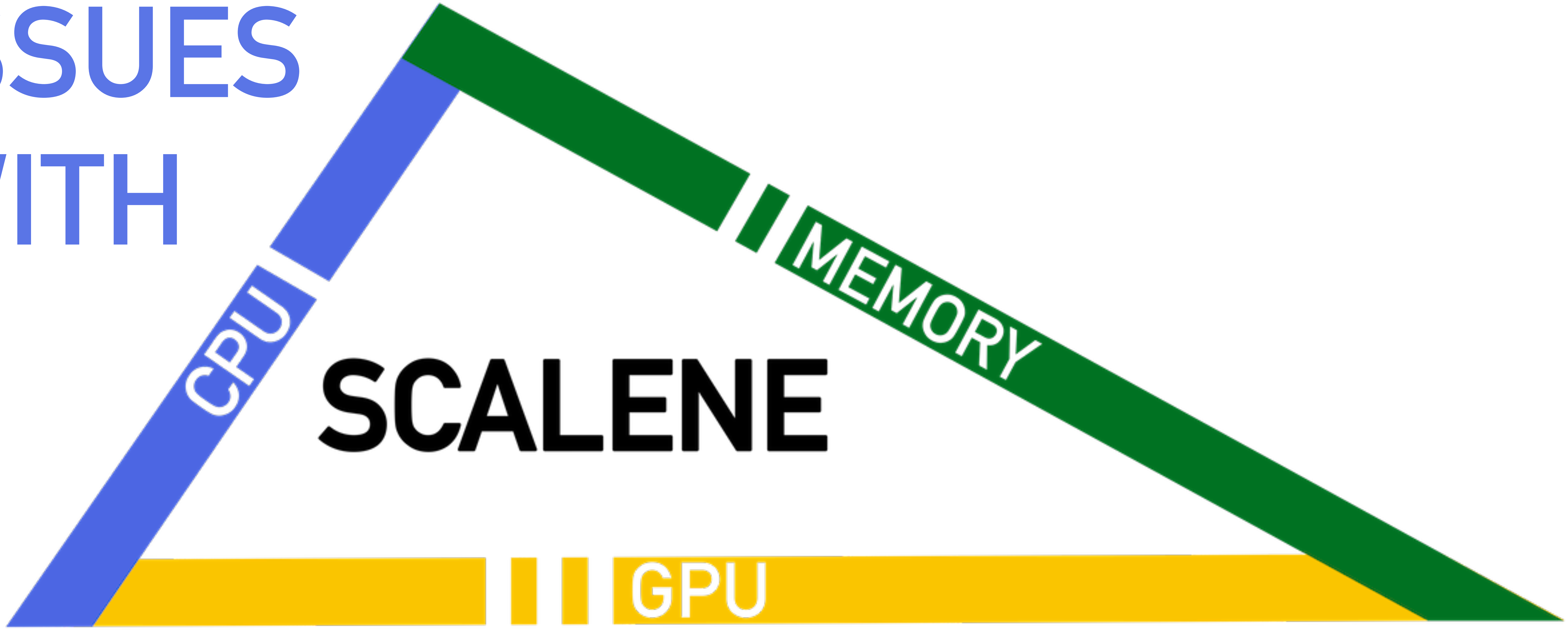
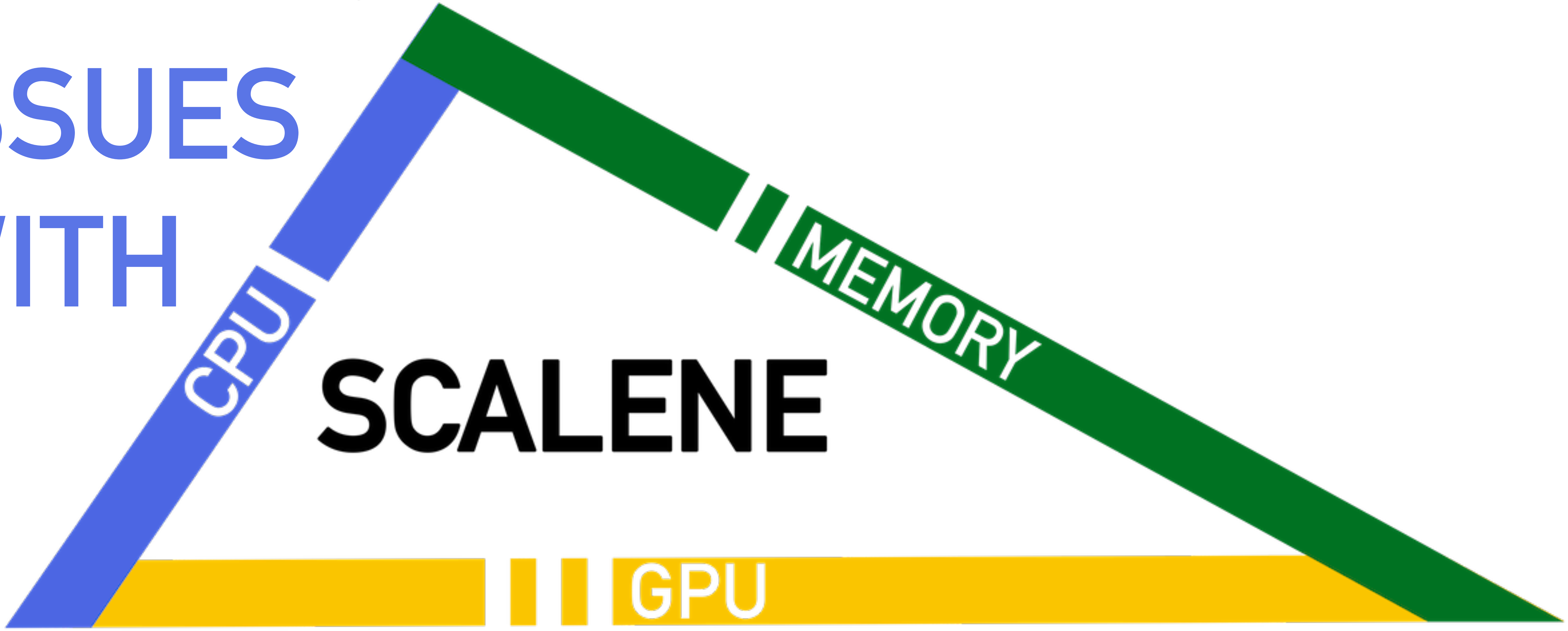
TRIANGULATING PYTHON PERFORMANCE ISSUES WITH



EMERY BERGER, SAM STERN, JUAN ALTMAYER PIZZORNO

UNIVERSITY OF MASSACHUSETTS AMHERST

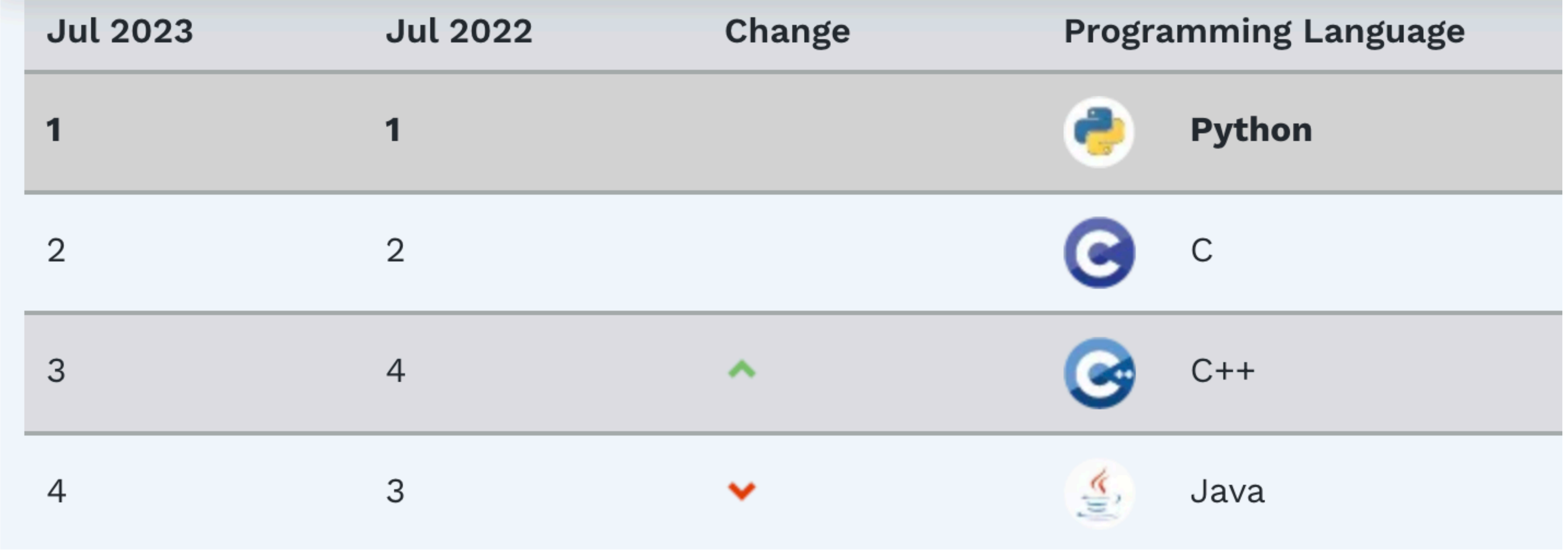
TRIANGULATING PYTHON PERFORMANCE ISSUES WITH



SCALENE

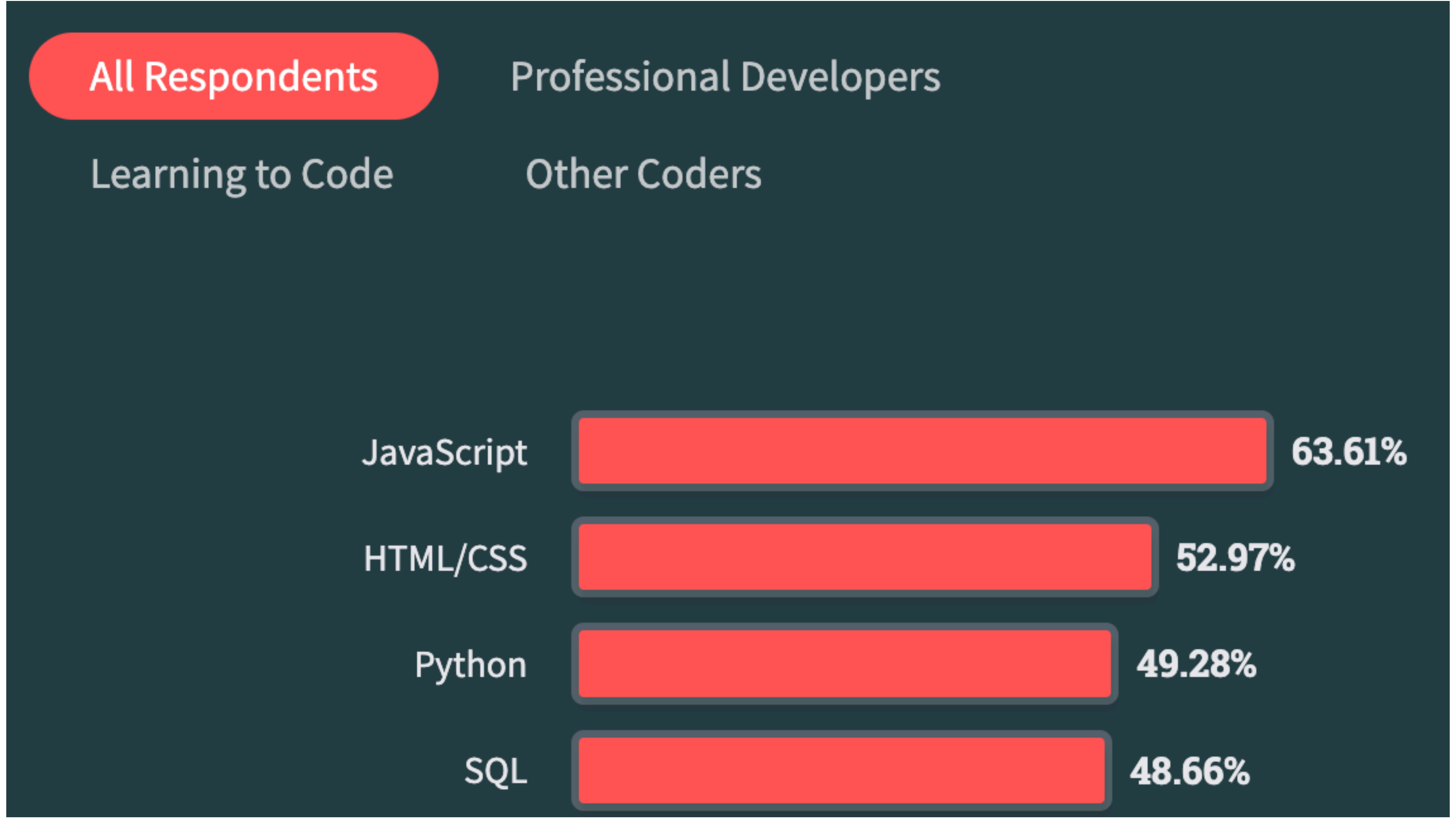
EMERY BERGER, SAM STERN,
JUAN ALTMAYER PIZZORNO
UNIVERSITY OF MASSACHUSETTS AMHERST

