Constructieve Algoritmen

Bas Terwijn <b.terwijn@uva.nl>

Algoritmen en Heuristieken

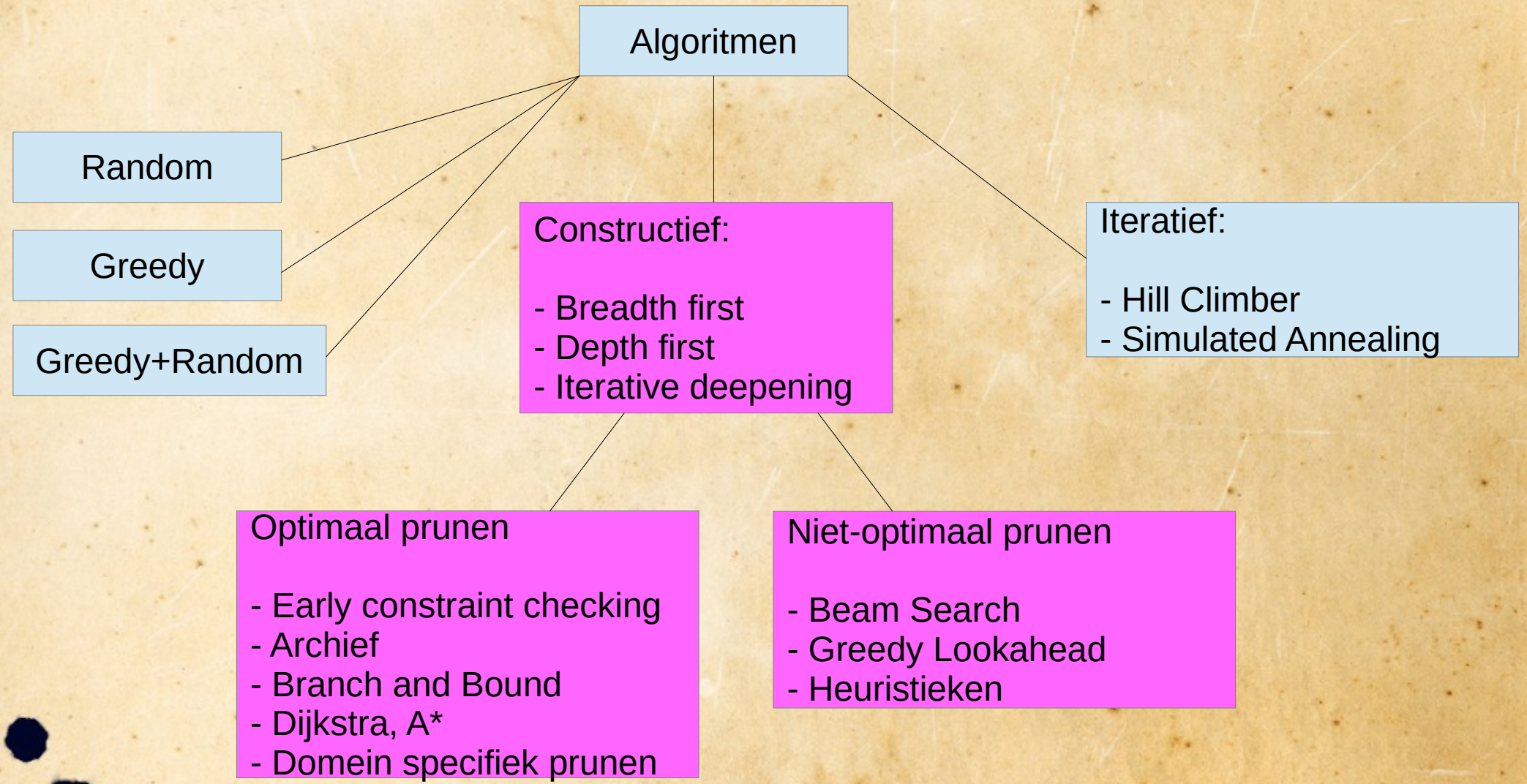
Minor Programmeren

Universiteit van Amsterdam

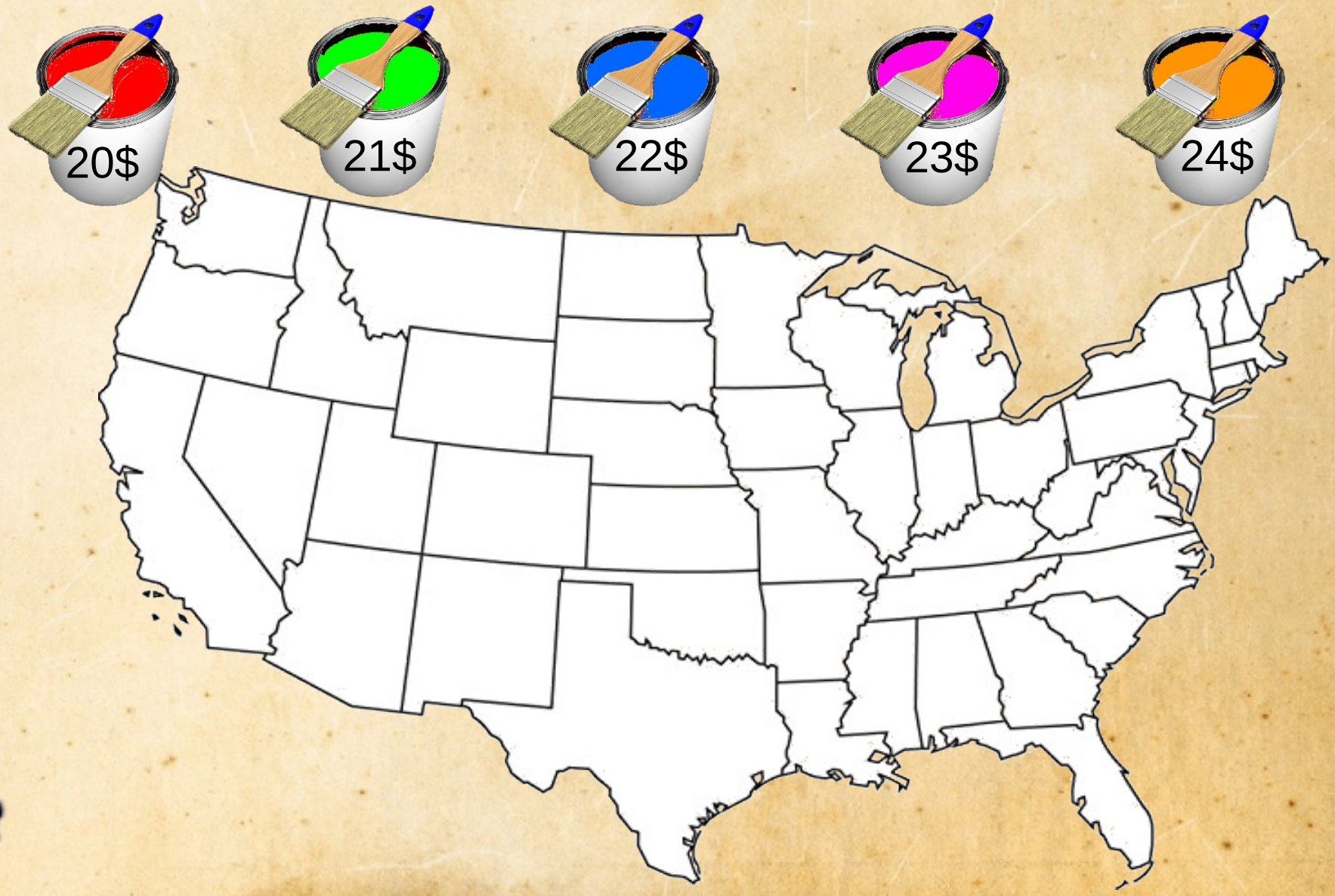
Wat ging vooraf

- Kick Off: Wouter Vrielink
- State Space, Representatie: Wouter Vrielink
- Live Coding: Wouter Vrielink, Quinten van der Post

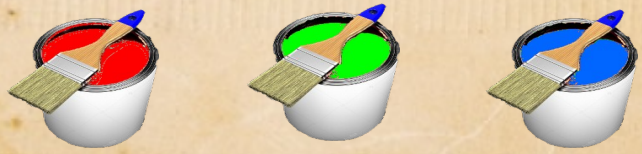
Algoritmen

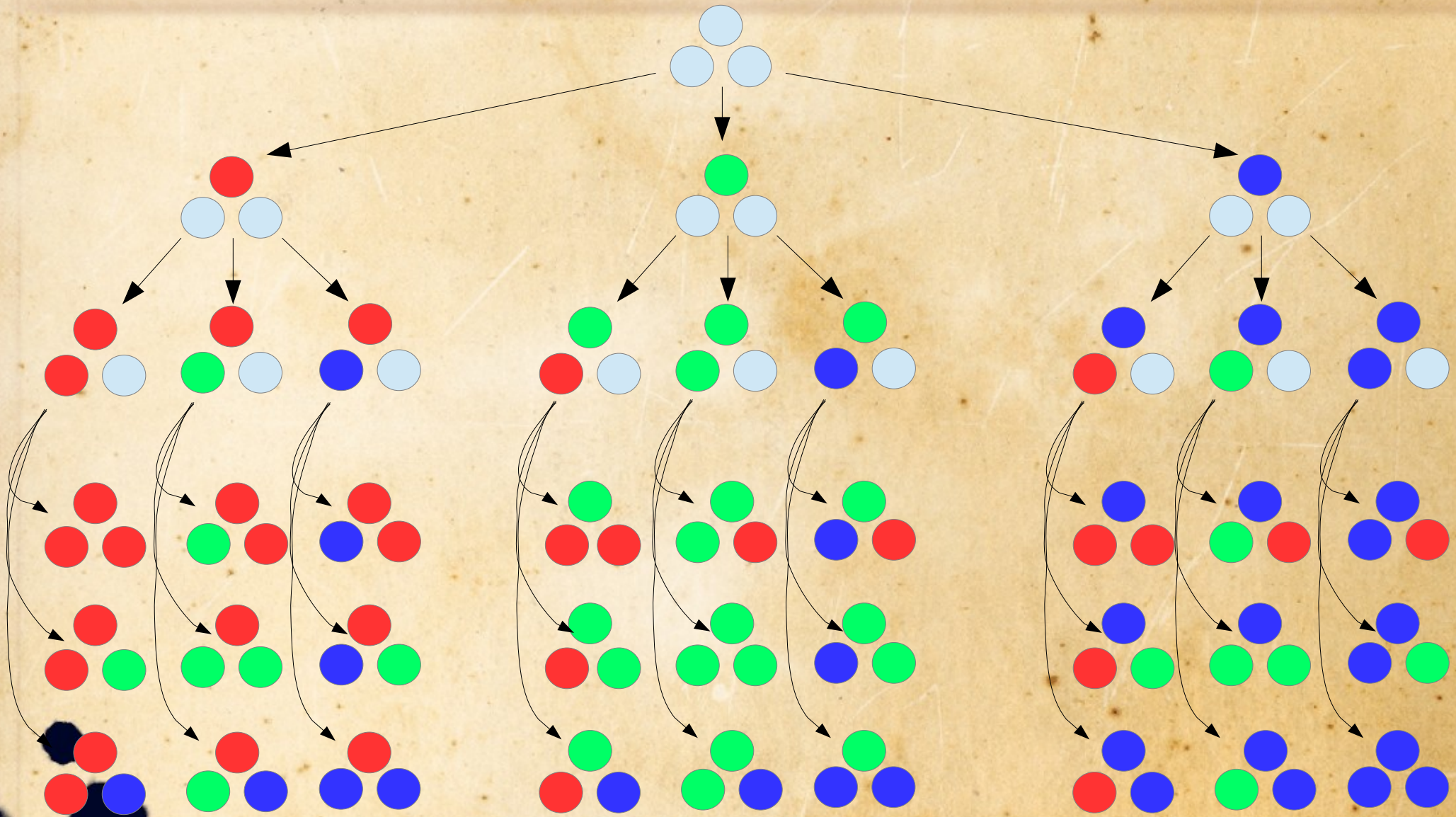

Algoritmen
Random
Greedy
Greedy+Random
Constructief:
- Breadth first
- Depth first
- Iterative deepening
Iteratief:
- Hill Climber
- Simulated Annealing
Optimaal prunen
- Early constraint checking
- Archief
- Branch and Bound
- Dijkstra, A*
- Domein specifiek prunen
Niet-optimaal prunen
- Beam Search
- Greedy Lookahead
- Heuristieken

Kaart kleuren



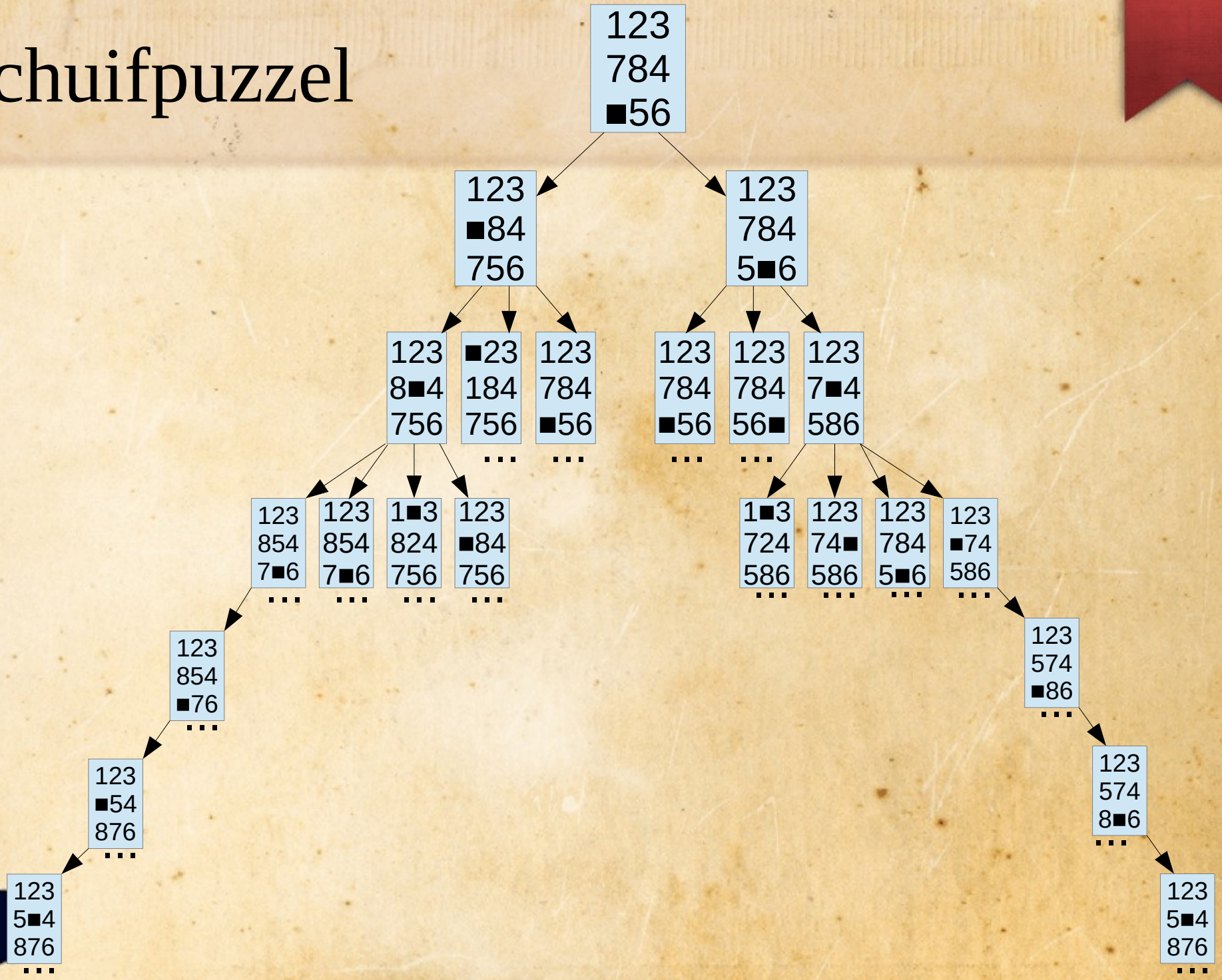
Kaart kleuren



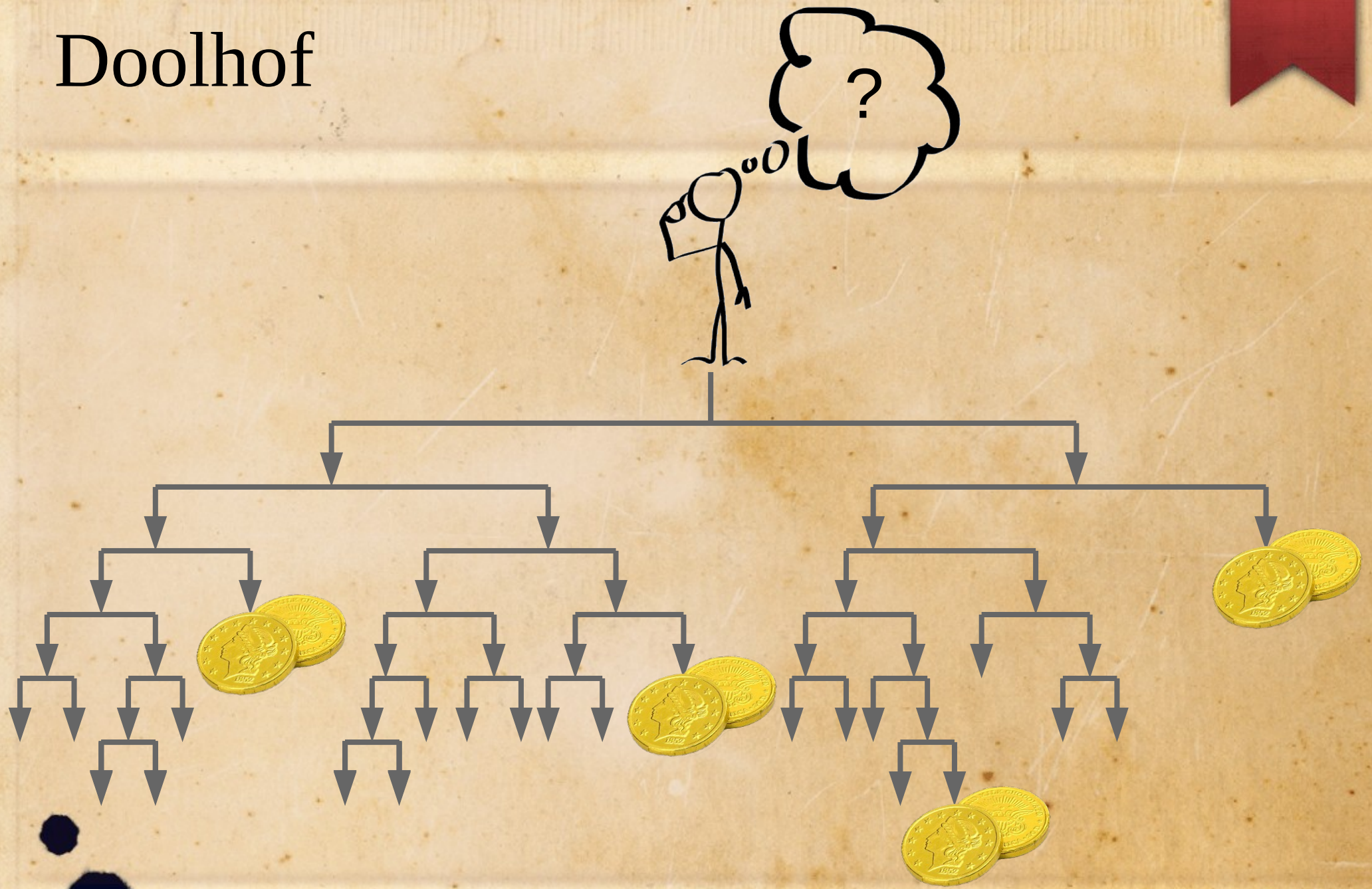


Schuifpuzzel

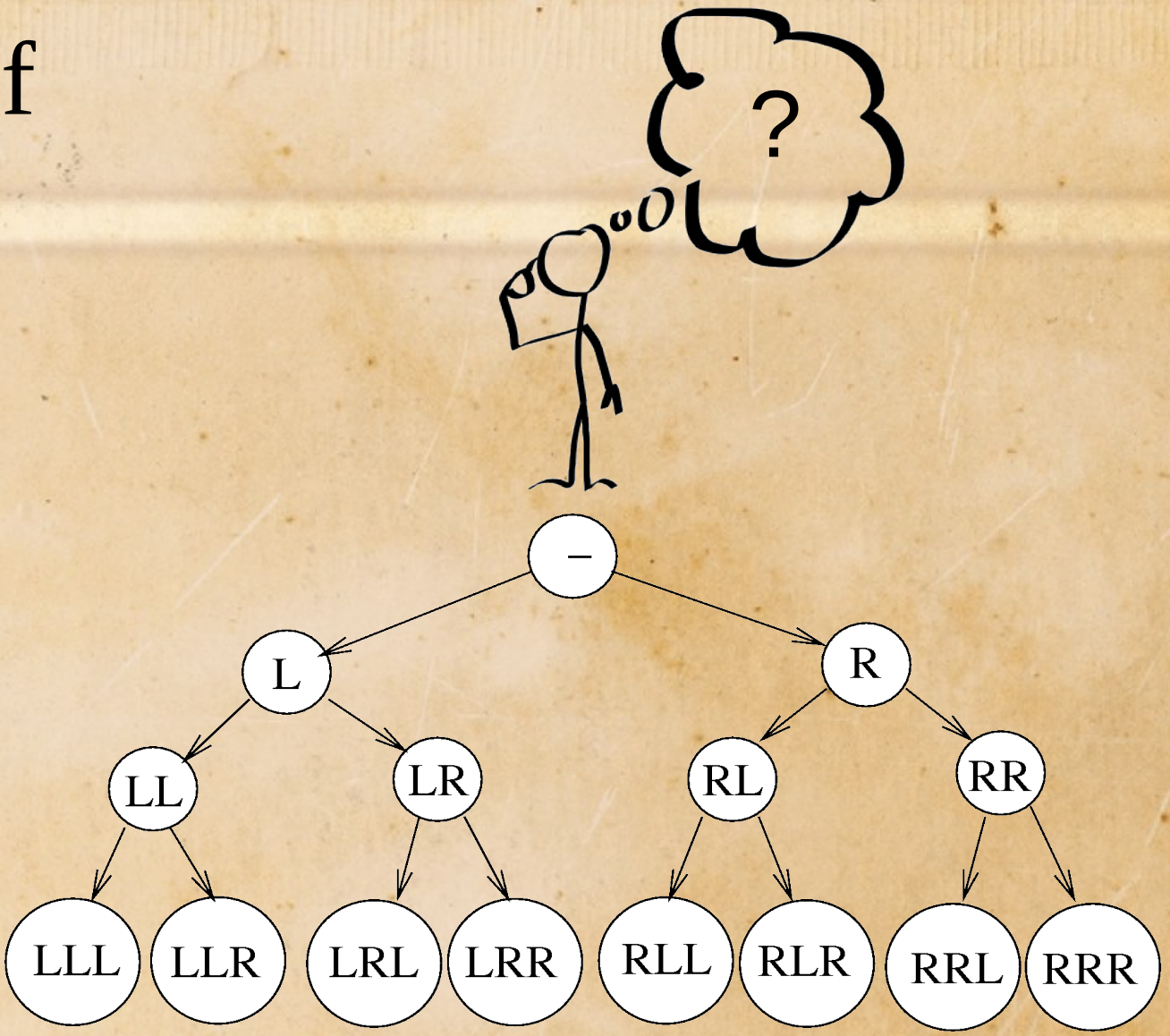
Schuifpuzzel



Doolhof



Doolhof



Constructieve Algoritmen

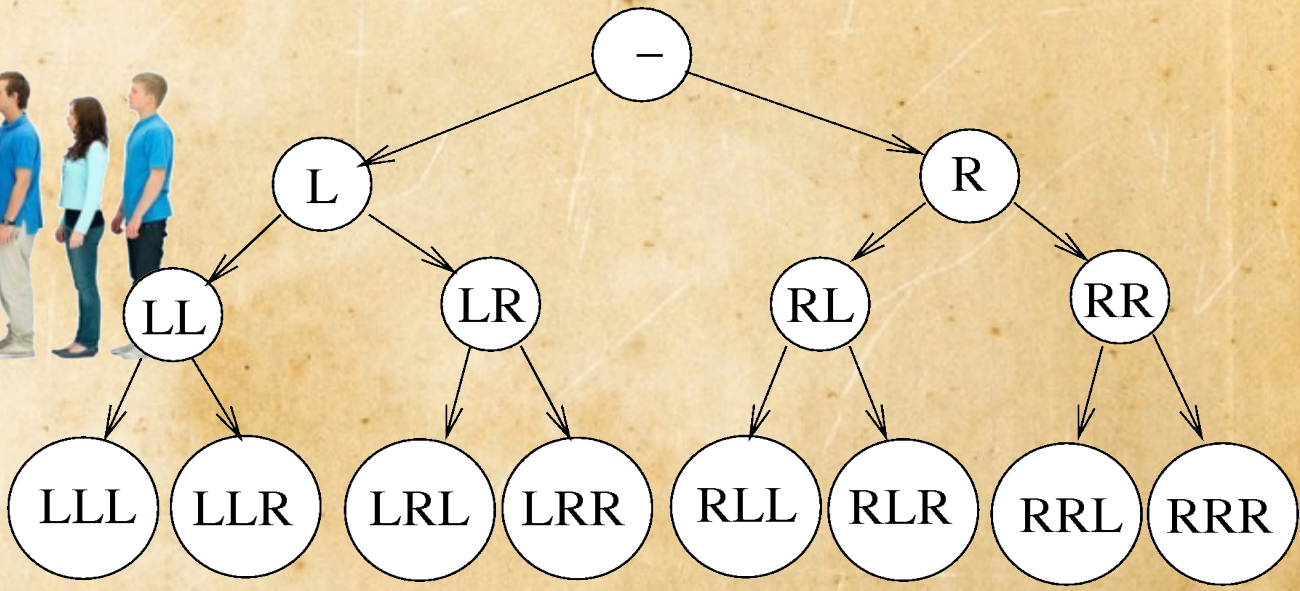
Breadth first

QUEUE (FIFO)

tijd

OUT

IN



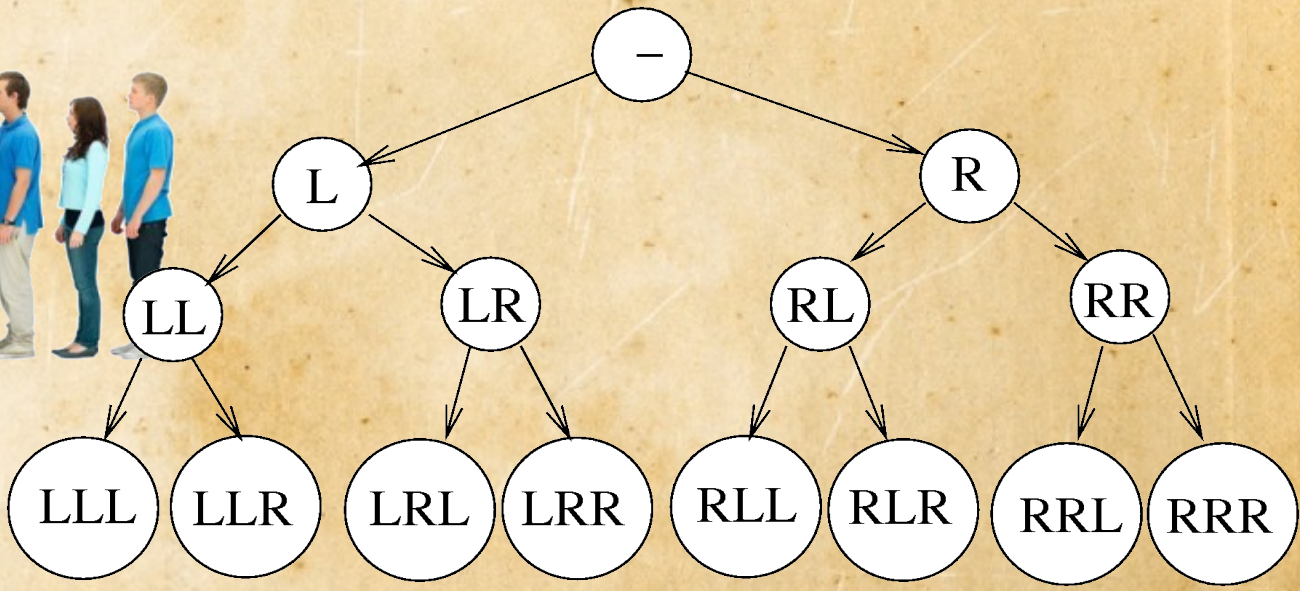
Breadth first

QUEUE (FIFO)

tijd

OUT

IN



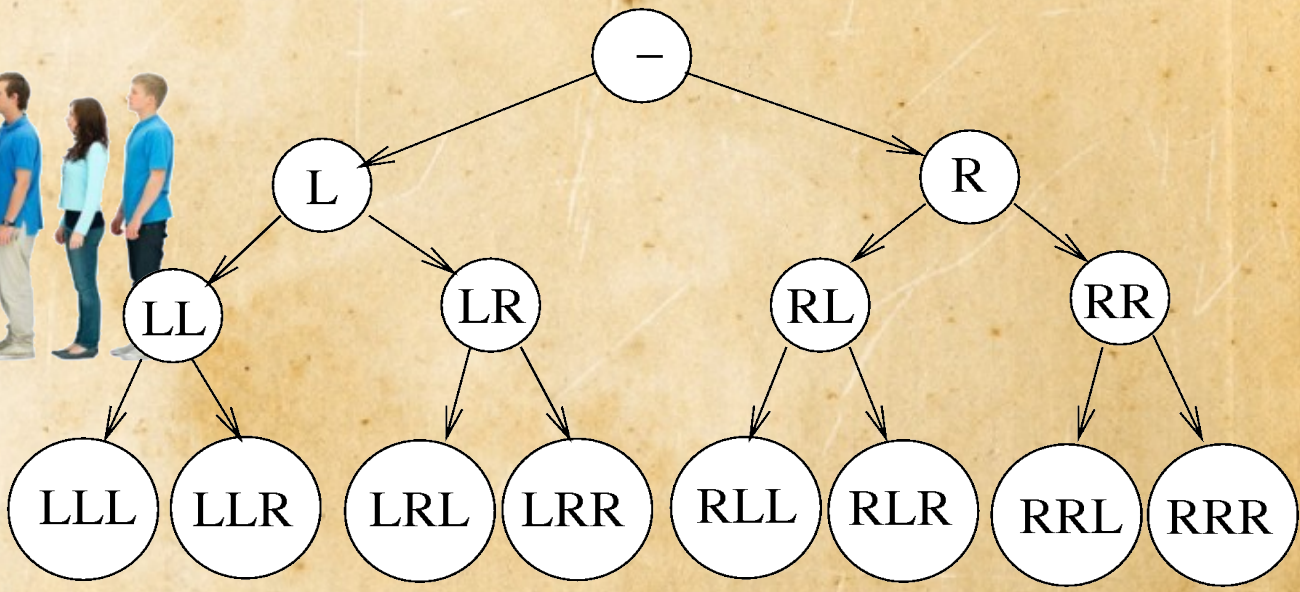
Breadth first

QUEUE (FIFO)

tijd

OUT

IN



Breadth first

QUEUE (FIFO)

tijd

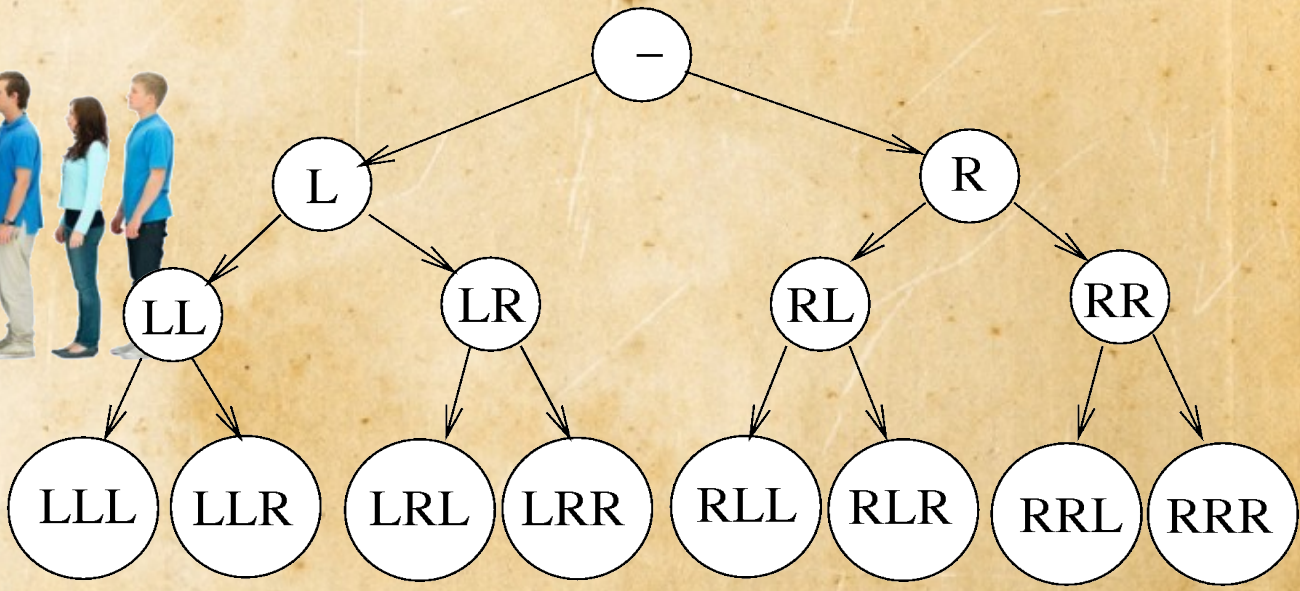
-

-

L, R

OUT

IN



Breadth first

QUEUE (FIFO)

tijd

-

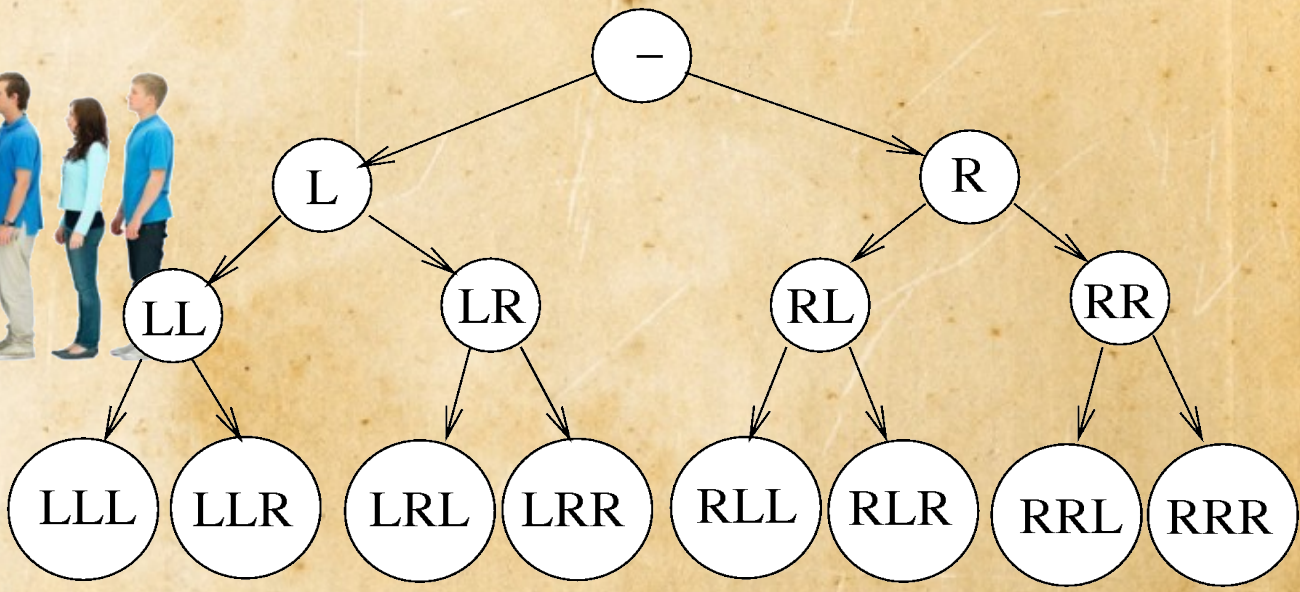
L, R

L

R

OUT

IN



Breadth first

QUEUE (FIFO)

tijd

-

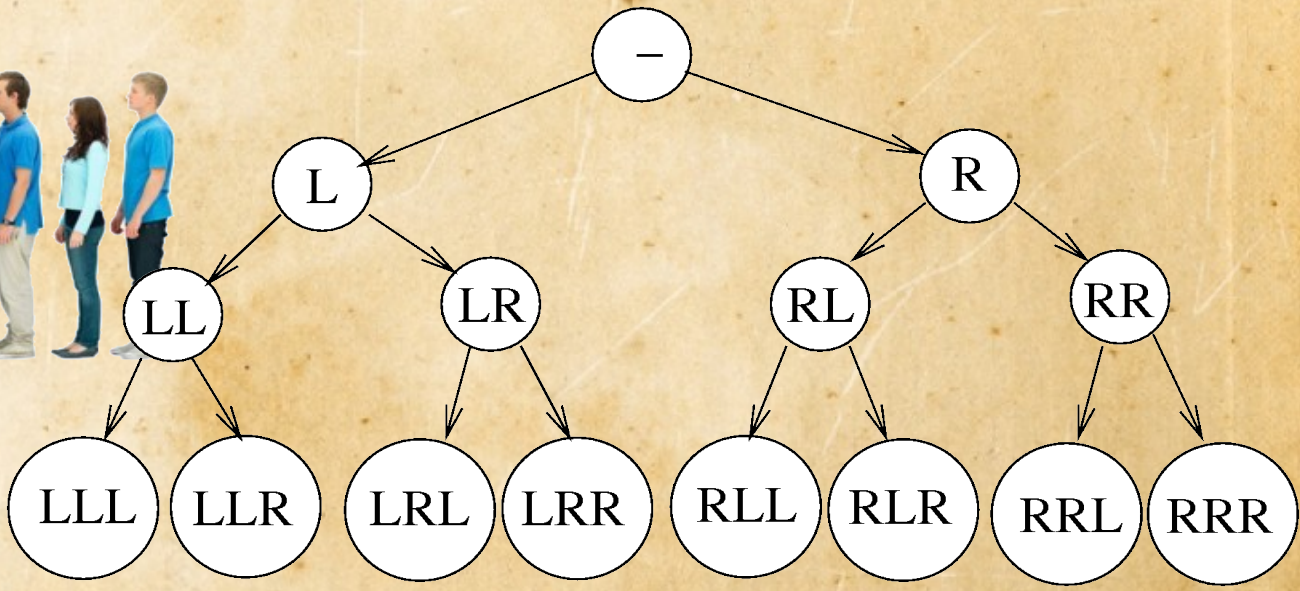
L, R

R, LL

L

OUT

IN



Breadth first

QUEUE (FIFO)

tijd

-

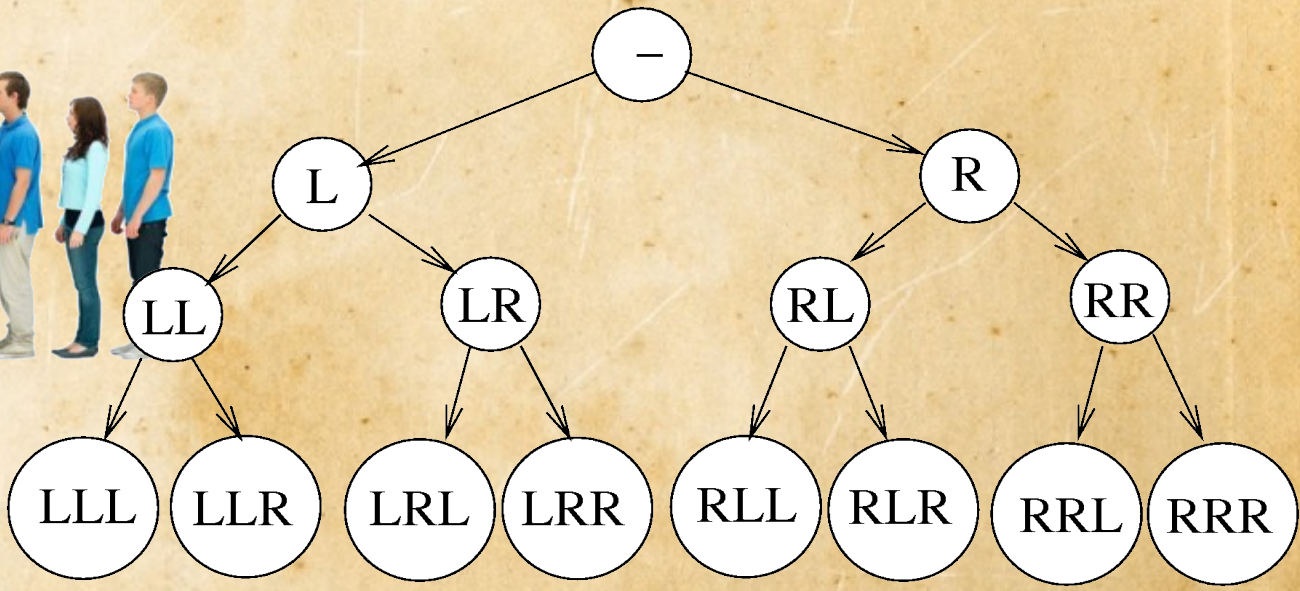
L, R

L

R, LL, LR

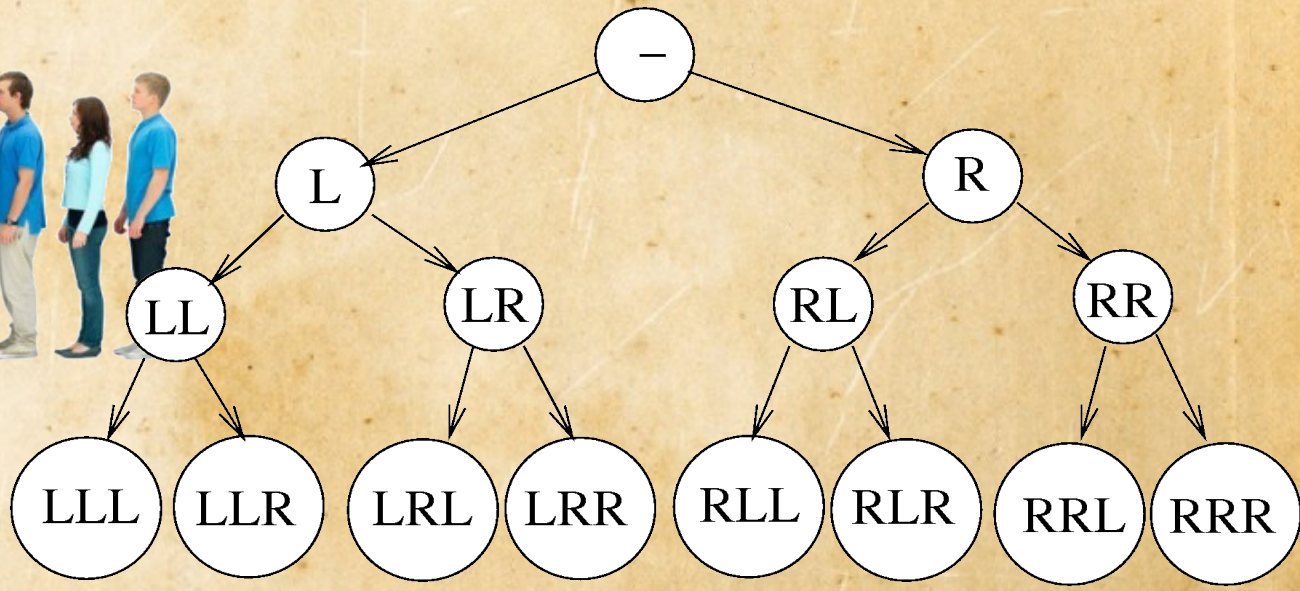
OUT

IN



Breadth first

QUEUE (FIFO)



