

# Optimization of a Wordle Algorithm

Morten Heine Sørensen  
mhs@formalit.dk

*When you have eliminated all which is impossible, then  
whatever remains,  
however improbable,  
must be the truth.*

**Sherlock Holmes**

*In Arthur Conan Doyle, The Case Book of Sherlock Holmes.*

# Wordle

<https://www.nytimes.com/games/wordle/index.html>

S	E	R	A	I
P	H	O	N	E
C	L	U	B	S
S	K	I	T	E
S	M	I	T	E

## Goal:

- Build algorithm that can guess secret in at most 6 attempts
- Check for every possible secret that max 6 attempts required

## Parts:

1. The algorithm (7 guesses required)
2. Performance Optimizations

# Wordle vs Master Mind

S	E	R	A	I
P	H	O	N	E
C	L	U	B	S
S	K	I	T	E
S	M	I	T	E

Aspect	Wordle	MasterMind
Colors	26	8
Holes	5	5
Attempts needed	6	7
Guesses/Secrets	12.972	32.768
Scores	242	20
Combinations	List	All
Score position known	Yes	No



# Overall Algorithm

## Algorithm 1. Super Mastermind/Wordle (1 secret)

1. Let  $S$  be the secret.
2. Let  $C$  be the **initial guess** from  $A$  (All valid combinations)
3. Let  $s = \text{score}(C, S)$ .
4. If  $s = s_w$ ,  $C$  is the secret.
5. Otherwise, proceed as follows.
6. Remove from  $A$  every  $C'$  with  $\text{score}(C', S) \neq s$ .
7. Pick a **next guess**  $C''$  and go to Step 3 with  $C = C''$

Well known MM(c,h) algorithm (**D. Knuth**)

Start from some **initial guess**

Guess / Score	Secret
S E R A I	S M I T E

If guess correct, we're done;  
 else remove any combination  
 against which guess **scores\*** differently

