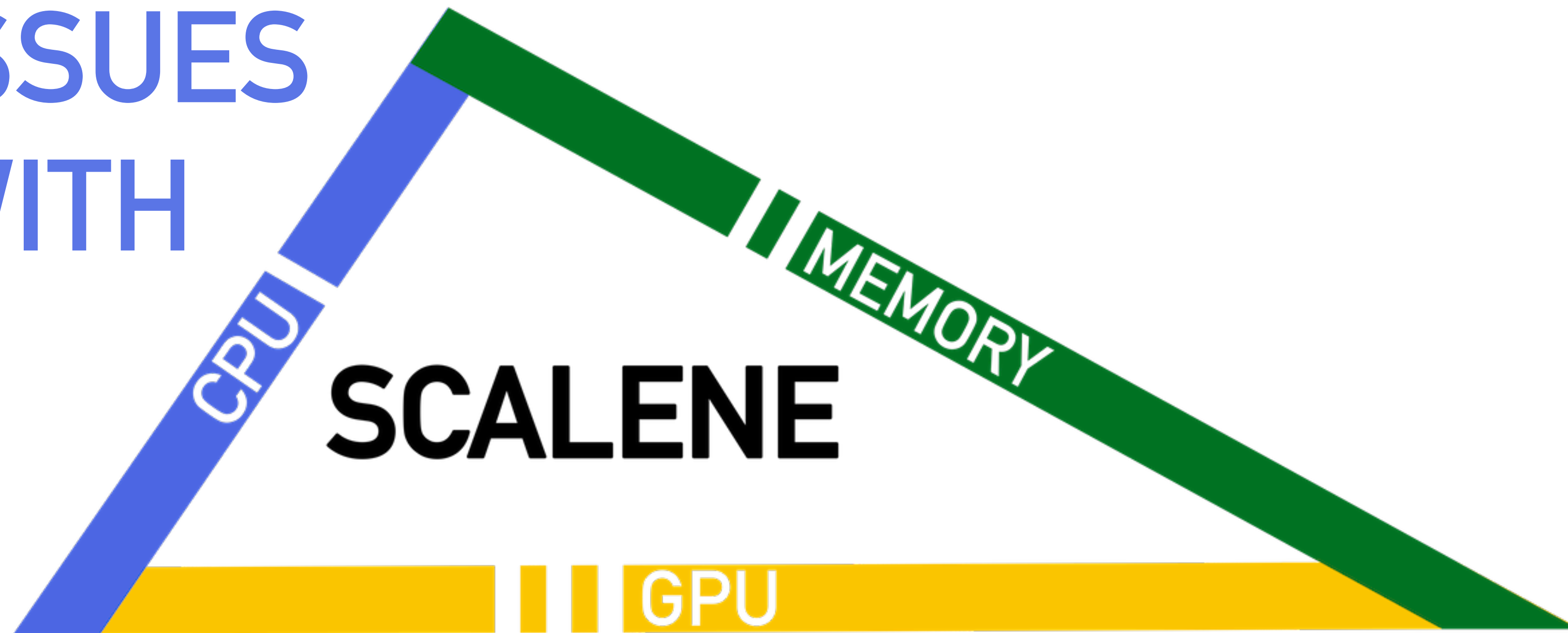
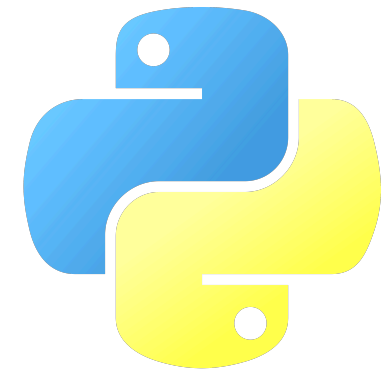


# TRIANGULATING PYTHON PERFORMANCE ISSUES WITH



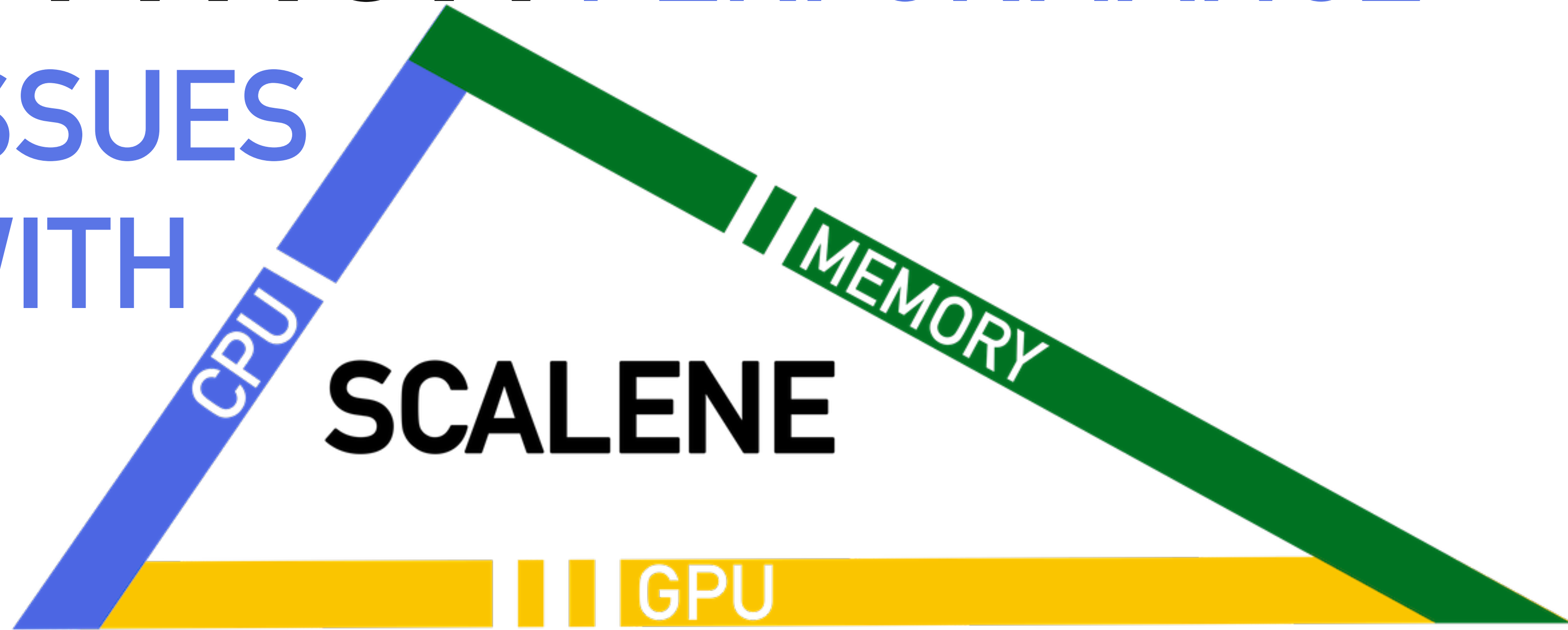
EMERY BERGER, SAM STERN,  
JUAN ALTMAYER PIZZORNO  
UNIVERSITY OF MASSACHUSETTS AMHERST







TRIANGULATING

# PYTHON PERFORMANCE

ISSUES  
WITH



EMERY BERGER, SAM STERN,  
JUAN ALTMAYER PIZZORNO  
UNIVERSITY OF MASSACHUSETTS AMHERST

Jul 2023	Jul 2022	Change	Programming Language
1	1		 <b>Python</b>
2	2		 C
3	4	↑	 C++
4	3	↓	 Java


**2023 Developer Survey**

All Respondents

Professional Developers

Learning to Code

Other Coders

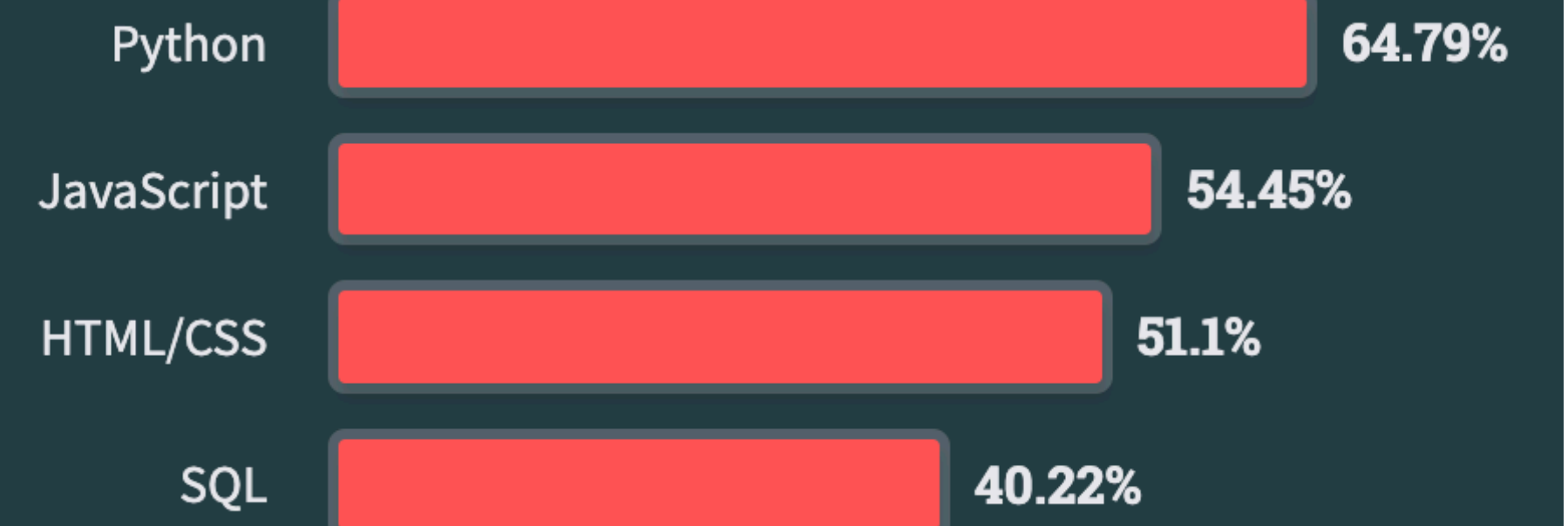


All Respondents

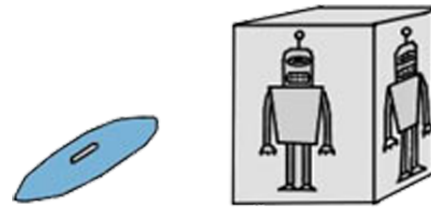
Professional Developers

Learning to Code

Other Coders



# "Metal" Languages - 1980s



```
MONITOR FOR 6802 3.4          3-14-82  DBC ABRAMS@SCLER PAGE 2
0000 00 00 70  STAYT   DD      00000000 MONITOR
*****
* FUNCTION: DATA - INSTALLATION MENU
* OBJECT: MENU
* CALL: menu
* RETURN: none
0021  8000  800  100000011
0022  C2000  800  100000011
0083  84 13  DATA  LSR A  8000000A  8000000A
0084  84 04  LSR A  800000  80000000
0085  84 11  LSR A  800000  80000000
0086  84 04  LSR A  800000  80000000
0087  78 00  011  JMP      800000  GO TO START OF MONITOR
*****
* FUNCTION: DATA - Input character
* OBJECT: MENU
* CALL: menu
* RETURN: none
*****
* DESCRIPTION: data 3 character from terminal
0089  84 04  DATA  LSR A  80000000
0090  84 04  LSR A  80000000
0091  84 04  LSR A  80000000
0092  84 04  LSR A  80000000
0093  84 04  LSR A  80000000
0094  84 04  LSR A  80000000
0095  84 04  LSR A  80000000
*****
* FUNCTION: MENU - INPUT MENU SHEET
* OBJECT: MENU
* CALL: menu
* RETURN: none
*****
0098  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0099  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0100  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0101  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0102  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0103  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0104  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0105  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0106  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0107  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0108  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0109  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0110  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0111  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0112  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0113  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0114  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0115  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0116  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0117  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0118  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0119  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0120  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0121  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0122  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0123  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0124  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0125  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0126  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0127  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0128  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0129  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0130  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0131  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0132  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0133  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0134  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0135  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0136  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0137  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0138  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0139  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0140  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0141  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0142  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0143  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0144  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0145  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0146  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0147  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0148  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0149  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0150  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0151  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0152  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0153  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0154  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0155  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0156  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0157  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0158  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0159  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0160  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0161  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0162  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0163  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0164  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0165  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0166  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0167  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0168  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0169  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0170  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0171  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0172  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0173  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0174  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0175  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0176  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0177  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0178  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0179  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0180  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0181  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0182  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0183  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0184  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0185  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0186  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0187  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0188  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0189  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0190  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0191  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0192  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0193  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0194  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0195  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0196  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0197  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0198  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0199  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
0200  84 00  011  JMP      800000  GO TO START OF MONITOR
*****
```

1949

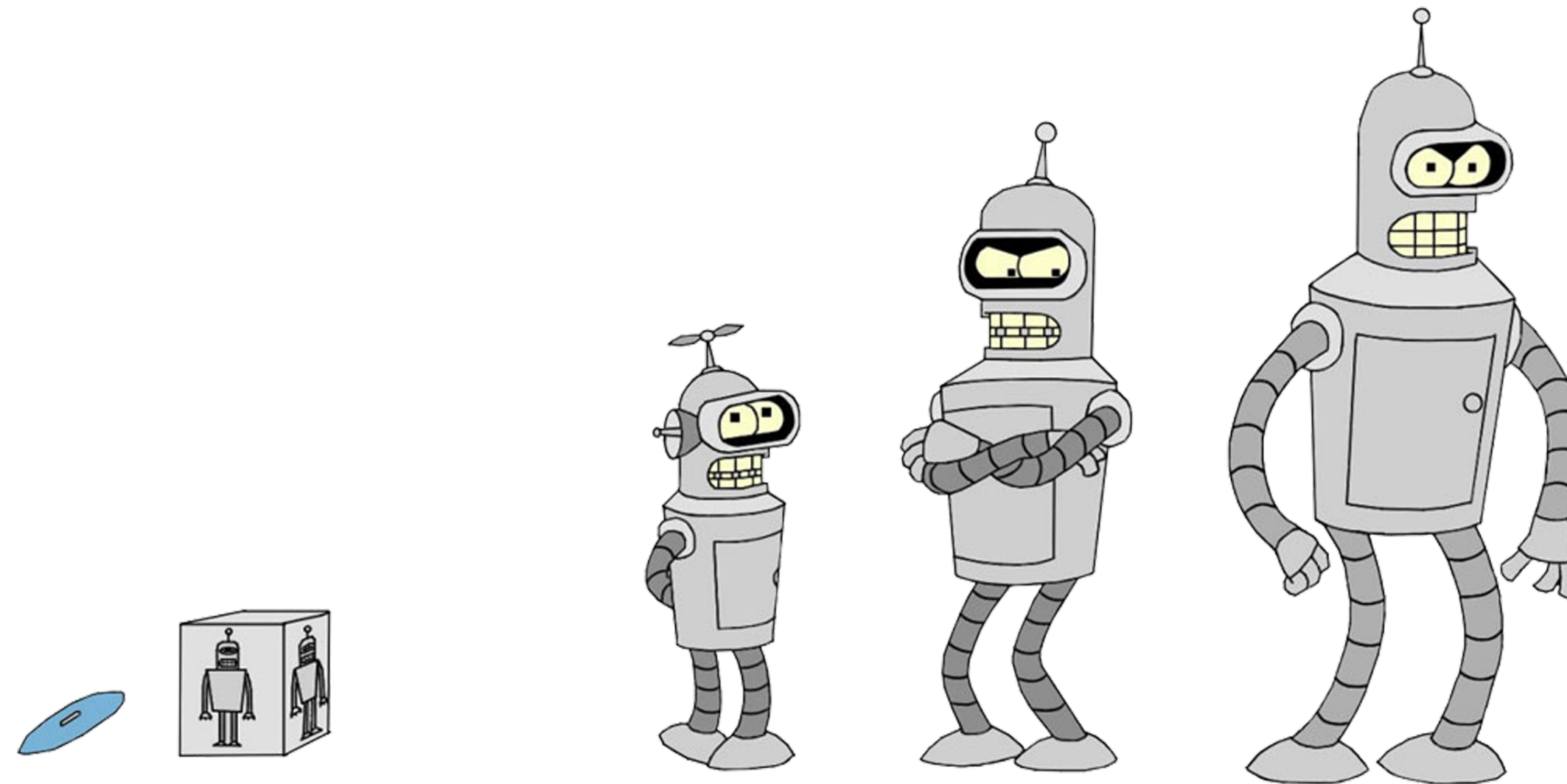




# "Metal" Languages

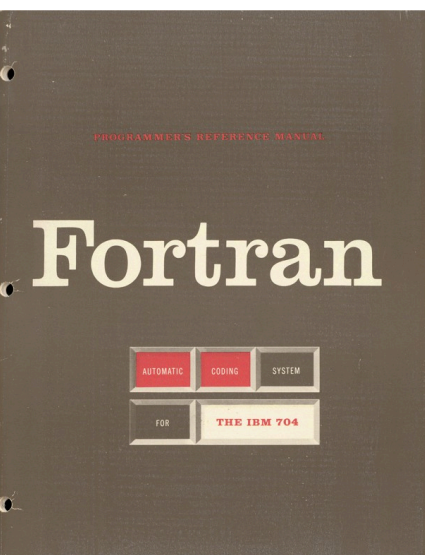
  

## -1980s

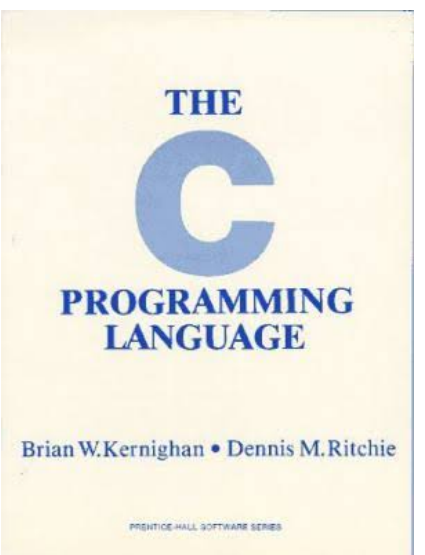


```
MONITOR FOR 6902 1.4          9-14-67  SOC ABBEDELKER  PAGE  2
0000 00 00 70  STAYT          SOB  RETH2C
*****
* FUNCTION:  CRTX - INSTALLS MFA
* INPUT:  none
* OUTPUT:  none
* CALLS:  none
* SECURITY:  not A
*****
0011 0000 000 10000011
0012          0000 000 10000011
0000 04 13  CRTX          LSA A  RBERTS  RBERT ACTA
0000 04 04          LSA A  RBRTS
0000 04 11          LSA A  RBRTS  RBRT R AND 2 800
0000 07 02 04          LSA A  RBRTS
0000 78 00 01          JOP  R2000  GO TO START OF MONITOR
*****
* FUNCTION:  DATA - Input character
* INPUT:  none
* OUTPUT:  char in var A
* CALLS:  none
* SECURITY:  not A
*****
* DESCRIPTION:  Data 1 character from terminal
0010 00 00 00  DATA          LSA A  RBRTS
0011 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0012 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0013 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0014 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0015 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
*****
* FUNCTION:  JUMP - JUMP
* INPUT:  none
* OUTPUT:  none
* CALLS:  none
* SECURITY:  not A
*****
* Description:  to monitor if not use input
0016 00 00 00  JUMP          R00  R200  GET A CHAR
0017 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0018 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0019 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0020 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0021 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0022 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0023 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0024 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0025 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0026 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0027 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0028 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0029 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0030 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0031 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0032 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0033 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0034 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0035 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0036 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0037 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0038 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0039 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
0040 00 00 00          LSA A  RBRTS  RBRT R AND 2 800
```

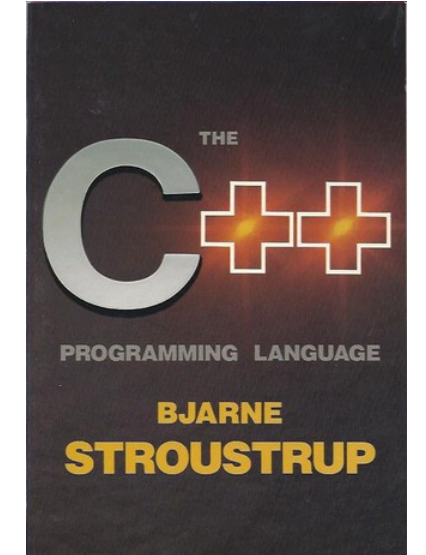
1949



1957



1972



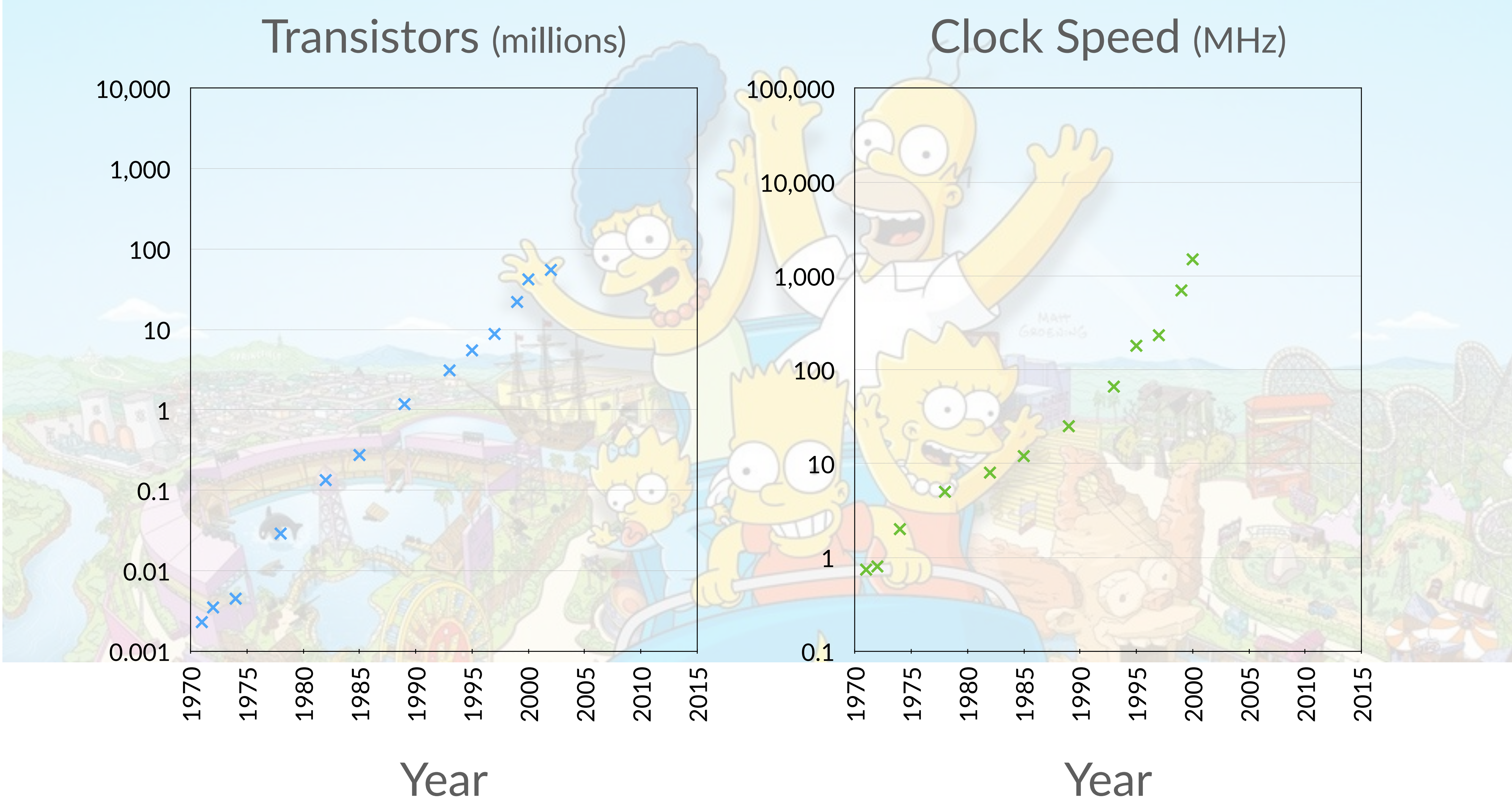
1985

# Wild Performance Ride (1980s–2010)!

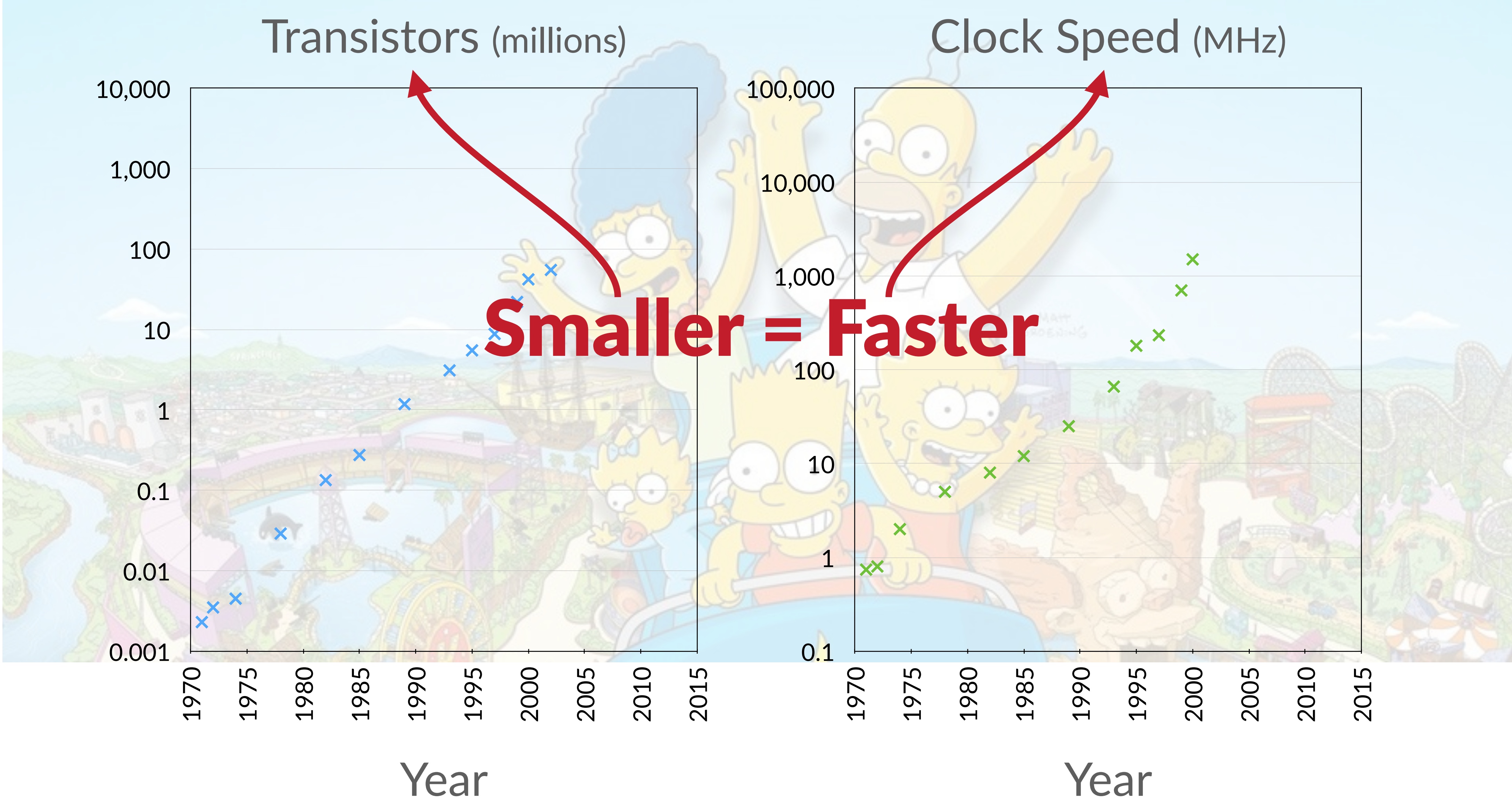




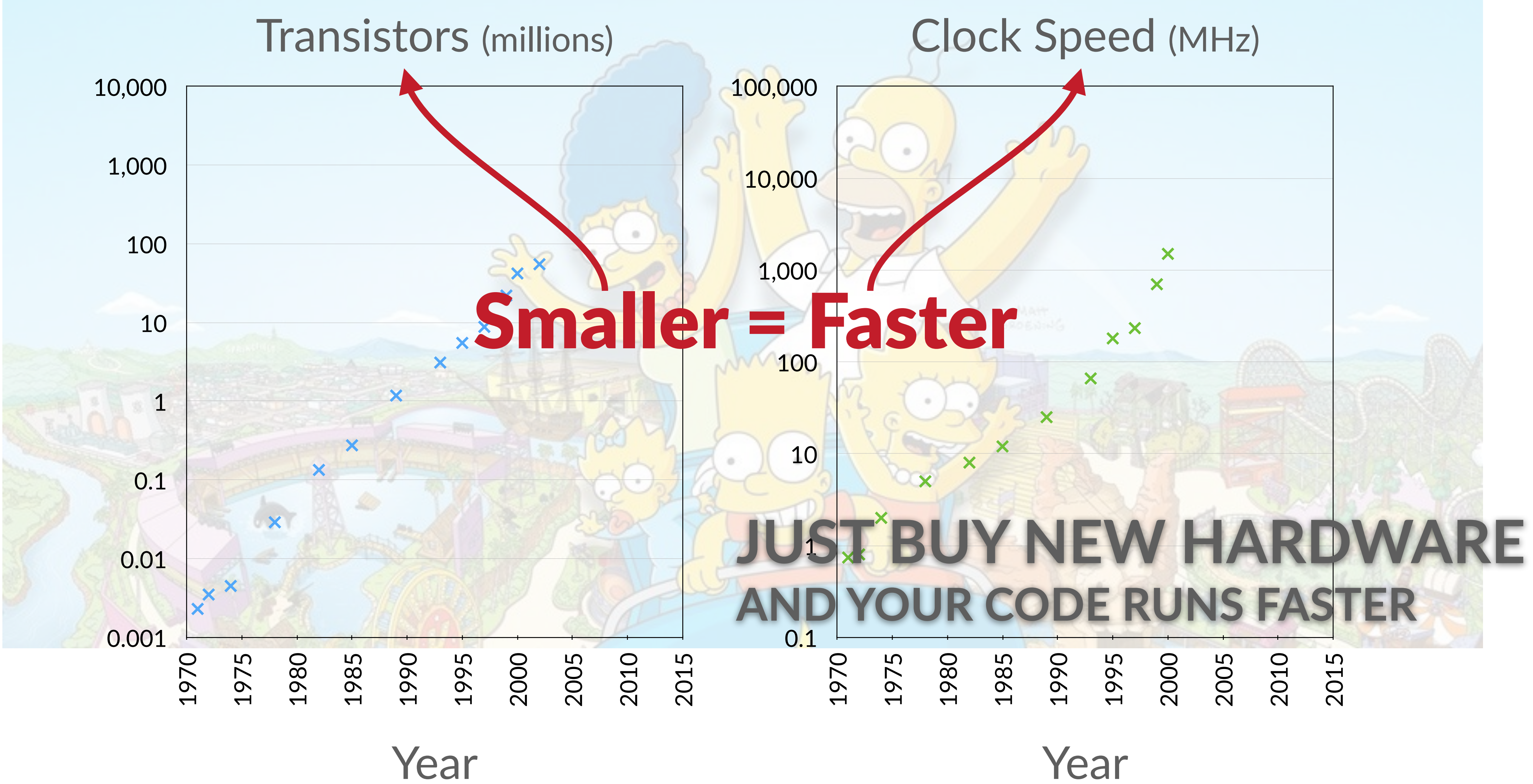
# Wild Performance Ride (1980s–2010)!



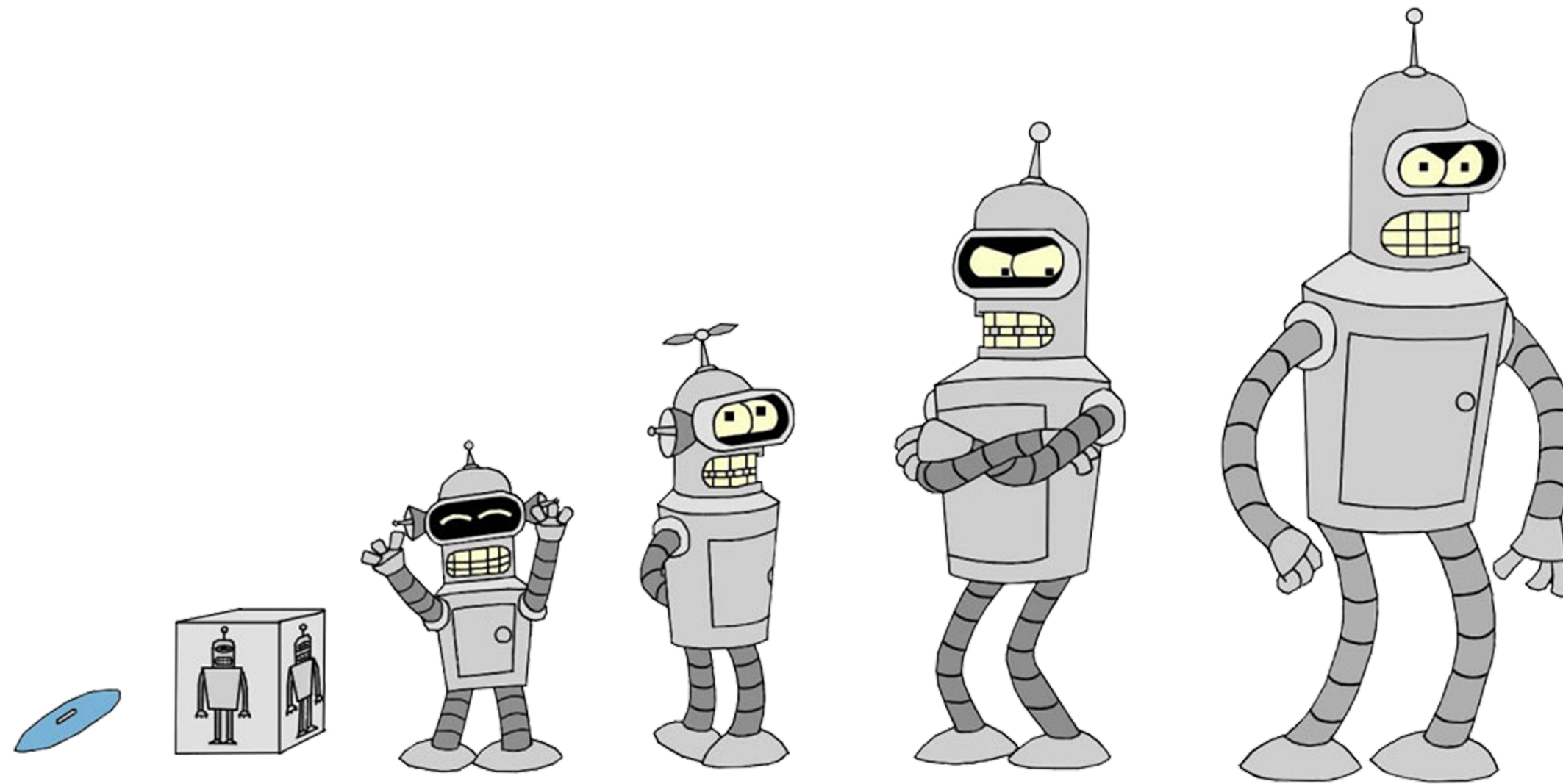
# Wild Performance Ride (1980s–2010)!



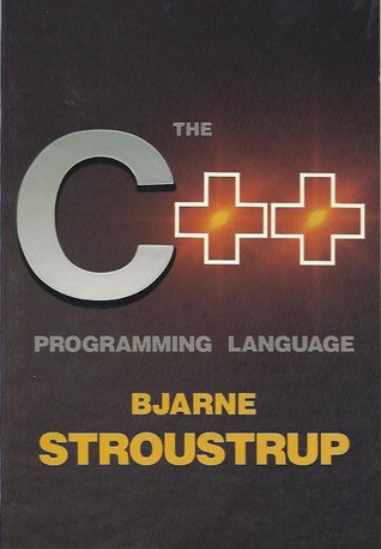
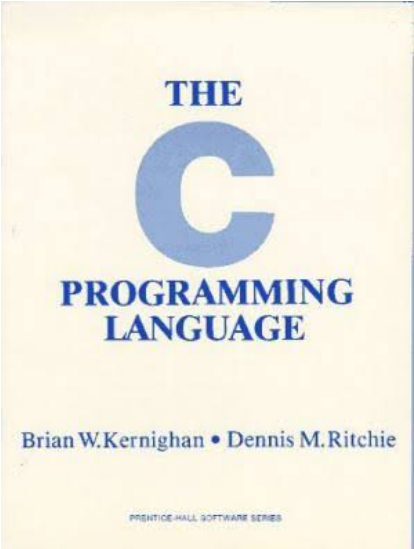
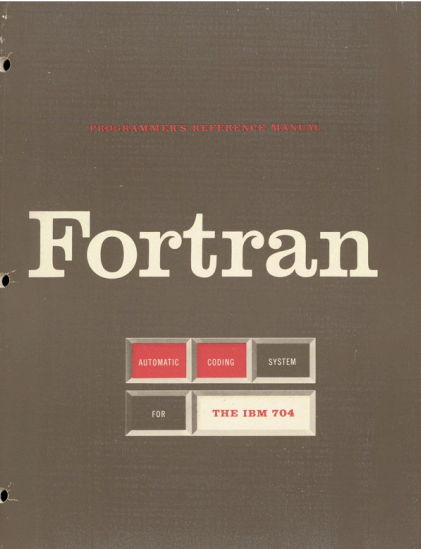
# Wild Performance Ride (1980s-2010)!



# "Metal" Languages -1980s



```
MONITOR FOR 6801 1.4      P-14-81  DEC 18/80 81MEL  PAGE  2
0000 00 00 70  STAYD  SDI  RETNCS
*****
* FUNCTION: DETA - Installan MIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* RETURN: none
0010 00 00 00 00 00
0020 00 00 00 00 00
0030 00 00 00 00 00
0040 00 00 00 00 00
0050 00 00 00 00 00
0060 00 00 00 00 00
0070 00 00 00 00 00
0080 00 00 00 00 00
0090 00 00 00 00 00
0100 00 00 00 00 00
0110 00 00 00 00 00
0120 00 00 00 00 00
0130 00 00 00 00 00
0140 00 00 00 00 00
0150 00 00 00 00 00
0160 00 00 00 00 00
0170 00 00 00 00 00
0180 00 00 00 00 00
0190 00 00 00 00 00
0200 00 00 00 00 00
0210 00 00 00 00 00
0220 00 00 00 00 00
0230 00 00 00 00 00
0240 00 00 00 00 00
0250 00 00 00 00 00
0260 00 00 00 00 00
0270 00 00 00 00 00
0280 00 00 00 00 00
0290 00 00 00 00 00
0300 00 00 00 00 00
0310 00 00 00 00 00
0320 00 00 00 00 00
0330 00 00 00 00 00
0340 00 00 00 00 00
0350 00 00 00 00 00
0360 00 00 00 00 00
0370 00 00 00 00 00
0380 00 00 00 00 00
0390 00 00 00 00 00
0400 00 00 00 00 00
0410 00 00 00 00 00
0420 00 00 00 00 00
0430 00 00 00 00 00
0440 00 00 00 00 00
0450 00 00 00 00 00
0460 00 00 00 00 00
0470 00 00 00 00 00
0480 00 00 00 00 00
0490 00 00 00 00 00
0500 00 00 00 00 00
0510 00 00 00 00 00
0520 00 00 00 00 00
0530 00 00 00 00 00
0540 00 00 00 00 00
0550 00 00 00 00 00
0560 00 00 00 00 00
0570 00 00 00 00 00
0580 00 00 00 00 00
0590 00 00 00 00 00
0600 00 00 00 00 00
0610 00 00 00 00 00
0620 00 00 00 00 00
0630 00 00 00 00 00
0640 00 00 00 00 00
0650 00 00 00 00 00
0660 00 00 00 00 00
0670 00 00 00 00 00
0680 00 00 00 00 00
0690 00 00 00 00 00
0700 00 00 00 00 00
0710 00 00 00 00 00
0720 00 00 00 00 00
0730 00 00 00 00 00
0740 00 00 00 00 00
0750 00 00 00 00 00
0760 00 00 00 00 00
0770 00 00 00 00 00
0780 00 00 00 00 00
0790 00 00 00 00 00
0800 00 00 00 00 00
0810 00 00 00 00 00
0820 00 00 00 00 00
0830 00 00 00 00 00
0840 00 00 00 00 00
0850 00 00 00 00 00
0860 00 00 00 00 00
0870 00 00 00 00 00
0880 00 00 00 00 00
0890 00 00 00 00 00
0900 00 00 00 00 00
0910 00 00 00 00 00
0920 00 00 00 00 00
0930 00 00 00 00 00
0940 00 00 00 00 00
0950 00 00 00 00 00
0960 00 00 00 00 00
0970 00 00 00 00 00
0980 00 00 00 00 00
0990 00 00 00 00 00
1000 00 00 00 00 00
```