tijd

-

L, R

R, LL, LR

LL, LR

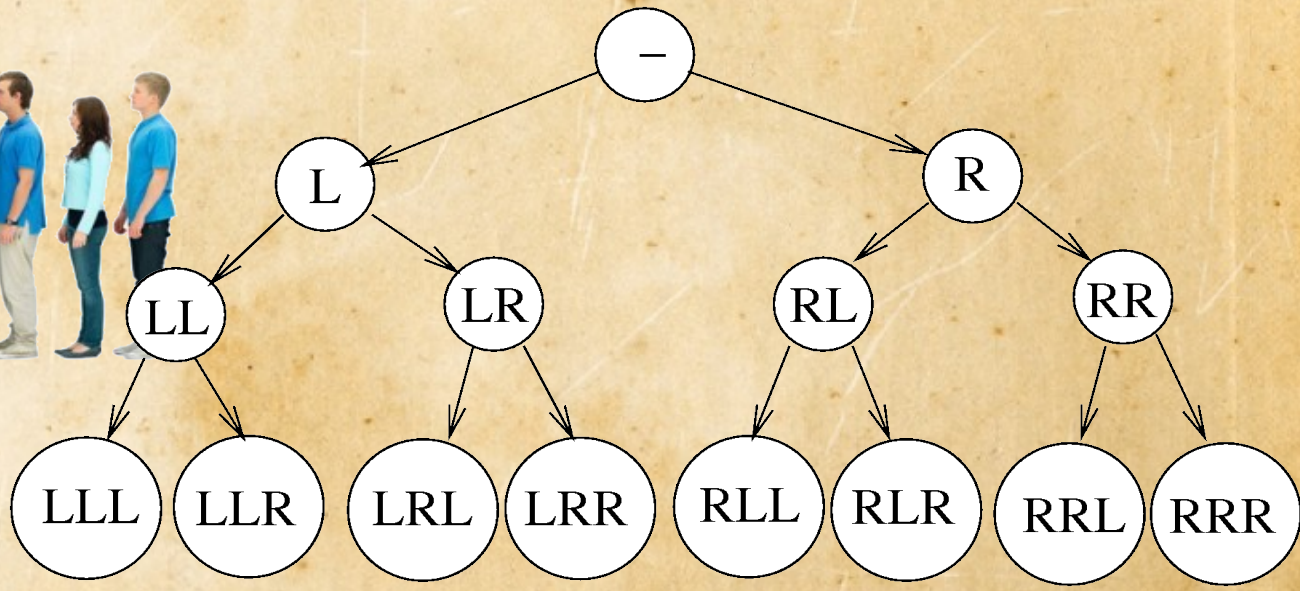
R

OUT

IN

Breadth first

QUEUE (FIFO)



tijd

-

L, R

R, LL, LR

LL, LR, RL

R

OUT

IN

Breadth first

QUEUE (FIFO)

tijd

-

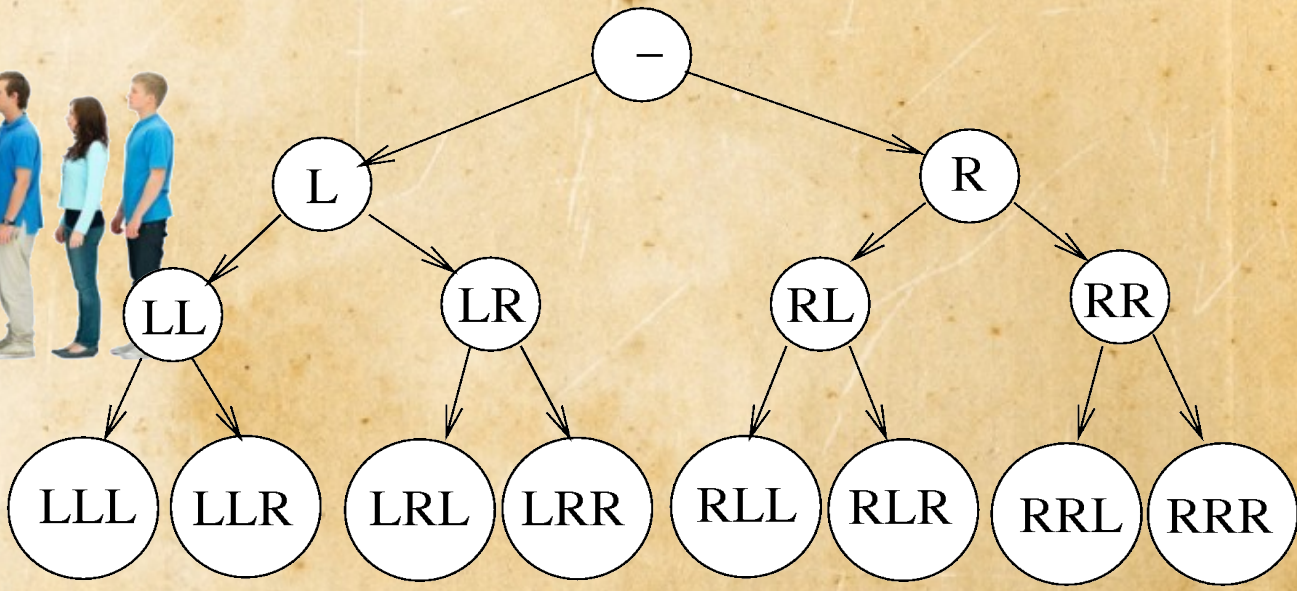
L, R

R, LL, LR

LL, LR, RL, RR

R

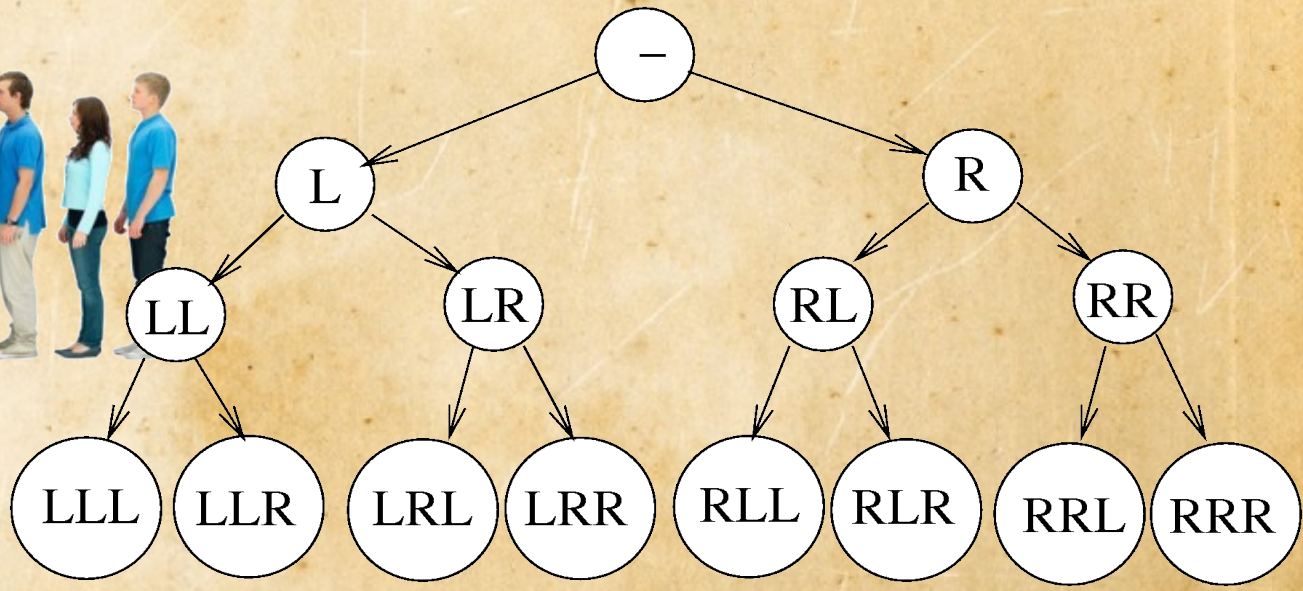
OUT



IN

Breadth first

QUEUE (FIFO)



tijd

-
L, R
R, LL, LR
LL, LR, RL, RR
LR, RL, RR, LLL, LLR

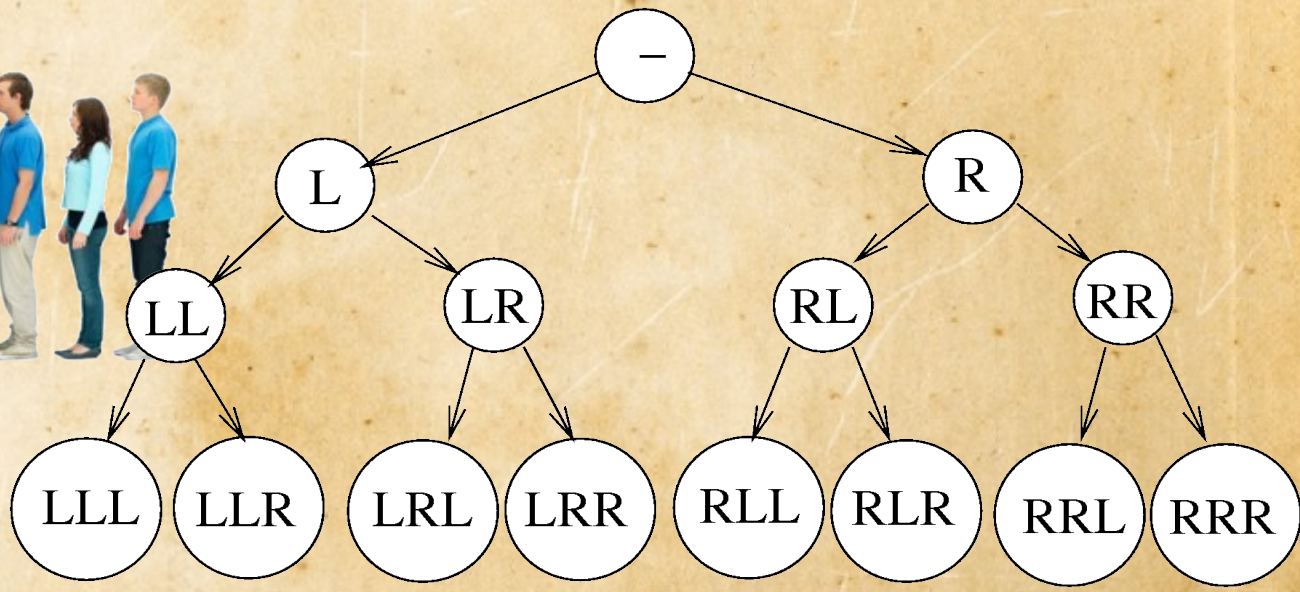
LL

OUT

IN

Breadth first

QUEUE (FIFO)



tijd

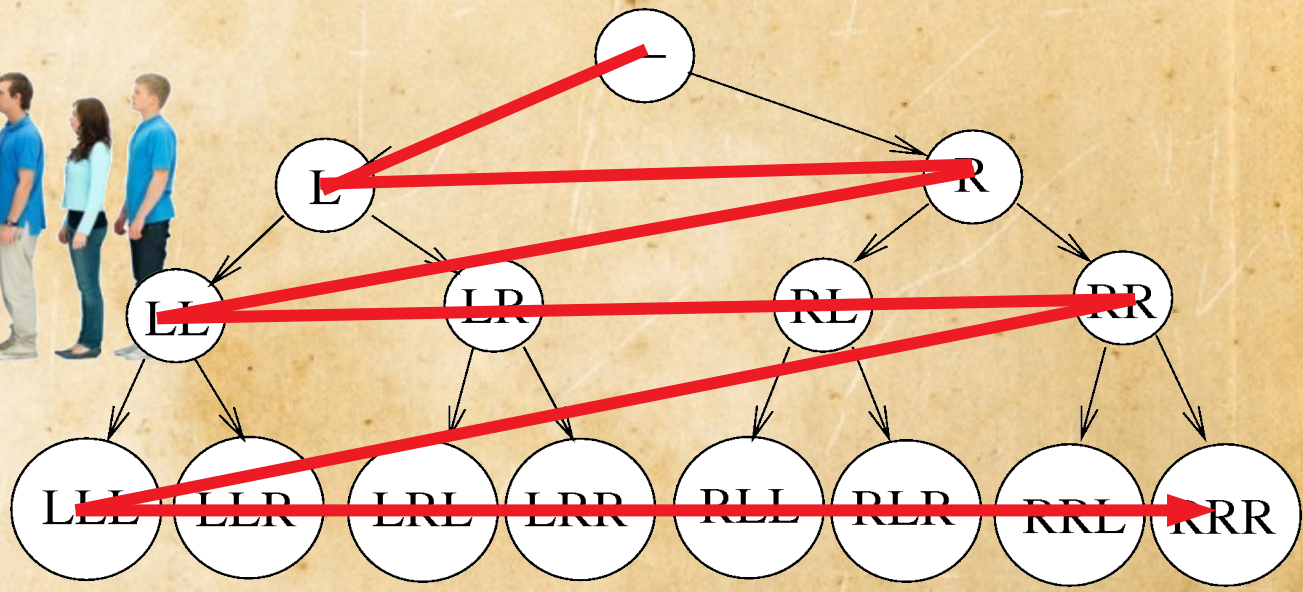
OUT

IN

-
L, R
R, LL, LR
LL, LR, RL, RR
LR, RL, RR, LLL, LLR
RL, RR, LLL, LLR, LRL, LRR
RR, LLL, LLR, LRL, LRR, RLL, RLR
LLL, LLR, LRL, LRR, RLL, RLR, RRL, RRR
LLR, LRL, LRR, RLL, RLR, RRL, RRR
LRL, LRR, RLL, RLR, RRL, RRR
LRR, RLL, RLR, RRL, RRR
RLL, RLR, RRL, RRR
RLR, RRL, RRR
RRL, RRR
RRR

Breadth first

QUEUE (FIFO)



tijd

OUT

IN

- L, R
- R, LL, LR
- LL, LR, RL, RR
- LR, RL, RR, LLL, LLR
- RL, RR, LLL, LLR, LRL, LRR
- RR, LLL, LLR, LRL, LRR, RLL, RLR
- LLL, LLR, LRL, LRR, RLL, RLR, RRL, RRR
- LLR, LRL, LRR, RLL, RLR, RRL, RRR
- LRL, LRR, RLL, RLR, RRL, RRR
- LRR, RLL, RLR, RRL, RRR
- RLL, RLR, RRL, RRR
- RLR, RRL, RRR
- RRL, RRR
- RRR

Breadth first in Python

```
import queue
import copy

depth = 3                                   # no deeper than 'depth'
queue = queue.Queue()
queue.put("")                               # add begin state to queue
while not queue.empty():
    state = queue.get()                     # get first from queue
    print(state)
    if len(state) < depth:                  # stop condition
        for i in ['L','R']:                     # for each possible action:
            child = copy.deepcopy(state)        # deepcopy the state
            child += i                          # make new child
            queue.put(child)                    # add new child
```

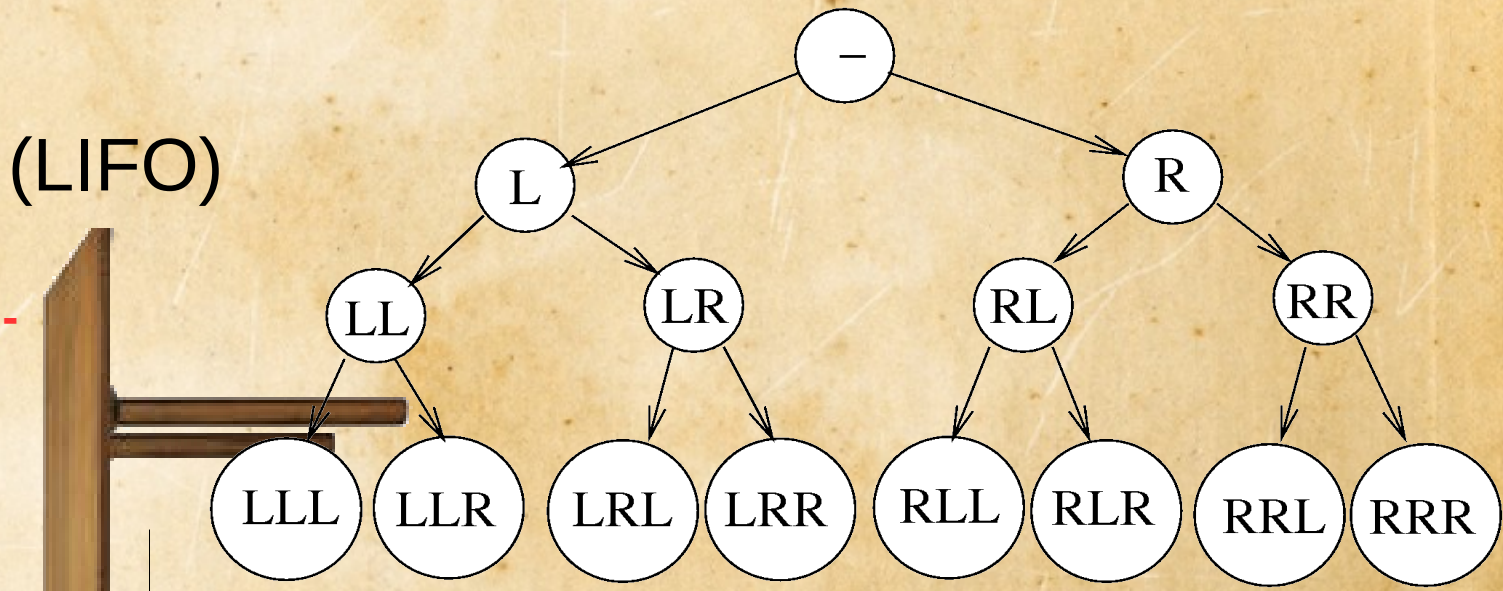
Breadth first in Python

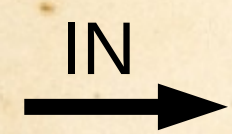
```
import queue
import copy

depth = 3                                   # no deeper than
queue = queue.Queue()
queue.put("")                               # add begin state
while not queue.empty():
    state = queue.get()                     # get first from
    print(state)
    if len(state) < depth:                  # stop condition
        for i in ['L','R']:                     # for each pc
            child = copy.deepcopy(state)        # deepcopy
            child += i                          # make new
            queue.put(child)                    # add new
```

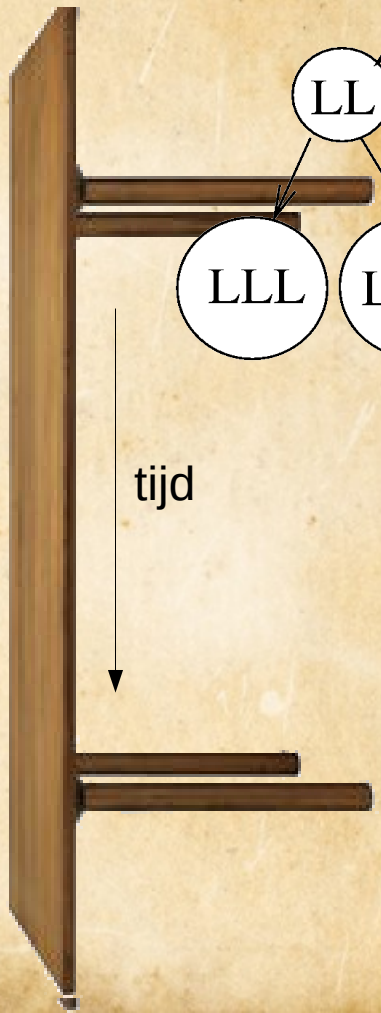
```
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR
```

Depth first

STACK (LIFO)

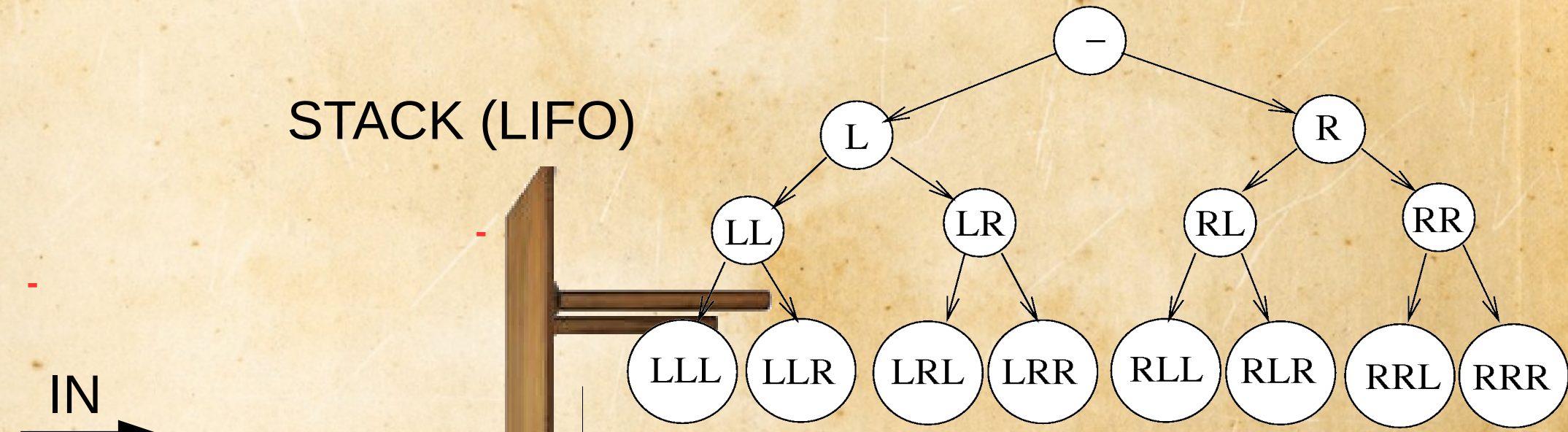


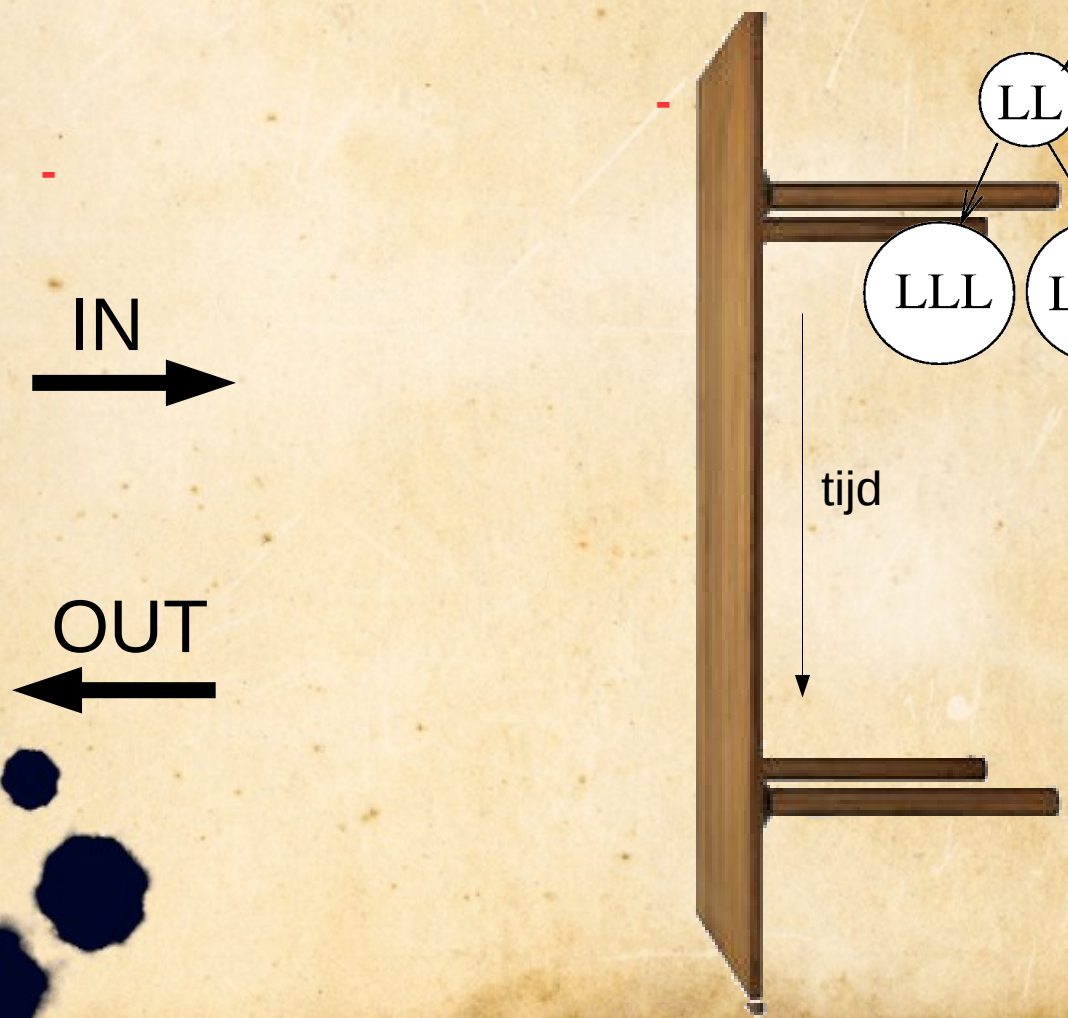




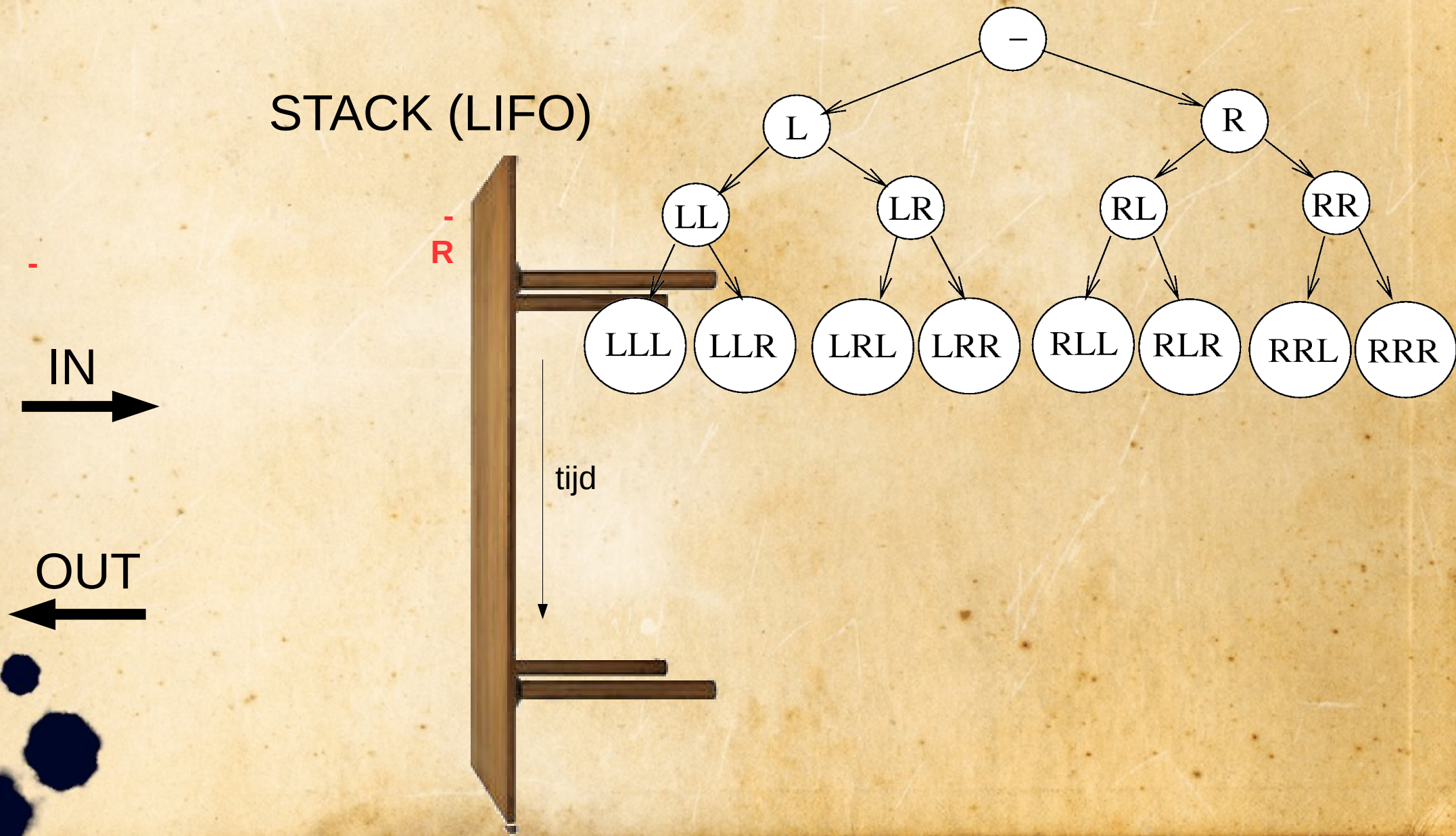
Depth first

STACK (LIFO)

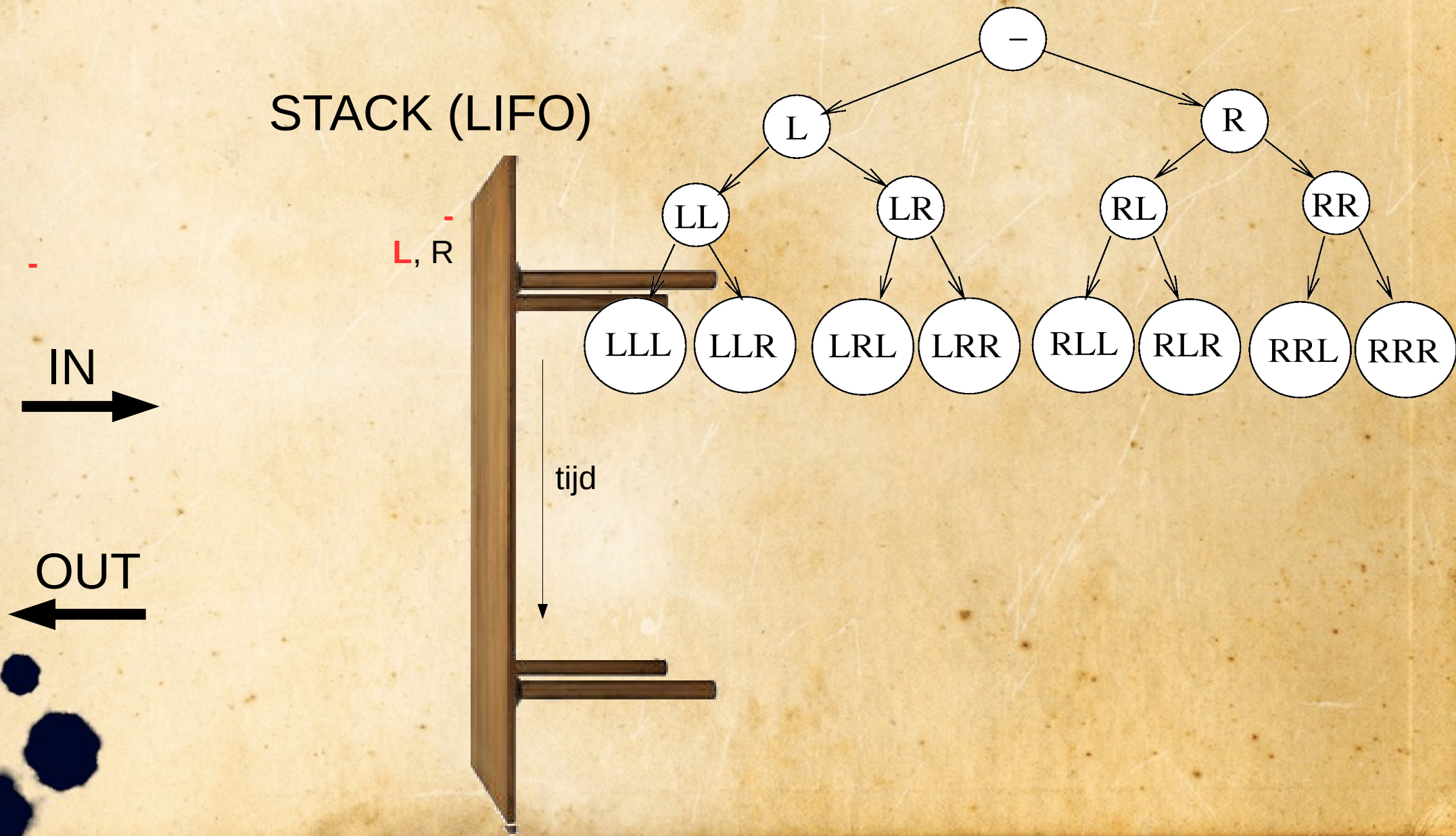




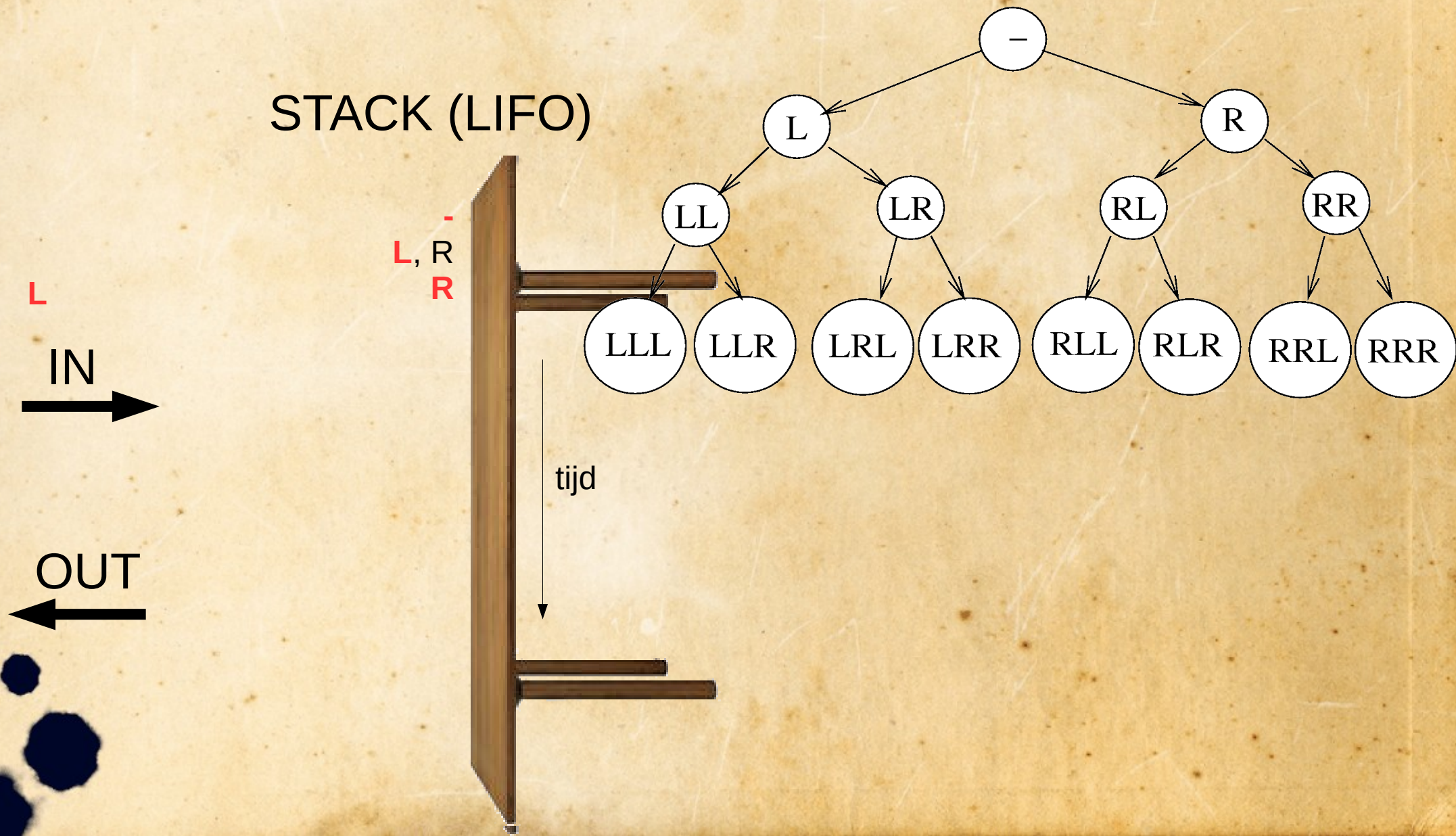
Depth first


STACK (LIFO)
-
R
-
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

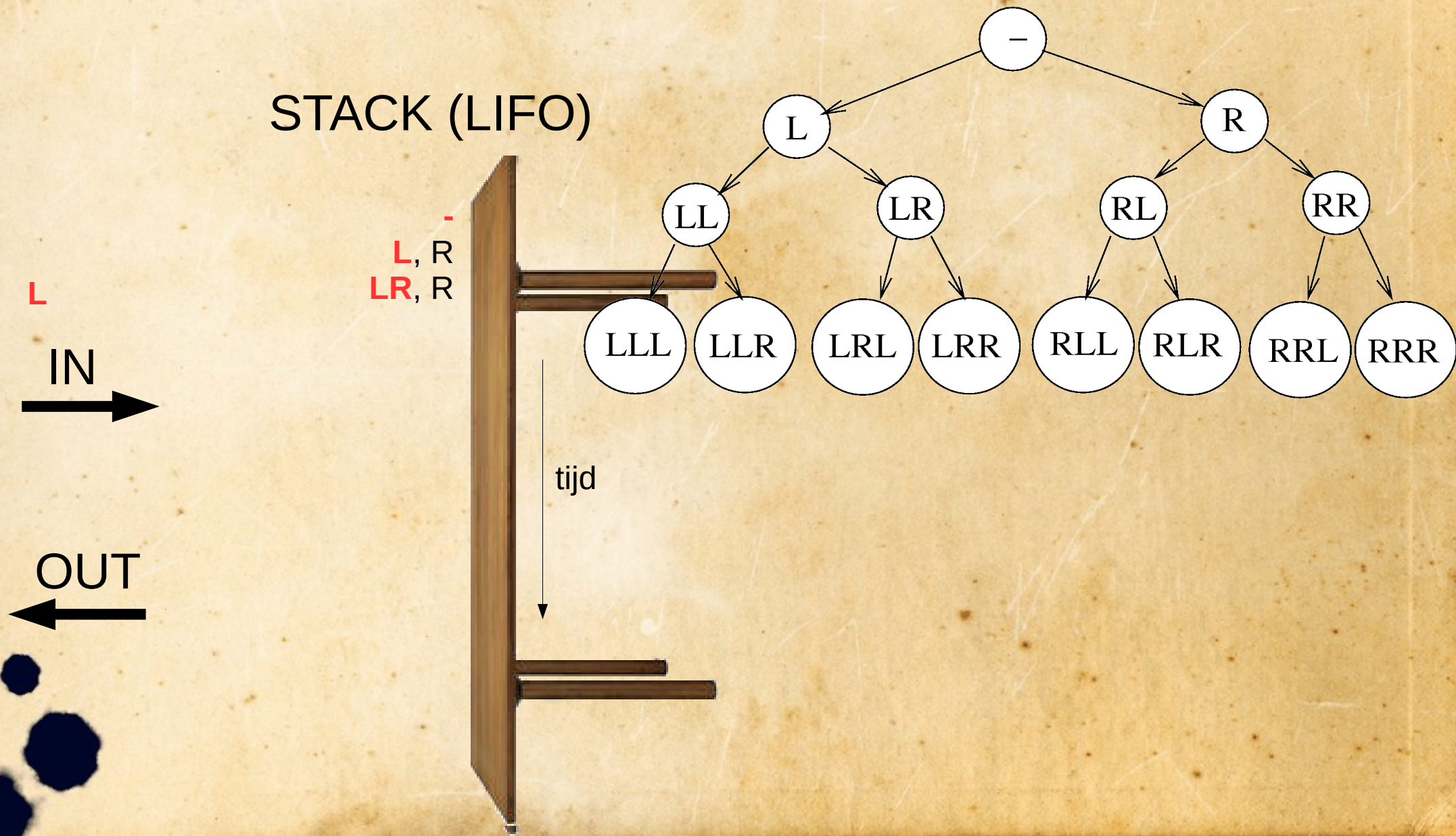
Depth first


STACK (LIFO)
-
L, R
-
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

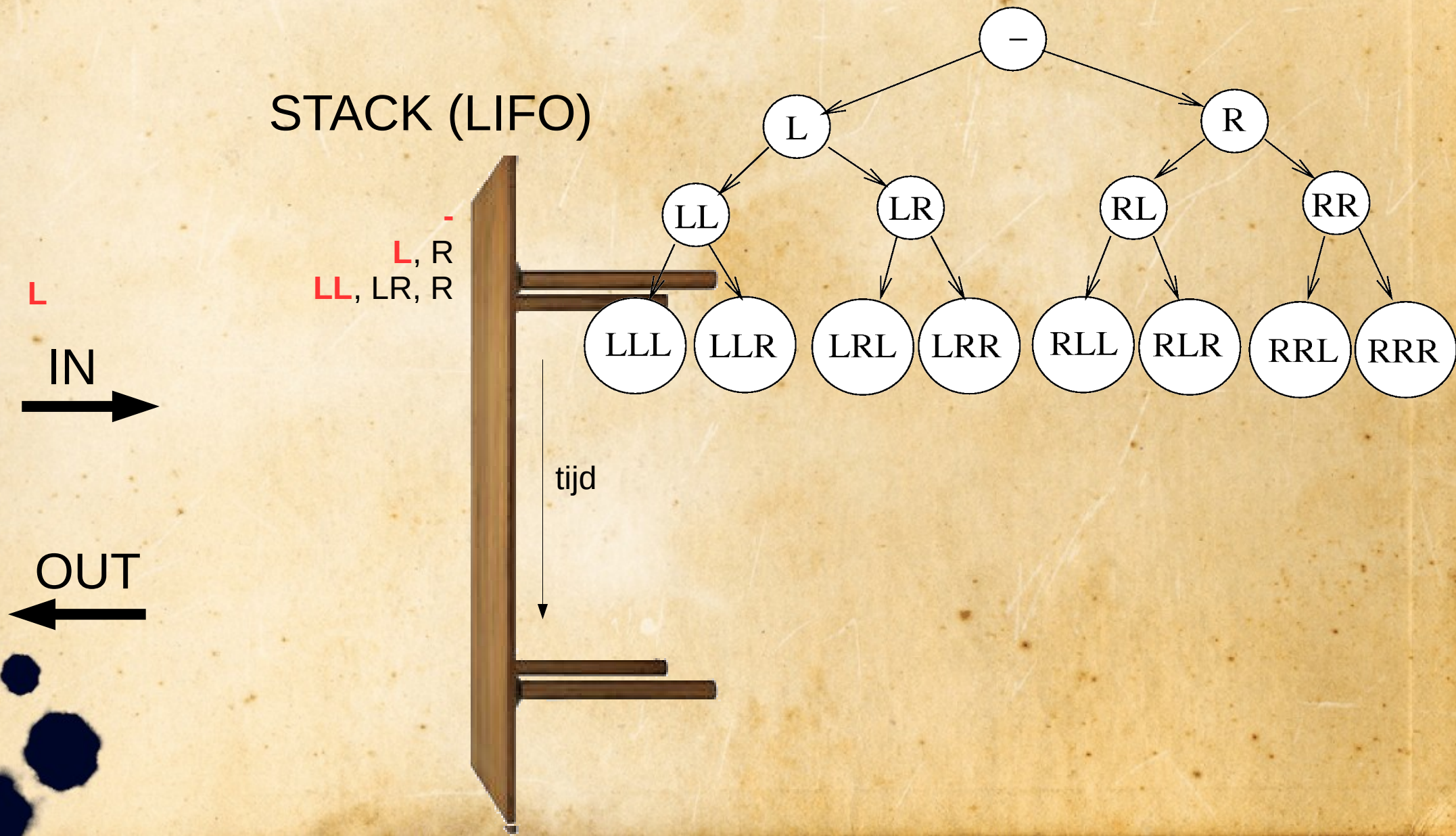
Depth first


STACK (LIFO)
-
L, R
R
L
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

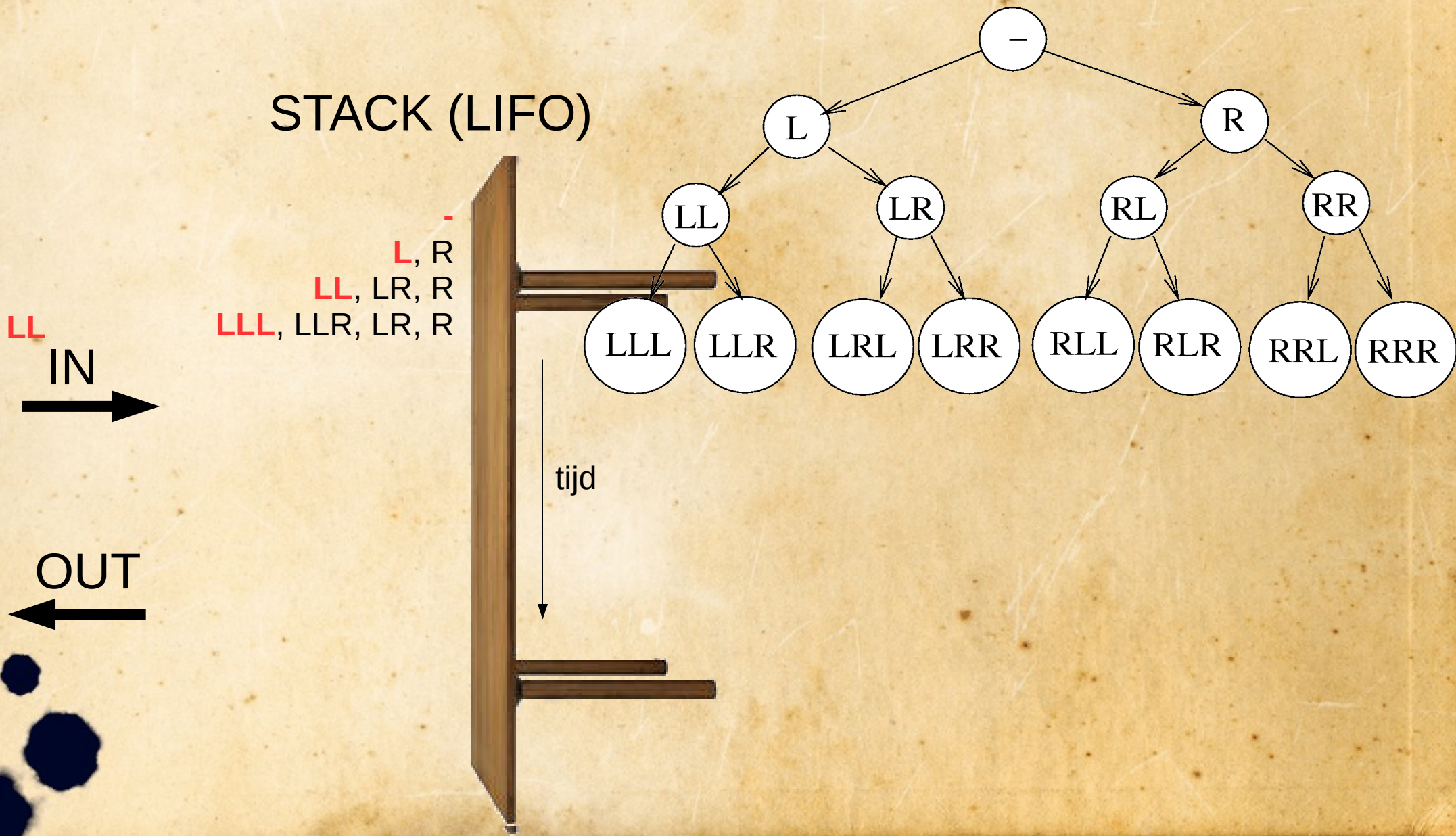
Depth first


STACK (LIFO)
-
L, R
LR, R
L
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