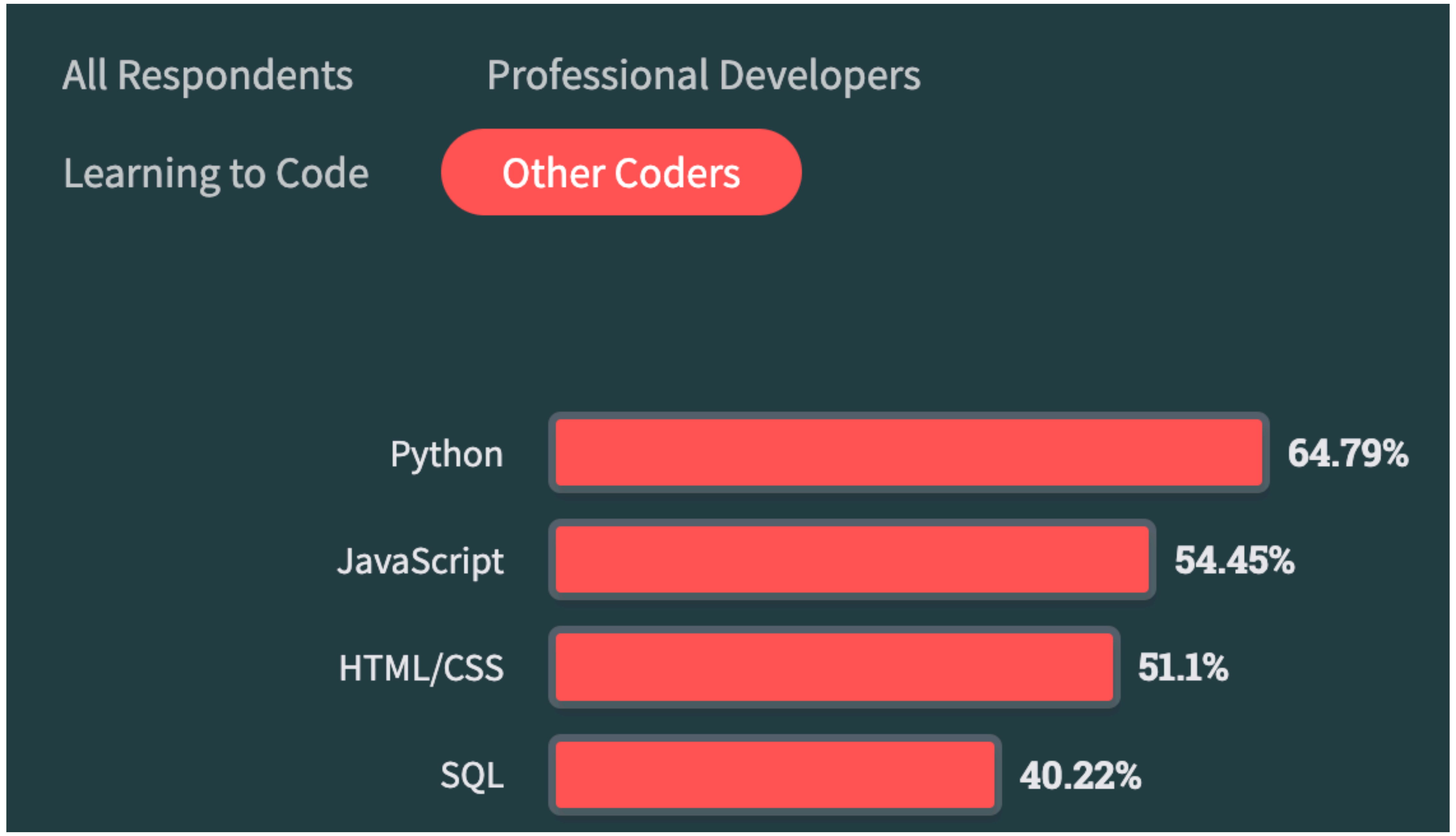
TIOBE
the software quality company



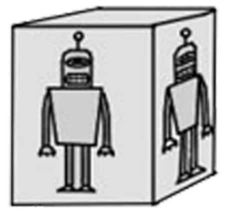
Jul 2023	Jul 2022	Change	Programming Language
1	**1**		**Python**
2	2		C
3	4	▲	C++
4	3	▼	Java

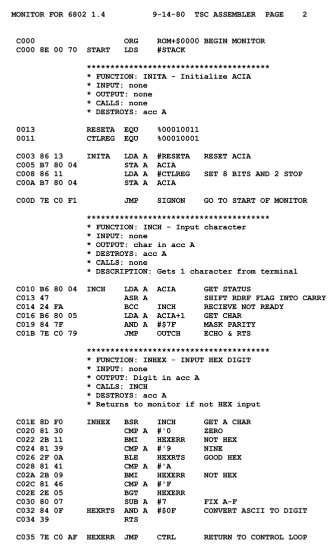
2023 Developer Survey





"Metal" Languages –1980s

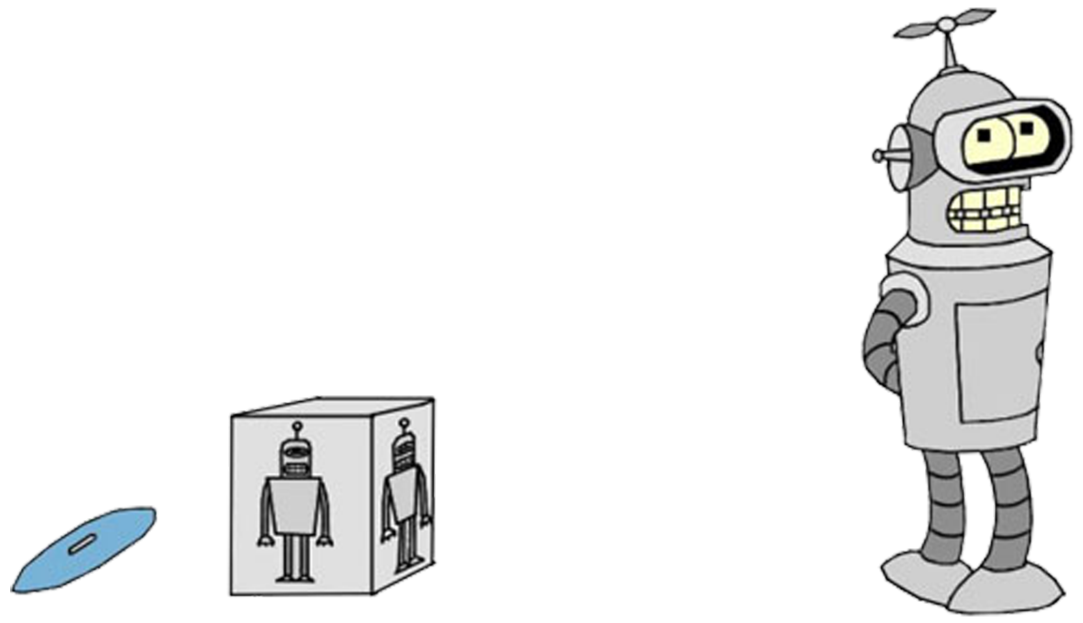


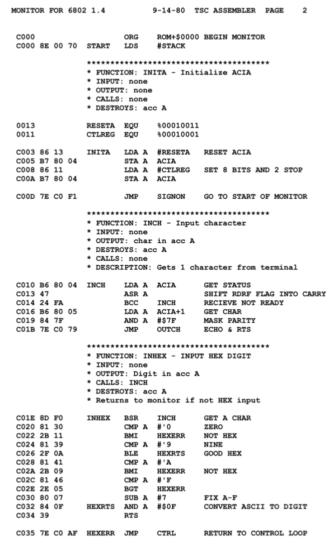


```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                ***************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ***************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ***************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

1949

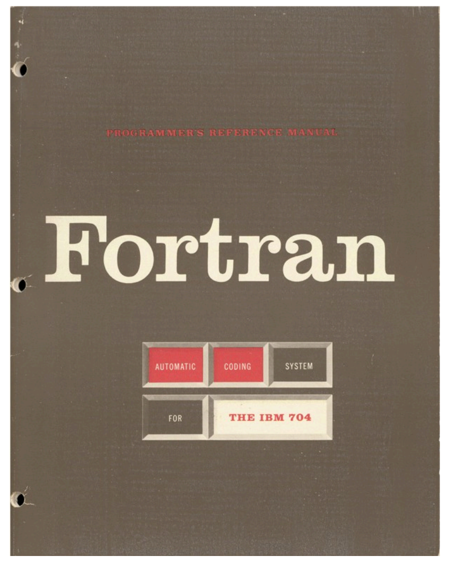
"Metal" Languages –1980s





```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                  ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START  LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

