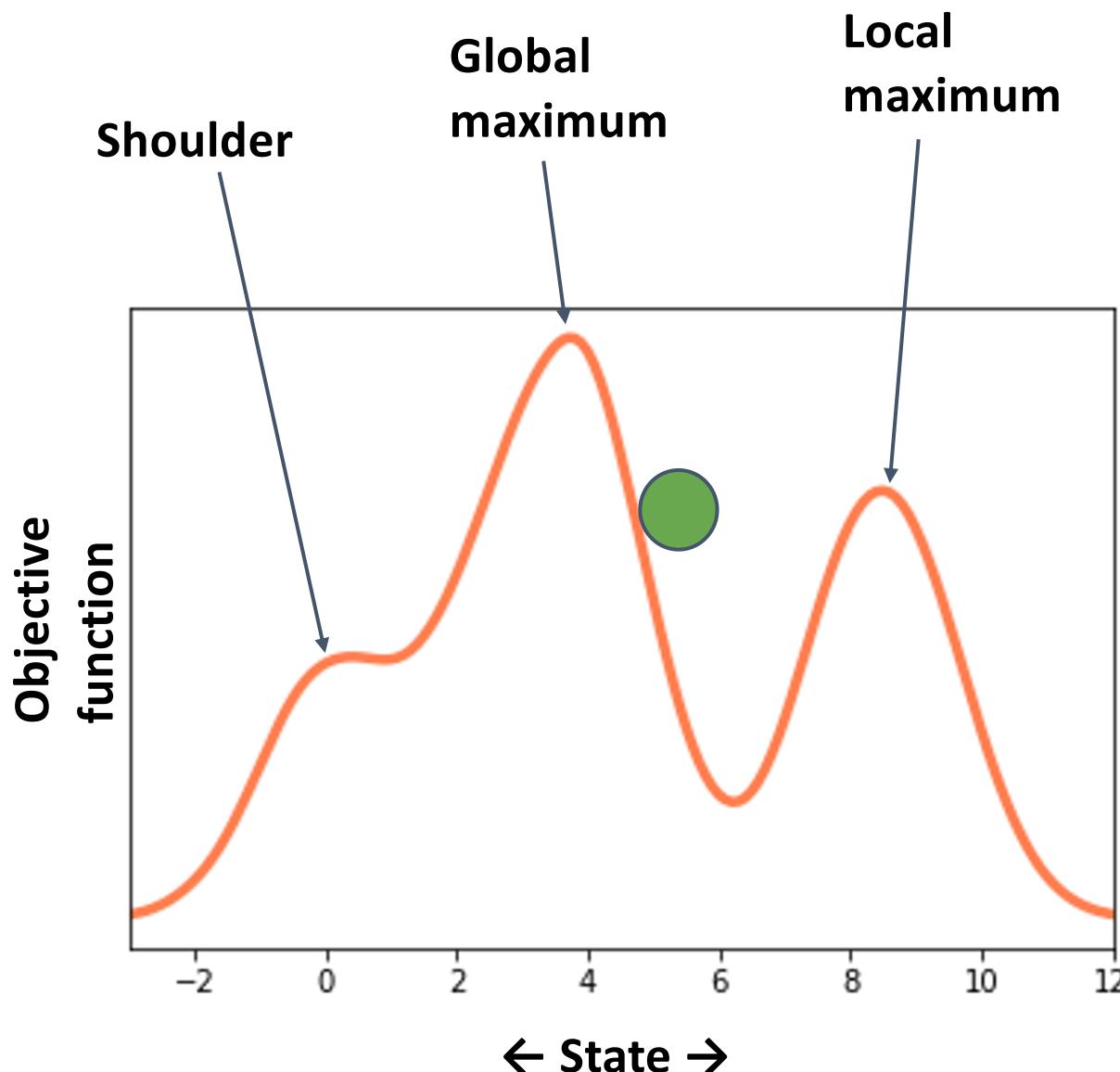
# Hill Climbing

---

AKA: steepest ascent

Some words...

- 1) Start in some initial state
- 2) Get the neighbors of that state
- 3) Move to the neighbor with the best value of the objective function



# Hill Climbing

Some potential issues...