1949

1957

1972

1985

# "Irrational Exuberance" Languages

1990s

# "Irrational Exuberance" Languages

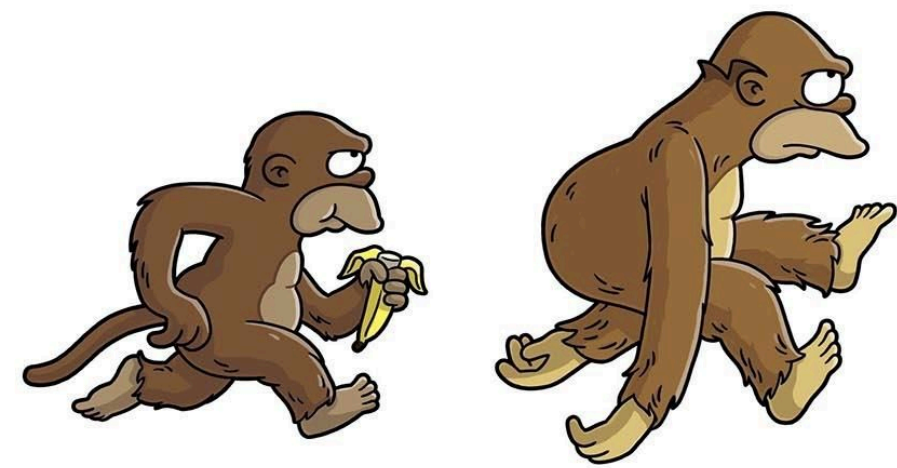
## 1990s



1991

# "Irrational Exuberance" Languages

## 1990s



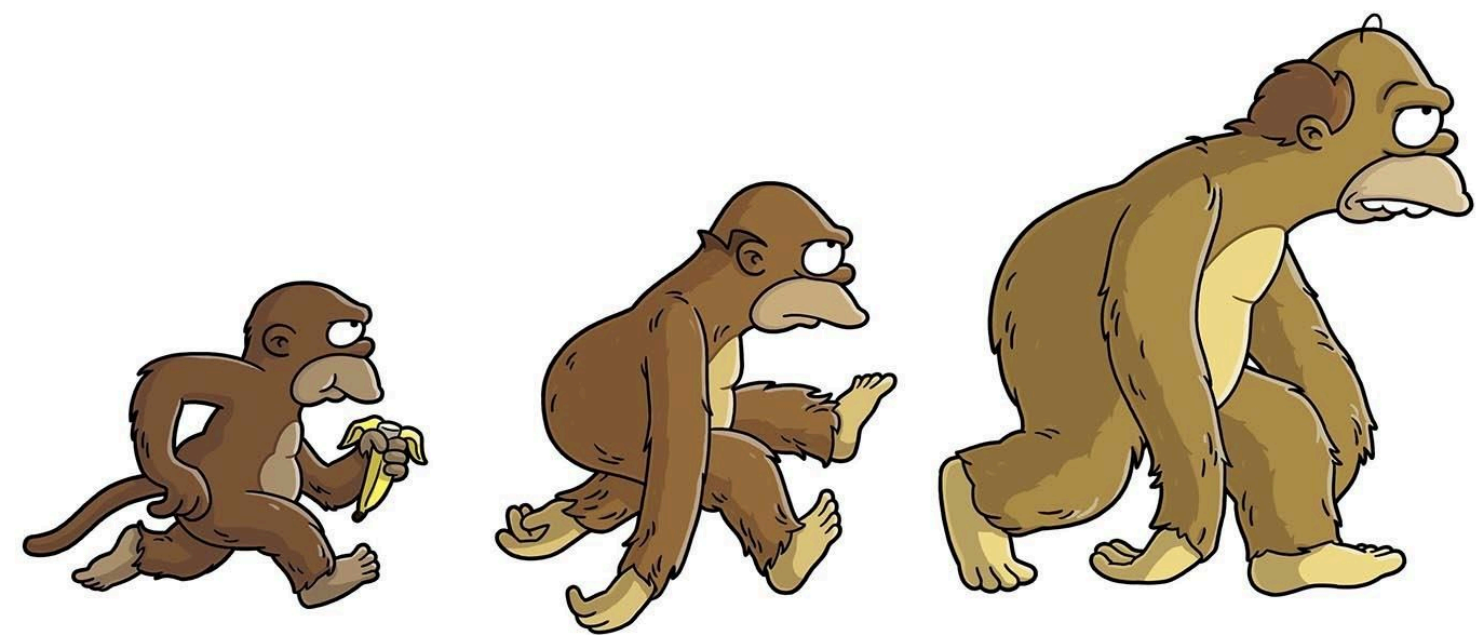
1991



1993

# "Irrational Exuberance" Languages

1990s



1991



1993

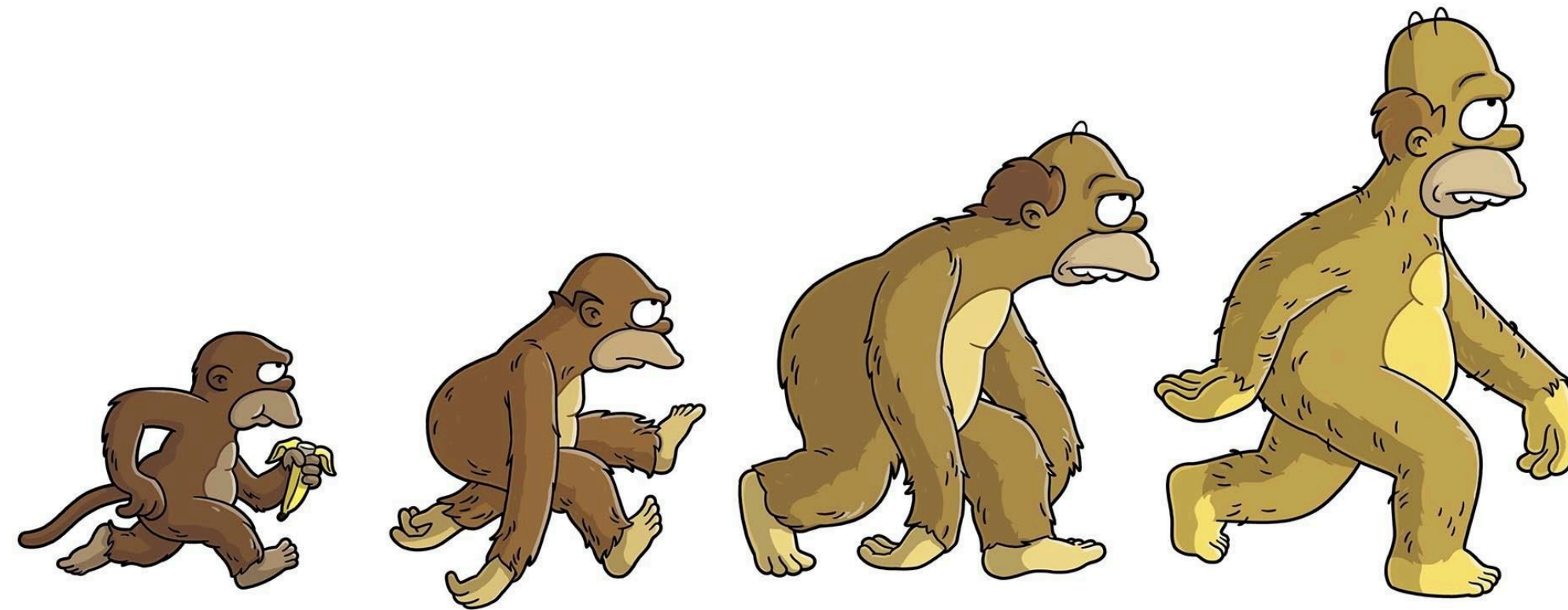


1995



# "Irrational Exuberance" Languages

1990s



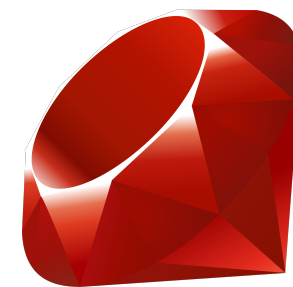
1991



1993



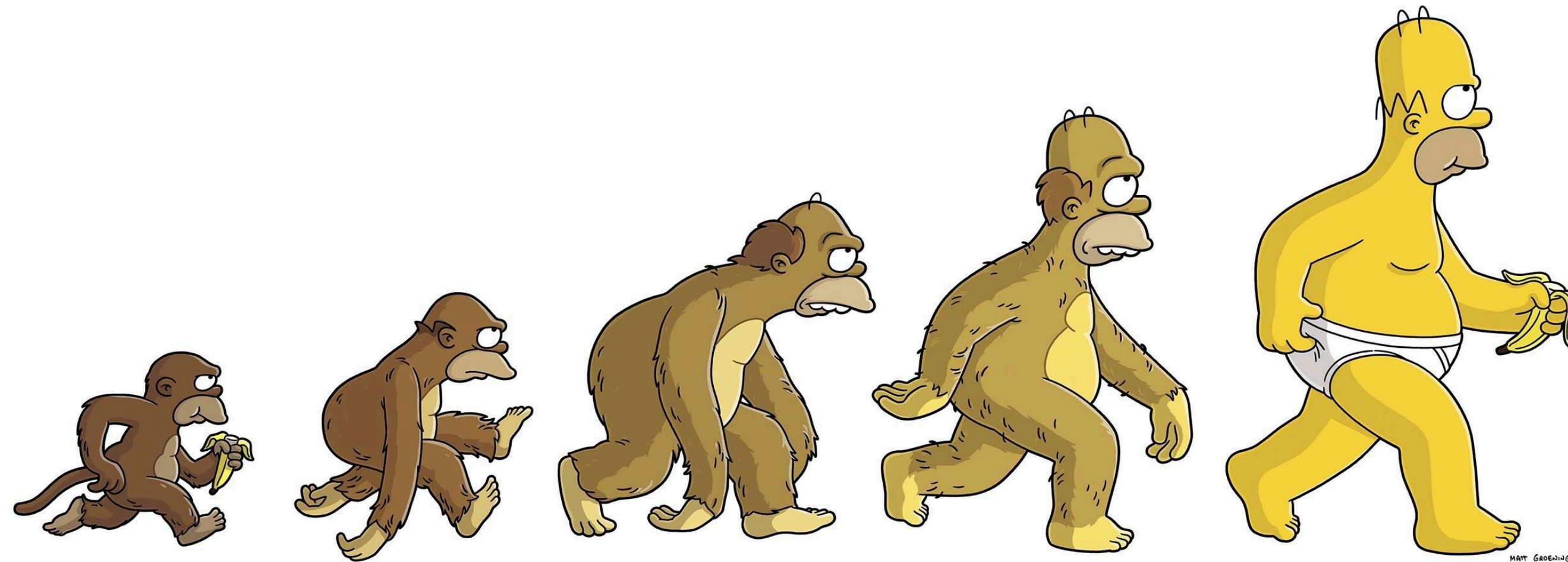
1995



1995

# "Irrational Exuberance" Languages

## 1990s



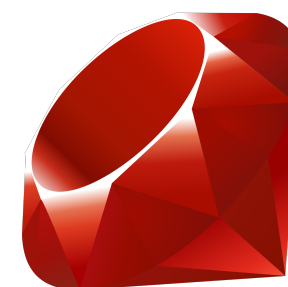
1991



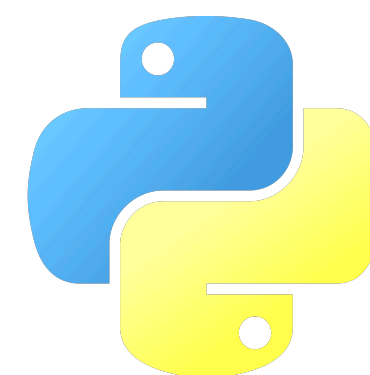
1993



1995

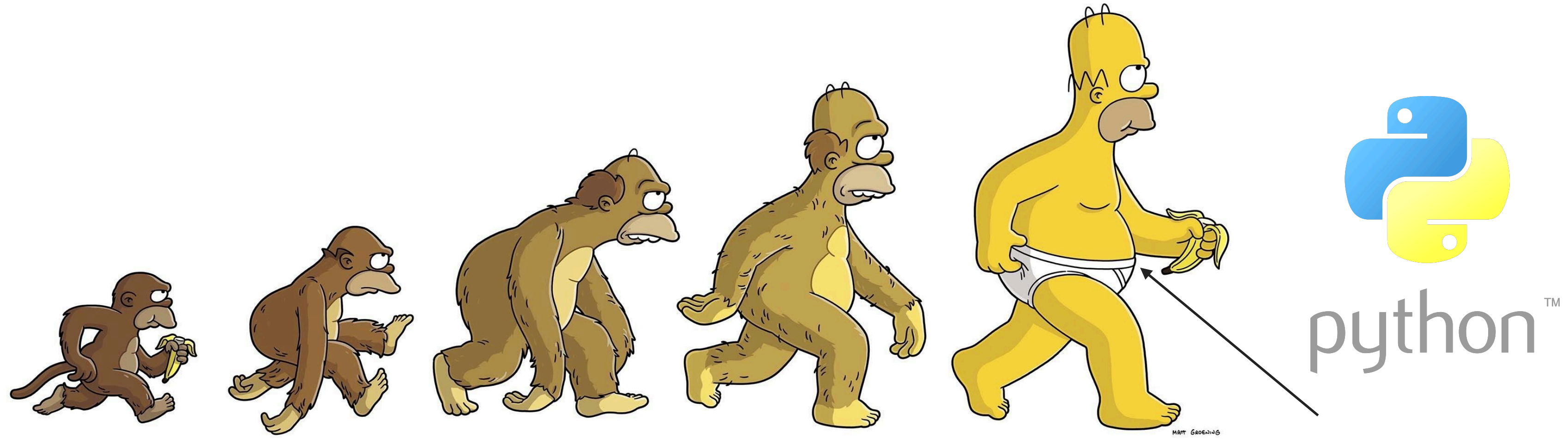


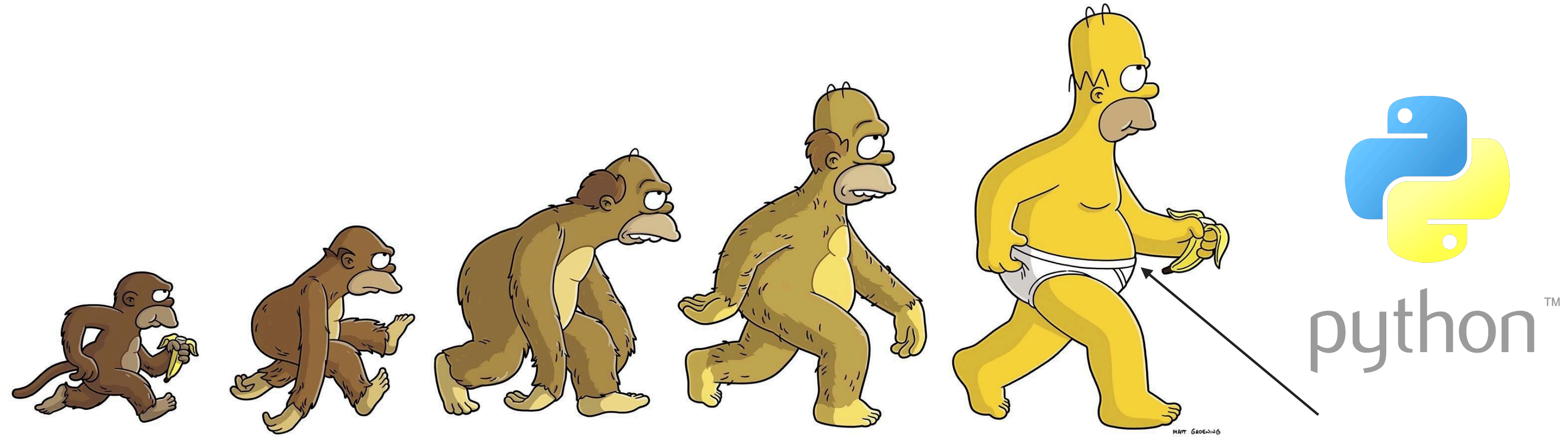
1995



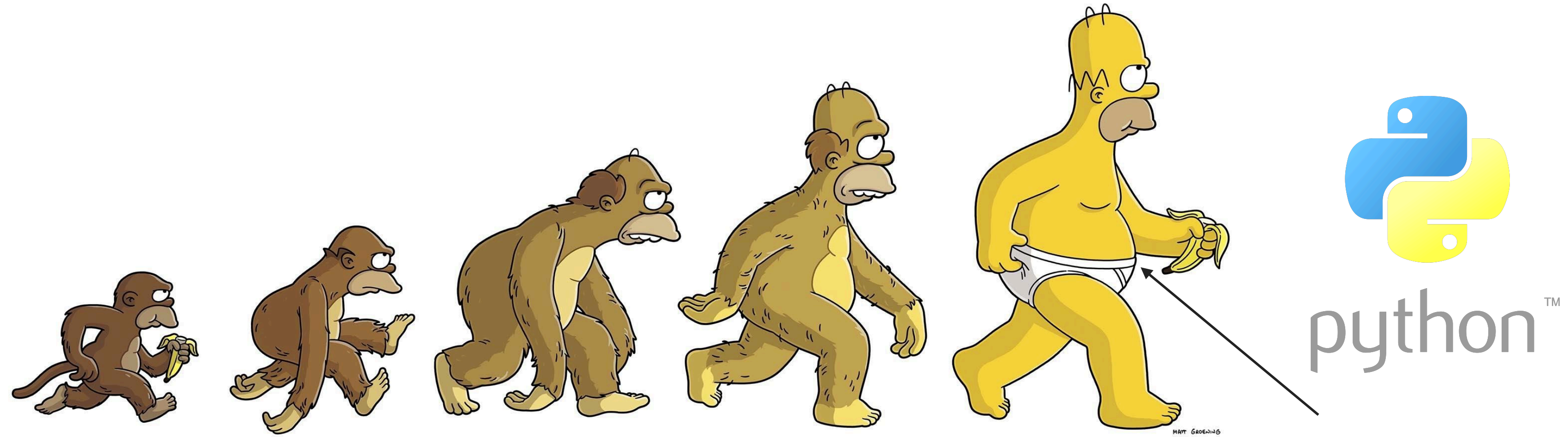
python™

1990



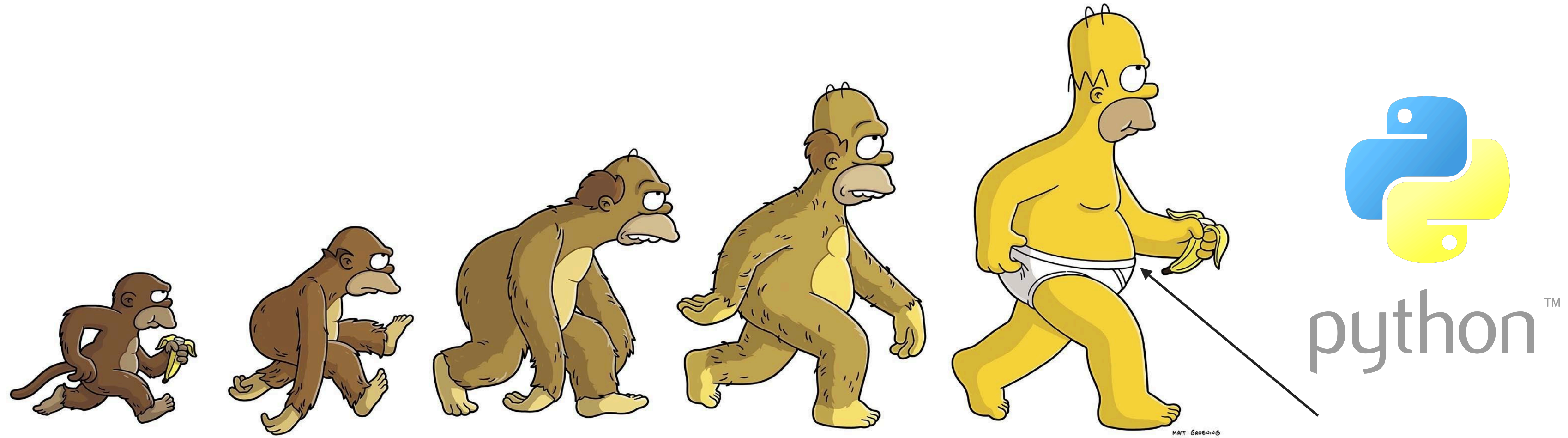


HOW MANY BYTES IN...(INT, LIST, DICT)?



HOW MANY BYTES IN...(INT, LIST, DICT)?

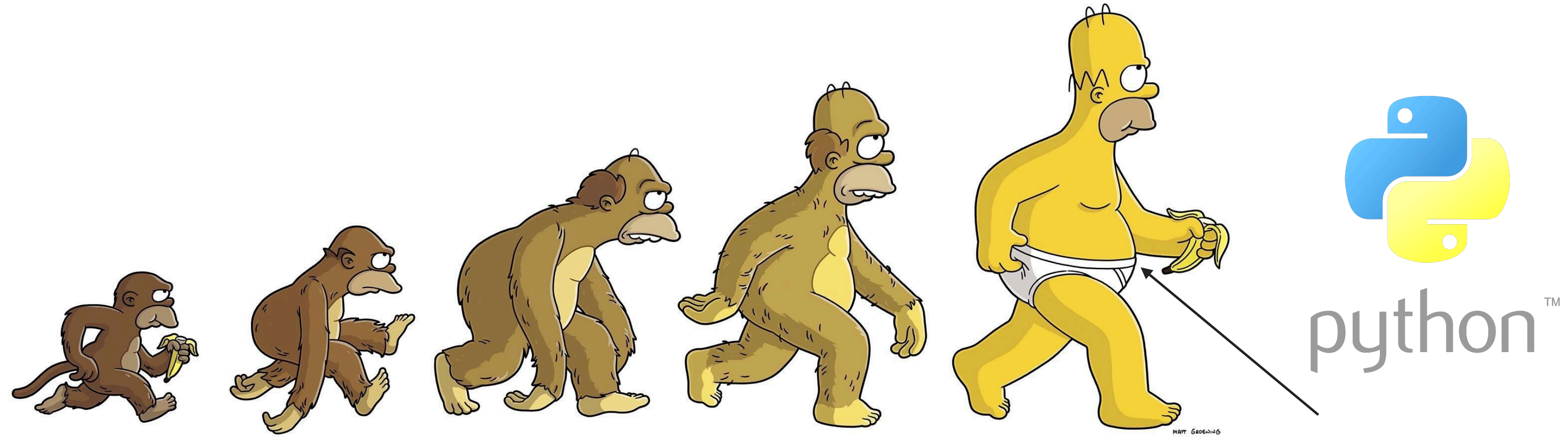




HOW MANY BYTES IN...(INT, LIST, DICT)?



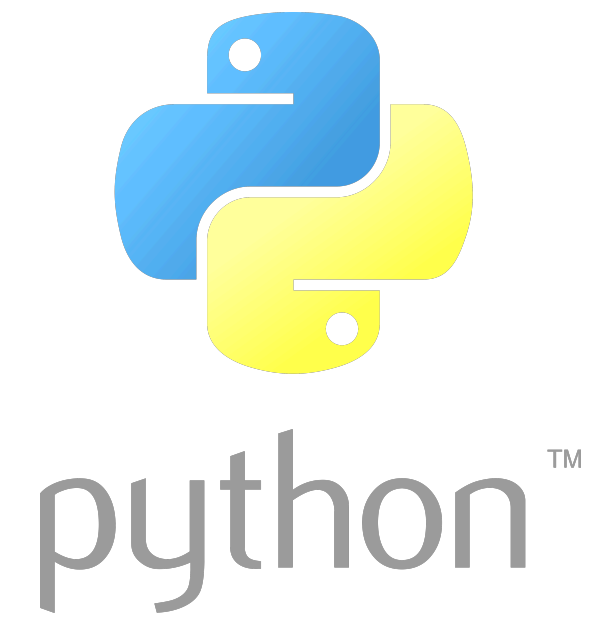
```
sizeof(1)  
→ 4
```

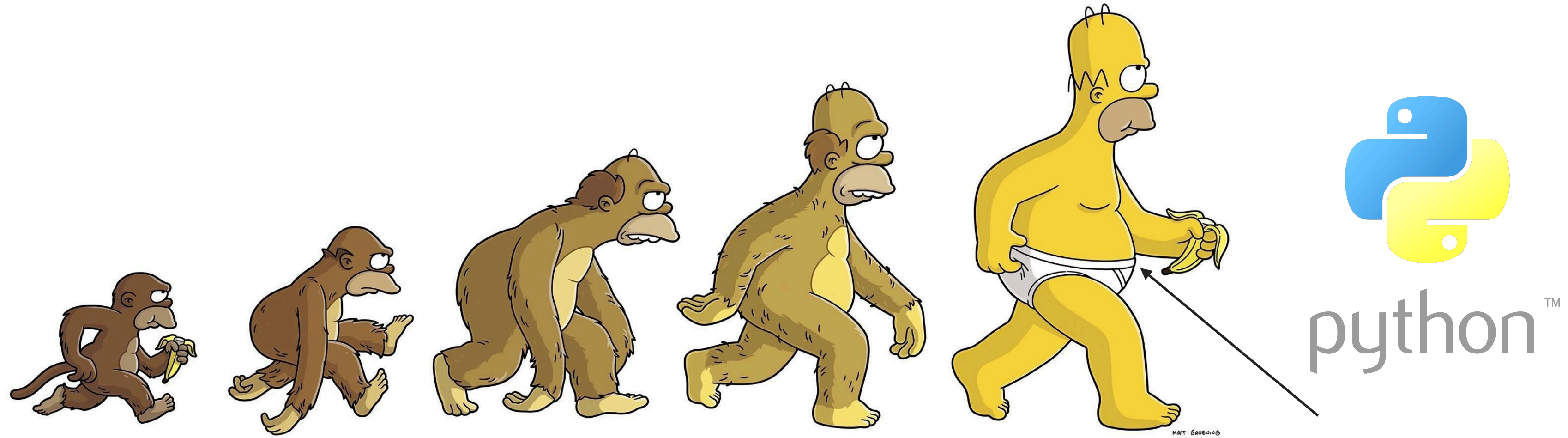


HOW MANY BYTES IN...(INT, LIST, DICT)?



```
sizeof(1)  
→ 4
```





## HOW MANY BYTES IN...(INT, LIST, DICT)?

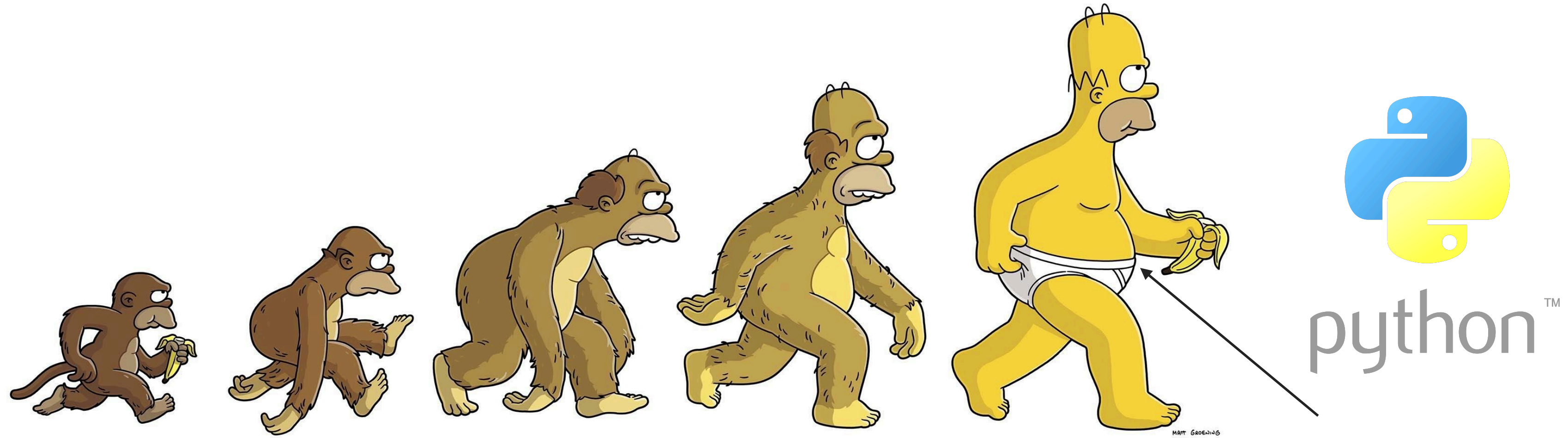


```
sizeof(1)  
→ 4
```



```
>>> sys.getsizeof(1)  
28
```





## HOW MANY BYTES IN...(INT, LIST, DICT)?

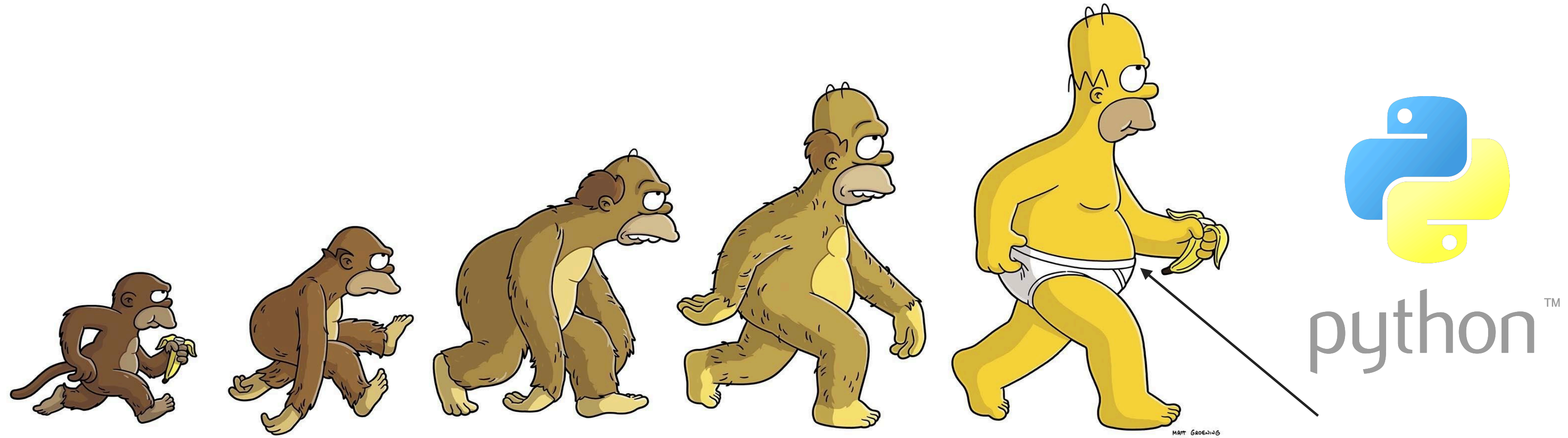


```
sizeof(1)  
→ 4
```

```
sizeof(list<int>)  
→ 24
```



```
>>> sys.getsizeof(1)  
28
```



## HOW MANY BYTES IN...(INT, LIST, DICT)?



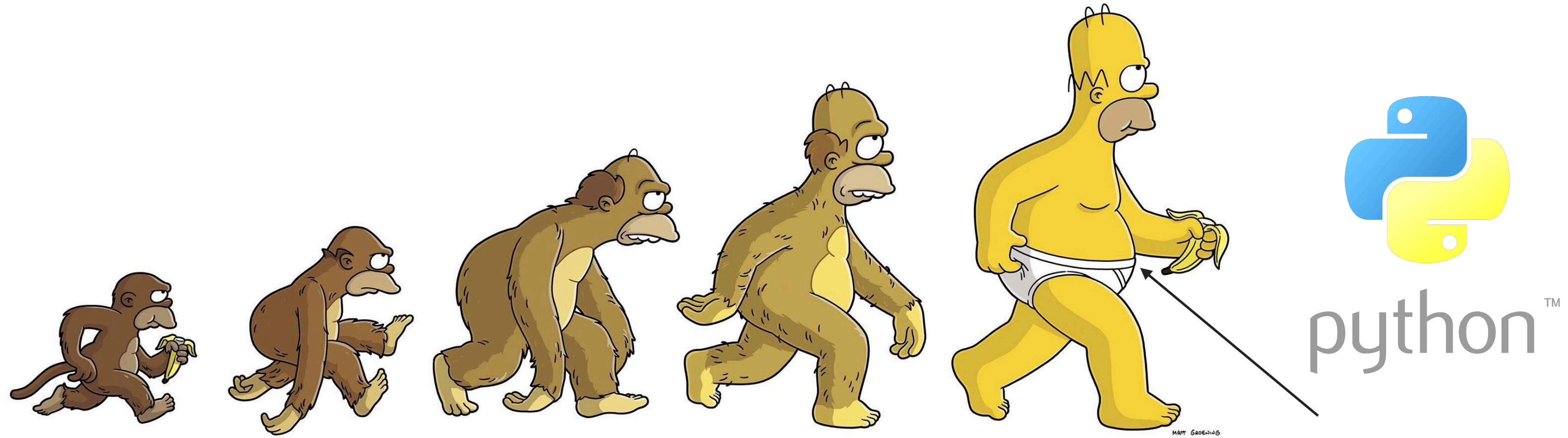
```
sizeof(1)  
→ 4
```

```
sizeof(list<int>)  
→ 24
```



```
>>> sys.getsizeof(1)  
28
```

```
>>> sys.getsizeof([])  
56
```



## HOW MANY BYTES IN...(INT, LIST, DICT)?



```
sizeof(1)  
→ 4
```

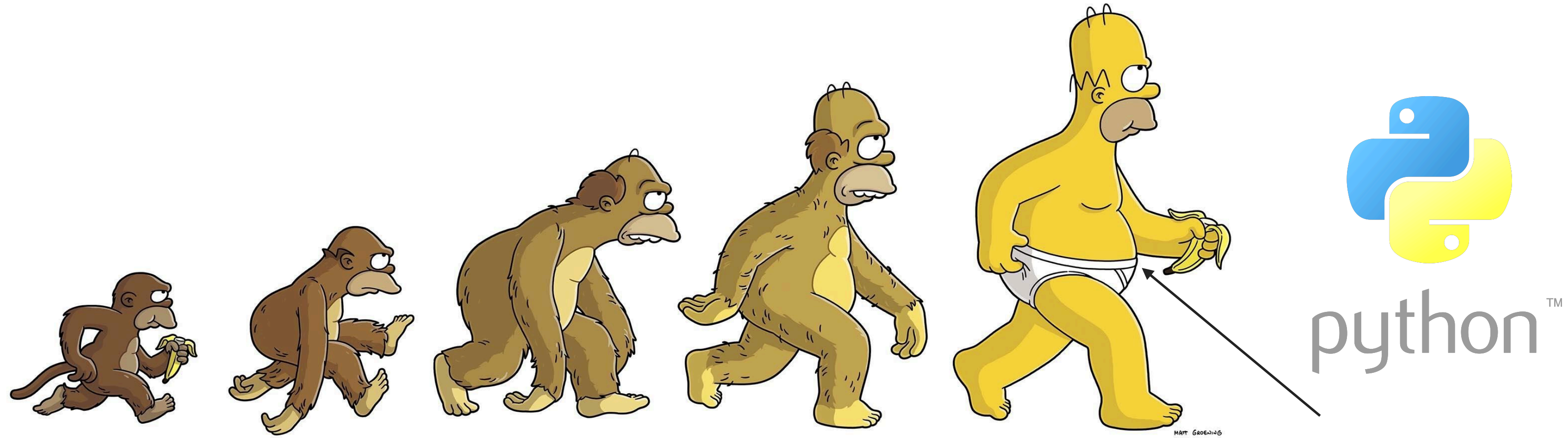
```
sizeof(list<int>)  
→ 24
```

```
sizeof(map<int,  
int>)  
→ 24
```



```
>>> sys.getsizeof(1)  
28
```

```
>>> sys.getsizeof([])  
56
```



## HOW MANY BYTES IN...(INT, LIST, DICT)?



```
sizeof(1)  
→ 4
```

```
sizeof(list<int>)  
→ 24
```

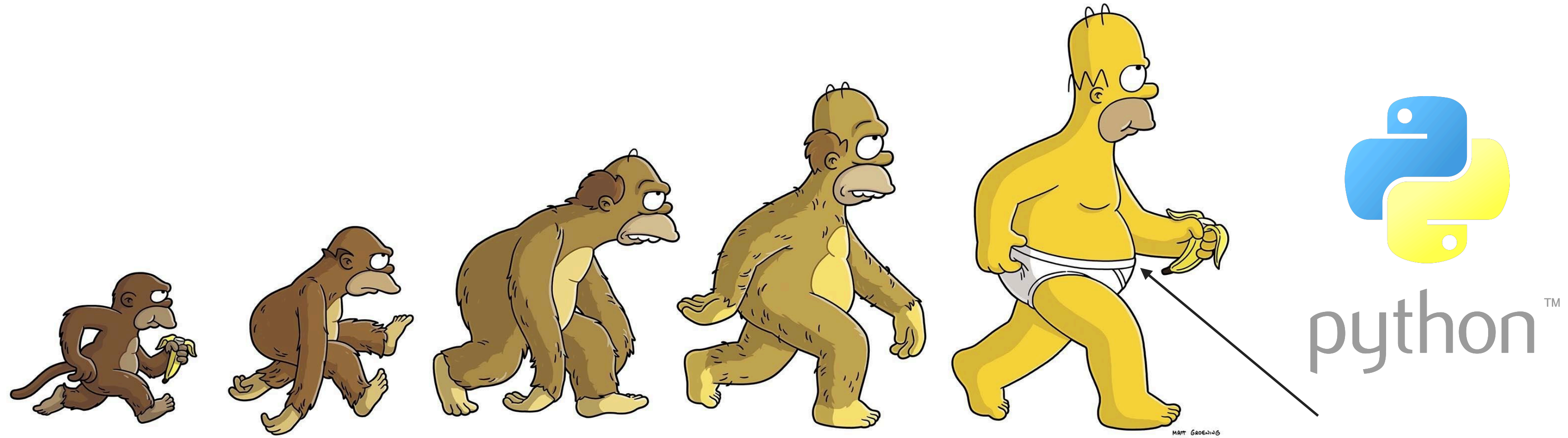
```
sizeof(map<int,  
int>)  
→ 24
```



```
>>> sys.getsizeof(1)  
28
```

```
>>> sys.getsizeof([])  
56
```

```
>>> sys.getsizeof({})  
64
```



## HOW MANY BYTES IN...(INT, LIST, DICT)?



```
sizeof(1)  
→ 4
```

```
sizeof(list<int>)  
→ 24
```

```
sizeof(map<int,  
int>)  
→ 24
```

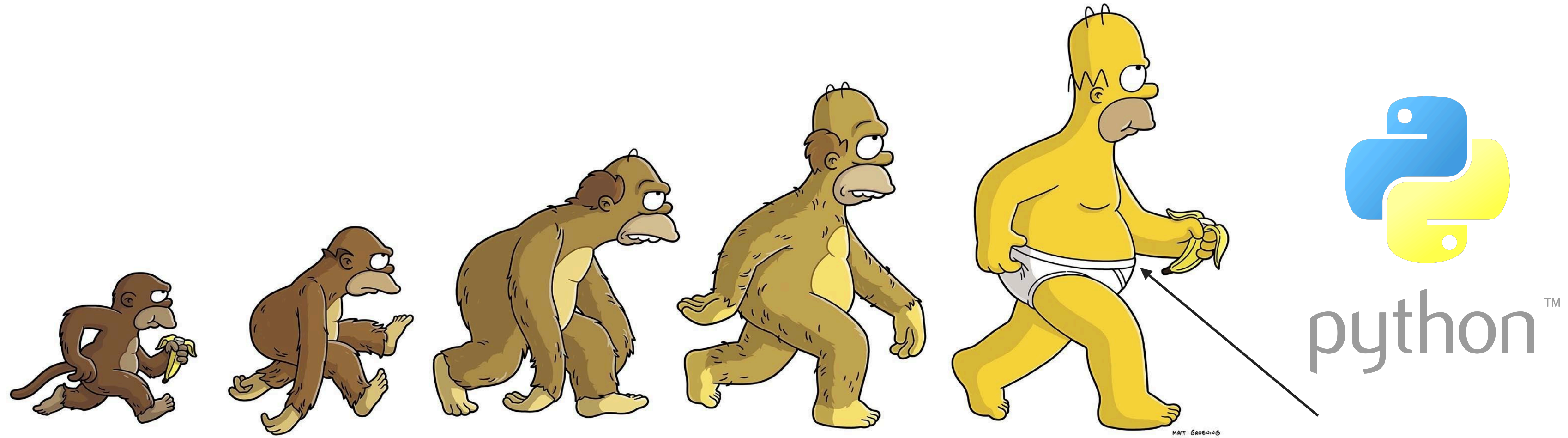


```
>>> sys.getsizeof(1)  
28
```

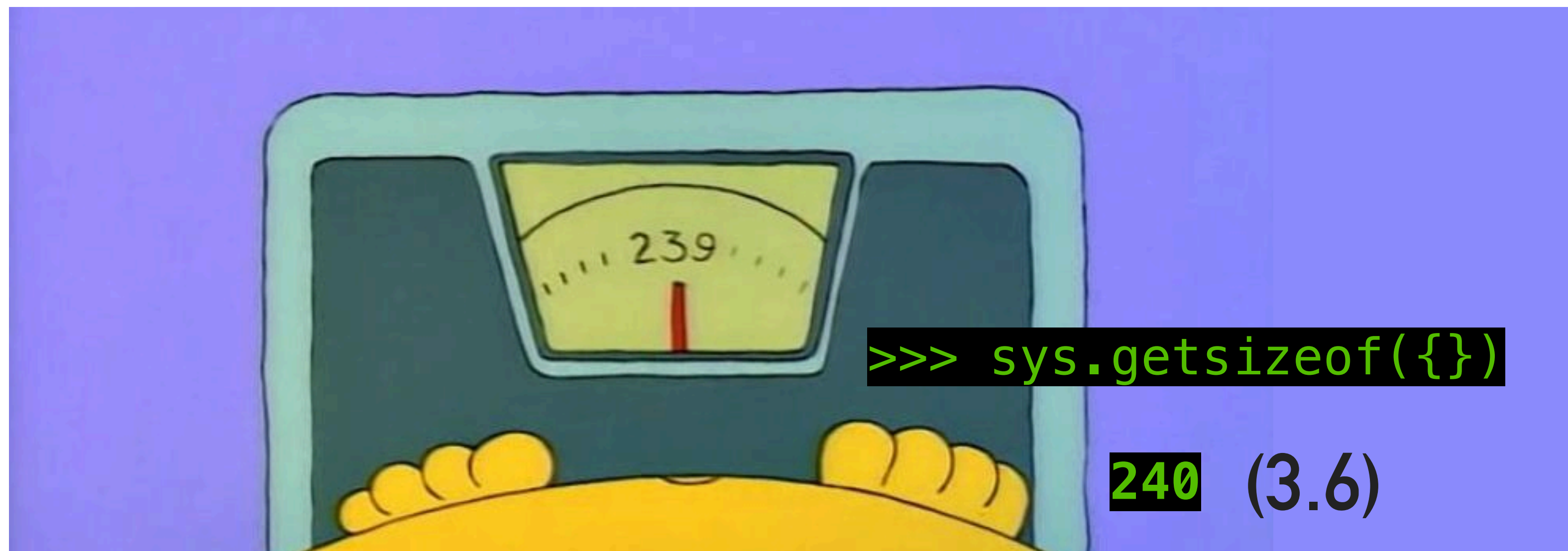
```
>>> sys.getsizeof([])  
56
```

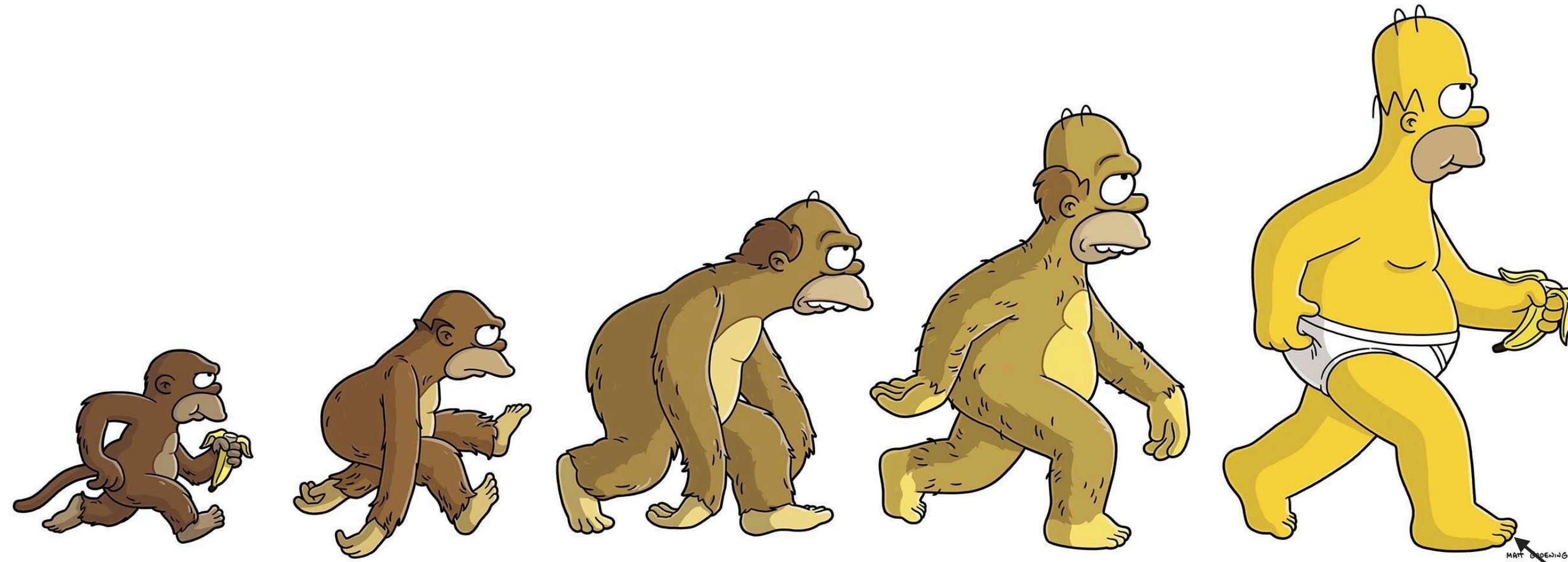
```
>>> sys.getsizeof({})
```

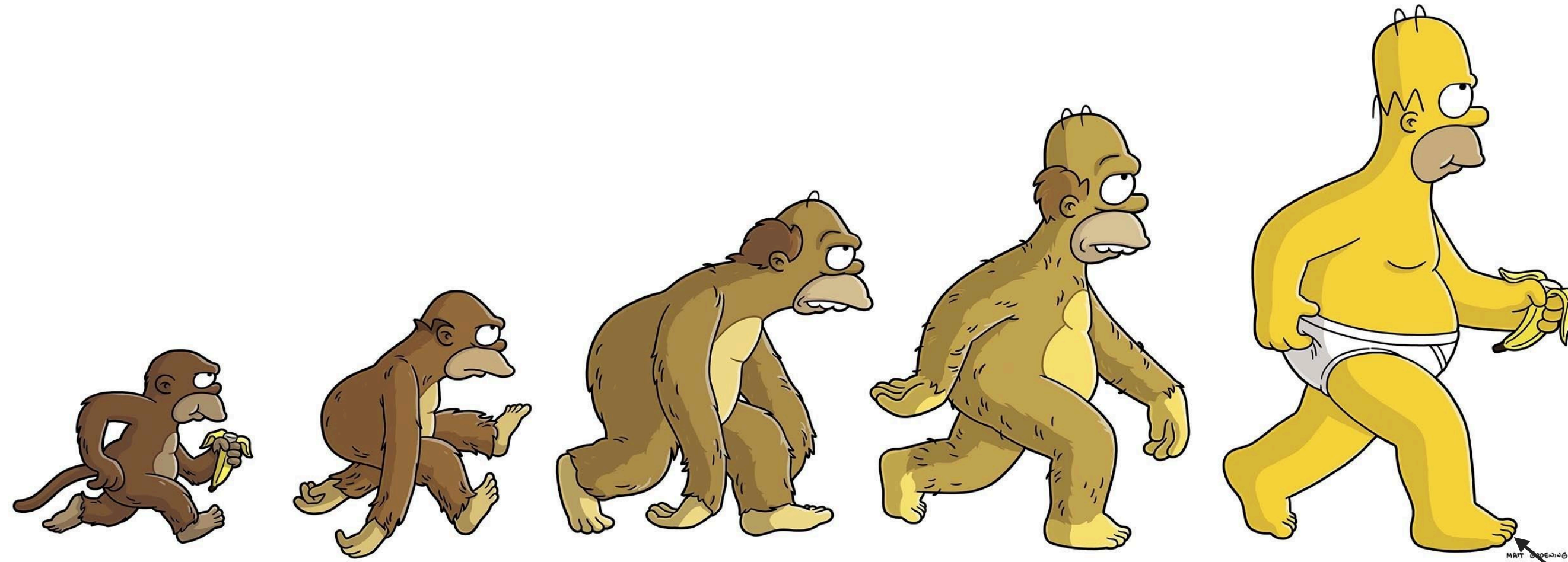
**240** (3.6)



HOW MANY BYTES IN...(INT, LIST, DICT)?





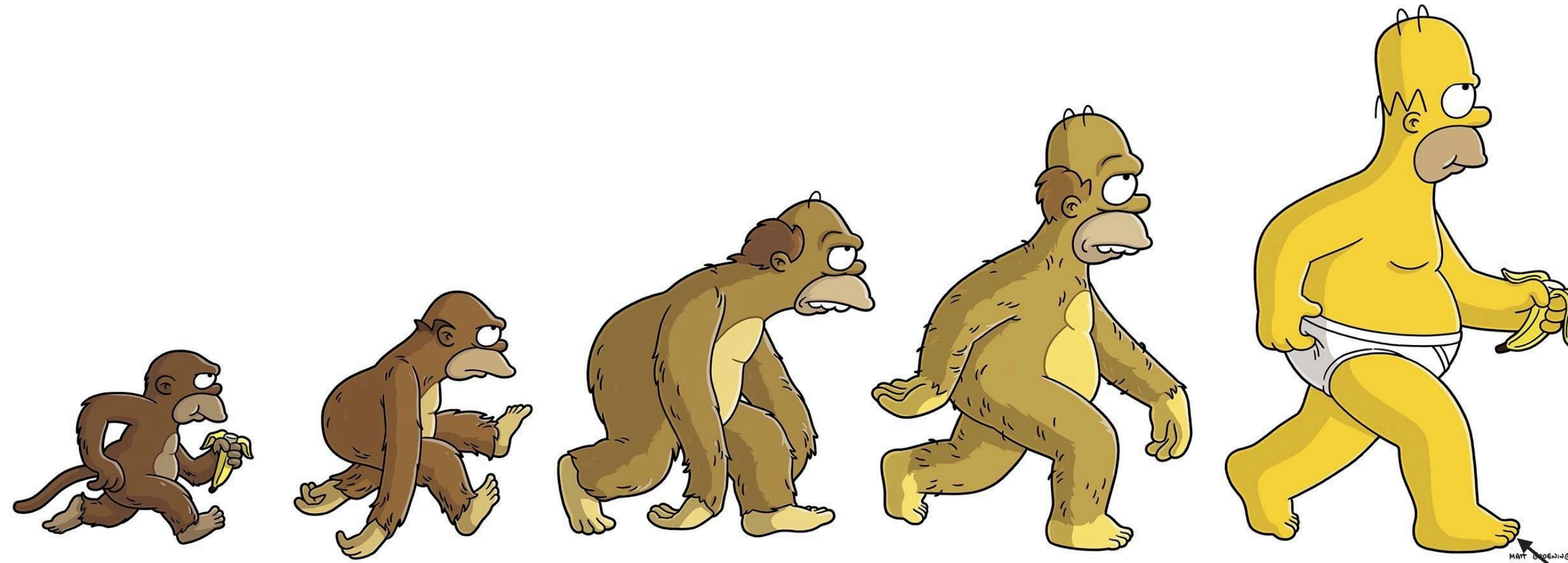


MATRIX-  
MATRIX  
MULTIPLY

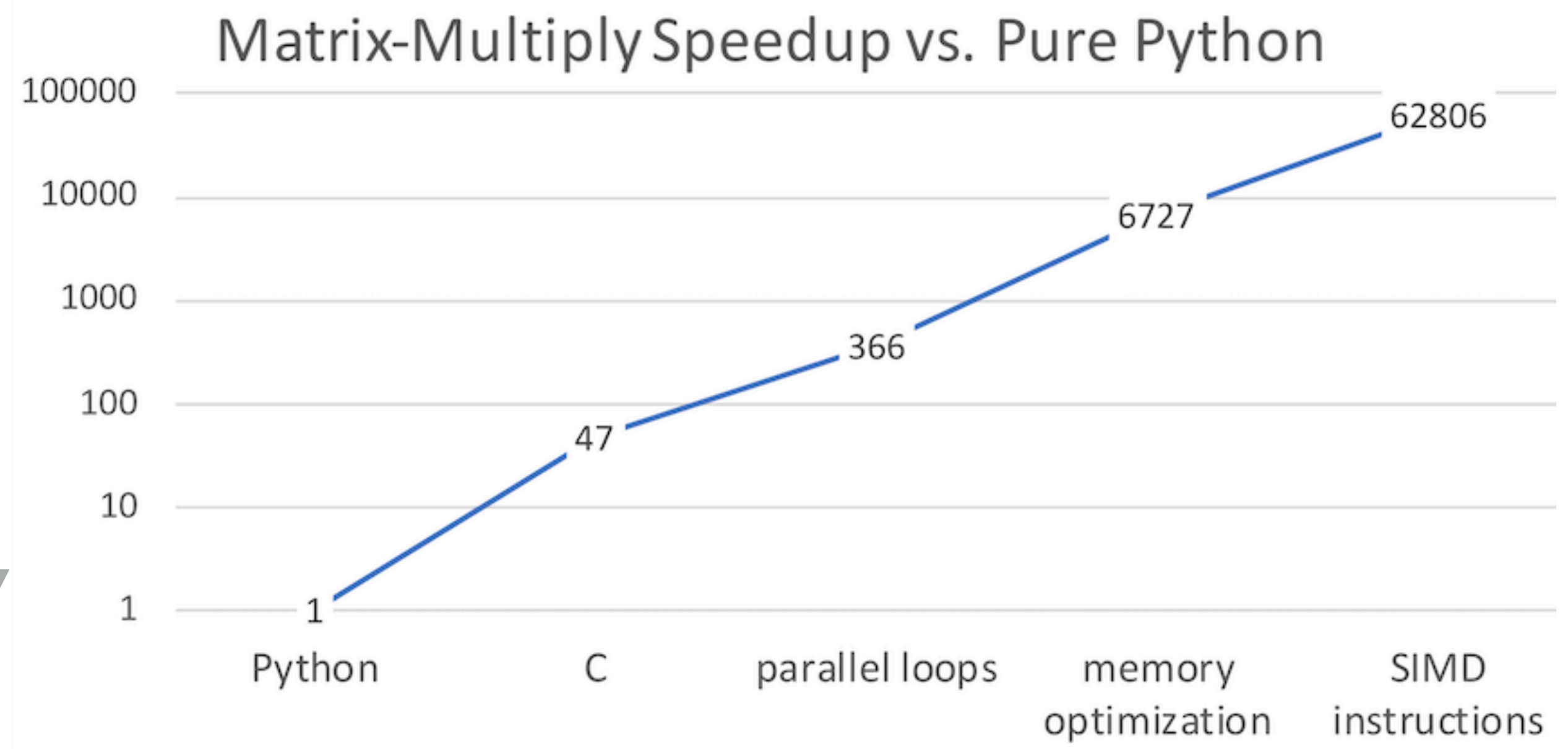
```
for i in range(n):  
    for j in range(n):  
        for k in range(n):  
            C[i][j] += A[i][k] * B[k][j]
```