1949



PROGRAMMER'S REFERENCE MANUAL

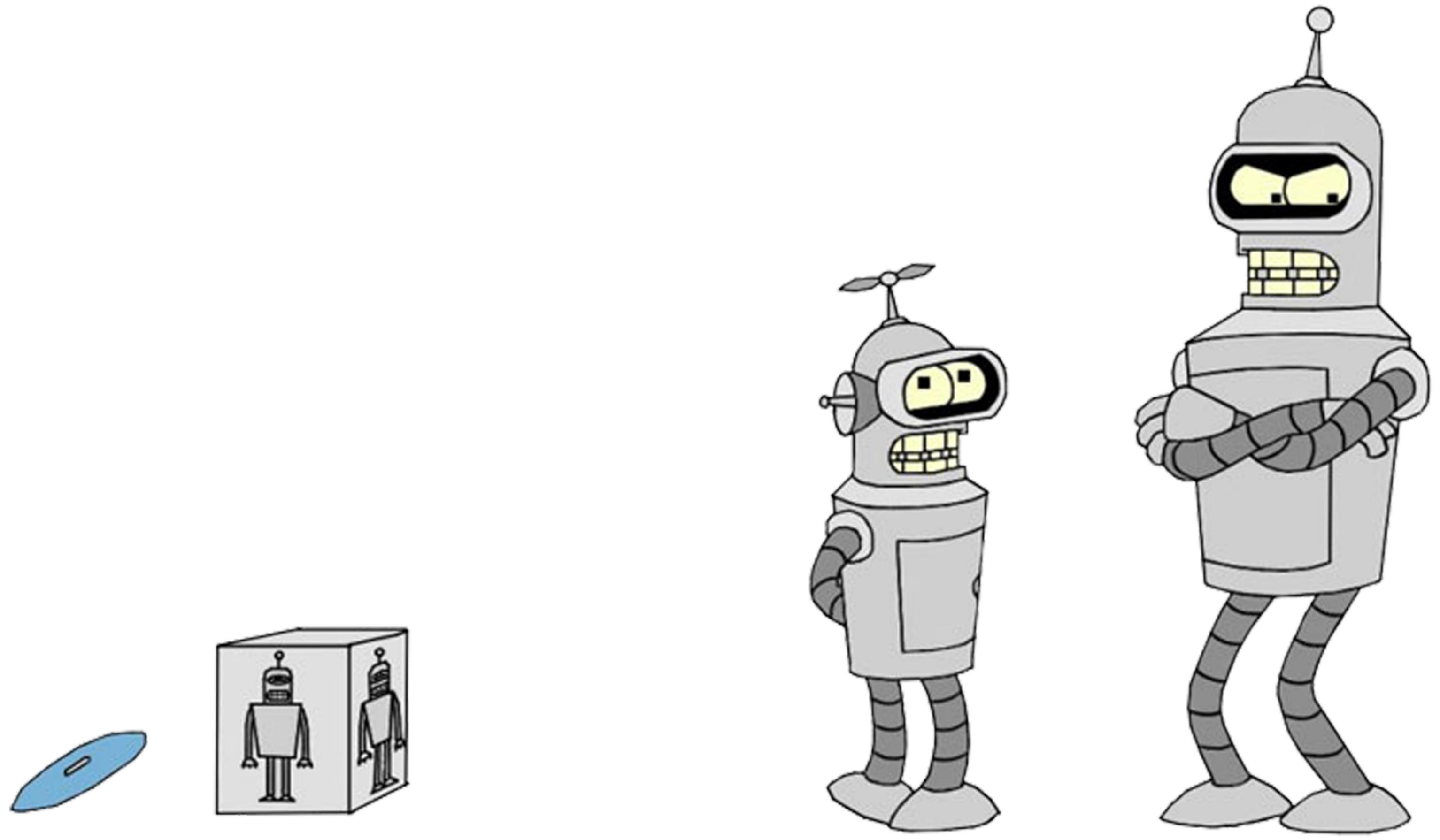
Fortran

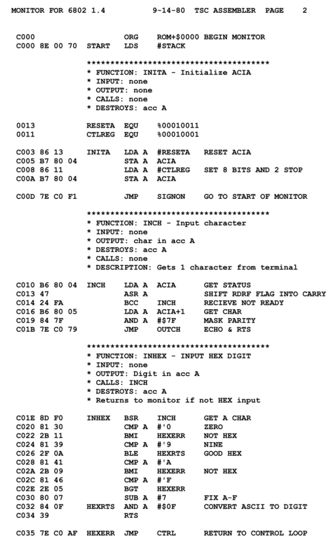
AUTOMATIC CODING SYSTEM

FOR THE IBM 704

1957

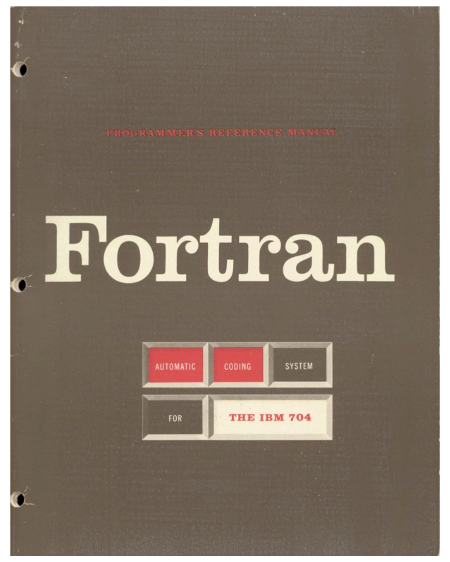
"Metal" Languages –1980s



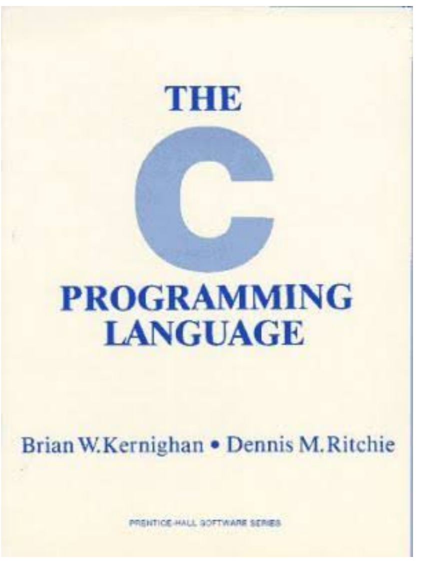


```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                ***************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ***************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ***************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

1949



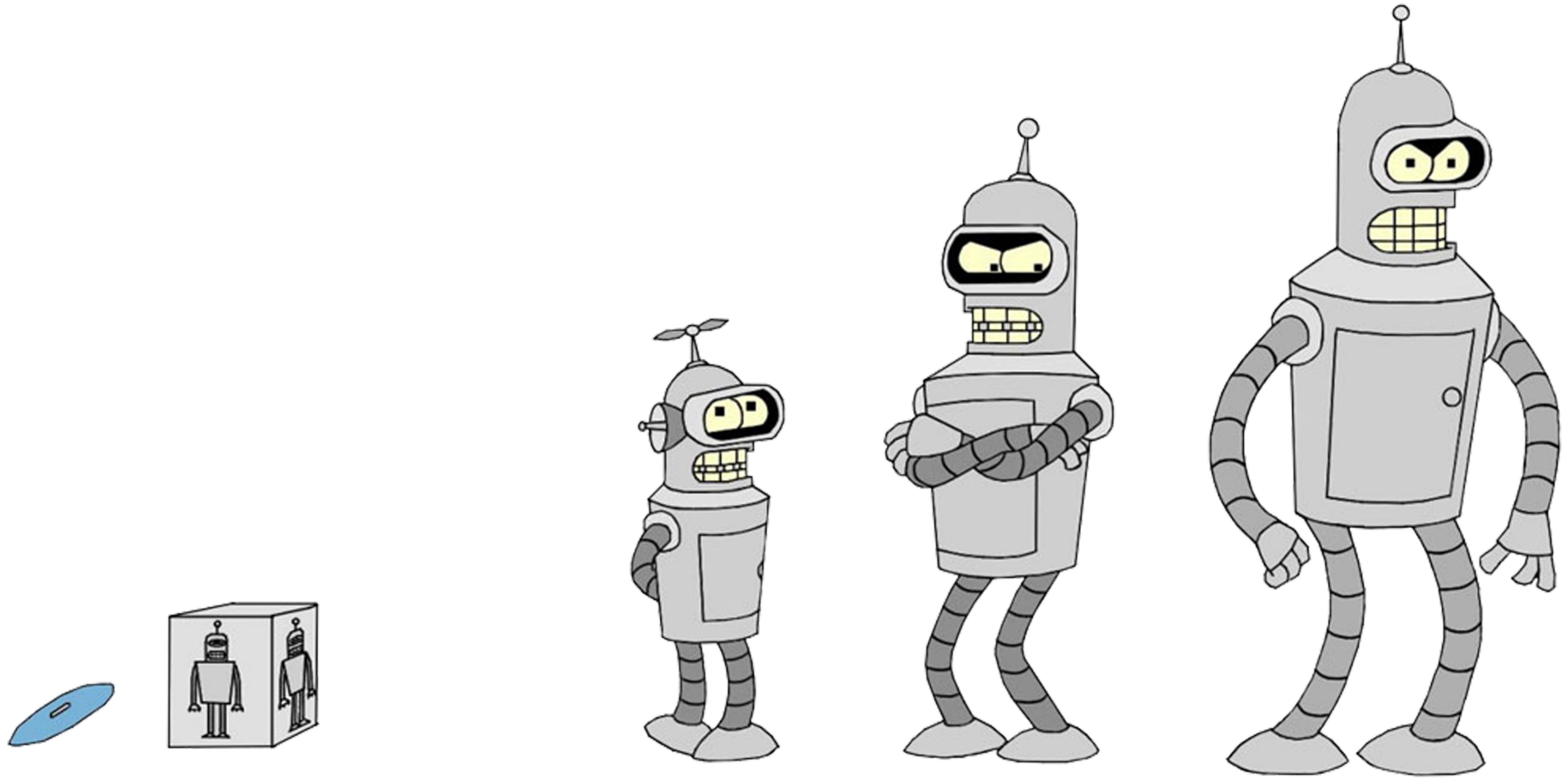
1957

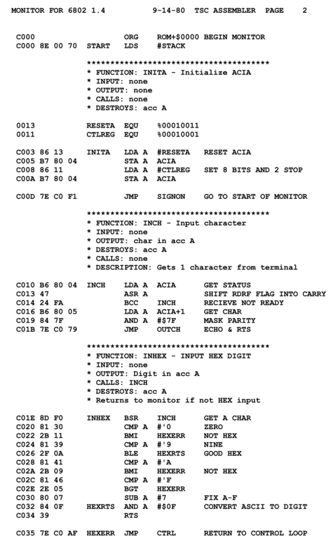


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES

1972

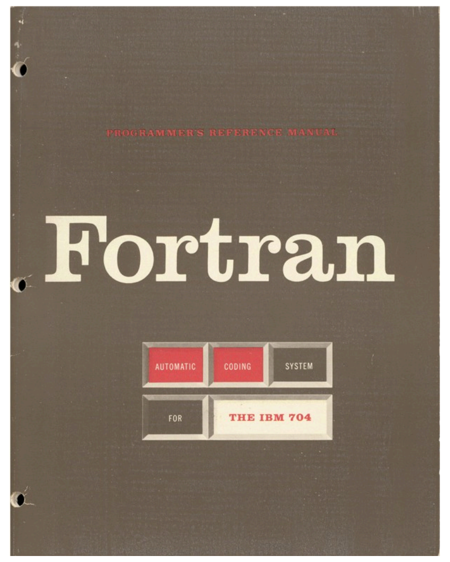
"Metal" Languages –1980s





```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                  ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START  LDS    #STACK

                ***************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA  LDA A  #RESETA   RESET ACIA
 C005 B7 80 04         STA A  ACIA
 C008 86 11            LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04         STA A  ACIA

 C00D 7E C0 F1         JMP    SIGNON    GO TO START OF MONITOR

                ***************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH   LDA A  ACIA      GET STATUS
 C013 47               ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA            BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05         LDA A  ACIA+1    GET CHAR
 C019 84 7F            AND A  #$7F      MASK PARITY
 C01B 7E C0 79         JMP    OUTCH     ECHO & RTS

                ***************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX  BSR    INCH      GET A CHAR
 C020 81 30            CMP A  #'0       ZERO
 C022 2B 11            BMI    HEXERR    NOT HEX
 C024 81 39            CMP A  #'9       NINE
 C026 2F 0A            BLE    HEXRTS    GOOD HEX
 C028 81 41            CMP A  #'A
 C02A 2B 09            BMI    HEXERR    NOT HEX
 C02C 81 46            CMP A  #'F
 C02E 2E 05            BGT    HEXERR
 C030 80 07            SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39               RTS