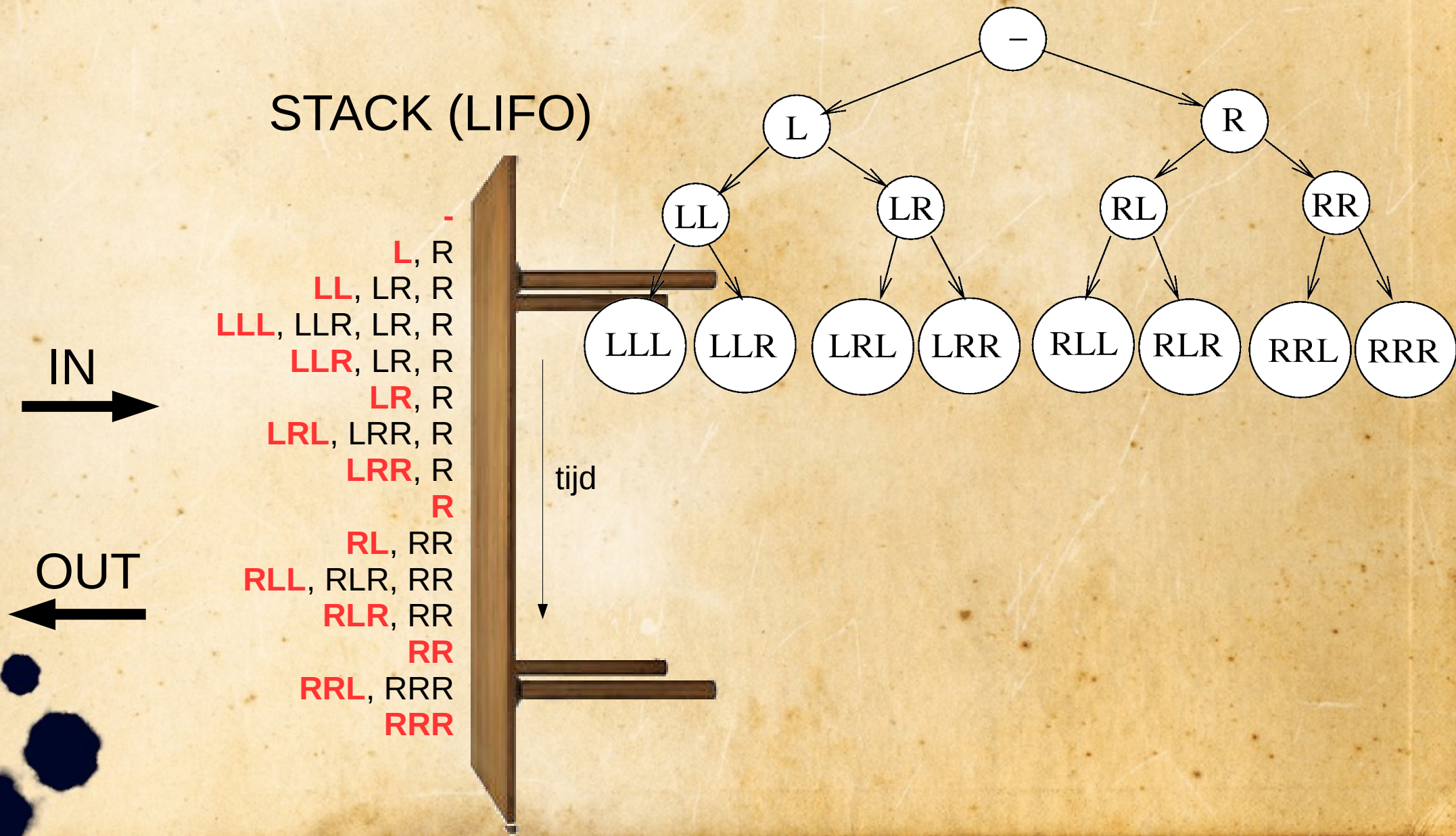
Depth first


STACK (LIFO)
-
L, R
LL, LR, R
L
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

Depth first

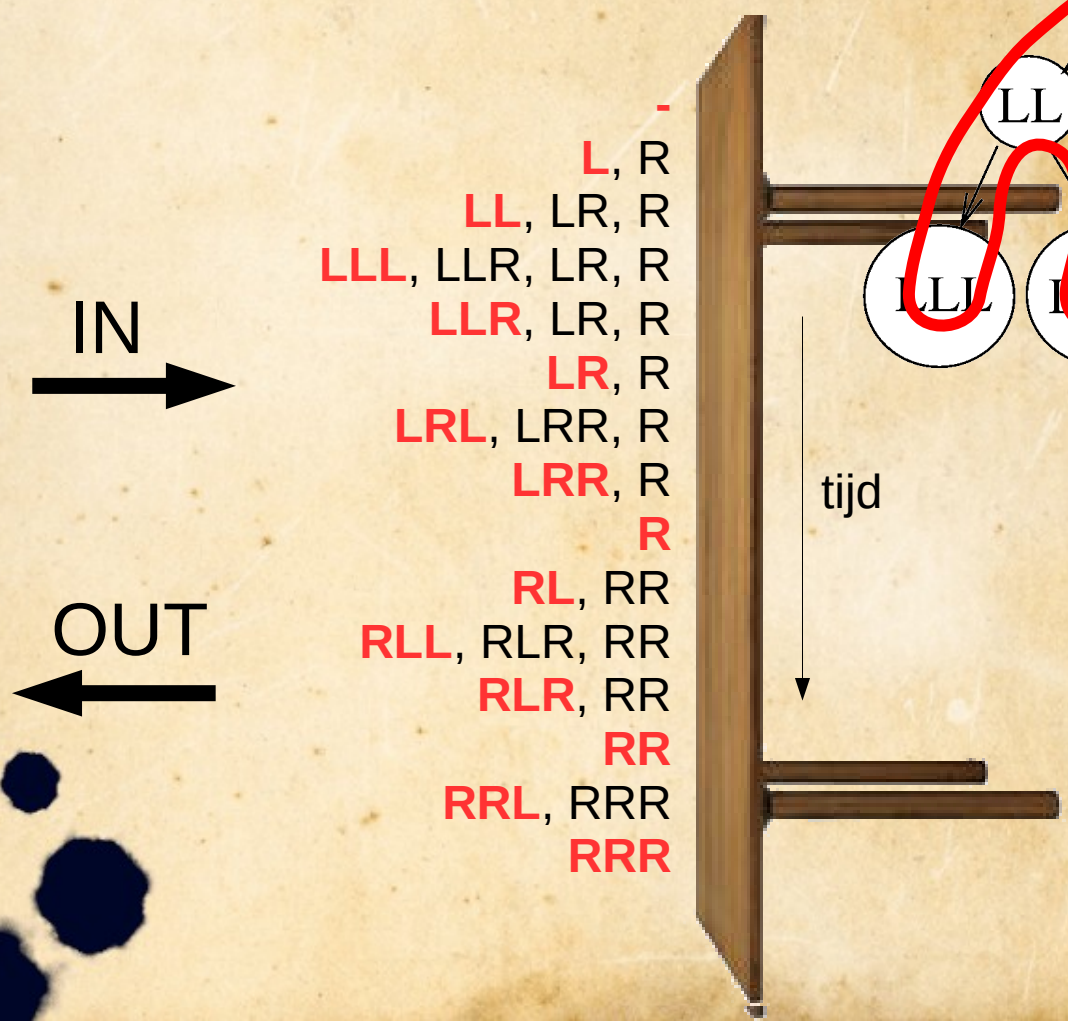

STACK (LIFO)
-
L, R
LL, LR, R
LLL, LLR, LR, R
LL
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

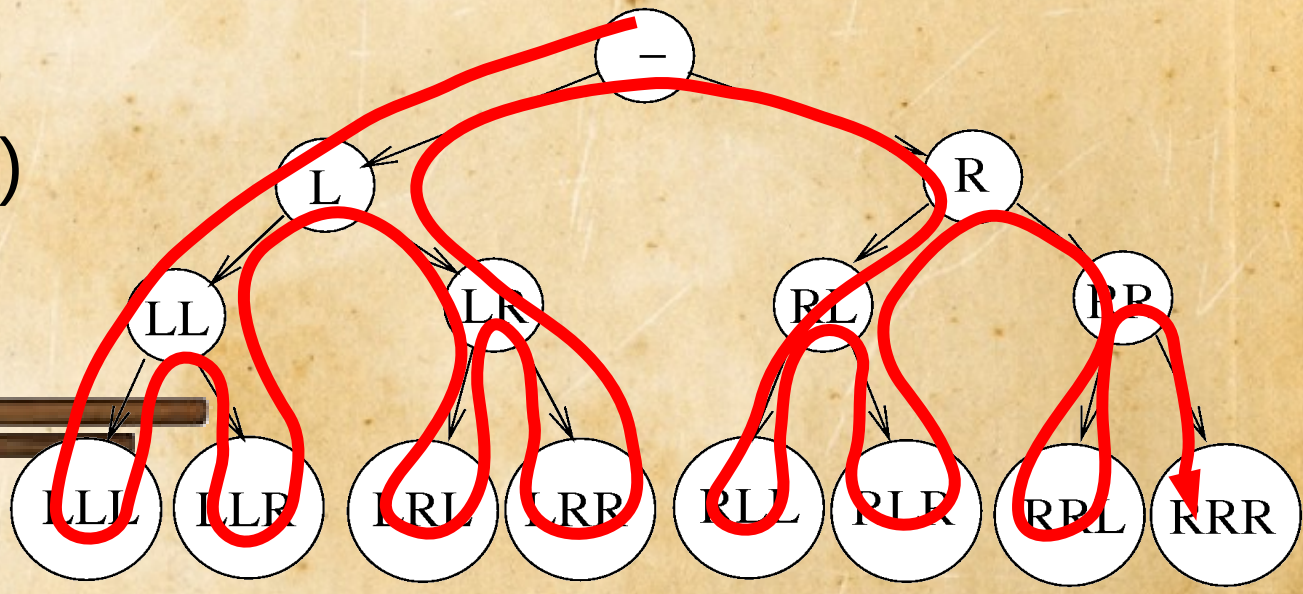
Depth first


STACK (LIFO)
-
L, R
LL, LR, R
LLL, LLR, LR, R
LLR, LR, R
LR, R
LRL, LRR, R
LRR, R
R
RL, RR
RLL, RLR, RR
RLR, RR
RR
RRL, RRR
RRR
IN
OUT
tijd
–
L
R
LL
LR
RL
RR
LLL
LLR
LRL
LRR
RLL
RLR
RRL
RRR

Depth first

STACK (LIFO)





Depth first in Python

```
import copy

depth = 3                                   # no deeper than 'depth'
stack = [""]                                # add begin state to stack
while len(stack)>0:
    state=stack.pop()                       # get top from stack
    print(state)
    if len(state) < depth:                  # stop condition
        for i in ['R','L']:                     # for each possible action:
            child = copy.deepcopy(state)        # deepcopy the state
            child += i                          # make new child
            stack.append(child)                 # put on stack
```

Depth first in Python

```
import copy

depth = 3                                   # no deeper than
stack = [""]                                # add begin stat
while len(stack)>0:
    state=stack.pop()                       # get top from s
    print(state)
    if len(state) < depth:                  # stop condition
        for i in ['R','L']:                     # for each p
            child = copy.deepcopy(state)        # deepcop
            child += i                          # make ne
            stack.append(child)                 # put on
```

```
L
LL
LLL
LLR
LR
LRL
LRR
R
RL
RLL
RLR
RR
RRL
RRR
```

Depth first, kan ook recursief

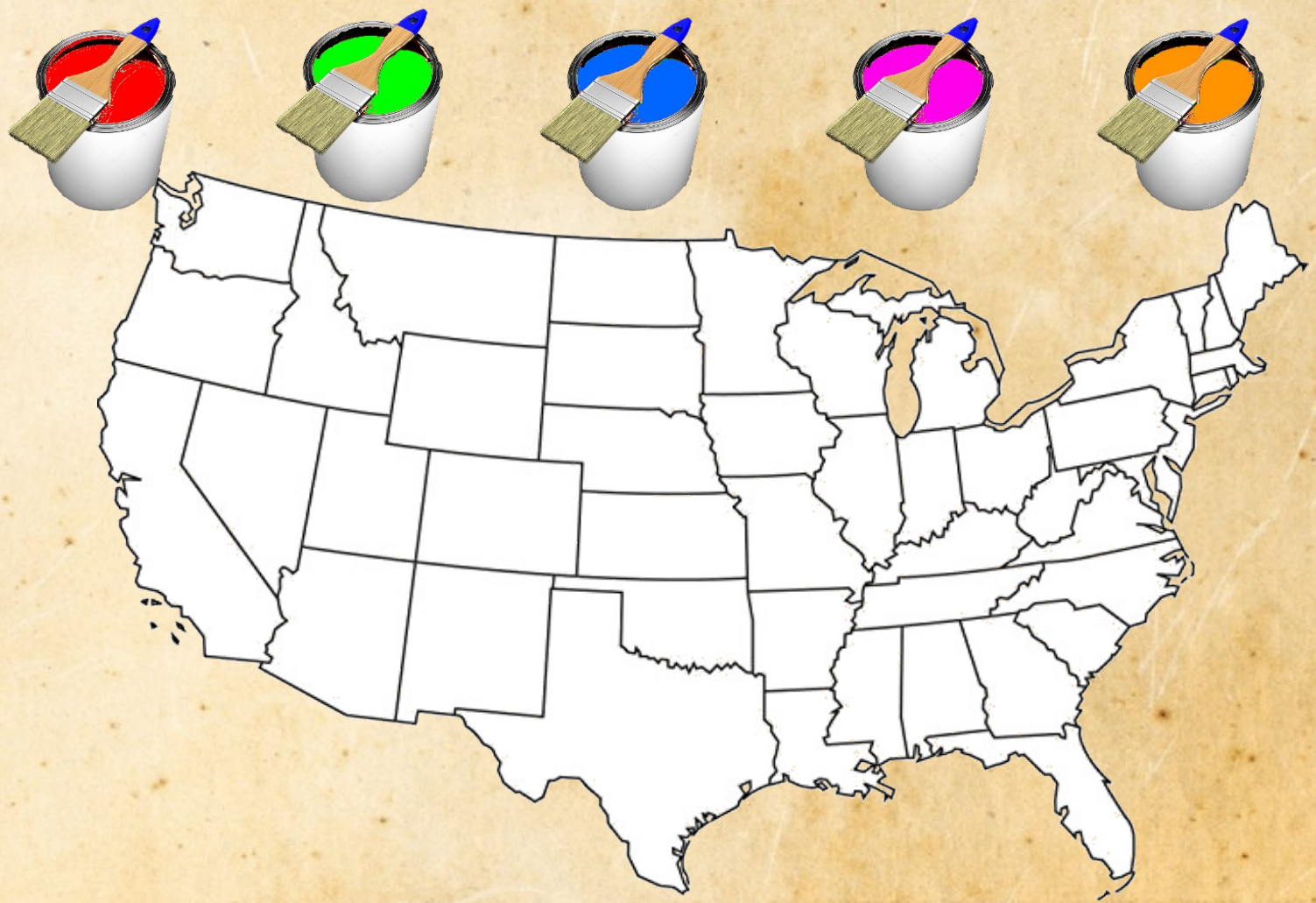
```
def depth_first(state,depth):
    print(state)
    if len(state) < depth:                  # no deeper than 'depth'
        for i in ['L','R']:                 # for each possible action
            depth_first(state+i,depth)      # recursive call

depth_first("",3) # start the recursion
```

Breadth first, kan ook met “generations”

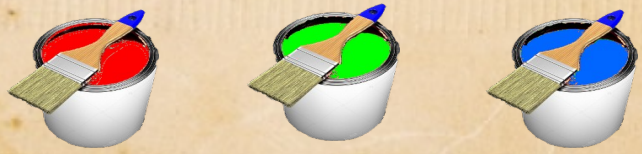
```
import copy

depth = 3                                       # no deeper than 'depth'
generation = [""]                               # add begin state to generation
while len(generation) > 0:
    new_generation = []                         # make a new generation
    for state in generation:                    # for each "parent" in generation
        print(state)
        if len(state) < depth:                  # stop condition
            for i in ['L','R']:                 # for each possible action
                child = copy.deepcopy(state)    # deepcopy the state
                child += i                      # make new child
                new_generation.append(child)    # add to new generation
    generation = new_generation                 # forget previous generation
```

Breadth of Depth first?

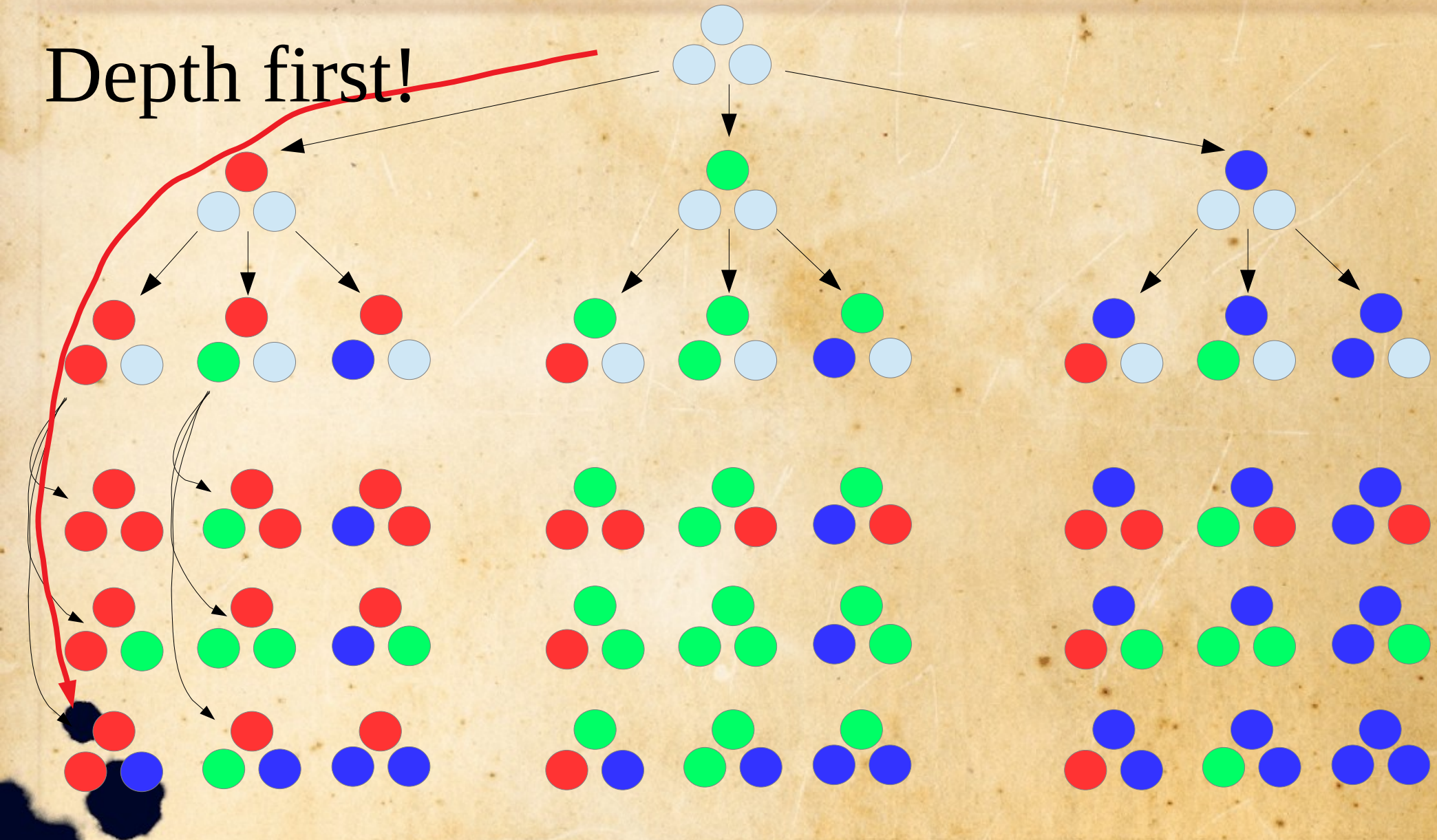
We willen snel oplossingen vinden, maakt niet uit wat het kost



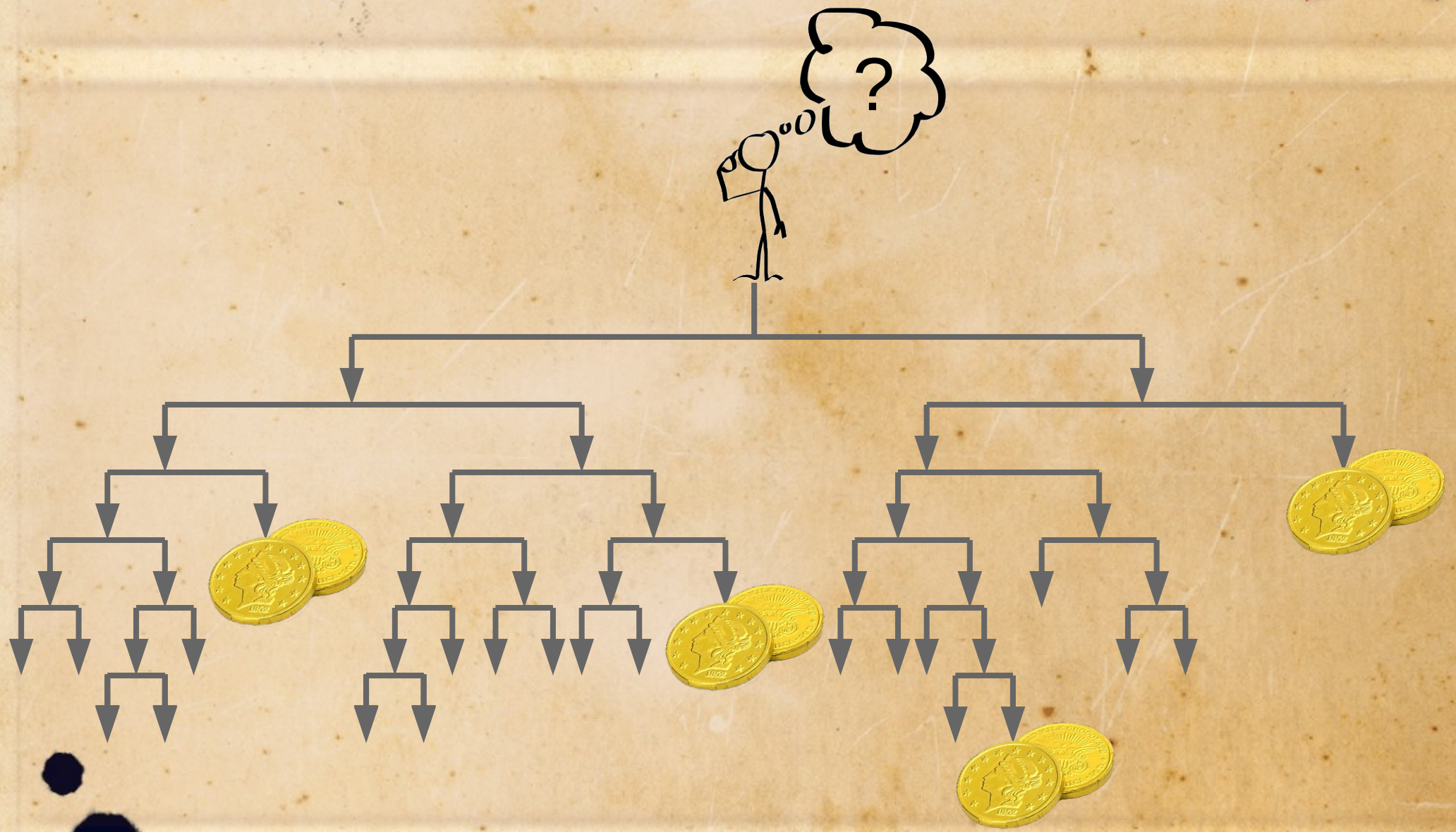
Kaart kleuren



Depth first!

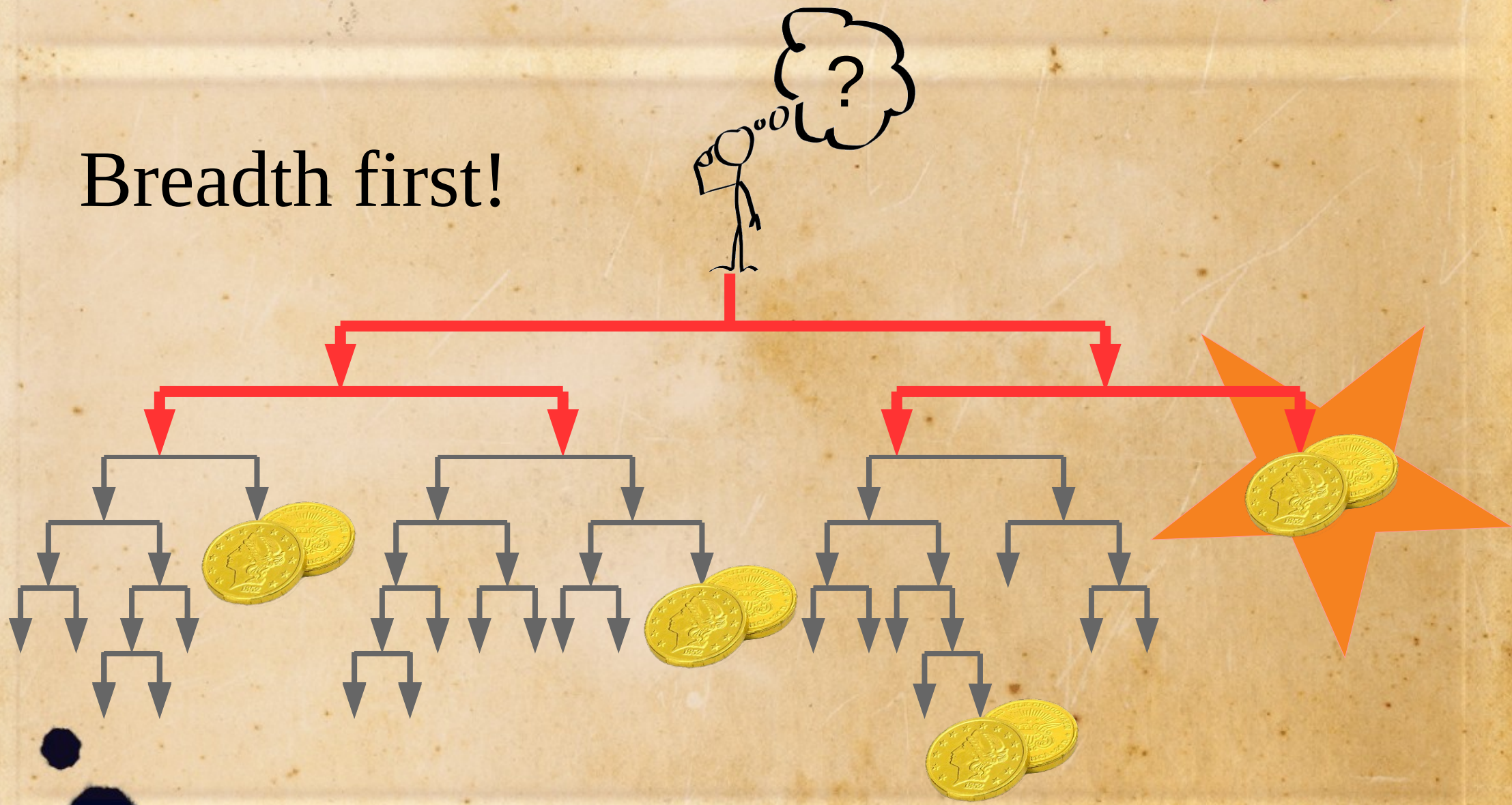


Breadth of Depth first?



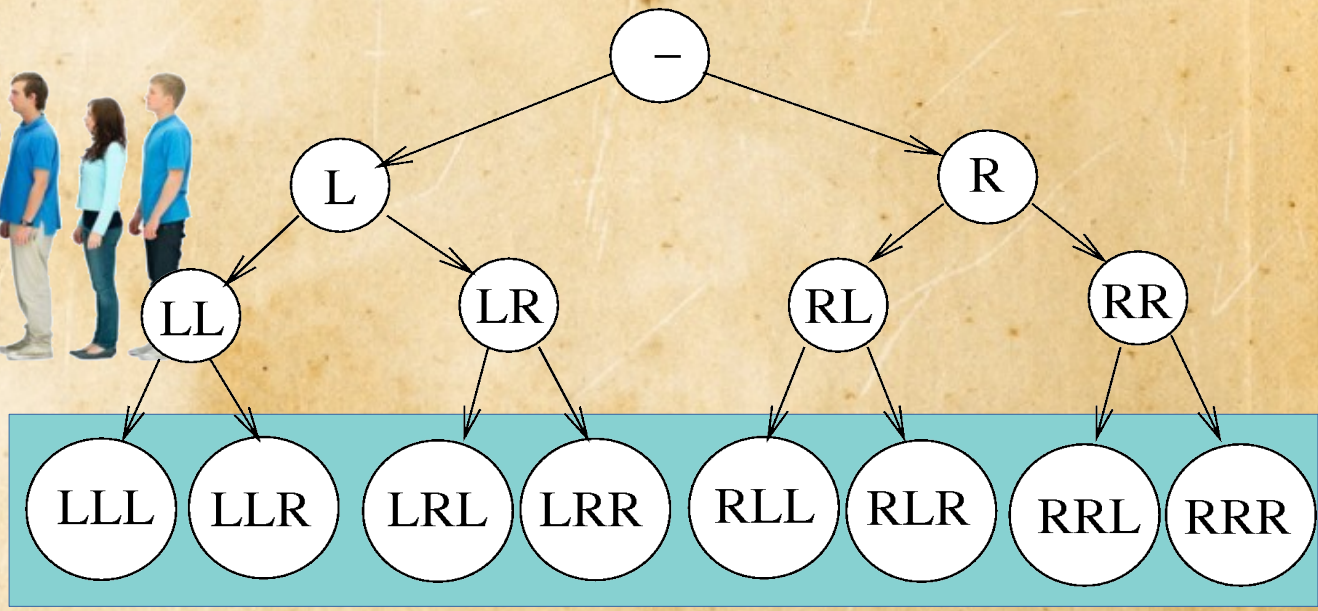
Breadth of Depth first?

Breadth first!



Breadth first, geheugen?

QUEUE (FIFO)



tijd

OUT

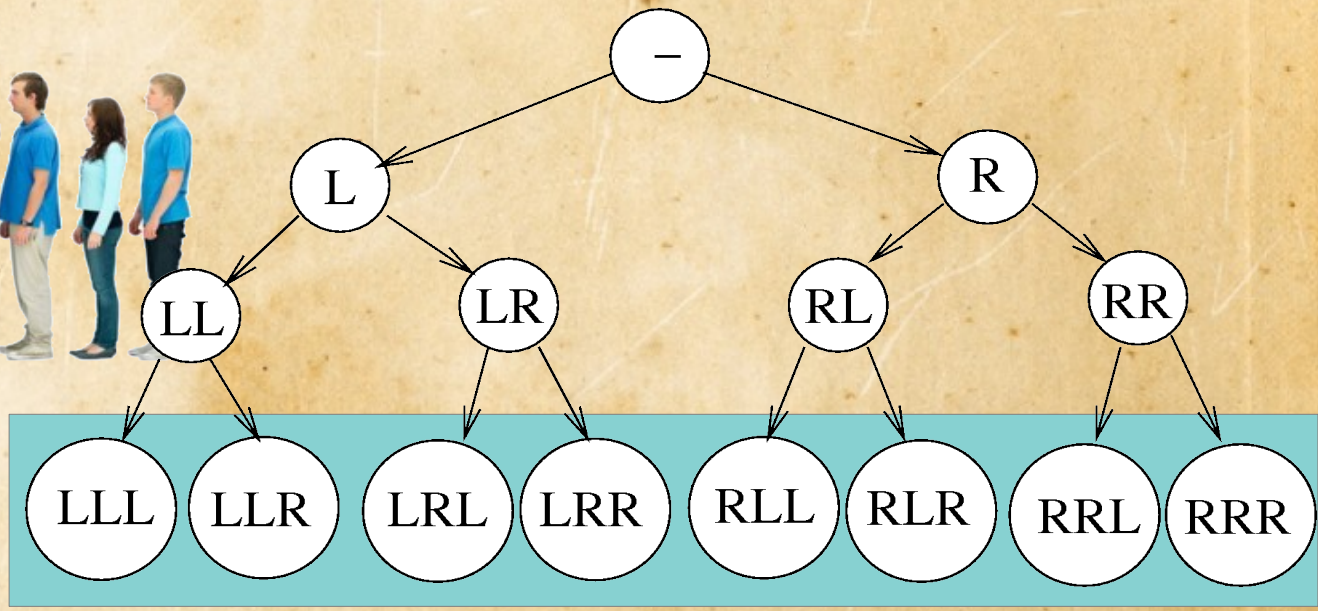
IN

-
L, R
R, LL, LR
LL, LR, RL, RR
LR, RL, RR, LLL, LLR
RL, RR, LLL, LLR, LRL, LRR
RR, LLL, LLR, LRL, LRR, RLL, RLR
LLL, LLR, LRL, LRR, RLL, RLR, RRL, RRR
LLR, LRL, LRR, RLL, RLR, RRL, RRR
LRL, LRR, RLL, RLR, RRL, RRR
LRR, RLL, RLR, RRL, RRR
RLL, RLR, RRL, RRR
RLR, RRL, RRR
RRL, RRR
RRR

Bij zoeken op diepte 3 max 8 states in de Queue
Hoeveel states op diepte 4?
Hoeveel states op diepte 5?
Hoeveel states op diepte N?

Breadth first, geheugen?

QUEUE (FIFO)



tijd

OUT

IN

-
L, R
R, LL, LR
LL, LR, RL, RR
LR, RL, RR, LLL, LLR
RL, RR, LLL, LLR, LRL, LRR
RR, LLL, LLR, LRL, LRR, RLL, RLR
LLL, LLR, LRL, LRR, RLL, RLR, RRL, RRR
LLR, LRL, LRR, RLL, RLR, RRL, RRR
LRL, LRR, RLL, RLR, RRL, RRR
LRR, RLL, RLR, RRL, RRR
RLL, RLR, RRL, RRR
RLR, RRL, RRR
RRL, RRR
RRR

Bij zoeken op diepte 3 max 8 states in de Queue
Hoeveel states op diepte 4?
Hoeveel states op diepte 5?

Hoeveel states op diepte N? 2^N

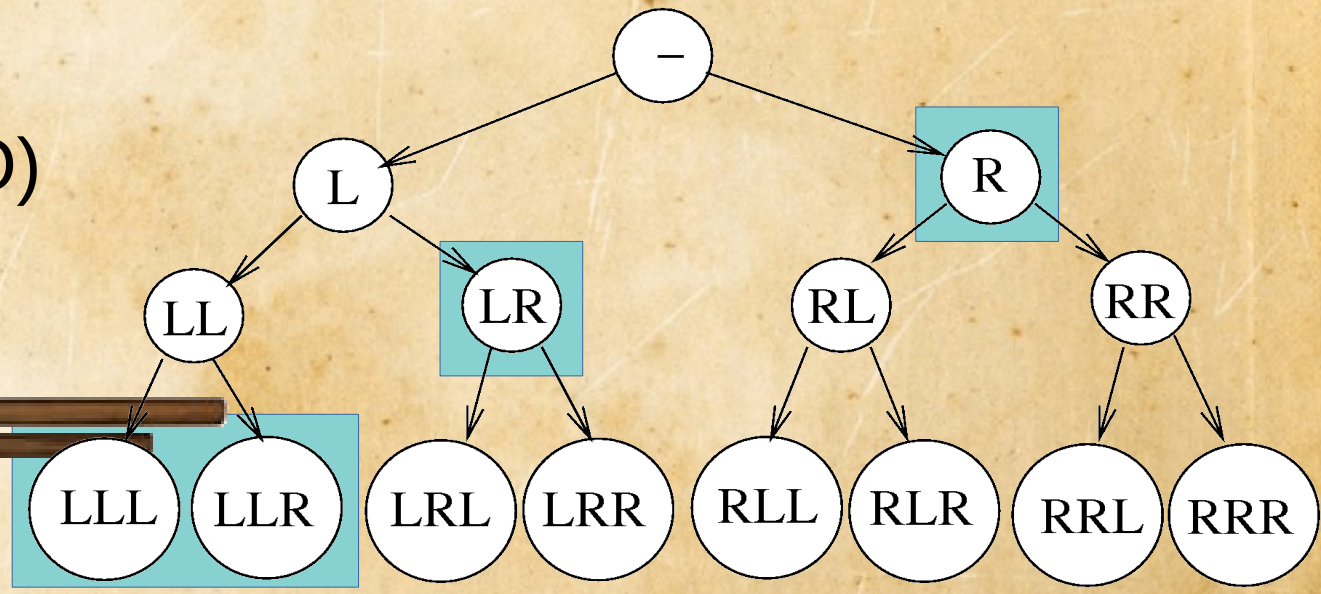
Depth first, geheugen?

STACK (LIFO)

IN

OUT

tijd



Bij zoeken op diepte 3 max 4 states op de Stack
Hoeveel states op diepte 4?
Hoeveel states op diepte N?