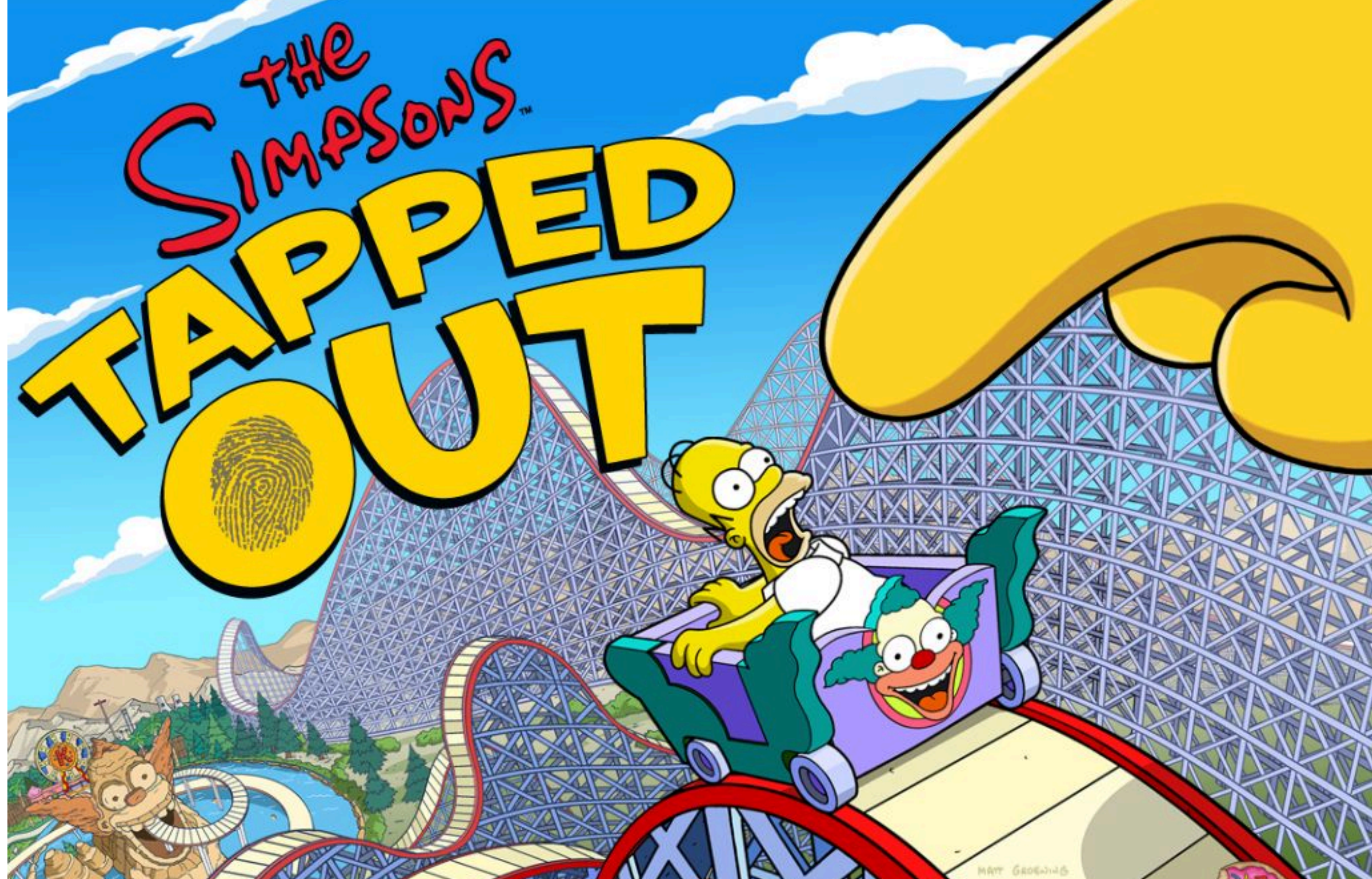




MATRIX-  
MATRIX  
MULTIPLY

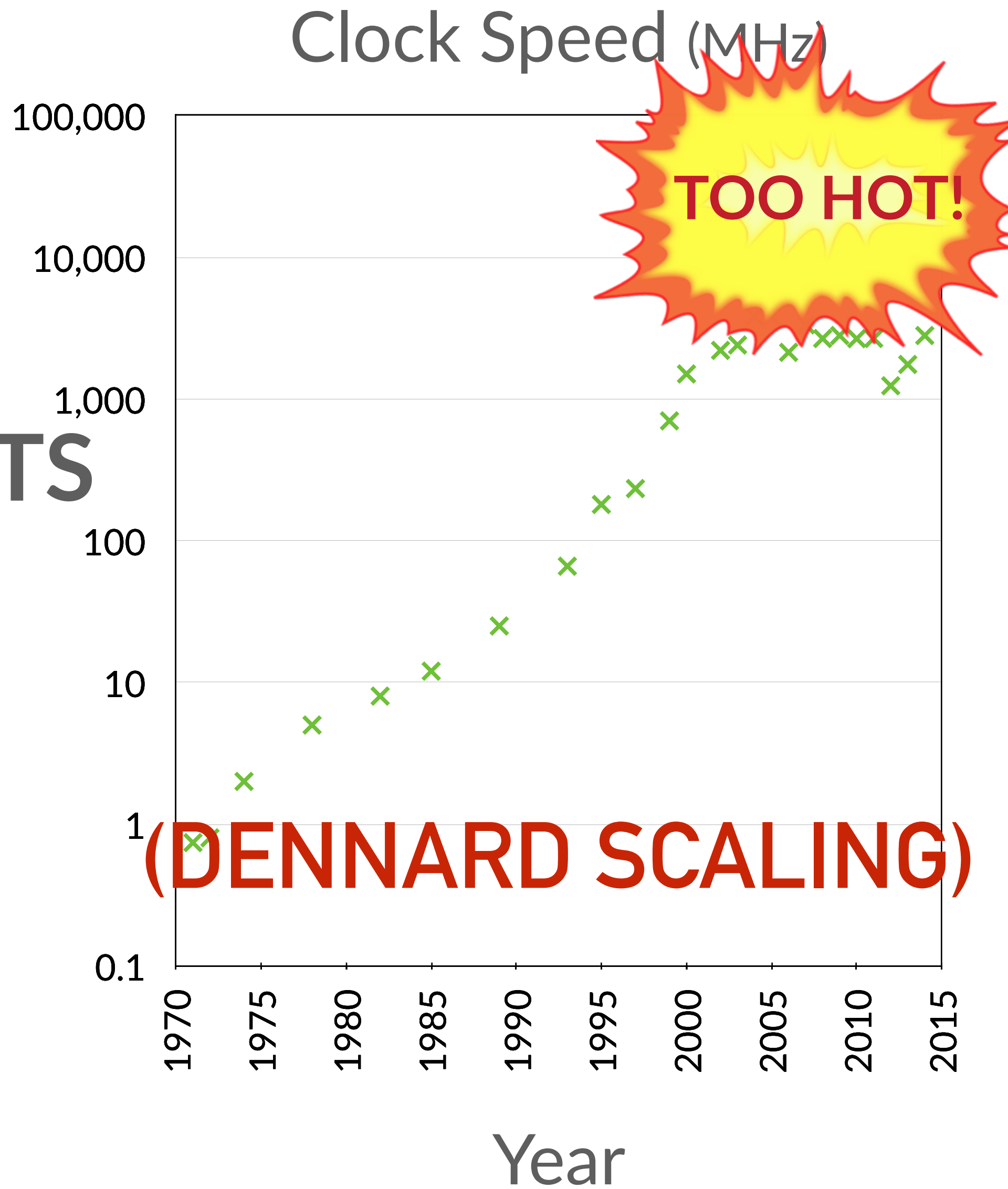
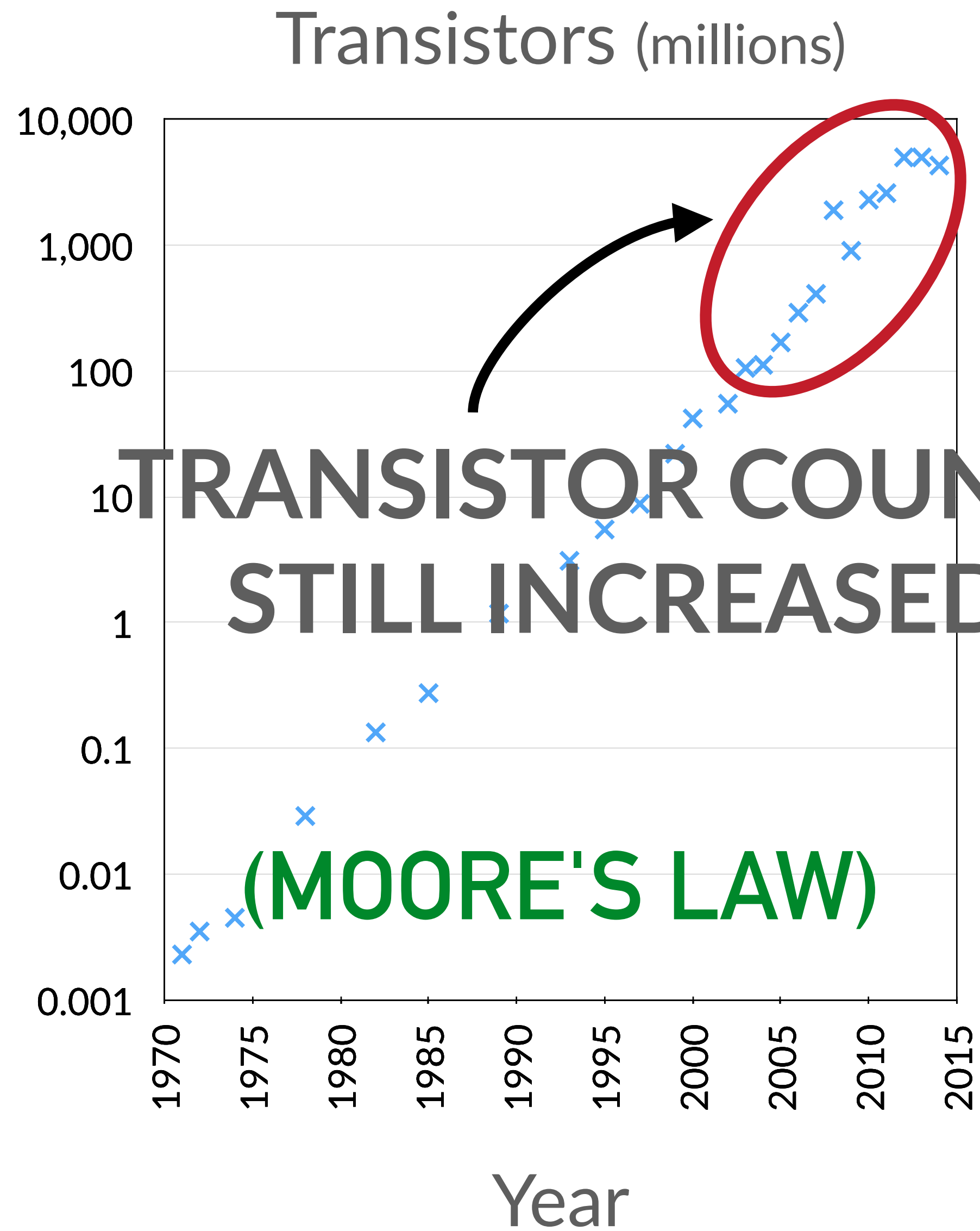


60,000X  
slowdown!



≈2010: THE RIDE IS OVER

# The Ride Is Over





```

import numpy as np

def main():
    for i in range(10):
        x = np.array(range(10**7))

```

76999 function calls (74718 primitive calls) in 6.307 seconds

Ordered by: cumulative time

# python3 -m cProfile

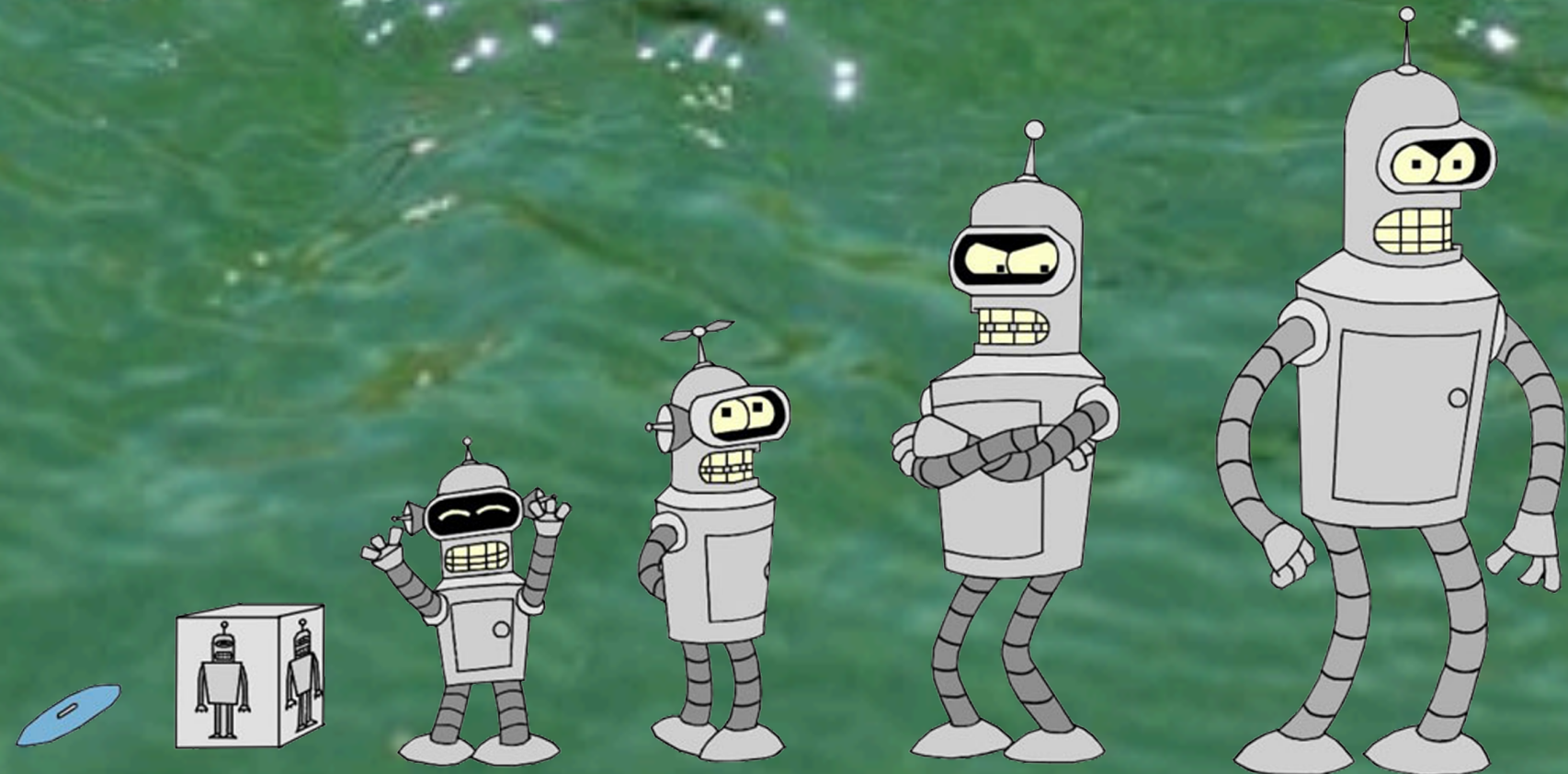
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
433/1	0.000	0.000	6.307	6.307	{built-in method builtins.exec}
1	0.022	0.022	6.307	6.307	test2-2.py:2(<module>)
1	0.155	0.155	6.216	6.216	test2-2.py:4(main)
131	3.191	0.024	3.191	0.024	{built-in method numpy.array}
6	2.870	0.478	2.870	0.478	{method 'uniform' of 'numpy.random.mtrand.RandomState' objects}
13	0.000	0.000	0.146	0.011	__init__.py:1(<module>)
156/1	0.000	0.000	0.069	0.069	<frozen importlib._bootstrap>:1002(_find_and_load)
156/1	0.000	0.000	0.069	0.069	<frozen importlib._bootstrap>:967(_find_and_load_unlocked)
145/1	0.000	0.000	0.069	0.069	<frozen importlib._bootstrap>:659(_load_unlocked)
112/1	0.000	0.000	0.069	0.069	<frozen importlib._bootstrap_external>:784(exec_module)
224/1	0.000	0.000	0.069	0.069	<frozen importlib._bootstrap>:220(_call_with_frames_removed)
179/16	0.000	0.000	0.066	0.004	<frozen importlib._bootstrap>:1033(_handle_fromlist)
326/9	0.000	0.000	0.066	0.007	{built-in method builtins.__import__}
112	0.000	0.000	0.024	0.000	<frozen importlib._bootstrap_external>:856(get_code)
112	0.000	0.000	0.016	0.000	<frozen importlib._bootstrap_external>:976(get_data)
112	0.014	0.000	0.014	0.000	{method 'read' of '_io.BufferedReader' objects}
145/142	0.000	0.000	0.011	0.000	<frozen importlib._bootstrap>:558(module_from_spec)
1	0.000	0.000	0.010	0.010	multiarray.py:1(<module>)
320	0.000	0.000	0.010	0.000	overrides.py:187(decorator)
32/30	0.000	0.000	0.010	0.000	<frozen importlib._bootstrap_external>:1106(create_module)
32/30	0.008	0.000	0.010	0.000	{built-in method _imp.create_dynamic}
1	0.000	0.000	0.009	0.009	overrides.py:1(<module>)
1	0.000	0.000	0.008	0.008	_pickle.py:1(<module>)
153	0.000	0.000	0.007	0.000	<frozen importlib._bootstrap>:901(_find_spec)



PYTHON



PYTHON



THE "C"

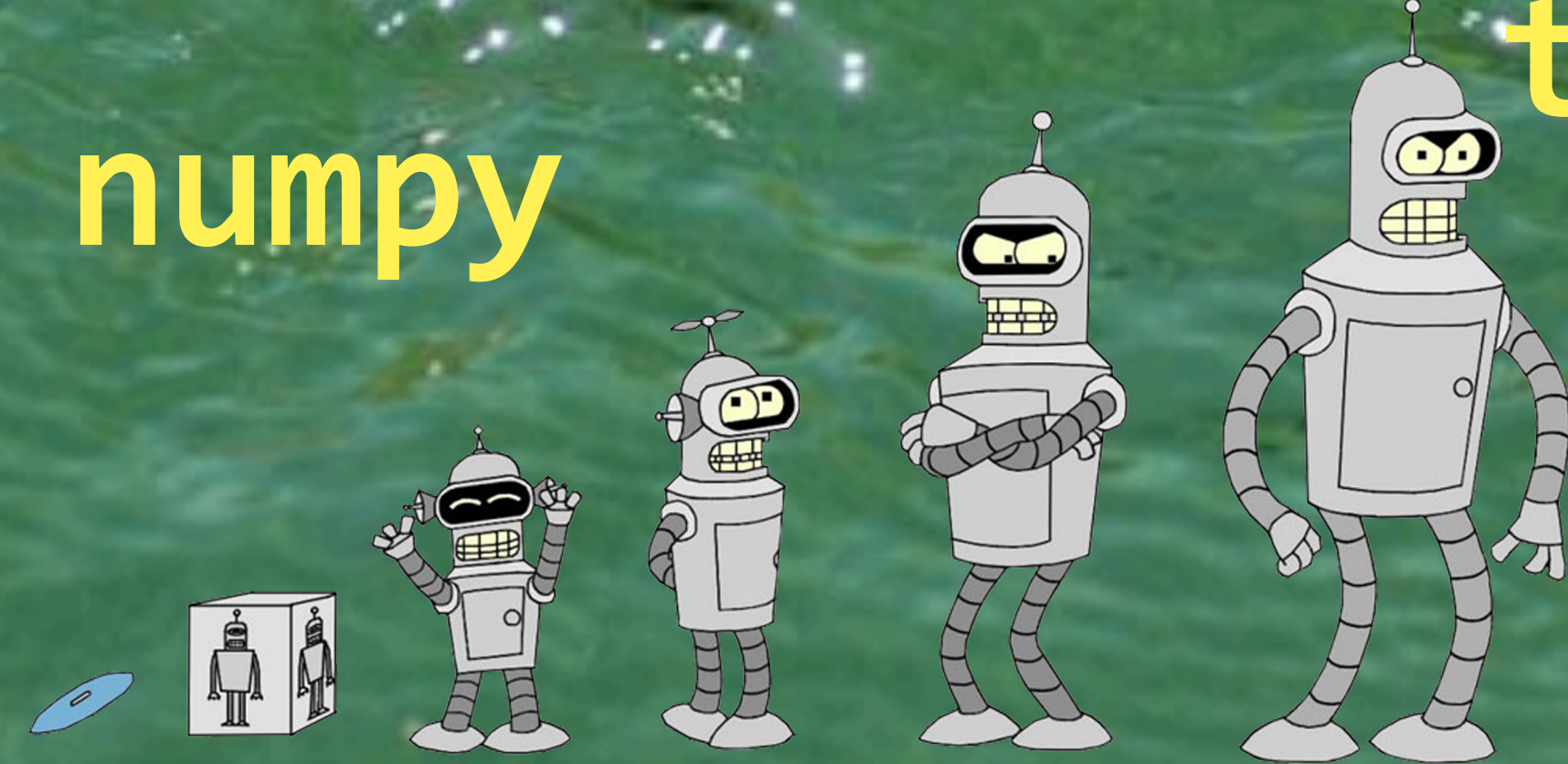


# PYTHON

scikit

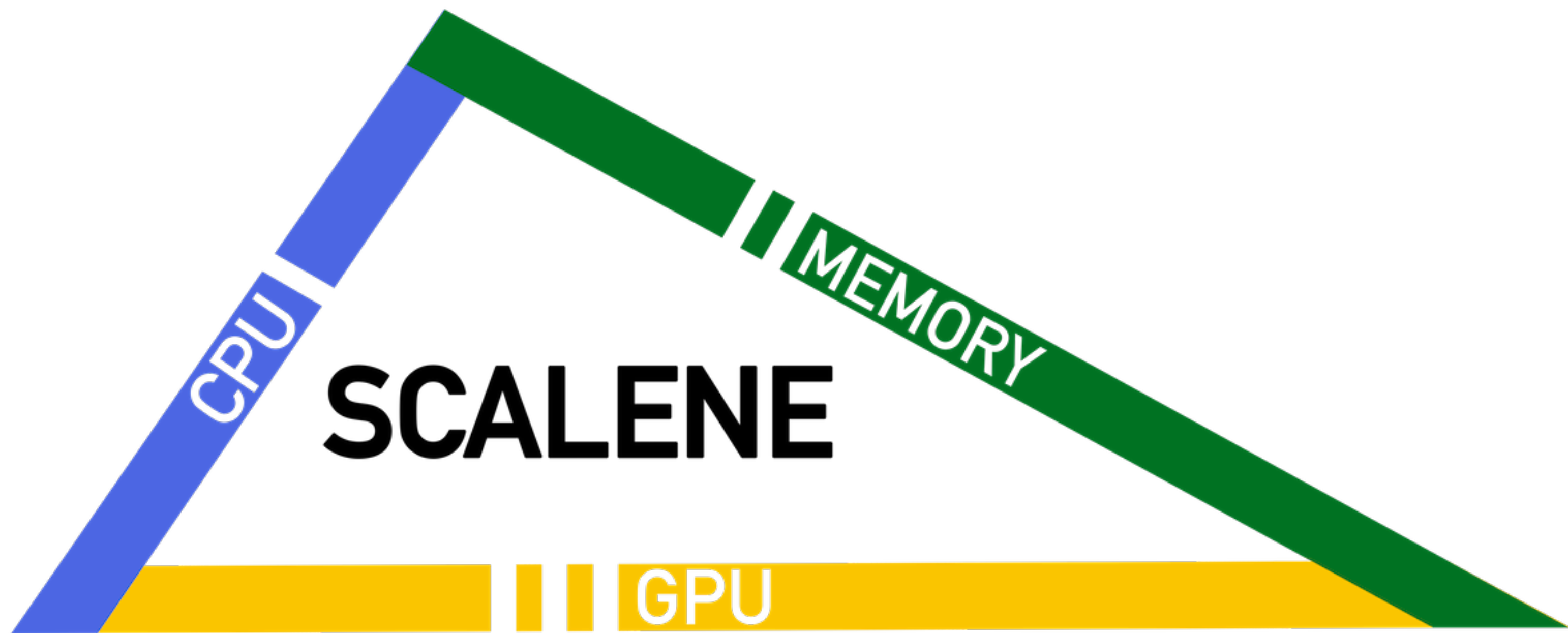
tensorflow

numpy



# THE "C"





---

```
import numpy as np
```

---

```
def main():
```

```
    for i in range(10):
```

```
        x = np.array(range(10**7))
```

```
        y = np.array(np.random.uniform(0, 100, size=(10**8)))
```

Time: Python | native | system



Memory: Python | native



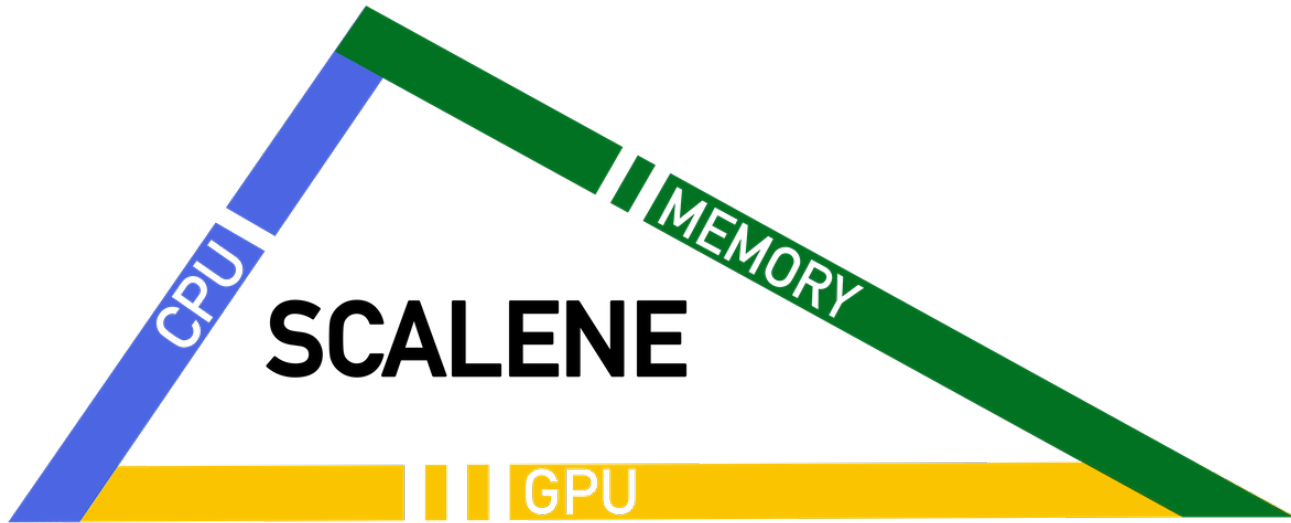
Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
				17%				5 for i in range(10):
								6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)

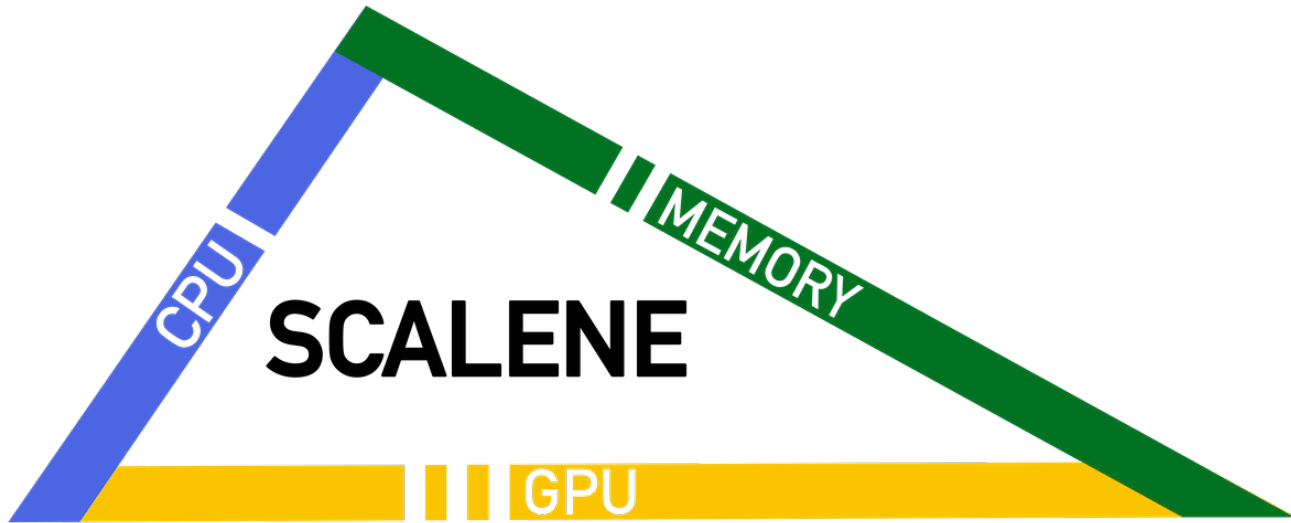


hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

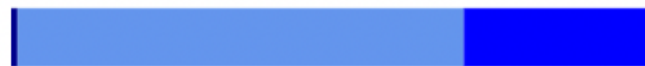
./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
				17%				2 import numpy as np
				83%	253			4 def main(): 5 for i in range(10): 6 x = np.array(range(10**7)) 7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

CPU  
 PYTHON  
 NATIVE  
 SYS%



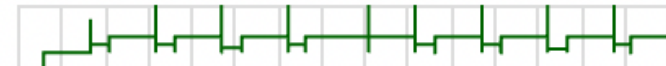
Time: Python | native | system



Memory: Python | native



Memory timeline: (max: 3135.8MB, growth: 3.1%)



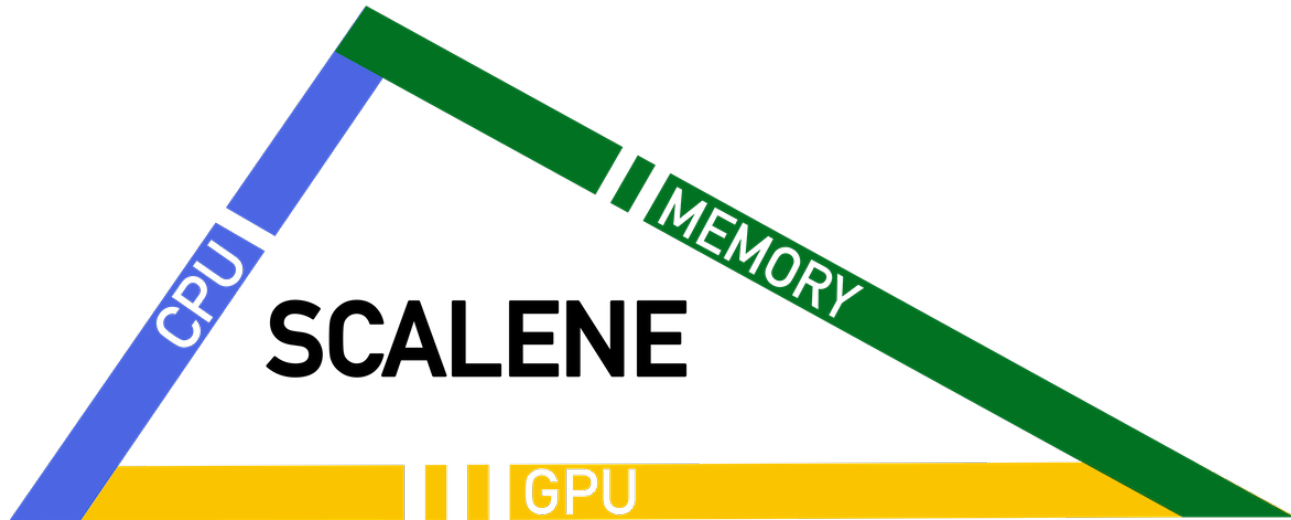
hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY</u> average	<u>MEMORY</u> peak	<u>MEMORY</u> timeline	<u>MEMORY</u> activity	<u>COPY</u> (MB/s)	<u>GPU</u> util.	<u>GPU</u> memory	<u>LINE PROFILE</u> (click to reset order) ./test2-2.py
				17%				2 import numpy as np
				83%	253			4 def main(): 5 for i in range(10): 6 x = np.array(range(10**7)) 7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

CPU  
PYTHON  
NATIVE  
SYS%

MEMORY  
PYTHON  
NATIVE  
AVERAGE &  
PEAK



Time: Python | native | system    Memory: Python | native    Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

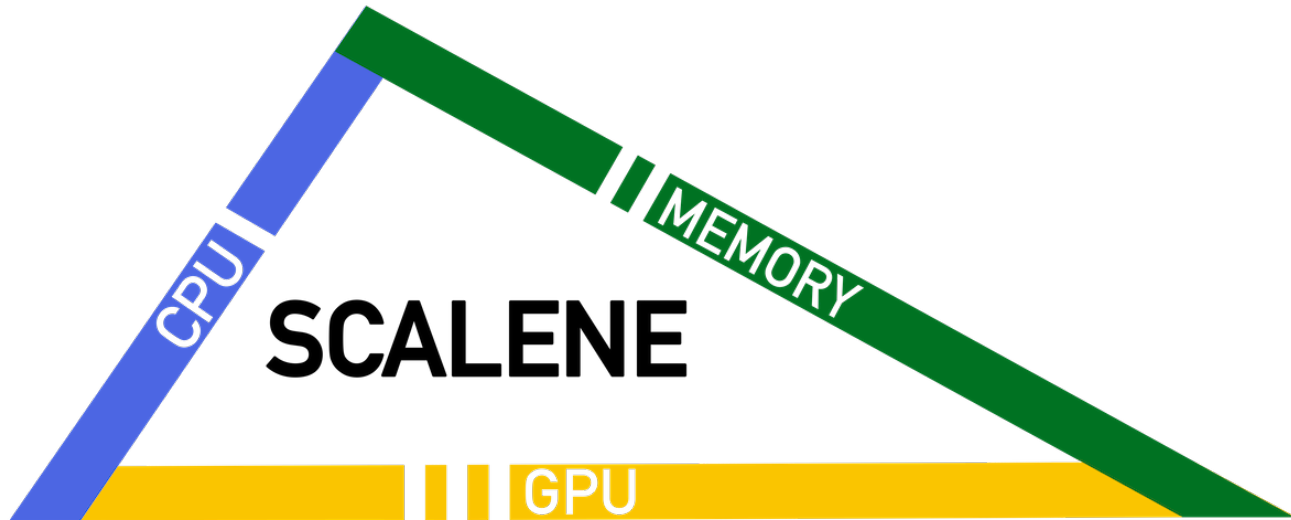
./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
				17%				2 import numpy as np
				83%	253			4 def main(): 5     for i in range(10): 6         x = np.array(range(10**7)) 7         y = np.array(np.random.uniform(0, 100, size=(10**8)))

**CPU  
PYTHON  
NATIVE  
SYS%**

**MEMORY  
PYTHON  
NATIVE  
AVERAGE &  
PEAK**

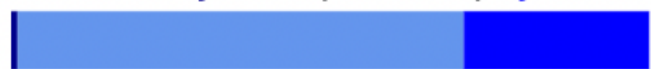
**MEMORY  
USAGE  
OVER TIME,  
% OF MEM  
ALLOCATED**



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

# MEMORY USAGE OVER TIME

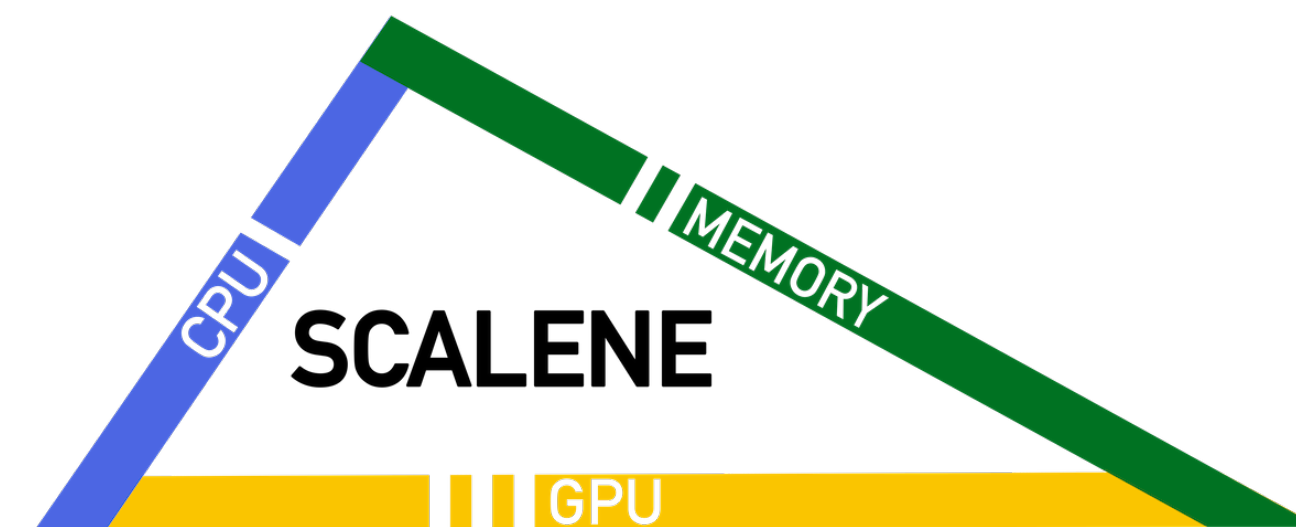
./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
				17%				2 import numpy as np
				83%	253			4 def main(): 5 for i in range(10): 6 x = np.array(range(10**7)) 7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

CPU  
PYTHON  
NATIVE  
SYS%

MEMORY  
PYTHON  
NATIVE  
AVERAGE &  
PEAK

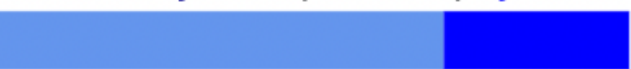
MEMORY  
USAGE  
OVER TIME,  
% OF MEM  
ALLOCATED



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

# MEMORY USAGE OVER TIME

./test2-2.py: % of time = 97.8% out of 30.1s.

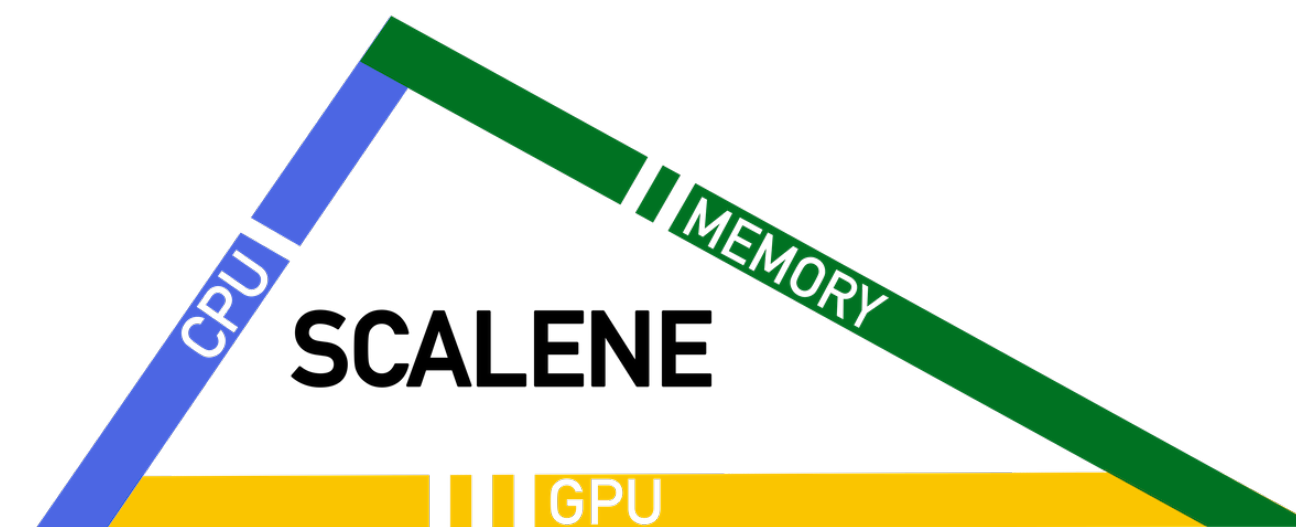
<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
				17% 83%	253			./test2-2.py 2 import numpy as np 4 def main(): 5     for i in range(10): 6         x = np.array(range(10**7)) 7         y = np.array(np.random.uniform(0, 100, size=(10**8)))

**CPU  
PYTHON  
NATIVE  
SYS%**

**MEMORY  
PYTHON  
NATIVE  
AVERAGE &  
PEAK**

**MEMORY  
USAGE  
OVER TIME,  
% OF MEM  
ALLOCATED**

**COPY  
VOLUME  
(MB/s)**

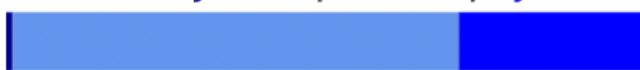




Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

# MEMORY USAGE OVER TIME

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
				17% 83%	253			./test2-2.py 2 import numpy as np 4 def main(): 5     for i in range(10): 6         x = np.array(range(10**7)) 7         y = np.array(np.random.uniform(0, 100, size=(10**8)))

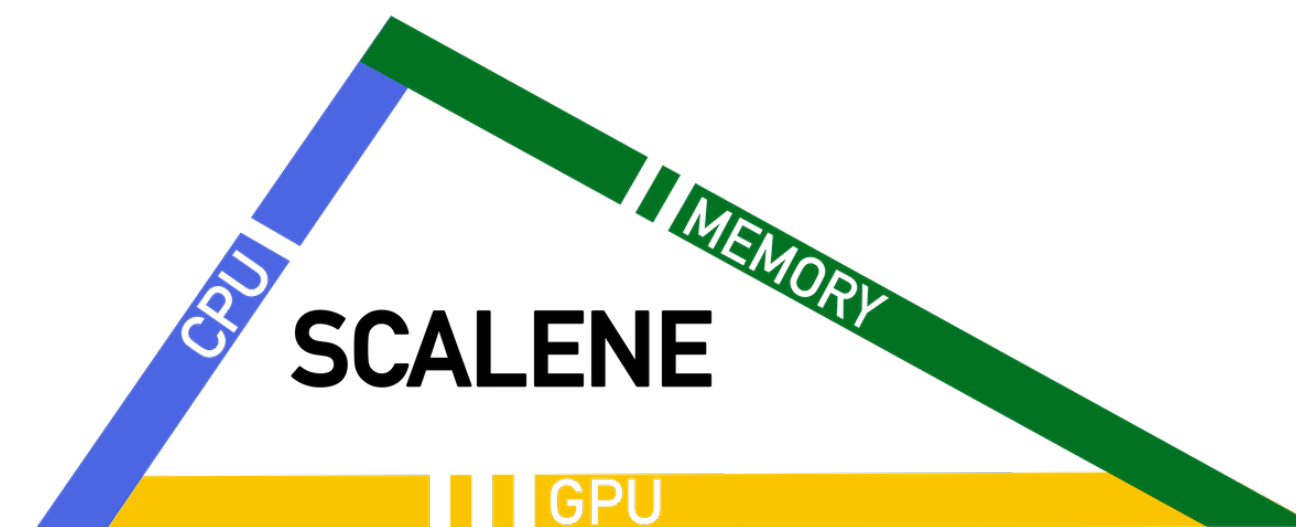
CPU  
PYTHON  
NATIVE  
SYS%

MEMORY  
PYTHON  
NATIVE  
AVERAGE &  
PEAK

MEMORY  
USAGE  
OVER TIME,  
% OF MEM  
ALLOCATED

COPY  
VOLUME  
(MB/s)

GPU  
UTIL %,  
PEAK  
MEMORY



Time: Python | native | system



Memory: Python | native



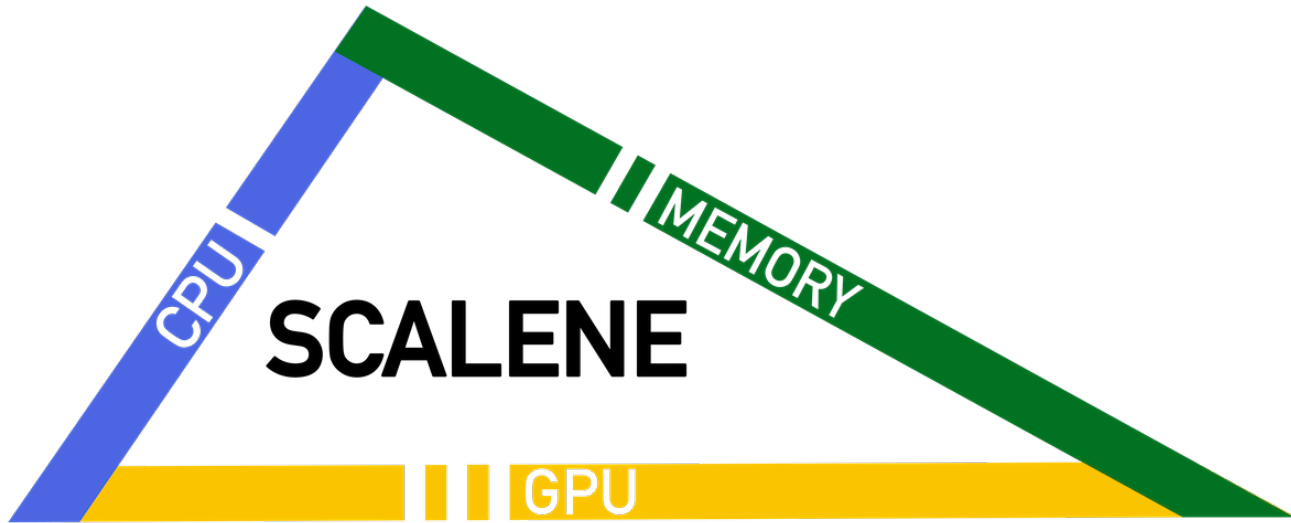
Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
								5 for i in range(10):
				17%				6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



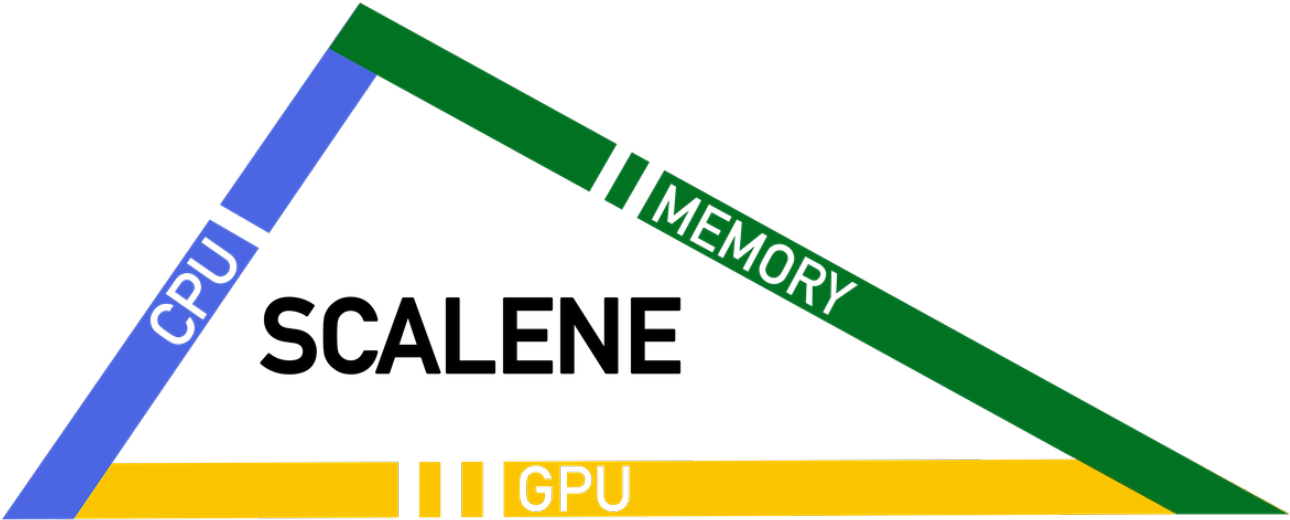
hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
								5 for i in range(10):
				17%				6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))



58% of runtime in native code



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



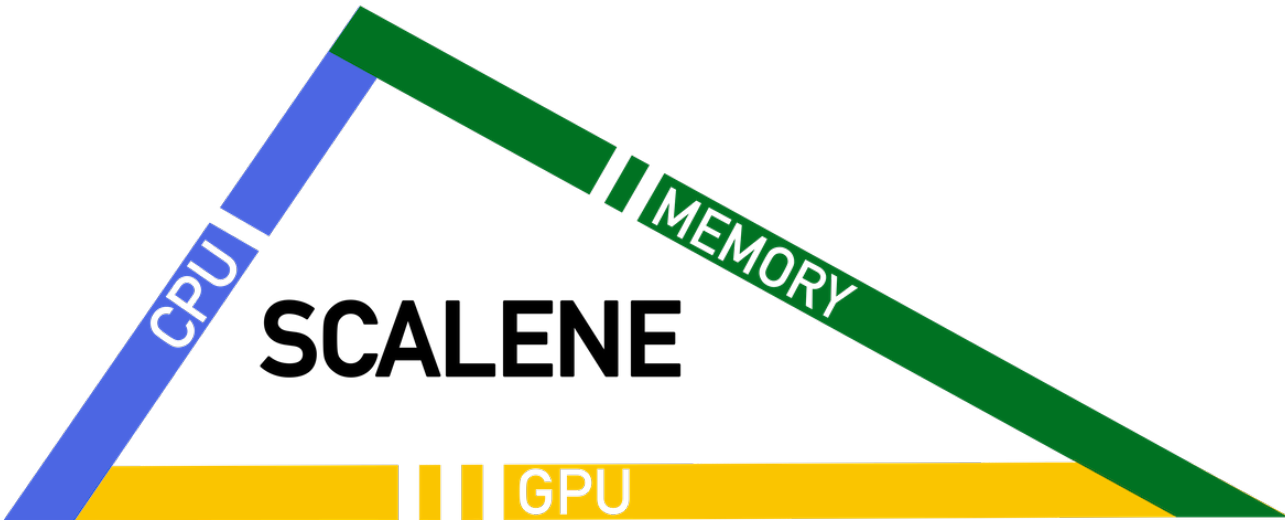
hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
								5 for i in range(10):
				17%				6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

58% of runtime in native code

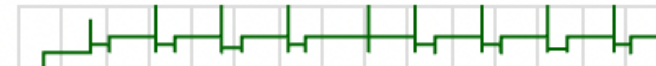
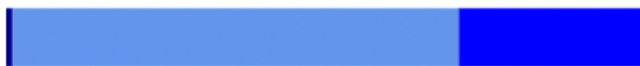
2GB allocated in native code



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

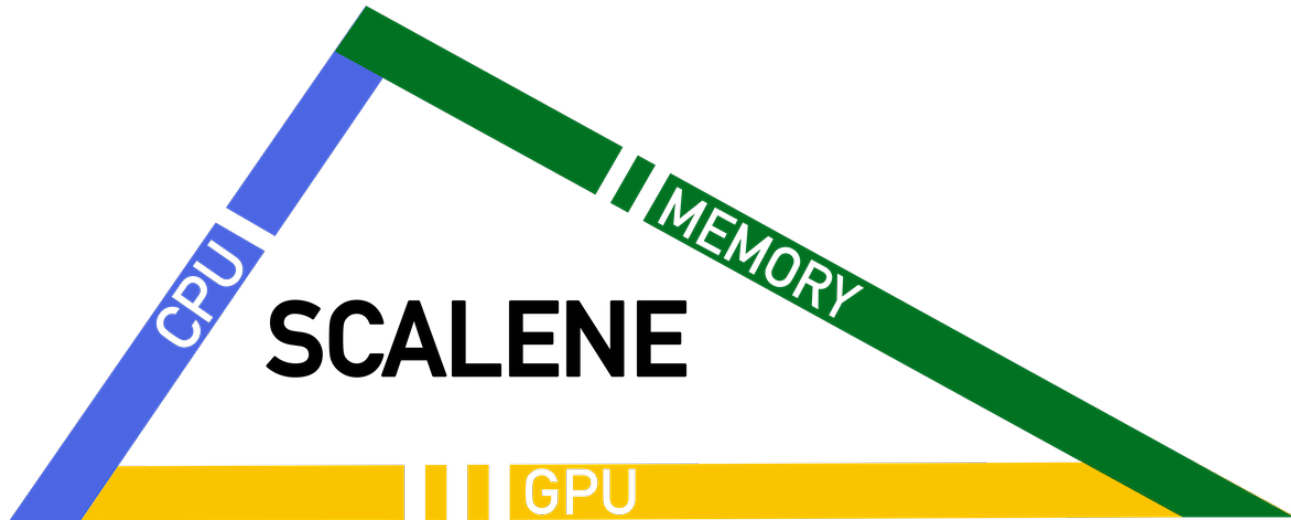
./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
								5 for i in range(10):
				17%				6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