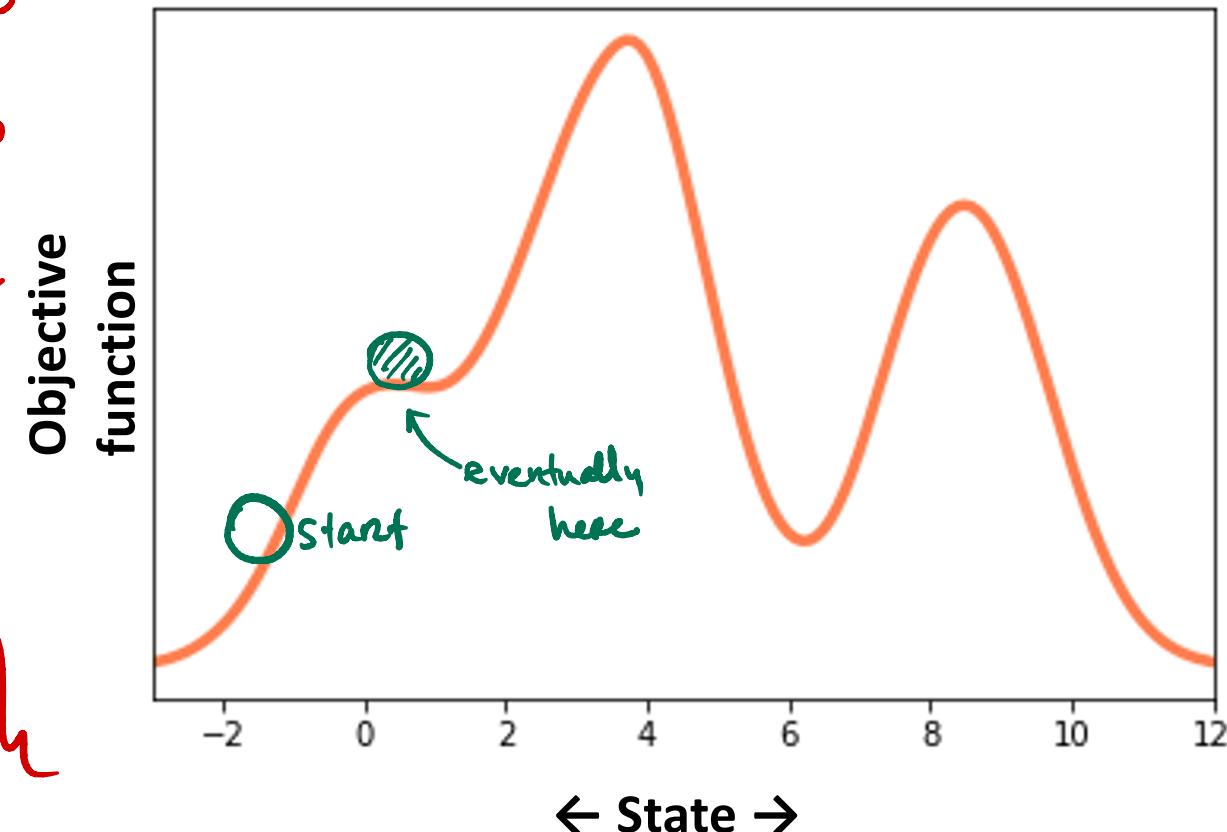
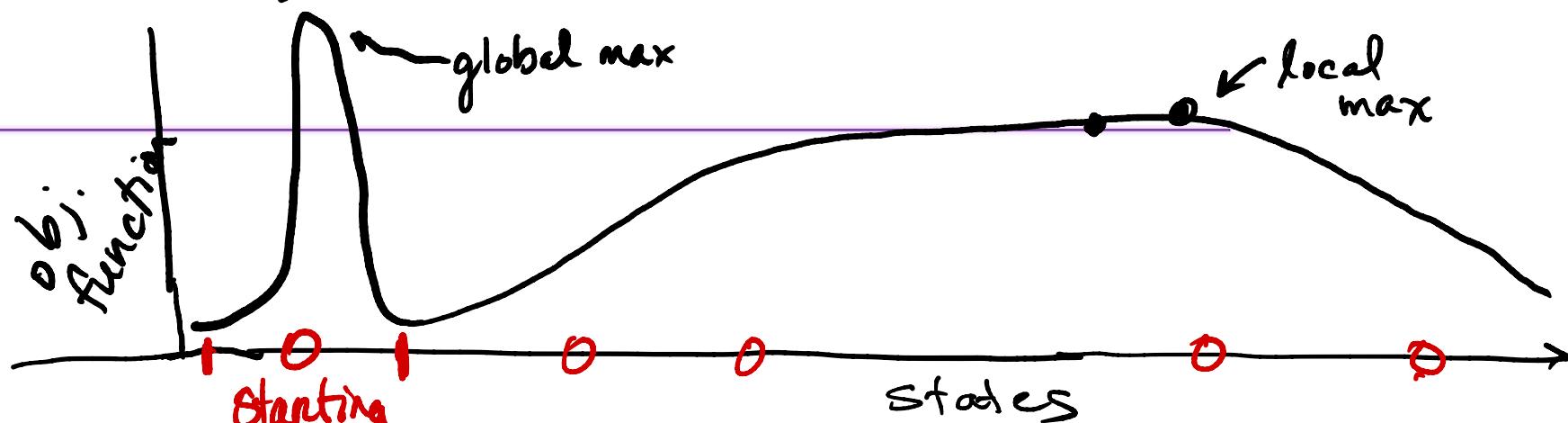
- Hit a shoulder?  
(or local max.)