Guess / Score	All Combinations
S E R A I	<del>A A H E D</del>

S E R A I	S M I L E
-----------	-----------

S E R A I	S M I T E
-----------	-----------

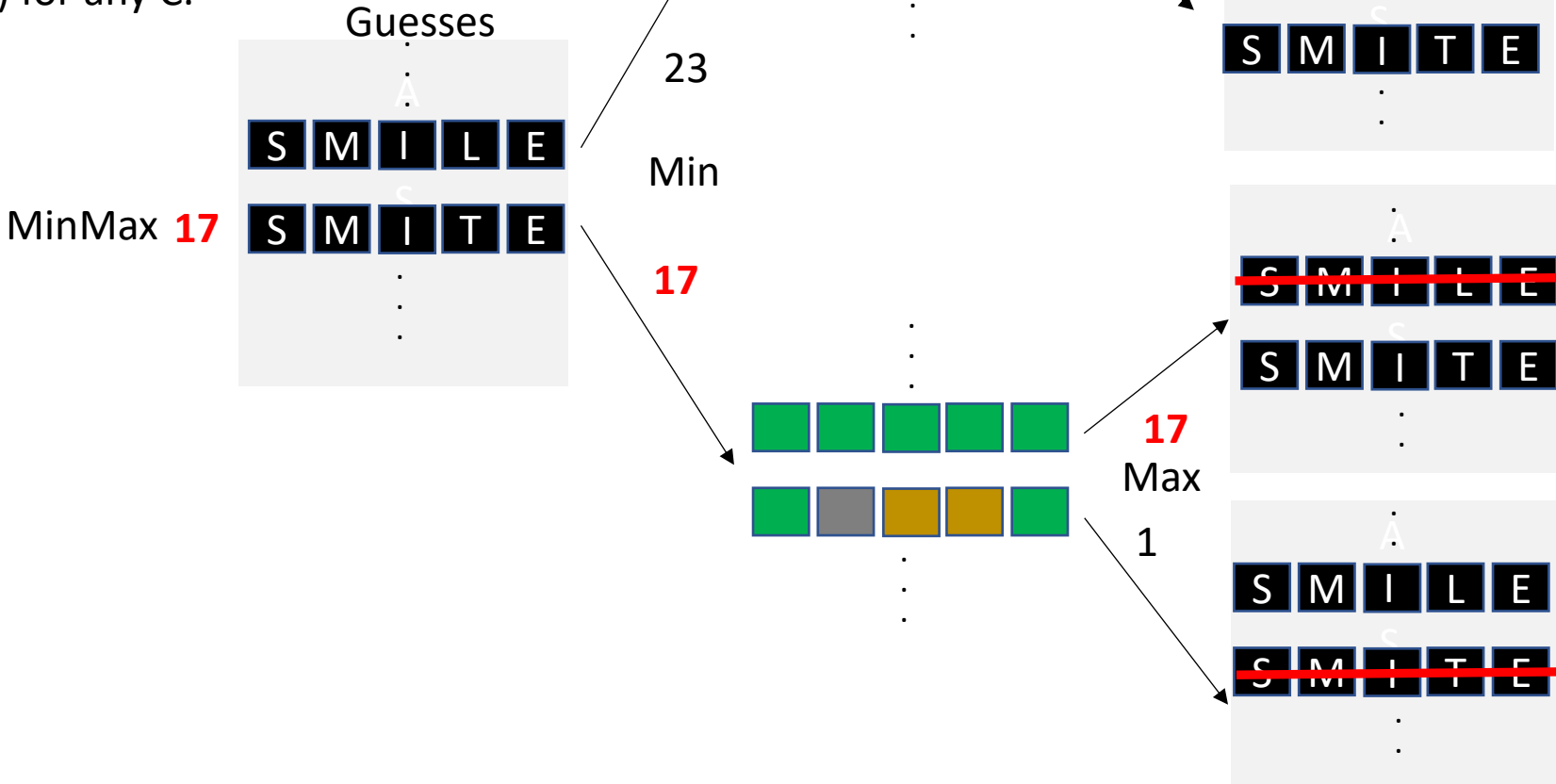
Try **next guess**

\* Not totally trivial – each guess peg can score against at one most one secret peg

# MinMax Next Guess Algorithm

**Algorithm 2. Next Guess**

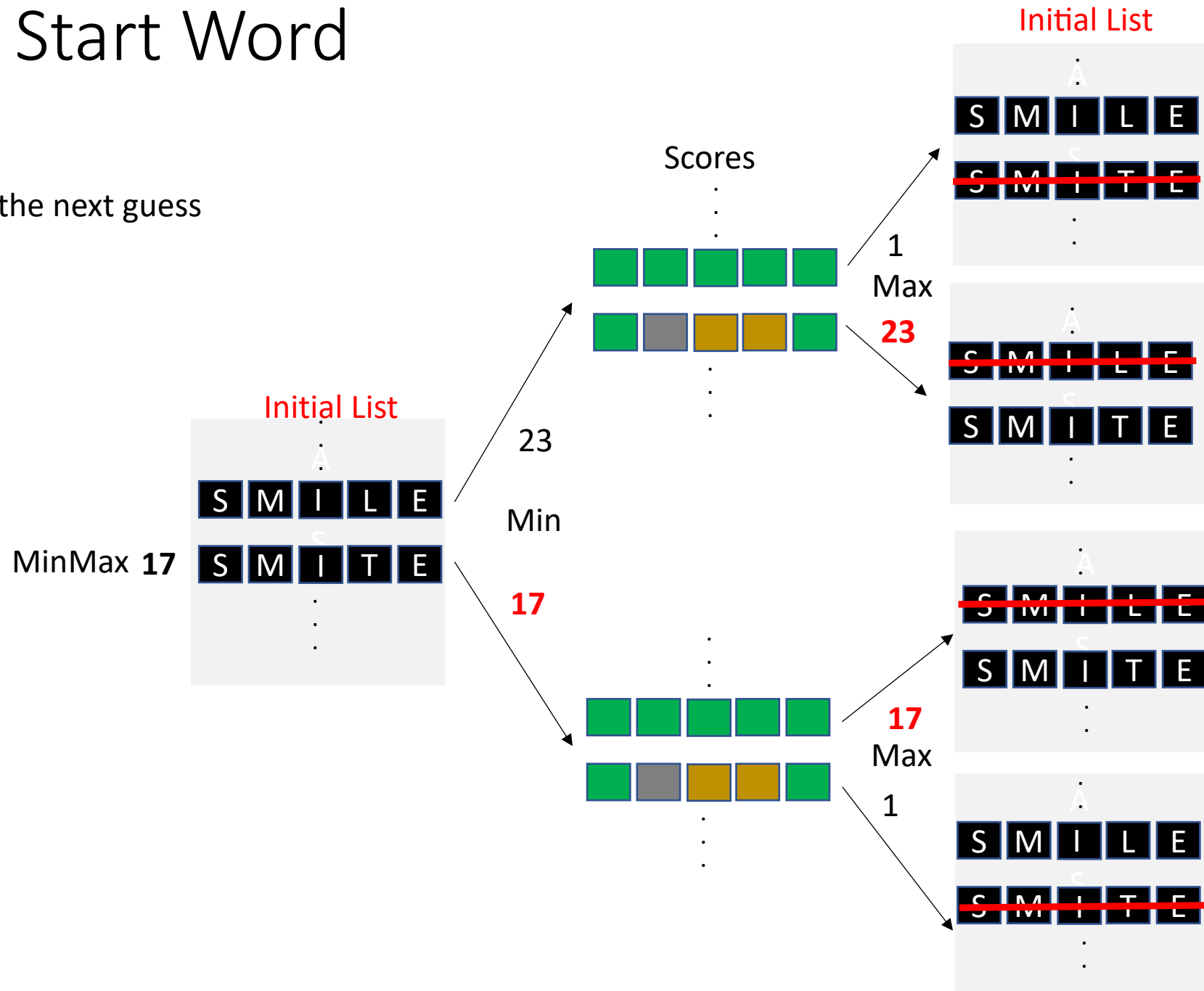
1. For each remaining combination C in A:
  - a. For each score s of C against any combination in A, let  $m(A,C,s)$  be the number of combinations C' in A with  $\text{score}(C,C')=s$ .
  - b. Let  $\text{Max}(A,C)$  be the **maximal**  $m(A,C,s)$  for any s.
2. Let  $\text{Min}(A)$  be the **minimal**  $\text{Max}(A,C)$  for any C.