58% of runtime in native code

2GB allocated in native code

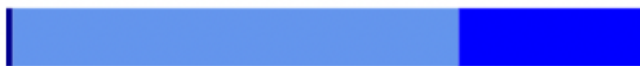
83% of memory activity "sawtooth" pattern



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

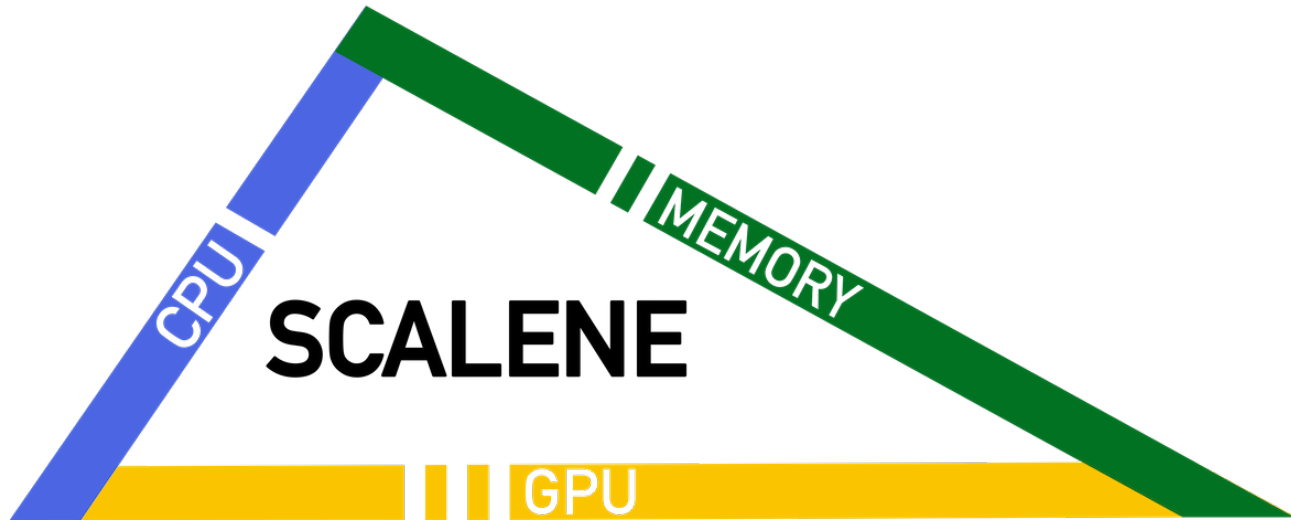
<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
								5 for i in range(10):
				17%				6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

58% of runtime in native code

2GB allocated in native code

83% of memory activity "sawtooth" pattern

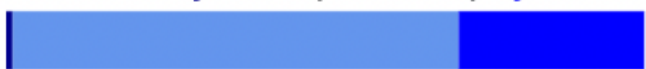
250MB/s copying



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								./test2-2.py
								2 import numpy as np
								4 def main():
								5 for i in range(10):
								6 x = np.array(range(10**7))
					253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

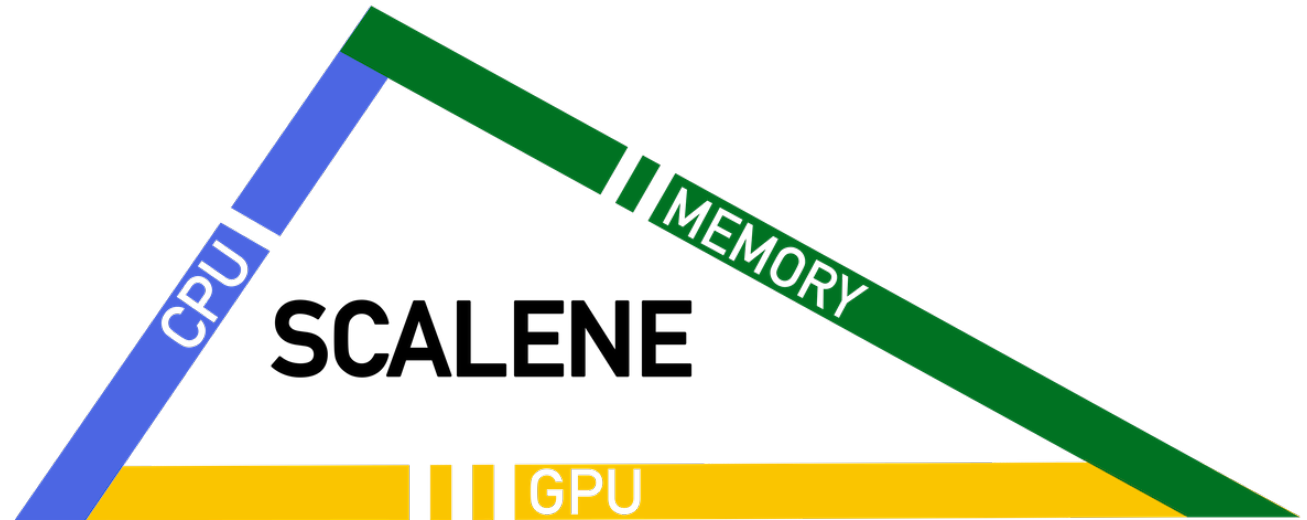
converts to numpy array

58% of runtime in native code

2GB allocated in native code

83% of memory activity "sawtooth" pattern

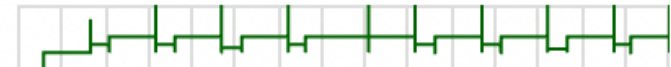
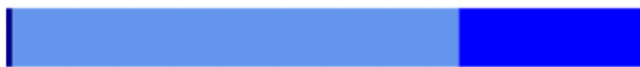
250MB/s copying



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								./test2-2.py
								2 import numpy as np
								4 def main():
								5 for i in range(10):
								6 x = np.array(range(10**7))
					253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

58% of runtime in native code

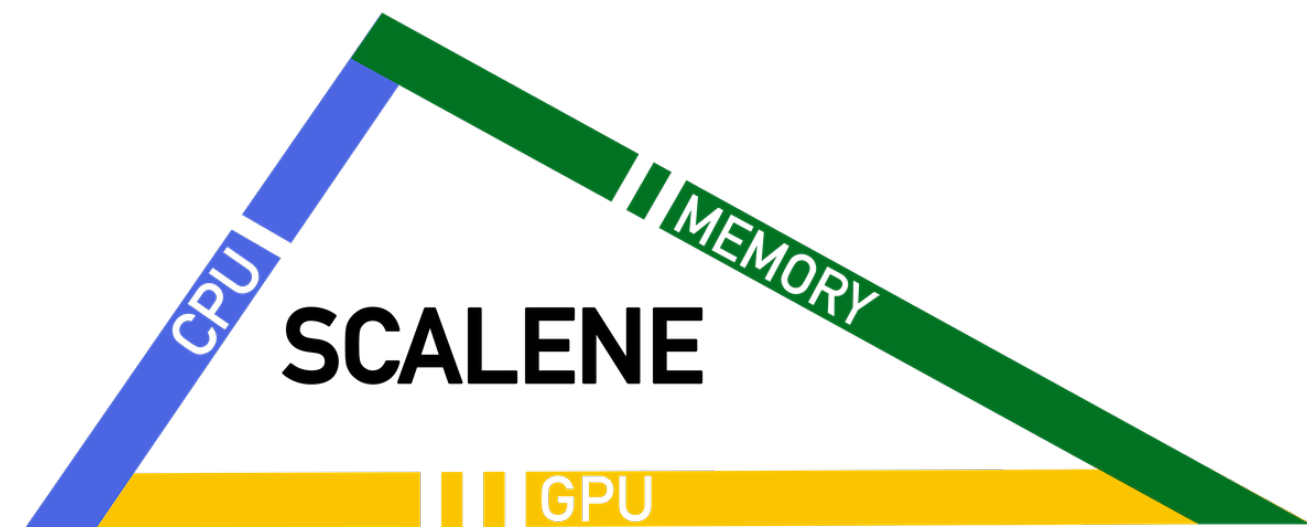
2GB allocated in native code

83% of memory activity "sawtooth" pattern

250MB/s copying

converts to numpy array

already a numpy array!





Time: Python | native | system

Memory: Python | native

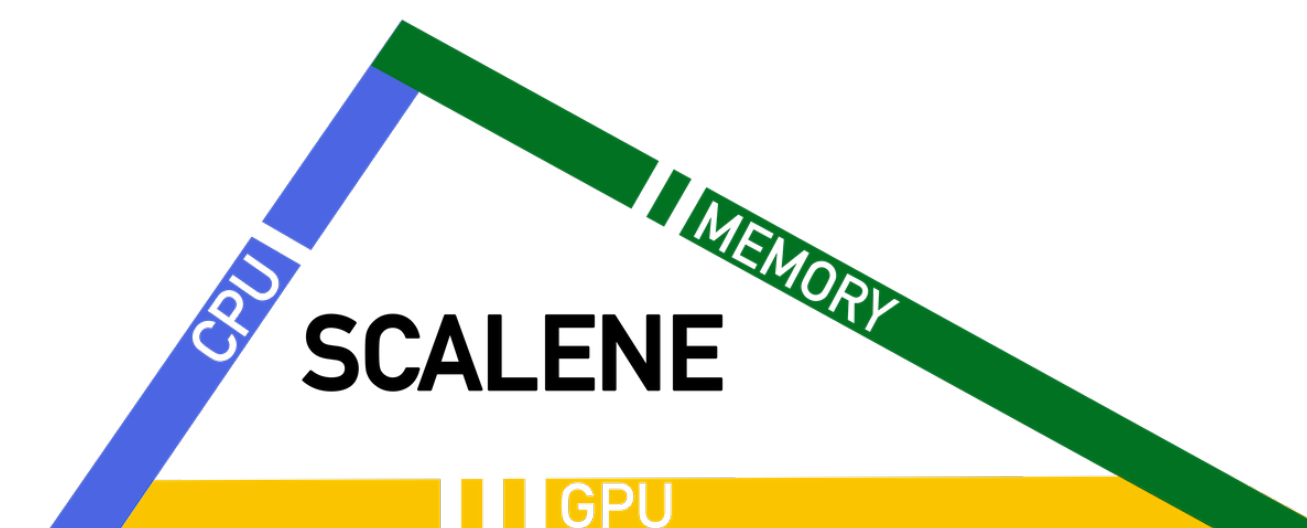
Memory timeline: (max: 3135.8MB, growth: 3.1%)



hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY</u> average	<u>MEMORY</u> peak	<u>MEMORY</u> timeline	<u>MEMORY</u> activity	<u>COPY</u> (MB/s)	<u>GPU</u> util.	<u>GPU</u> memory	<u>LINE PROFILE</u> (click to reset order) ./test2-2.py
								2 import numpy as np
								4 def main():
				17%				5 for i in range(10):
								6 x = np.array(range(10**7))
				83%	253			7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

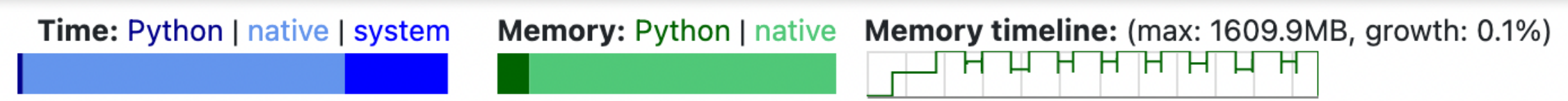




hover over bars to see breakdowns; click on [COLUMN HEADERS](#) to sort.

`./test2-2.py`: % of time = 97.8% out of 30.1s.

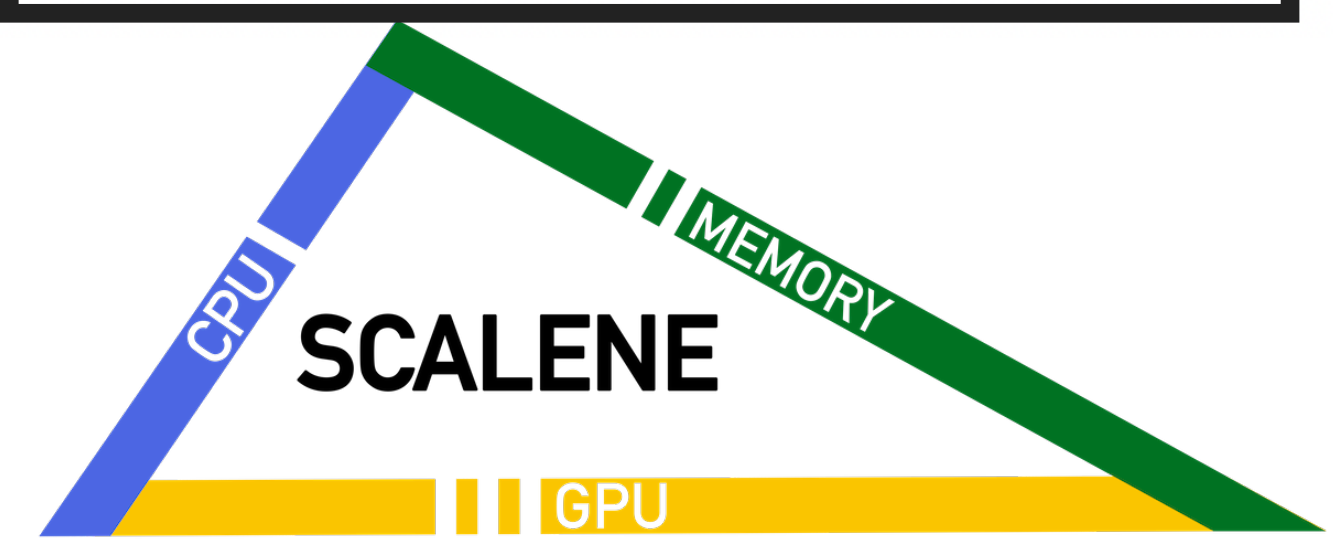
<a href="#">TIME</a>	<a href="#">MEMORY average</a>	<a href="#">MEMORY peak</a>	<a href="#">MEMORY timeline</a>	<a href="#">MEMORY activity</a>	<a href="#">COPY (MB/s)</a>	<a href="#">GPU util.</a>	<a href="#">GPU memory</a>	<a href="#">LINE PROFILE</a> (click to reset order)
								2 <code>import numpy as np</code>
								4 <code>def main():</code>
								5 <code>for i in range(10):</code>
				17%				6 <code>x = np.array(range(10**7))</code>
				83%	253			7 <code>y = np.array(np.random.uniform(0, 100, size=(10**8)))</code>



hover over bars to see breakdowns; click on [COLUMN HEADERS](#) to sort.

`./test2-2-optimized.py`: % of time = 95.2% out of 23.4s.

<a href="#">TIME</a>	<a href="#">MEMORY average</a>	<a href="#">MEMORY peak</a>	<a href="#">MEMORY timeline</a>	<a href="#">MEMORY activity</a>	<a href="#">COPY (MB/s)</a>	<a href="#">GPU util.</a>	<a href="#">GPU memory</a>	<a href="#">LINE PROFILE</a> (click to reset order)
								2 <code>import numpy as np</code>
								4 <code>def main():</code>
								5 <code>for i in range(10):</code>
				38%				6 <code>x = np.array(range(10**7))</code>
				62%				7 <code>y = np.random.uniform(0, 100, size=(10**8))</code>



Time: Python | native | system

Memory: Python | native

Memory timeline: (max: 3135.8MB, growth: 3.1%)

hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

./test2-2.py: % of time = 97.8% out of 30.1s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
				17%				5 for i in range(10):
				83%	253			6 x = np.array(range(10**7))
								7 y = np.array(np.random.uniform(0, 100, size=(10**8)))

Time: Python | native | system

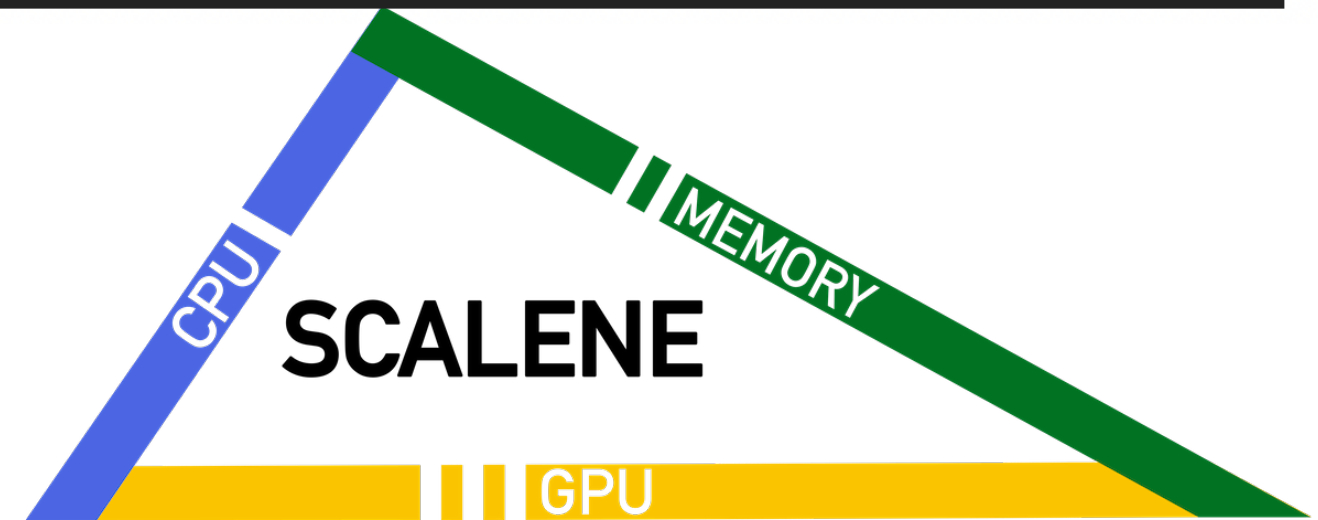
Memory: Python | native

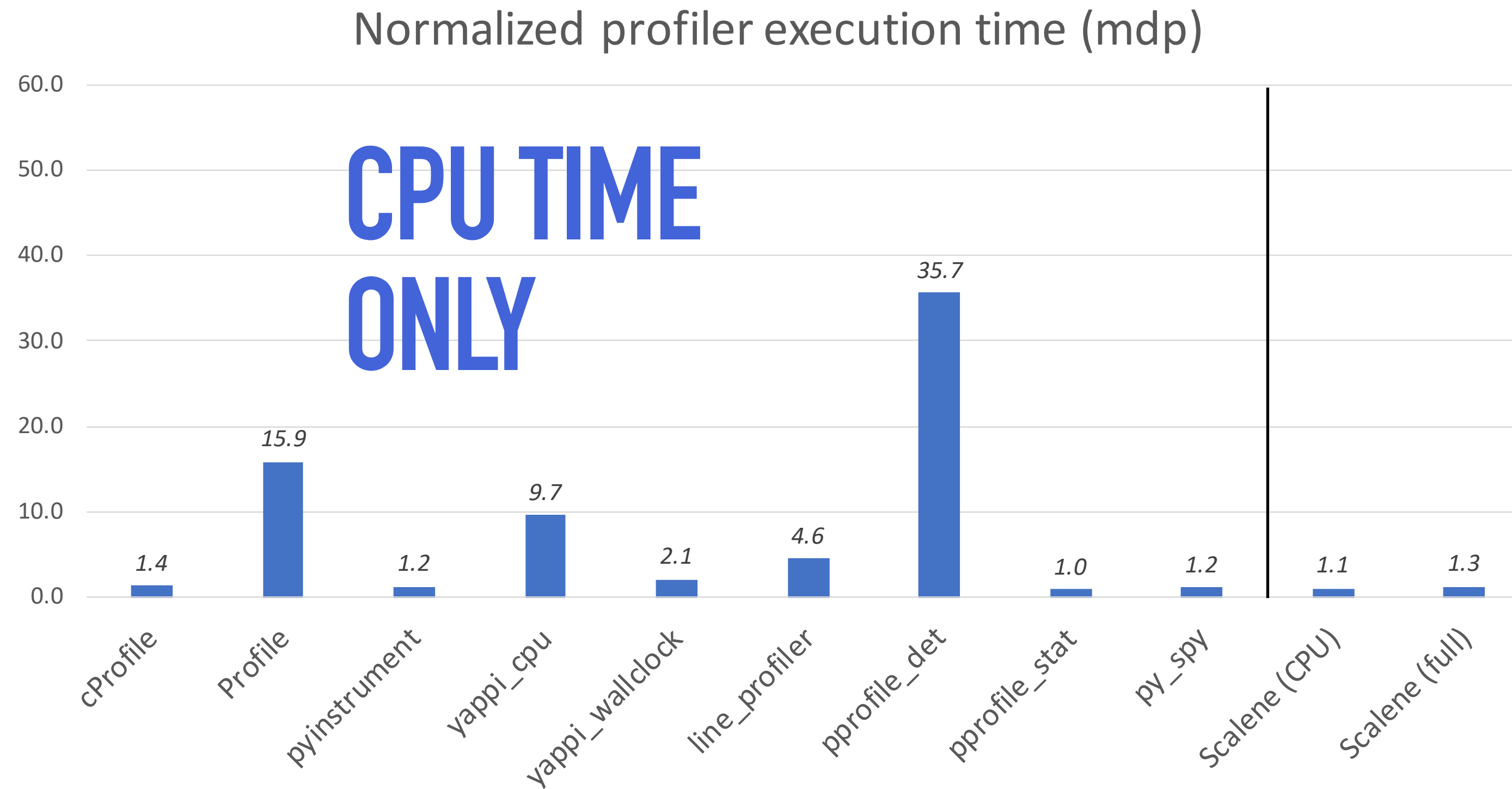
Memory timeline: (max: 1609.9MB, growth: 0.1%)

hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

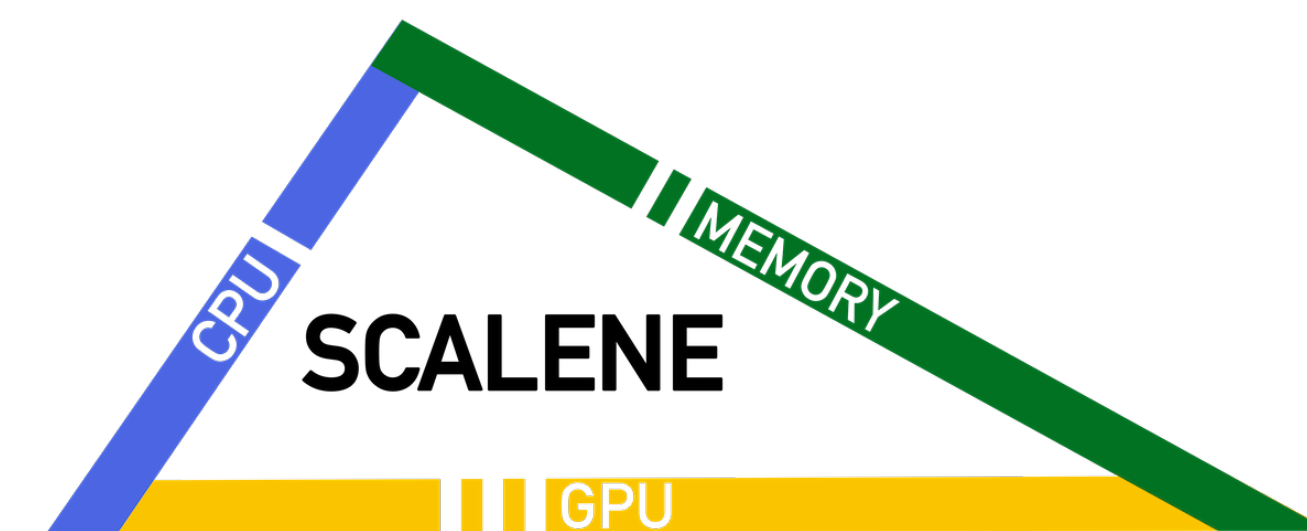
./test2-2-optimized.py: % of time = 95.2% out of 23.4s.

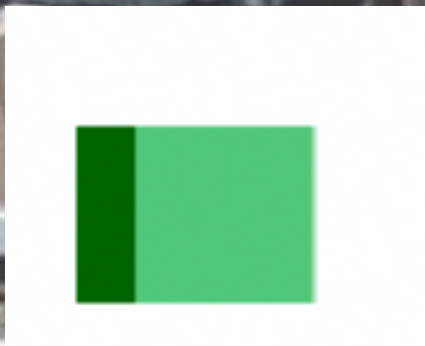
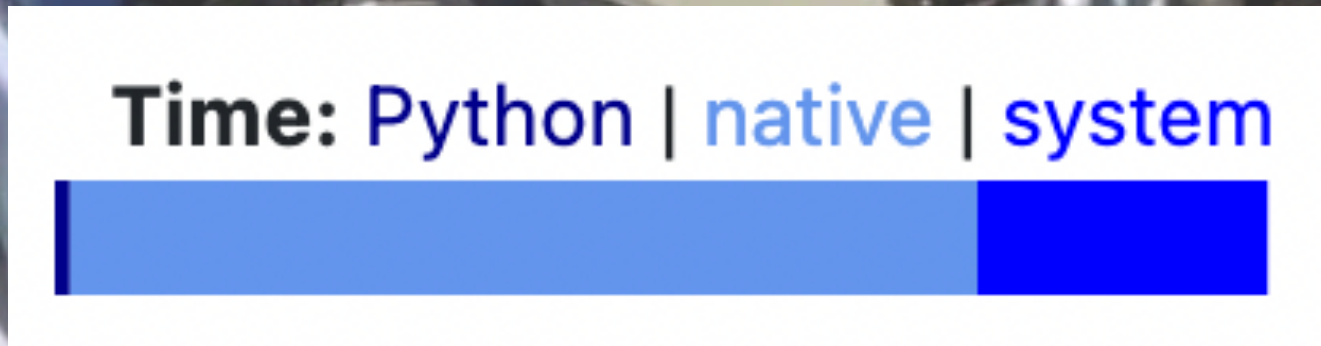
<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY (MB/s)</u>	<u>GPU util.</u>	<u>GPU memory</u>	<u>LINE PROFILE</u> (click to reset order)
								2 import numpy as np
								4 def main():
				38%				5 for i in range(10):
				62%				6 x = np.array(range(10**7))
								7 y = np.random.uniform(0, 100, size=(10**8))





memory\_profiler: ~300X slower  
(from 5s to 20 minutes!)





**NATIVE VS.  
PYTHON TIME**

**LOW-OVERHEAD  
MEMORY PROFILING**





# SAMPLING -BASED PROFILING





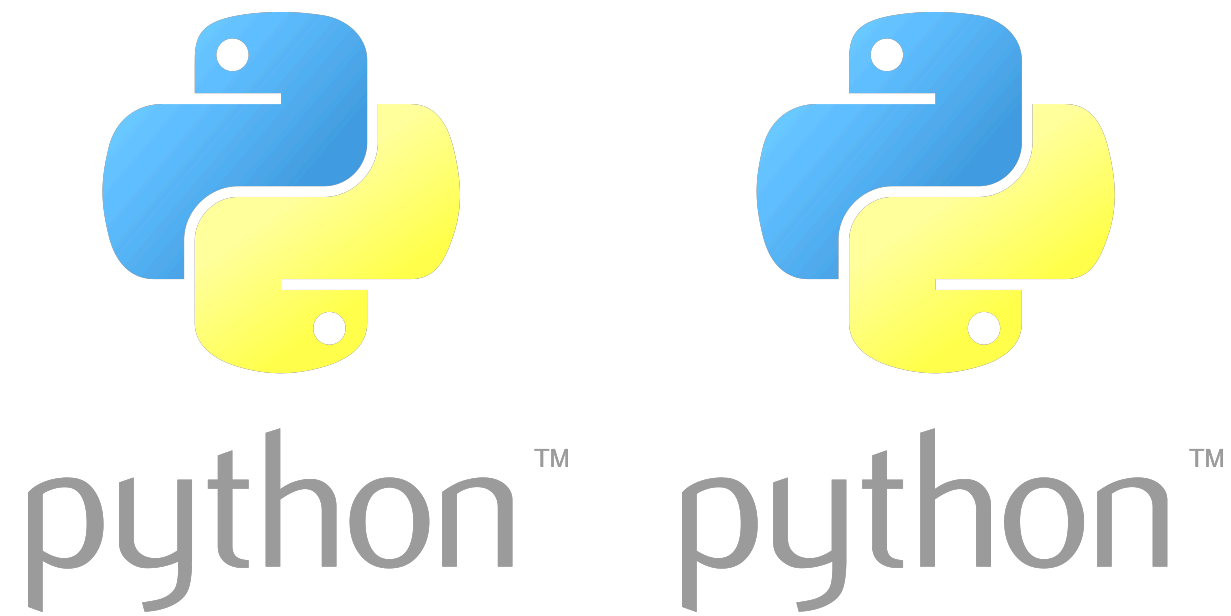
# SAMPLING -BASED PROFILING



foo

function	time
<b>foo</b>	<b>1</b>

# SAMPLING -BASED PROFILING



foo

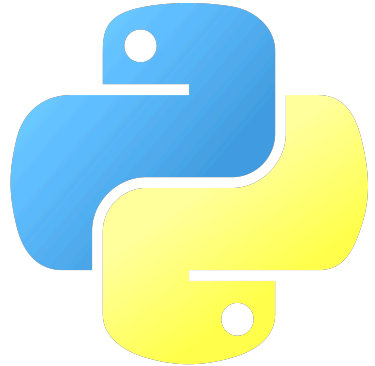
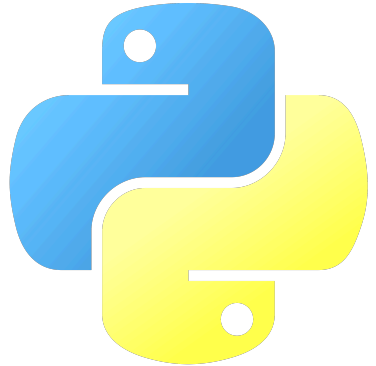
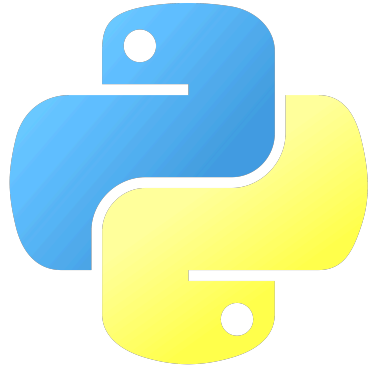





bar

function	time
<b>foo</b>	<b>1</b>
<b>bar</b>	<b>1</b>



# SAMPLING -BASED PROFILING

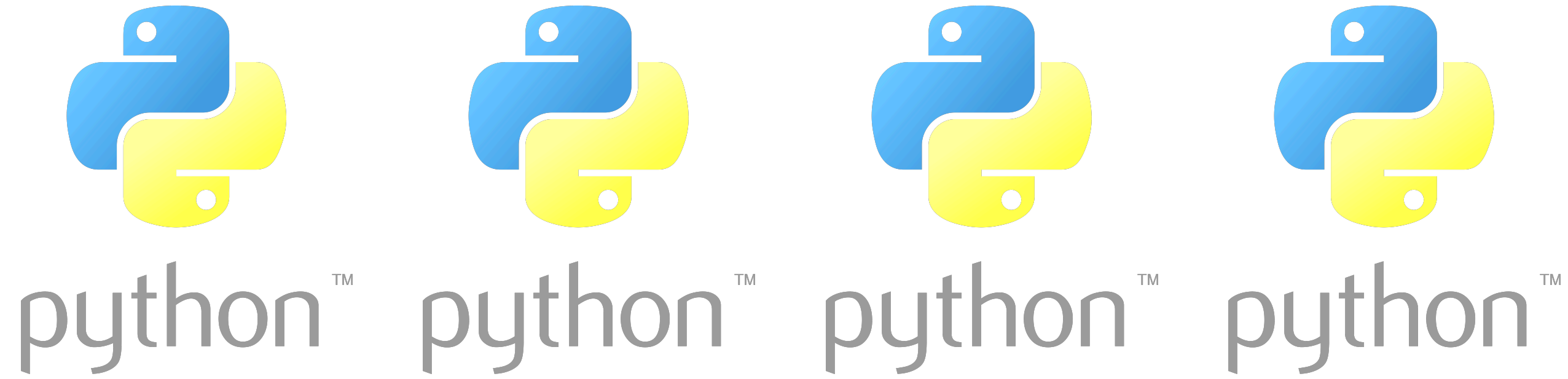
 python™	 python™	 python™
		
foo	bar	bar

function	time
<b>foo</b>	<b>1</b>
<b>bar</b>	<b>2</b>

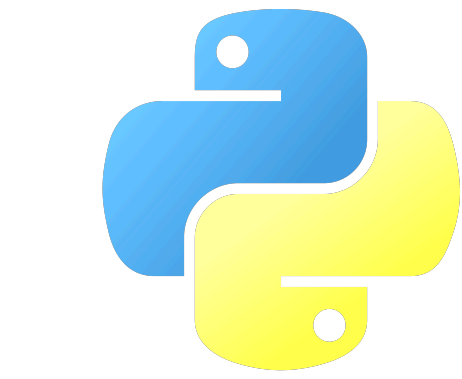
# SAMPLING -BASED PROFILING

 python™	 python™	 python™	 python™
			
foo	bar	bar	foo

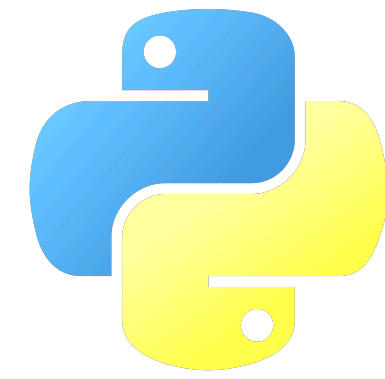
function	time
<b>foo</b>	<b>2</b>
bar	2



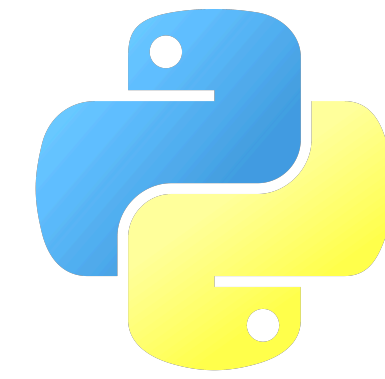
**DEFERRED  
SIGNAL  
DELIVERY**



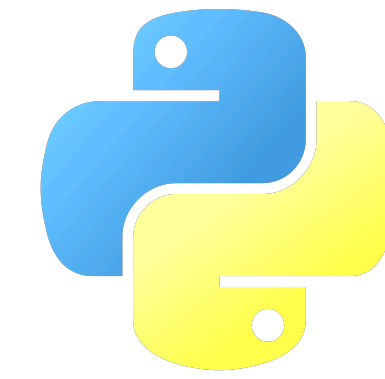
python™



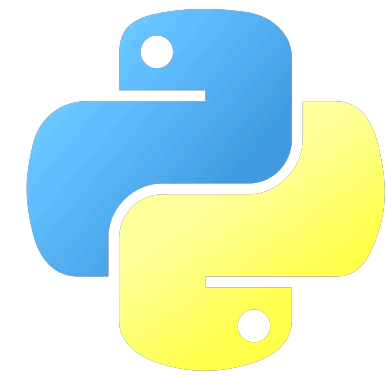
python™



python™



python™

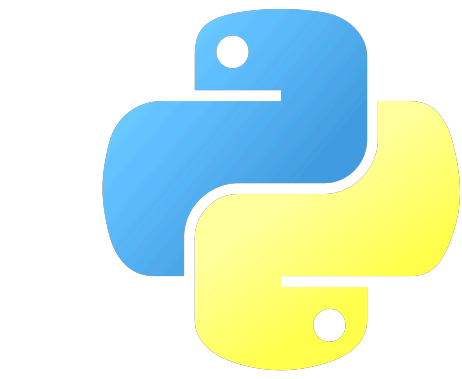


python™

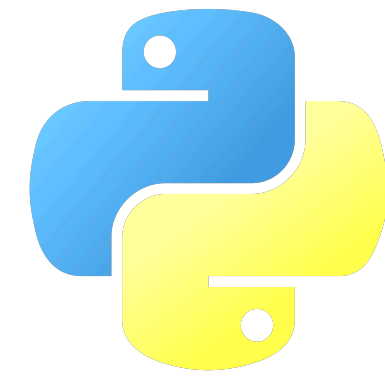


DEFERRED  
SIGNAL  
DELIVERY

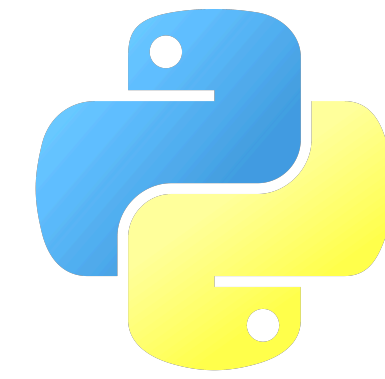
function	time
<b>foo</b>	<b>1</b>



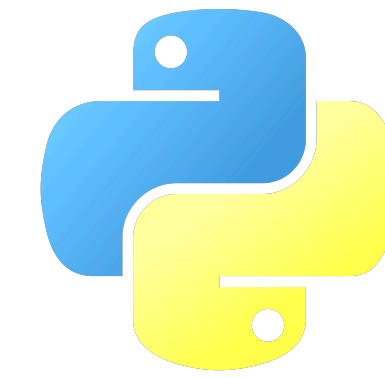
python™



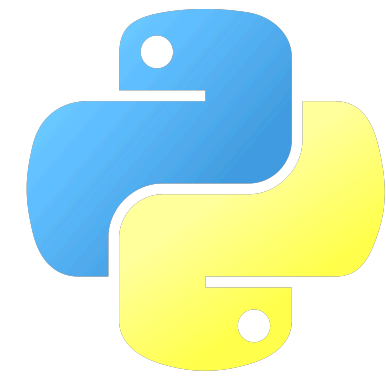
python™



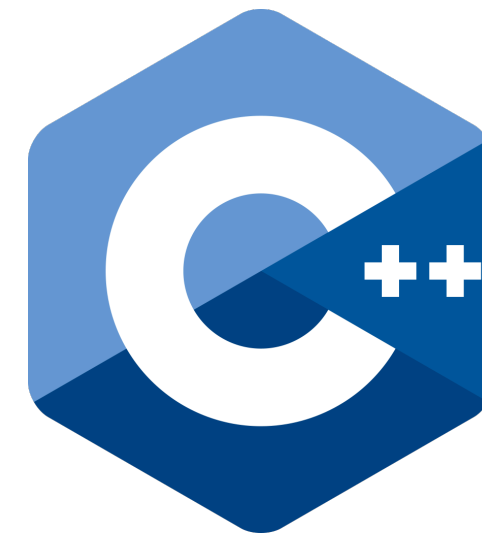
python™



python™



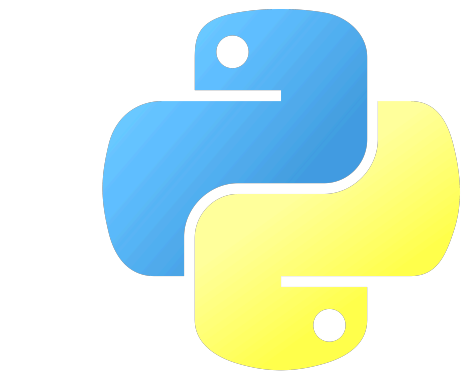
python™



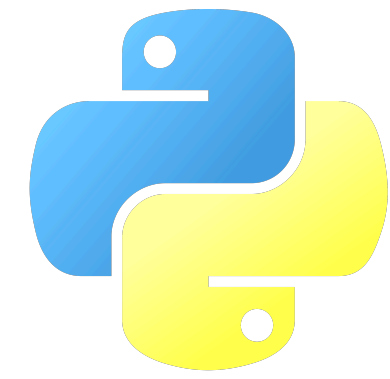
**DEFERRED  
SIGNAL  
DELIVERY**



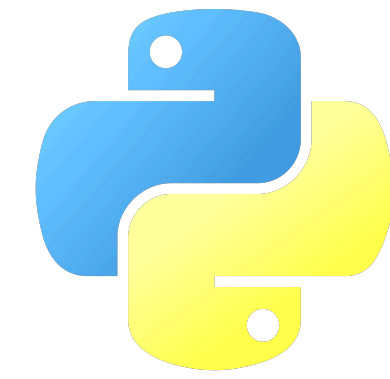
function	time
<b>foo</b>	<b>1</b>



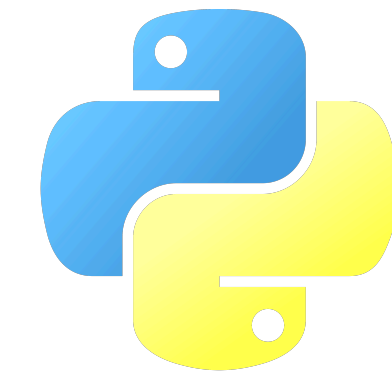
python™



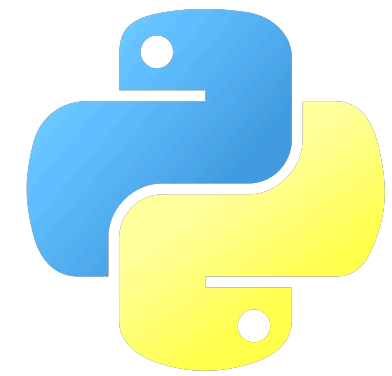
python™



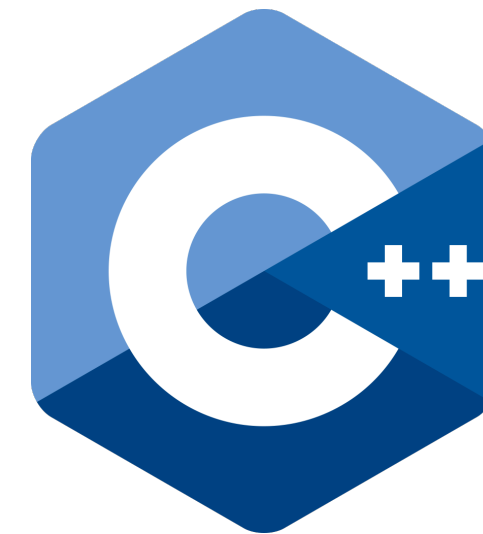
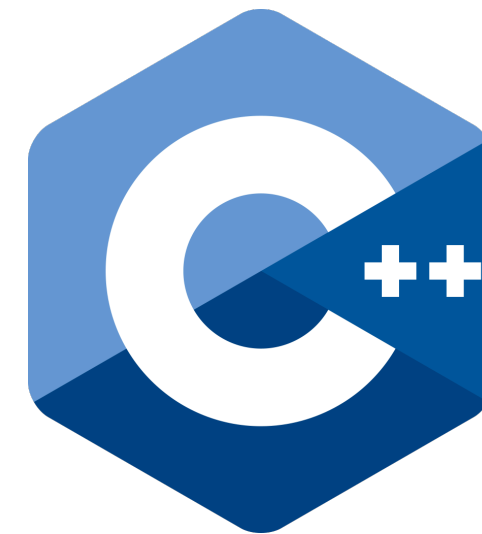
python™



python™

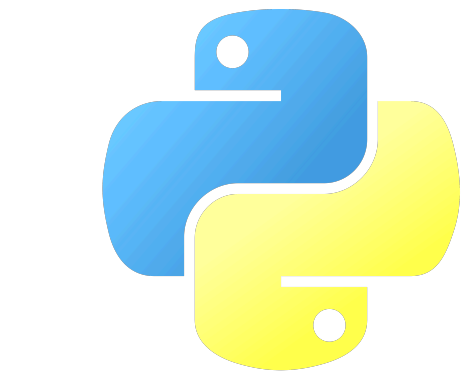


python™

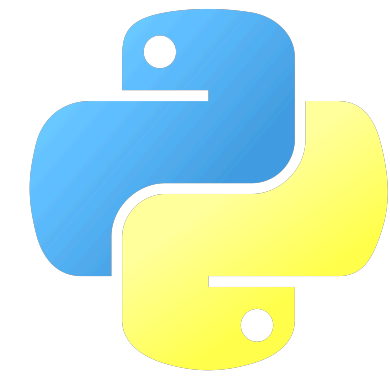


**DEFERRED  
SIGNAL  
DELIVERY**

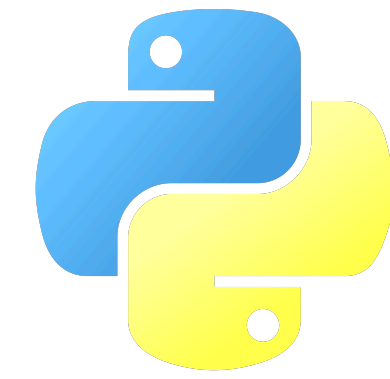
function	time
<b>foo</b>	<b>1</b>



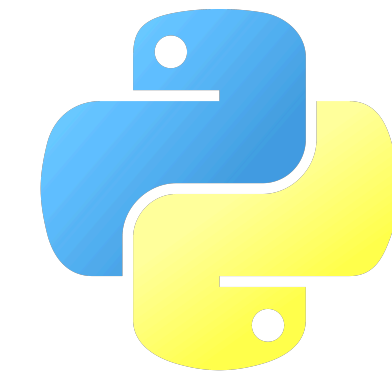
python™



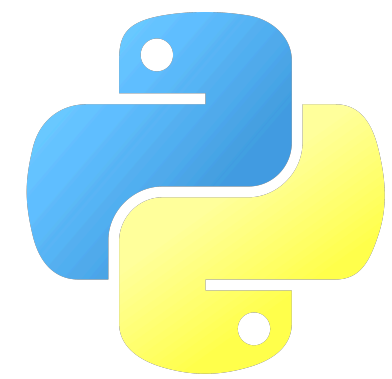
python™



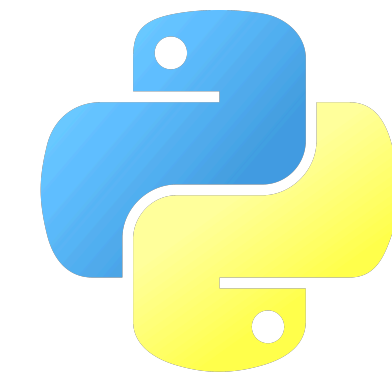
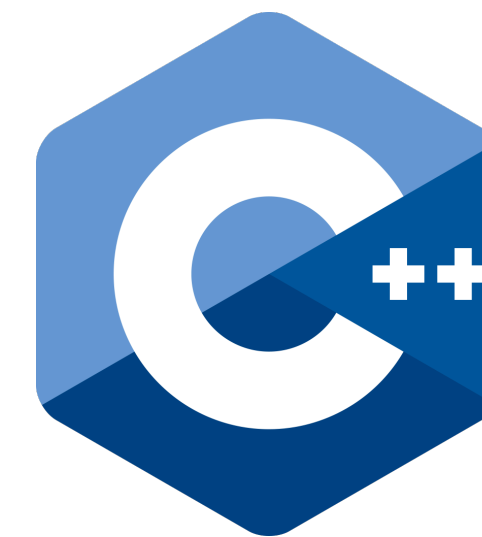
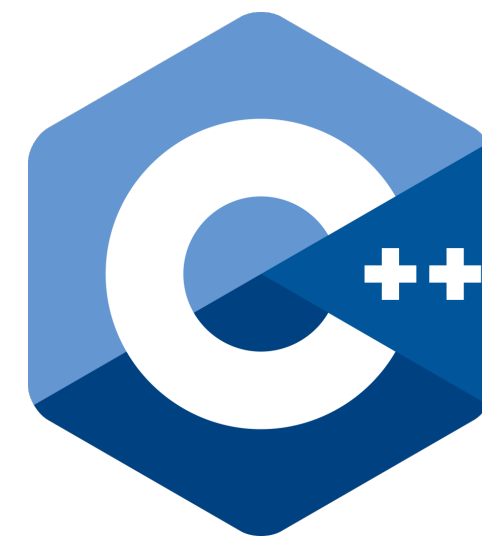
python™