 C035 7E C0 AF  HEXERR JMP    CTRL      RETURN TO CONTROL LOOP
```

1949

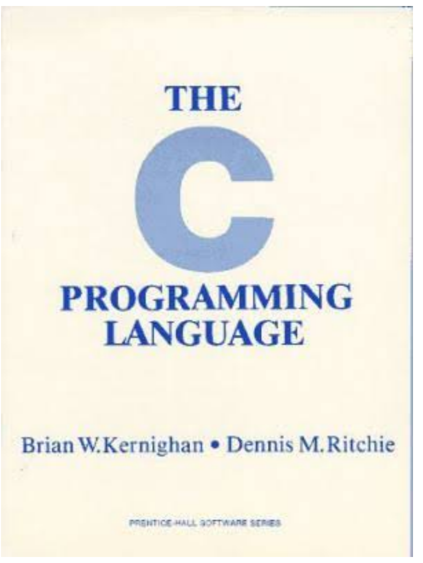


PROGRAMMER'S REFERENCE MANUAL

Fortran

AUTOMATIC CODING SYSTEM FOR THE IBM 704

1957

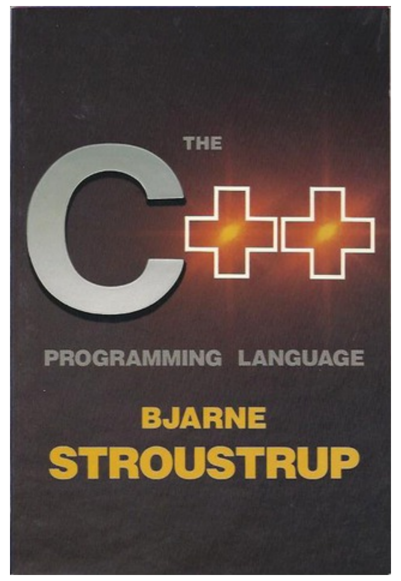


THE C PROGRAMMING LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE-HALL SOFTWARE SERIES

1972



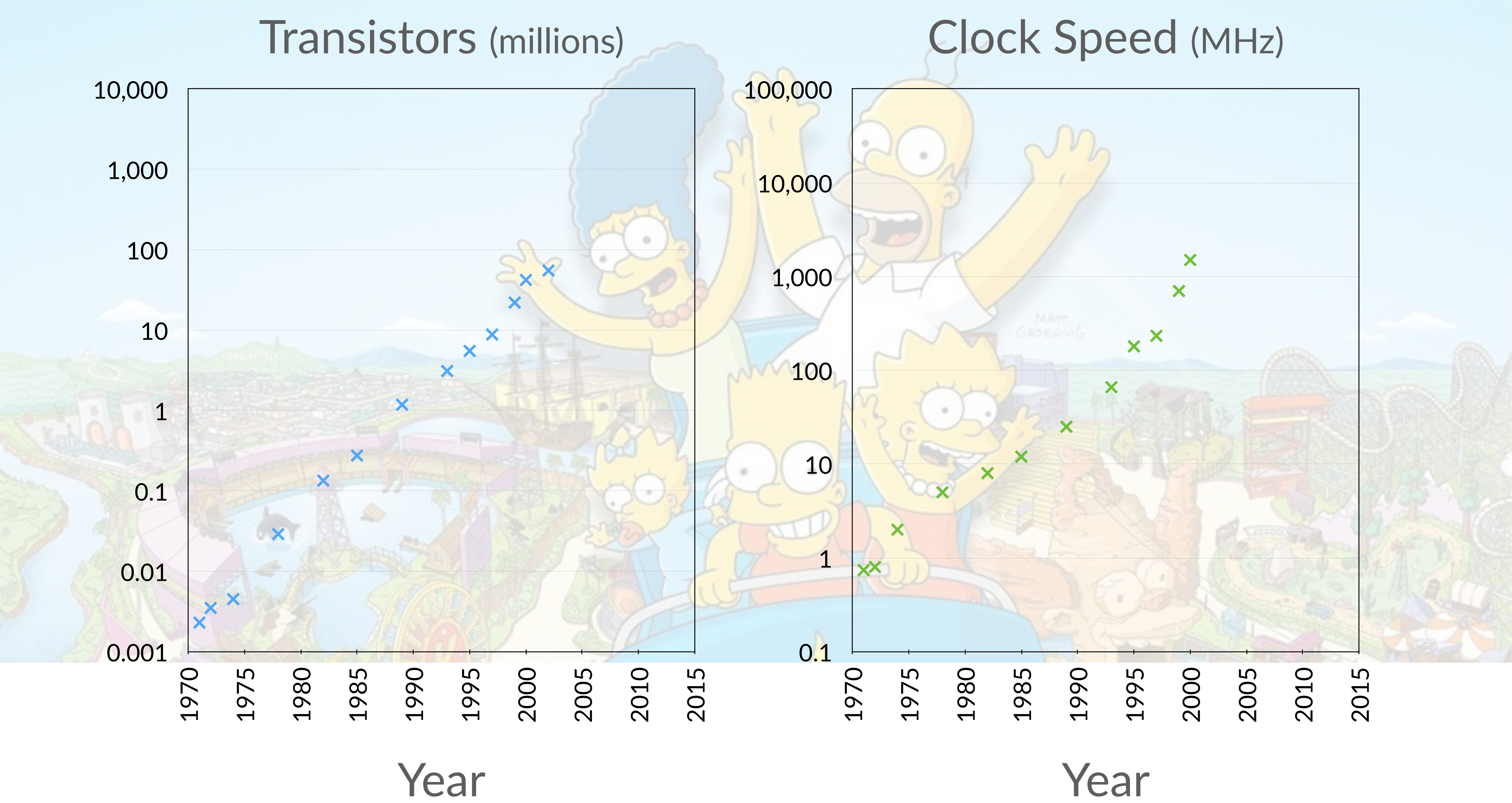
THE C++ PROGRAMMING LANGUAGE

BJARNE STROUSTRUP

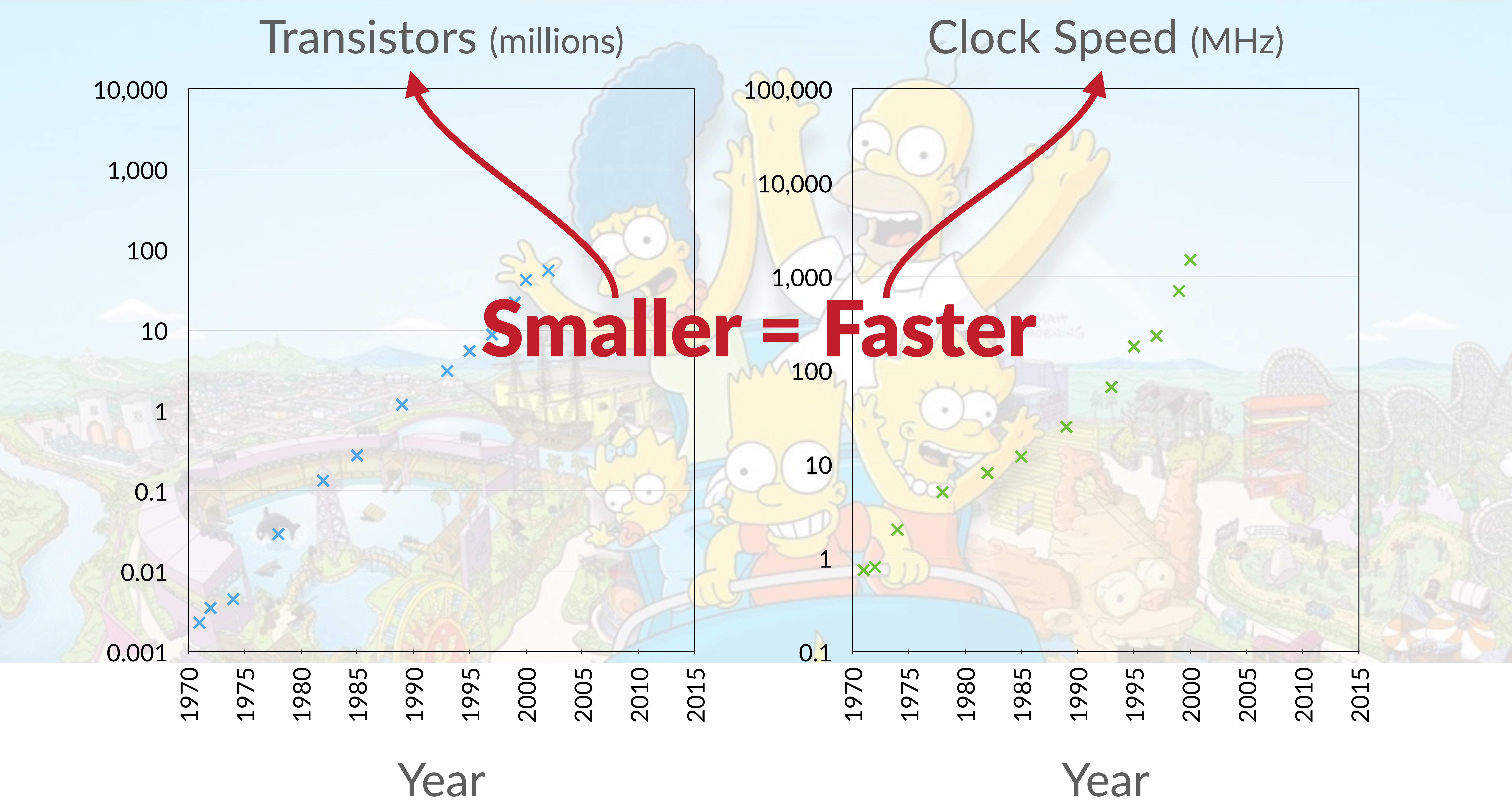
1985

Wild Performance Ride (1980s–2010)!

Wild Performance Ride (1980s–2010)!

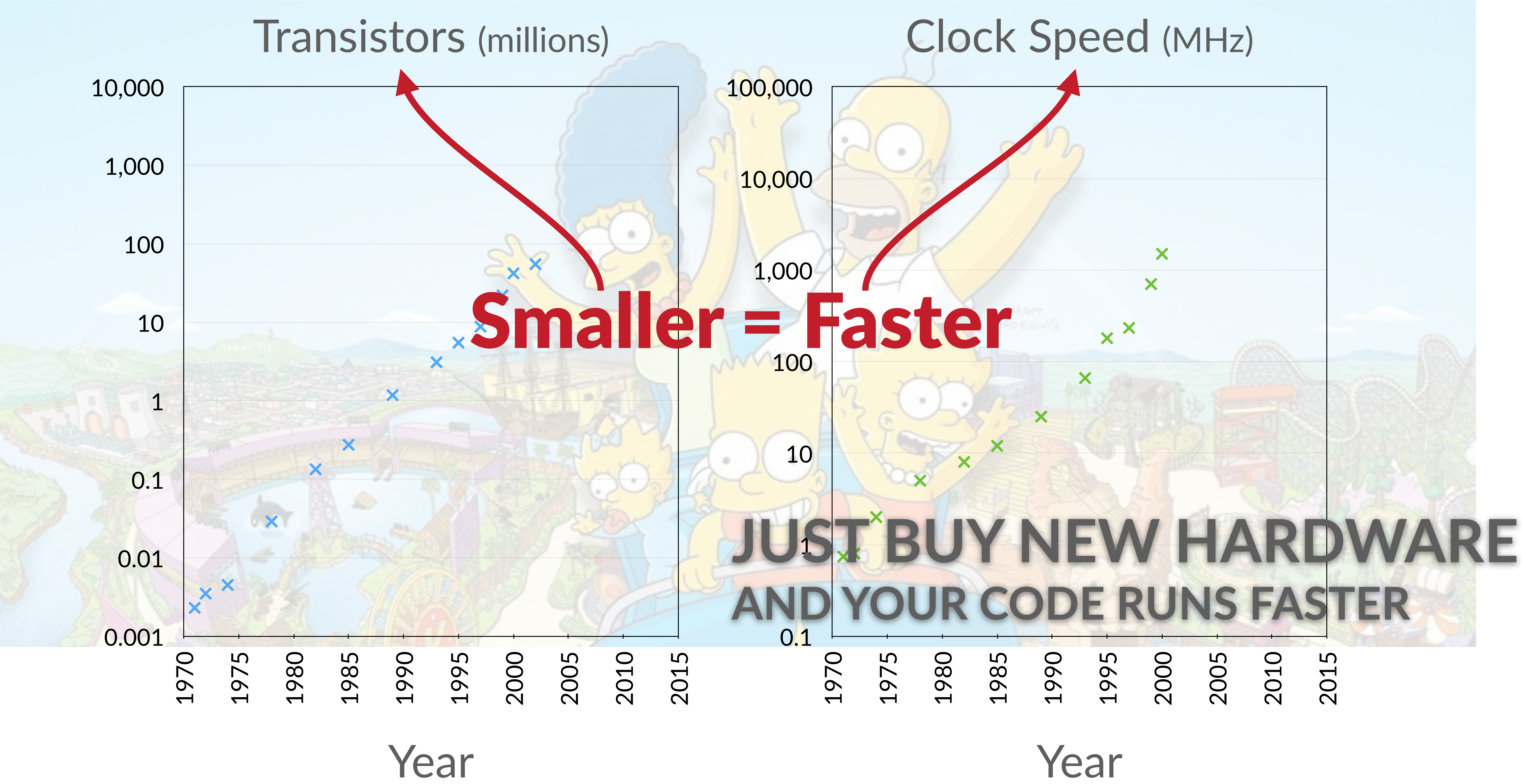


Wild Performance Ride (1980s–2010)!



Smaller = Faster

Wild Performance Ride (1980s–2010)!

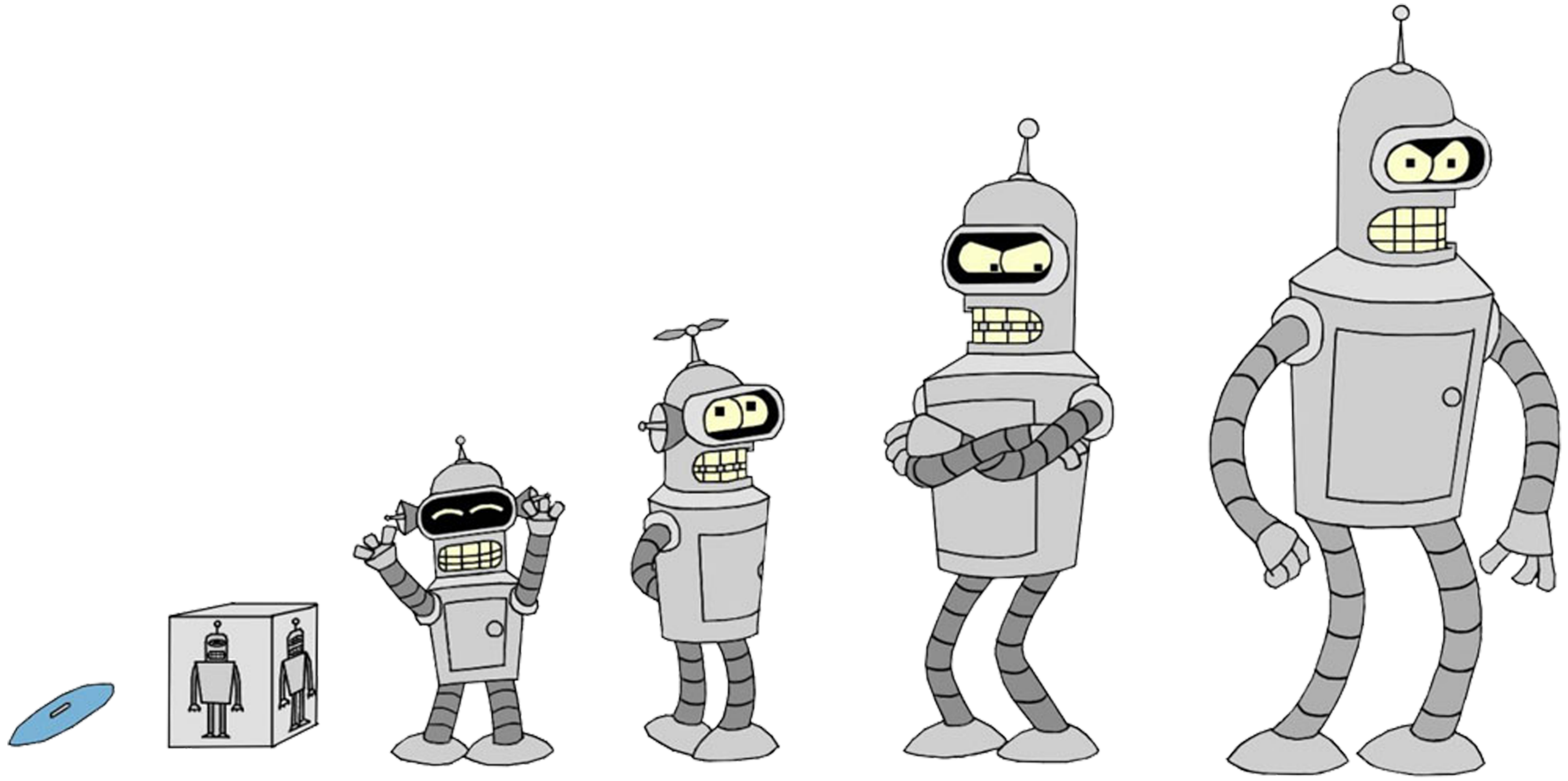


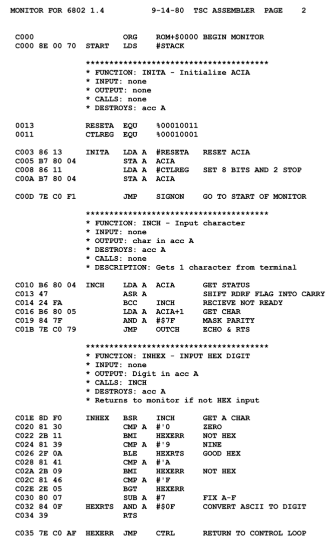
Smaller = Faster

JUST BUY NEW HARDWARE

AND YOUR CODE RUNS FASTER

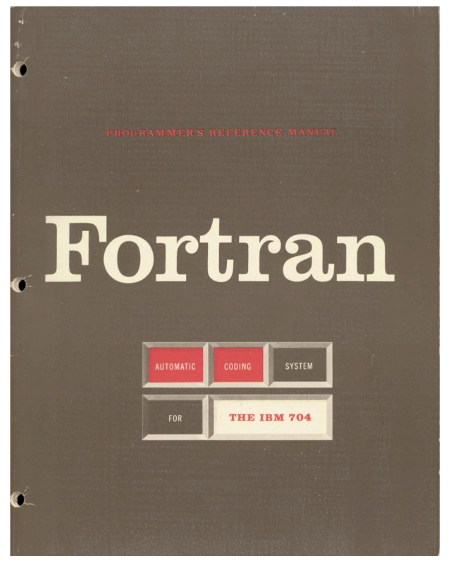
"Metal" Languages –1980s