# Initial Guess aka Start Word

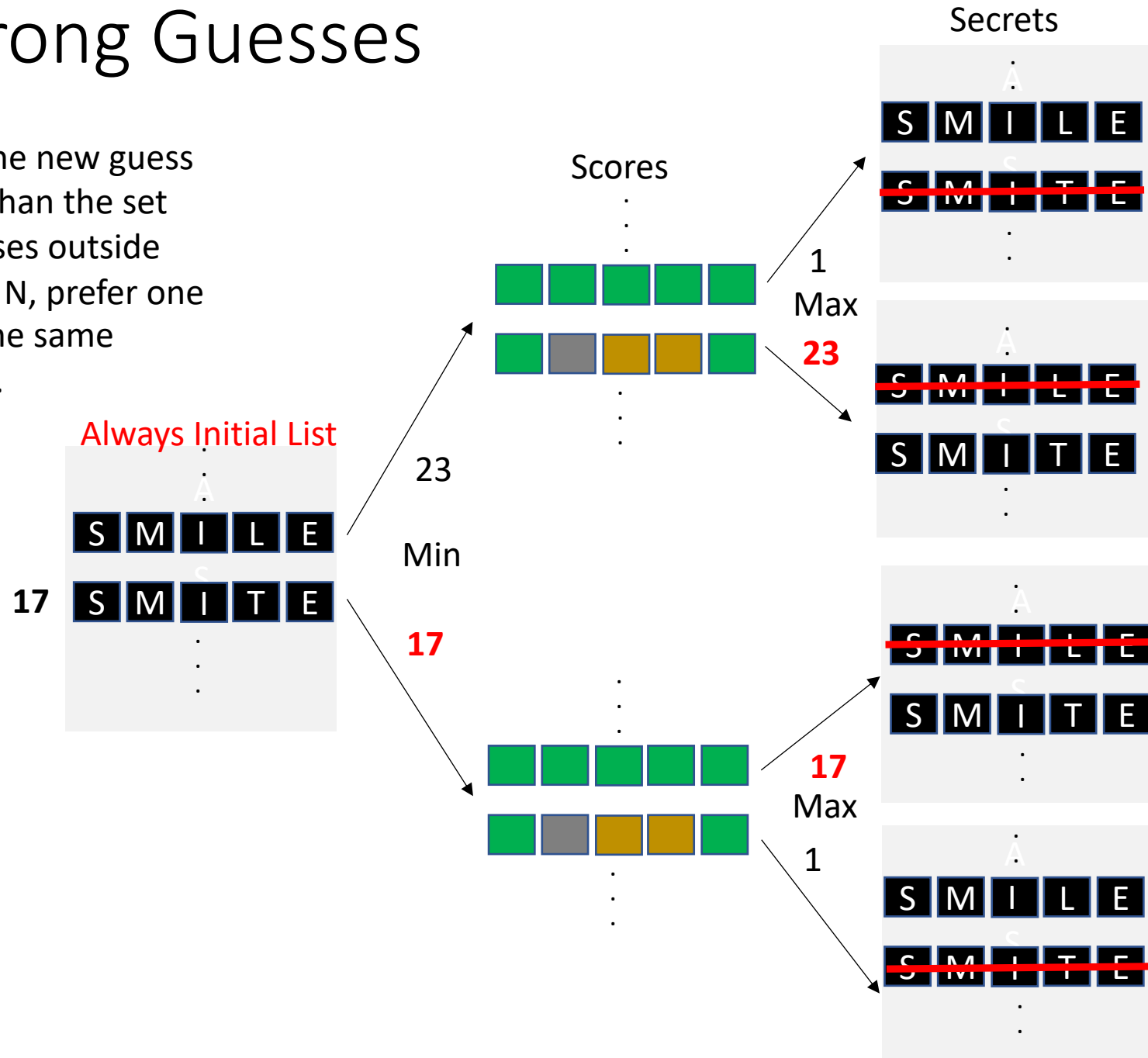
## Algorithm 3. Initial guess

For the initial guess, we can simply run the next guess algorithm on the initial set.



# Elimination using Wrong Guesses

**Algorithm 4. Improved next guess** In step 1 pick the new guess C from the initial set A of all combinations rather than the set of remaining combinations. If there are both guesses outside and inside the reduced set with the same number N, prefer one in the reduced set. When multiple guesses have the same number, choose the first one in alphabetical order.



# Start Word Tinkering

## ATTEMPTS NEEDED WITH SERAI

Guesses	Count	Avg
1	1	1
2	66	132
3	1740	5220
4	6412	25648
5	4059	20295
6	652	3912
7	38	266
8	4	32
<b>SUM</b>	<b>12972</b>	<b>4,27890842</b>

## LETTER FREQUENCIES

LETTER	HOLE 1	HOLE 2	HOLE 3	HOLE 4	HOLE 5
A	0,06	<b>0,17</b>	<b>0,10</b>	0,08	0,05
B	0,07	0,01	0,03	0,02	0,00
C	<b>0,07</b>	0,01	0,03	0,03	0,01
D	0,05	0,01	0,03	0,04	0,06
E	0,02	0,13	0,07	<b>0,18</b>	0,12
F	0,05	0,00	0,01	0,02	0,01
G	0,05	0,01	0,03	0,03	0,01
H	0,04	0,04	0,01	0,02	0,03
I	0,01	0,11	0,08	0,07	0,02
J	0,02	0,00	0,00	0,00	0,00
K	0,03	0,01	0,02	0,04	0,02
L	0,04	0,05	0,07	0,06	0,04
M	0,05	0,01	0,04	0,03	0,01
N	0,03	0,03	0,07	0,06	0,04
O	0,02	0,16	0,08	0,05	0,03
P	0,07	0,02	0,03	0,03	0,01
Q	0,01	0,00	0,00	0,00	0,00
R	0,05	0,07	<b>0,09</b>	0,06	0,05
S	<b>0,12</b>	0,01	0,04	0,04	<b>0,31</b>
T	0,06	0,02	0,05	0,07	0,06
U	0,01	0,09	0,05	0,03	0,01
V	0,02	0,00	0,02	0,01	0,00
W	0,03	0,01	0,02	0,01	0,00
X	0,00	0,00	0,01	0,00	0,01
Y	0,01	0,02	0,02	0,01	0,10
Z	0,01	0,00	0,01	0,01	0,00
MAX	<b>0,12</b>	<b>0,17</b>	<b>0,10</b>	<b>0,18</b>	<b>0,31</b>
	C	A	R	E	S
	4	3	5	2	1