python™



python™

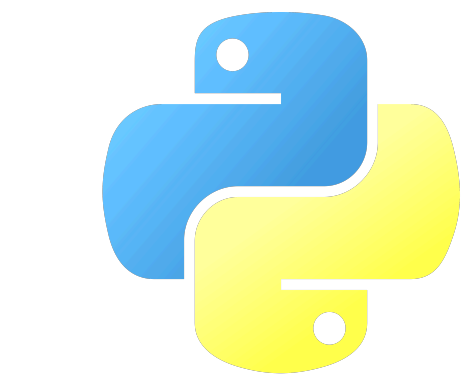


python™

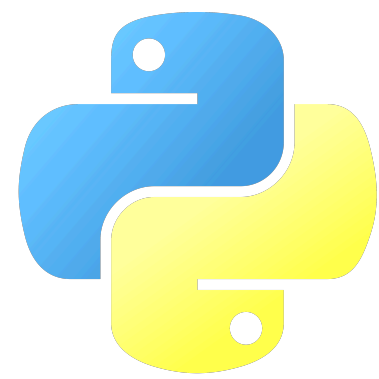


DEFERRED  
SIGNAL  
DELIVERY

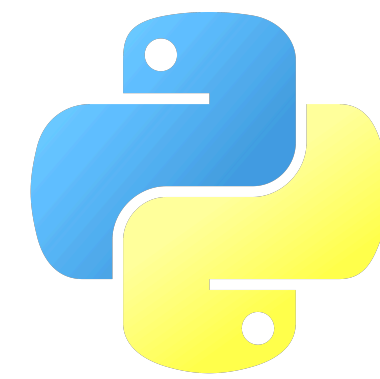
function	time
<b>foo</b>	<b>2</b>



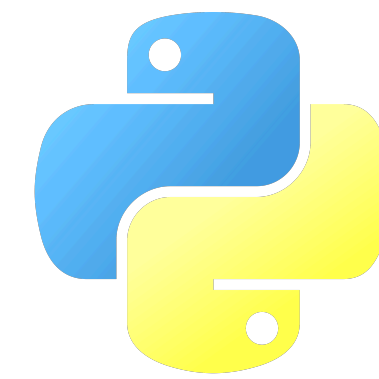
python™



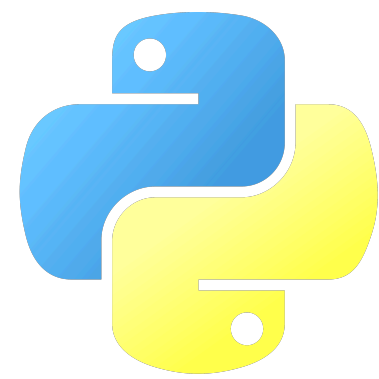
python™



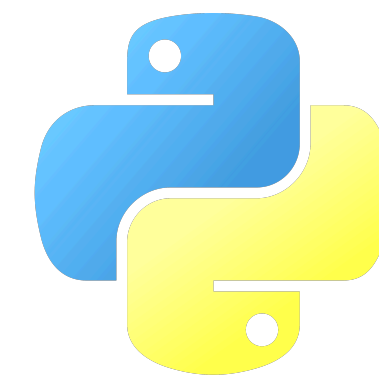
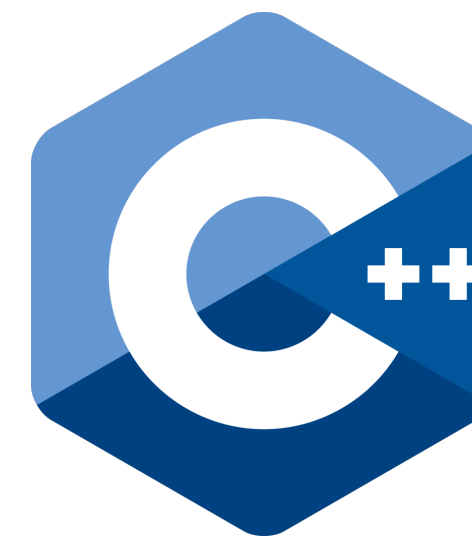
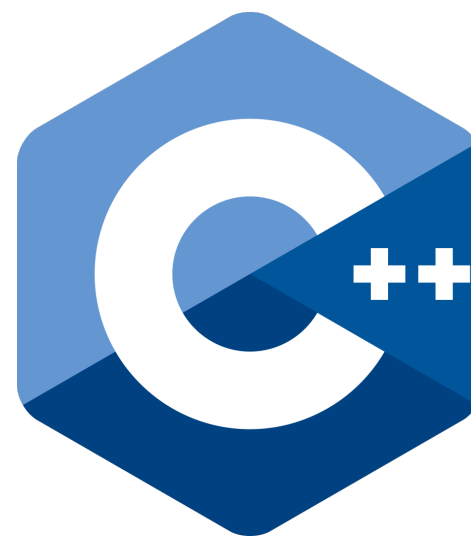
python™



python™



python™



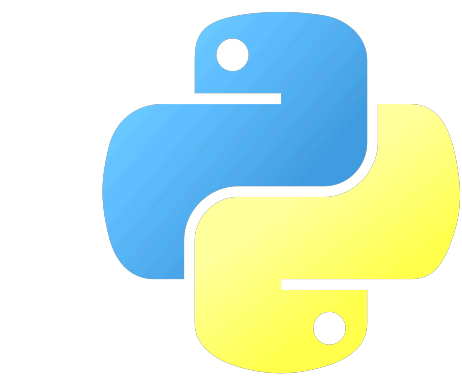
python™



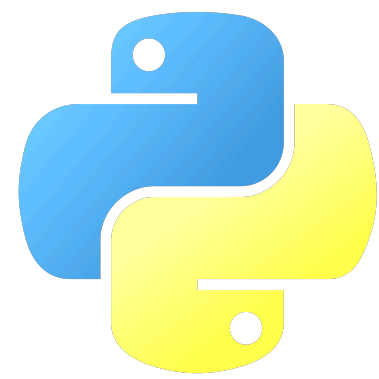
DEFERRED  
SIGNAL  
DELIVERY

function	time
<b>foo</b>	<b>2</b>

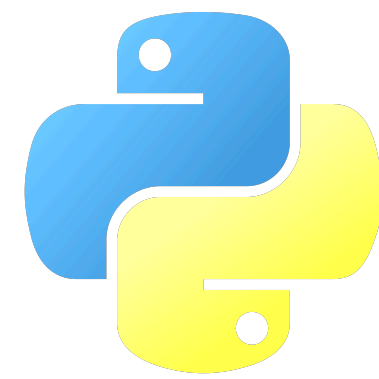




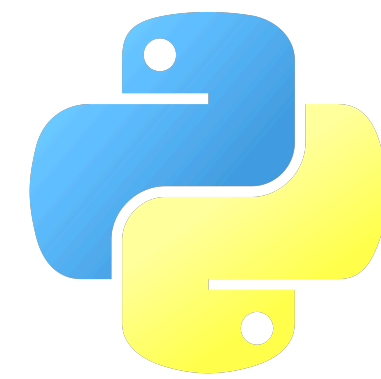
python™



python™



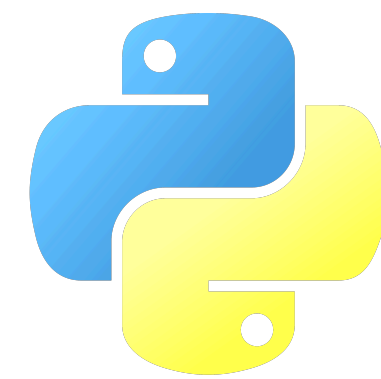
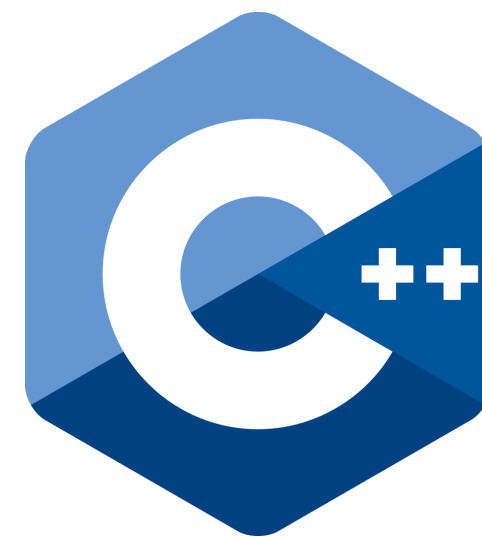
python™



python™



python™



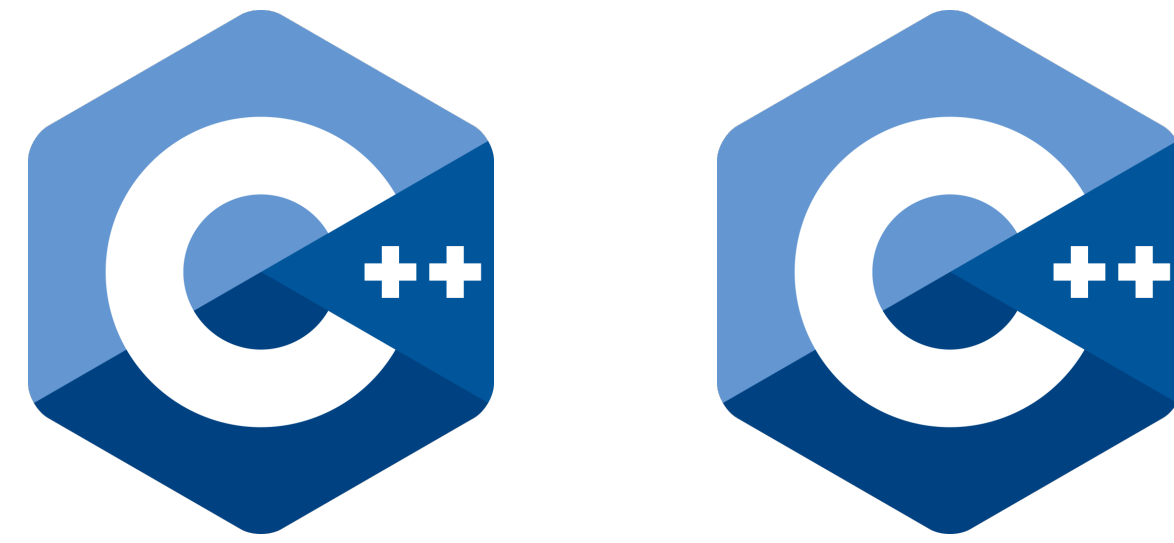
python™



function	time
<b>foo</b>	<b>2</b>

DEFERRED  
SIGNAL  
DELIVERY

(pprofile)

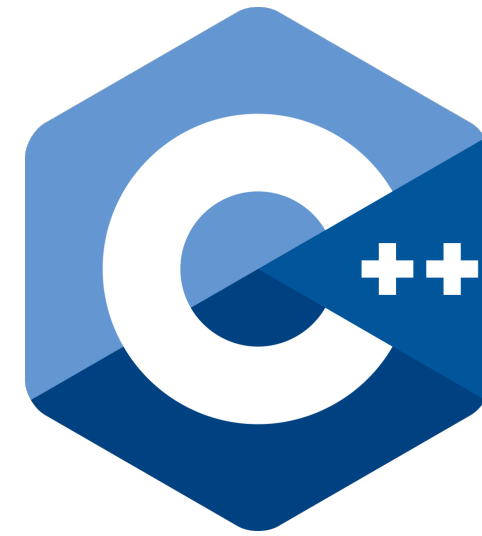


**(VIRTUAL TIME)  
INFERRING  
EXECUTION TIME**



**delay**





(VIRTUAL TIME)

INFERRING  
EXECUTION TIME

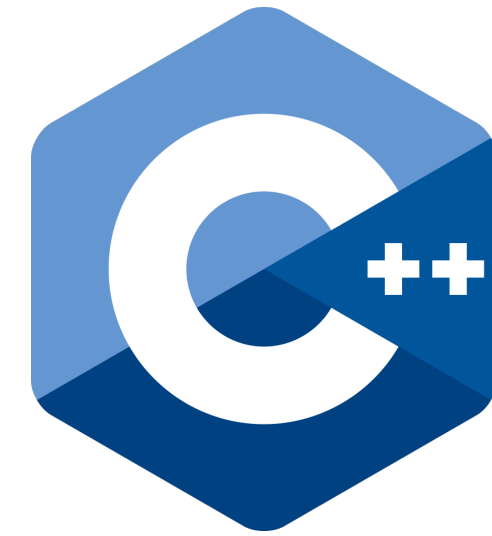


delay



```
python_time += interval
c_time += delay - interval
```

function	time
foo	2
bar	2



(VIRTUAL TIME)  
INFERRING  
EXECUTION TIME



delay



```
python_time += interval
c_time += delay - interval
```

Time: Python | native | system

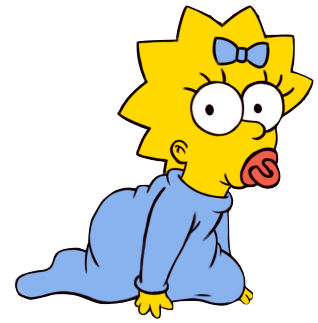


function	time
foo	2
bar	2

memory\_profiler:

memory\_profiler: ~300x slower

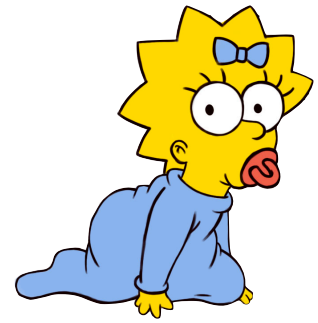
memory\_profiler: ~300x slower



record  
malloc  
info

tracks every  
malloc/free

memory\_profiler: ~300x slower



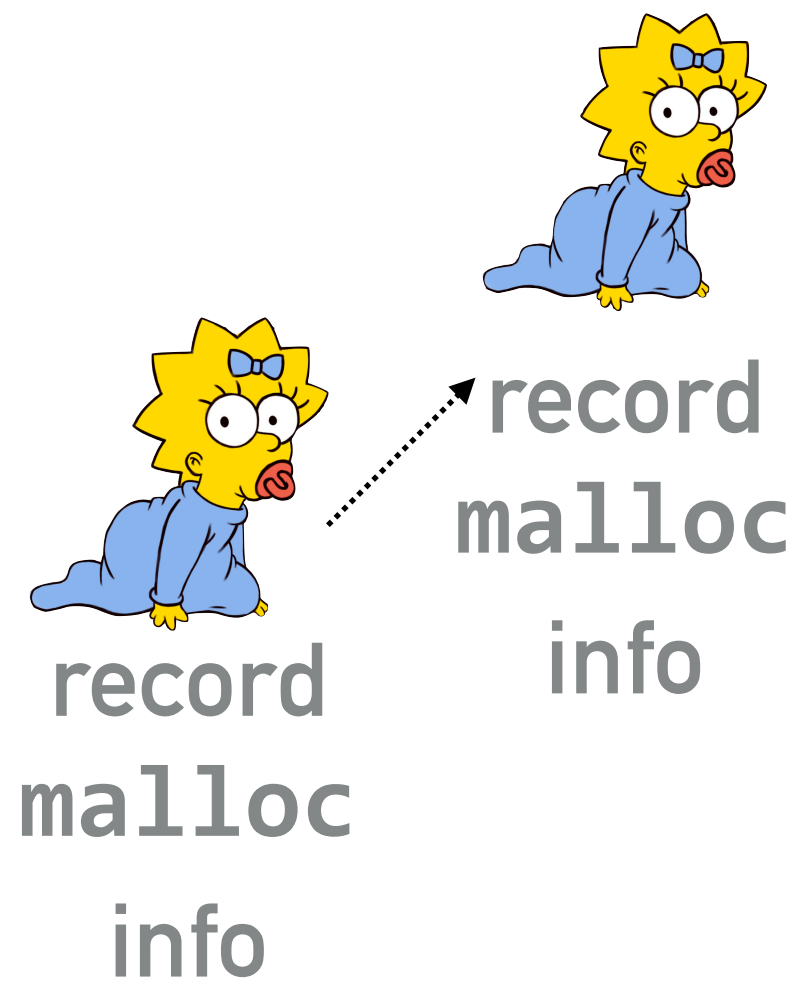
record  
malloc  
info

tracks every  
malloc/free

**invokes  
getrusage!**

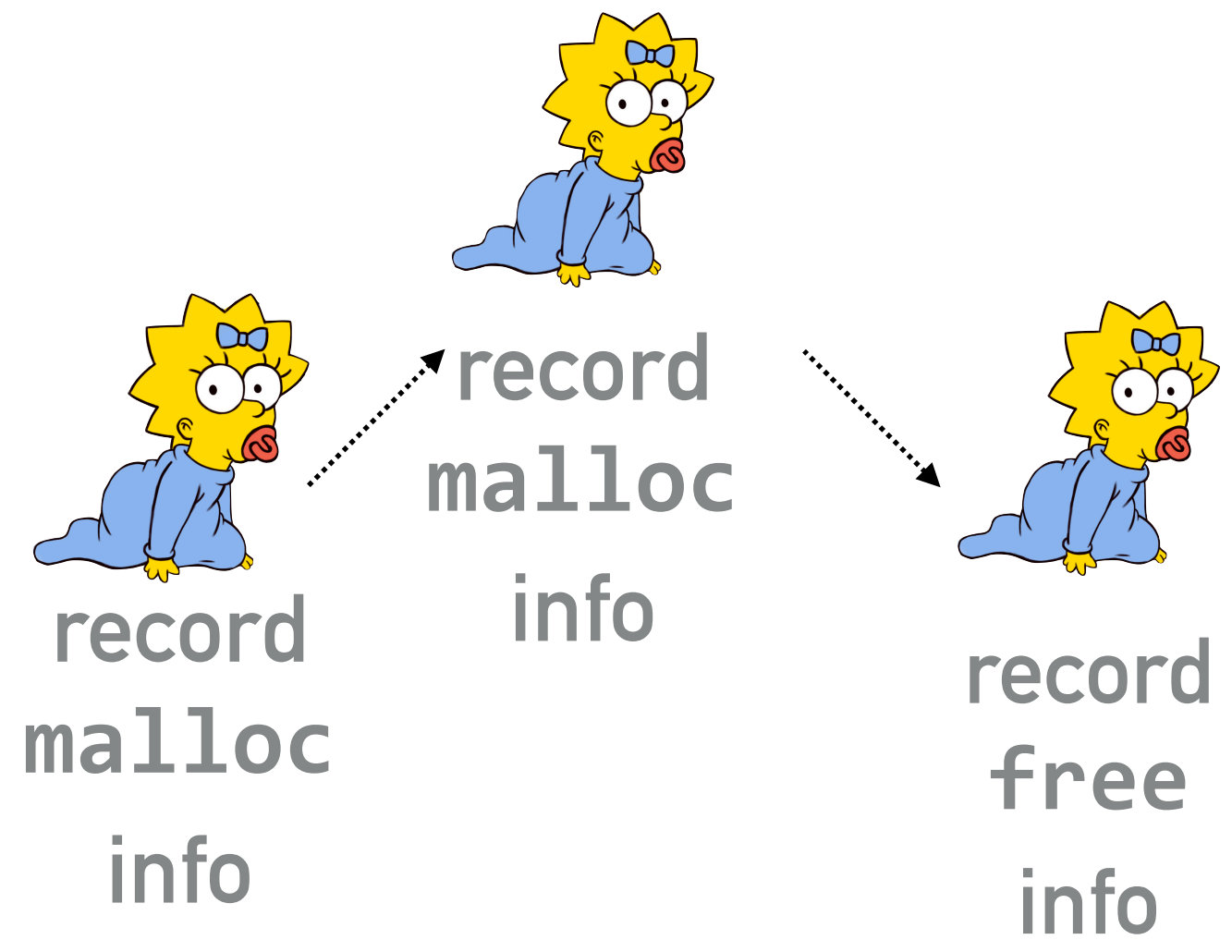


memory\_profiler: ~300x slower



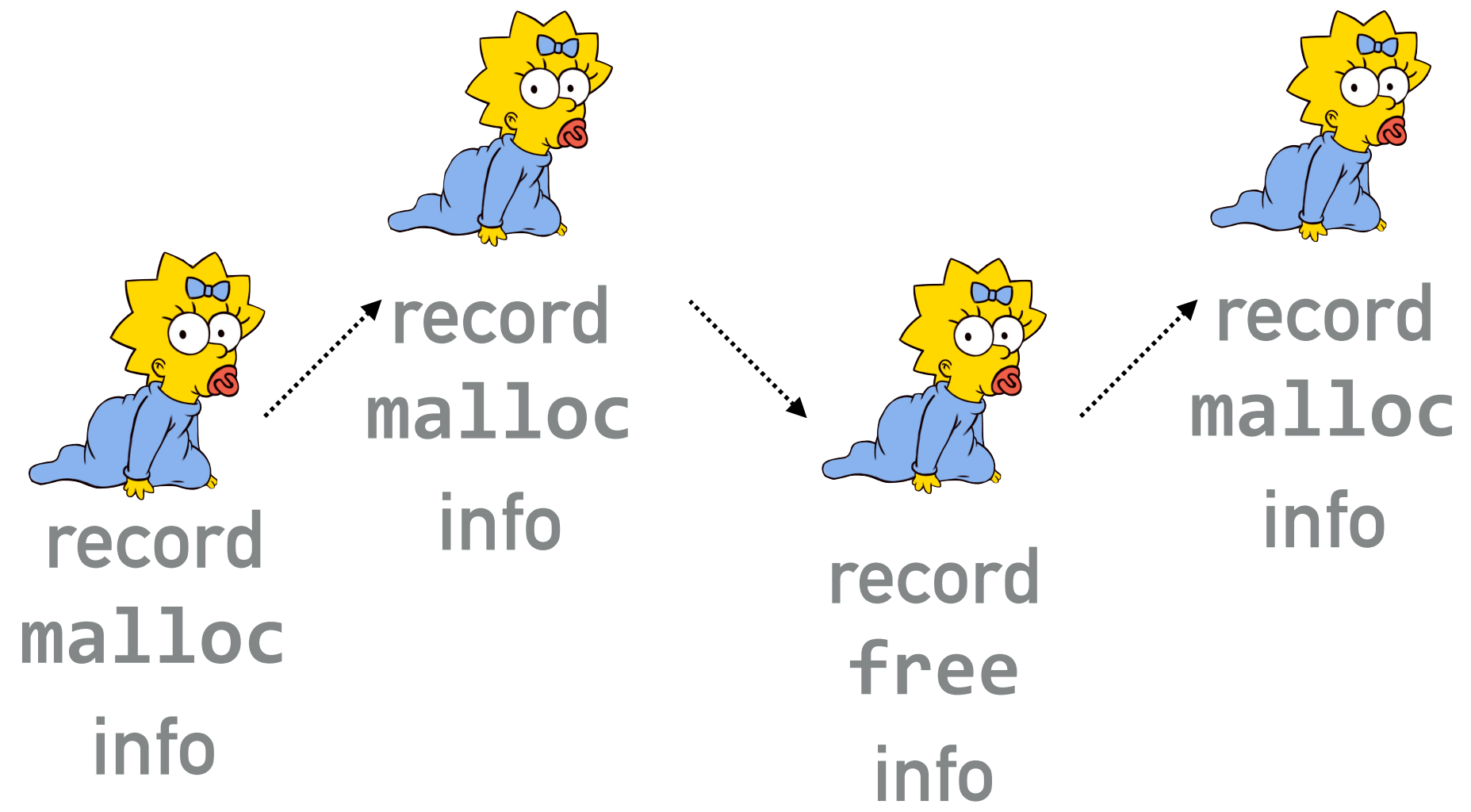
tracks every  
malloc/free

memory\_profiler: ~300x slower



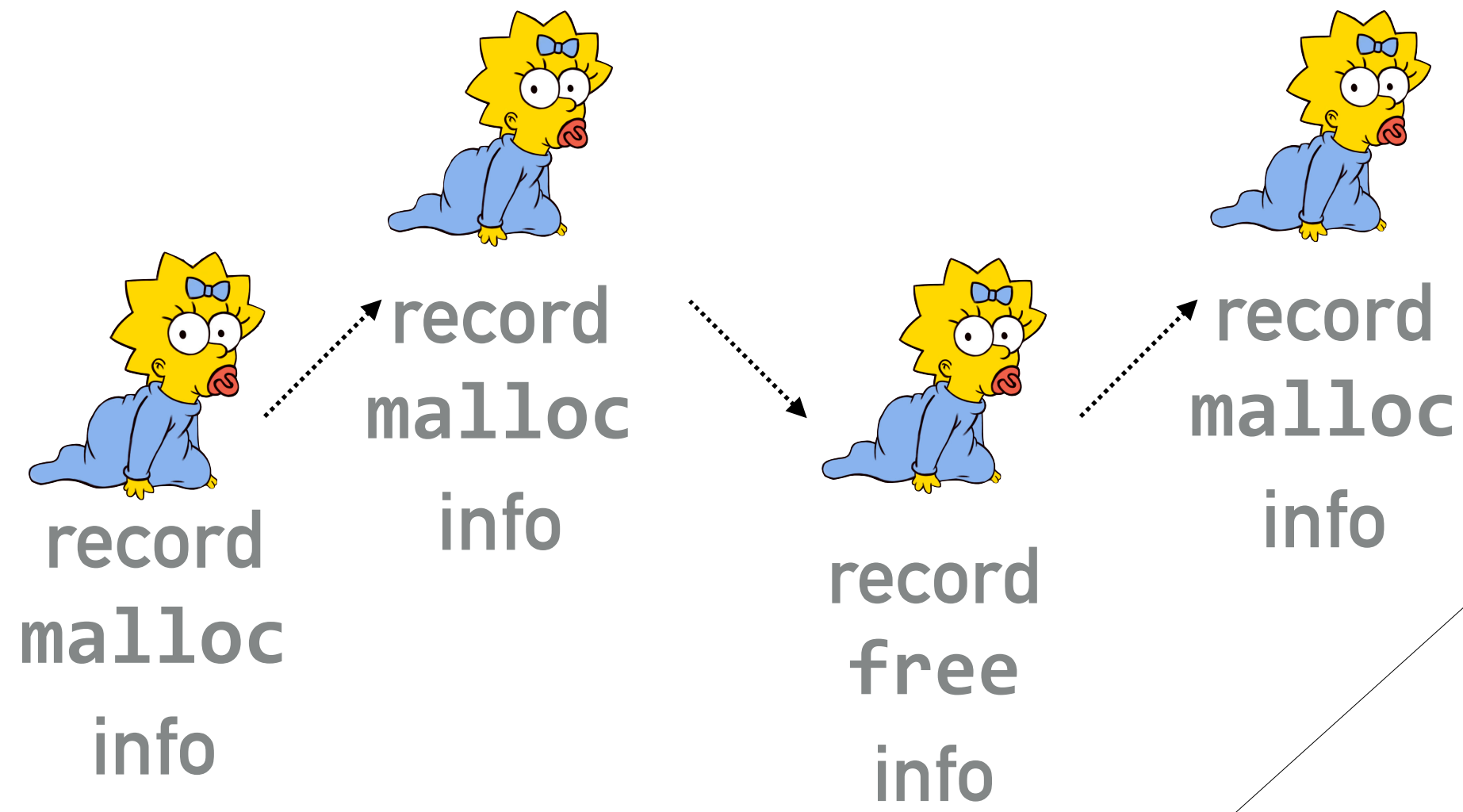
tracks every  
malloc/free

memory\_profiler: ~300x slower

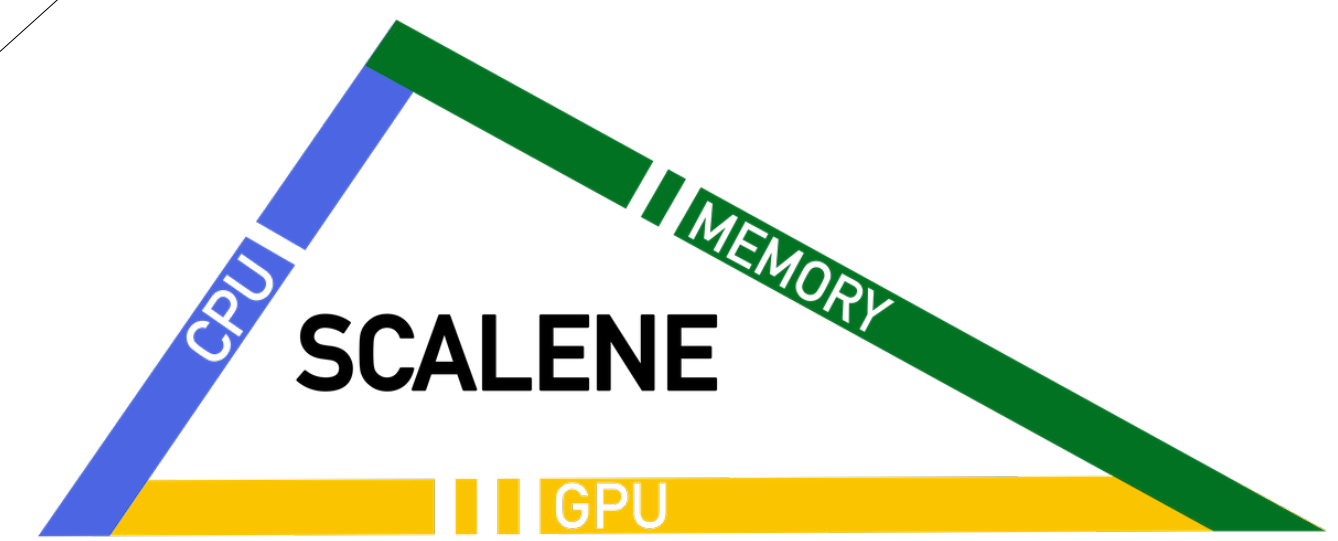


tracks every  
malloc/free

memory\_profiler: ~300x slower

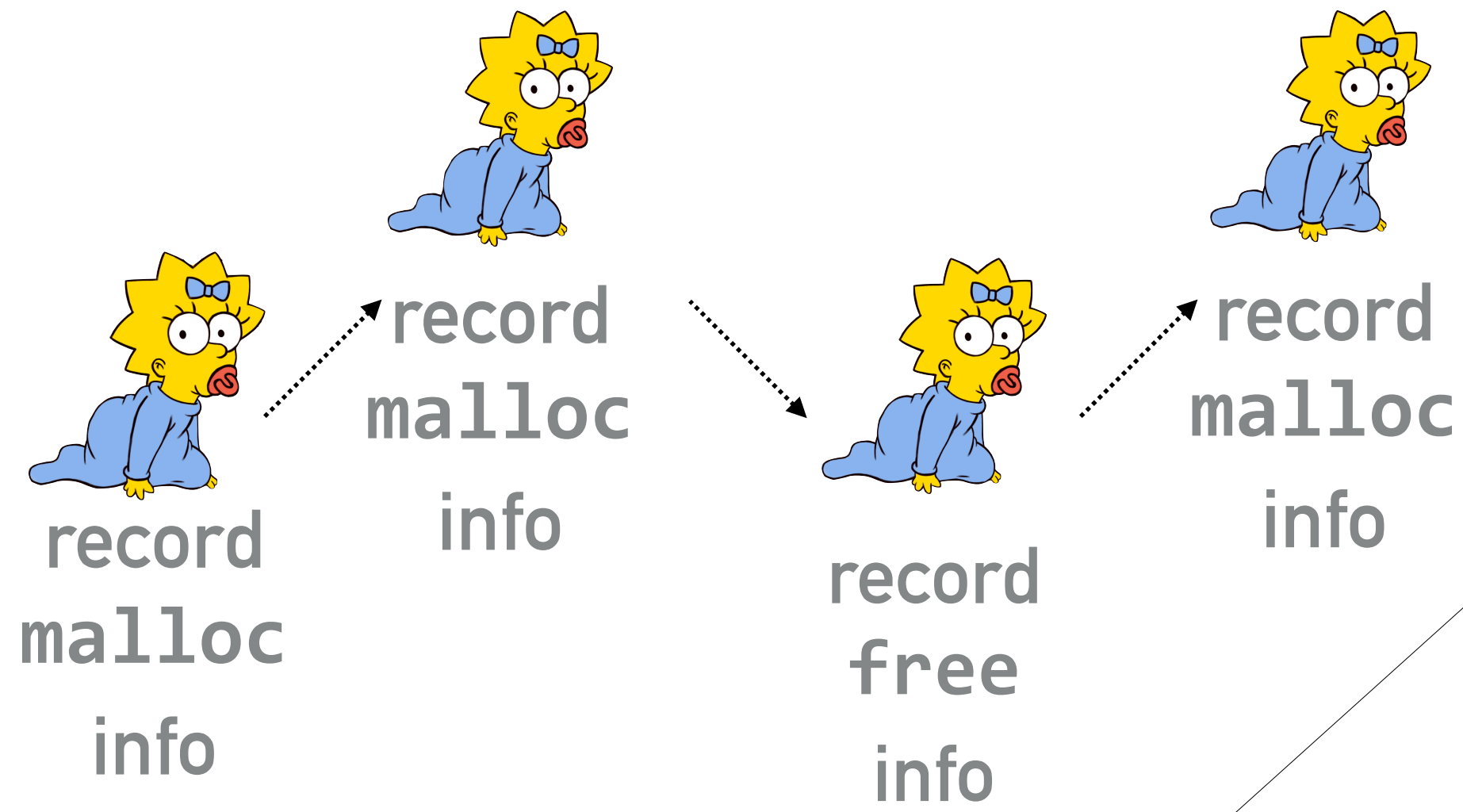


tracks every  
malloc/free

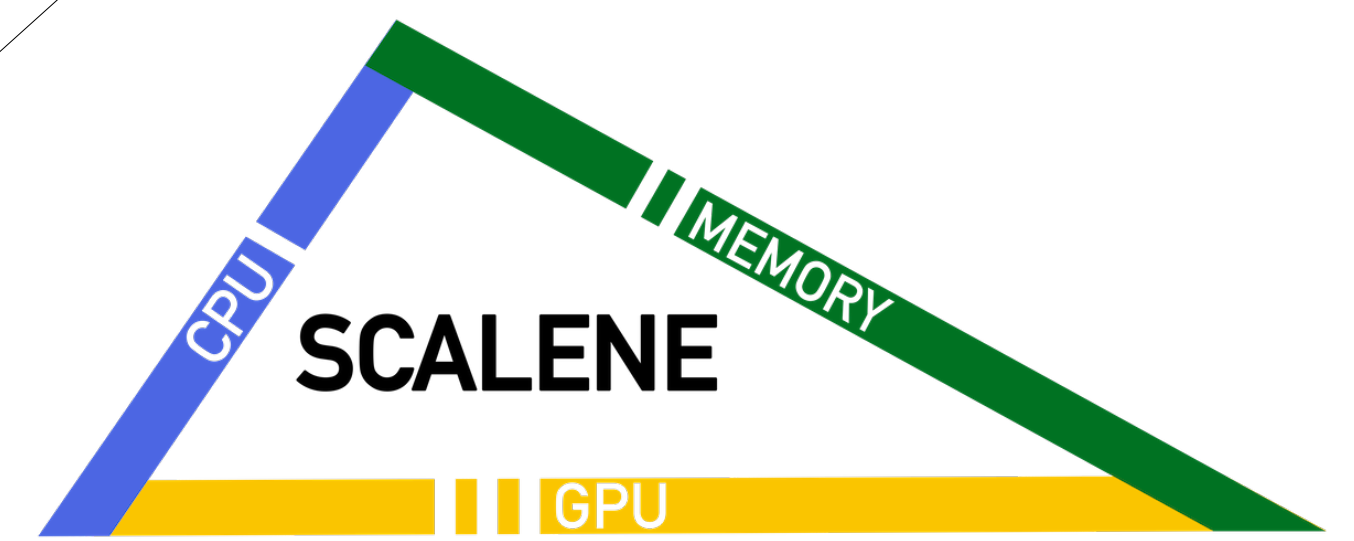


threshold-  
based  
sampling

memory\_profiler: ~300x slower

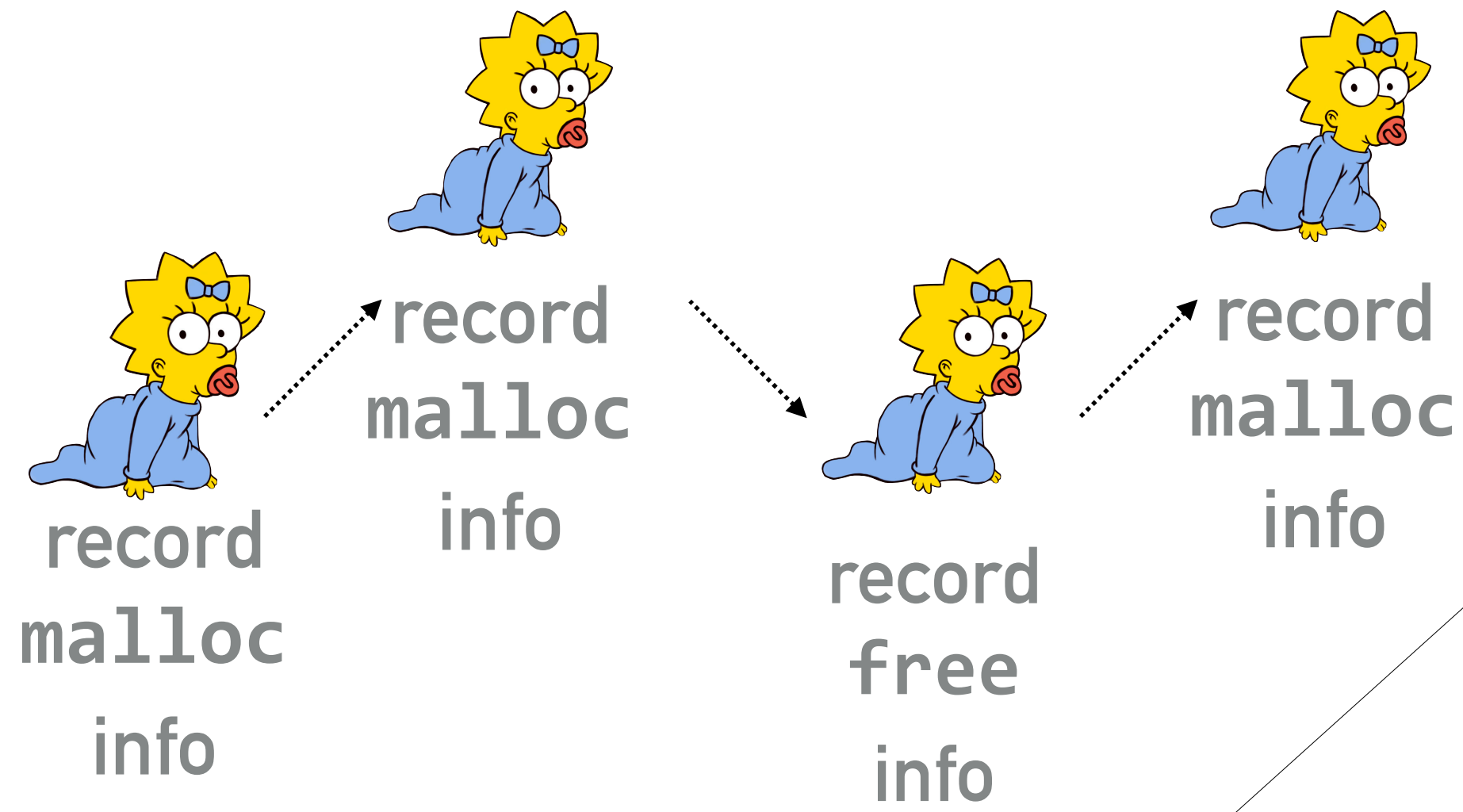


tracks every  
malloc/free

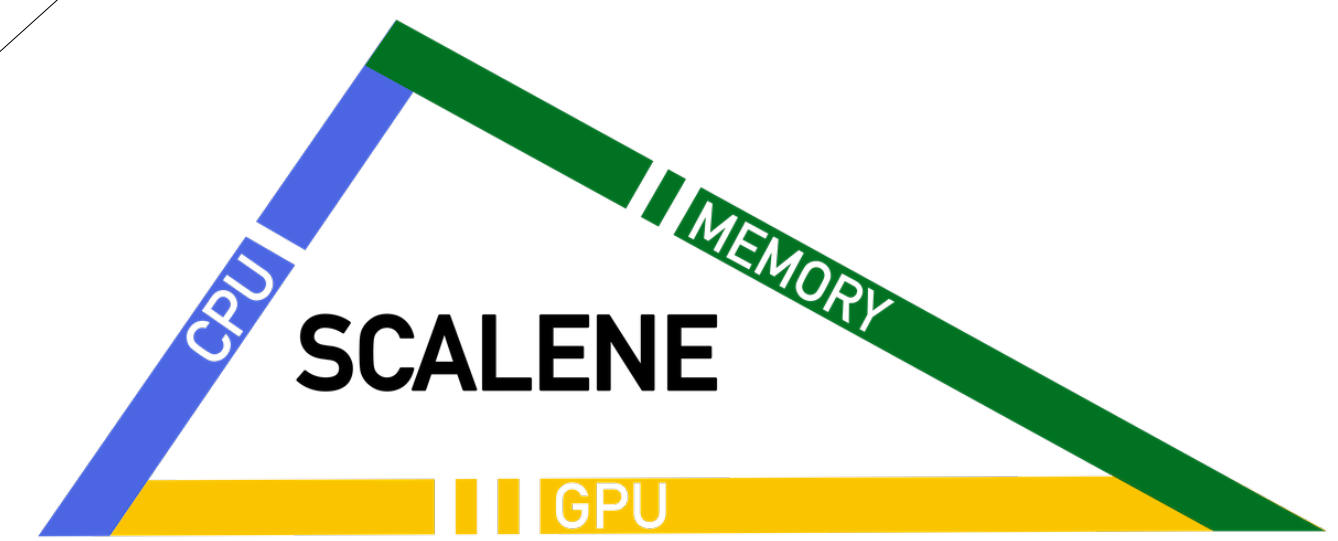
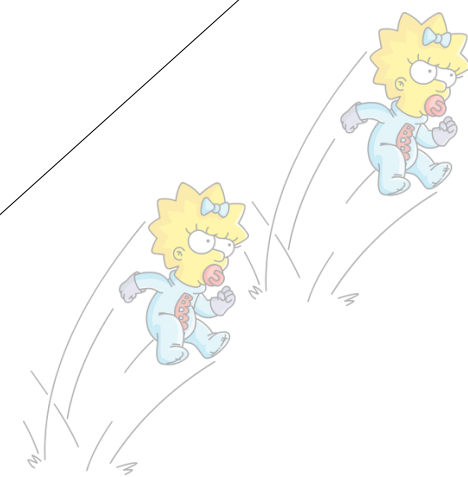


threshold-  
based  
sampling

memory\_profiler: ~300x slower

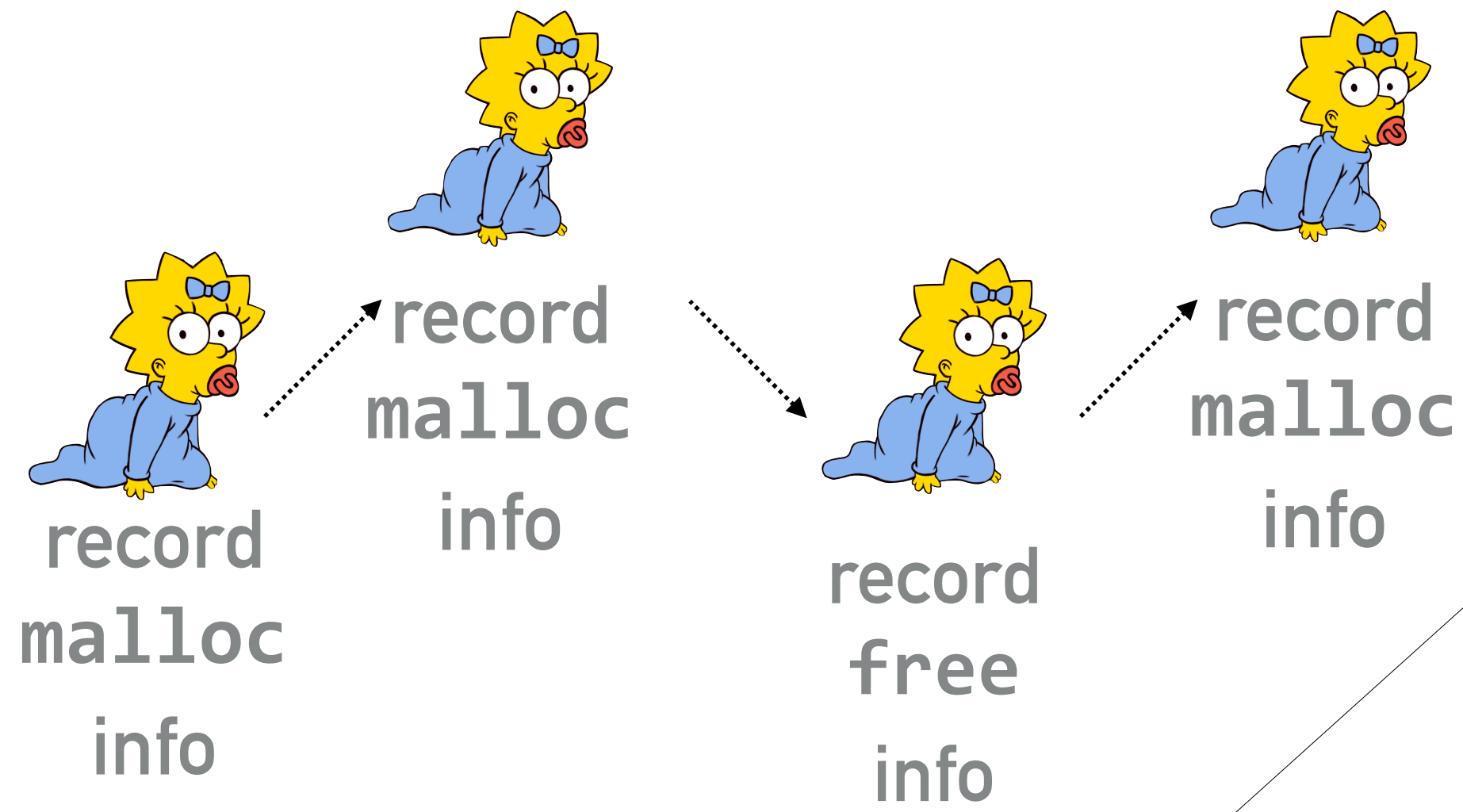


tracks every  
malloc/free

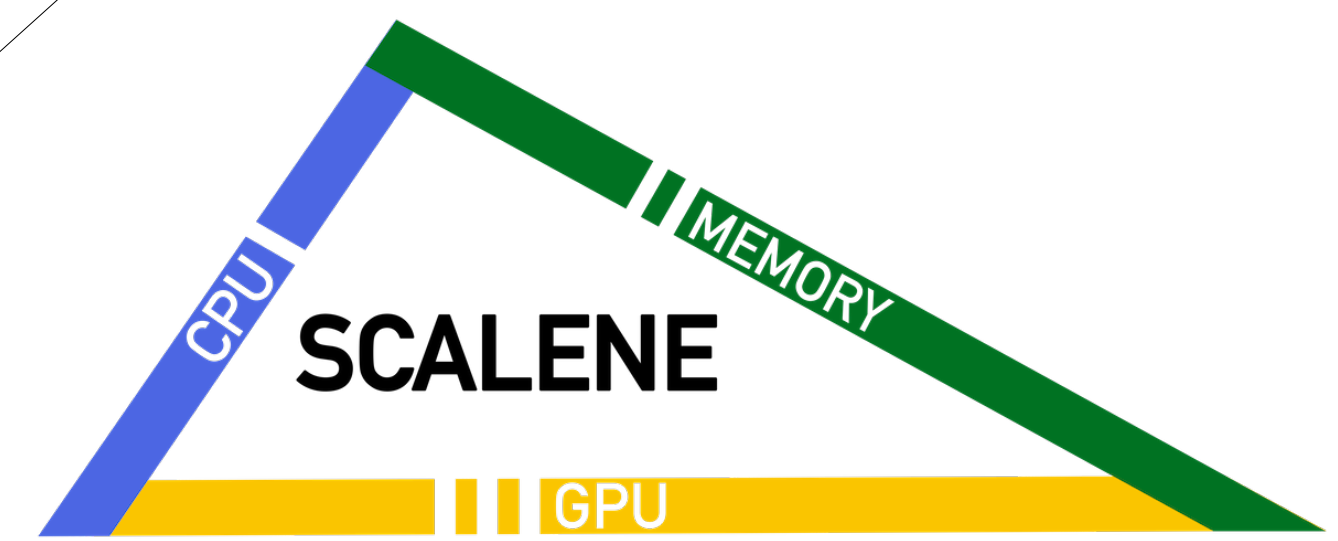
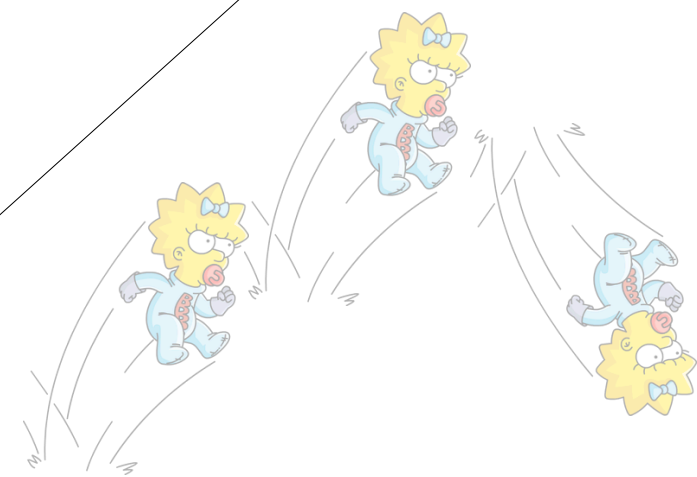