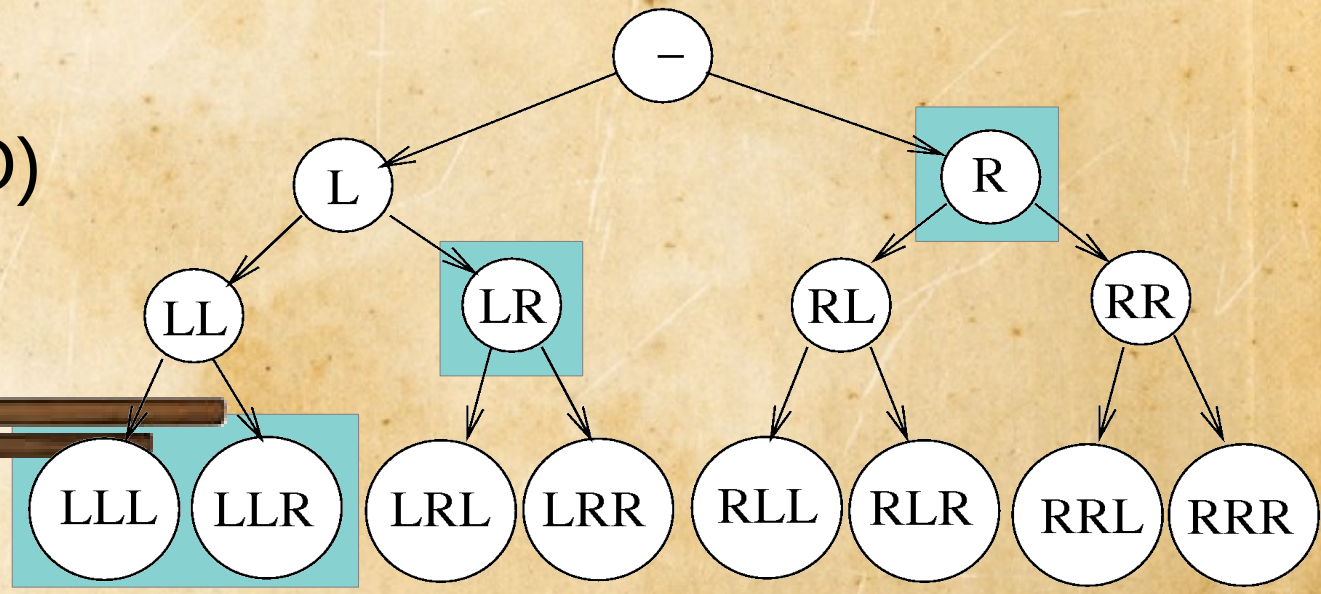
Depth first, geheugen?

STACK (LIFO)

IN

OUT

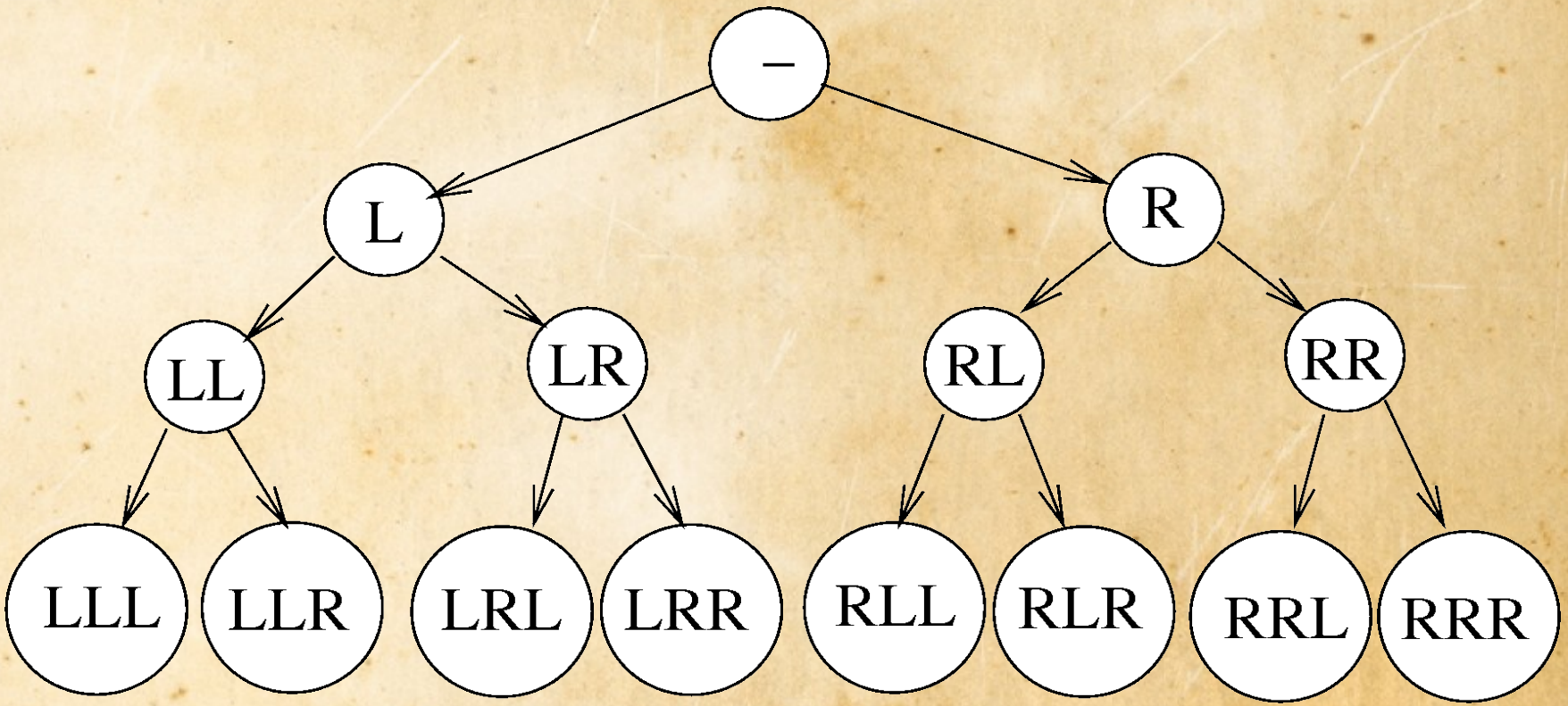
tijd



Bij zoeken op diepte 3 max 4 states op de Stack
Hoeveel states op diepte 4?
Hoeveel states op diepte N?

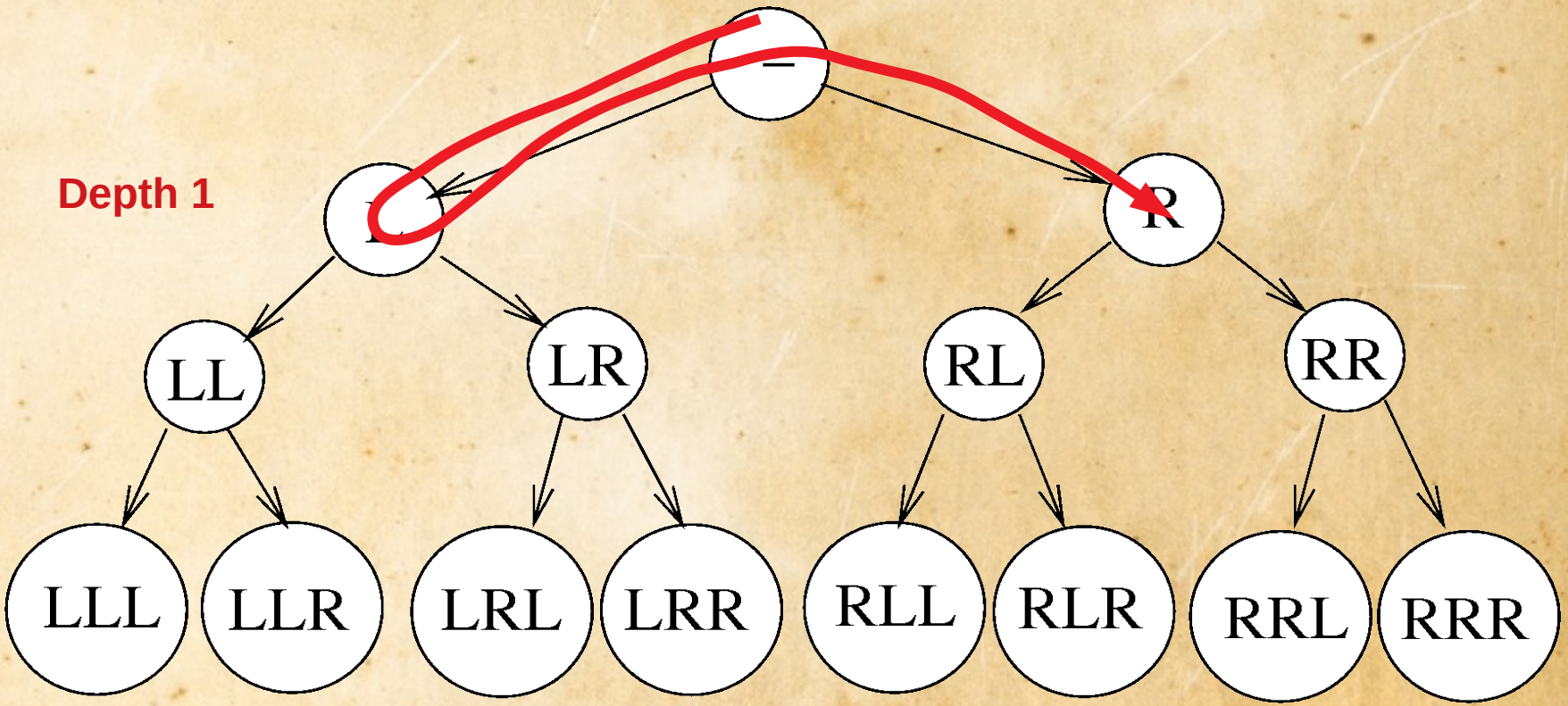
$2 + (2-1) \times (N-2) < \mathbf{2 \times N}$

Iterative deepening depth first



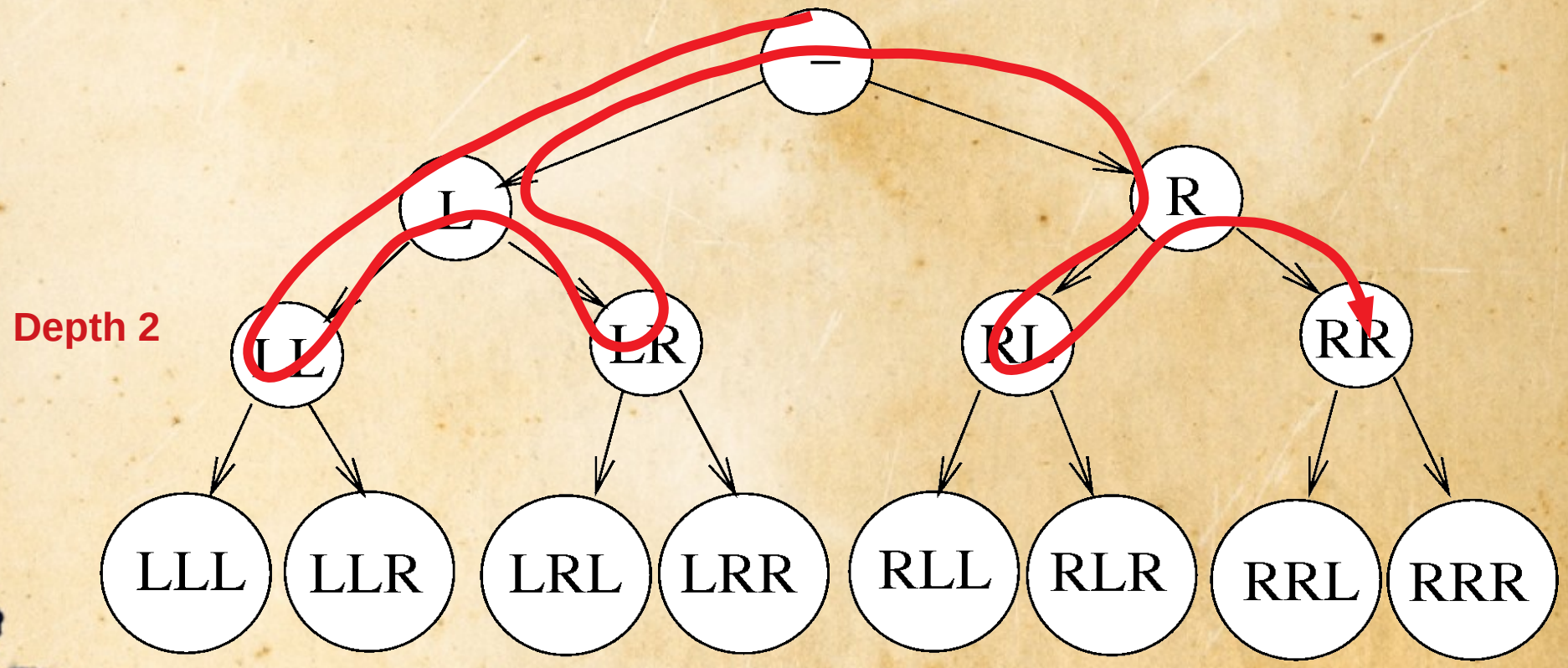
Iterative deepening depth first

- Eerst depth first zoeken tot diepte 1



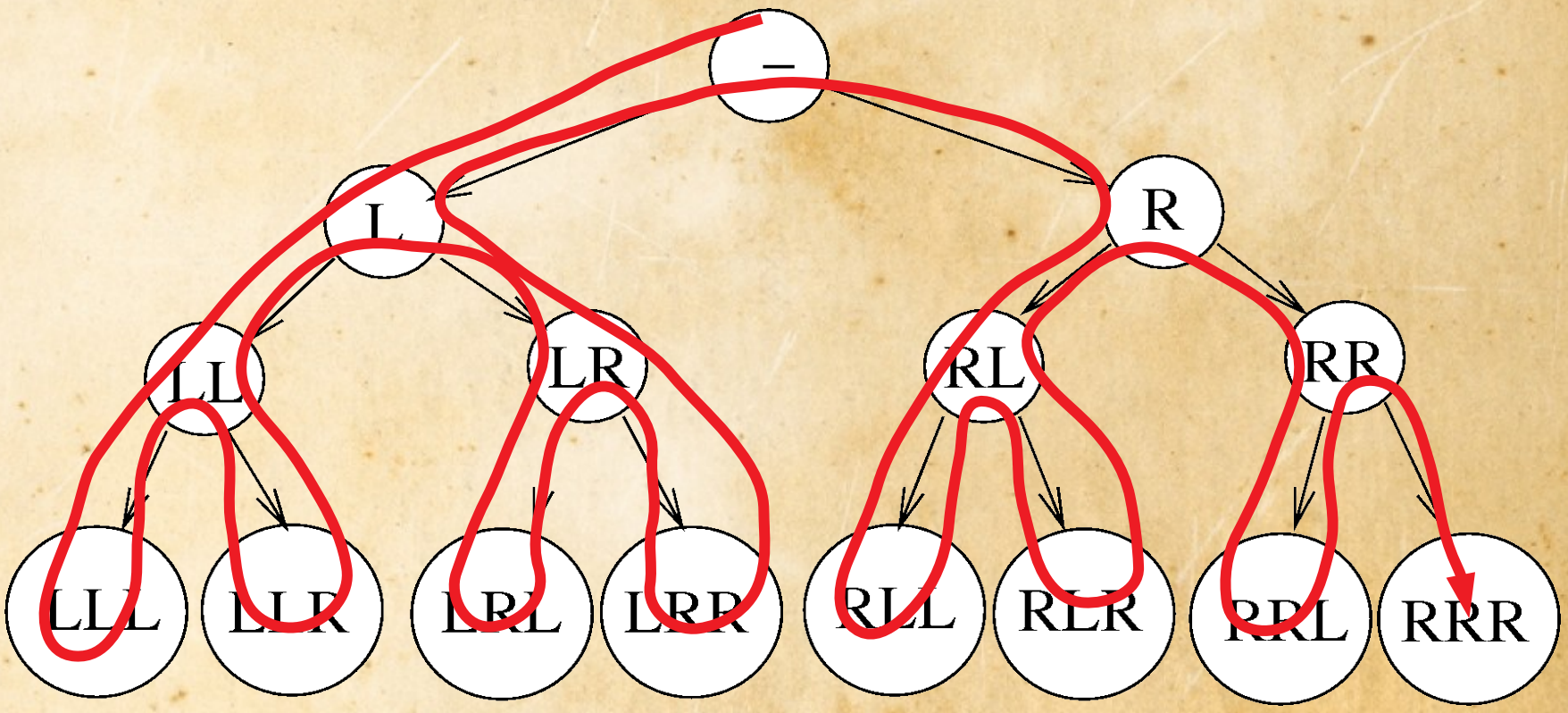
Iterative deepening depth first

- Dan opnieuw beginnen met zoeken tot diepte 2



Iterative deepening depth first

- Dan opnieuw beginnen met zoeken tot diepte 3

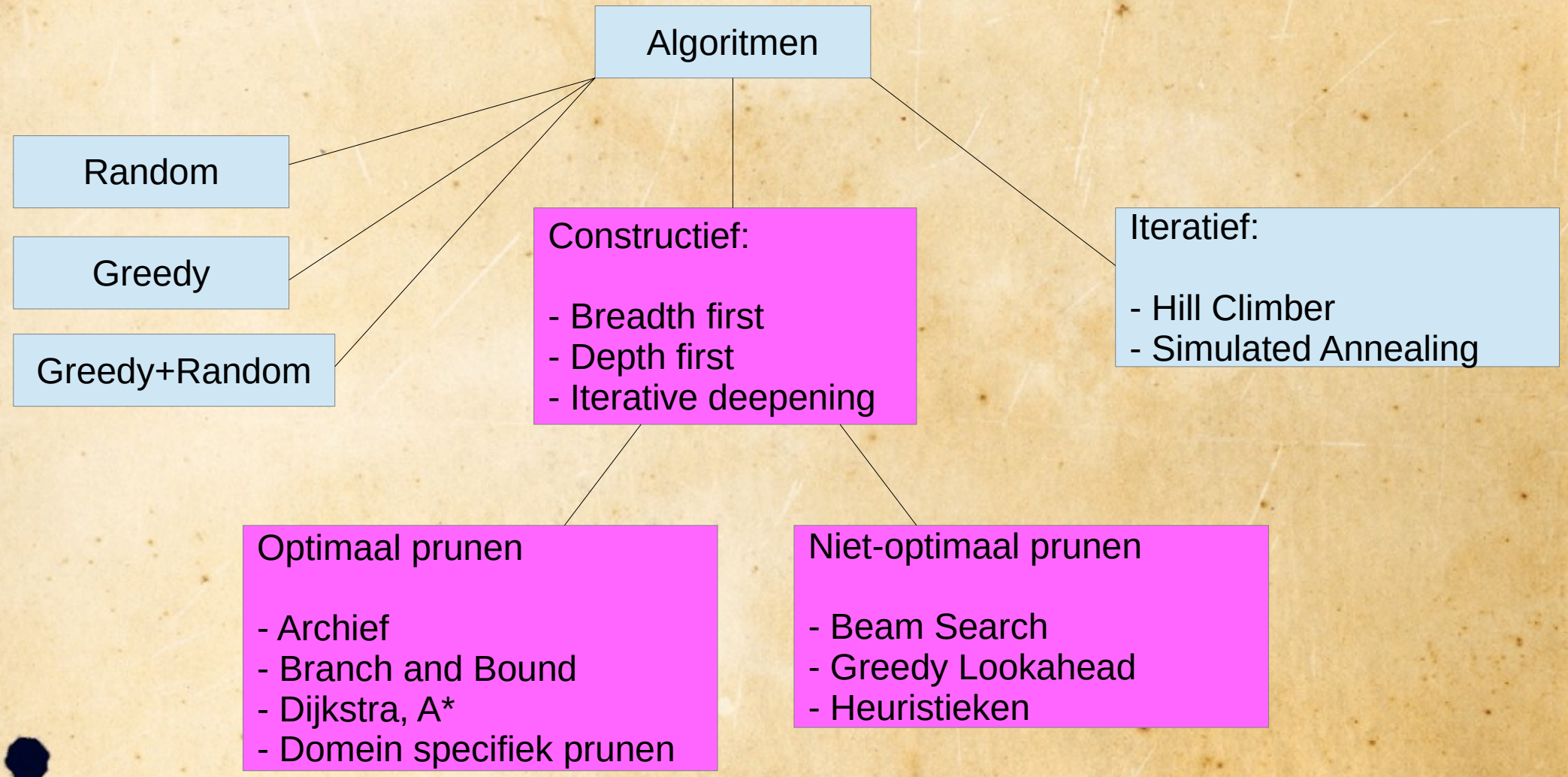


Depth 3

Breadth/Depth first demo

- Path finding in grid world met depth=4
  - python breadthFirstViz.py grid.gr 55 0 4
  - python depthFirstViz.py grid.gr 55 0 4

Algoritmen

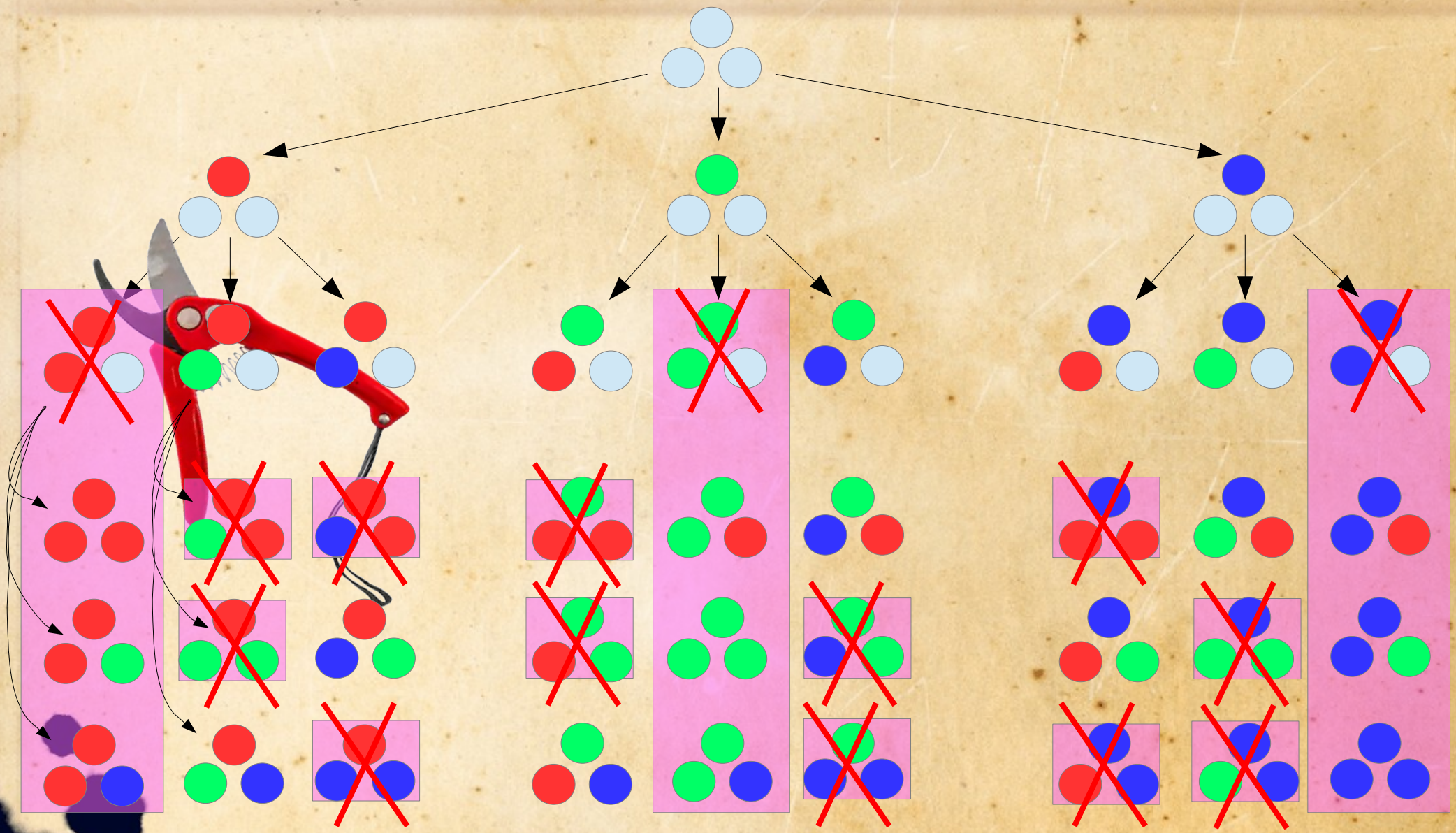

Algoritmen
Random
Greedy
Greedy+Random
Constructief:
- Breadth first
- Depth first
- Iterative deepening
Iteratief:
- Hill Climber
- Simulated Annealing
Optimaal prunen
- Archief
- Branch and Bound
- Dijkstra, A*
- Domein specifiek prunen
Niet-optimaal prunen
- Beam Search
- Greedy Lookahead
- Heuristieken

Optimaal prunen

Early Constraint Checking

- Breadth of Depth first zoeken:
  - Constraints zo **vroeg mogelijk** controleren

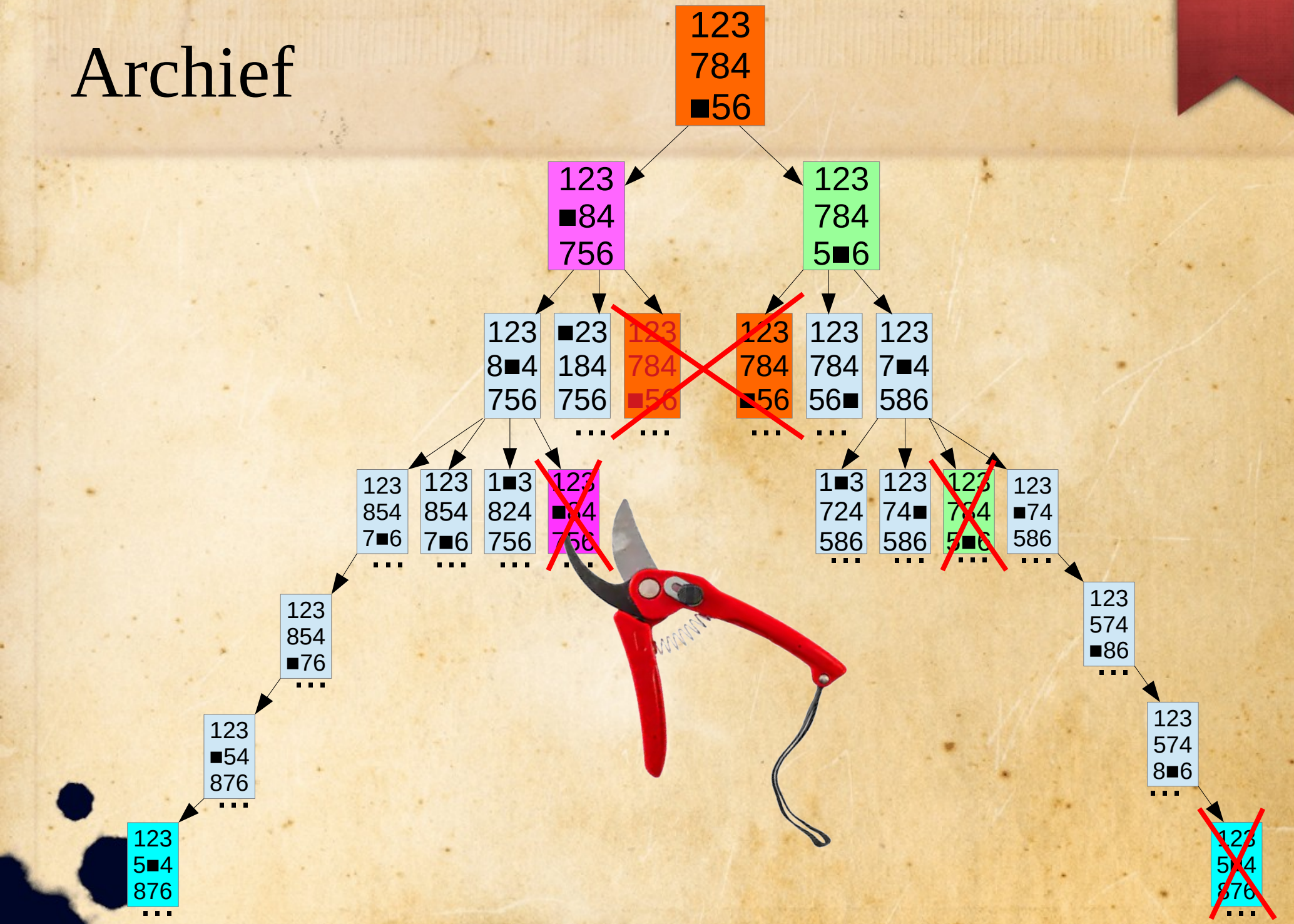
Early Constraint Checking



Archief

- Breadth of Depth first zoeken
  - **Onthouden van states** die al bezocht zijn
    - bv door deze op te slaan in een Python **set** of **dictionary** zodat je **snel** states kunt terugvinden

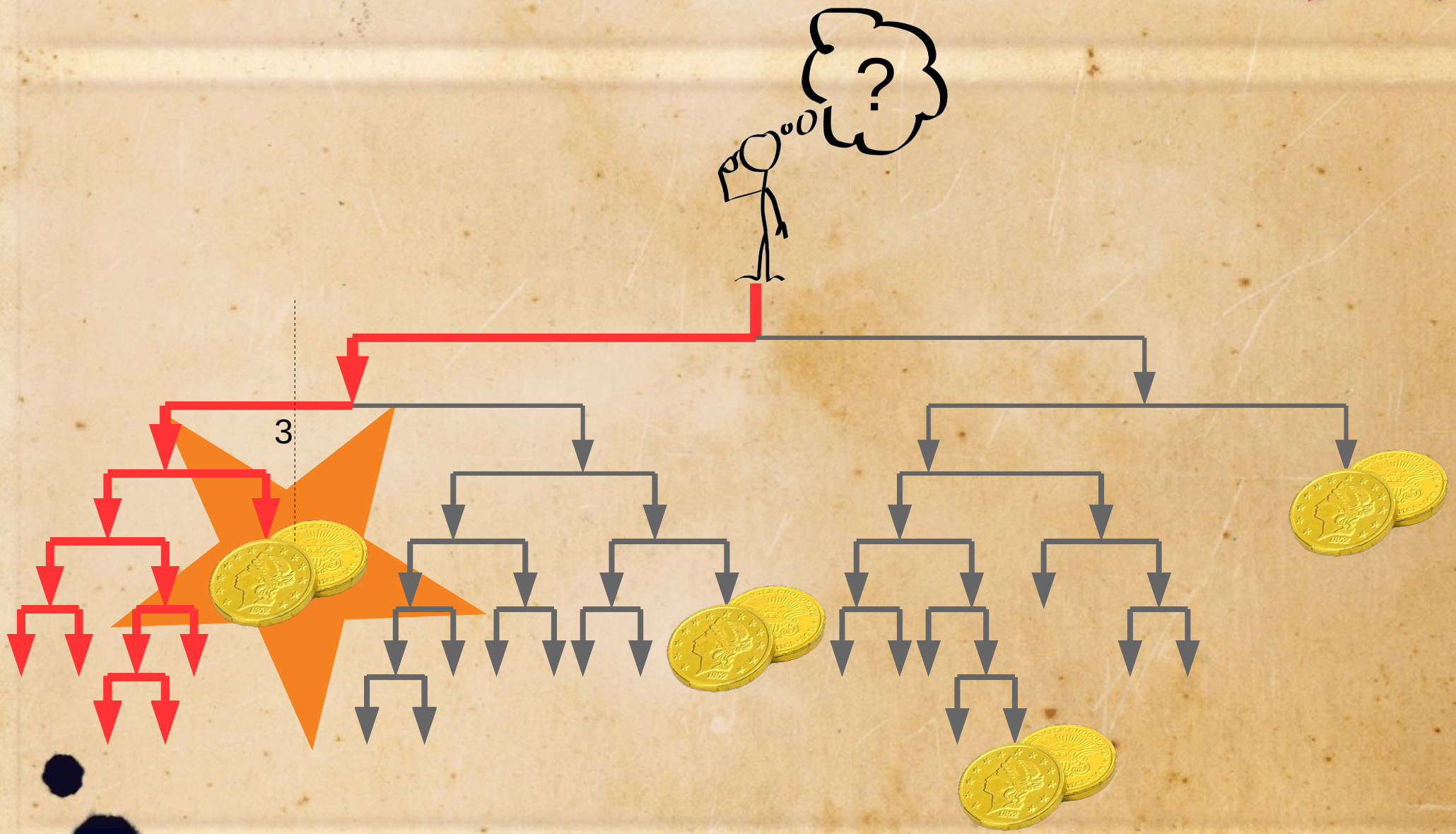
Archief


123
784
■56
123
■84
756
123
784
5■6
123
8■4
756
■23
184
756
123
784
■56
123
784
56■
123
7■4
586
123
854
7■6
123
854
7■6
1■3
824
756
1■3
724
586
123
74■
586
123
■74
586
123
854
■76
123
574
■86
123
■54
876
123
574
8■6
123
5■4
876

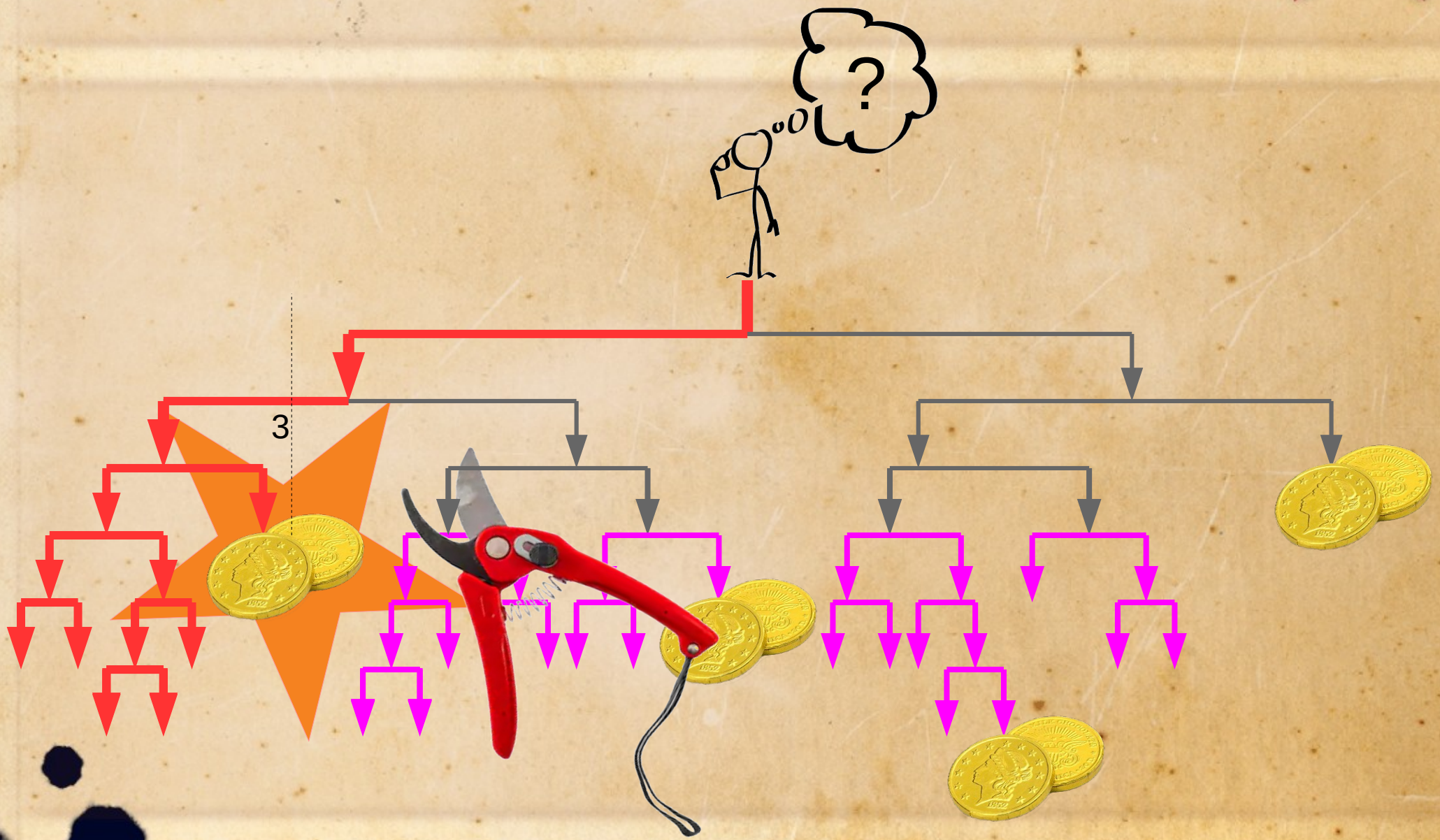
Branch and Bound

- Depth first zoeken
  - Maar niet **dieper** dan **kortst** gevonden pad

Branch and Bound

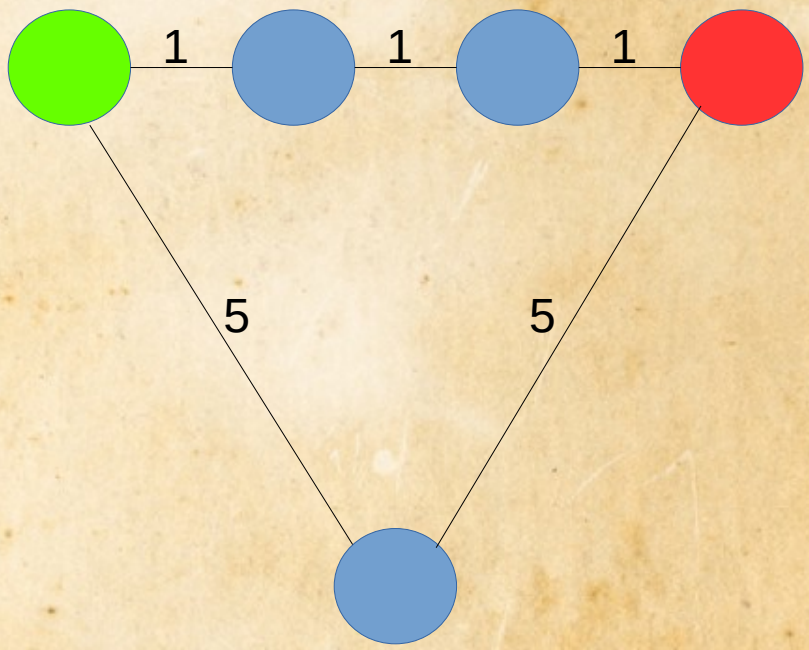


Branch and Bound



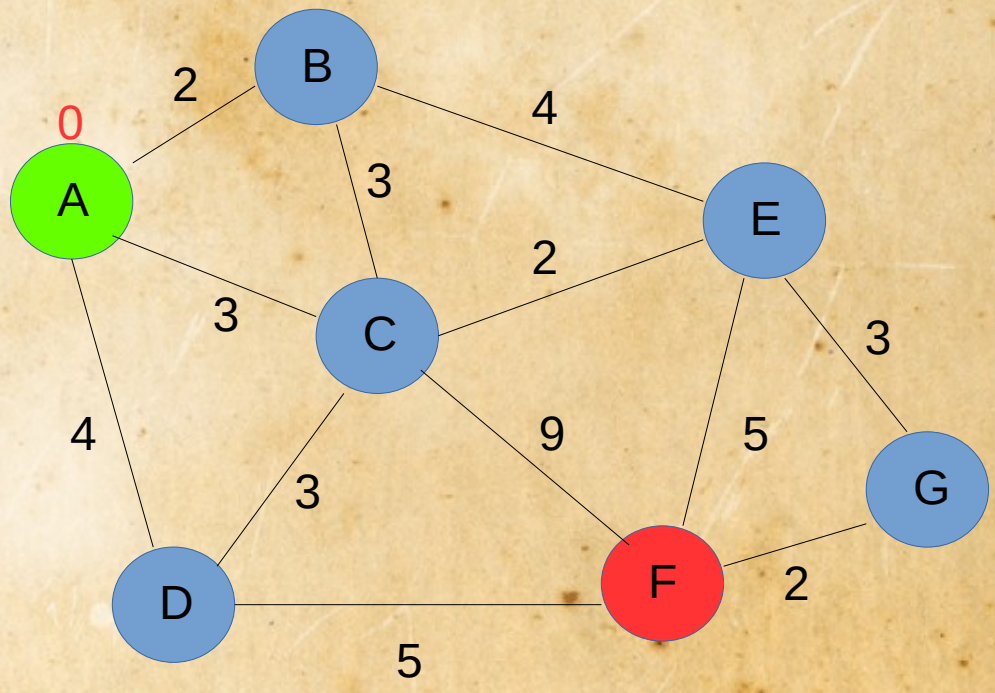
Dijkstra Kortste Pad Algoritme

- Voor kortste pad in een graaf met verschillende kosten
  - Kortste pad garantie bij **geen** negatieve kosten



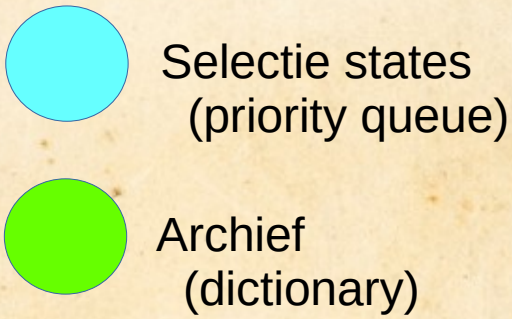
Dijkstra Kortste Pad

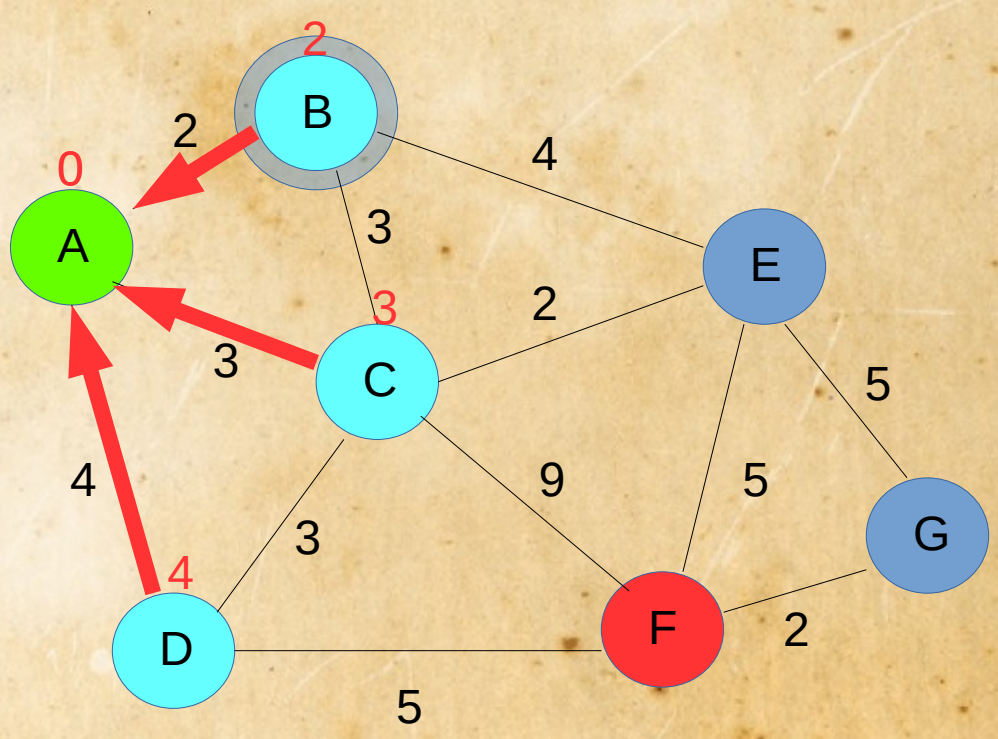
- Selecteer steeds de state met laagste totale kosten