→ Permit sideways moves?

→ “Convergence” criteria?

→ Stochastic hill climbing

→ Random restart



# Hill Climbing

---

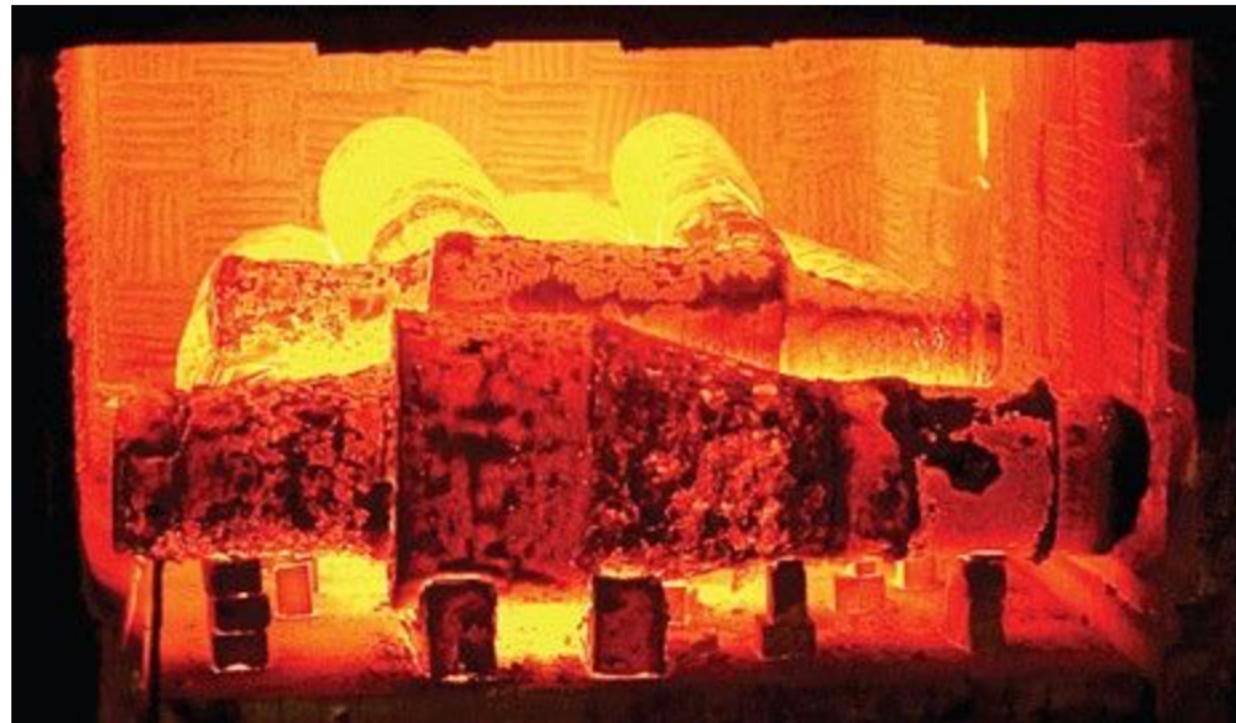
Some pseudocodes...

```
class state:  
  
    # we need to be able to keep track of states and their associated  
    # objective function values  
  
class problem:  
  
    # We want to include everything that defines the problem here:  
    # - initial state  
    # - current state  
    # - what is the objective function we're trying to maximize/minimize?  
    # - what are the choices of action that we have?  
    # - what do we need to know to select our action?  
  
def hill_climb(problem, number of iterations):  
  
    for t in some number of iterations:  
        1. get a list of our available moves  
        2. which move optimizes the objective function?  
        3. do that move; update our state  
        4. possible goal/convergence check
```

# Annealing – Metallurgy

---

Visualization of what happens when you heat treat (e.g. Anneal) a metal: [Video](#)