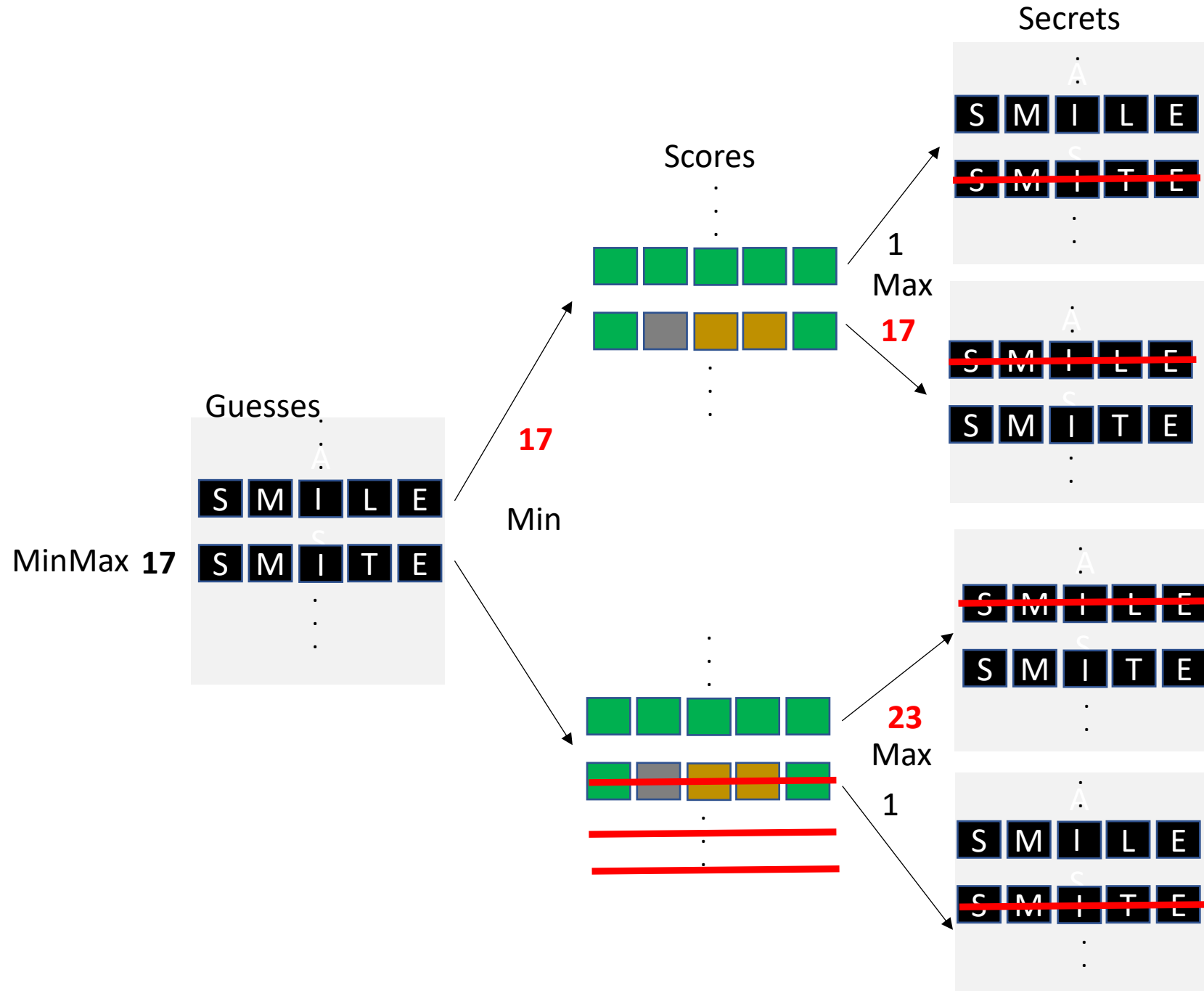
-> LARES 7 guesses needed

# Initial version

- Python
- Handles Mastermind and Wordle with any holes and colors
- Only difference in score computation and score equality
- **Game**: startword against one secret
- 1 game > 85m
- All games (secrets) ~ 2y
- < 0.5 Gb

# MinMax

- Minimum so far passed to the calculation of next guess' maximum.
- When maximum calculation exceeds minimum so far, go to next guess
- 1 game > 25m
- All games > 223d
- < 0.5 Gb
- Factor 3.5



# Precomputing the scores


- Profile shows score calculation expensive
  - Code generic
  - Computation tricky
- Precompute and store in large array
- 1 game ~ 1m (2m preload)
- All games ~ 9d
- 11.5 Gb
- Factor 25
- **This was the version from the event**

```
1884630907 function calls (1884630541 primitive calls) in 710.813 seconds
```