Dijkstra Kortste Pad

- Selecteer steeds de state met laagste totale kosten



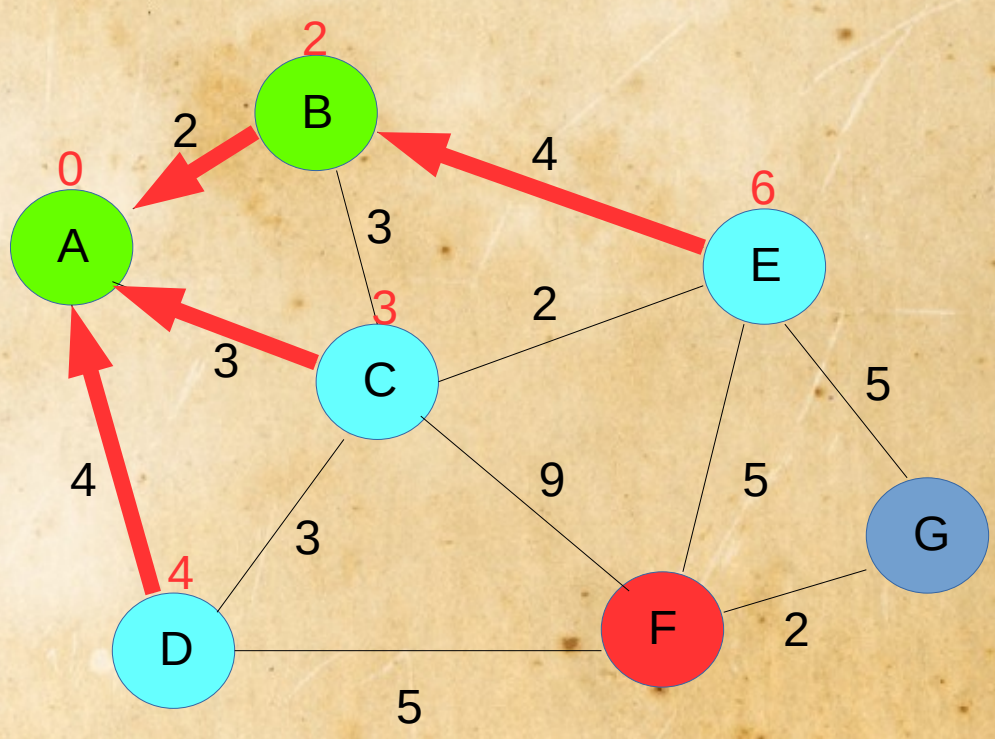


Dijkstra Kortste Pad

- Selecteer steeds de state met laagste totale kosten

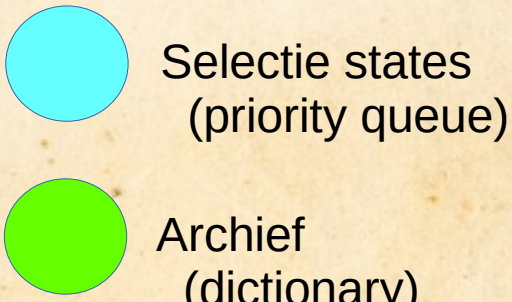
Selectie states
(priority queue)

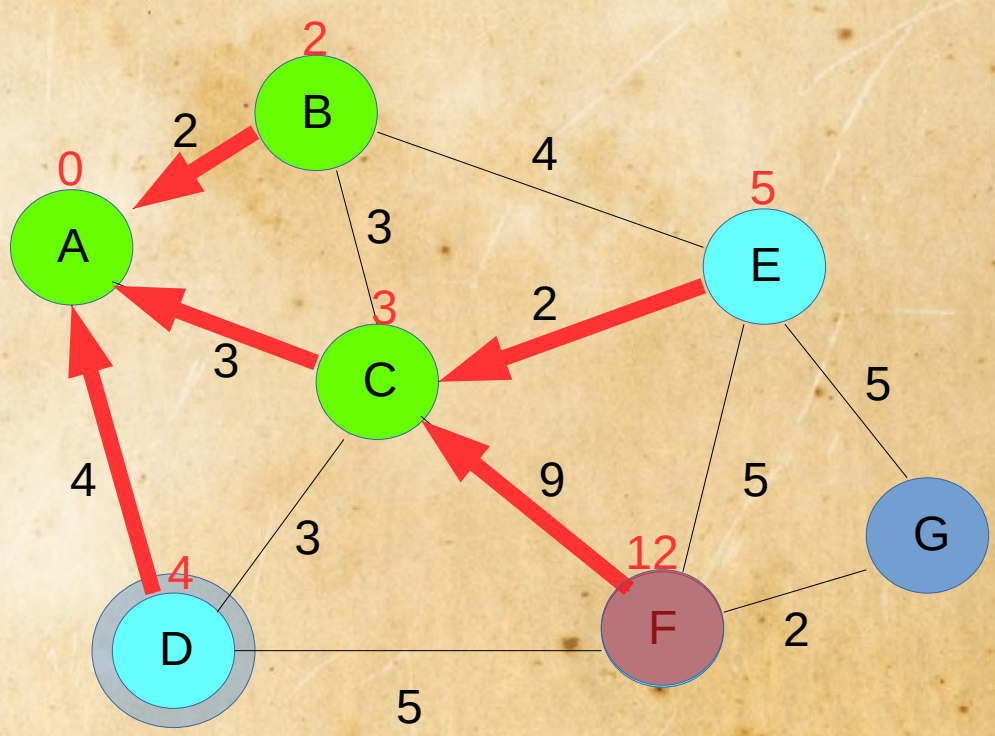
Archief
(dictionary)



Dijkstra Kortste Pad

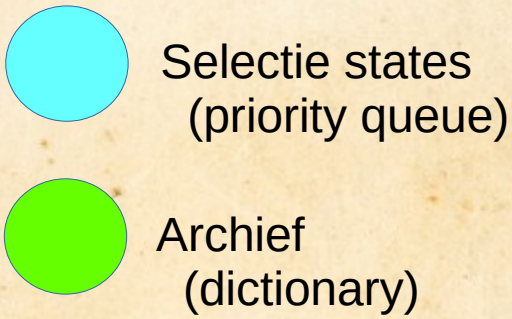
- Selecteer steeds de state met laagste totale kosten

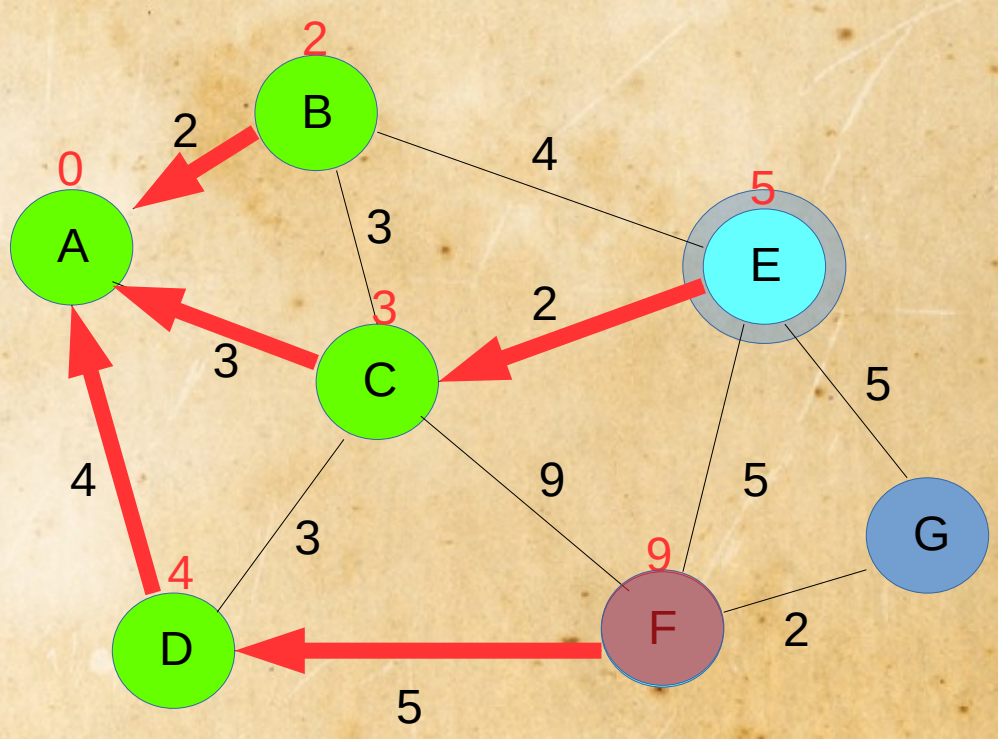




Dijkstra Kortste Pad

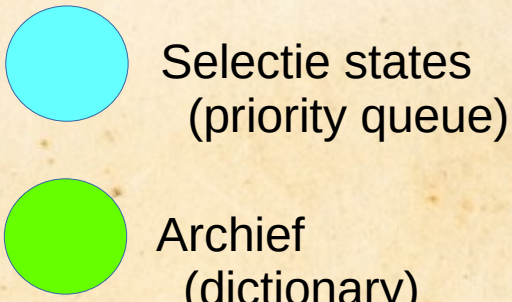
- Selecteer steeds de state met laagste totale kosten

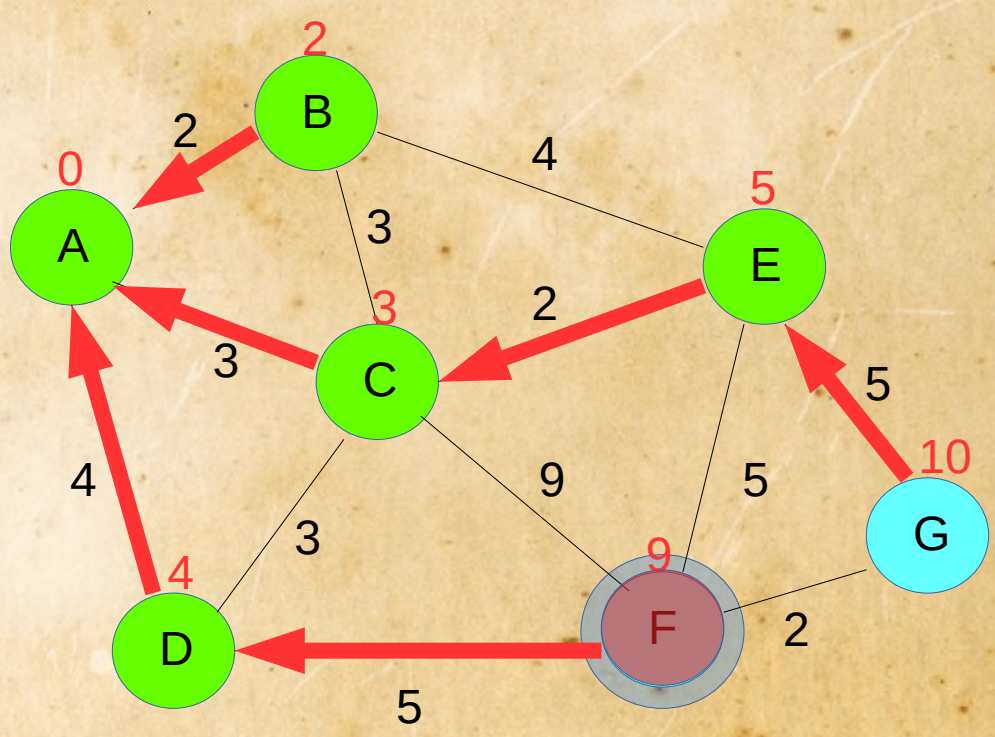




Dijkstra Kortste Pad

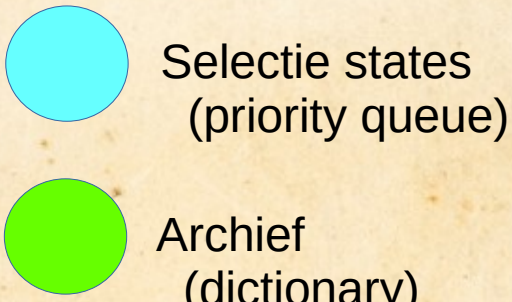
- Selecteer steeds de state met laagste totale kosten

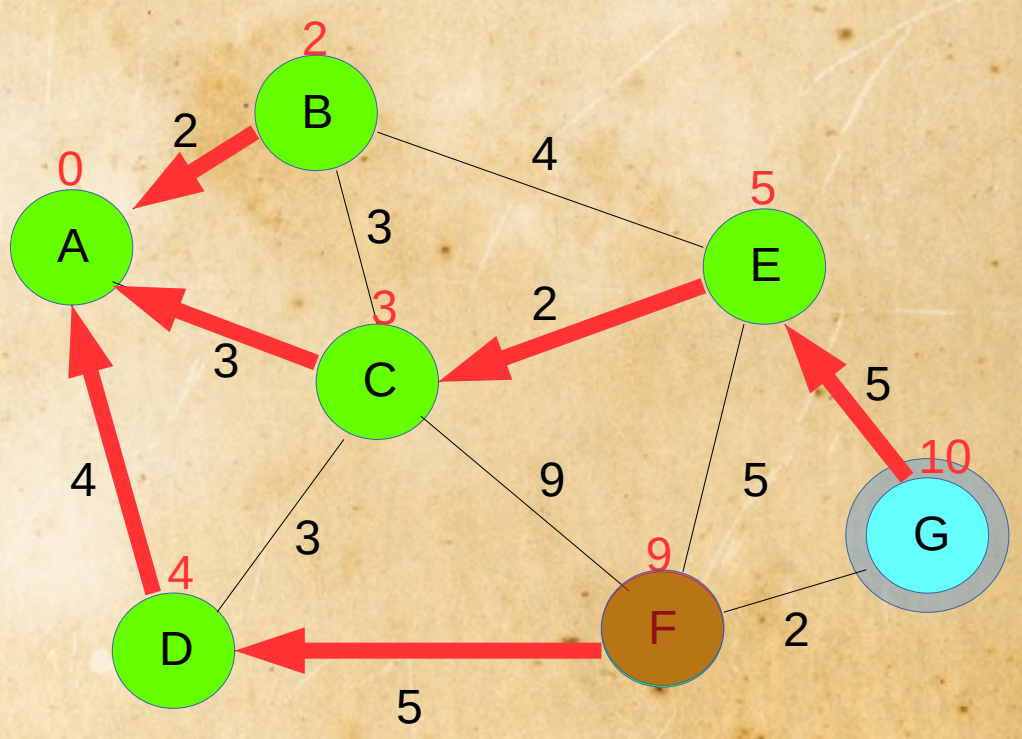




Dijkstra Kortste Pad

- Selecteer steeds de state met laagste totale kosten





A*

- Optimalisatie op Dijkstra met gebruik van **heuristiek**
  - De **heuristiek** geeft een schatting van de kosten van het pad wat we nog moeten afleggen
  - Korste pad garantie wanneer:
    - er **geen** negatieve kosten voorkomen
    - wanneer de heuristiek nooit een overschatting van de kosten van een pad geeft (**admissable)**

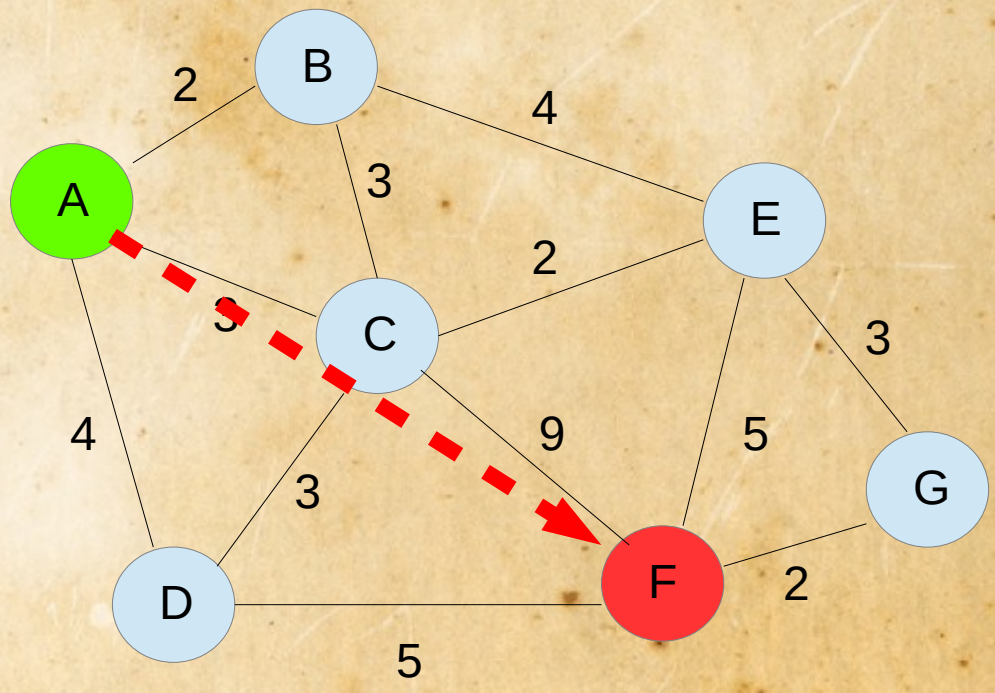
A*

- **Admissable,** geen overschatting van kosten

A*

- Selecteer steeds de state met laagste som van kosten en heuristiek

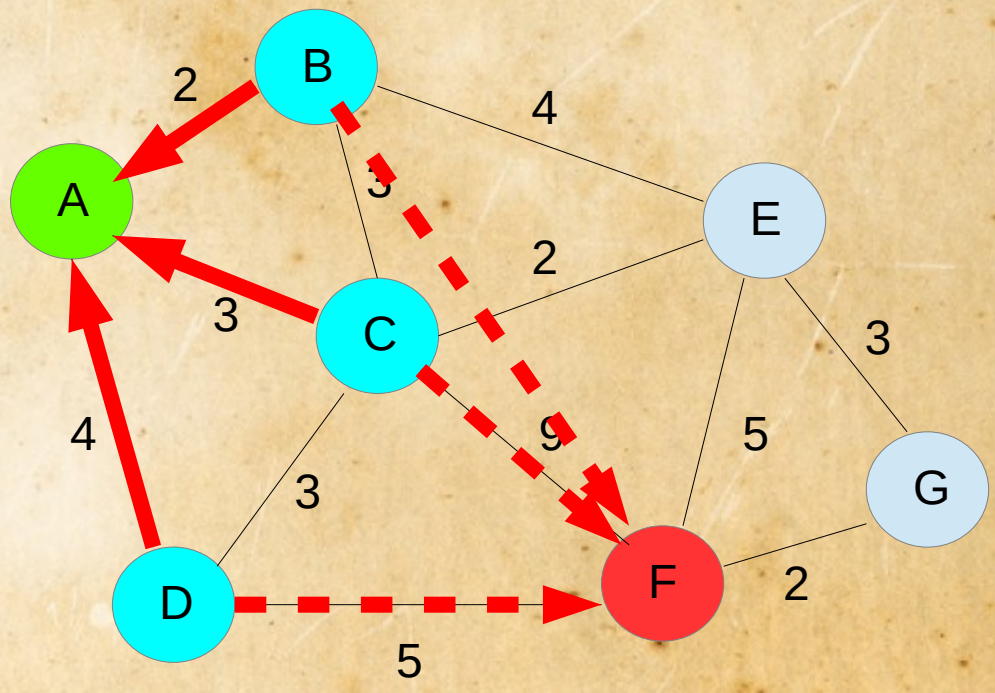
node	prev	cost	heur	sum
A	-	0	8	8



A*

- Selecteer steeds de state met laagste som van kosten en heuristiek

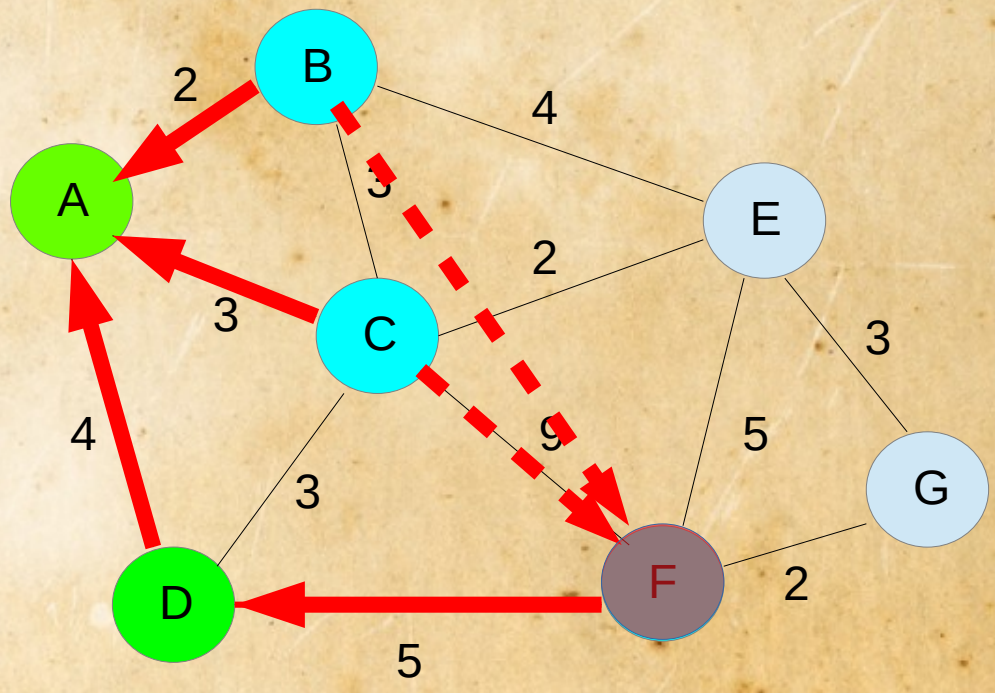
node	prev	cost	heur	sum
A	-	0	8	8
B	A	2	9	11
C	A	3	7	10
D	A	4	5	9



A*

- Selecteer steeds de state met laagste som van kosten en heuristiek

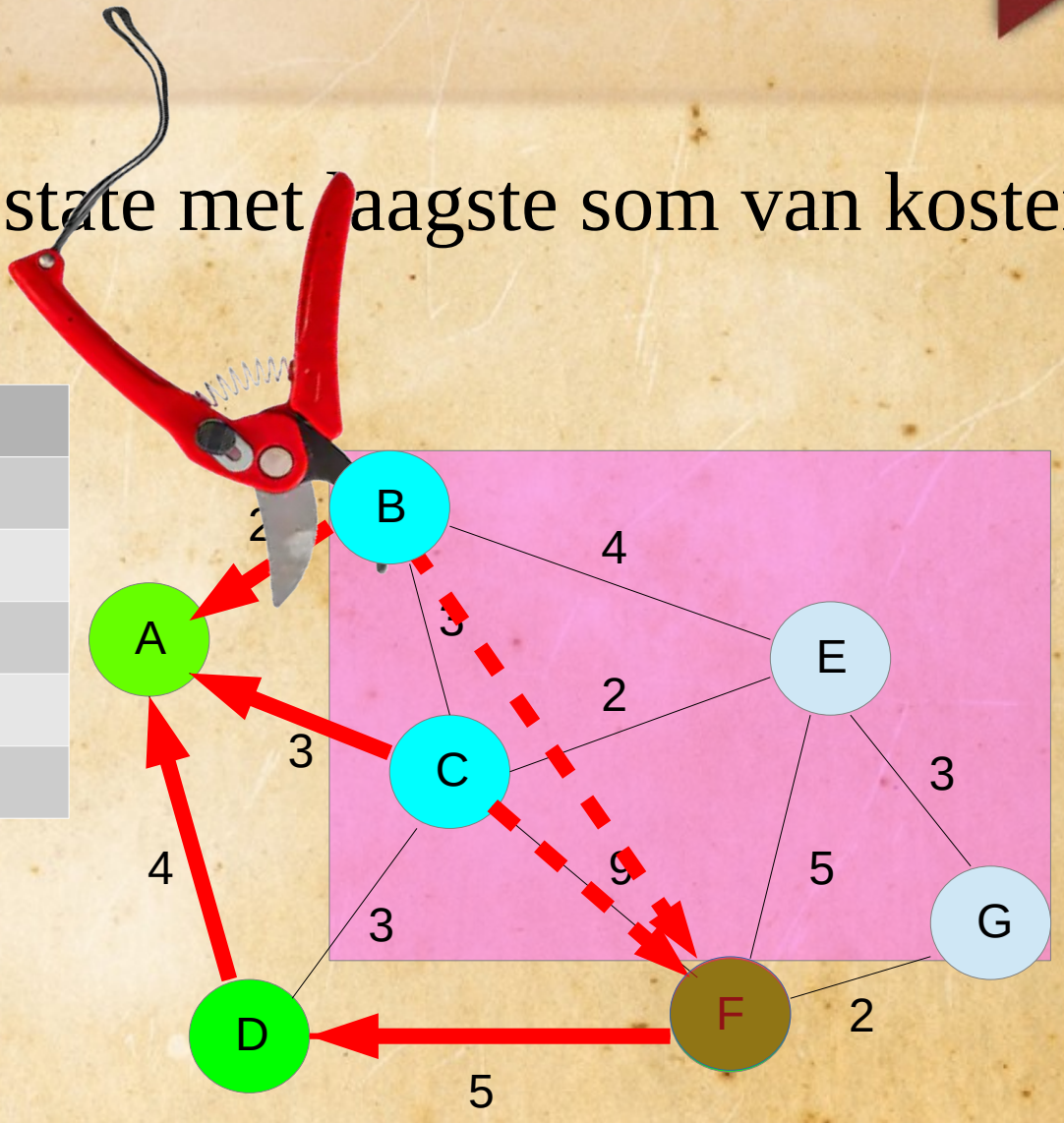
node	prev	cost	heur	sum
A	-	0	8	8
B	A	2	9	11
C	A	3	7	10
D	A	4	5	9
F	D	9	0	9



A*

- Selecteer steeds de state met laagste som van kosten en heuristiek

node	prev	cost	heur	sum
A	-	0	8	8
B	A	2	9	11
C	A	3	7	10
D	A	4	5	9
F	D	9	0	9

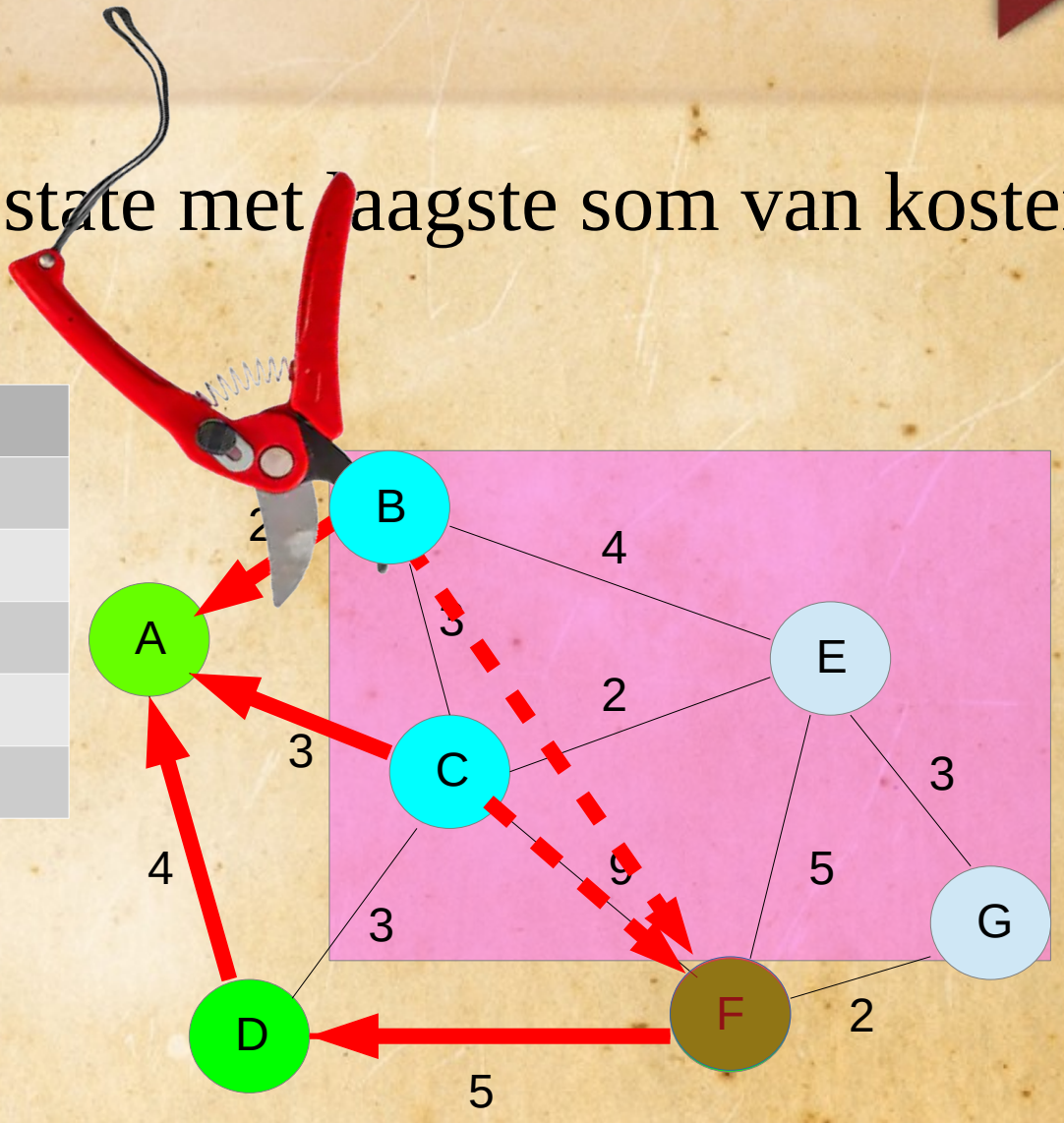


A*

- Selecteer steeds de state met laagste som van kosten en heuristiek

node	prev	cost	heur	sum
A	-	0	0	8
B	A	2	0	2
C	A	3	0	3
D	A	4	0	4
F	D	9	0	9

Als heur=0 dan is
A* gelijk aan Dijkstra



A*

- Demos:
  - http://qiao.github.io/PathFinding.js/visual/
  - Path Planning in games, Daan van der Berg
    - Gras: 1
    - Kleine heuvel: 2
    - Heuvel: 3
    - Bos: 12
    - Berg: 14
    - Water: 1000000

A*

- Slide puzzle

3	1
2	

Heuristic?