threshold-  
based  
sampling

memory\_profiler: ~300x slower

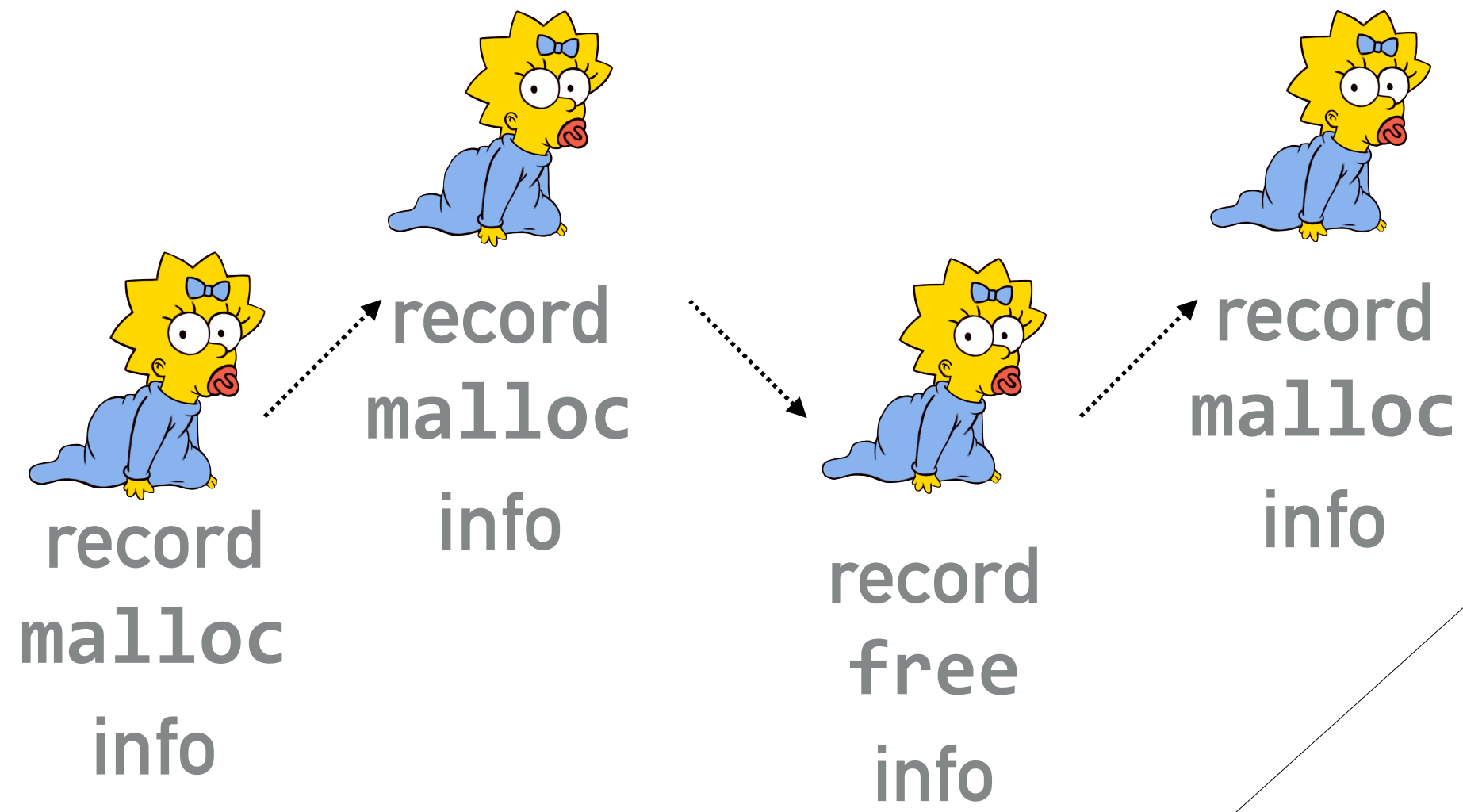


tracks every  
malloc/free

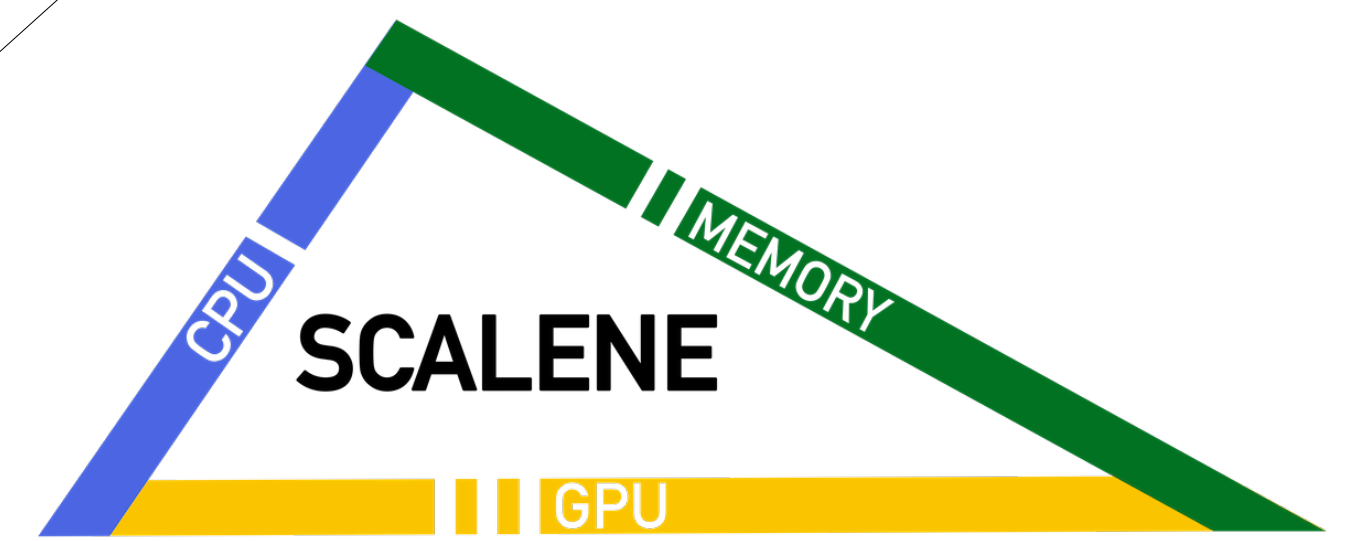
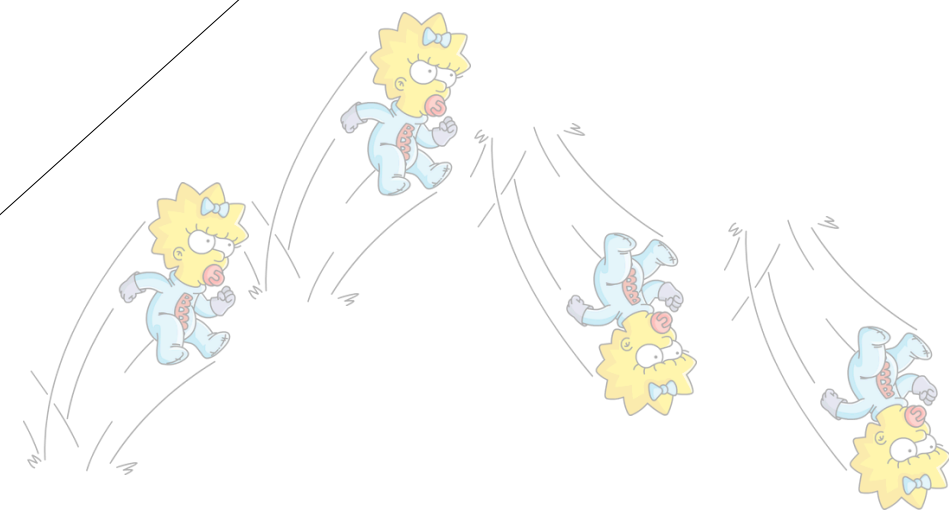


threshold-  
based  
sampling

memory\_profiler: ~300x slower



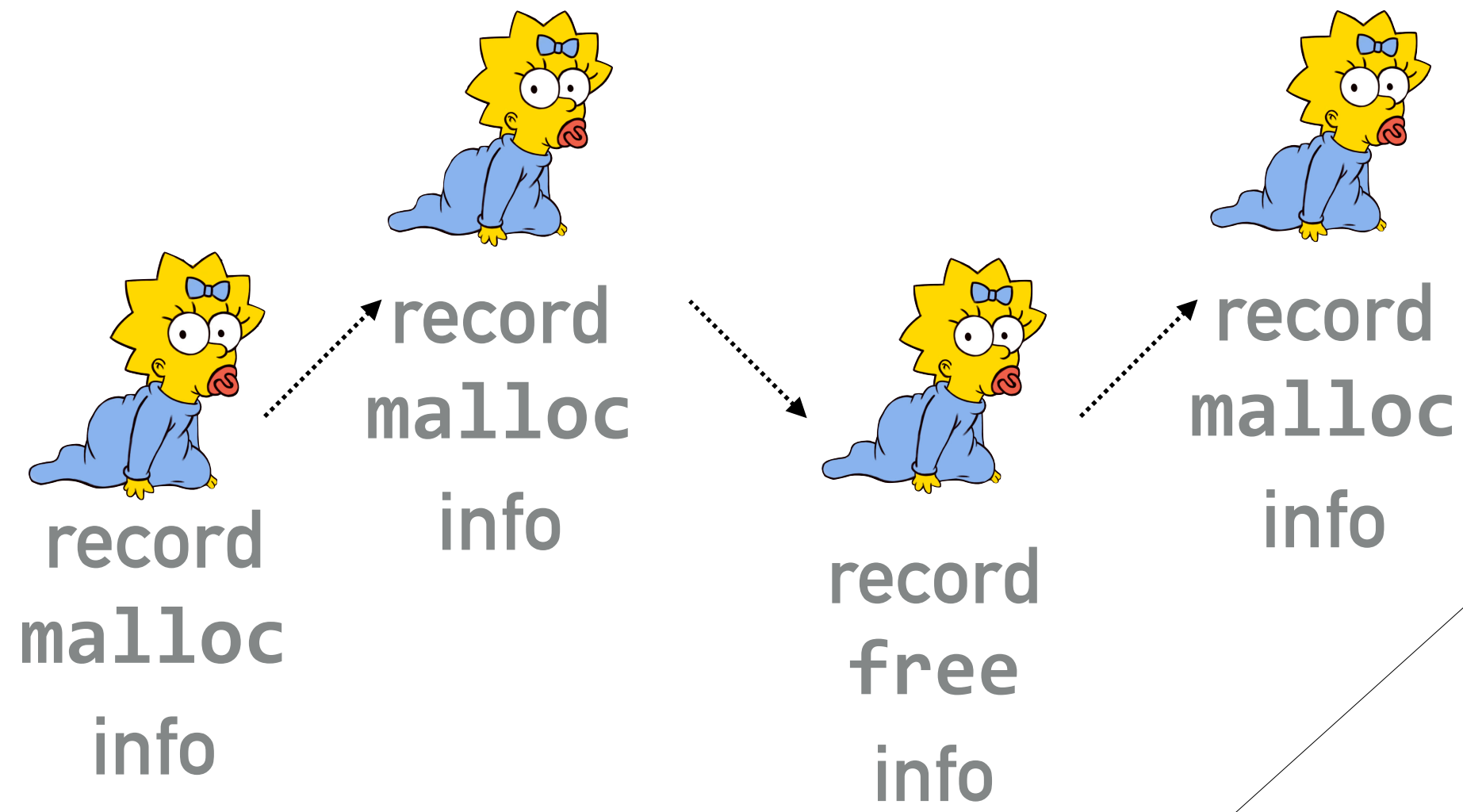
tracks every  
malloc/free



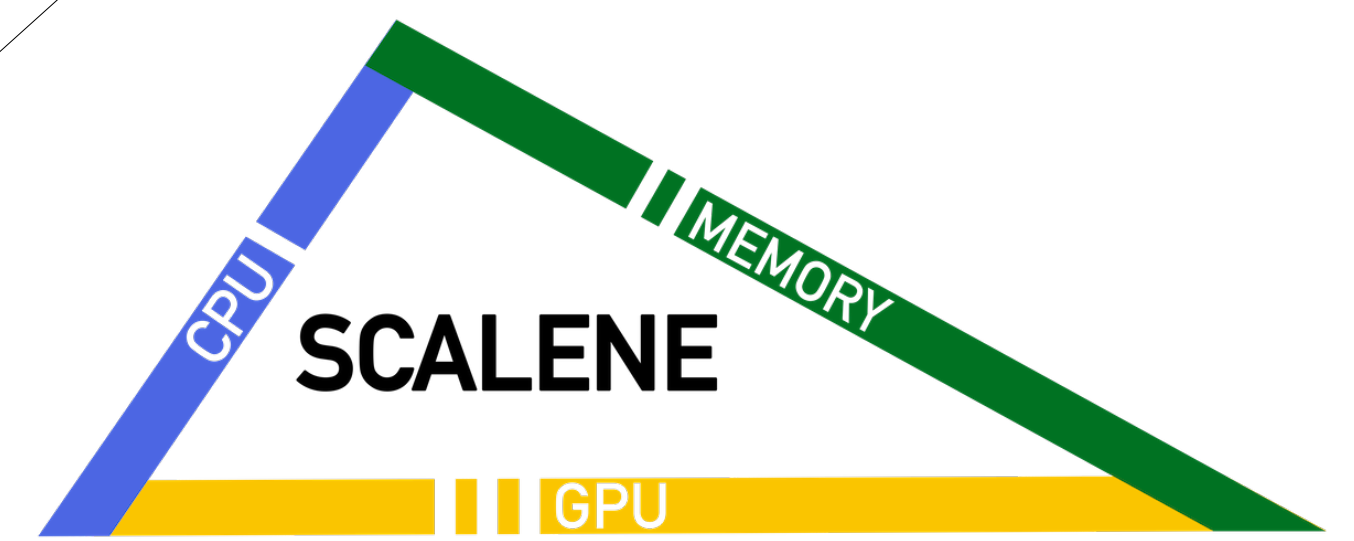
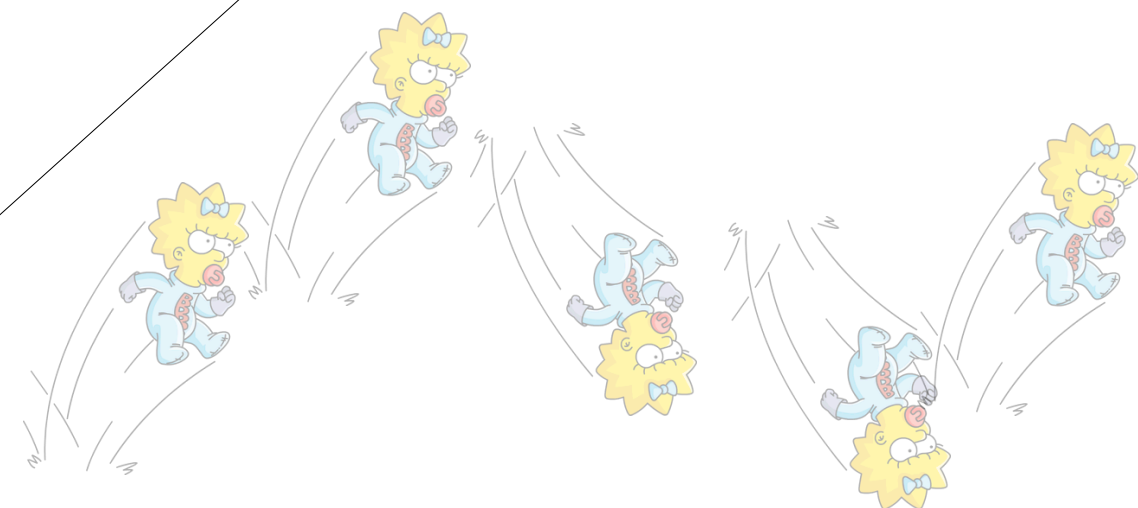
threshold-  
based  
sampling



memory\_profiler: ~300x slower

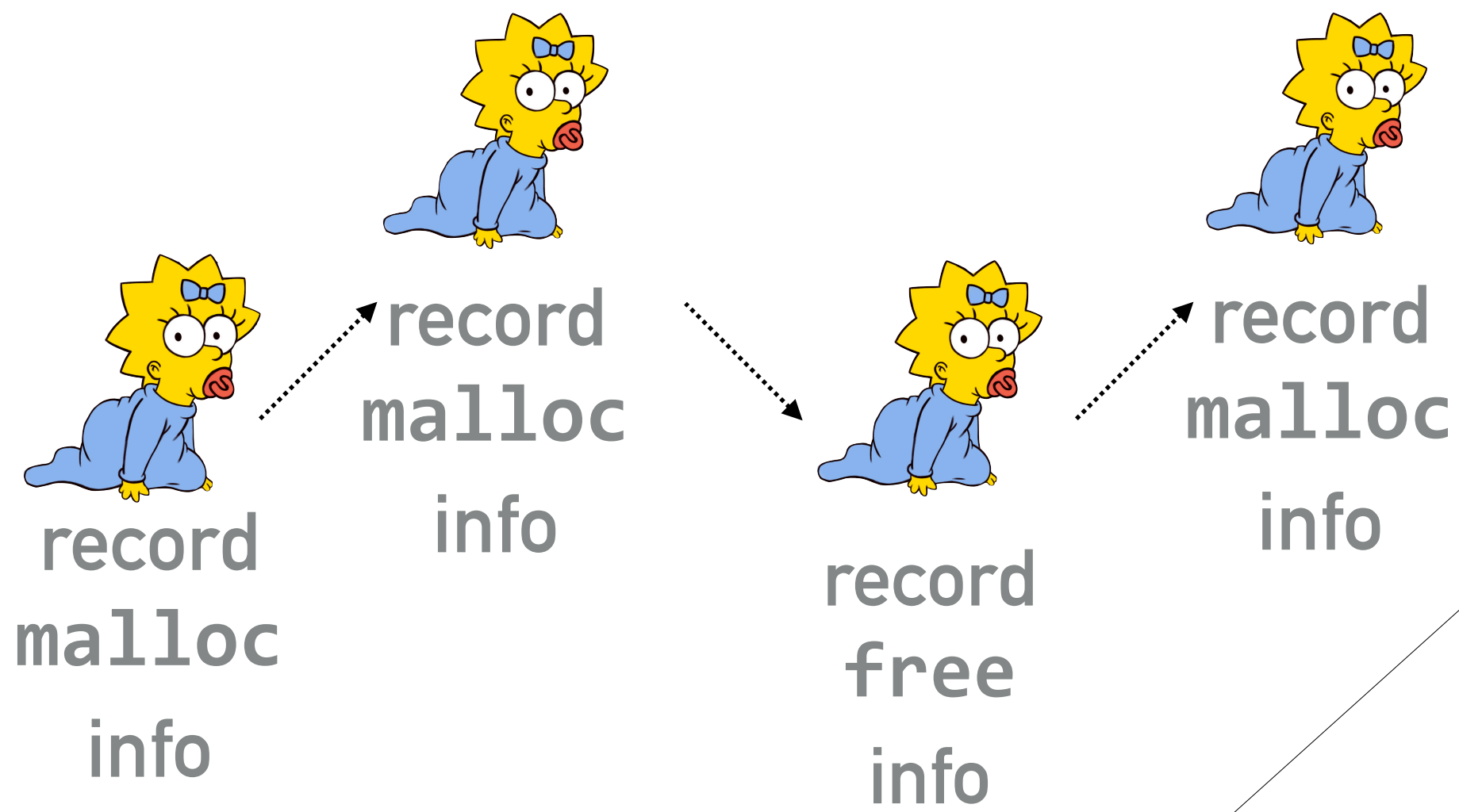


tracks every  
malloc/free

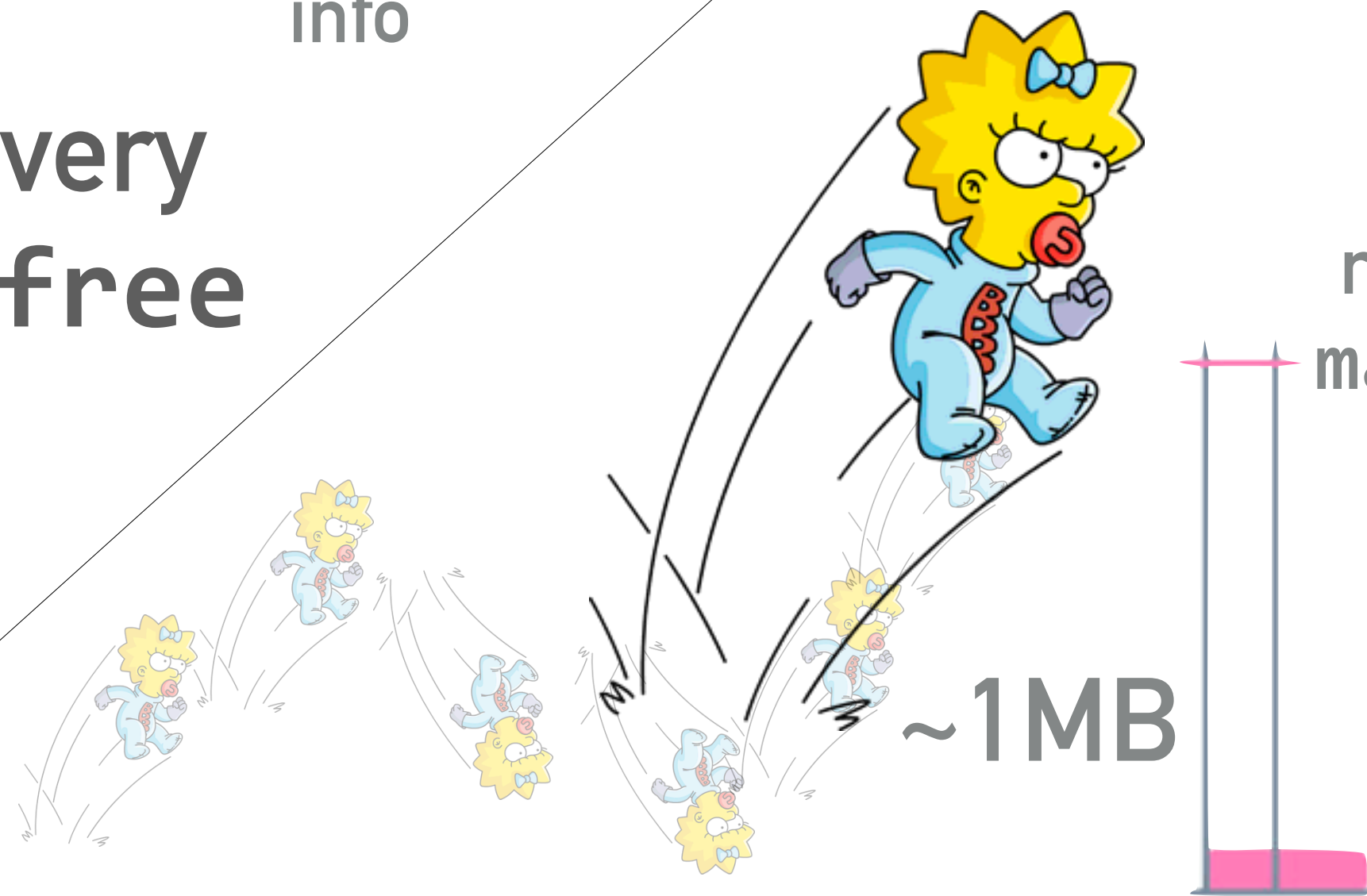


threshold-  
based  
sampling

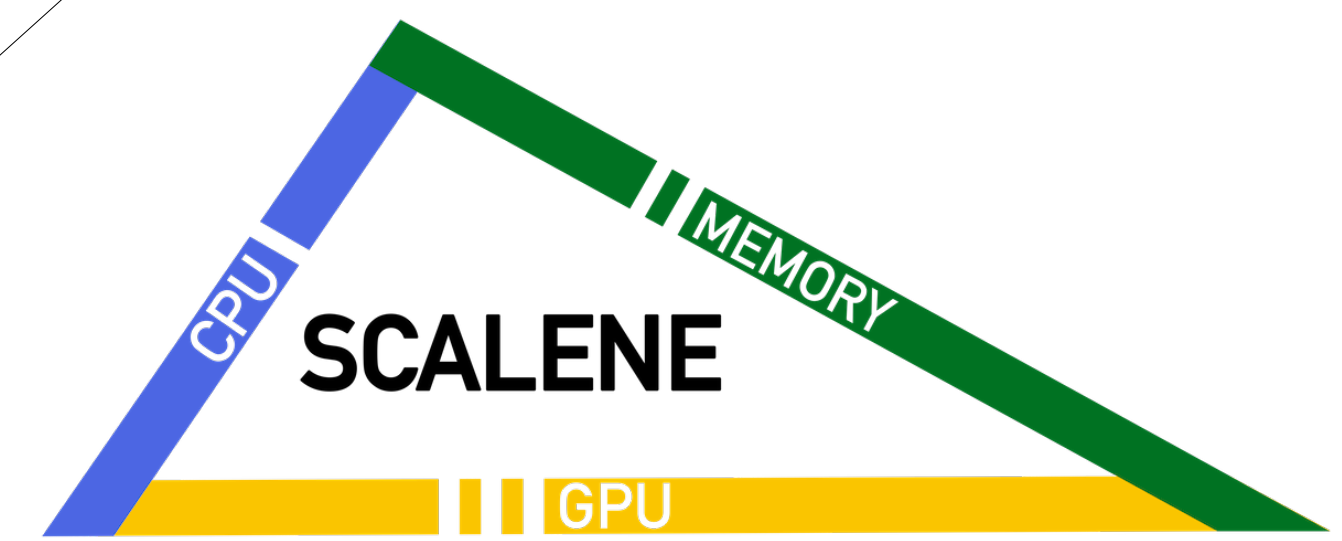
memory\_profiler: ~300x slower



tracks every malloc/free



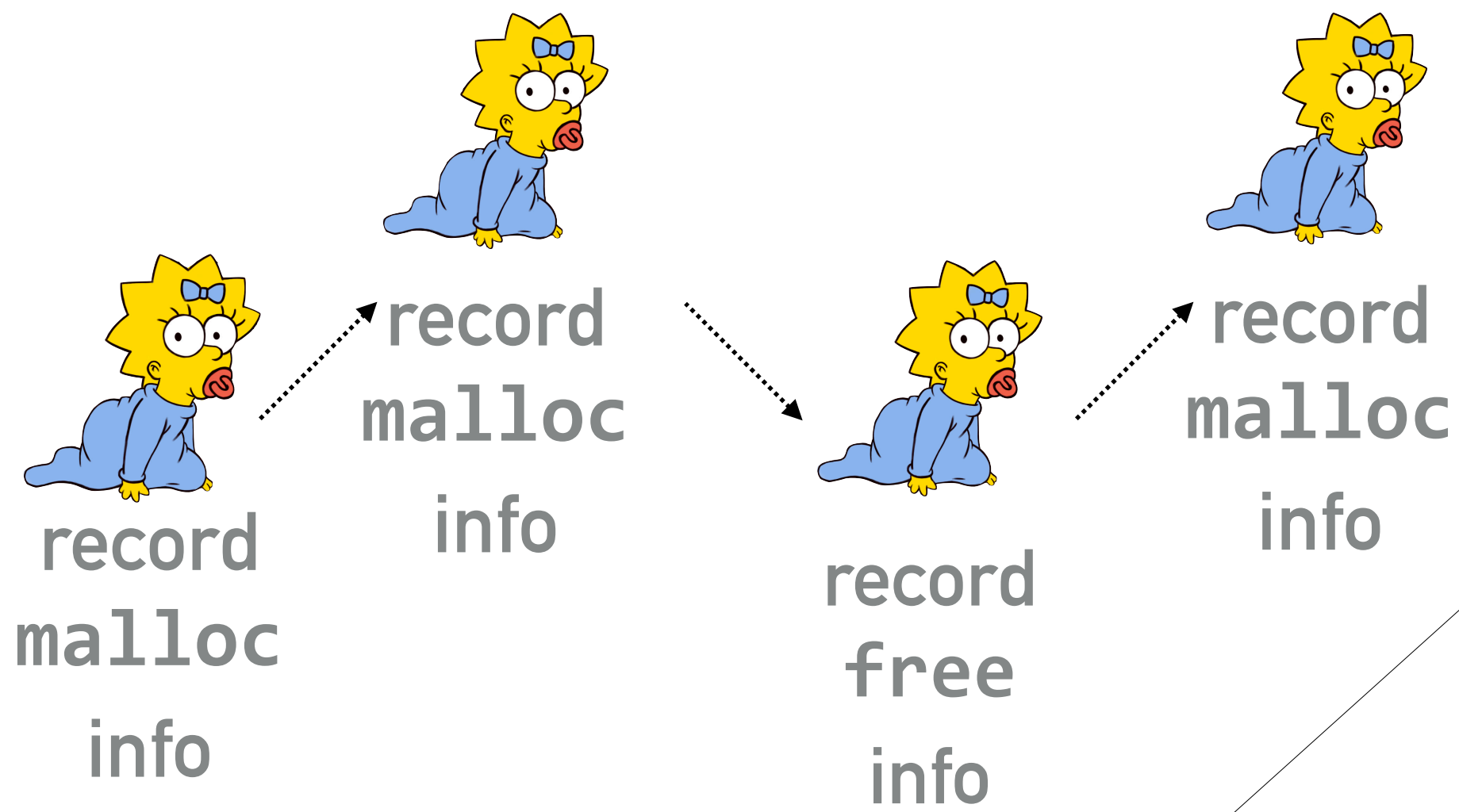
only tracks every  $\Delta \geq 1\text{MB}$



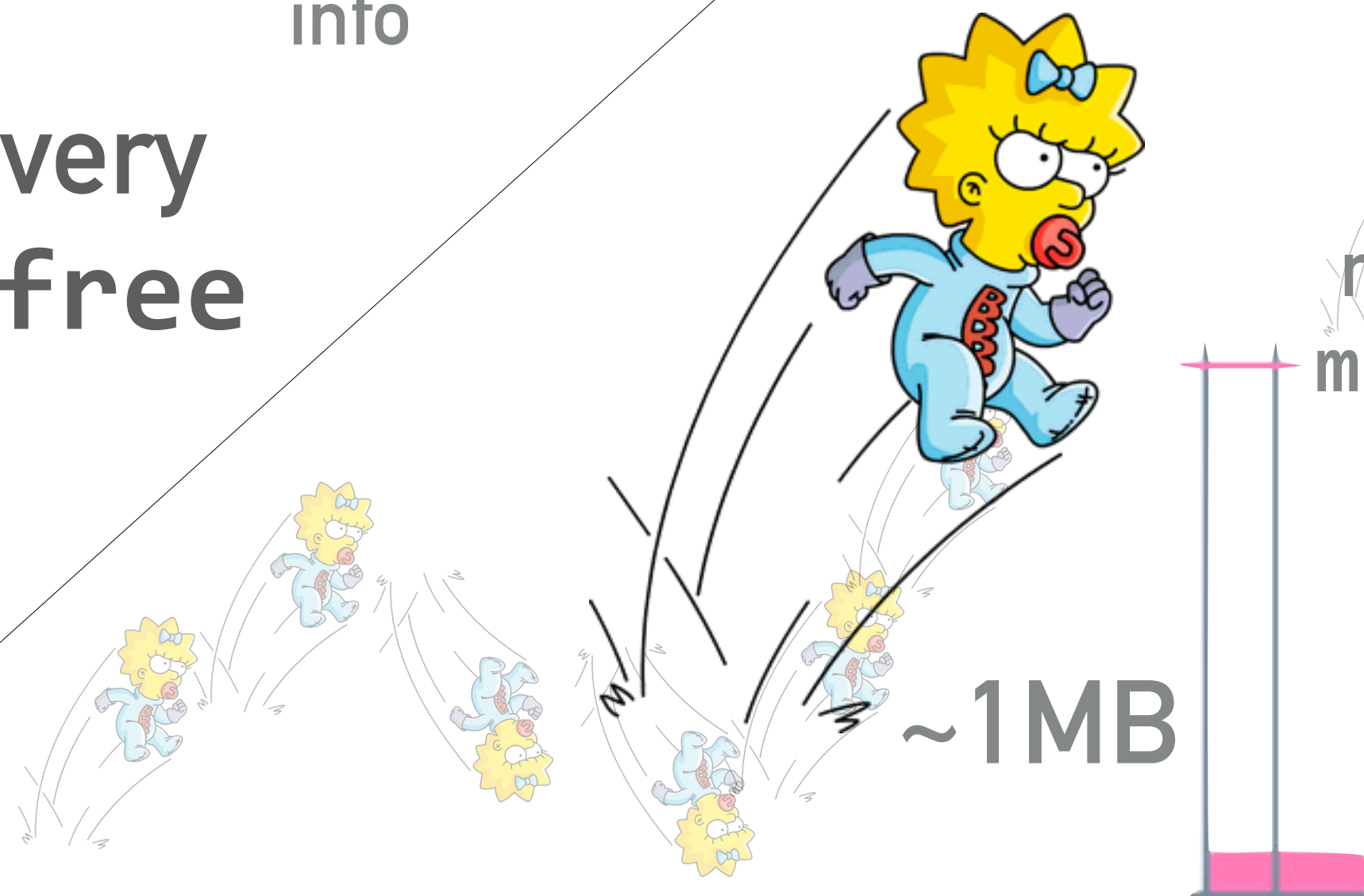
threshold-  
based  
sampling



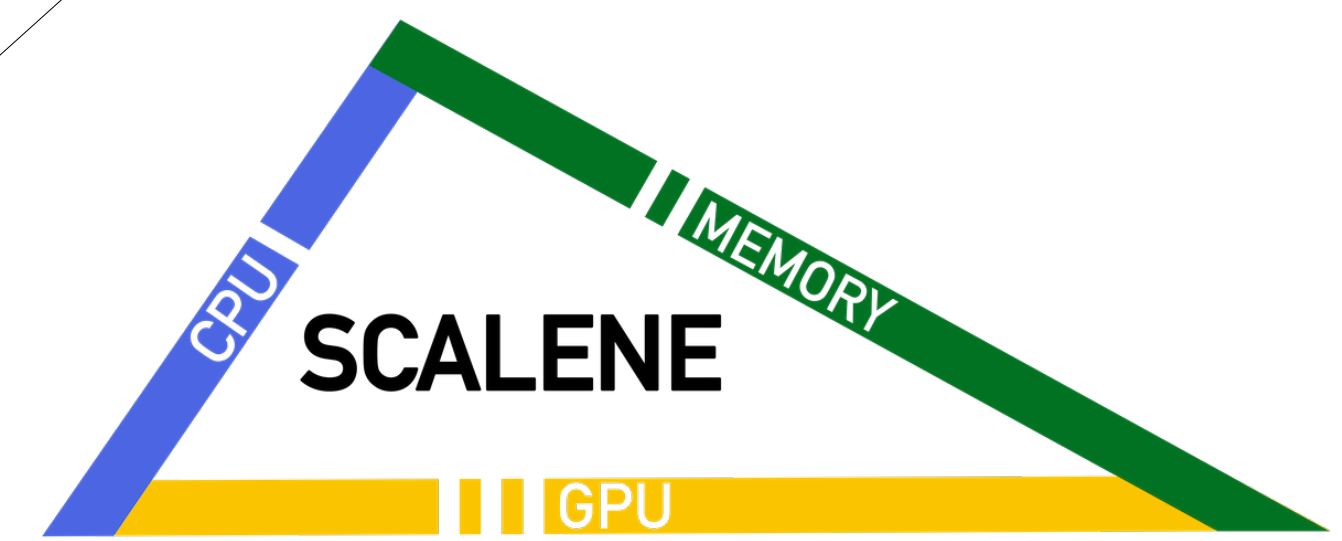
memory\_profiler: ~300x slower



tracks every malloc/free



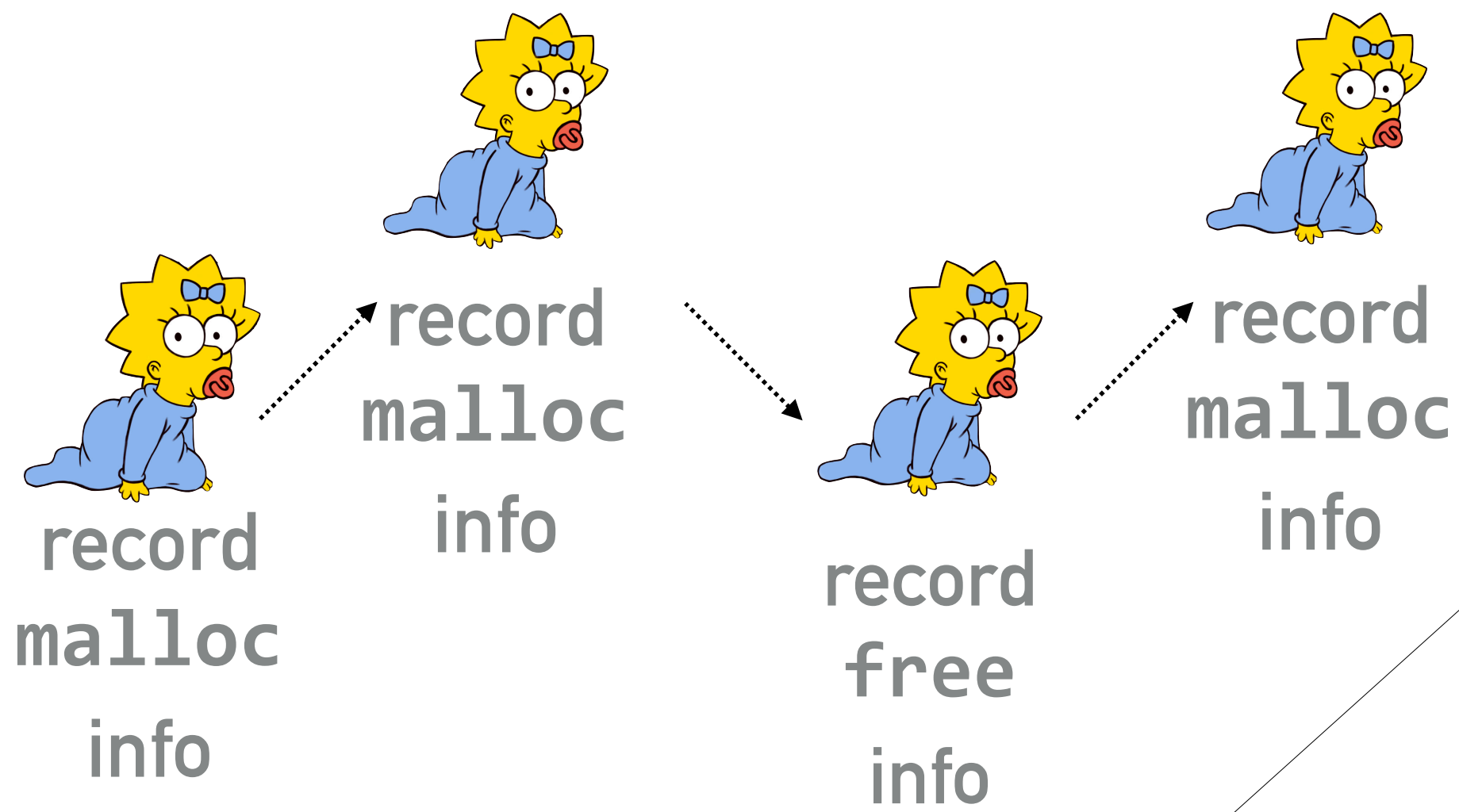
only tracks every  $\Delta \geq 1\text{MB}$



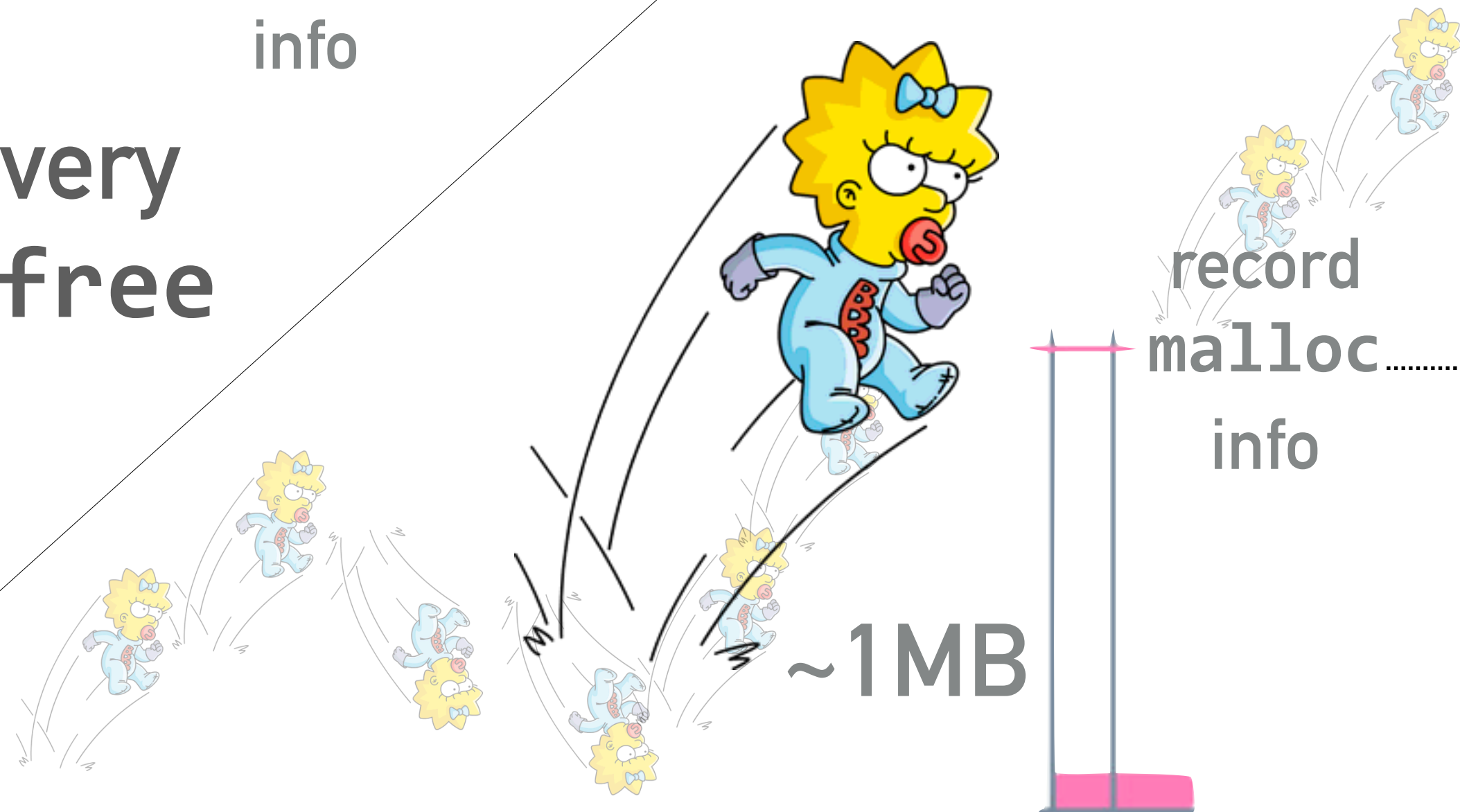
threshold-  
based  
sampling



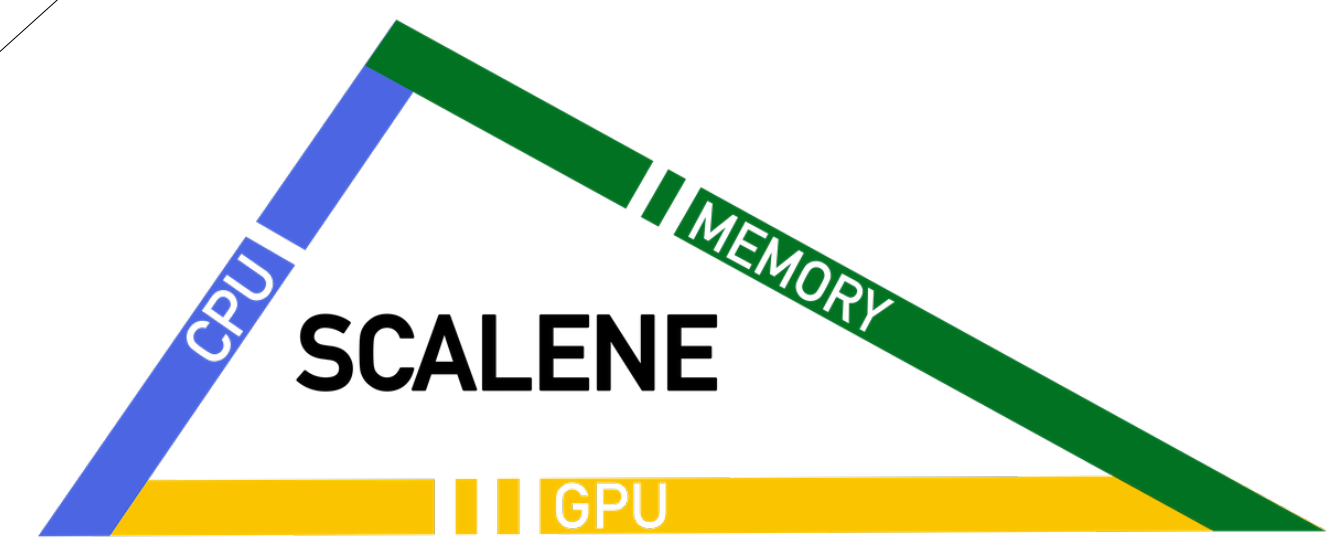
memory\_profiler: ~300x slower



tracks every  
malloc/free



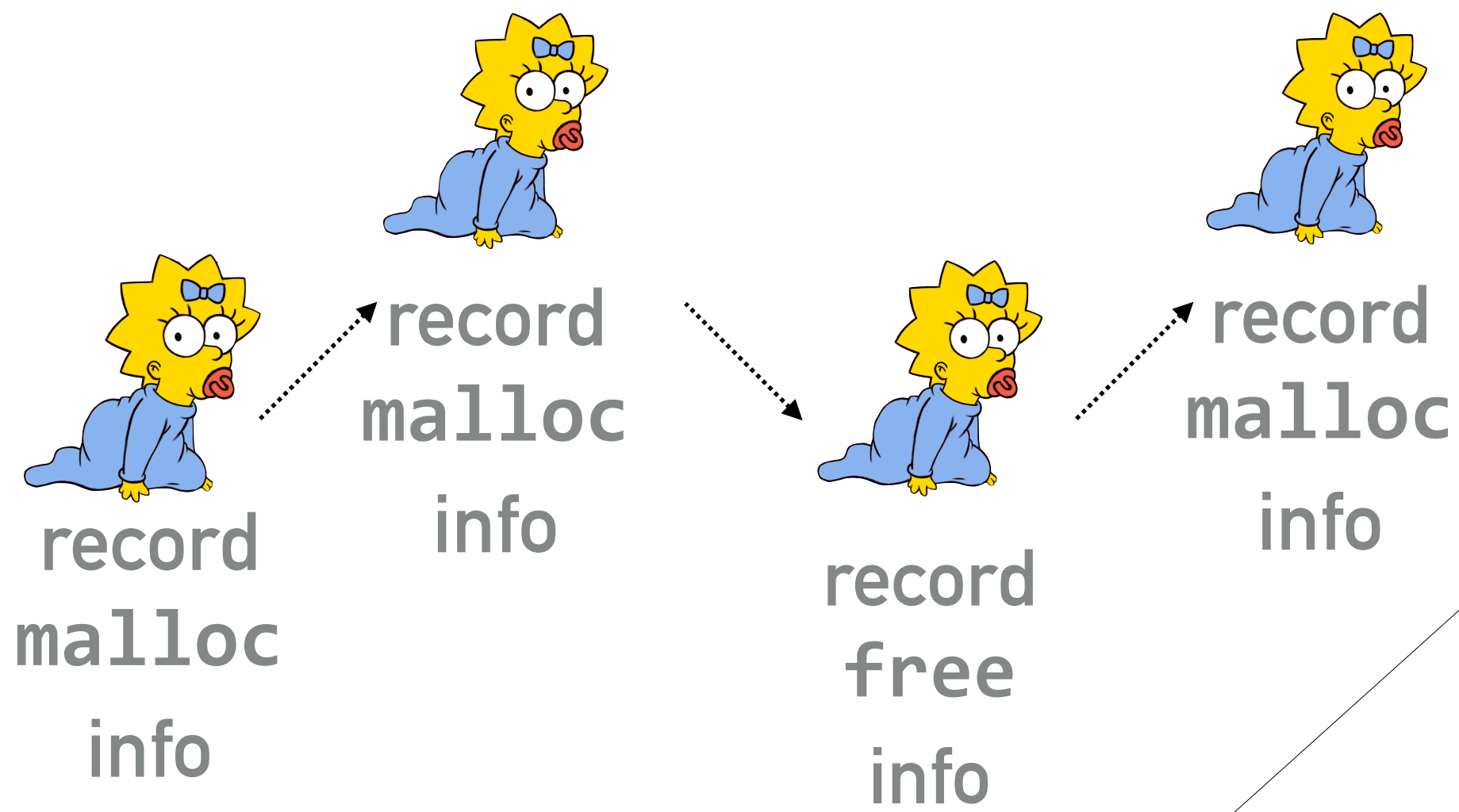
only tracks  
every  
 $\Delta \geq 1\text{MB}$