Ordered by: standard name					
nccalls	totttime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	710.813	710.813	<string>:1(<module>)
2	0.000	0.000	0.000	0.000	0.000 _bootlocale.py:33(getpreferredencoding)
1	0.000	0.000	0.000	0.000	0.000 _weakrefset.py:38(_remove)
1	0.000	0.000	0.000	0.000	0.000 codecs.py:186(__init__)
2	0.000	0.000	0.000	0.000	0.000 codecs.py:260(__init__)
12972	0.001	0.000	0.001	0.000	0.000 codecs.py:276(reset)
2	0.000	0.000	0.000	0.000	0.000 codecs.py:309(__init__)
381	0.000	0.000	0.001	0.000	0.000 codecs.py:319(decode)
12972	0.004	0.000	0.005	0.000	0.000 codecs.py:327(reset)
488/122	0.000	0.000	0.005	0.000	0.000 frontendcomm.py:255(_call__)
141	0.000	0.000	0.004	0.000	0.000 iostream.py:197(schedule)
122	0.000	0.000	0.000	0.000	0.000 iostream.py:310(_is_master_process)
122	0.000	0.000	0.000	0.000	0.000 iostream.py:323(_schedule_flush)
122	0.000	0.000	0.004	0.000	0.000 iostream.py:386(write)
141	0.000	0.000	0.000	0.000	0.000 iostream.py:93(_event_pipe)
141	0.003	0.000	0.003	0.000	0.000 socket.py:432(send)
141	0.000	0.000	0.000	0.000	0.000 threading.py:1017(_wait_for_tstate_lock)
141	0.000	0.000	0.000	0.000	0.000 threading.py:1071(is_alive)
141	0.000	0.000	0.000	0.000	0.000 threading.py:513(is_set)
12608784	3.737	0.000	3.737	0.000	0.000 wordle.py:100(is_latest_comb)
12556896	11.626	0.000	11.626	0.000	0.000 wordle.py:106(next_comb)
122249730	463.222	0.000	541.458	0.000	0.000 wordle.py:121(score)
51888	0.529	0.000	710.142	0.014	0.014 wordle.py:160(worst_elimination)
4	0.342	0.086	710.488	177.622	177.622 wordle.py:171(best_guess_and_elimination)
4	0.000	0.000	710.569	177.642	177.642 wordle.py:200(play_round)
1	0.000	0.000	710.569	710.569	710.569 wordle.py:209(play_game)
1	0.000	0.000	710.812	710.812	710.812 wordle.py:238(play_all_games)
1	0.116	0.116	0.239	0.239	0.239 wordle.py:260(readFile)
1	0.003	0.003	0.243	0.243	0.243 wordle.py:282(all_word_combinations)
135	0.000	0.000	0.000	0.000	0.000 wordle.py:33(my_char)
64865	0.041	0.000	0.041	0.000	0.000 wordle.py:4(my_ord)
1	0.000	0.000	0.000	0.000	0.000 wordle.py:63(strScore)
27	0.000	0.000	0.000	0.000	0.000 wordle.py:69(strPegs)
125571267	66.473	0.000	82.004	0.000	0.000 wordle.py:75(eqPegs)
51888	0.020	0.000	20.985	0.000	0.000 wordle.py:86(all_scores)
51888	4.736	0.000	20.965	0.000	0.000 wordle.py:89(all_combinations)
381	0.000	0.000	0.000	0.000	0.000 {built-in method codecs.utf_8_decode}
2	0.000	0.000	0.000	0.000	0.000 {built-in method locale.nl_langinfo}
1	0.000	0.000	710.813	710.813	710.813 {built-in method builtins.exec}
122	0.000	0.000	0.000	0.000	0.000 {built-in method builtins.isinstance}
251435012	15.549	0.000	15.549	0.000	0.000 {built-in method builtins.len}
61	0.000	0.000	0.005	0.000	0.000 {built-in method builtins.print}
2	0.002	0.001	0.002	0.001	0.001 {built-in method io.open}
122	0.000	0.000	0.000	0.000	0.000 {built-in method posix.getpid}
141	0.000	0.000	0.000	0.000	0.000 {method 'acquire' of '_thread.lock' objects}
141	0.000	0.000	0.000	0.000	0.000 {method 'append' of 'collections.deque' objects}
1235379000	79.103	0.000	79.103	0.000	0.000 {method 'append' of 'list' objects}
2	0.000	0.000	0.000	0.000	0.000 {method 'close' of '_io.TextIOWrapper' objects}
1	0.000	0.000	0.000	0.000	0.000 {method 'disable' of '_lsprof.Profiler' objects}
1	0.000	0.000	0.000	0.000	0.000 {method 'discard' of 'set' objects}
12972	0.001	0.000	0.001	0.000	0.000 {method 'isalpha' of 'str' objects}
135	0.000	0.000	0.000	0.000	0.000 {method 'keys' of 'dict' objects}
1	0.024	0.024	0.025	0.025	0.025 {method 'readlines' of '_io._IOBase' objects}
279498	0.024	0.000	0.024	0.000	0.000 {method 'strip' of 'str' objects}
25944	0.003	0.000	0.003	0.000	0.000 {method 'upper' of 'str' objects}
12972	0.005	0.000	0.009	0.000	0.000 {method 'write' of '_io.TextIOWrapper' objects}