1	2
3	

A*

- Slide puzzle

3	1
2	

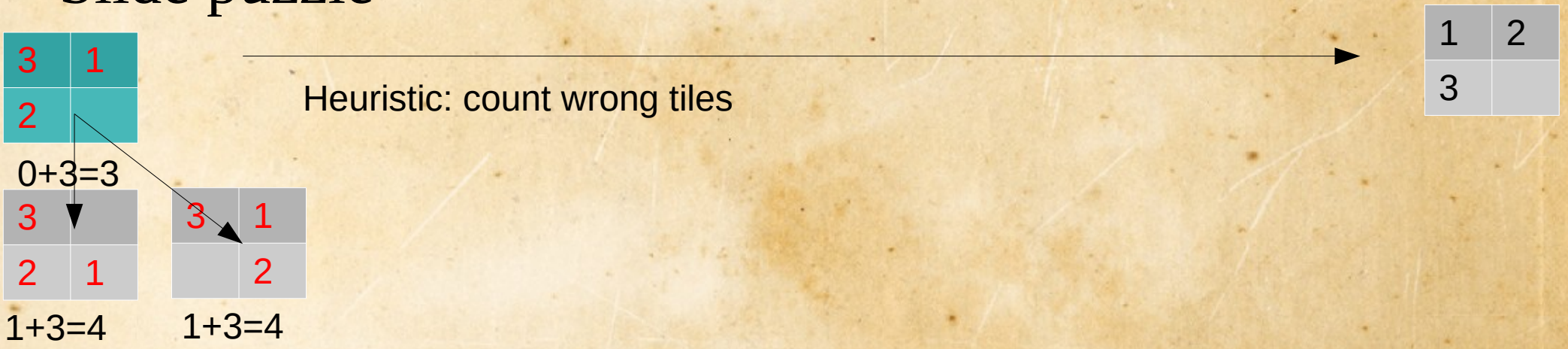
0+3=3

Heuristic: count wrong tiles

1	2
3	

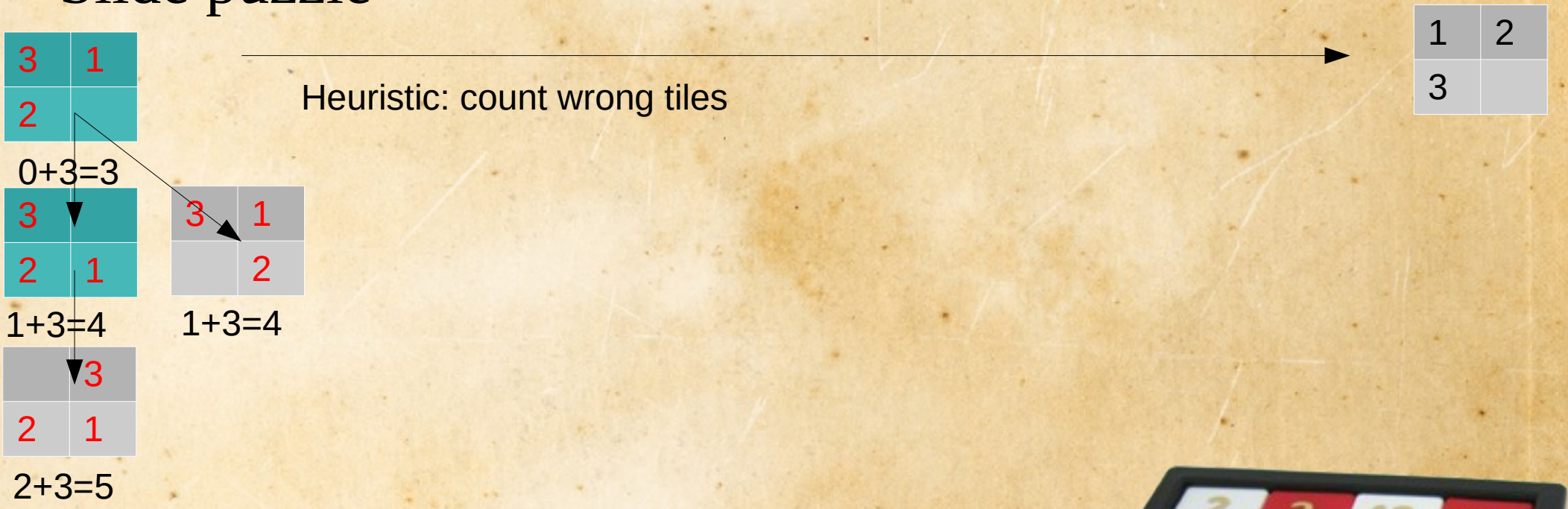
A*

- Slide puzzle



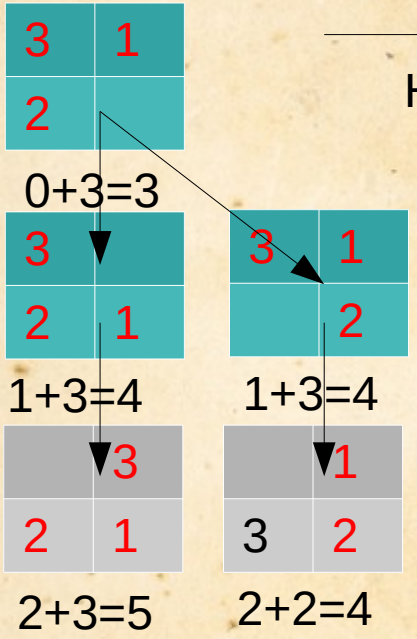
A*

- Slide puzzle



A*

- Slide puzzle

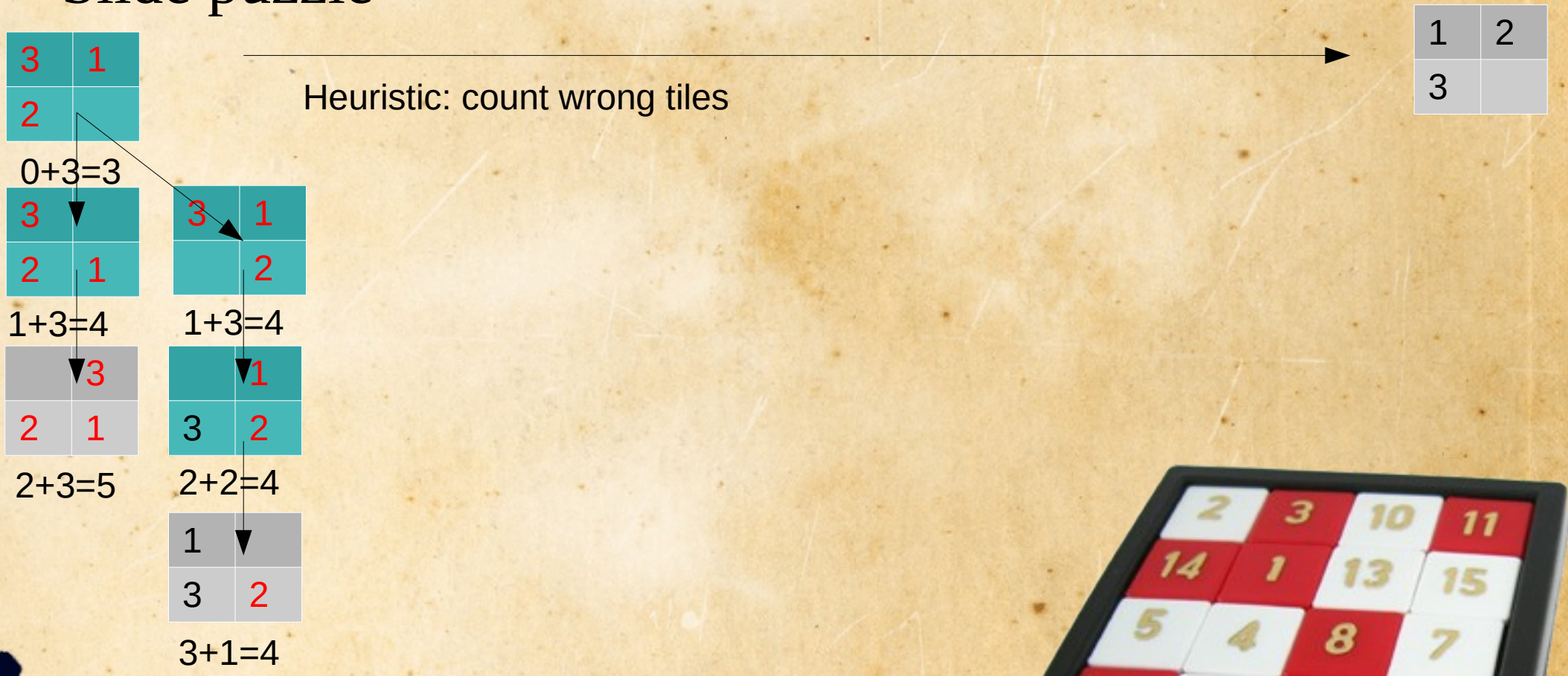


Heuristic: count wrong tiles

1	2
3	

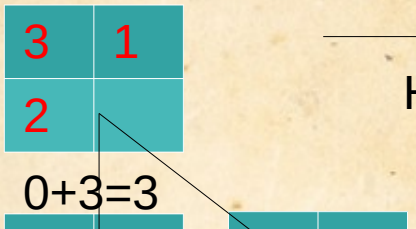
A*

- Slide puzzle

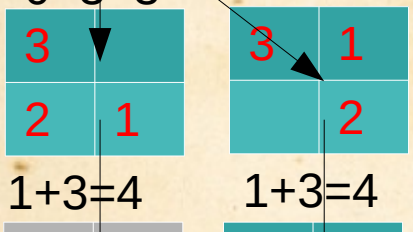


A*

- Slide puzzle

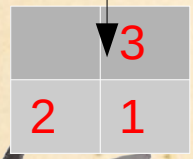


0+3=3

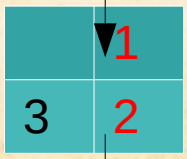


1+3=4

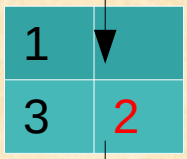
1+3=4



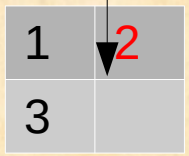
2+3=5



2+2=4



3+1=4



4+0=4

Heuristic: count wrong tiles

A*

- Slide puzzle

3	1
2	

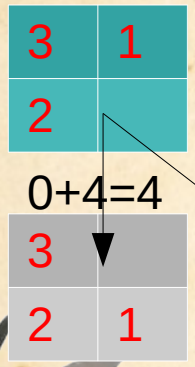
0+4=4

Heuristic: sum of tile distances

1	2
3	

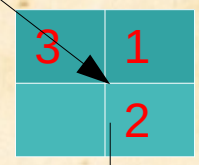
A*

- Slide puzzle

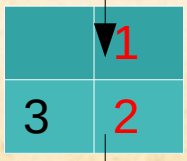


0+4=4

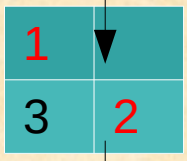
1+5=6



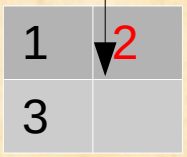
1+3=4



2+2=4



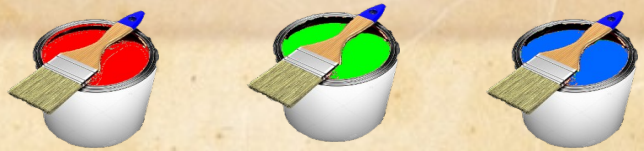
3+1=4



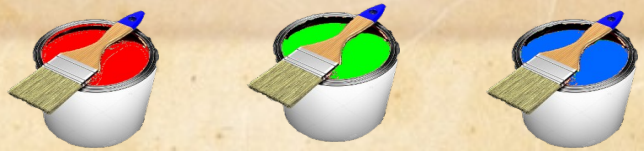
4+0=4

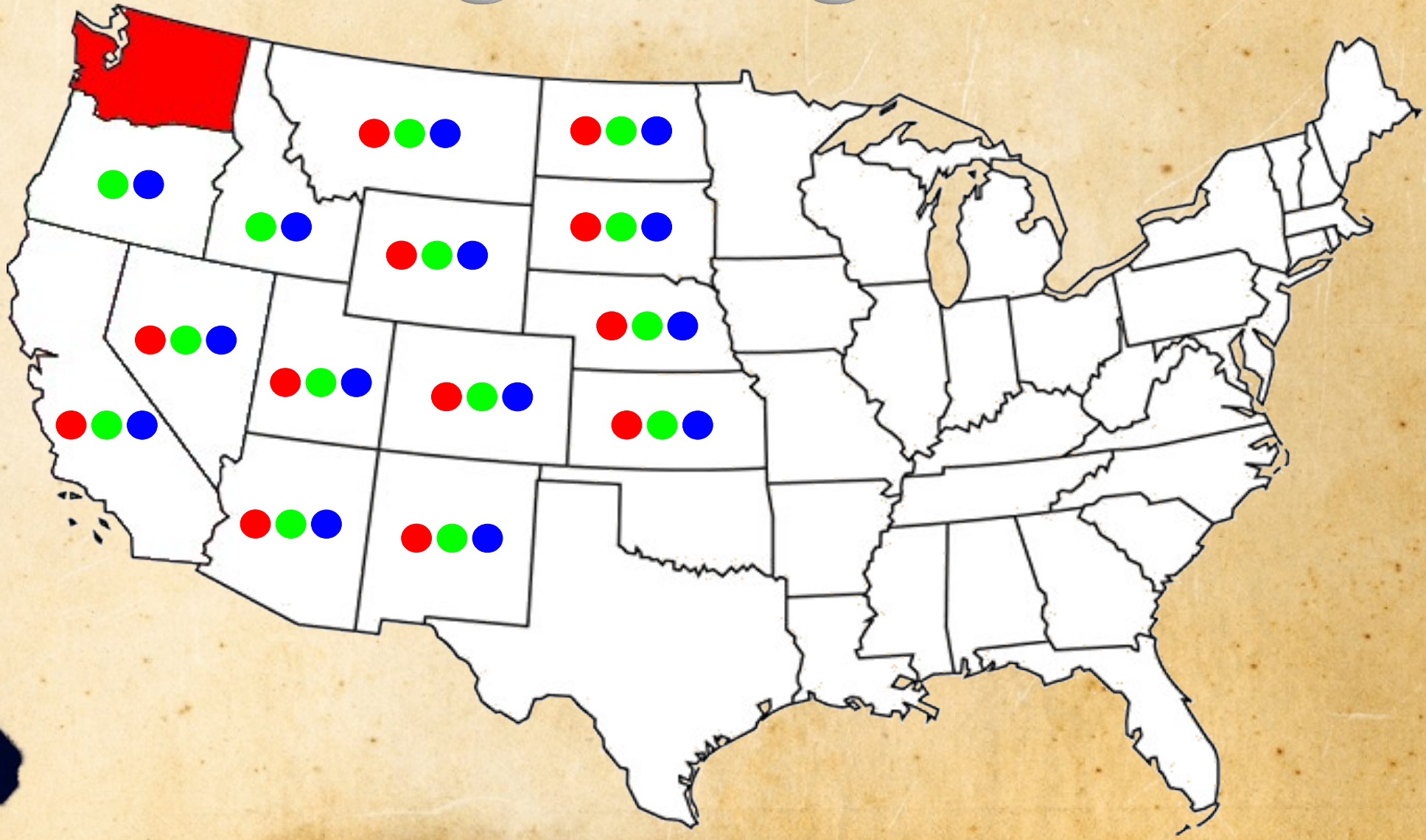
Heuristic: sum of tile distances

Domein specifiek prunen

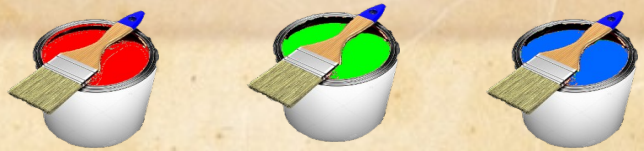


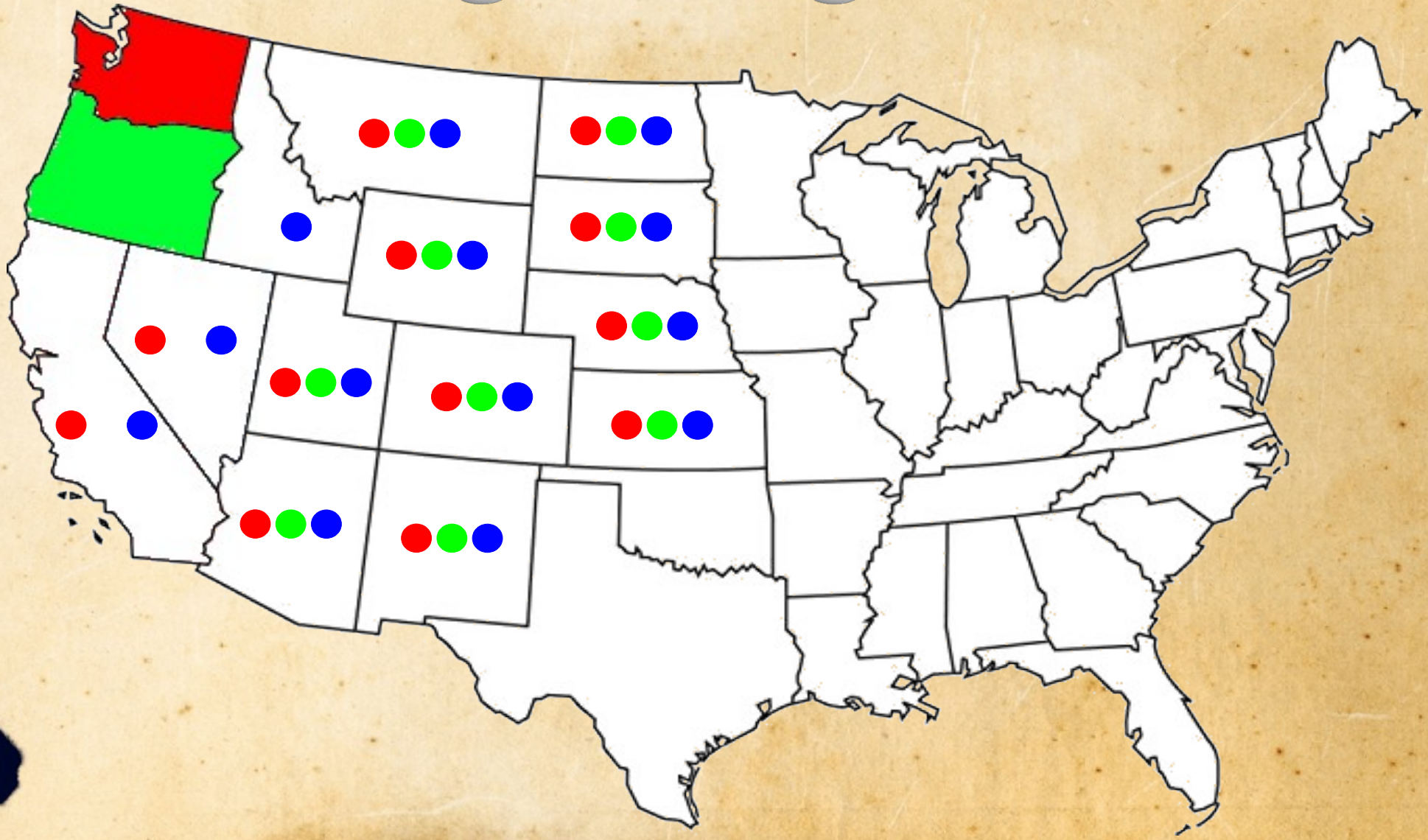
Domein specifiek prunen



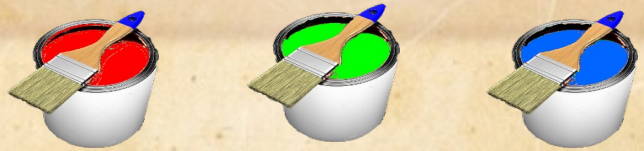


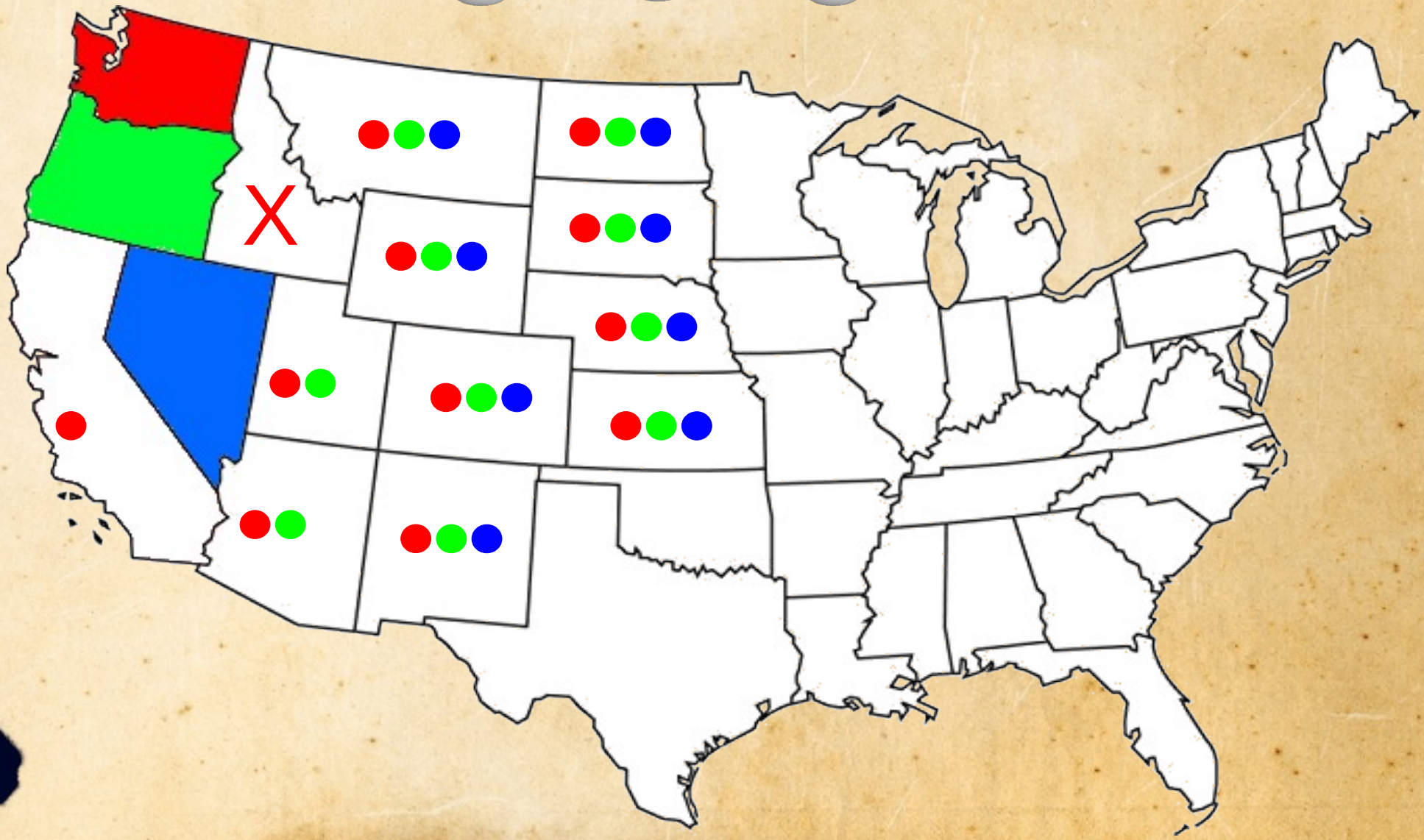
Domein specifiek prunen



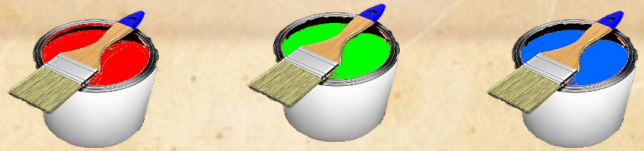


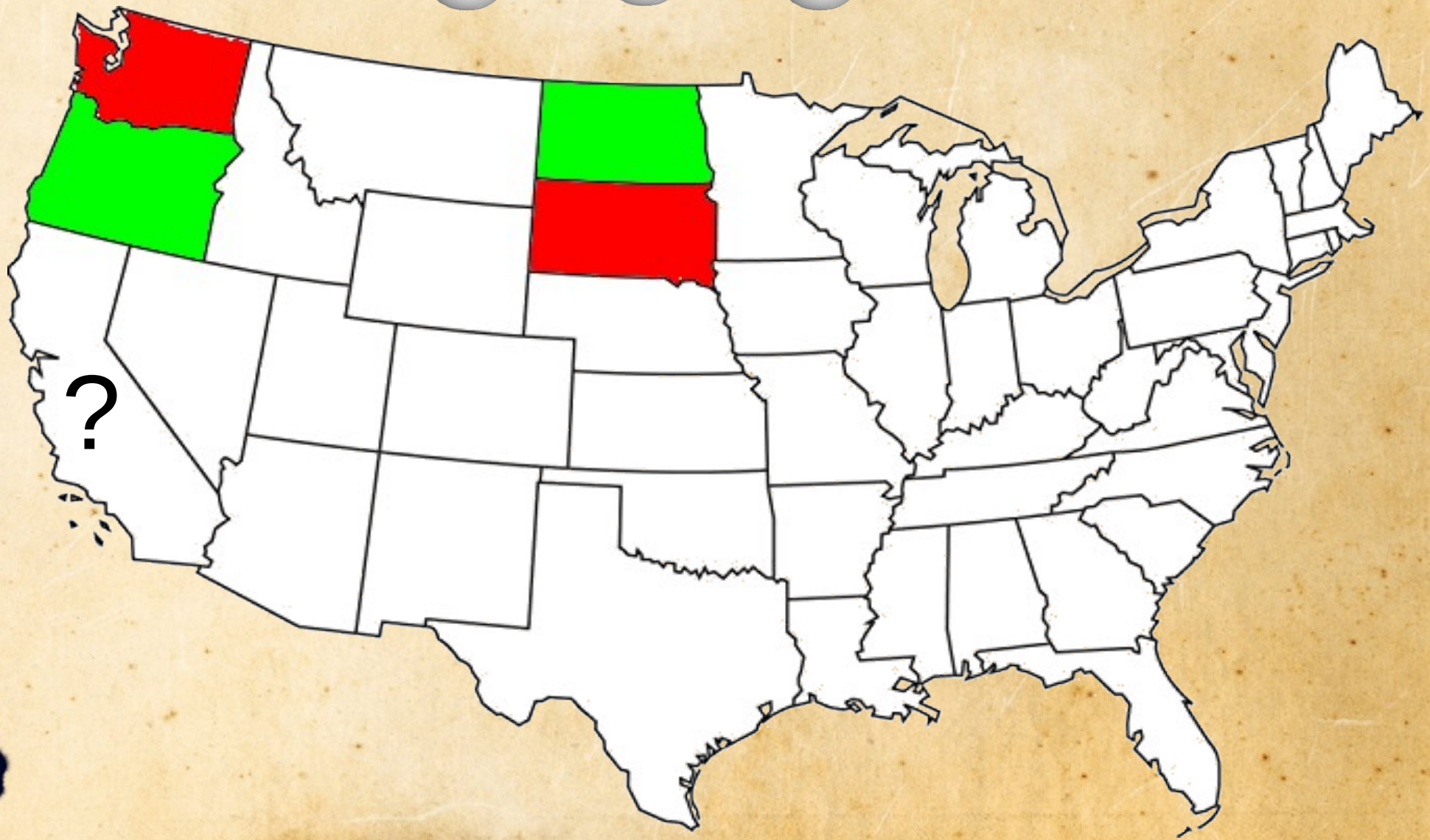
Domein specifiek prunen



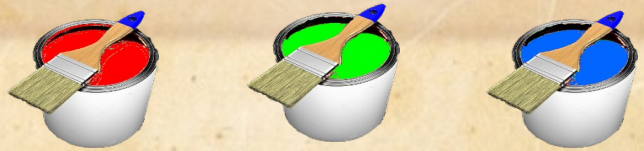


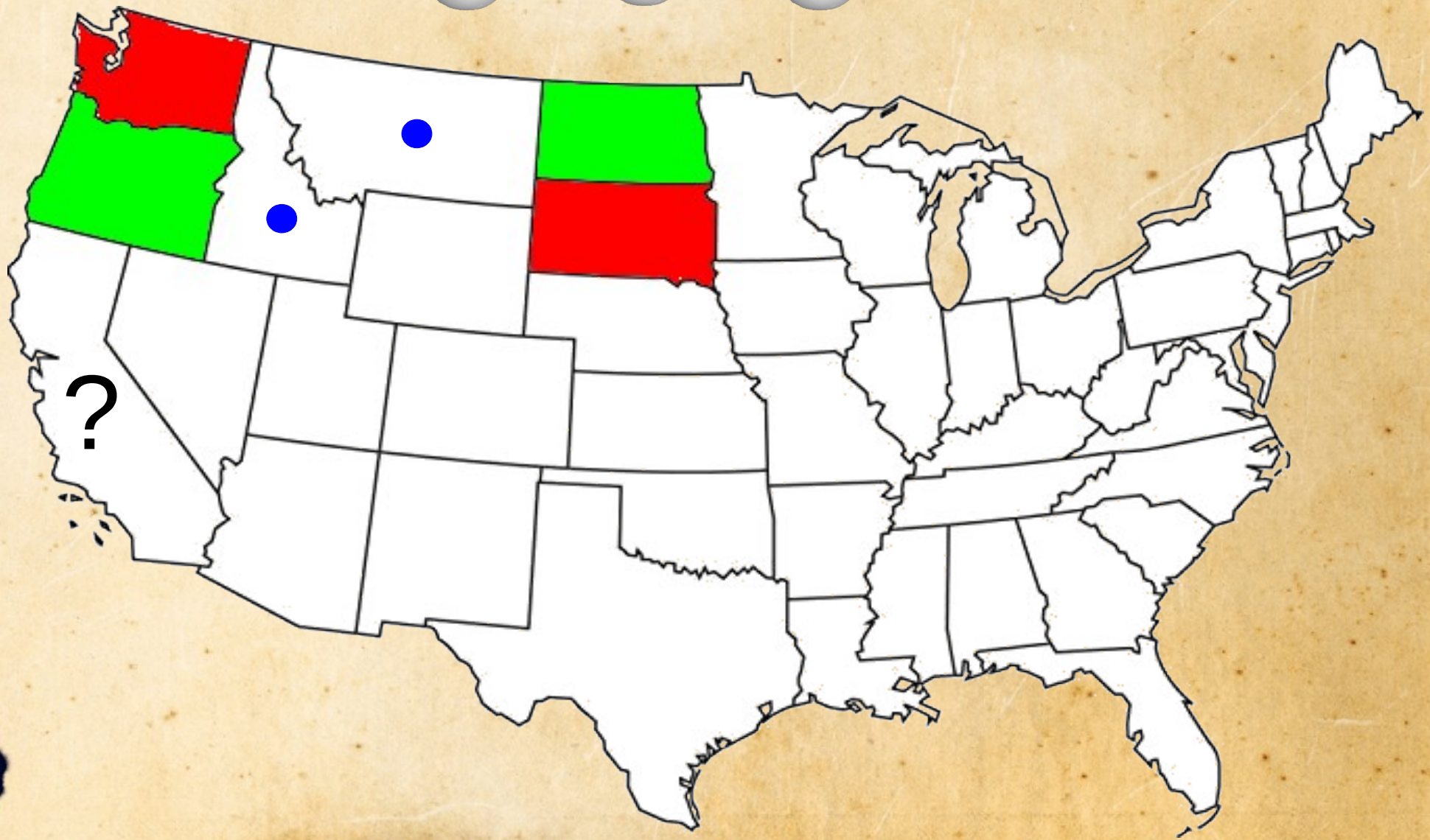
Domein specifiek prunen



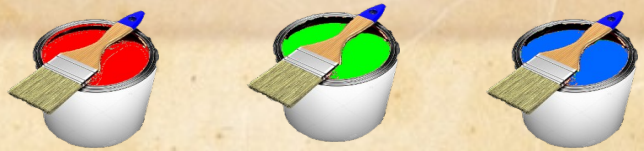


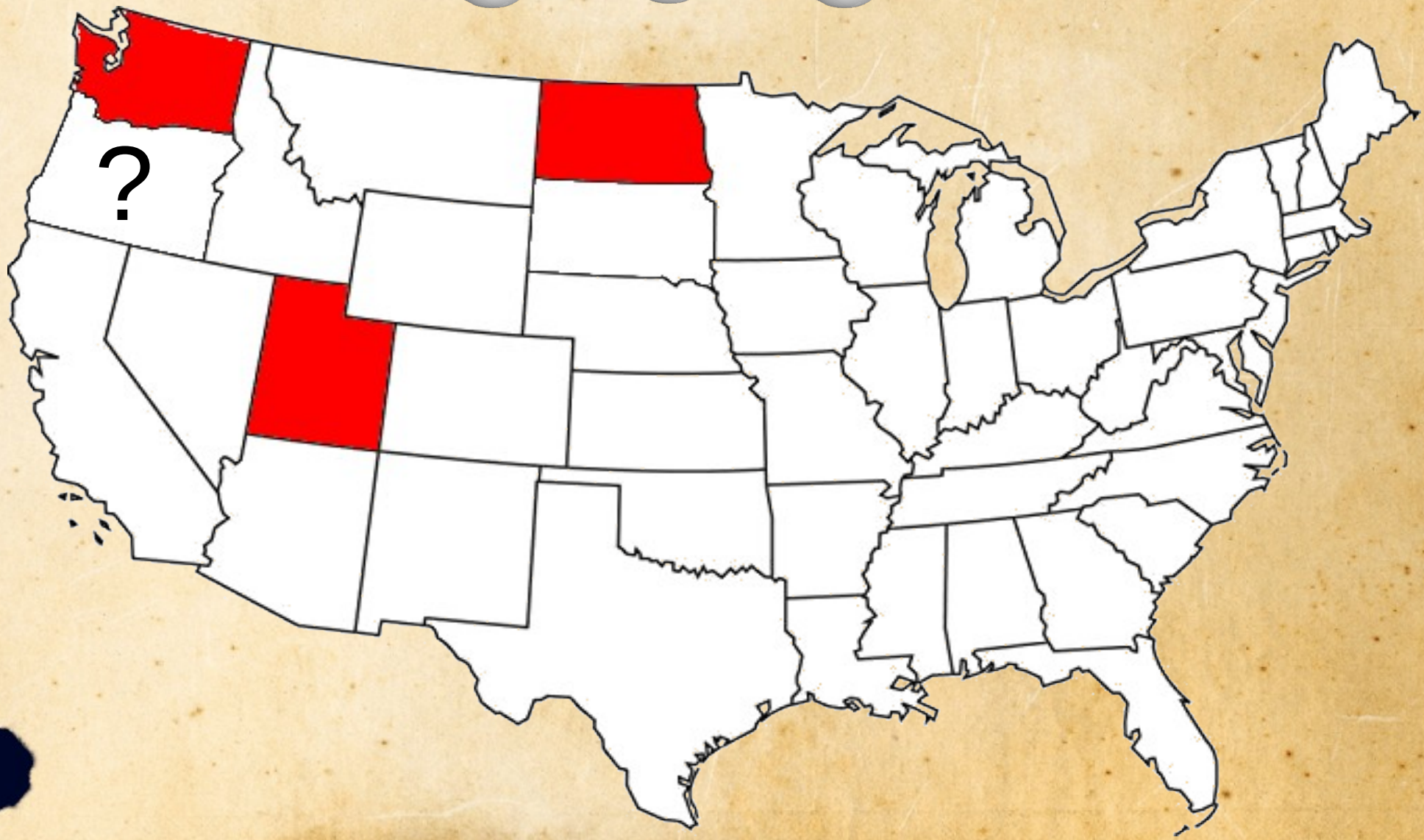
Domein specifiek prunen



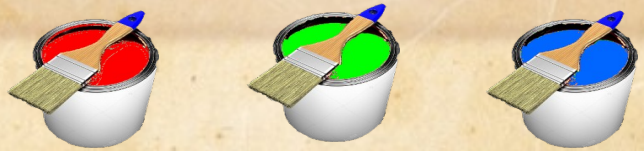


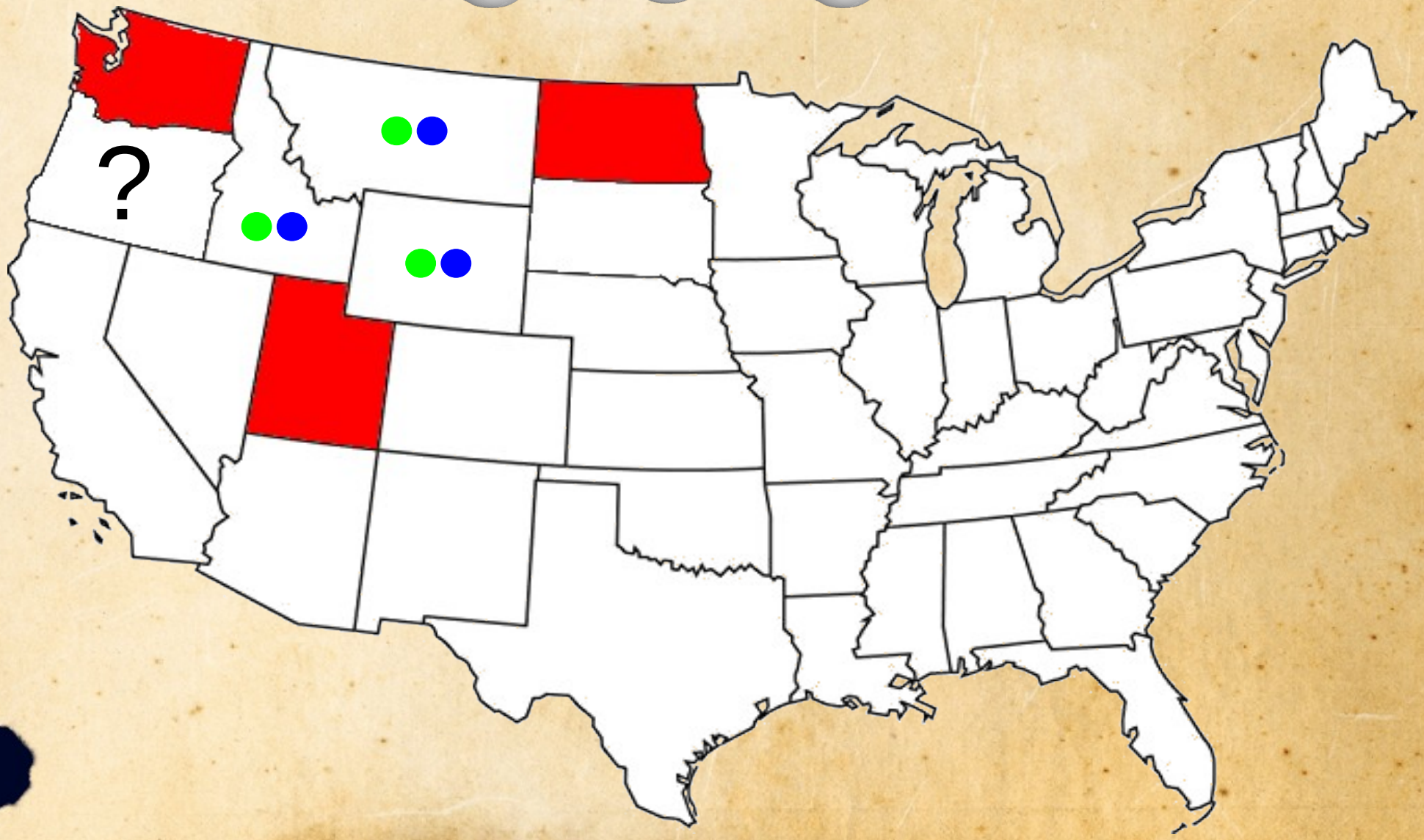
Domein specifiek prunen



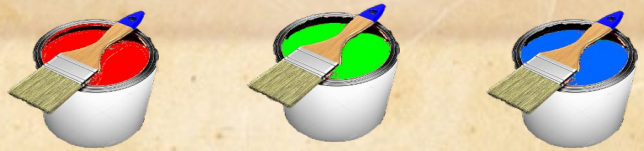


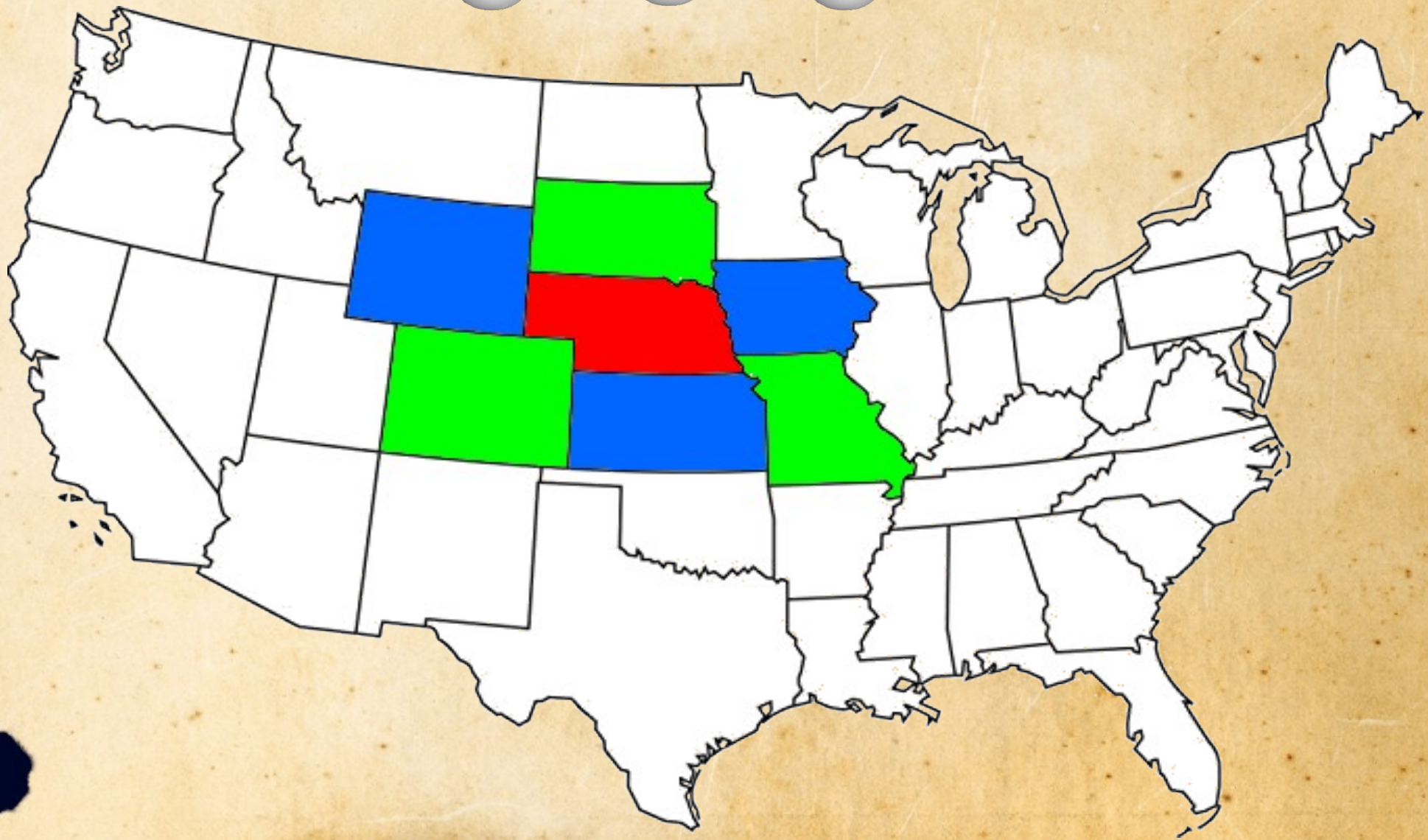
Domein specifiek prunen



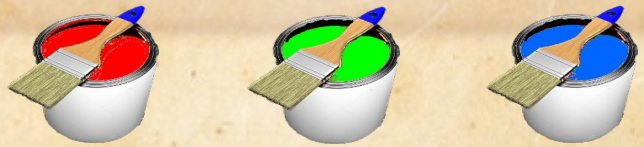


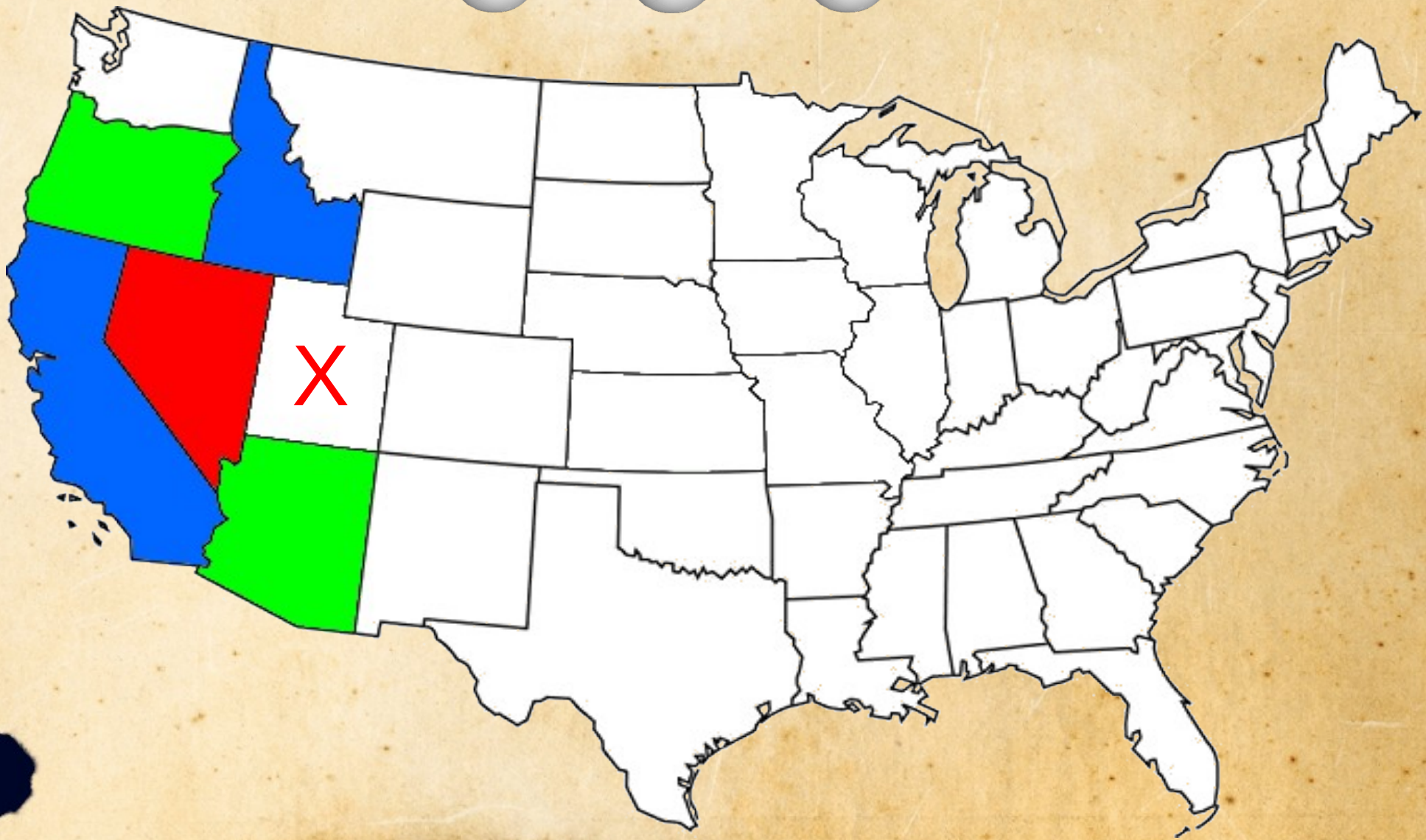
Domein specifiek prunen





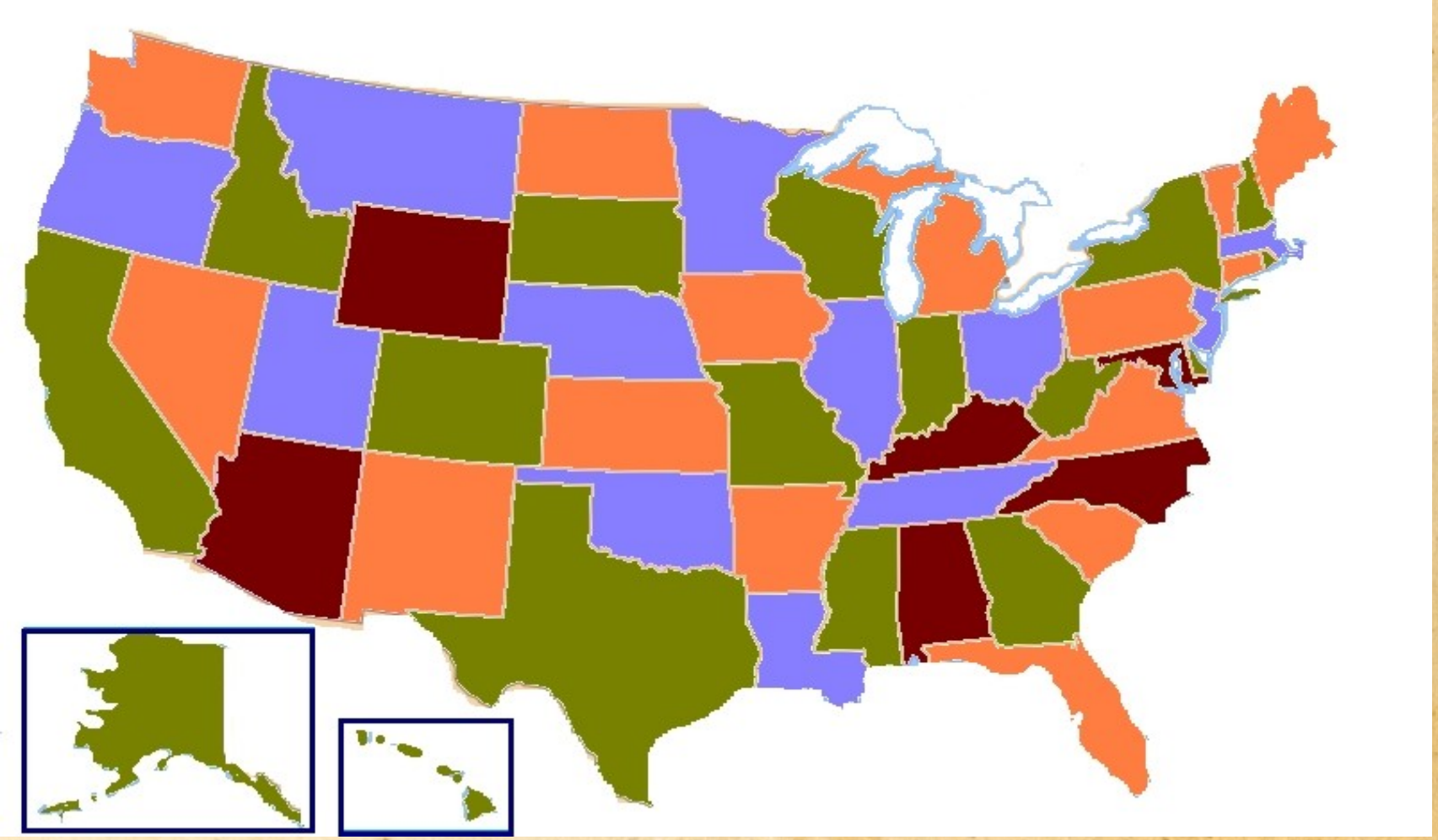
Domein specifiek prunen





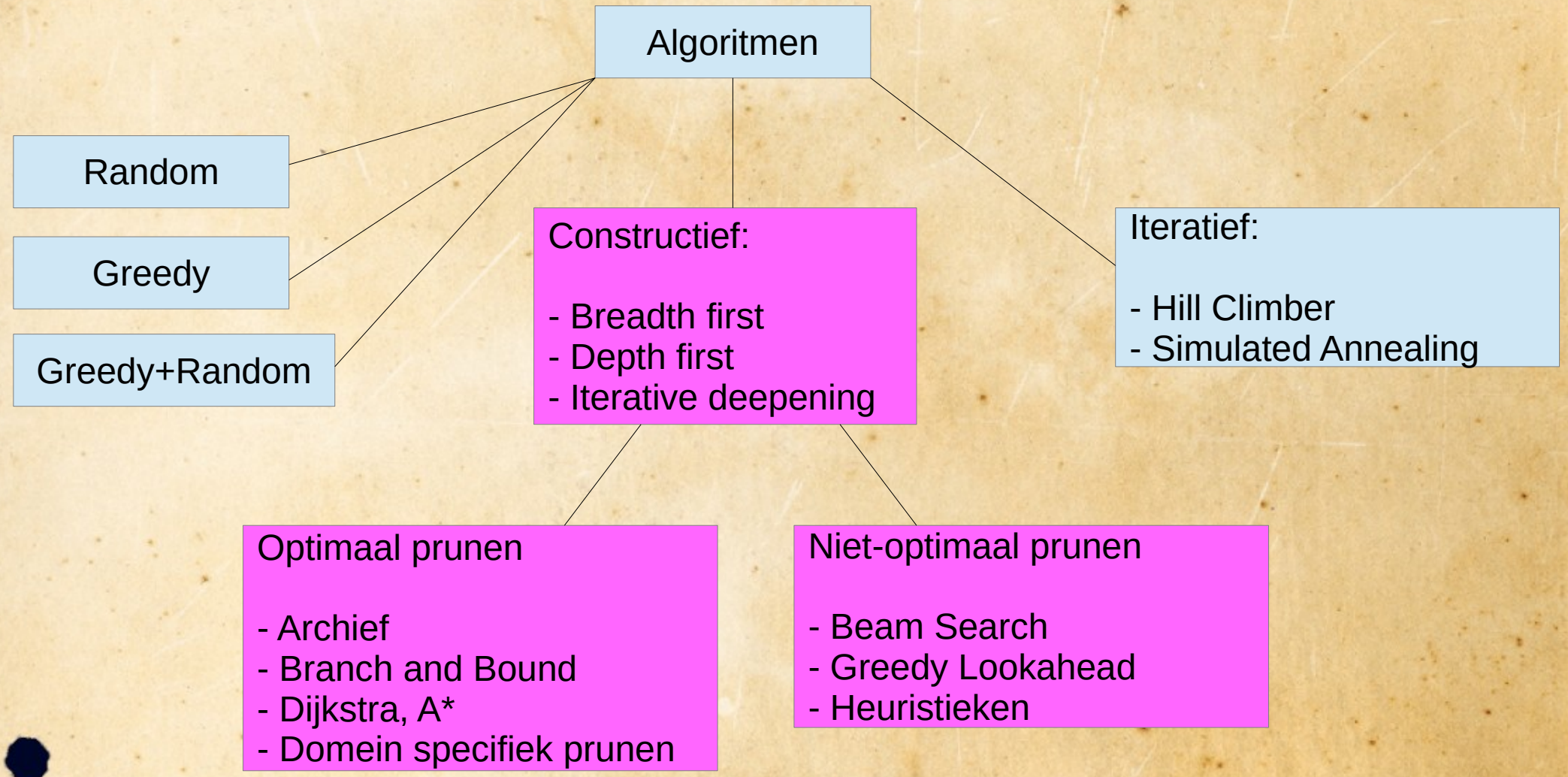
Domein specifiek prunen

Met 4 kleuren kan het wel



https://en.wikipedia.org/wiki/Four_color_theorem

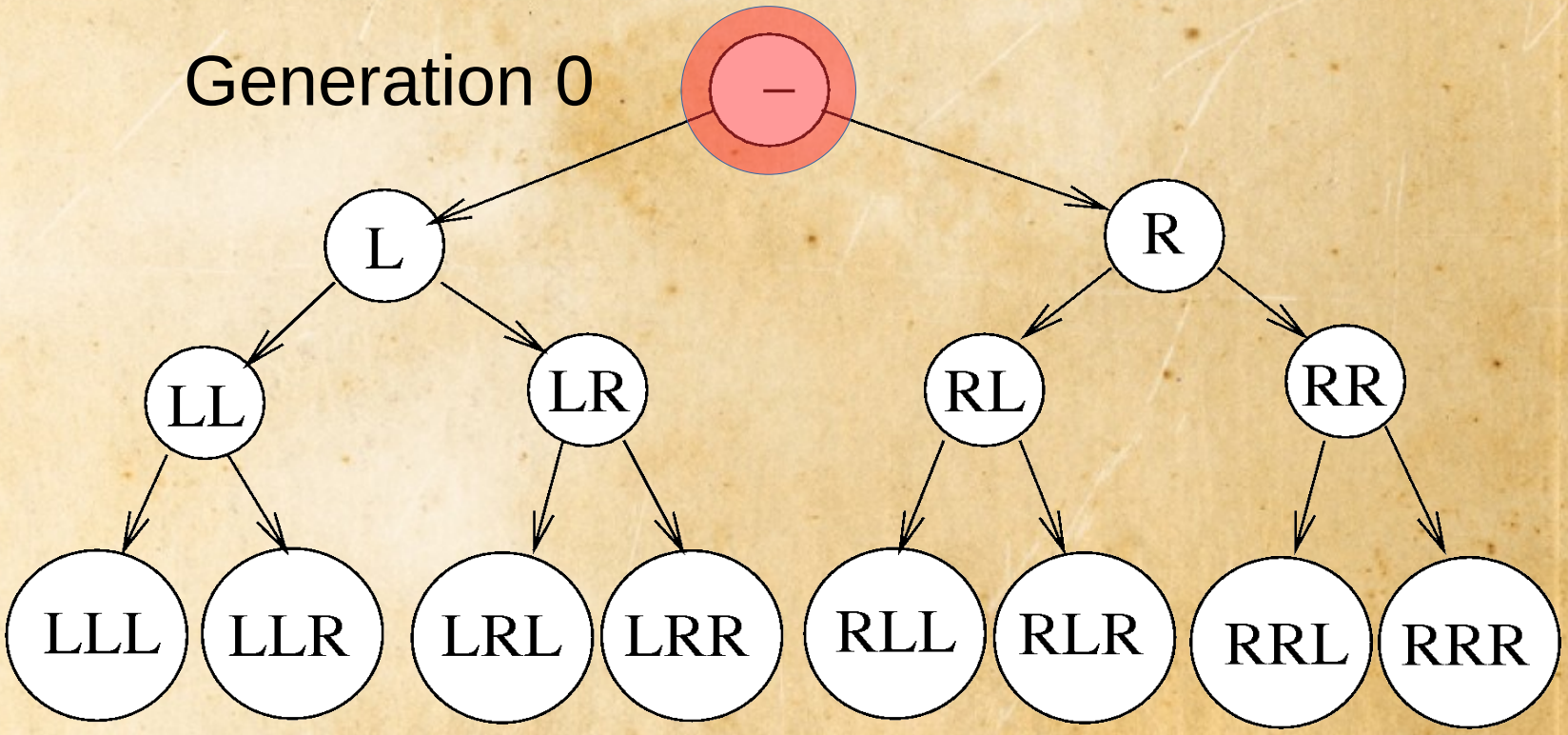
Algoritmen


Algoritmen
Random
Greedy
Greedy+Random
Constructief:
- Breadth first
- Depth first
- Iterative deepening
Iteratief:
- Hill Climber
- Simulated Annealing
Optimaal prunen
- Archief
- Branch and Bound
- Dijkstra, A*
- Domein specifiek prunen
Niet-optimaal prunen
- Beam Search
- Greedy Lookahead
- Heuristieken