```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                ***************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ***************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ***************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

1949

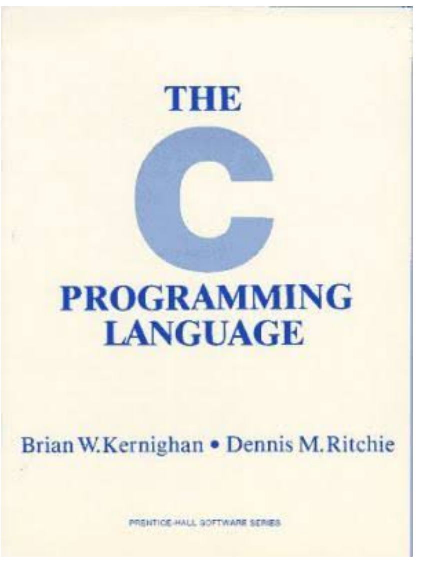


PROGRAMMER'S REFERENCE MANUAL

Fortran

AUTOMATIC CODING SYSTEM FOR THE IBM 704

1957

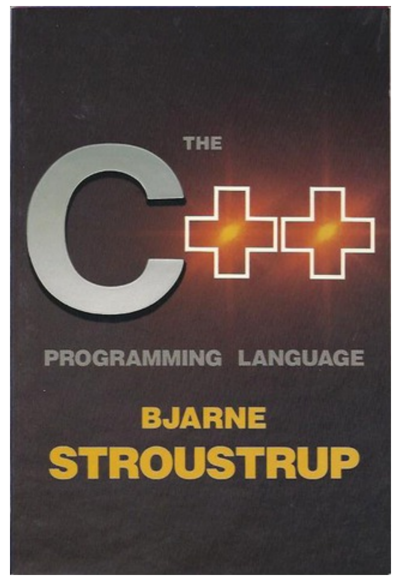


THE C PROGRAMMING LANGUAGE

Brian W.Kernighan • Dennis M.Ritchie

PRENTICE-HALL SOFTWARE SERIES

1972



THE C++ PROGRAMMING LANGUAGE

BJARNE STROUSTRUP

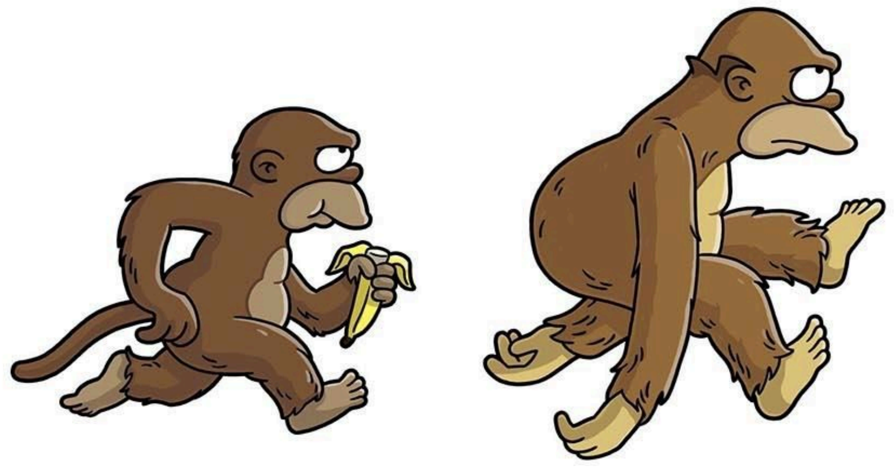
1985

"Irrational Exuberance" Languages