# numPy (Jonatan K.)

- Python lib
- Better large arrays
- 1 game ~ 1m30s (2m preload)
- All games ~ 13.5d
- 3.5 Gb
- Factor 0.67
- Worse performance
- Smaller mem footprint allows parallelization
- Shared cache not possible



```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

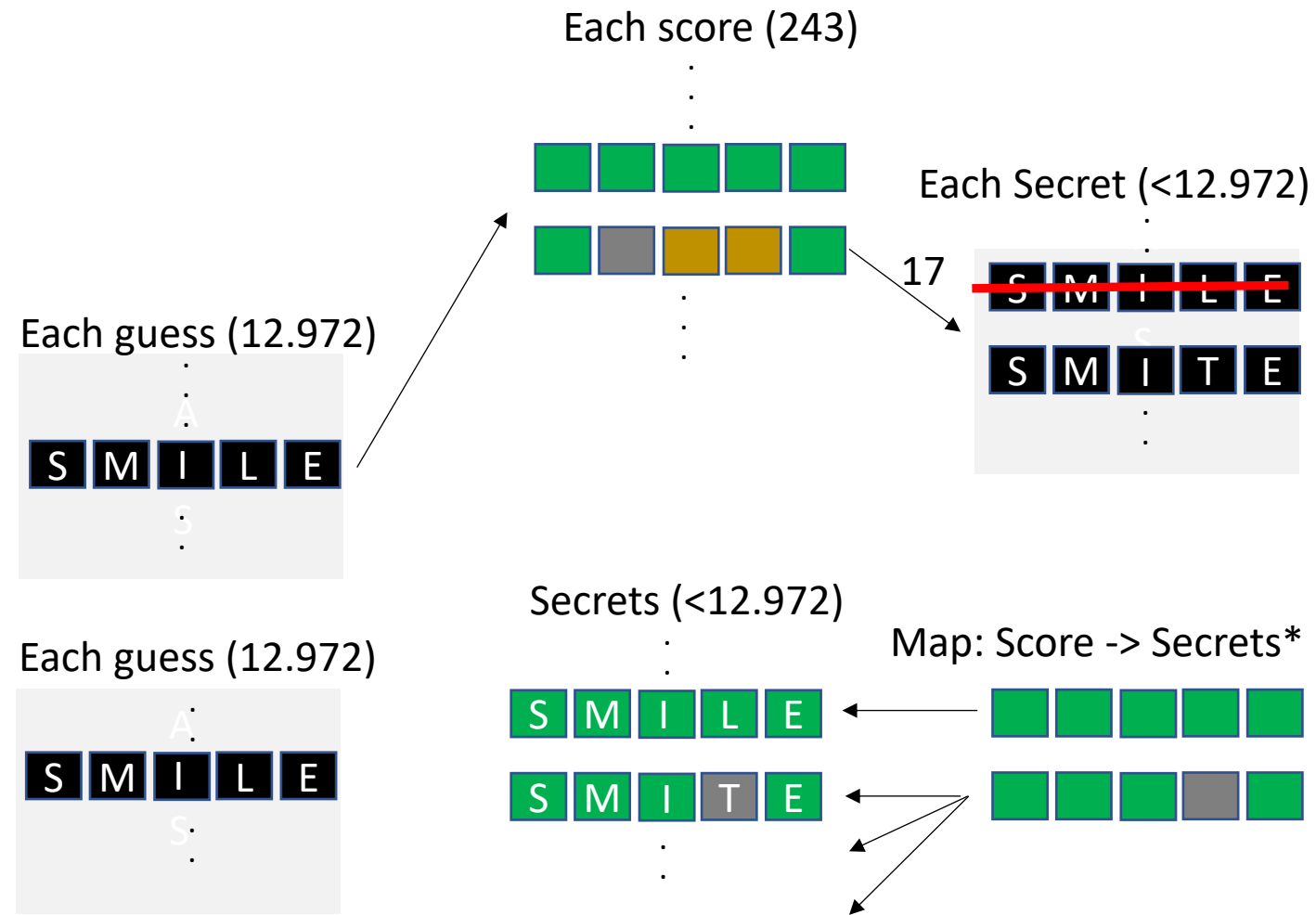
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

# Parallelization

- 3 threads
- Each their segment of game collection, e.g. 1-4000
- 1 game ~ 30s (2m preload)
- All games ~ 4.5d
- 10.5 Gb
- Factor 2 (compared to pre-numPy)

# Real scores, faster

- **Map:** Score -> Secrets
- 3 threads
- 1 game ~ 1.7s (2m preload)
- All games ~ 6h
- 10.5 Gb
- Factor 18

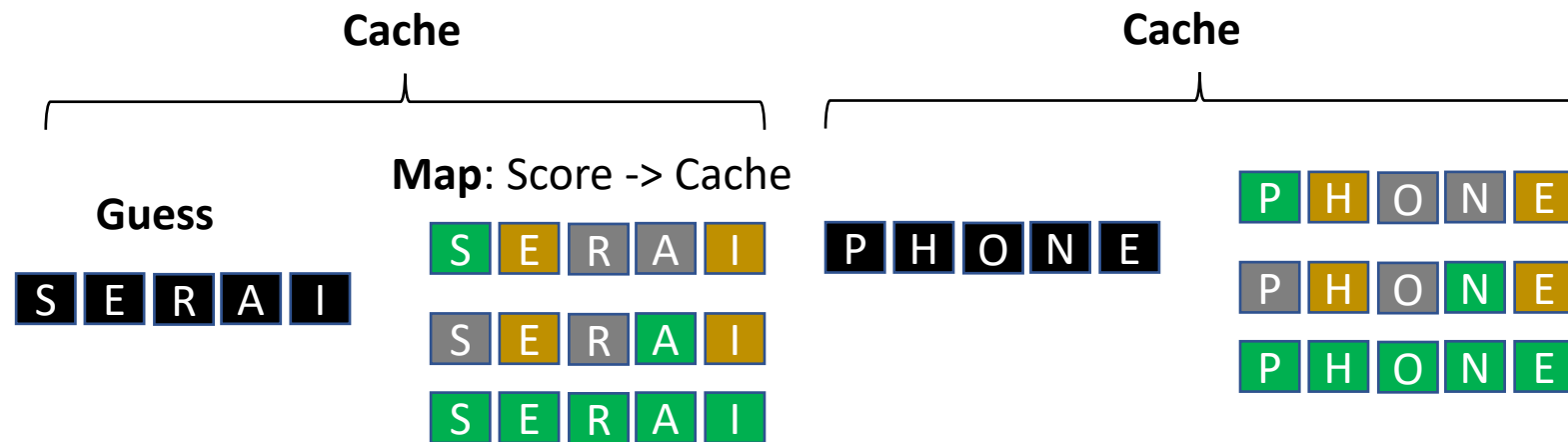


# Cache next guess from previous guess + score

- Next guess depends only on guesses and scores up to now
- Cross game cache
- 12.972 games to check, 7-8 guesses in each
- Only 243 different scores in Step 1, save 12.972-243 guesses
- Though 243 for each thread
- Cache:

- **Cache = Guess x Map**
- **Map: Score -> Cache**

- 3 threads
- 1 game ~ 0.28s (2m preload)
- All games ~ 1h
- 10.5 Gb
- Factor 6
- Cache is tabular representation of game strategy
- A bit tricky in C++ btw



# C++ (Olkhovski)

- Only 0.5 Gb per thread
- So more parallelization
- 16 threads
- 1 game ~ 0.018s (4m preload)
- All games ~ 2.5m
- 10.5 Gb