Niet-optimaal prunen

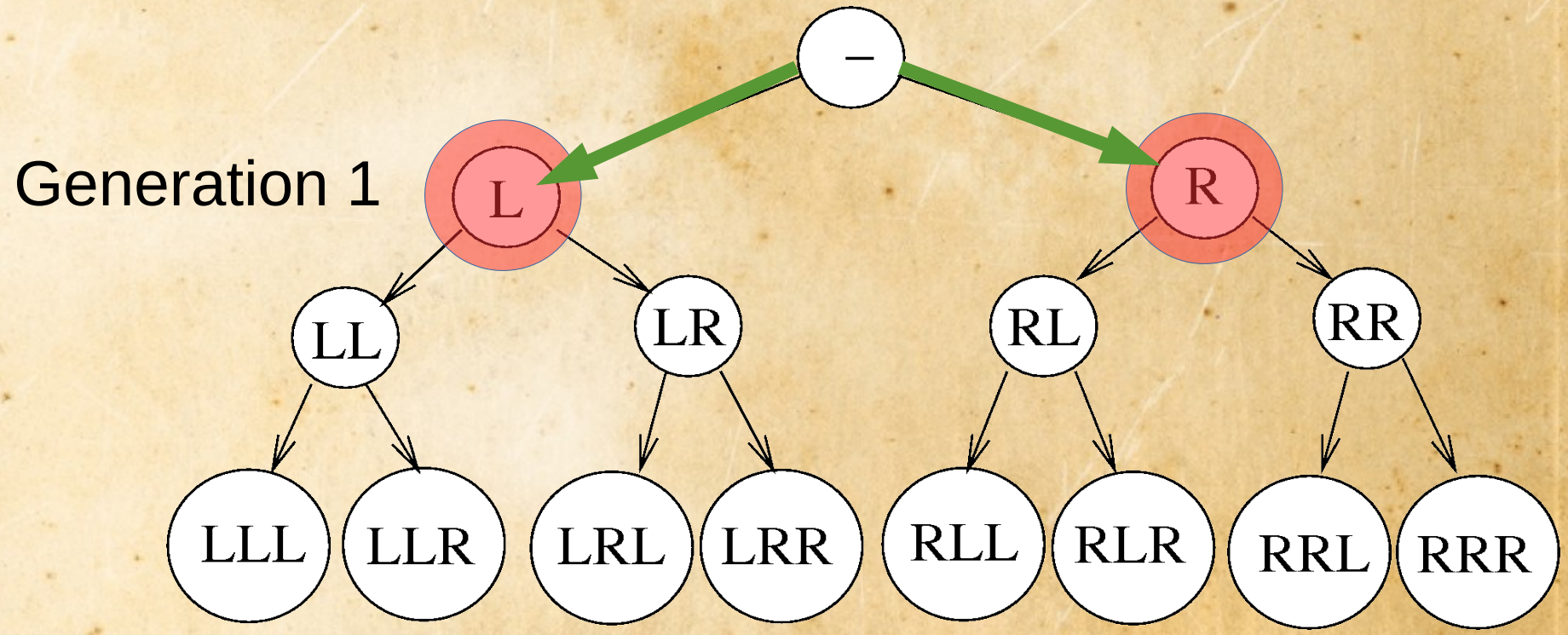
Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



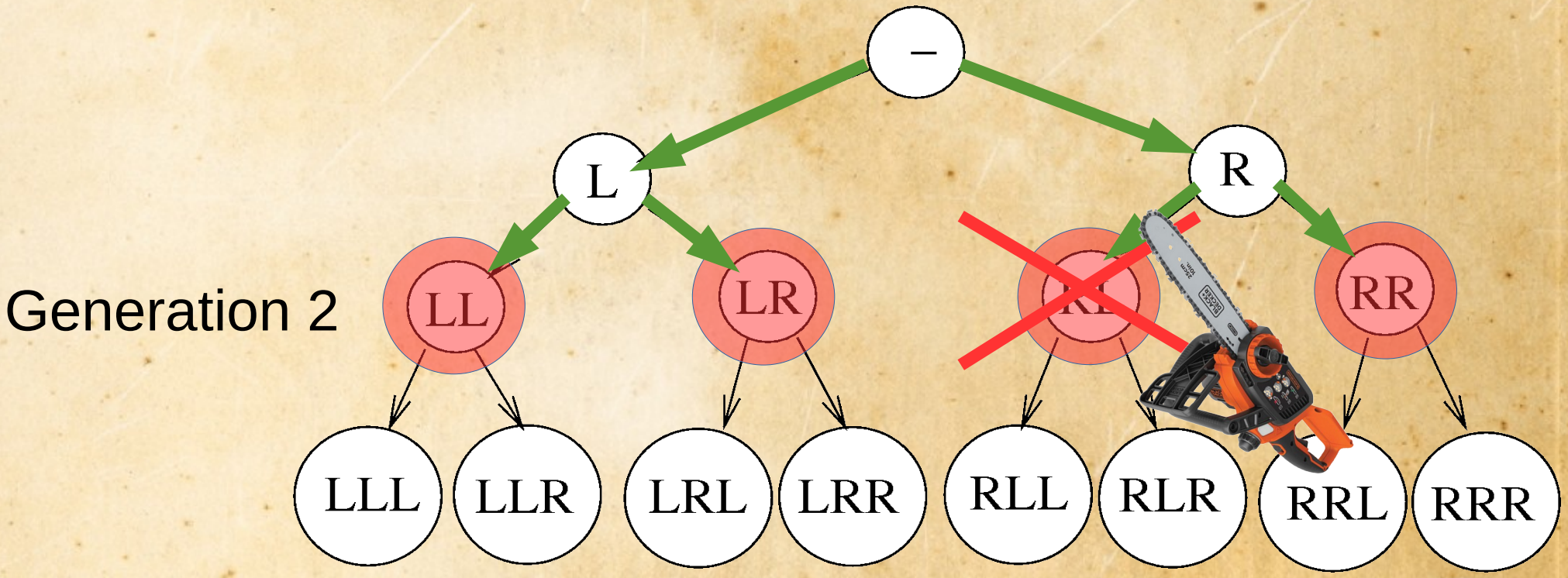
Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



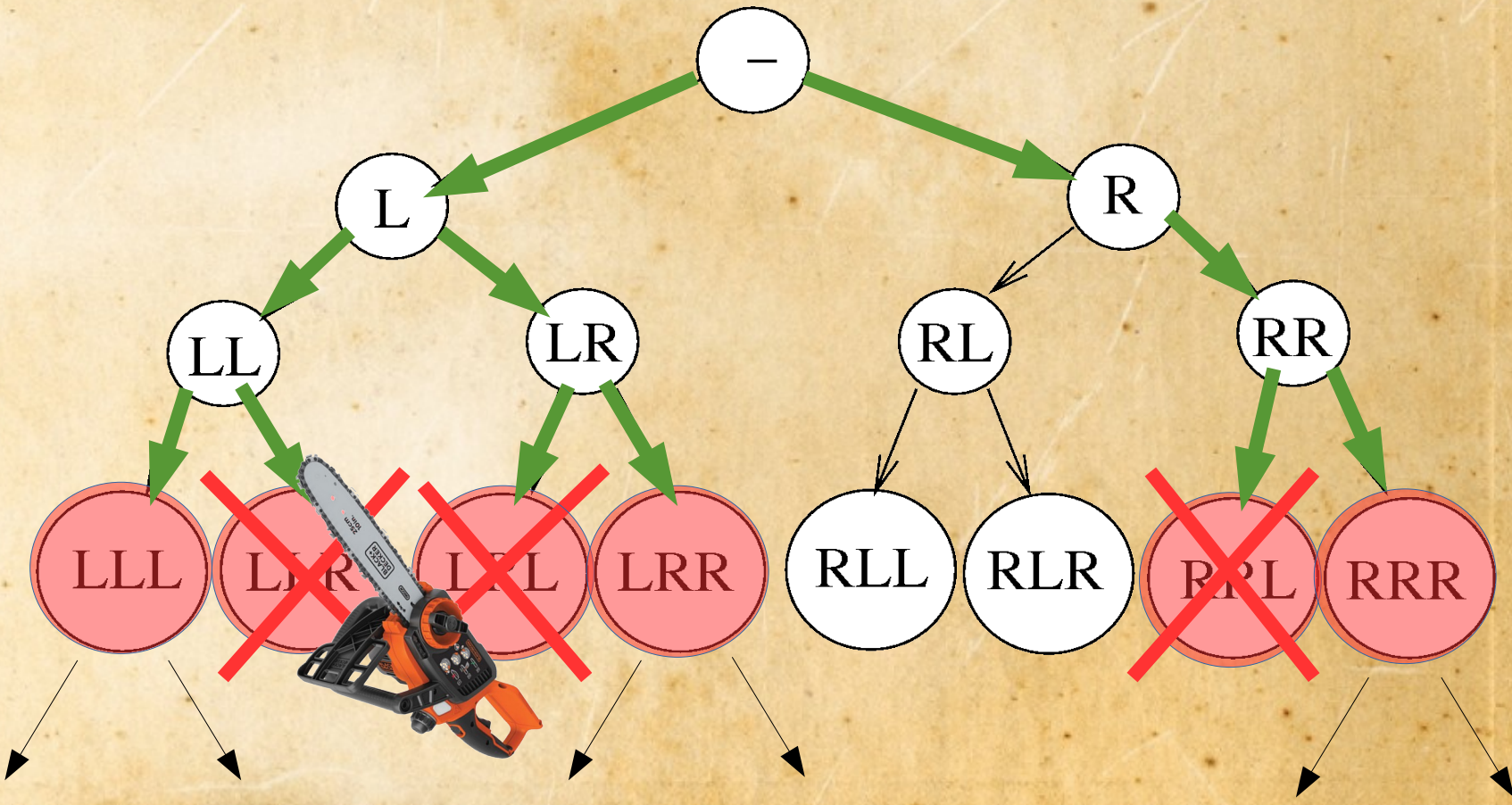
Beam Search

- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



Beam Search

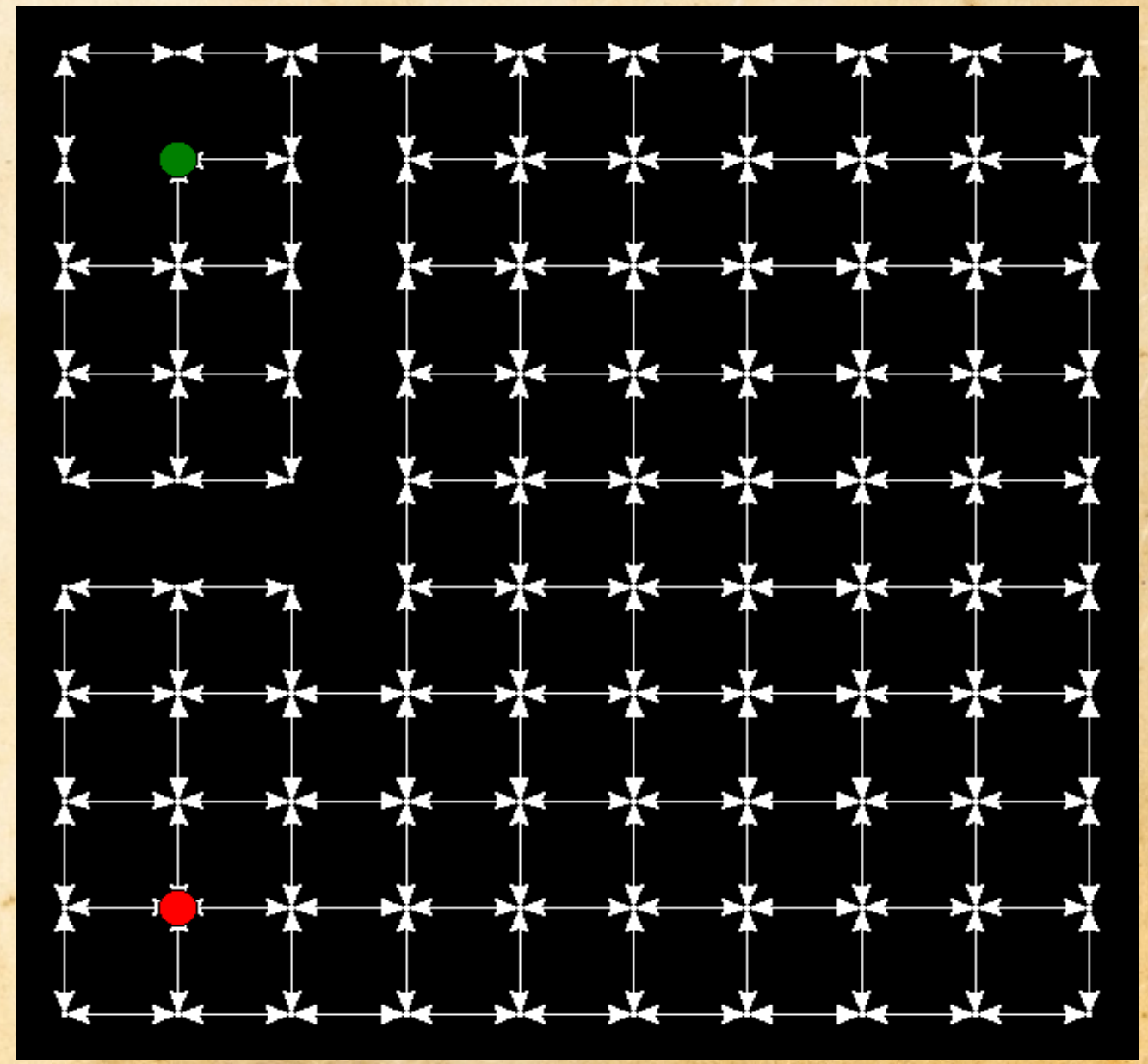
- Breadth First met Beam Search
  - Zoek verder met alleen de **beste** Beam=3 states



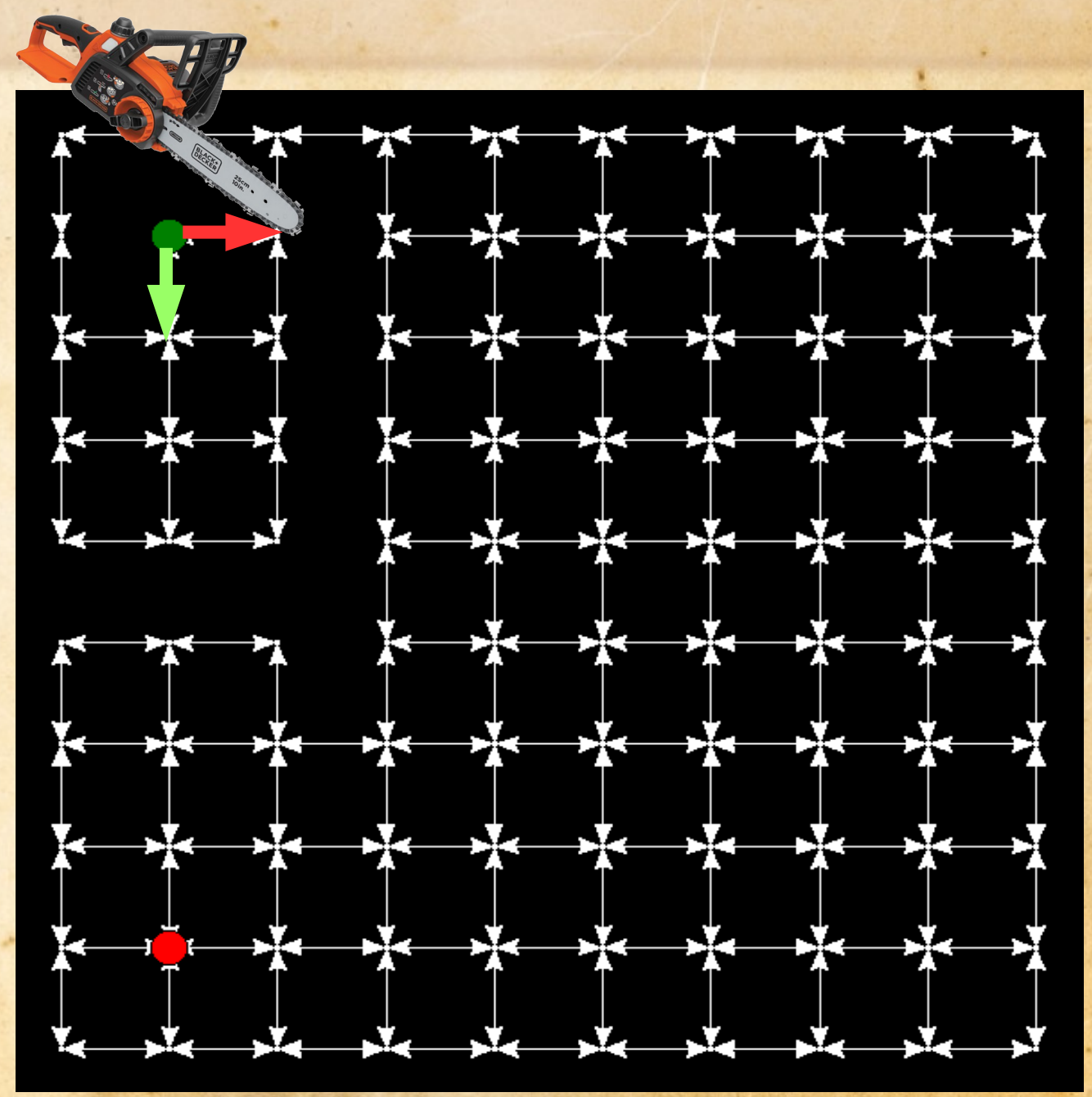
Beam Search

- Hoe groot kiezen we de Beam?
  - Beam=1 state (Greedy)
    - Super snel maar waarschijnlijk slechte oplossing
  - Beam=∞ states (Breadth first)
    - Super traag en geheugen problemen maar optimale oplossing

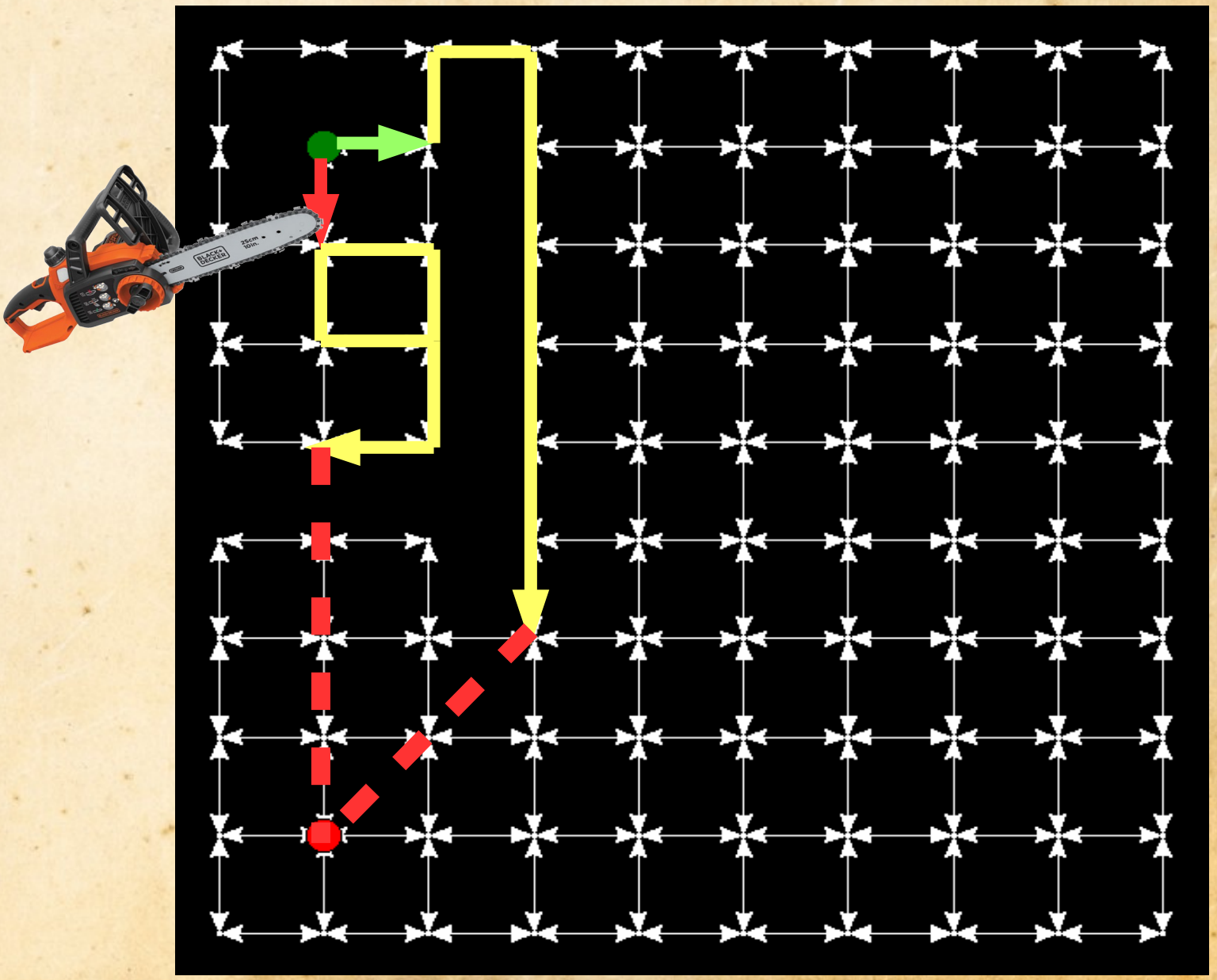
Greedy without look-ahead



Greedy without look-ahead



Greedy with 8 step look-ahead



Greedy with look-ahead

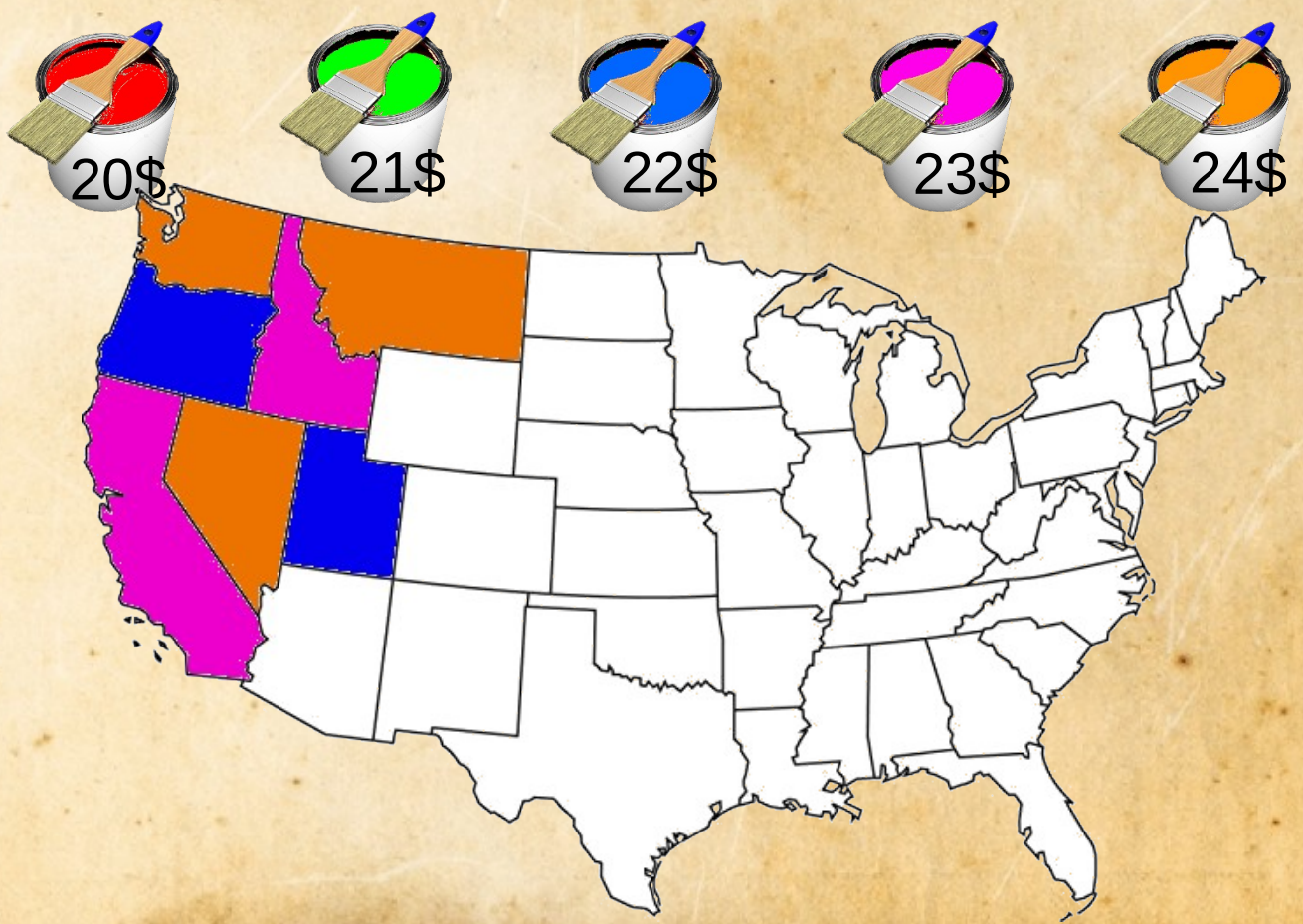
- Hoe ver look-en we ahead?
  - Steps=0 stappen (Greedy)
    - Super snel maar waarschijnlijk slechte oplossing
  - Steps=∞ stappen (Depth first)
    - Super traag maar optimale oplossing

Combinaties?

- Breadth first + Beam search + look-ahead?
- Depth first + pruning?

Combinaties?

- Breadth first + Beam search + look-ahead?
- Depth first + pruning?

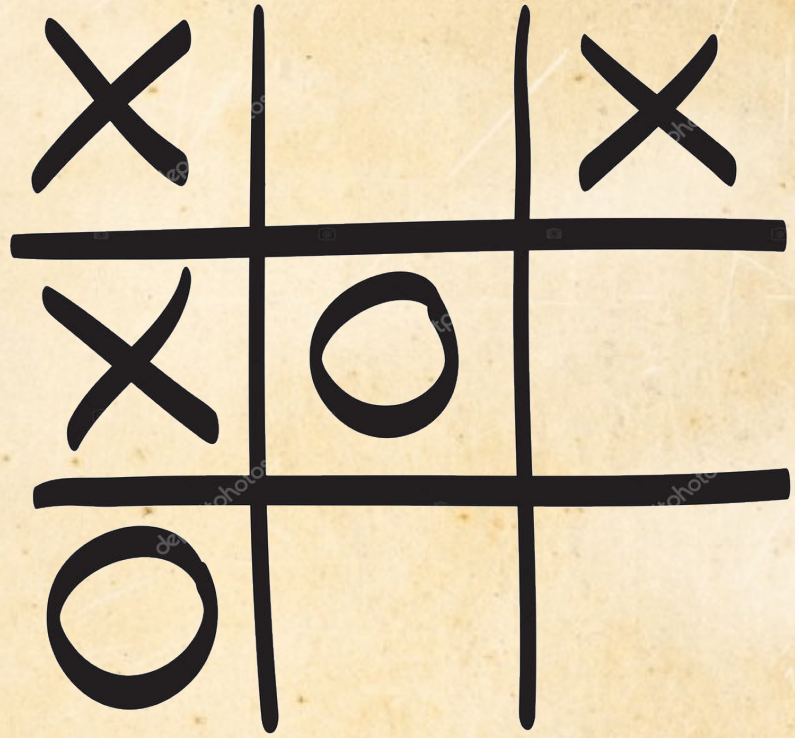


Heuristieken

- Oud Grieks:
  - heurískō: vinden, vergelijken
  - heurèka: ik heb het gevonden
- Wikipedia:
  - Is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect or rational, but which is nevertheless sufficient for reaching an immediate, short-term goal.
- Bas:
  - Praktische vuistregel waarmee je (sneller) tot een (betere maar) (misschien) niet optimale oplossingen komt.

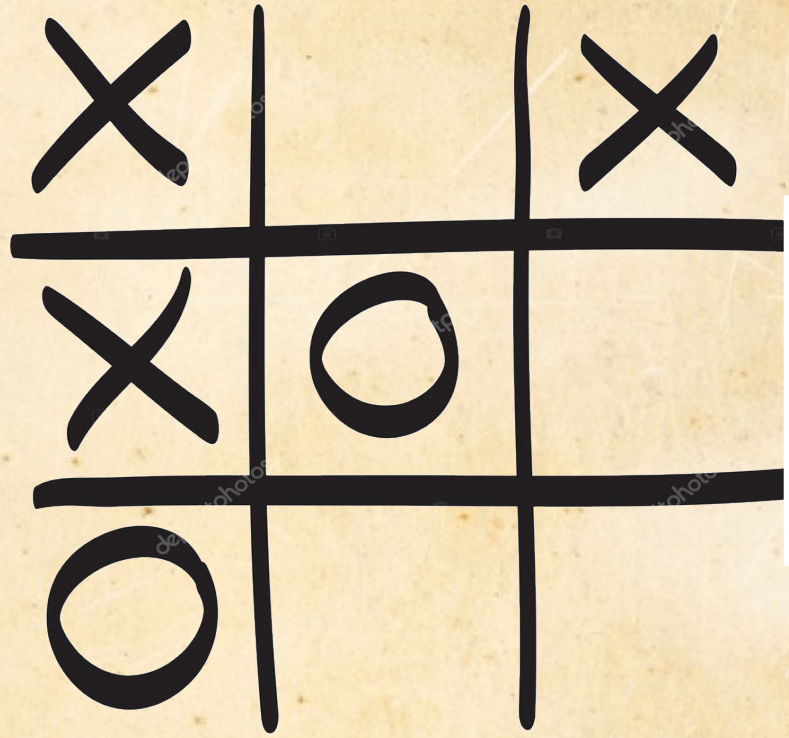
Heuristieken

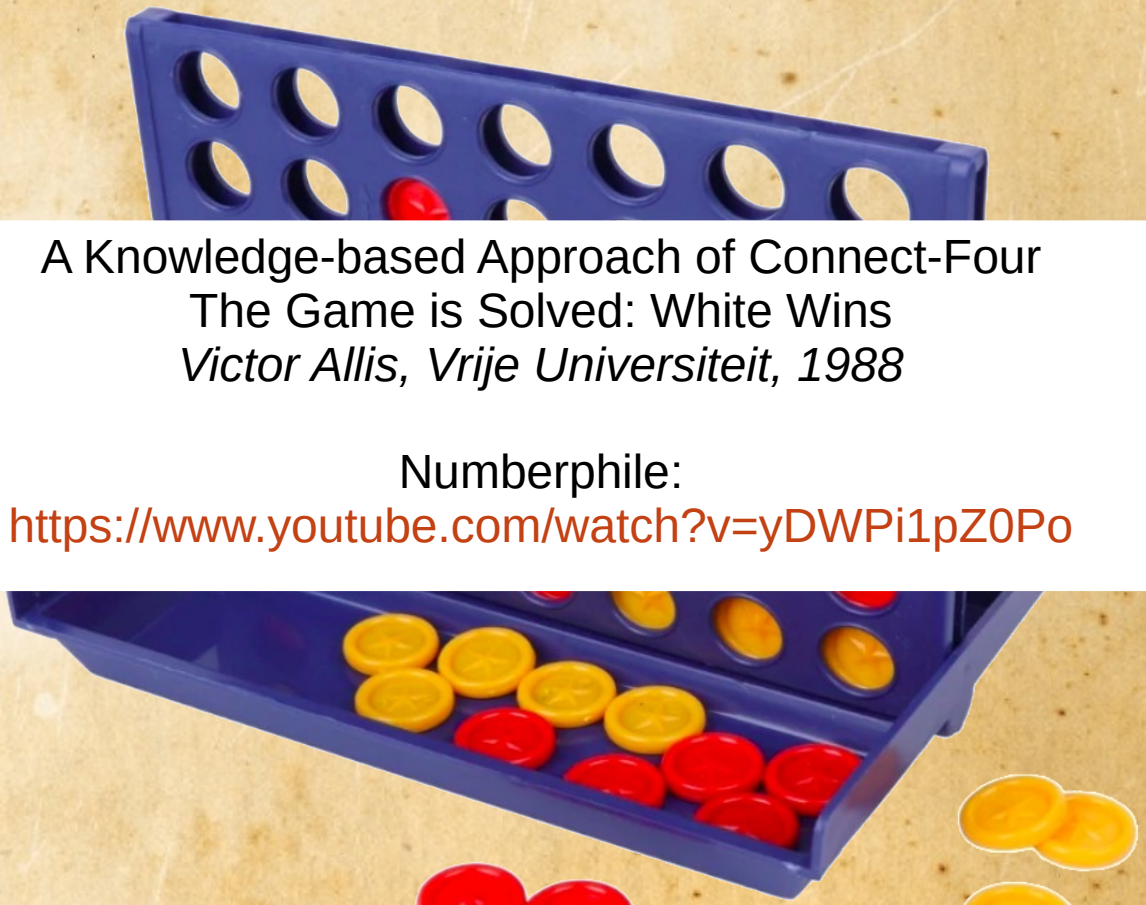
- Spelletjes: eerste zet doe ik in het midden



Heuristieken

- Spelletjes: eerste zet doe ik in het midden





A Knowledge-based Approach of Connect-Four
The Game is Solved: White Wins
Victor Allis, Vrije Universiteit, 1988

Numberphile:
https://www.youtube.com/watch?v=yDWPi1pZ0Po

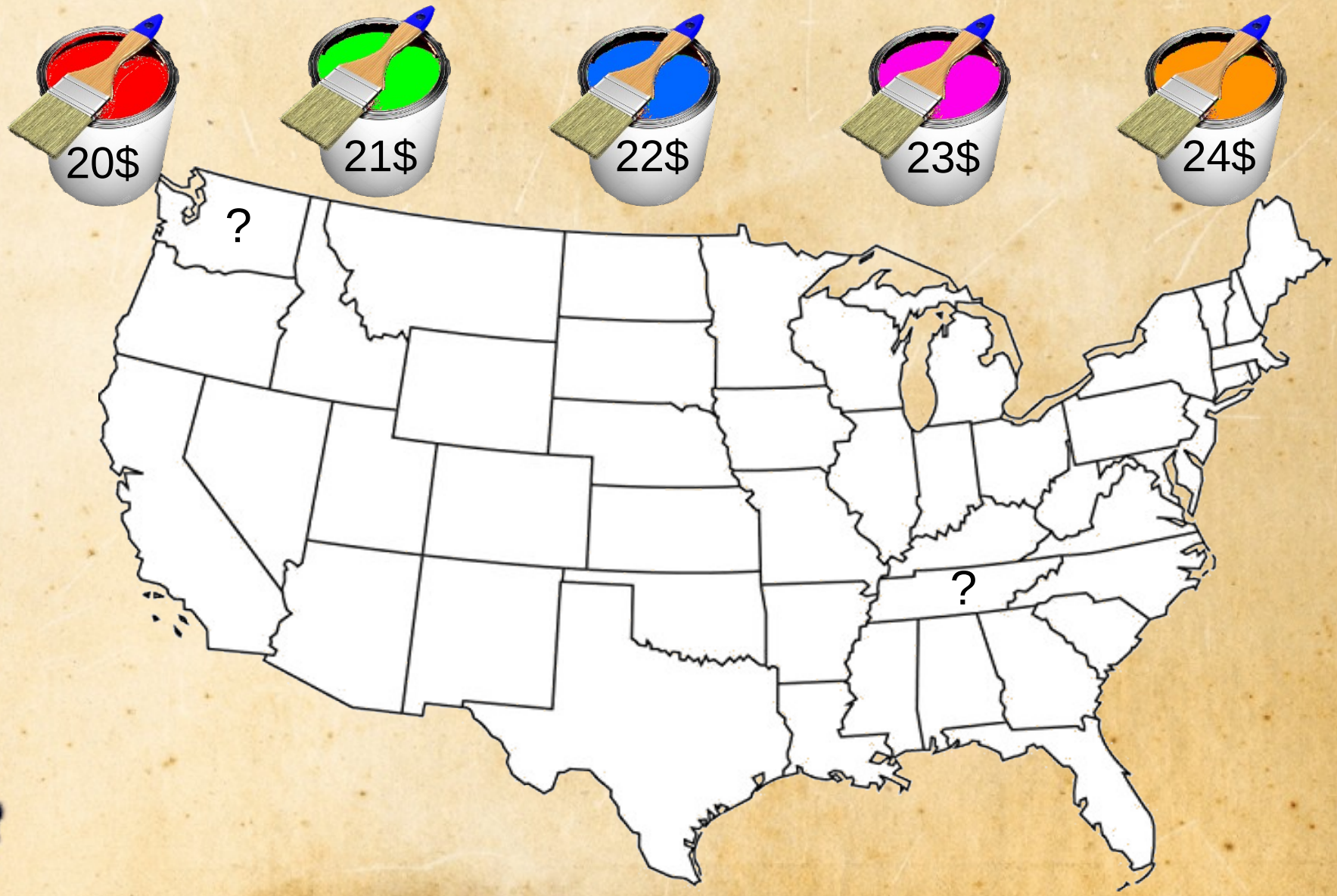
Heuristieken

- Spelletjes: hoe goed is een positie?