1990s

"Irrational Exuberance" Languages
1990s

1991

"Irrational Exuberance" Languages
1990s



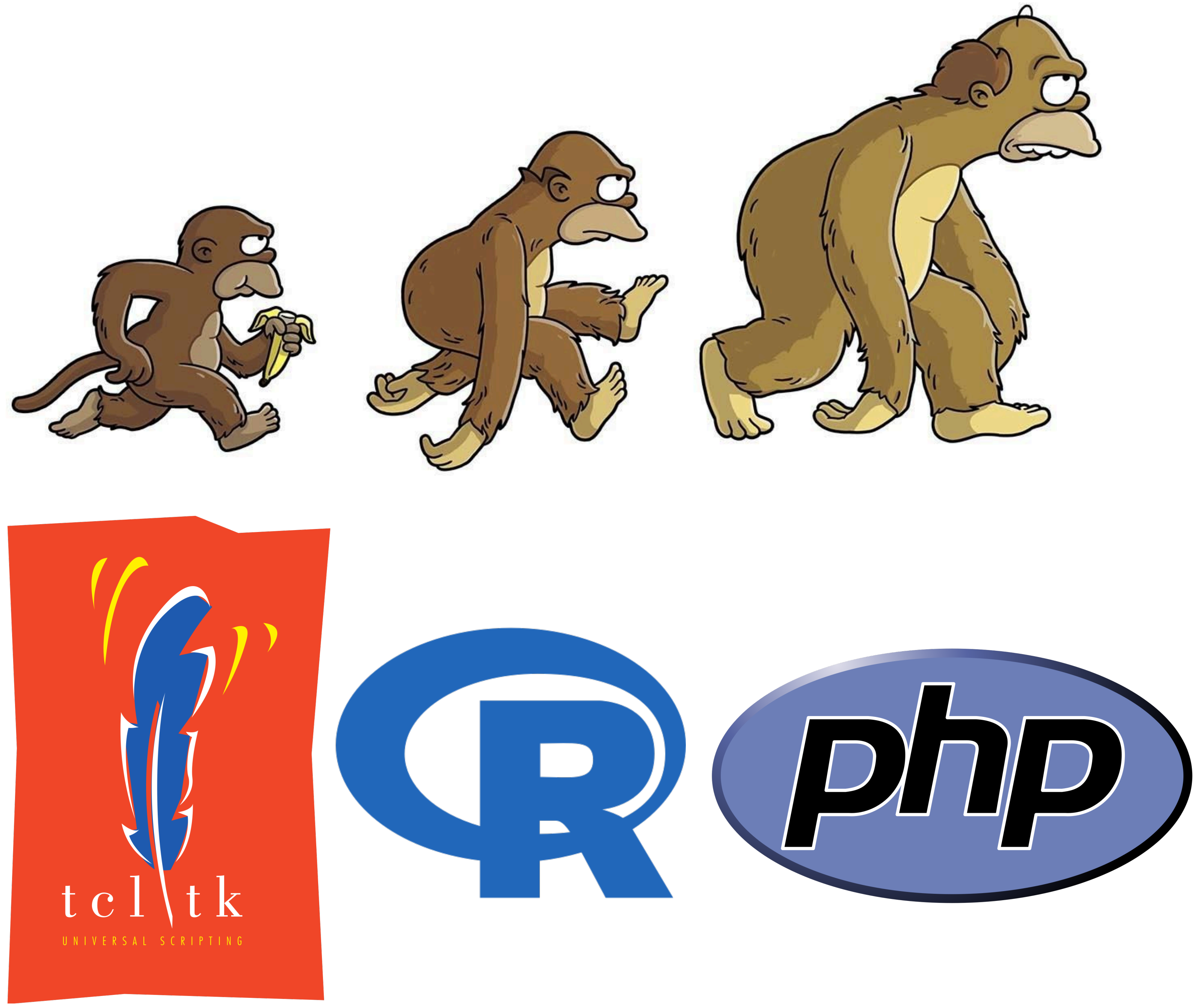
tcl tk
UNIVERSAL SCRIPTING

R

1991 1993

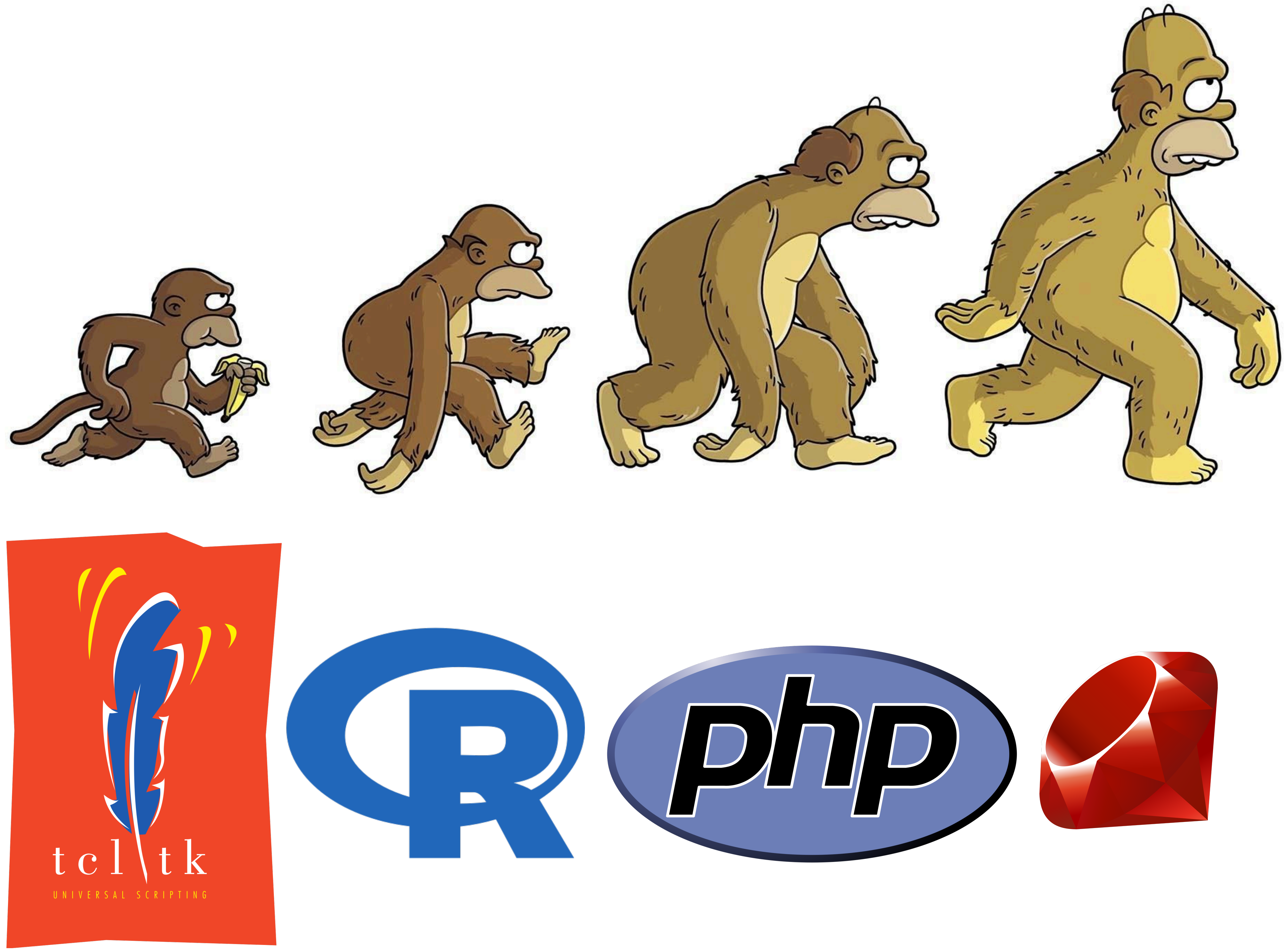
"Irrational Exuberance" Languages

1990s



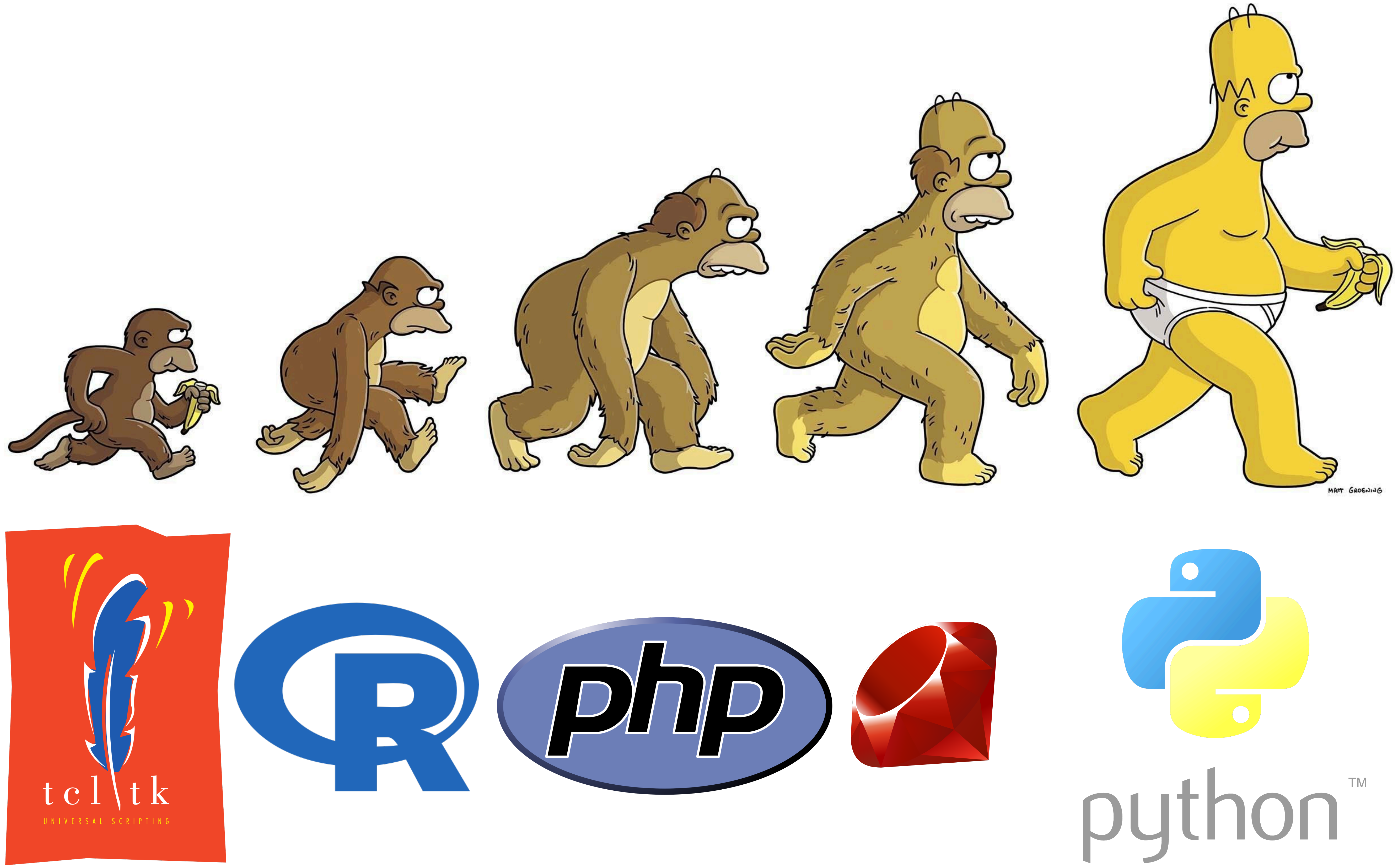
1991 1993 1995

"Irrational Exuberance" Languages
1990s

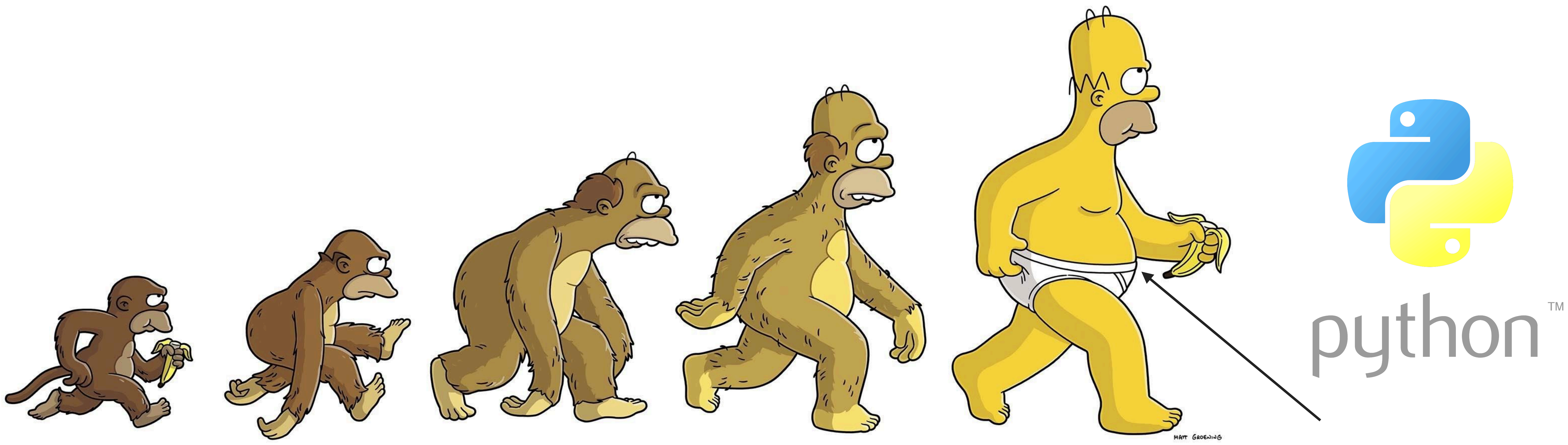


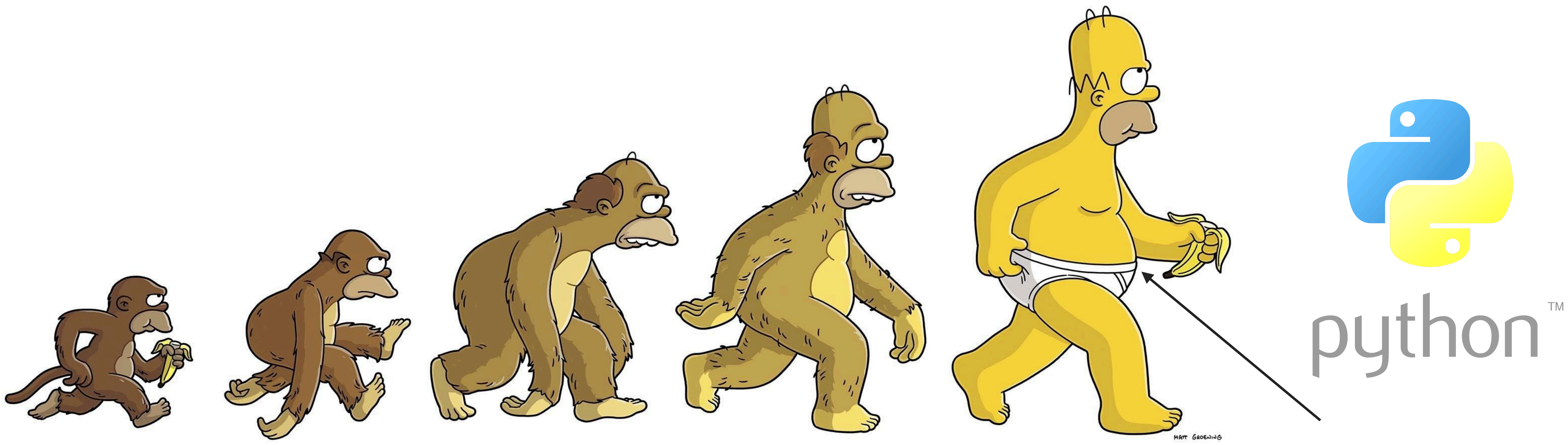
1991 1993 1995 1995

"Irrational Exuberance" Languages
1990s

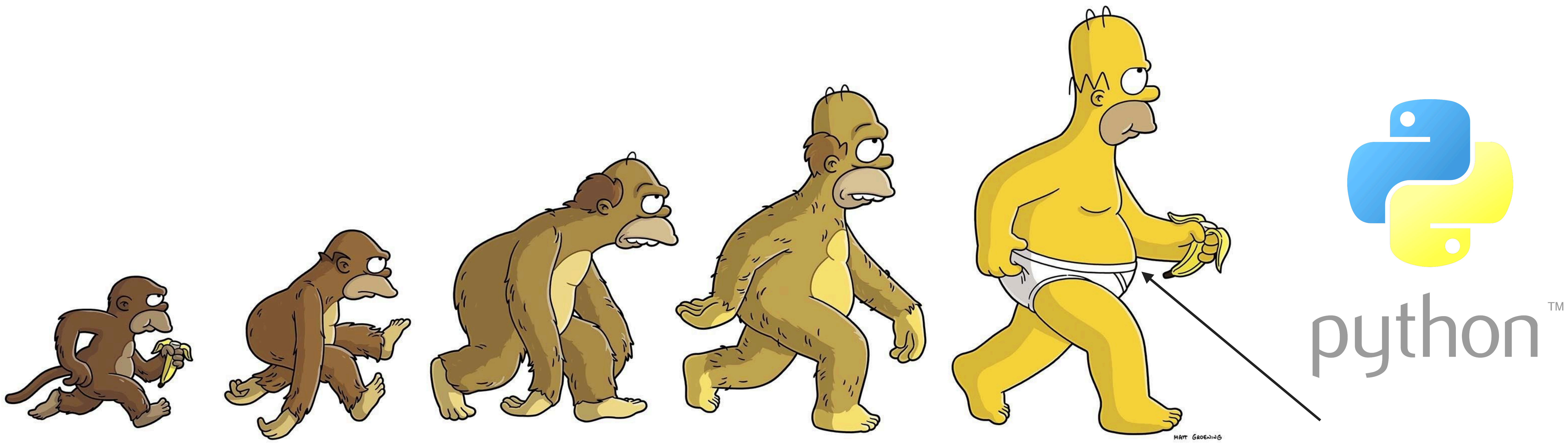


1991 1993 1995 1995 1990

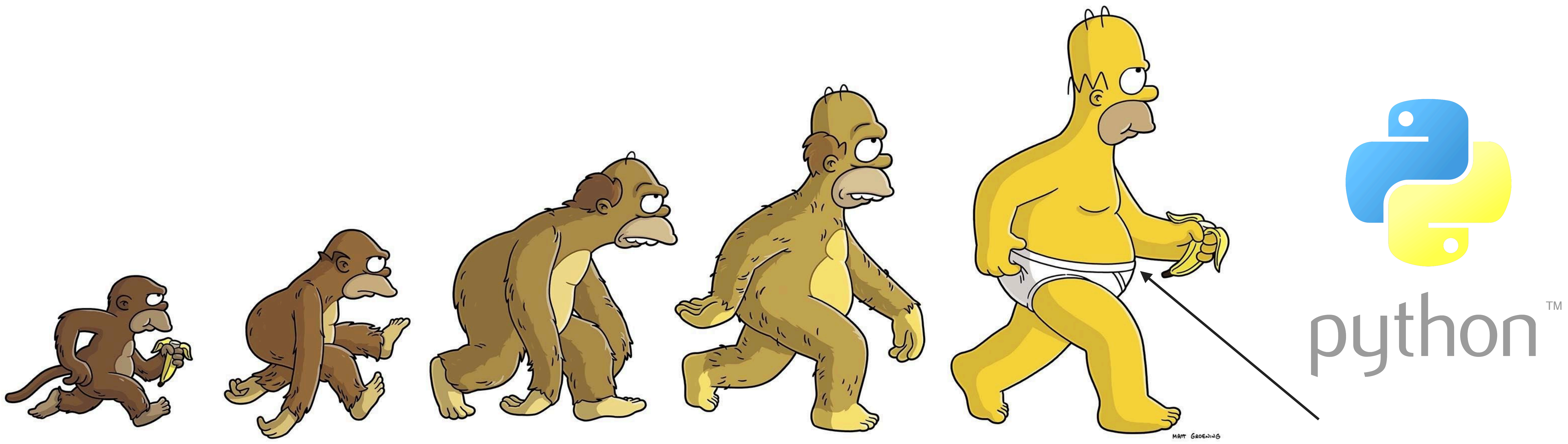

MATT GROENING
python™



HOW MANY BYTES IN…(INT, LIST, DICT)?