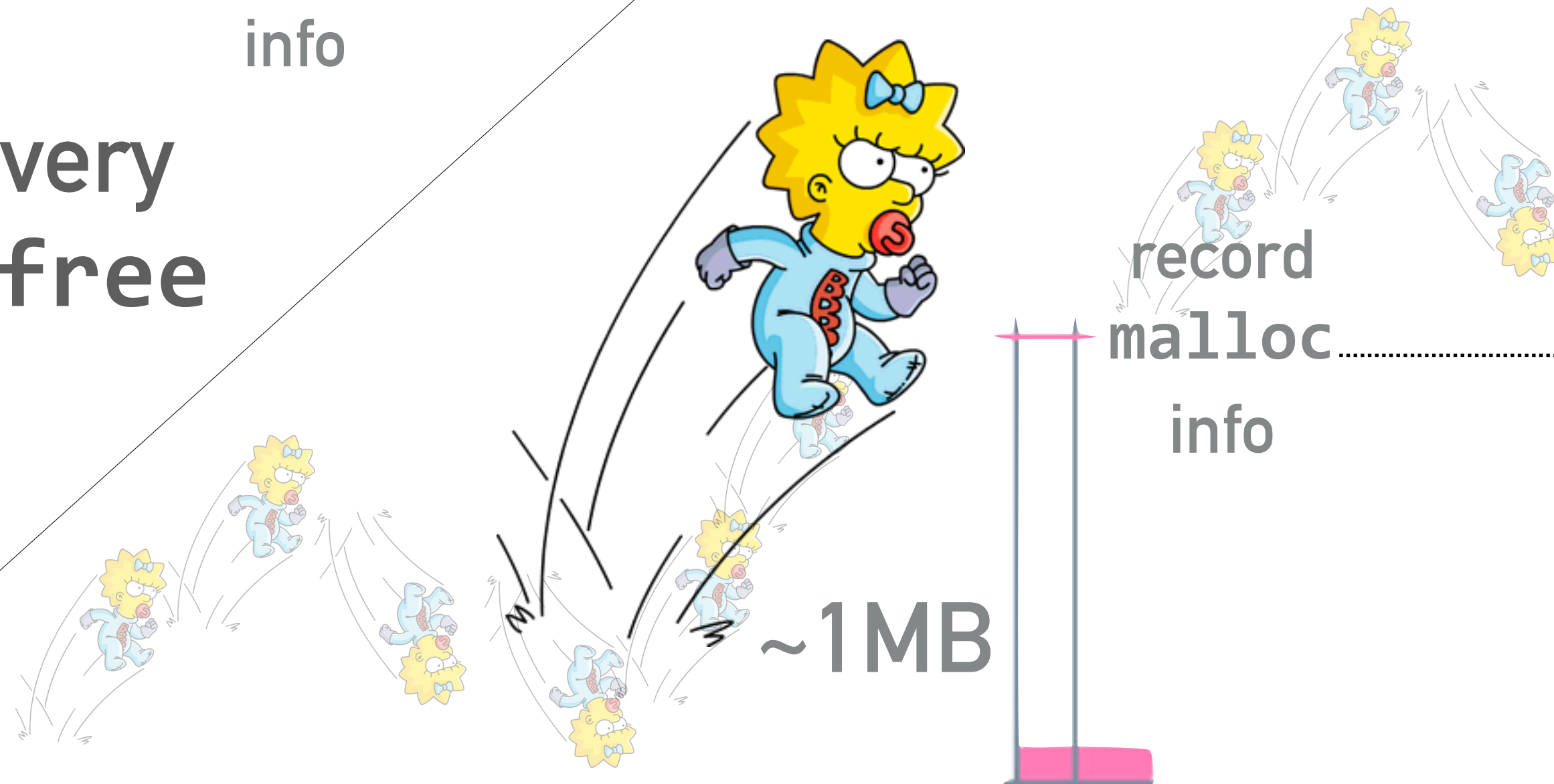
threshold-  
based  
sampling



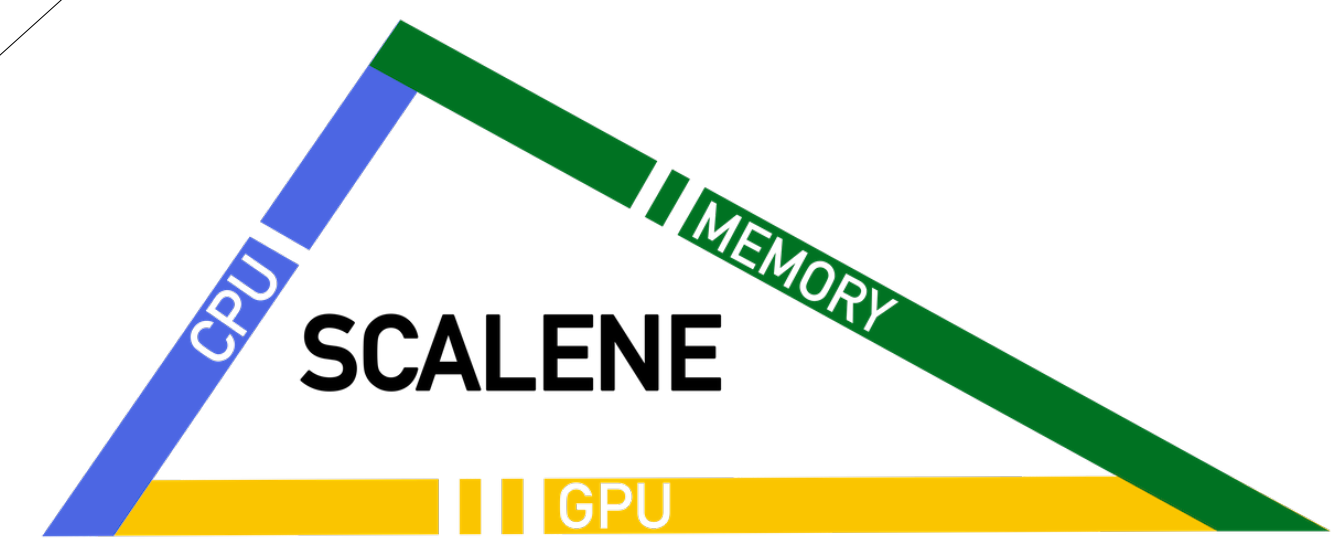
memory\_profiler: ~300x slower



tracks every malloc/free



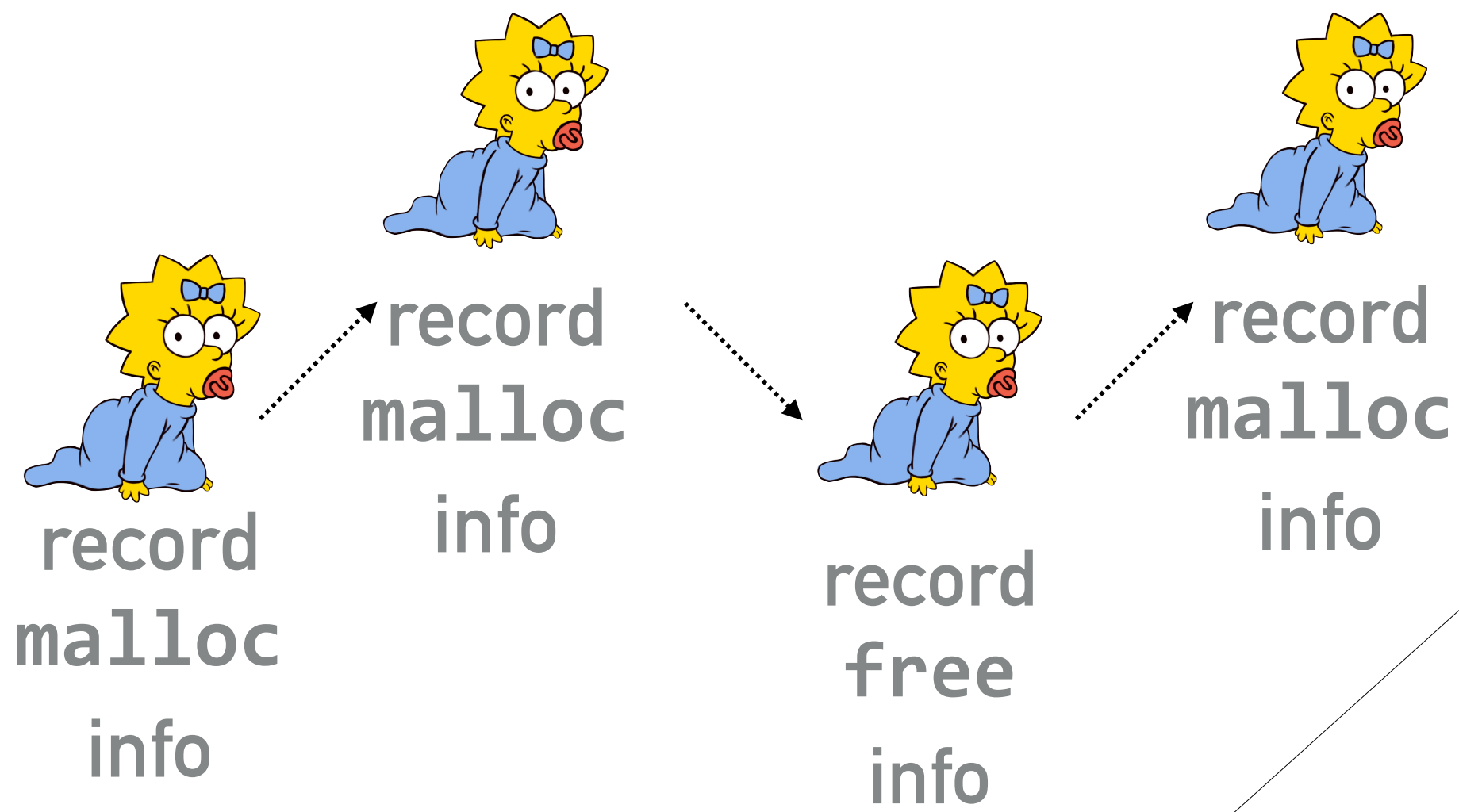
only tracks every  $\Delta \geq 1\text{MB}$



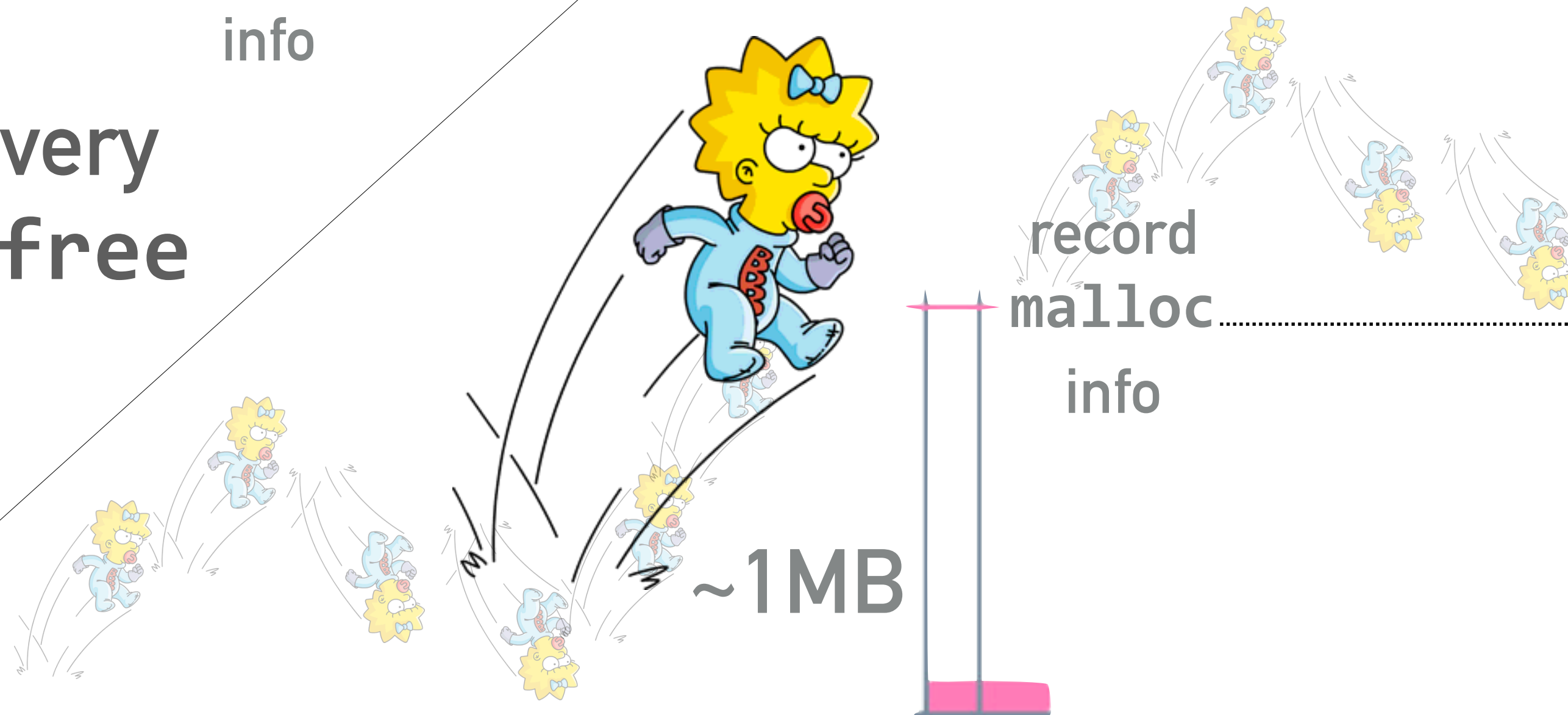
threshold-  
based  
sampling



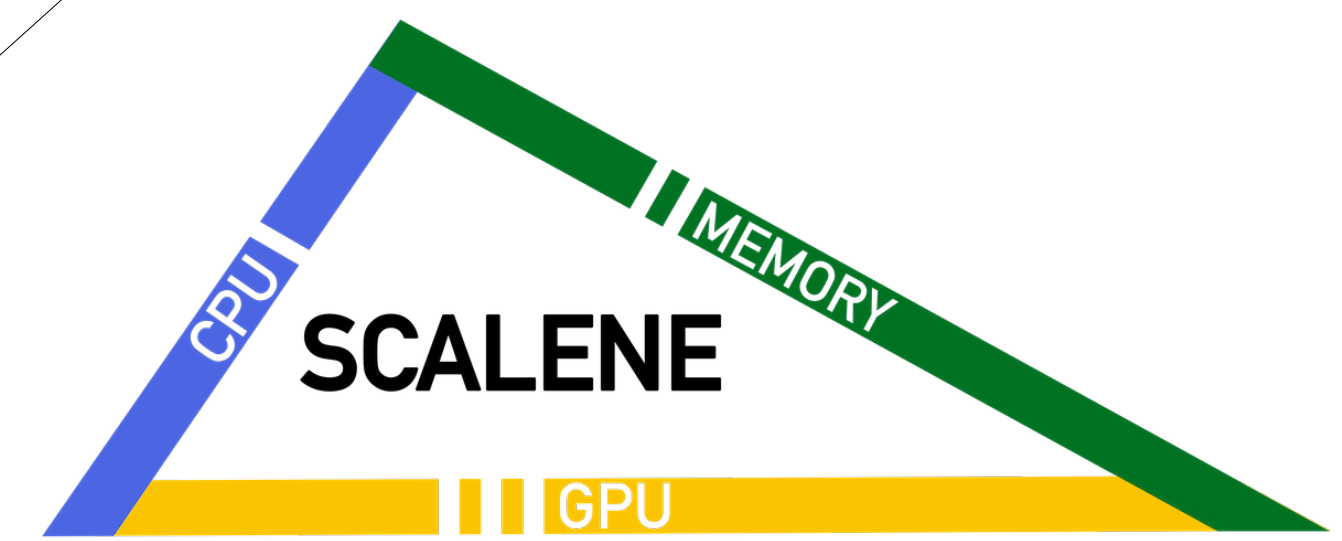
memory\_profiler: ~300x slower



tracks every malloc/free



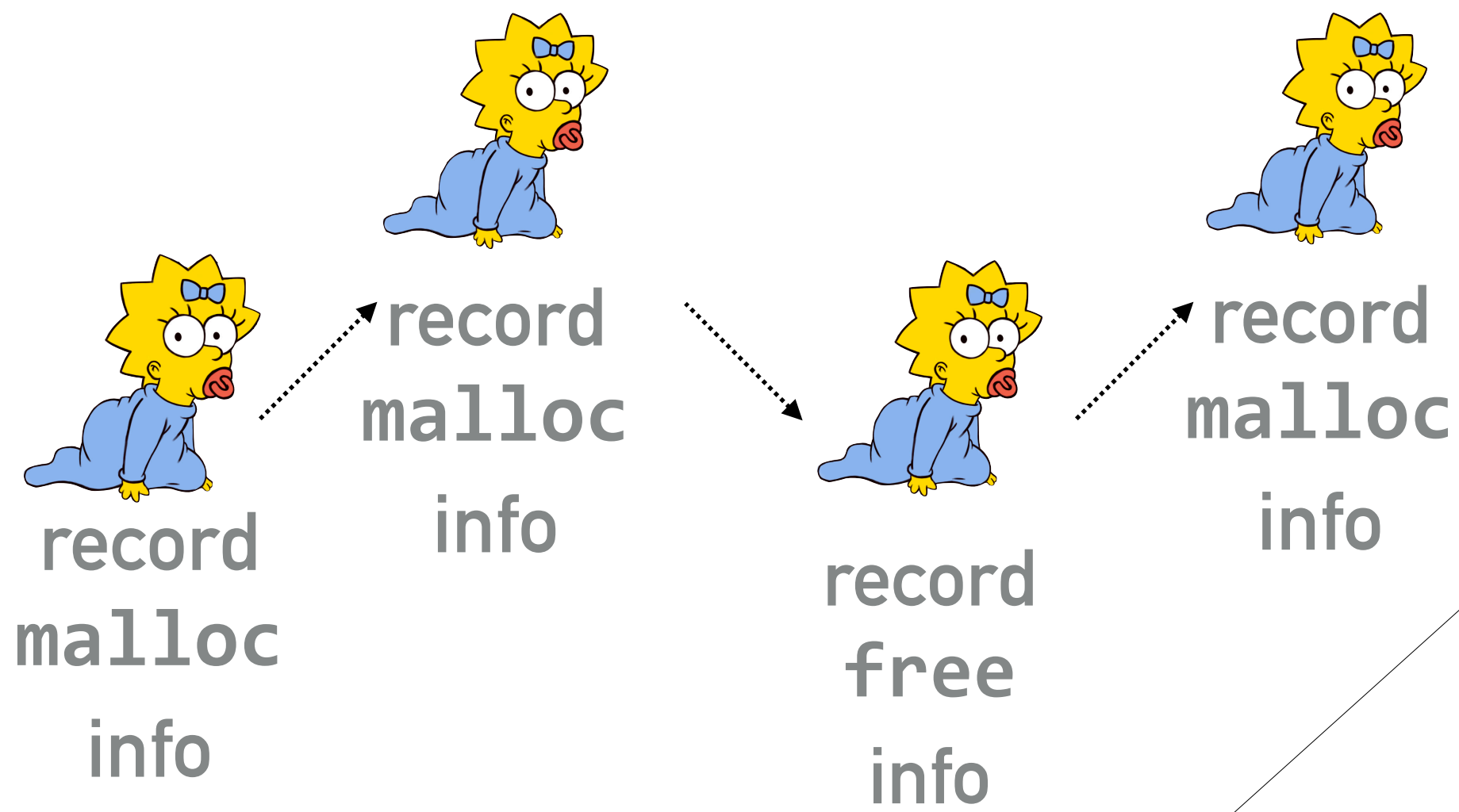
only tracks every  $\Delta \geq 1\text{MB}$



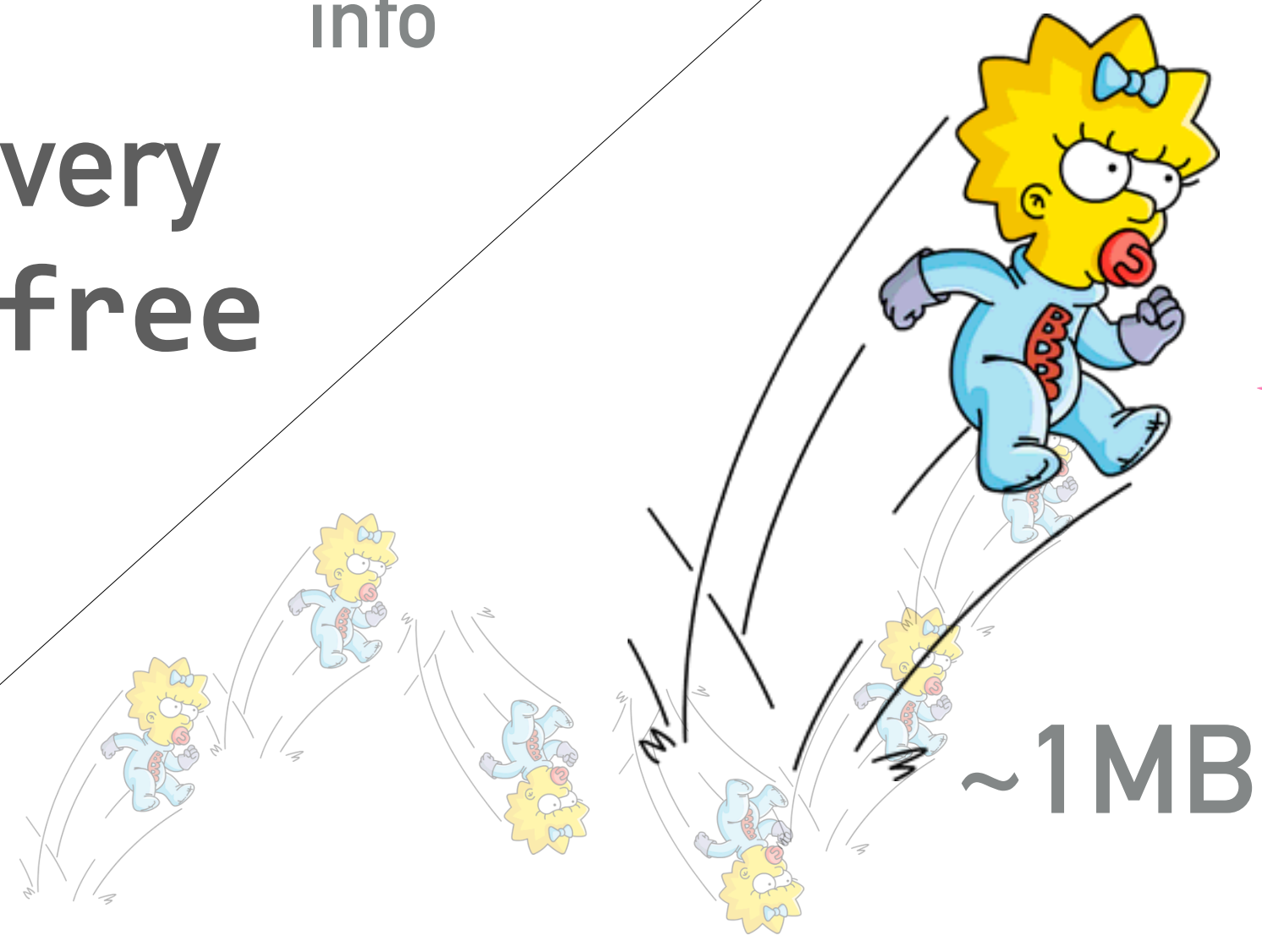
threshold-  
based  
sampling



memory\_profiler: ~300x slower

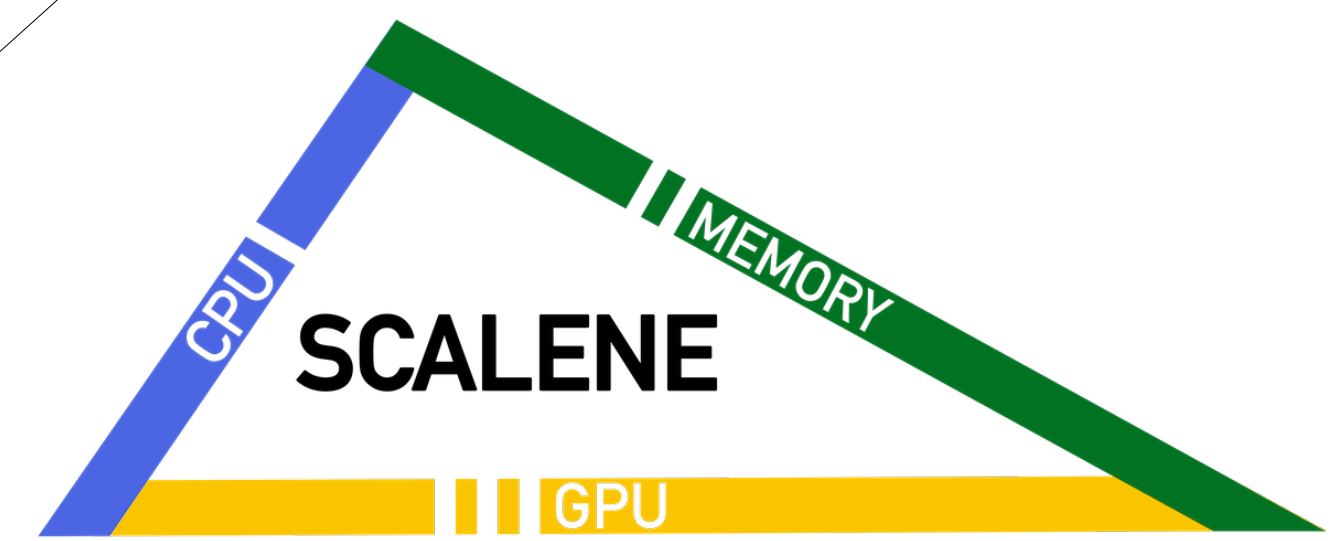


tracks every  
malloc/free



record  
malloc  
info

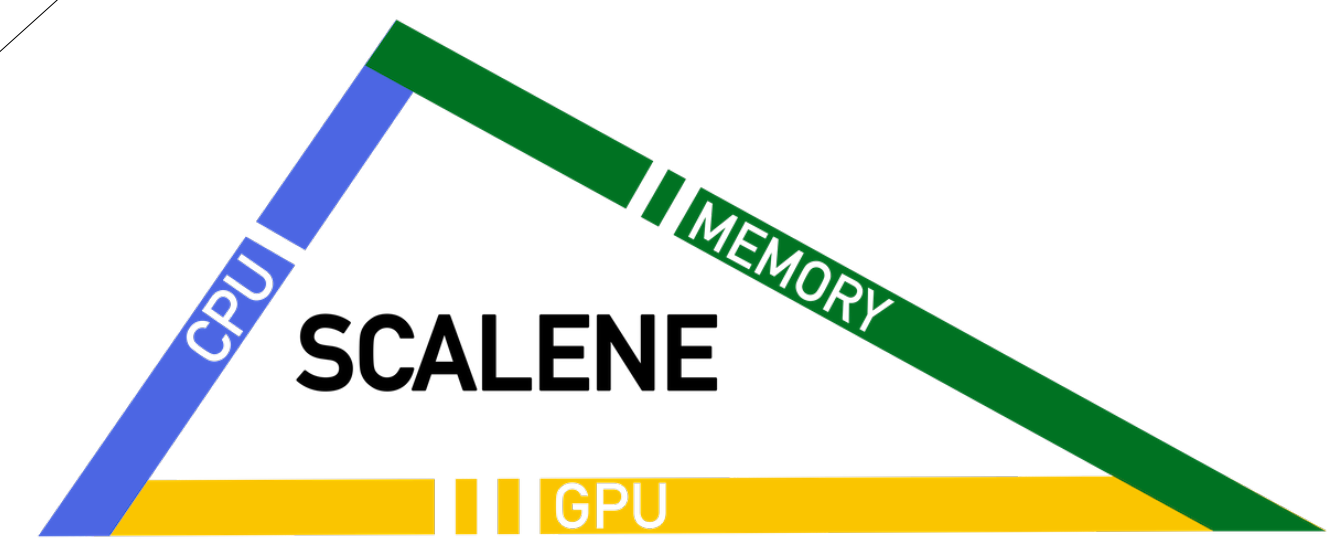
only tracks  
every  
 $\Delta \geq 1\text{MB}$



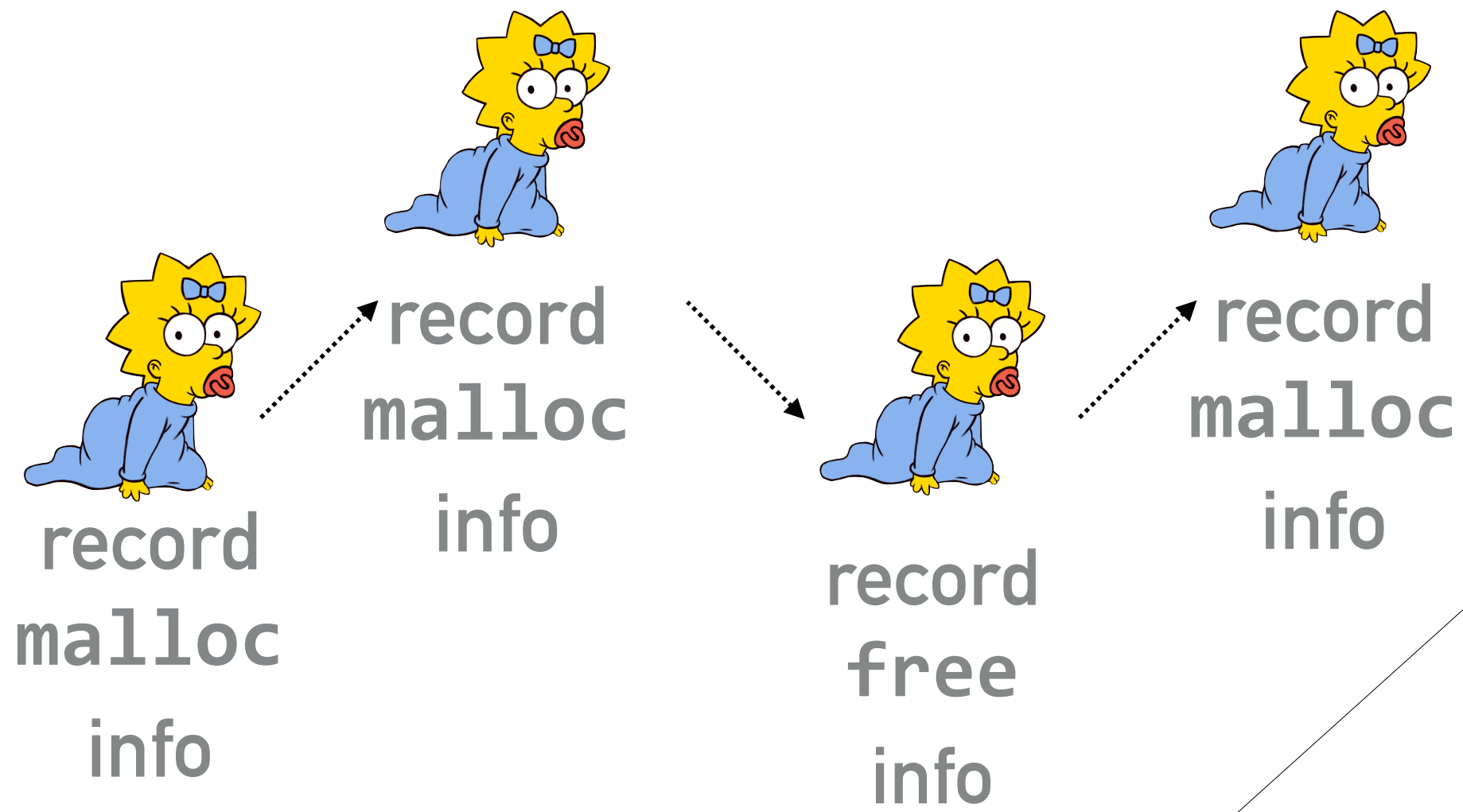
threshold-  
based  
sampling



memory\_profiler: ~300x slower

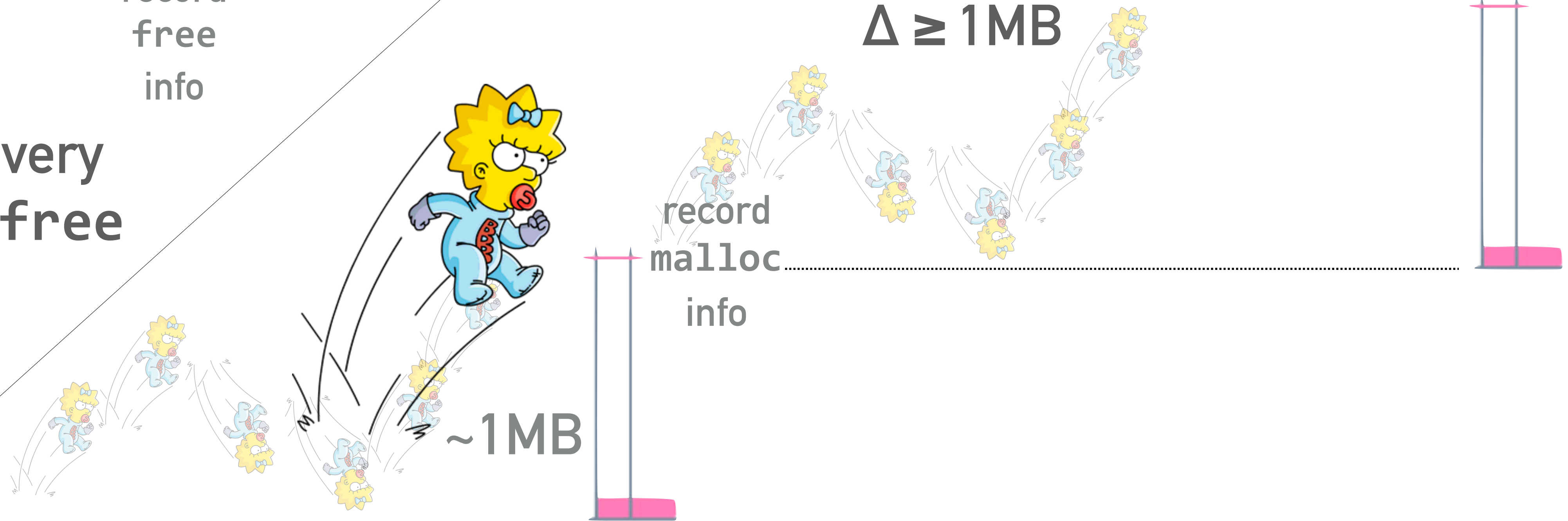


threshold-  
based  
sampling



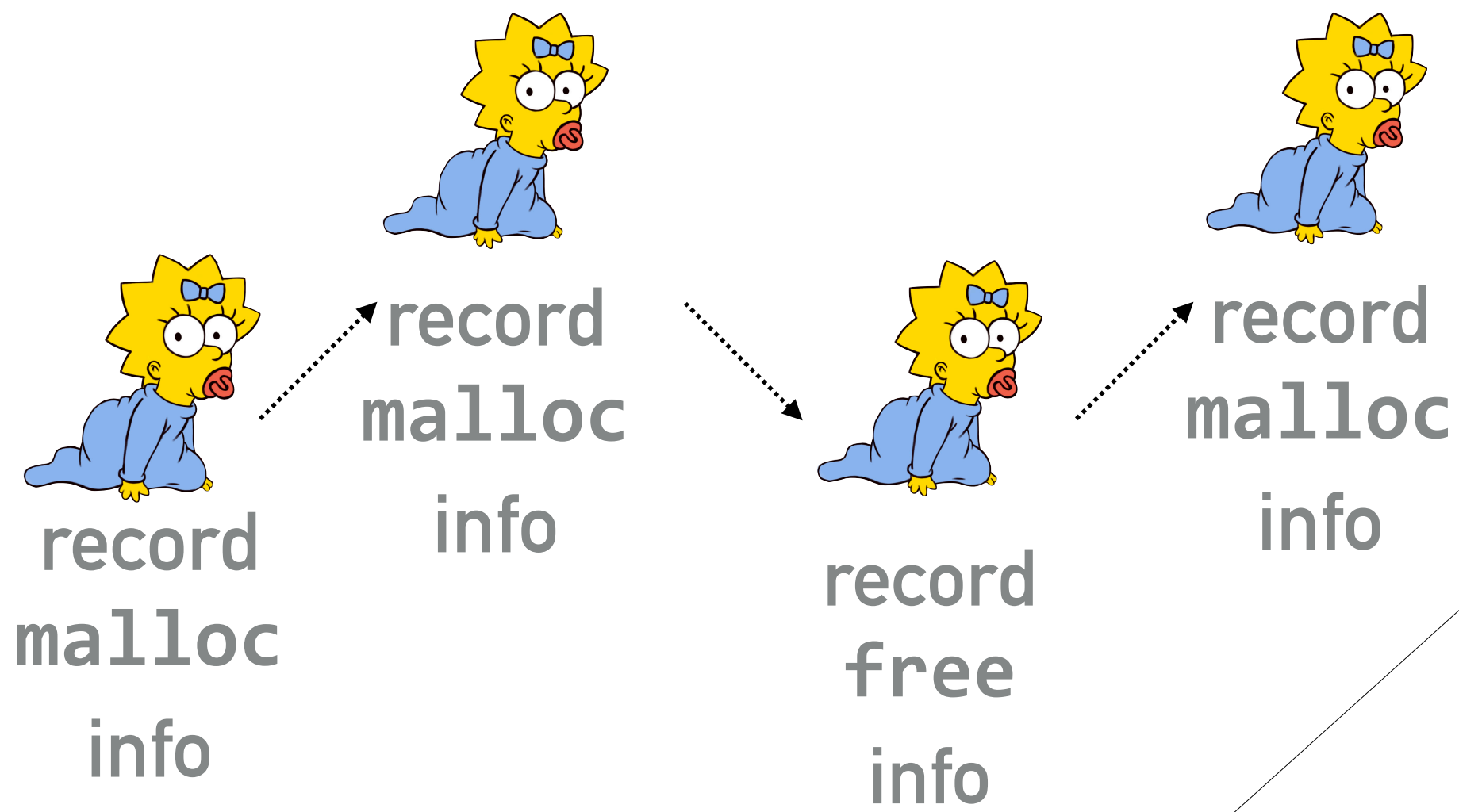
tracks every  
malloc/free

only tracks  
every  
 $\Delta \geq 1\text{MB}$

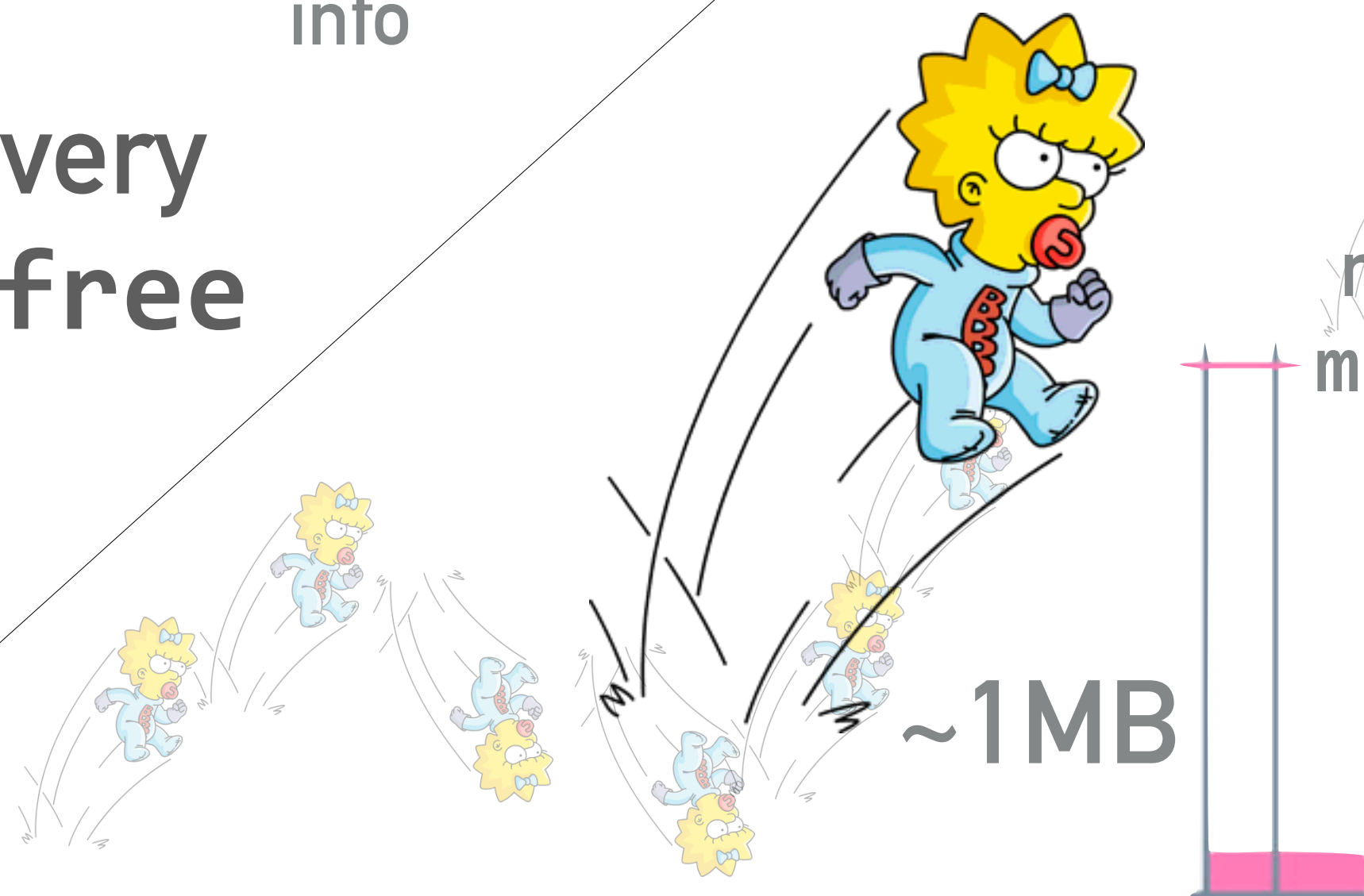




memory\_profiler: ~300x slower

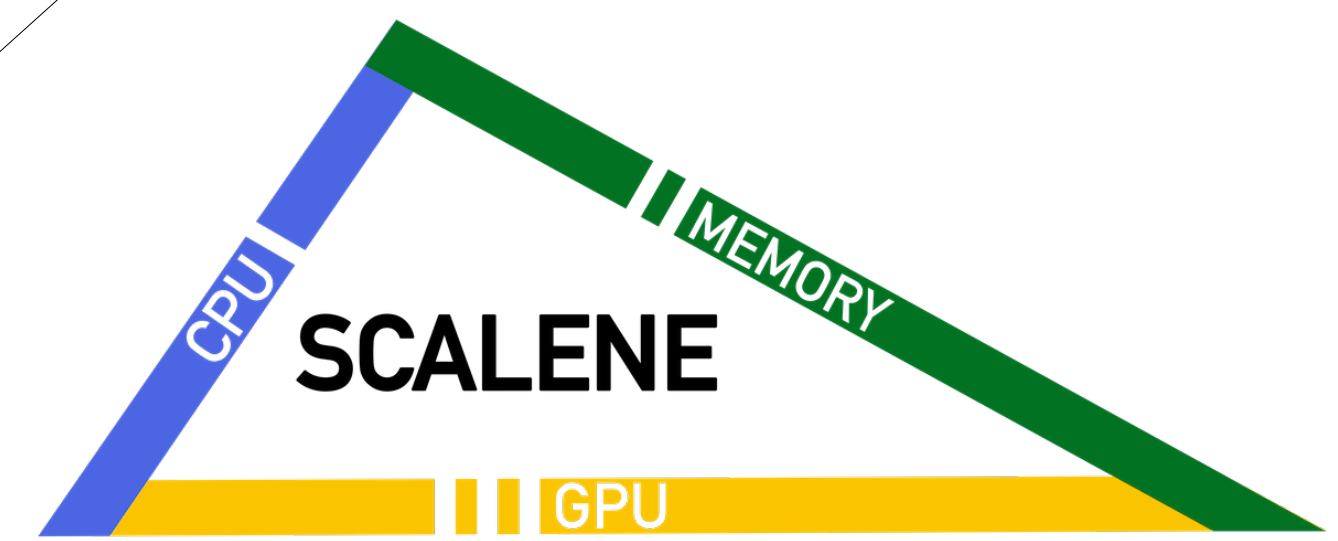
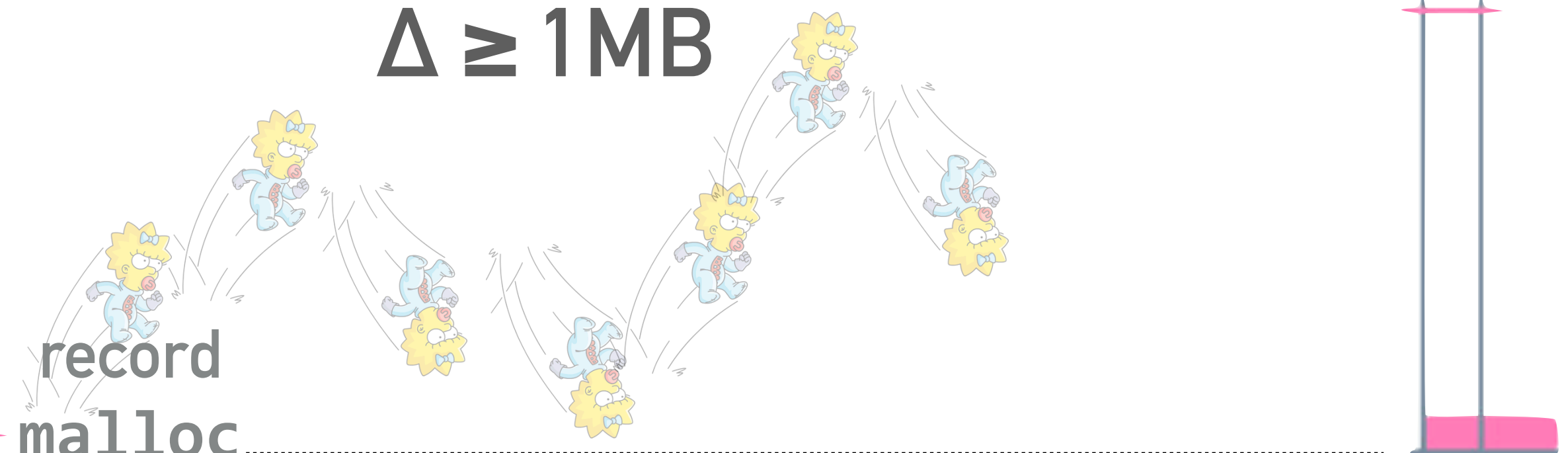