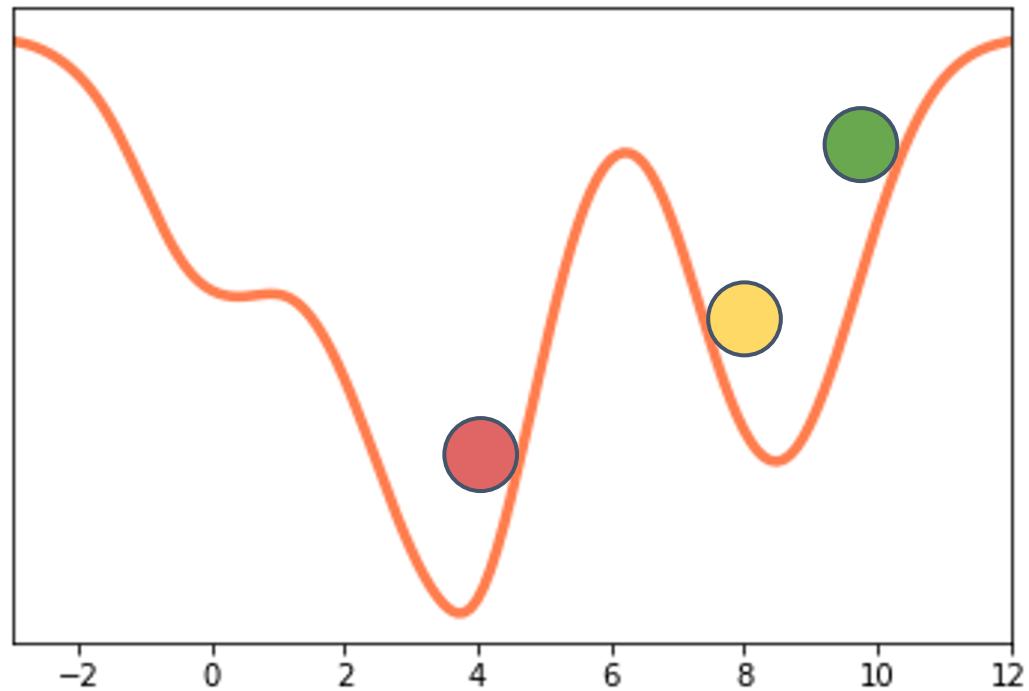


# Simulated Annealing

---

Trying to find global minimum (in this case)

- Propose **random** moves
- Accept with a probability that depends on:
  - 1) "temperature"
  - 2) how "good" the move is
    - Accept move without question if it's better than our current situation
    - Accept move with some probability  $< 1$  if it's worse

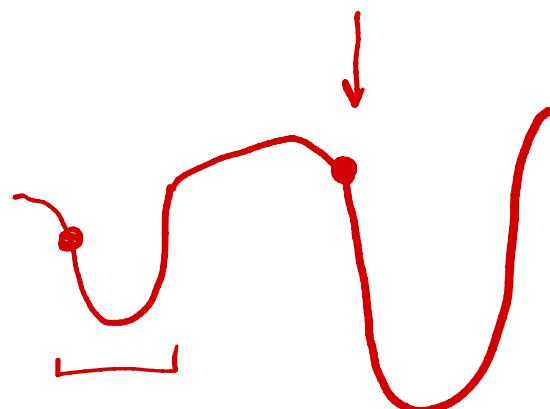


# Simulated Annealing

Visualization of Simulated Annealing: [Video](#)

Meant to mimic the slow cooling of metals.

Each new solution  $x_j$  is accepted with a temperature dependent probability.



$$P_T = \begin{cases} 1 & \text{if } f(x_j) \leq f(x_i) \\ e^{\frac{f(x_i)-f(x_j)}{kT}} & \text{if } f(x_j) \geq f(x_i) \end{cases}$$

*$x_i$  current state*

T - current Temperature  
k - Boltzmann constant

- ❖ [Further reading](#) - optimization is a huge area, we are only scratching the surface

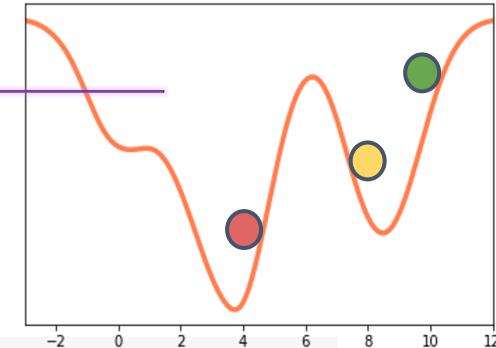
# Simulated Annealing

Some pseudocode...

```
def schedule(time):
    '''some sort of mapping from time to temperature, to represent how we should be
    "cooling off" - that is, accepting wacky solutions with lower and lower probability'''
    # math goes here
    return temperature

def simulated_annealing(problem, some number of iterations):

    for t in some number of iterations:
        #1. update the "temperature", T(t) = schedule(time)
        #2. which moves can we make from the current node?
        #3. pick a random move
        #4. calculate difference in objective function between
        #   proposed new state and the current state (deltaE)
        #5. if proposed new state is better than the current state,
        #   then accept the proposed move with probability 1
        #6. otherwise...
        #       ACCEPT the move with probability exp(-deltaE/T(t)),
        #       or REJECT with prob 1-exp(-deltaE/T(t))
```



# Next Time

---

- *Optimization!*
- *Genetic algorithms!*

But first...