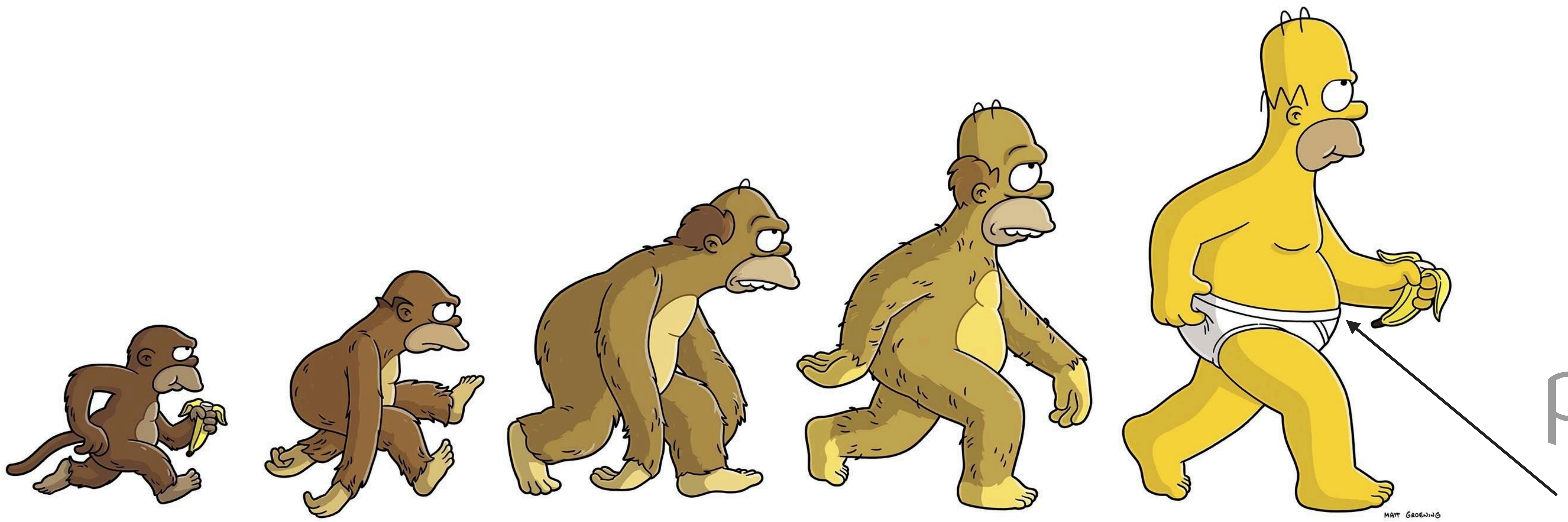


HOW MANY BYTES IN…(INT, LIST, DICT)?



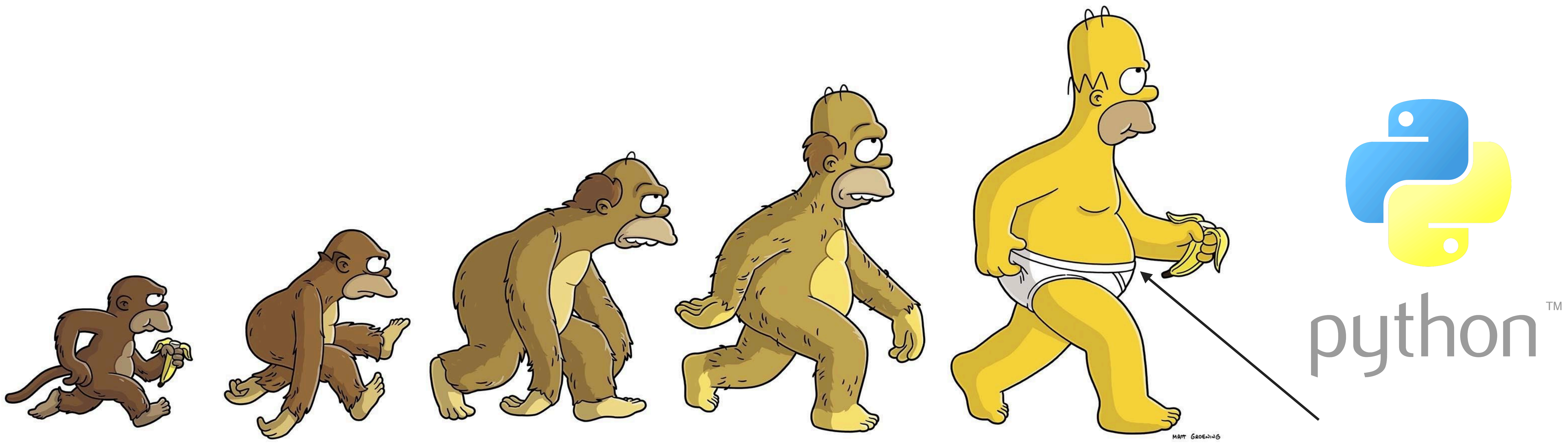
HOW MANY BYTES IN…(INT, LIST, DICT)?

```
sizeof(1)
→ 4
```



HOW MANY BYTES IN…(INT, LIST, DICT)?

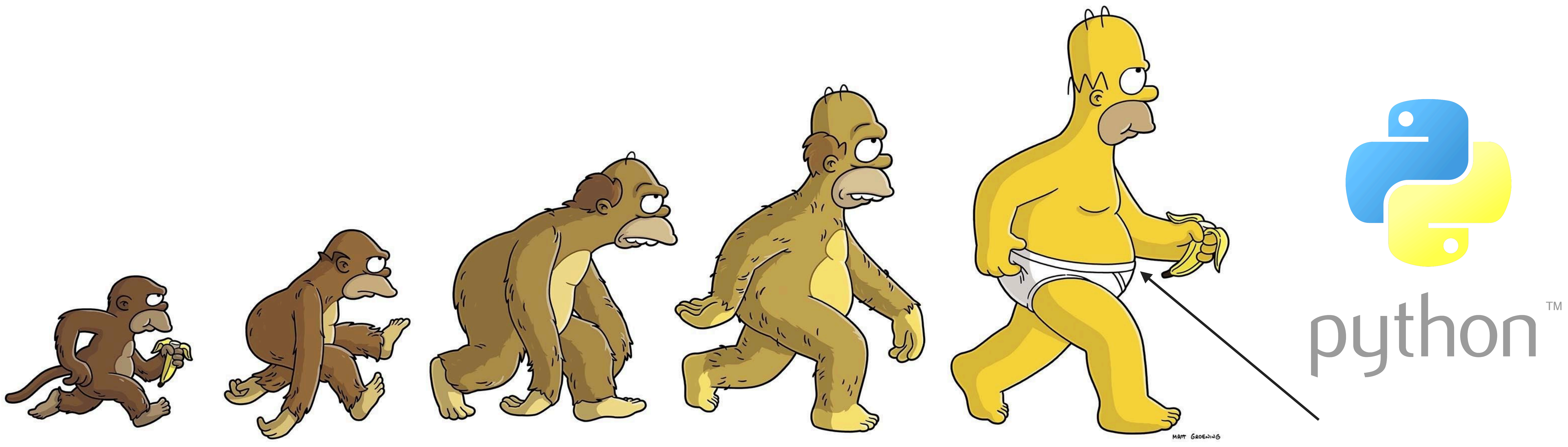
```
sizeof(1)
→ 4
```



HOW MANY BYTES IN…(INT, LIST, DICT)?

```
sizeof(1)
→ 4
```

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



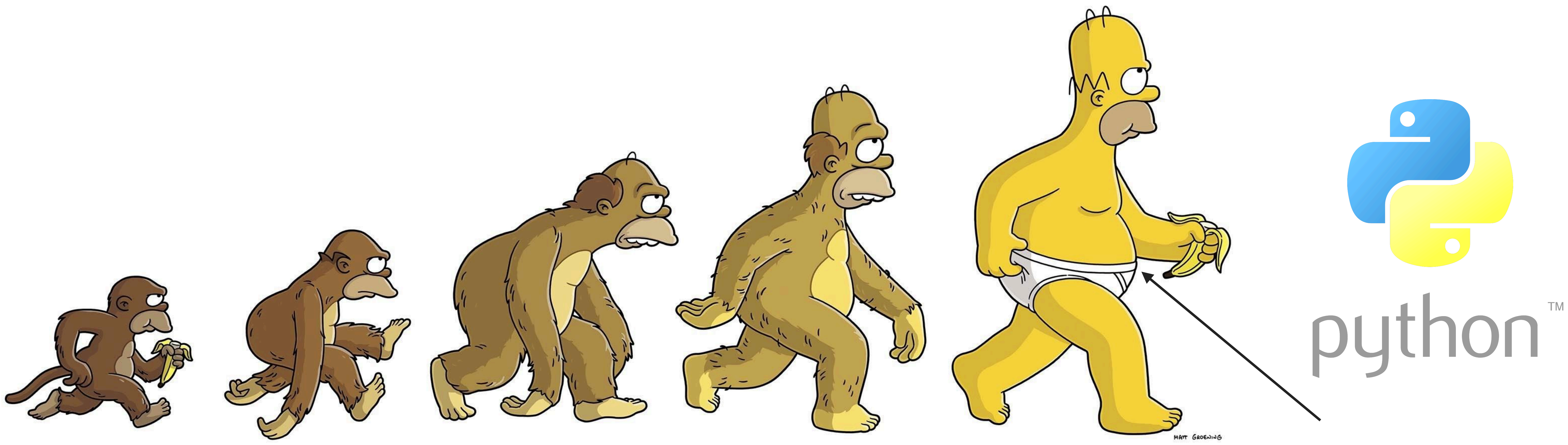
HOW MANY BYTES IN…(INT, LIST, DICT)?

```
sizeof(1)
→ 4
```

```
sizeof(list<int>)
→ 24
```

python™

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



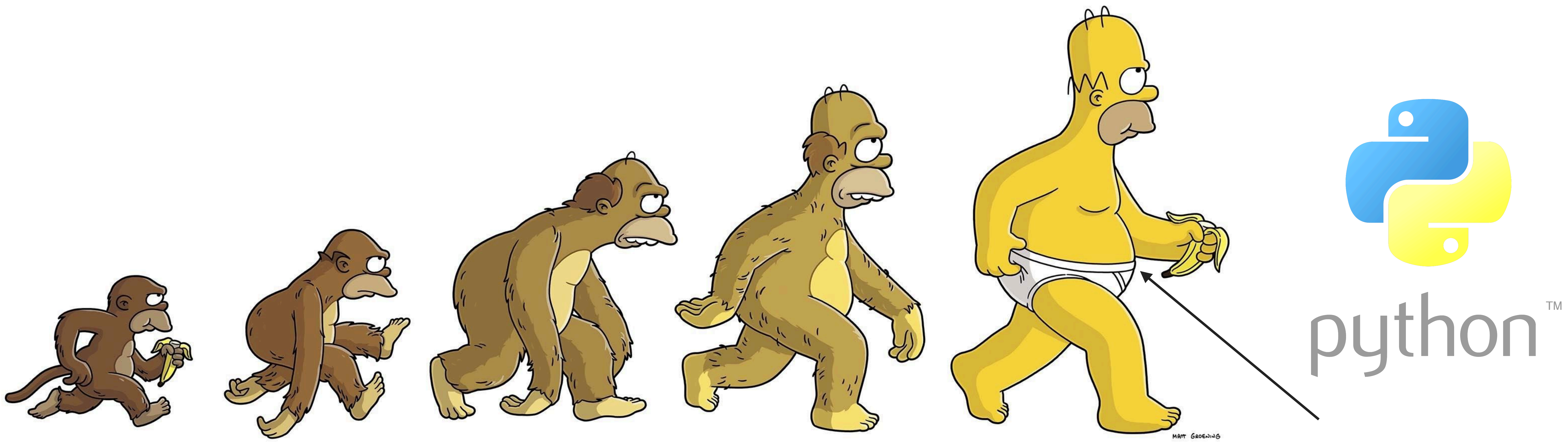
HOW MANY BYTES IN…(INT, LIST, DICT)?

```
sizeof(1)
→ 4
```

```
sizeof(list<int>)
→ 24
```

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

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



HOW MANY BYTES IN…(INT, LIST, DICT)?