- Factor 24 (Raw C++ factor 10, rest is further parallelization)
- Original with shared next guess cache
- Core dumps, refactored to separate gamg segments

# Conclusion

Code version	Lang	Time 1 game	Time total	Mem	*
Initial version	Python	> 85m	> 779d	< 0,5 Gb	-
MinMax	Python	> 25m	> 223d	< 0,5 Gb	3.5
Precomp scores	Python	1m 2m preload	~9d	11,5 Gb	25
numPy	Python	1m30s 2m preload	~13.5d	3,5 Gb	-
numPy 3 Thread	Python	30s 2m preload	~4.5d	10,5 Gb	2
Real scores	Python	5s 2m preload	~18h	3,5 Gb	(18)
Real scores 3 Thread	Python	1,7s 2m preload	~6h	10,5 Gb	18
Cache guesses 3 Thread	Python	0,28s	~1h	10,5 Gb	6
C++ All optim except guesses 1 thread	C++	0,5s 2m preload	~110m 2m preload	0,5 Gb	(9.8)
C++ All optim except guesses 16 thread	C++	~0,07s 2m preload	15m 2m preload	0,5 Gb	24
C++ All optim 16 thread	C++	~ 0,018s	2.5m 4m preload	8 Gb	7,5

The main vehicles in the optimization were the following:

- 1.Precomputation** of scores, the same for every game and start word
- 2.Caching** of games, these vary with the start word
- 3.Parallelization** to multiple threads and CPUs
- 4.Avoiding unneeded computation** by modifying the algorithm
- 5.Changing data structure** to faster operations
- 6.Changing language** to compiled code and data structures closer to physical structures.
- 7.Profiling to identify time usage.**

Final factor **567.000**, from **2 years to 1.5 minute.**