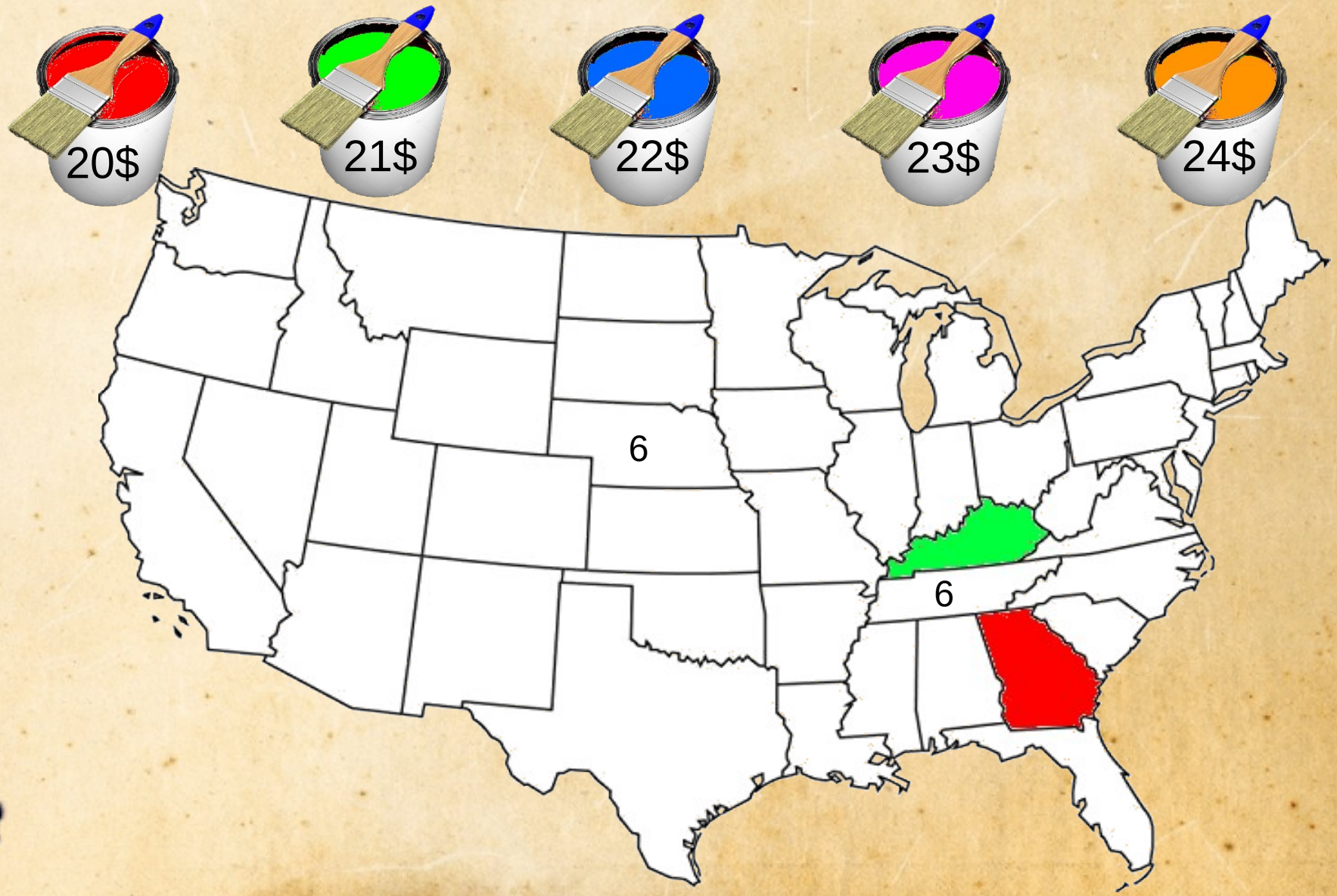
Heuristieken

- Inpakken: grootste spullen eerst

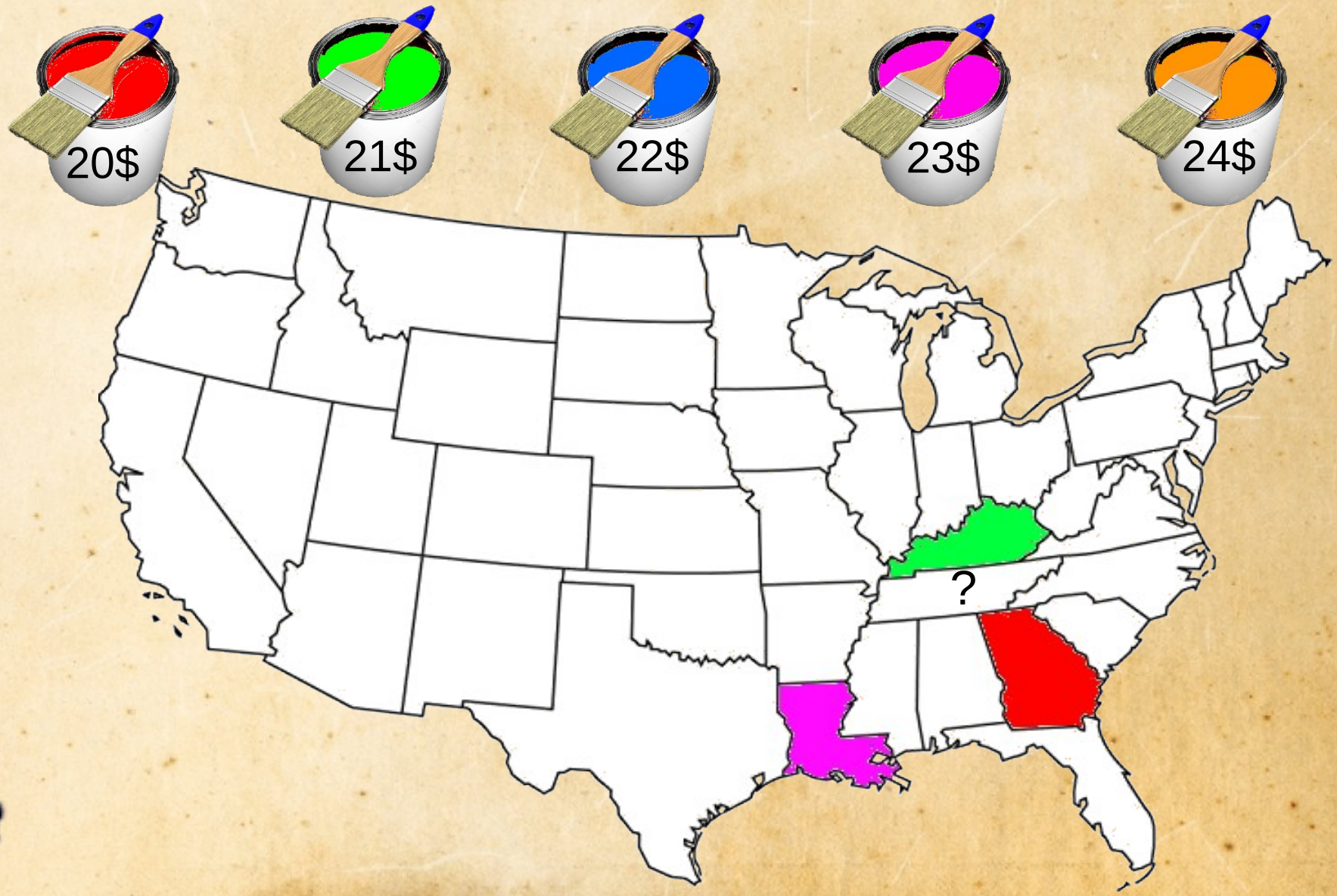
Kaart kleuren



Kaart kleuren



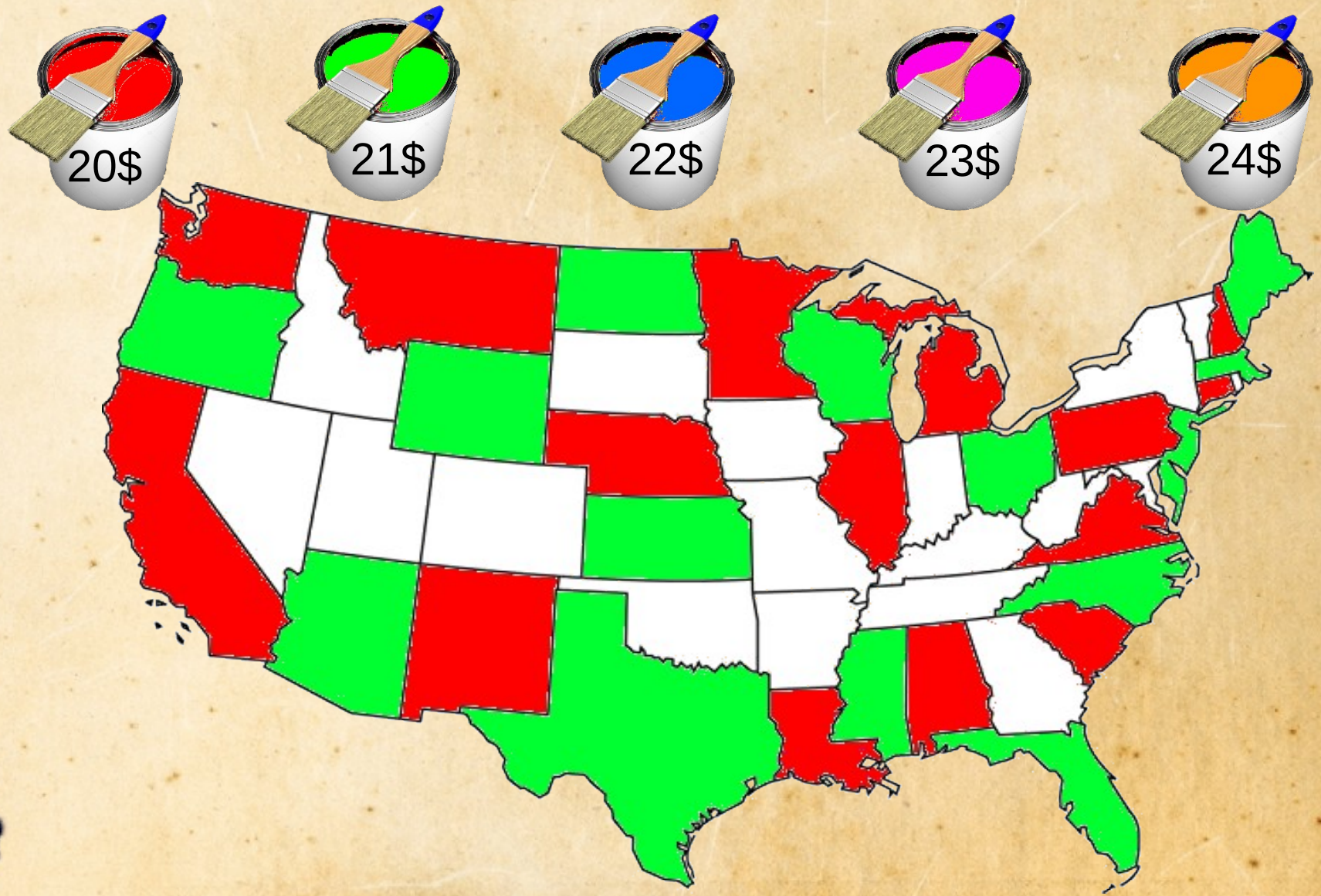
Kaart kleuren



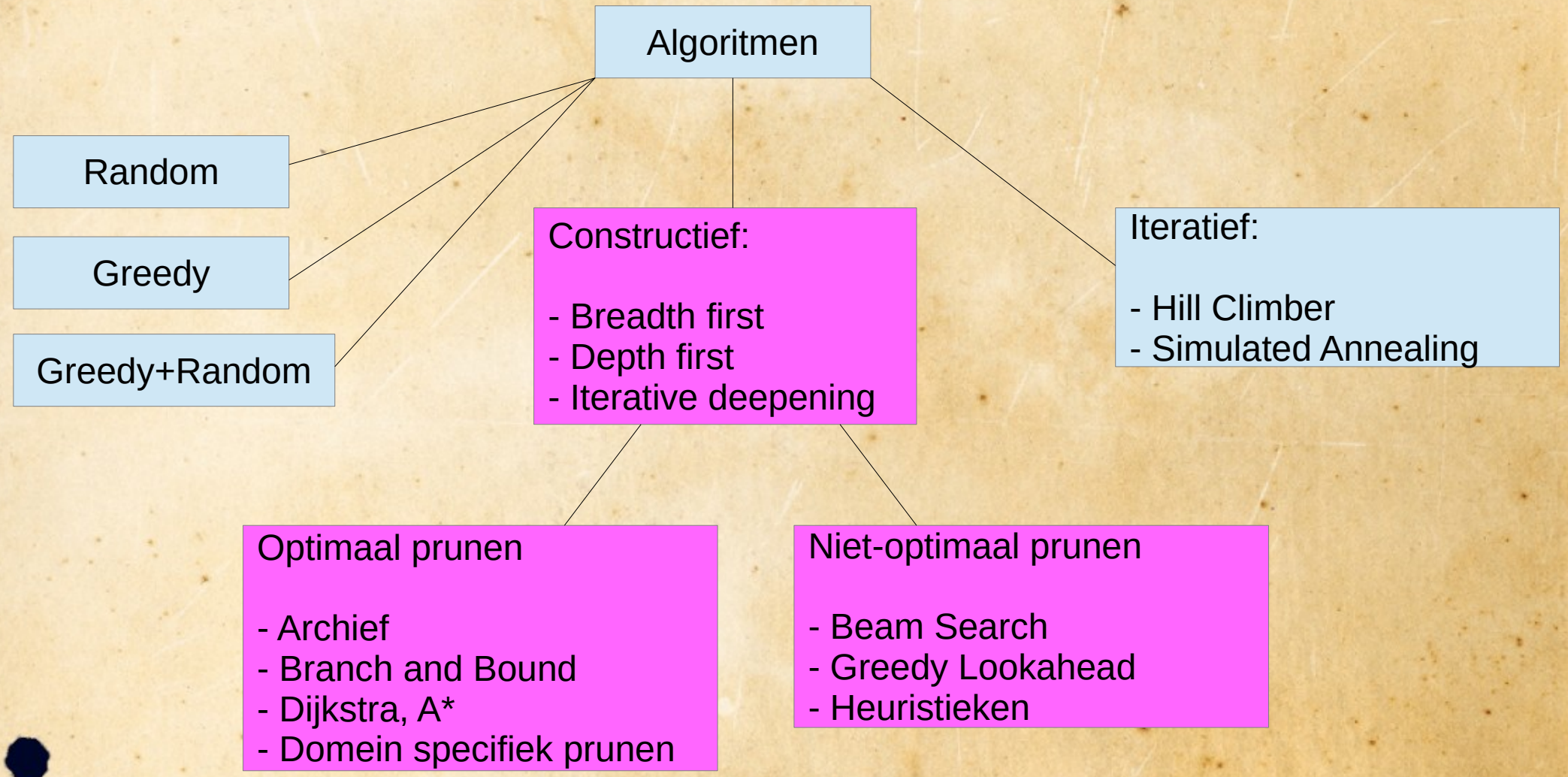
Heuristiek

- Most Constraint Variable (MCV)
  - Meeste constraints met andere variabele
  - Minst aantal mogelijke waarden over
- Least Constraining Value (LCV)

Heuristiek: goedkoopste eerst?



Algoritmen


Algoritmen
Random
Greedy
Greedy+Random
Constructief:
- Breadth first
- Depth first
- Iterative deepening
Iteratief:
- Hill Climber
- Simulated Annealing
Optimaal prunen
- Archief
- Branch and Bound
- Dijkstra, A*
- Domein specifiek prunen
Niet-optimaal prunen
- Beam Search
- Greedy Lookahead
- Heuristieken

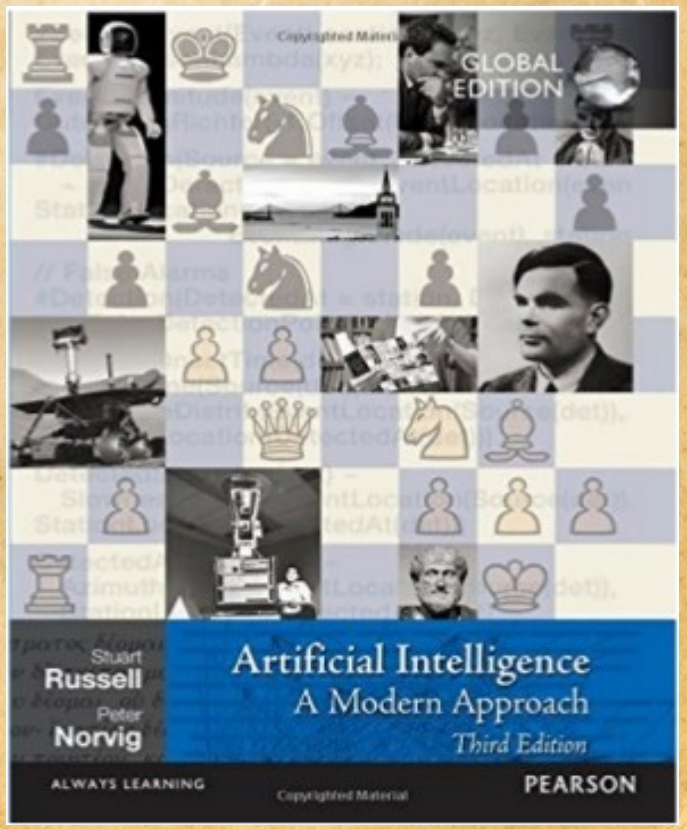
Literatuur

Artificial Intelligence: A Modern Approach

Third Edition

Stuart J. Russell and Peter Norvig

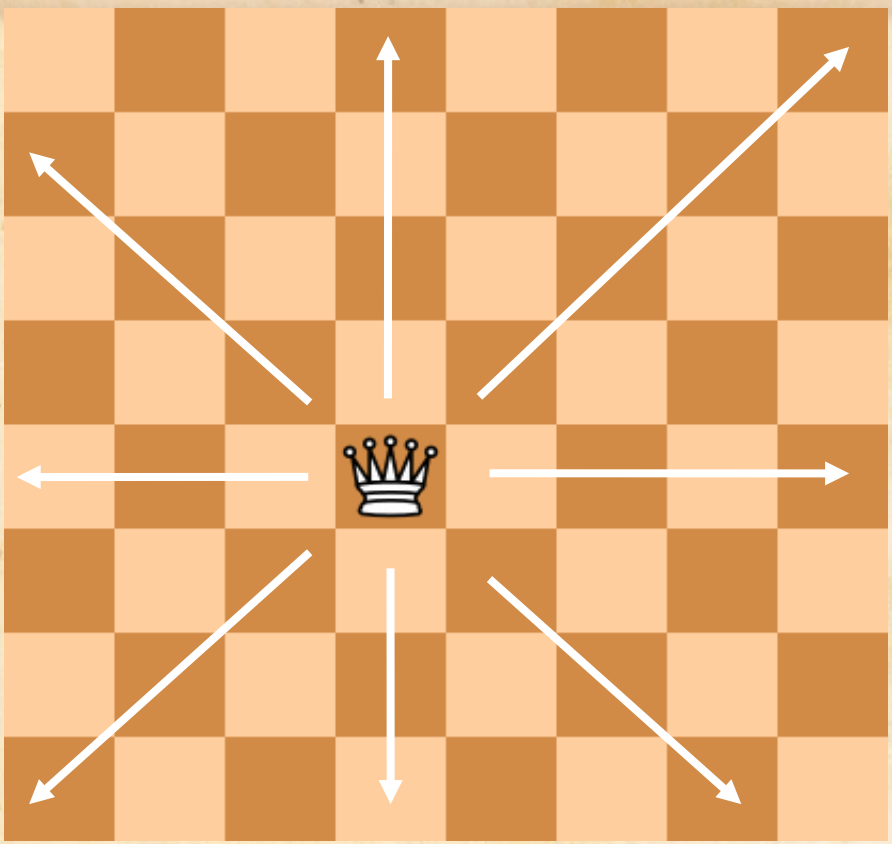
prgrf 3.4 Breadth first, Depth first
prgrf 3.5 A*
prgrf 6.3 Most Constraint Variable heuristic

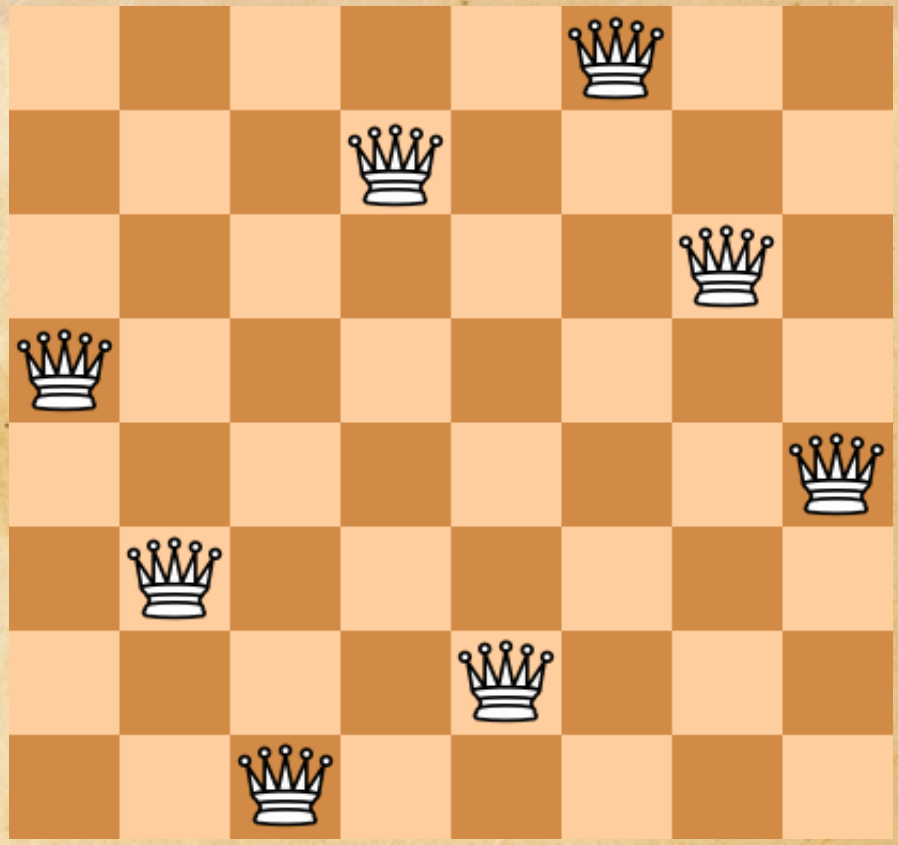


Lecture video 2020, terugkijken

- Youtube playlist, Constructieve algoritme
  - https://www.youtube.com/playlist?list=PLJBtJTYGPSzIfEzXpszM8Ewsllwfa0d6T

Toetje: N-Queens





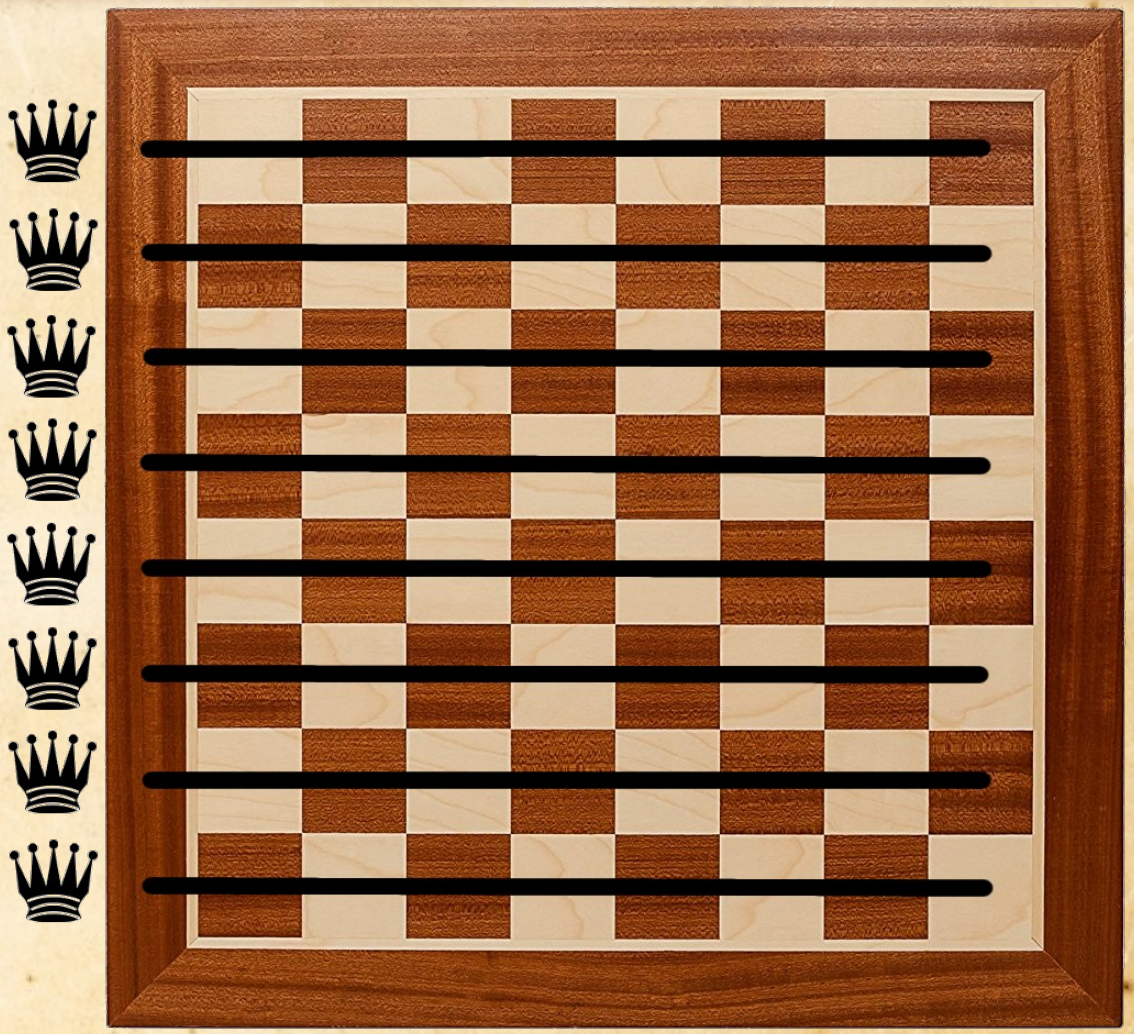
State space? Order:no Repetition:no $\frac{(NxN)!}{(N! \times (NxN - N)!)}$

N=8: $\frac{(8x8)!}{(8! \times (8x8 - 8)!)} = \frac{64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57}{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 4426165368$

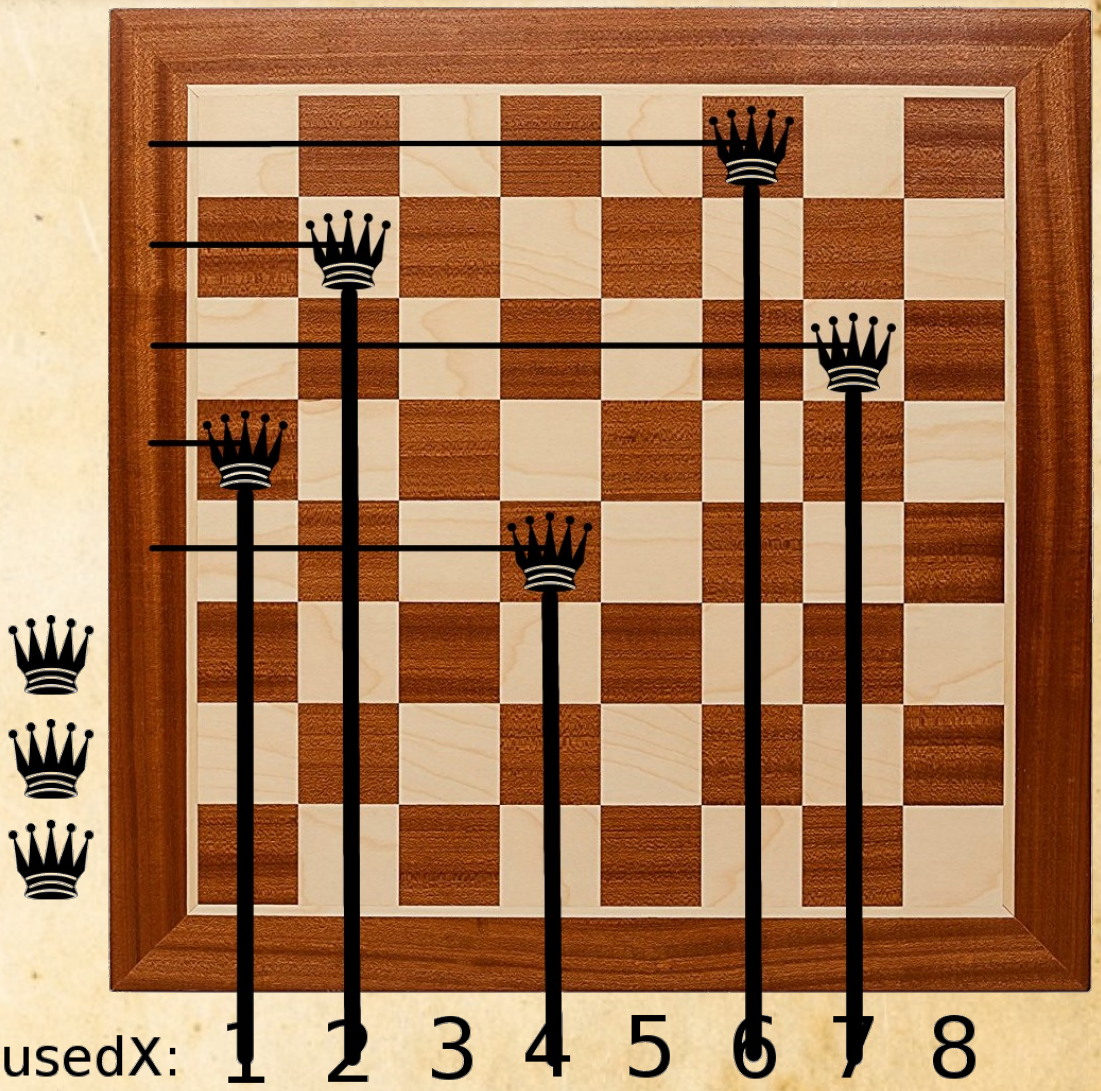
N-Queens

Queens	Squares	State Space	Approximate Time to Solve
8	64	4426165368	4.42 Seconds
9	81	260887834350	4.34 Minutes
10	100	17310309456440	4.8 Hours
11	121	1276749965026540	14.77 Days
12	144	103619293824707000	39.40 Months
13	169	9176358300744340000	290.78 Years
14	196	880530516383349000000	27 902.92 Years
15	225	9100556781117750000000	2 883 854.02 Years
16	256	1007875160202230000000000	319 383 187 Years
17	289	119073904434449000000000000	$3.77330493 \times 10^{10}$ Years
18	324	14948249233419500000000000000	$4.73691552 \times 10^{12}$ Years
19	361	1987086705354380000000000000000	$6.29683229 \times 10^{14}$ Years
20	400	278836098367090000000000000000000	$8.83597149 \times 10^{16}$ Years

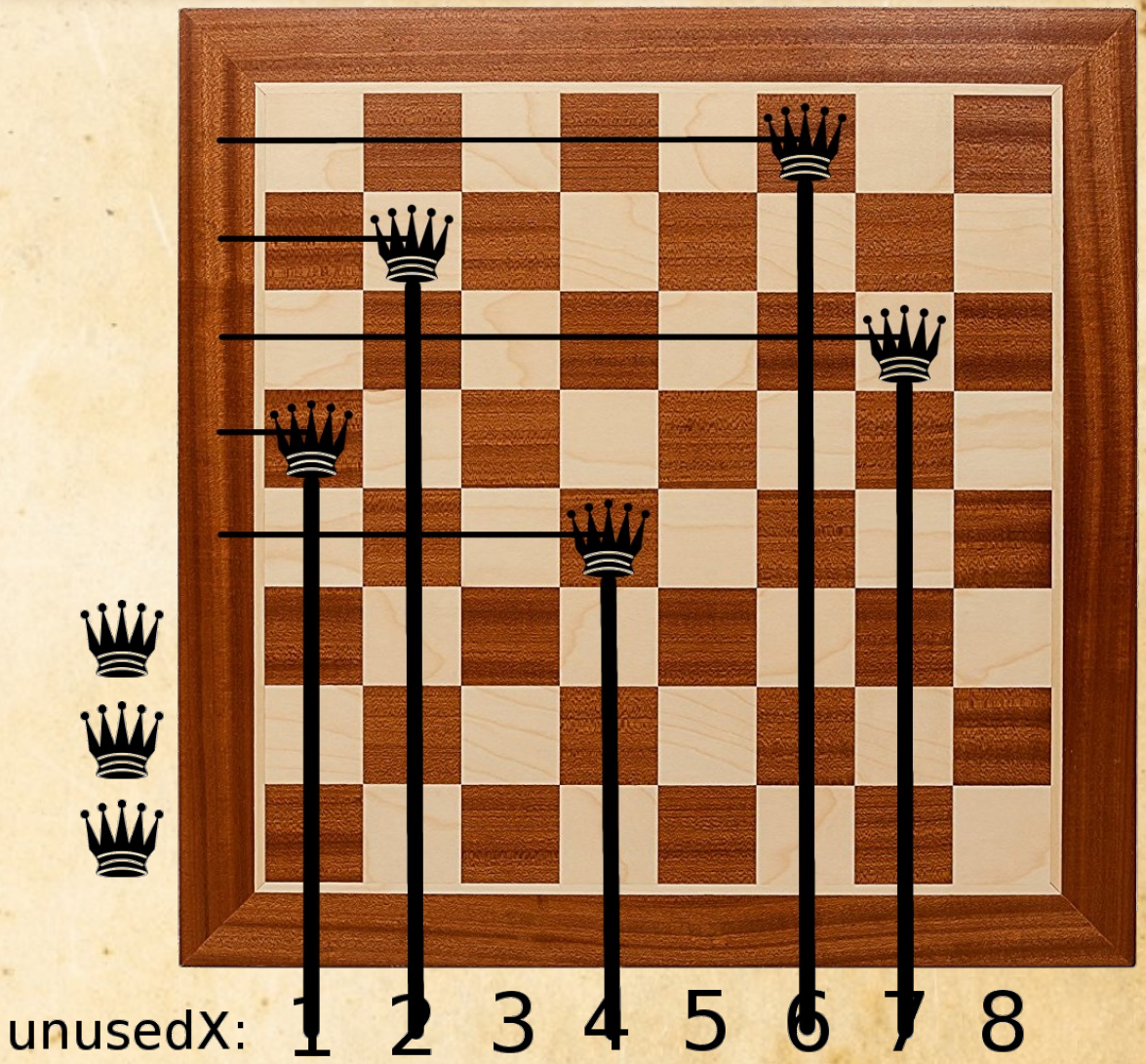
N-Queens, representatie



N-Queens, representatie



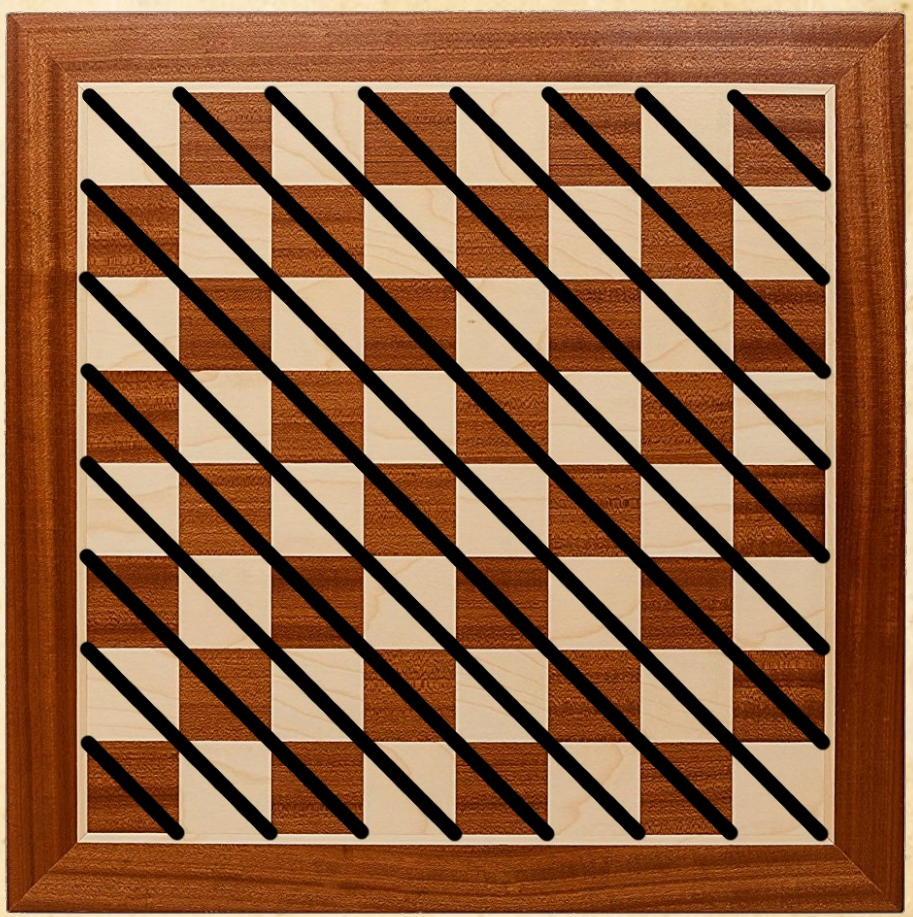
N-Queens, representatie

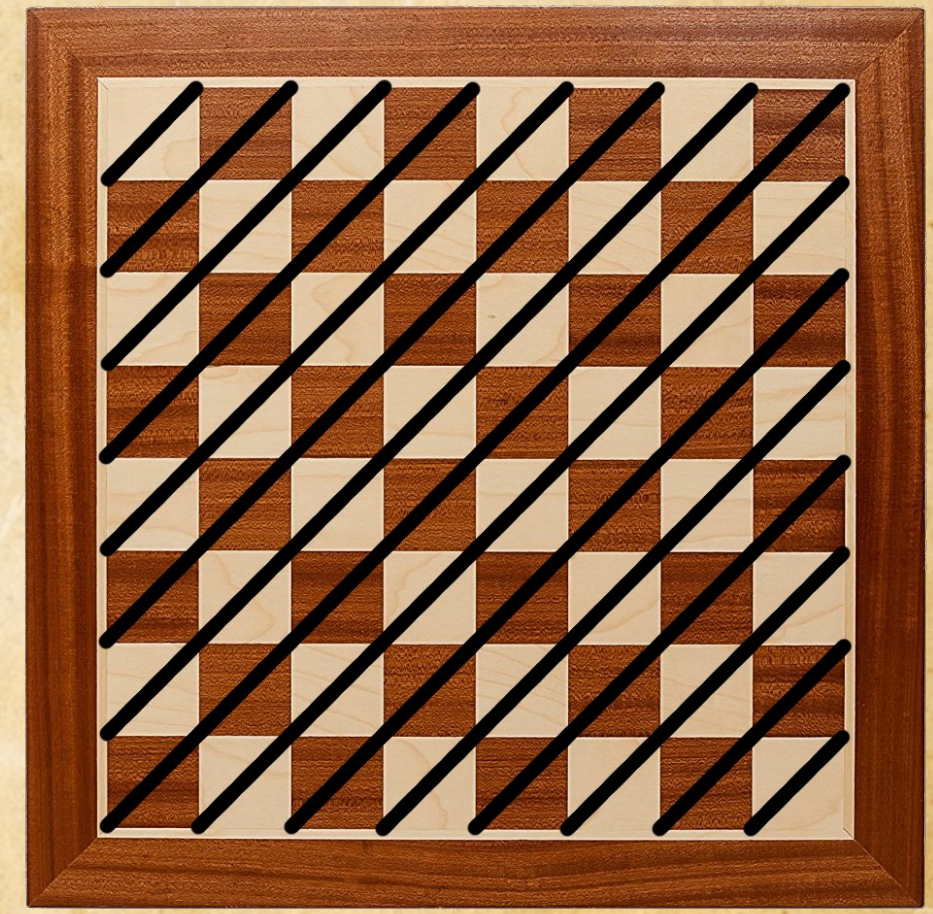


State space: N! (symmetrie?)

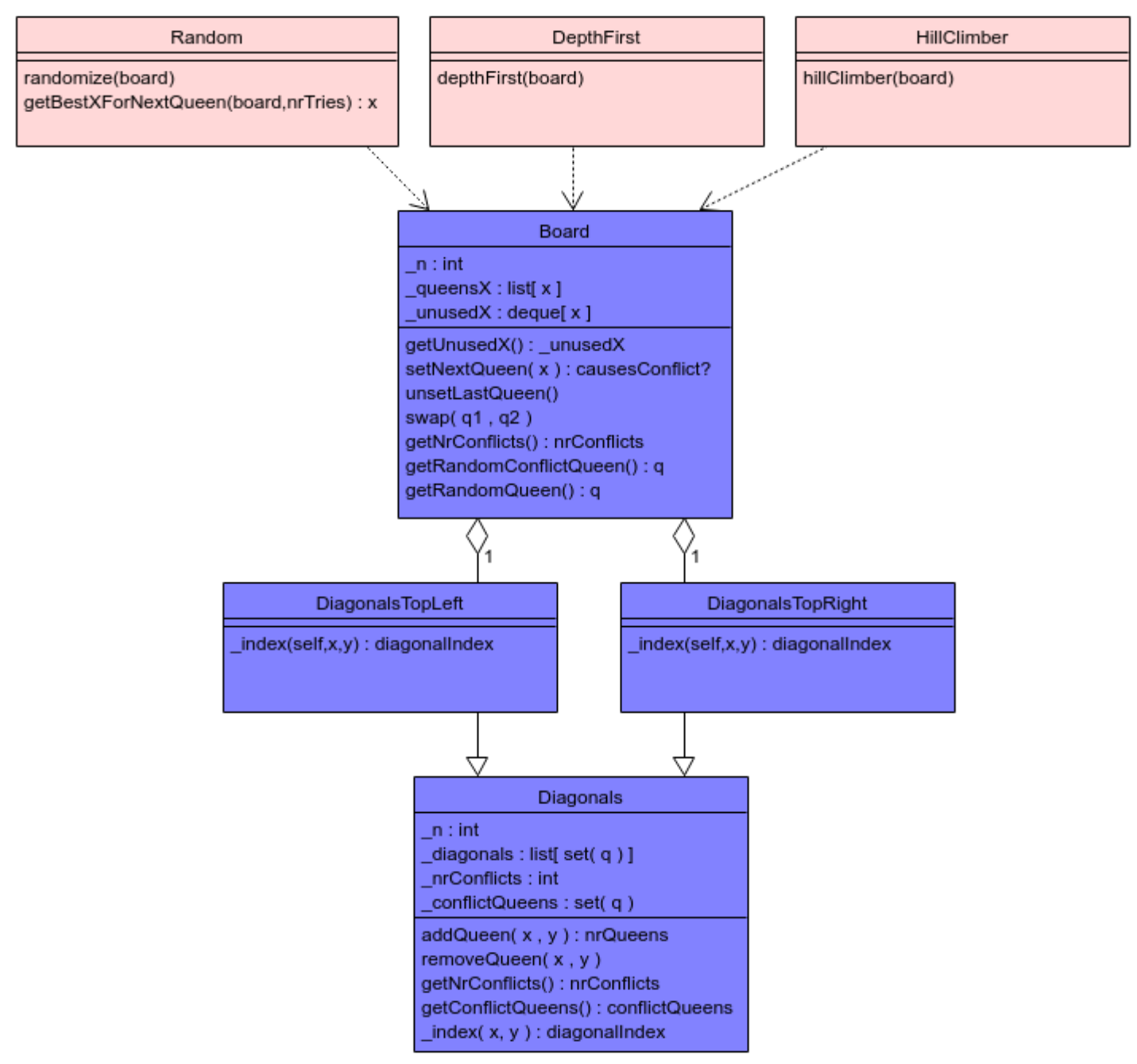
N-Queens, representatie

Alleen nog conflicten op de N+N-1 ‘Top-Left’ en N+N-1 ‘Top-Right’ diagonalen





N-Queens, representatie



N-Queens, Demo

GitHub: https://github.com/bterwijn/NQueens