tracks every malloc/free



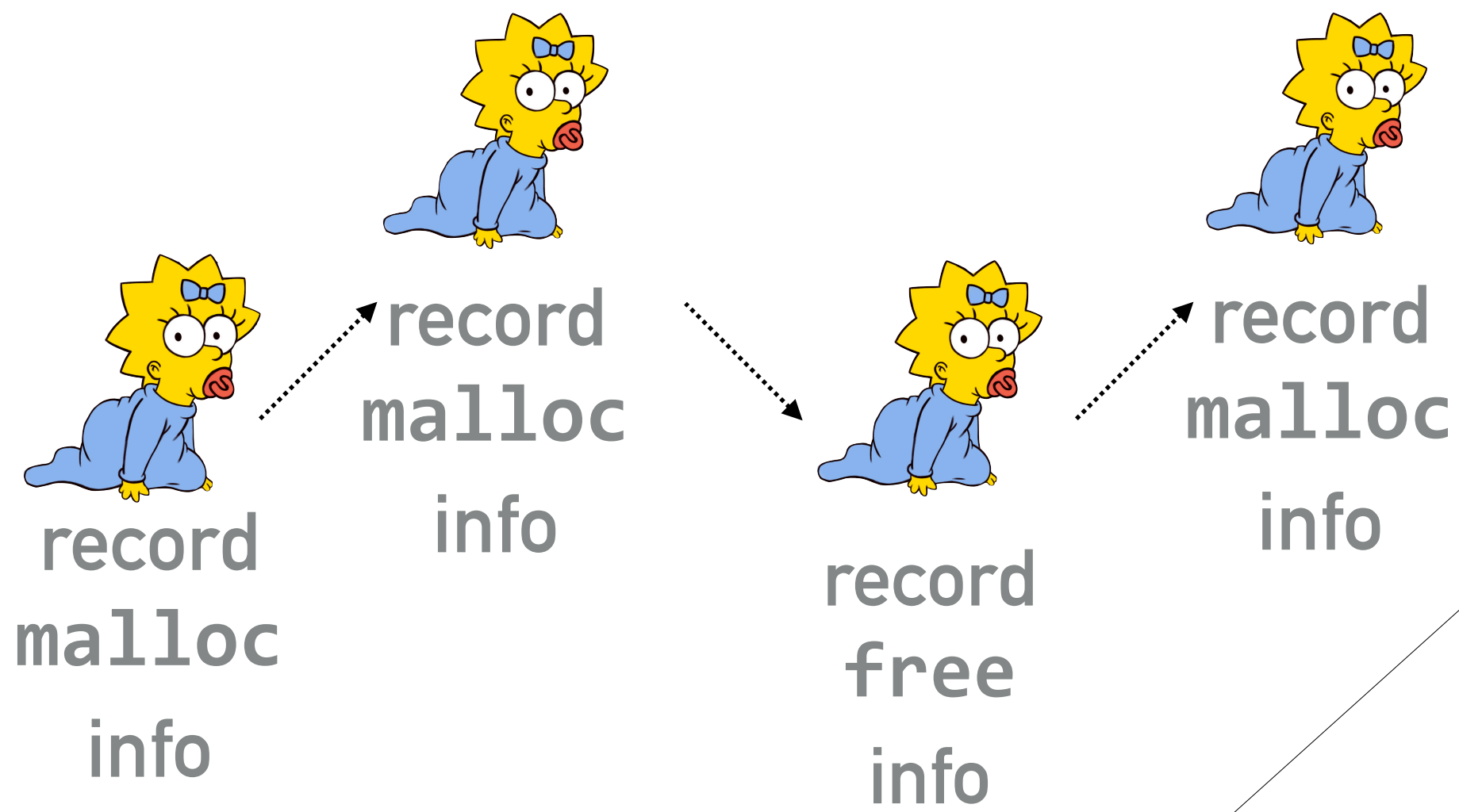
record malloc info

only tracks every  $\Delta \geq 1\text{MB}$

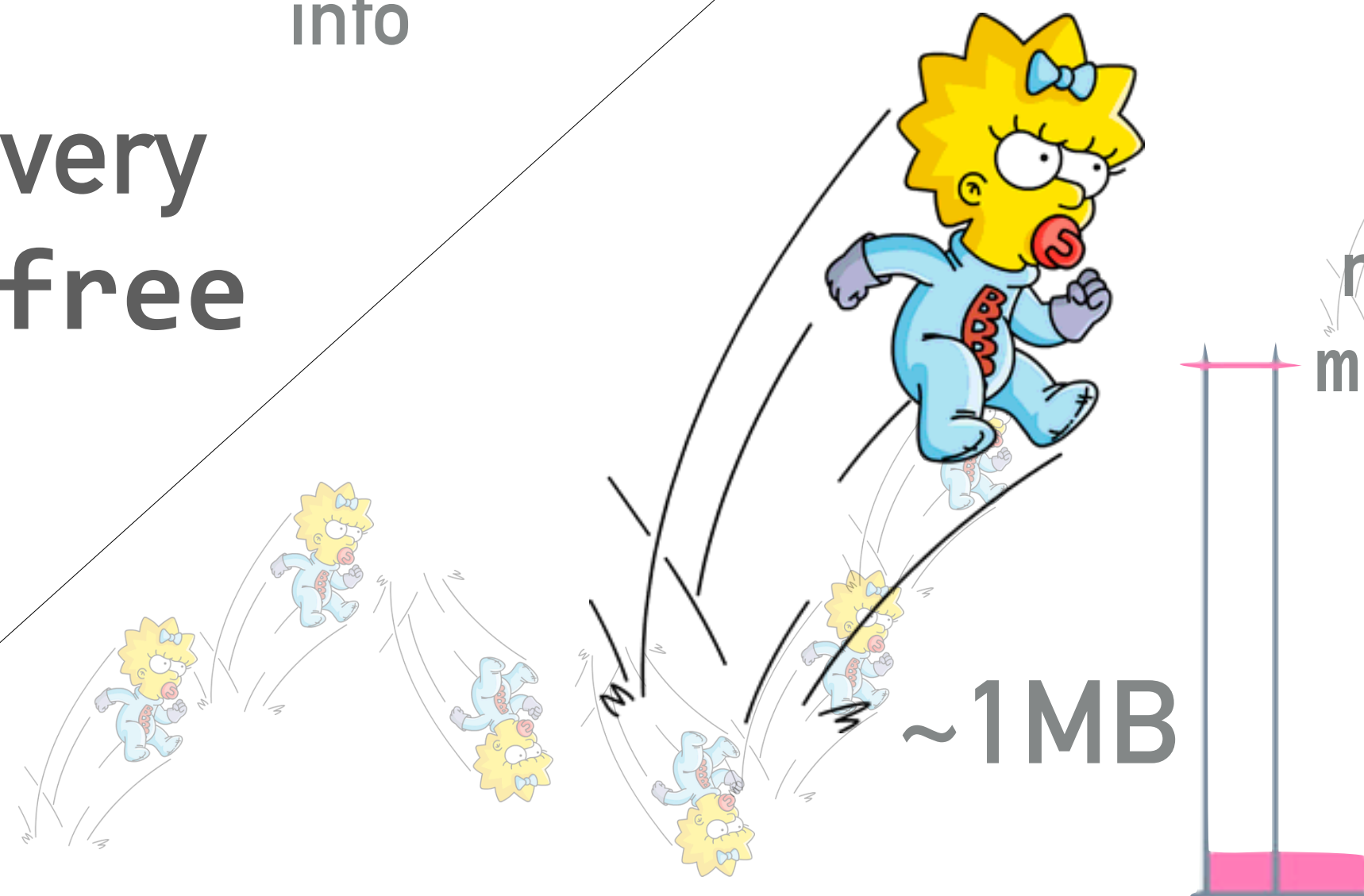


threshold-  
based  
sampling

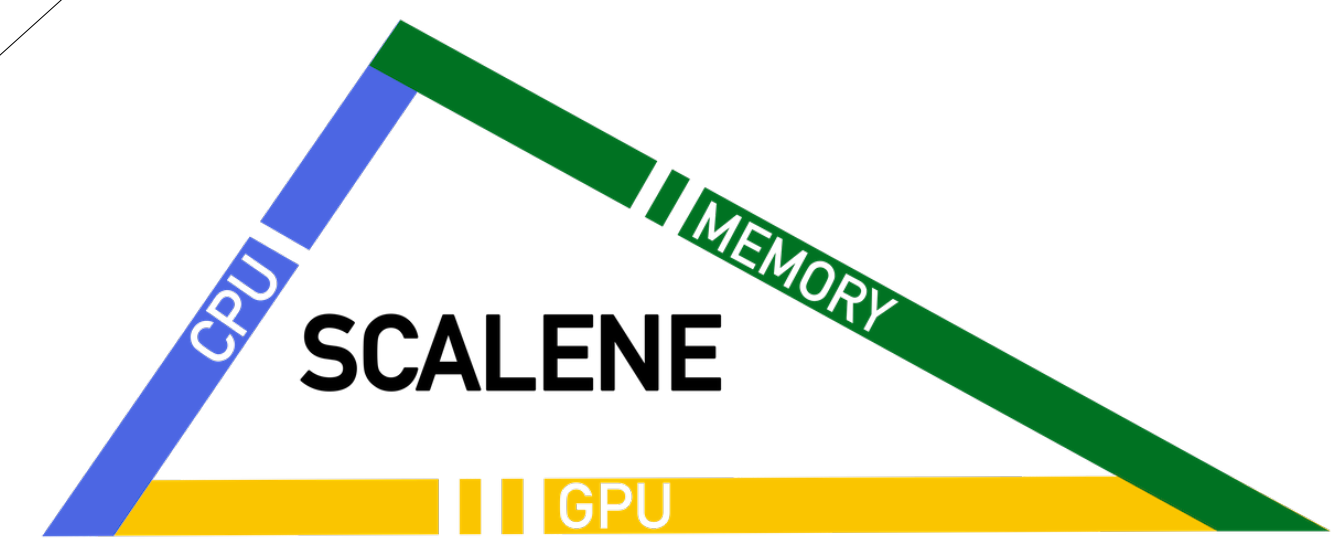
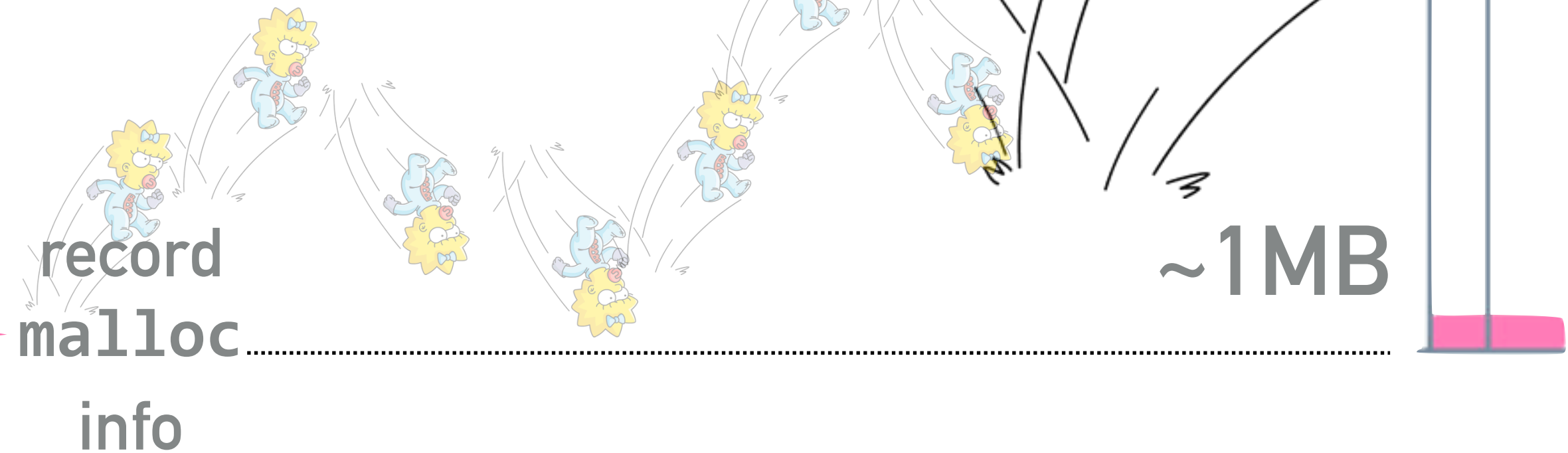
memory\_profiler: ~300x slower



tracks every  
malloc/free



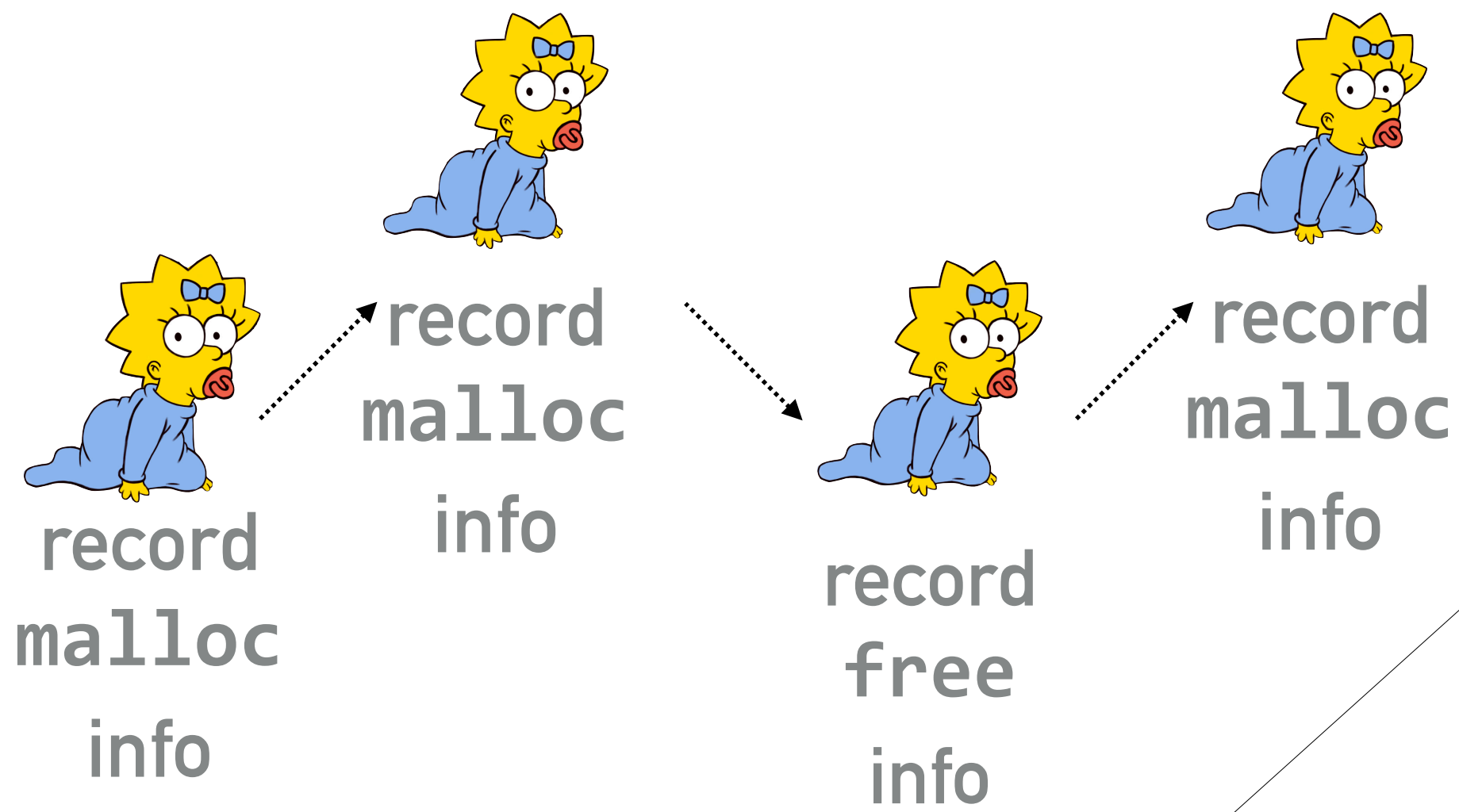
only tracks  
every  
 $\Delta \geq 1\text{MB}$



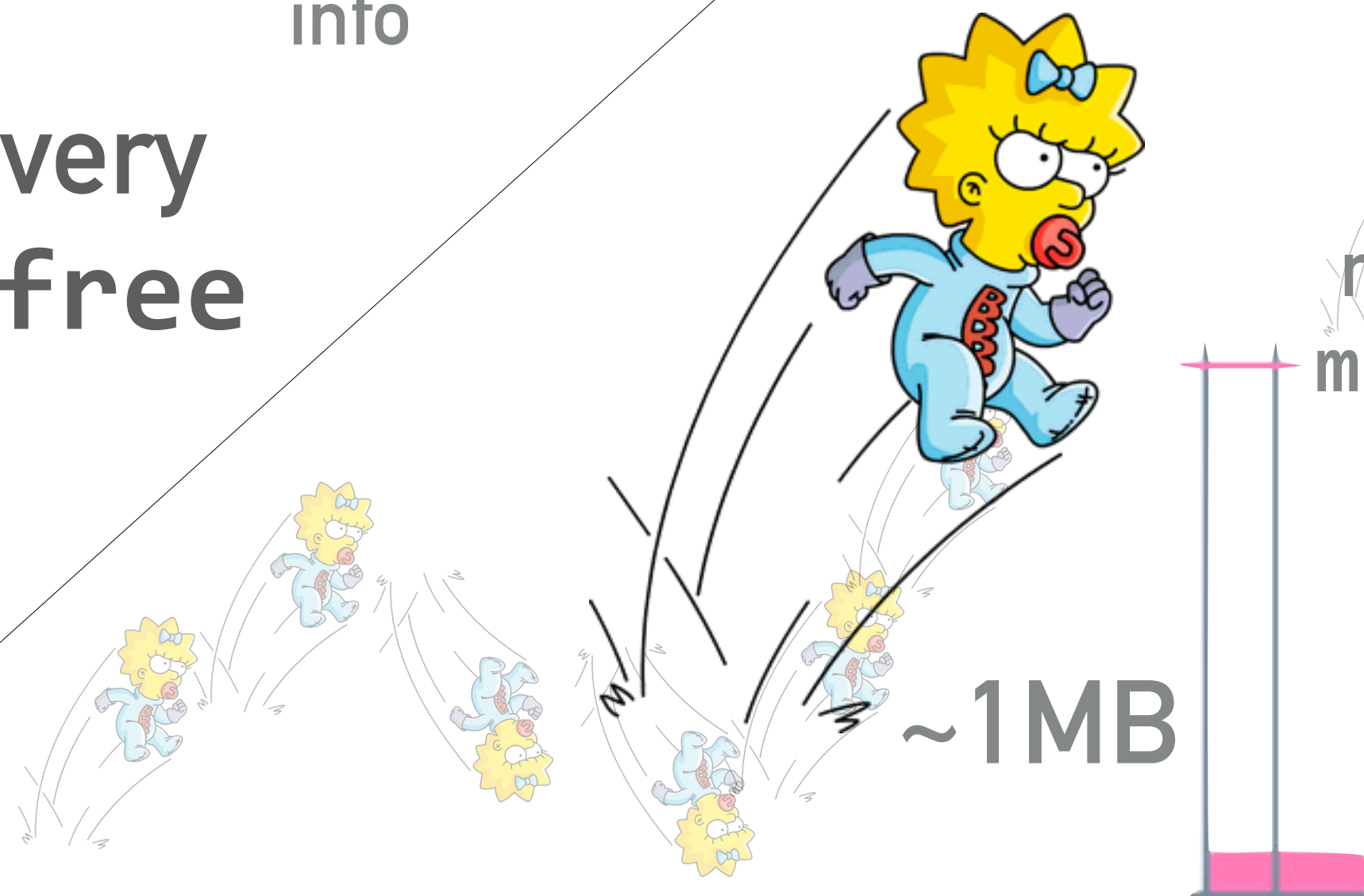
threshold-  
based  
sampling

record  
malloc  
info

memory\_profiler: ~300x slower



tracks every  
malloc/free

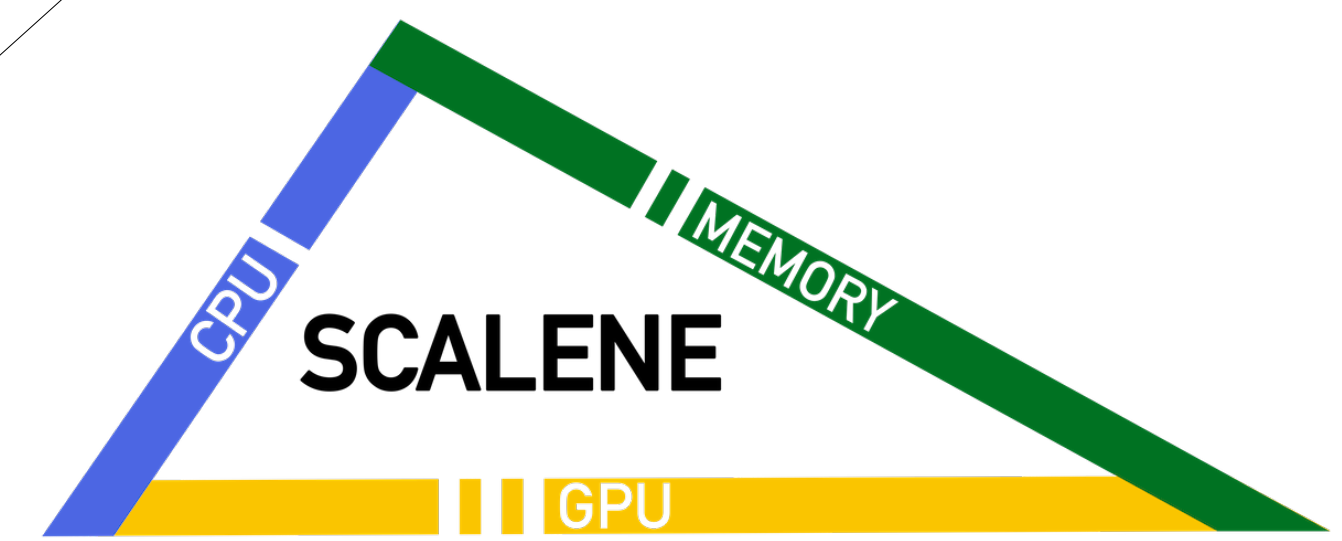


only tracks  
every  
 $\Delta \geq 1\text{MB}$

record  
malloc  
info





~1MB

record  
malloc  
info

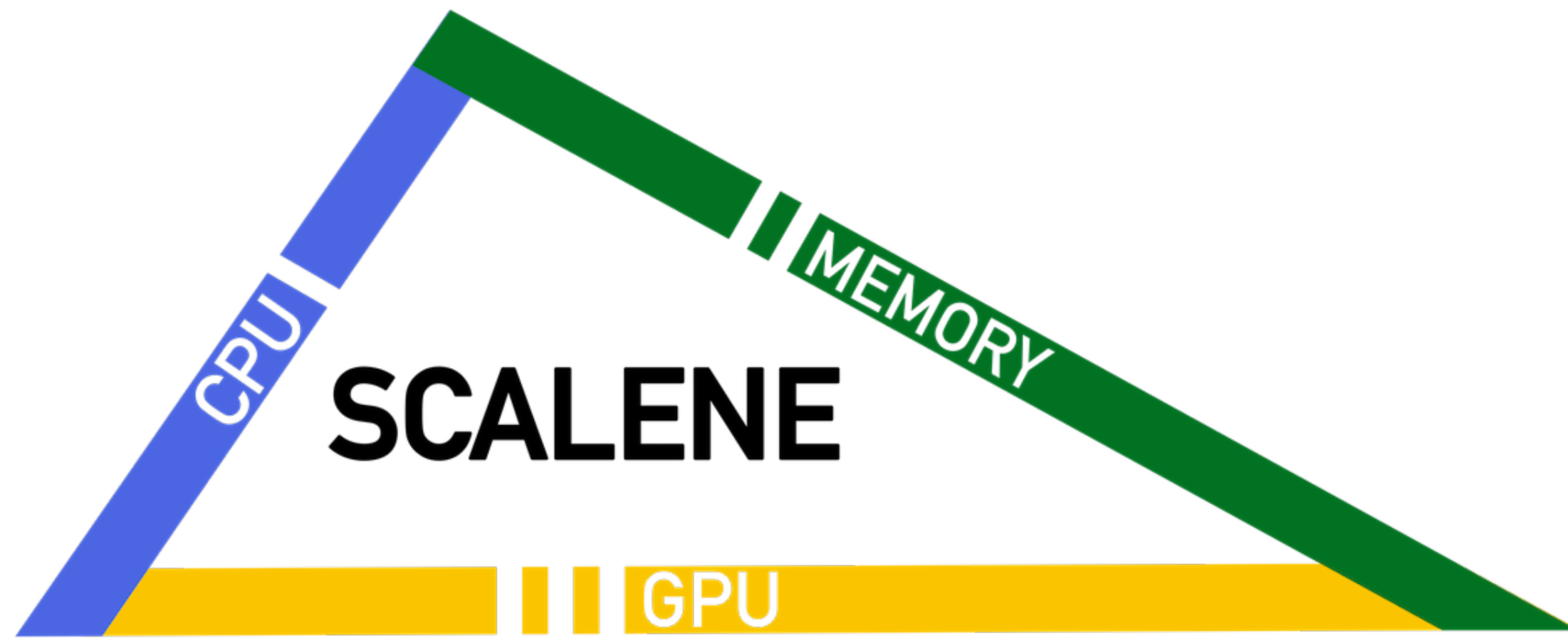


HIGHER ACCURACY  
+ ~no overhead!

# reports leak volume (MB/s) per line

<u>TIME</u>	<u>MEMORY</u> <i>average</i>	<u>MEMORY</u> <i>peak</i>	<u>MEMORY</u> <i>timeline</i>	<u>MEMORY</u> <i>activity</i>	<u>COPY</u> <i>(MB/s)</i>	<u>GPU</u> <i>util.</i>	<u>GPU</u> <i>memory</i>	<u>LINE PROFILE</u> <i>(click to reset order)</i> <i>leaky/test-leaky.py</i>
			 <b>possible leak</b>	100%	430		562 <sup>3</sup>	<code>for i in range(1000000):</code> <code>leak.leak("I want to run\nI want to hide\nI want to tear down the walls")</code>
							562 <sup>4</sup>	

memory: 1596.0MB (@ 3s); possible leak (0.3 MB/s)



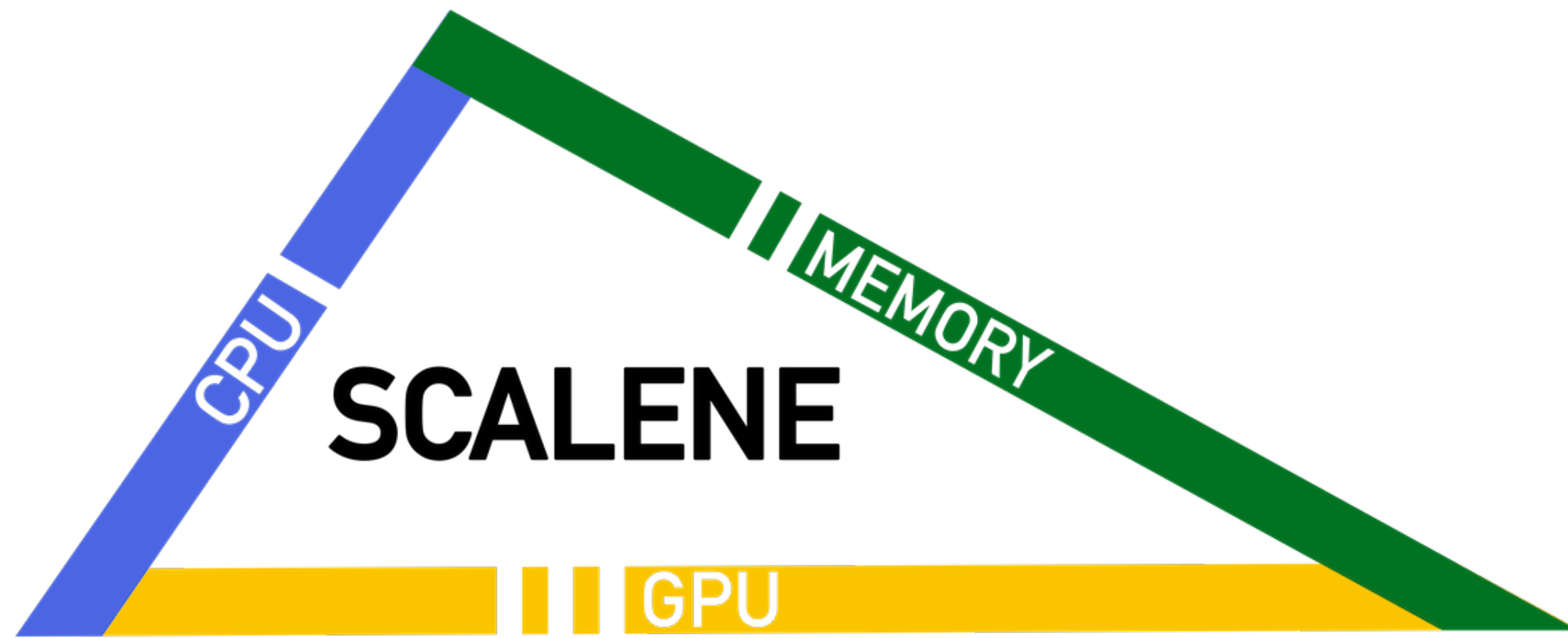
<u>CPU</u>	<u>MEMORY</u>	<u>MEMORY</u>	<u>COPY</u>	<u>GPU</u>
<b>PYTHON</b>	<b>PYTHON</b>	<b>USAGE</b>	<b>VOLUME</b>	<b>UTIL %,</b>
<b>NATIVE</b>	<b>NATIVE</b>	<b>OVER TIME,</b>	<b>(MB/s)</b>	<b>PEAK</b>
<b>SYS%</b>	<b>AVERAGE &amp; PEAK</b>	<b>% OF MEM ALLOCATED</b>		<b>MEMORY</b>

% pip install -U scalene

downloads 715k

downloads/month 27k

[GitHub.com/plasma-umass/scalene](https://github.com/plasma-umass/scalene)



<u>CPU</u> PYTHON NATIVE SYS%	<u>MEMORY</u> PYTHON NATIVE AVERAGE & PEAK	<u>MEMORY</u> USAGE OVER TIME, % OF MEM ALLOCATED	<u>COPY</u> VOLUME (MB/s)	<u>GPU</u> UTIL %, PEAK MEMORY
--	--	---	---------------------------------	---

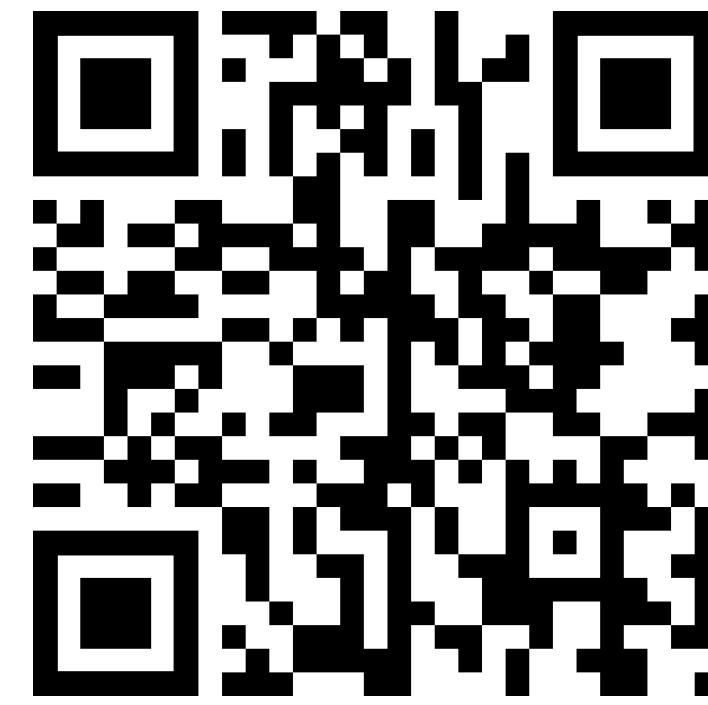
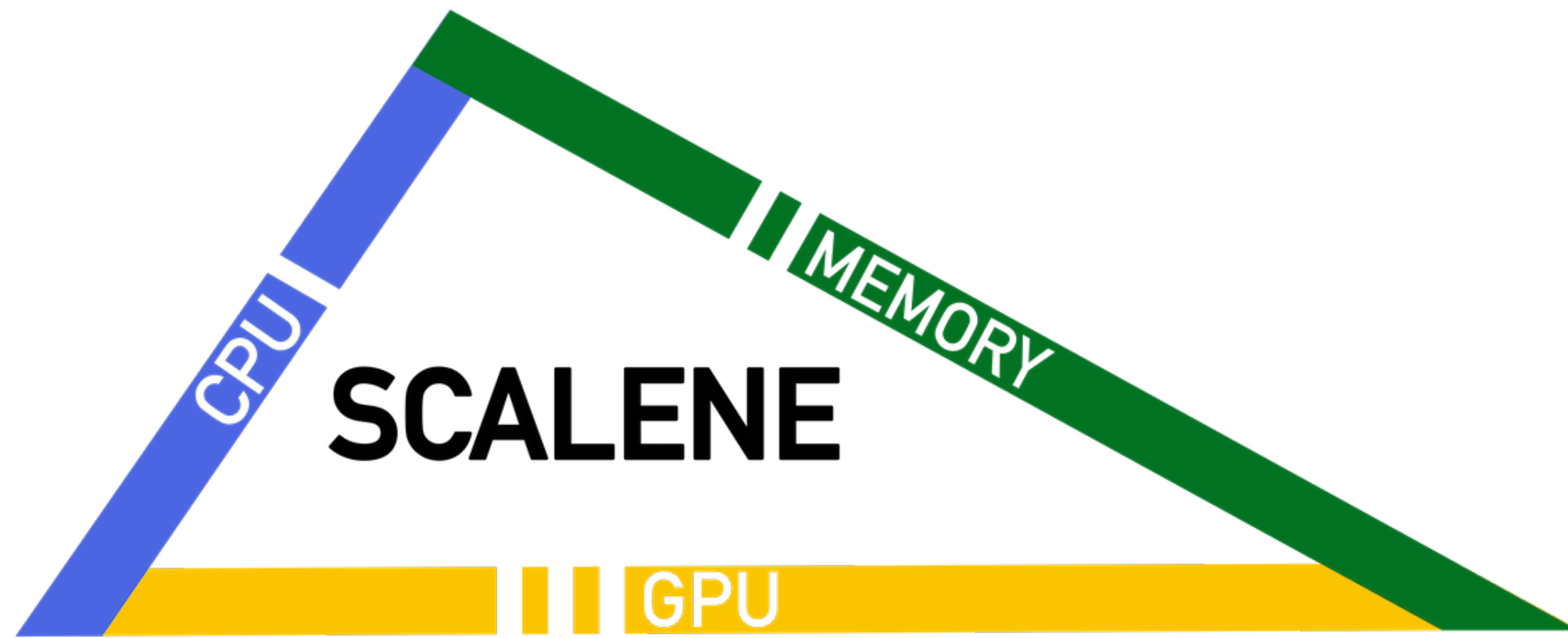
% pip install -U scalene

downloads 715k

downloads/month 27k

[GitHub.com/plasma-umass/scalene](https://github.com/plasma-umass/scalene)





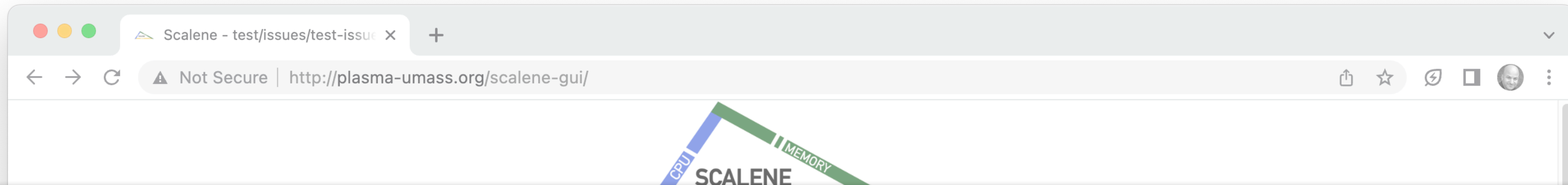
<b>CPU</b>	<b>MEMORY</b>	<b>MEMORY</b>	<b>COPY</b>	<b>GPU</b>
<b>PYTHON</b>	<b>PYTHON</b>	<b>USAGE</b>	<b>VOLUME</b>	<b>UTIL %,</b>
<b>NATIVE</b>	<b>NATIVE</b>	<b>OVER TIME,</b>	<b>(MB/s)</b>	<b>PEAK</b>
<b>SYS%</b>	<b>AVERAGE &amp;</b>	<b>% OF MEM</b>		<b>MEMORY</b>
	<b>PEAK</b>	<b>ALLOCATED</b>		

% pip install -U scalene

downloads 715k    downloads/month 27k

[GitHub.com/plasma-umass/scalene](https://github.com/plasma-umass/scalene)



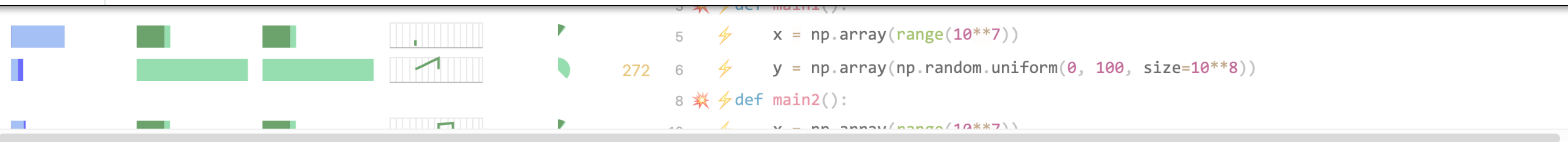


cmwilhelm commented 2 days ago • edited



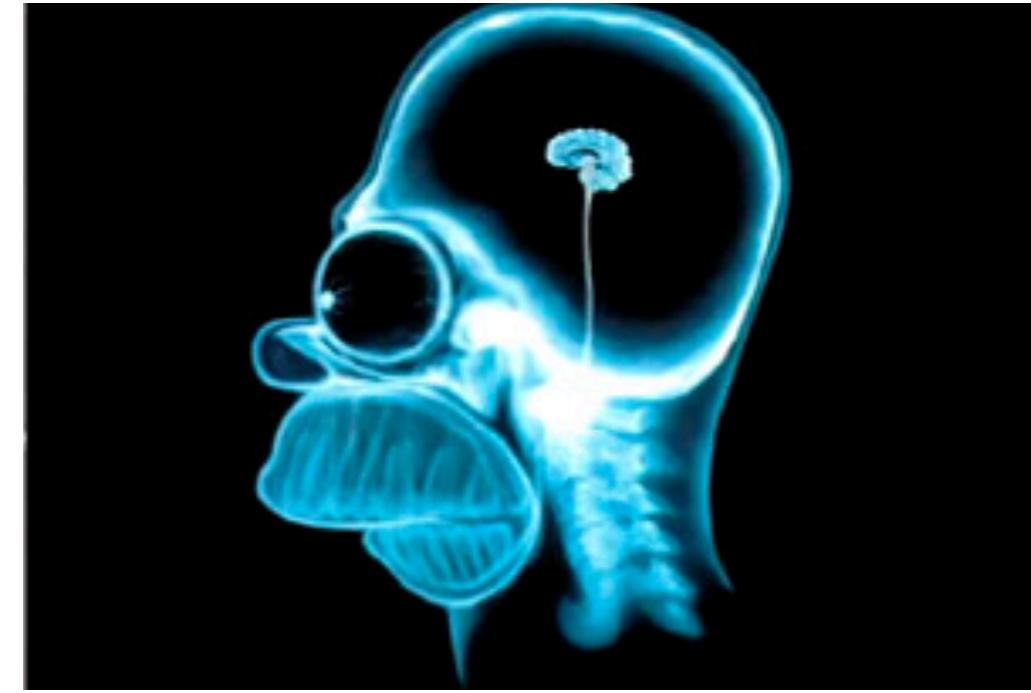
We've started using scalene over at Semantic Scholar ([www.semanticscholar.org](http://www.semanticscholar.org)) as part of our toolsuite for operationalizing machine learning models. Recently we found a model of ours was cost prohibitive and put an entire product direction in jeopardy. We generated a set of test data and ran our models with Scalene mounted -- the html output was able to pin point our squeakiest wheels and help us validate our changes were having an impact. The process was iterative, precise and repeatable. In the end, we were able to reduce costs by a staggering 92%.

With these models, there is also always the question of whether things would be more cost effective running inference services on GPUs rather than CPU. Scalene allowed us to quickly ascertain what fraction of our runtime would benefit from the hardware acceleration, and what CPU-bound code we'd need to pare down to achieve our goals.



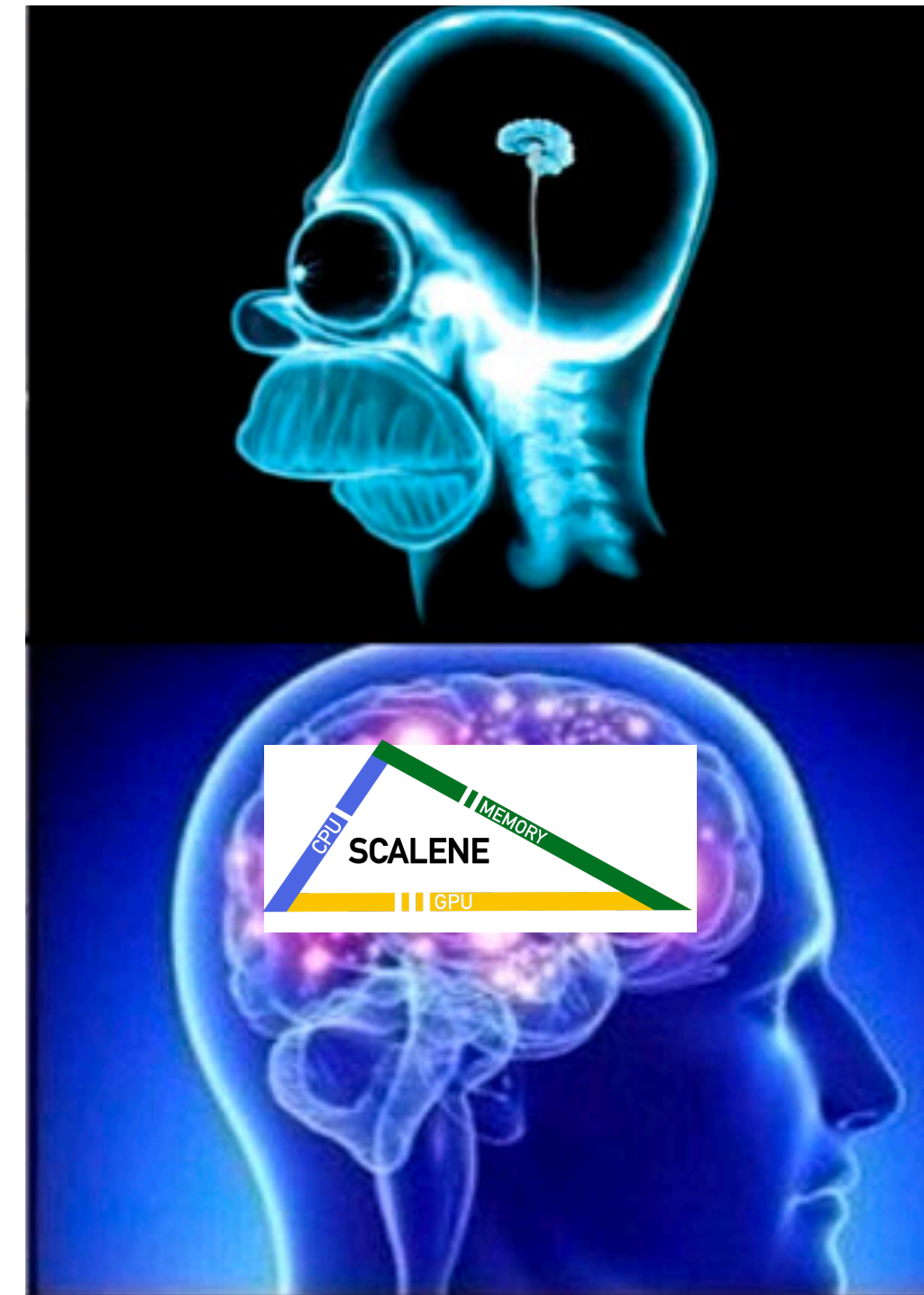


# writing your code in Python



writing your code in  
Python

profiling your Python  
code with Scalene



writing your code in  
Python


profiling your Python  
code with Scalene

getting Scalene to  
optimize your code!



Scalene - test/issues/test-issue X

Not Secure | http://plasma-umass.org/scalene-gui/



Select a profile (.json)

▼ advanced options

**Proposed optimizations**

Enter an [OpenAI key](#) to enable: sk- [REDACTED] ✓

Optimize runtime performance


Optimize memory efficiency

Include GPU optimizations

Click on an explosion (💣) to see proposed optimizations for a region of code, or on a lightning bolt (⚡) to propose optimizations for a specific line. Click again to generate a different one.

*Note that optimizations are AI-generated and may not be correct.*

**Time:** Python | native | system **Memory:** Python | native **Memory timeline:** (max: 1.653 GB, growth: 21.0%)




*hover over bars to see breakdowns; click on COLUMN HEADERS to sort.*

show all | hide all | only display profiled lines

Scalene - test/issues/test-issue X

Not Secure | http://plasma-umass.org/scalene-gui/



Select a profile (.json)

▼ advanced options

**Proposed optimizations**


Enter an [OpenAI key](#) to enable: sk- [redacted] ✓

- Optimize runtime performance
- Optimize memory efficiency
- Include GPU optimizations

Click on an explosion (💣) to see proposed optimizations for a region of code, or on a lightning bolt (⚡) to propose optimizations for a specific line. Click again to generate a different one.

*Note that optimizations are AI-generated and may not be correct.*

**Time:** Python | native | system **Memory:** Python | native **Memory timeline:** (max: 1.653 GB, growth: 21.0%)



*hover over bars to see breakdowns; click on COLUMN HEADERS to sort.*

show all | hide all | only display profiled lines

Scalene - test/issues/test-issue31.py

Not Secure | http://plasma-umass.org/scalene-gui/

# SCALENE

CPU MEMORY GPU

Select a profile (.json)

▶ advanced options

Time: Python | native | system    Memory: Python | native    Memory timeline: (max: 1.653 GB, growth: 21.0%)

*hover over bars to see breakdowns; click on COLUMN HEADERS to sort.*

show all | hide all | only display profiled lines

▼ test/issues/test-issue31.py: % of time = 100.0% (2.897s) out of 2.897s.

<u>TIME</u>	<u>MEMORY average</u>	<u>MEMORY peak</u>	<u>MEMORY timeline</u>	<u>MEMORY activity</u>	<u>COPY</u>	<u>LINE PROFILE</u> (click to reset order)
					17	1 ⚡ import numpy as np
						⚡ def main1():
						3 # Proposed optimization: # Vectorize the code to reduce the number of loops and improve performance. x = np.arange(10**7) y = np.random.uniform(0, 100, size=10**8)

## 90x speedup

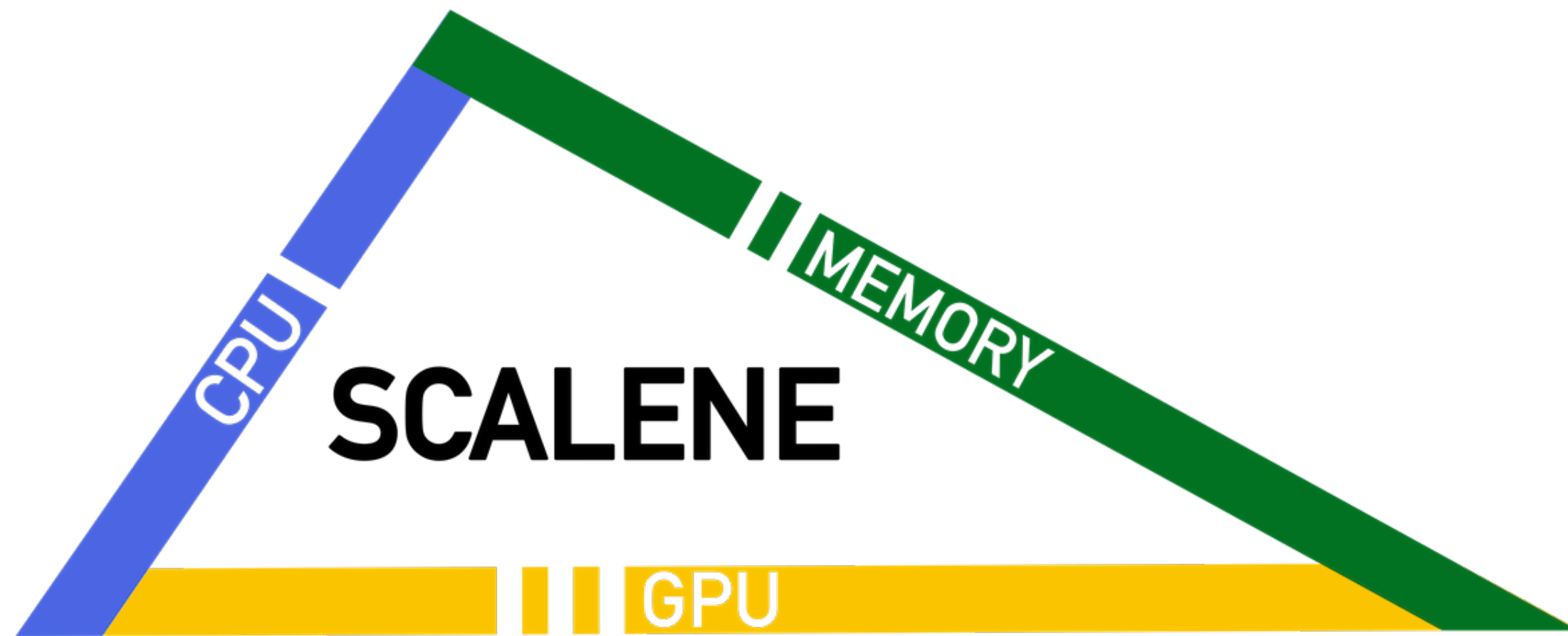
[#58 \(comment\)](#) presents the following code:

```
for i in range(n_features):
    for n in range(n_samples):
        subgrad[i] += (- y[n] * X[n][i]) if y[n] * (np.dot(X[n], w) + b) < 1 else 0
    subgrad[i] += self.lambda1 * (-1 if w[i] < 0 else 1) + 2 * self.lambda2 * w[i]
```

Scalene proposes the following optimization:

```
# Vectorized operations to replace for loops
subgrad[:-1] = np.sum(-y[:, None] * X * (y * (X.dot(w) + b) < 1)[: , None], axis=0)
subgrad[:-1] += self.lambda1 * np.sign(w) + 2 * self.lambda2 * w
subgrad[-1] = np.sum(-y * (y * (X.dot(w) + b) < 1))
```

Scalene's proposed optimization accelerates the original code by at least 90x (89 seconds to 1 second, when running 500 iterations), and takes full advantage of multiple cores.



<b>CPU</b>	<b>MEMORY</b>	<b>MEMORY</b>	<b>COPY</b>	<b>GPU</b>
<b>PYTHON</b>	<b>PYTHON</b>	<b>USAGE</b>	<b>VOLUME</b>	<b>UTIL %,</b>
<b>NATIVE</b>	<b>NATIVE</b>	<b>OVER TIME,</b>	<b>(MB/s)</b>	<b>PEAK</b>
<b>SYS%</b>	<b>AVERAGE &amp;</b>	<b>% OF MEM</b>		<b>MEMORY</b>
	<b>PEAK</b>	<b>ALLOCATED</b>		

% pip install -U scalene

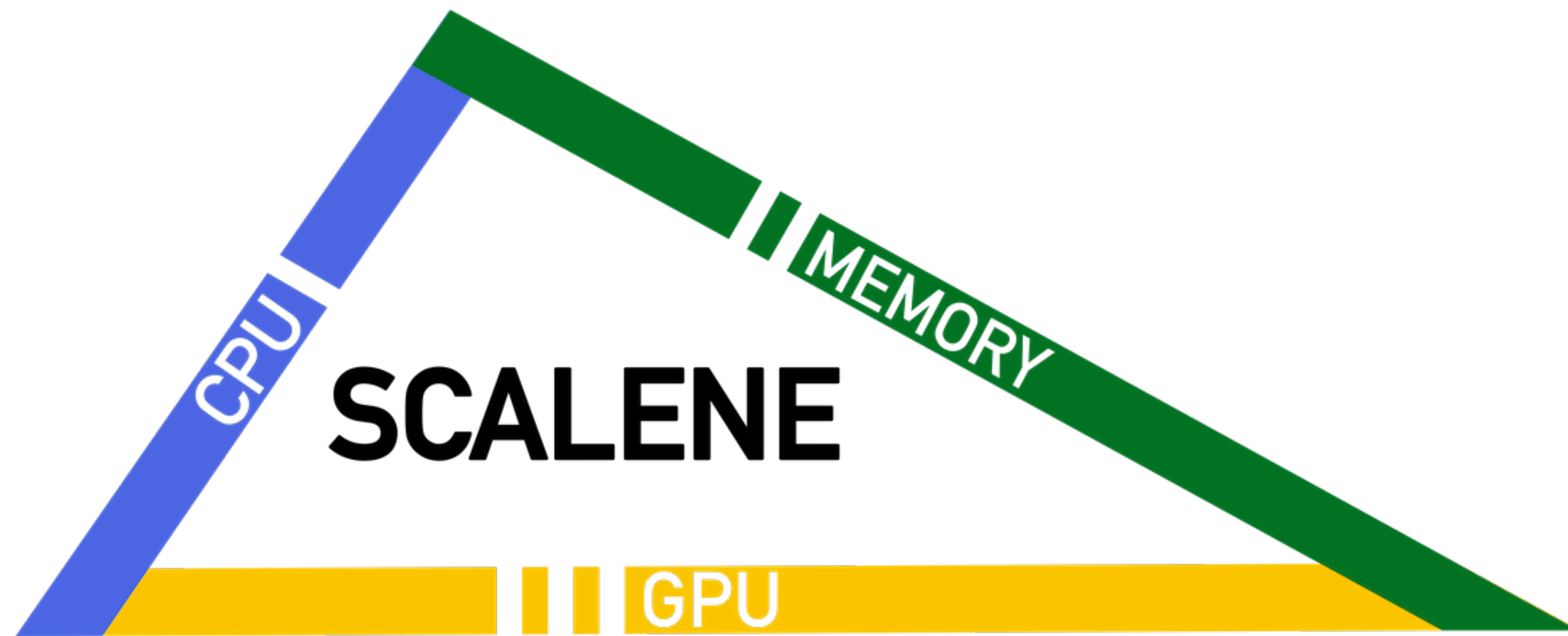
downloads 715k

downloads/month 27k

[GitHub.com/plasma-umass/scalene](https://github.com/plasma-umass/scalene)







**AI powered optimizations!**

<u>CPU</u>	<u>MEMORY</u>	<u>MEMORY</u>	<u>COPY</u>	<u>GPU</u>
<b>PYTHON</b>	<b>PYTHON</b>	<b>USAGE</b>	<b>VOLUME</b>	<b>UTIL %,</b>
<b>NATIVE</b>	<b>NATIVE</b>	<b>OVER TIME,</b>	<b>(MB/s)</b>	<b>PEAK</b>
<b>SYS%</b>	<b>AVERAGE &amp; PEAK</b>	<b>% OF MEM ALLOCATED</b>		<b>MEMORY</b>

% pip install -U scalene

downloads 715k    downloads/month 27k

[GitHub.com/plasma-umass/scalene](https://github.com/plasma-umass/scalene)