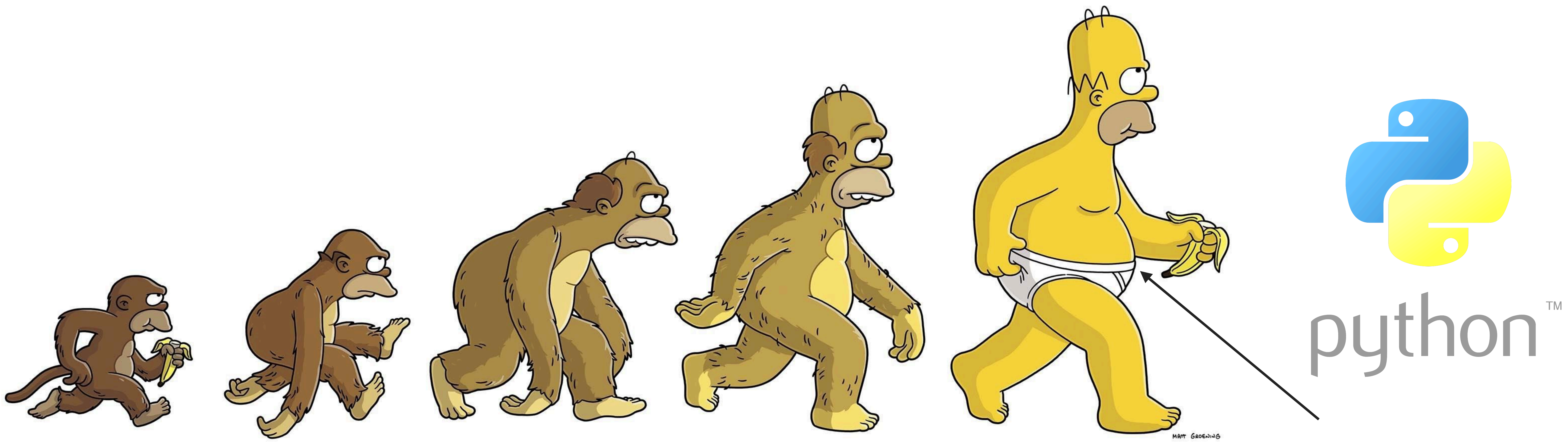
```
sizeof(1)
→ 4
```

```
sizeof(list<int>)
→ 24
```

```
sizeof(map<int,
int>)
→ 24
```

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

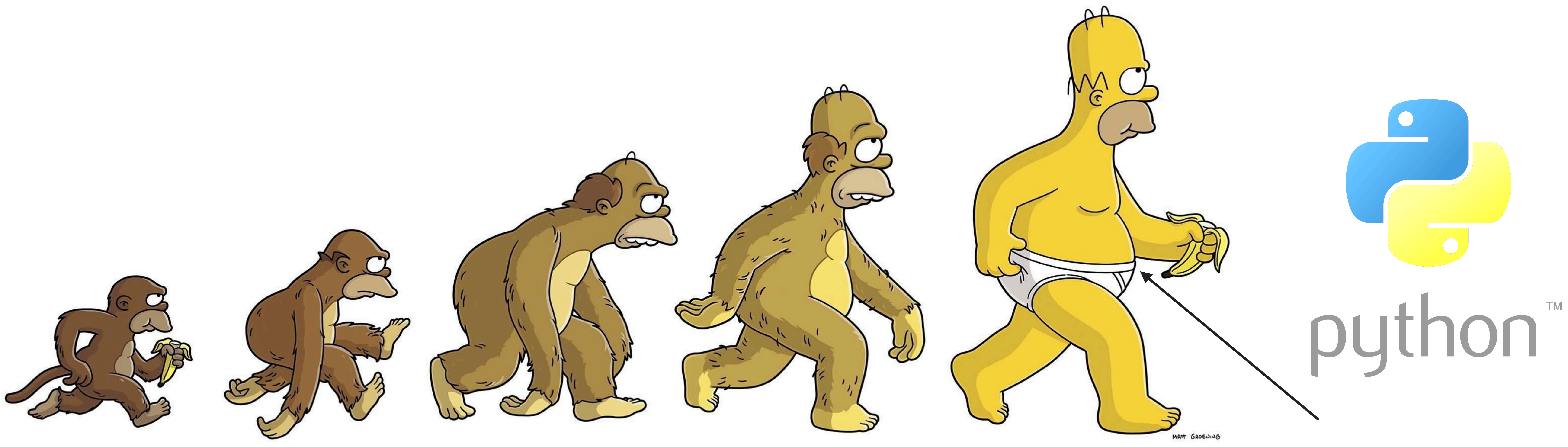
```
>>> sys.getsizeof([])
56
```



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
```

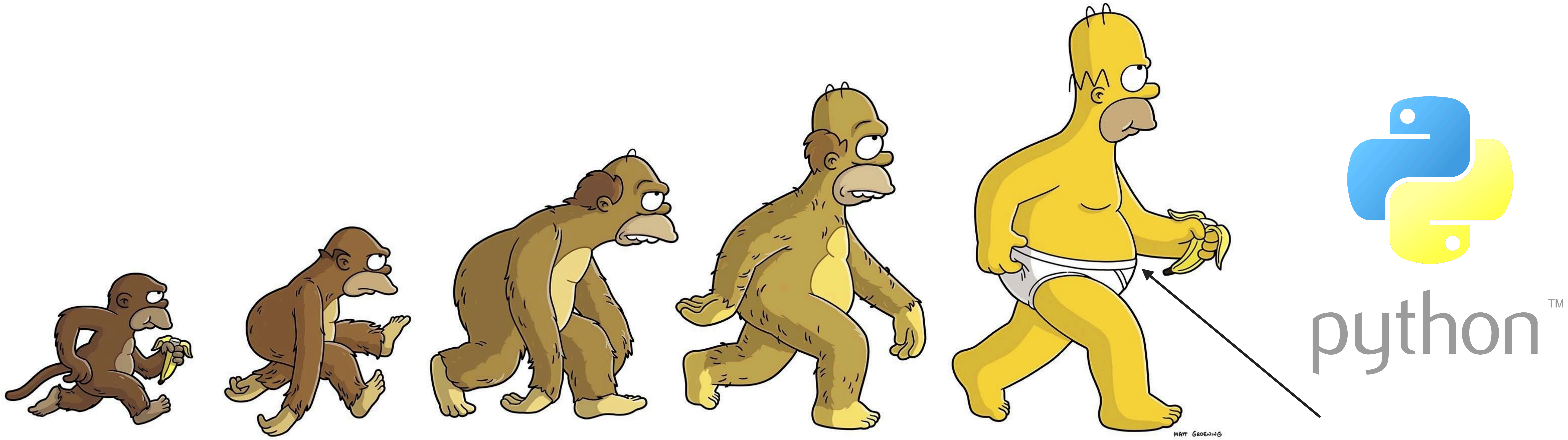
```
sizeof(map<int,
int>)
→ 24
```

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

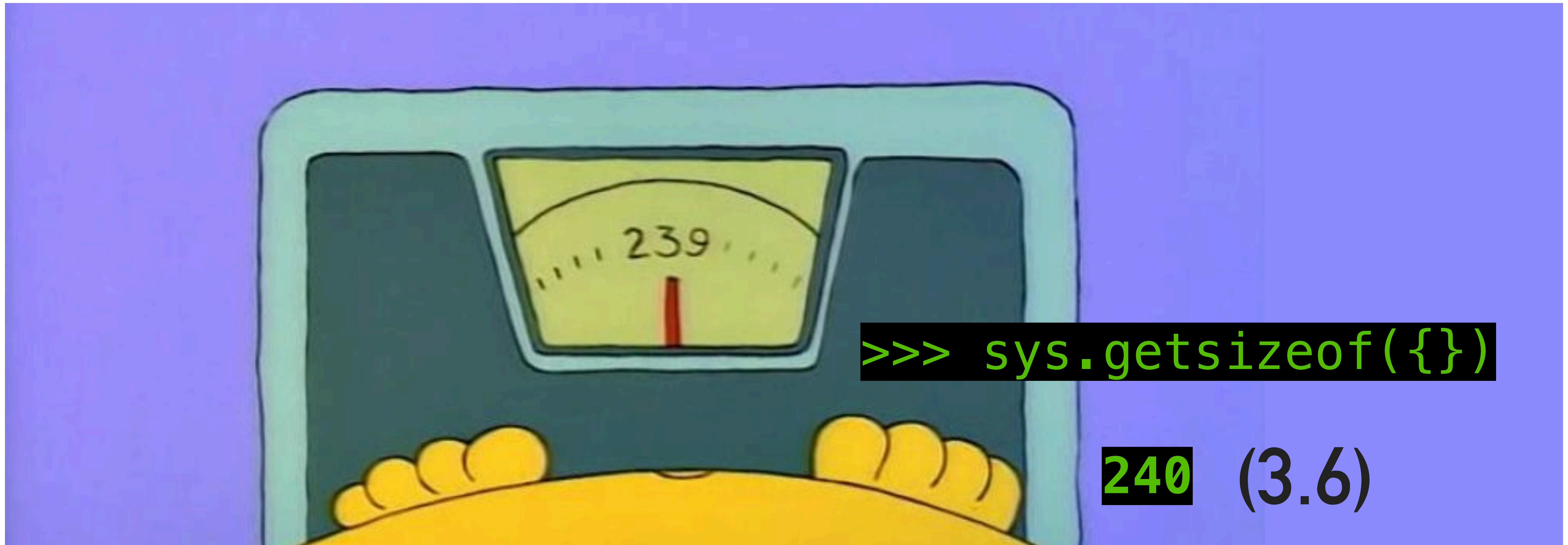
```
>>> sys.getsizeof([])
56
```

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

`240` (3.6)

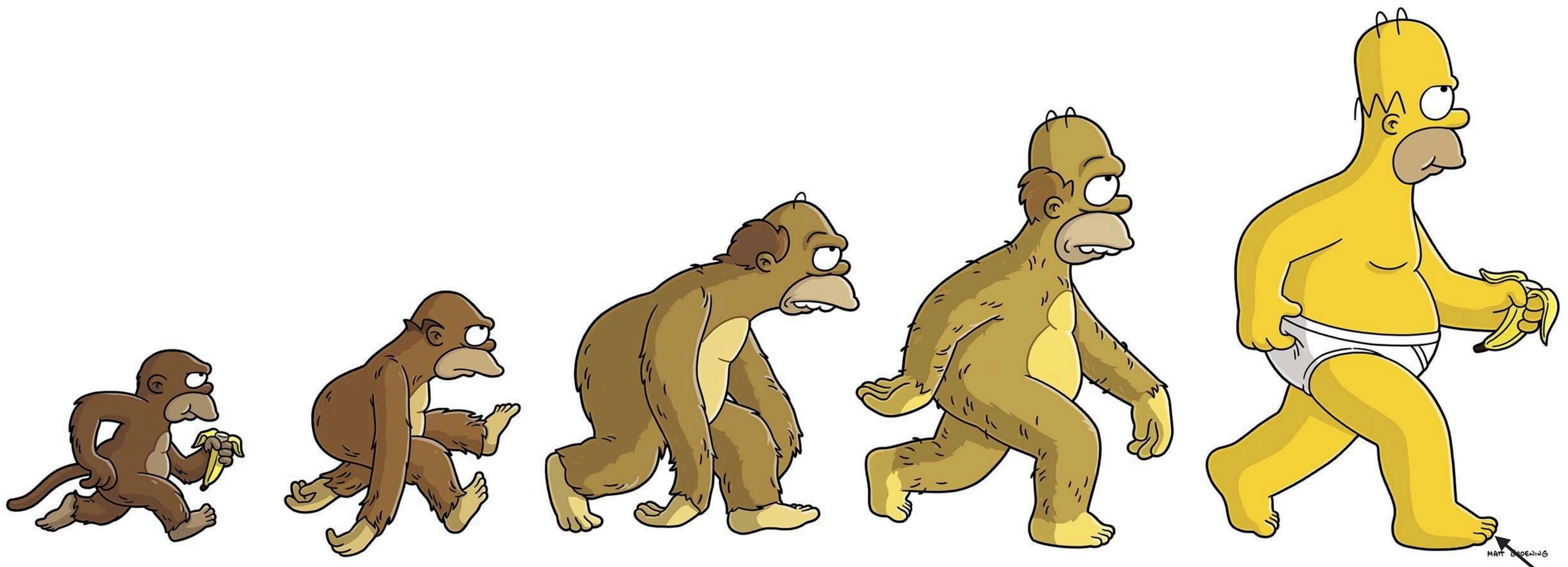


HOW MANY BYTES IN…(INT, LIST, DICT)?

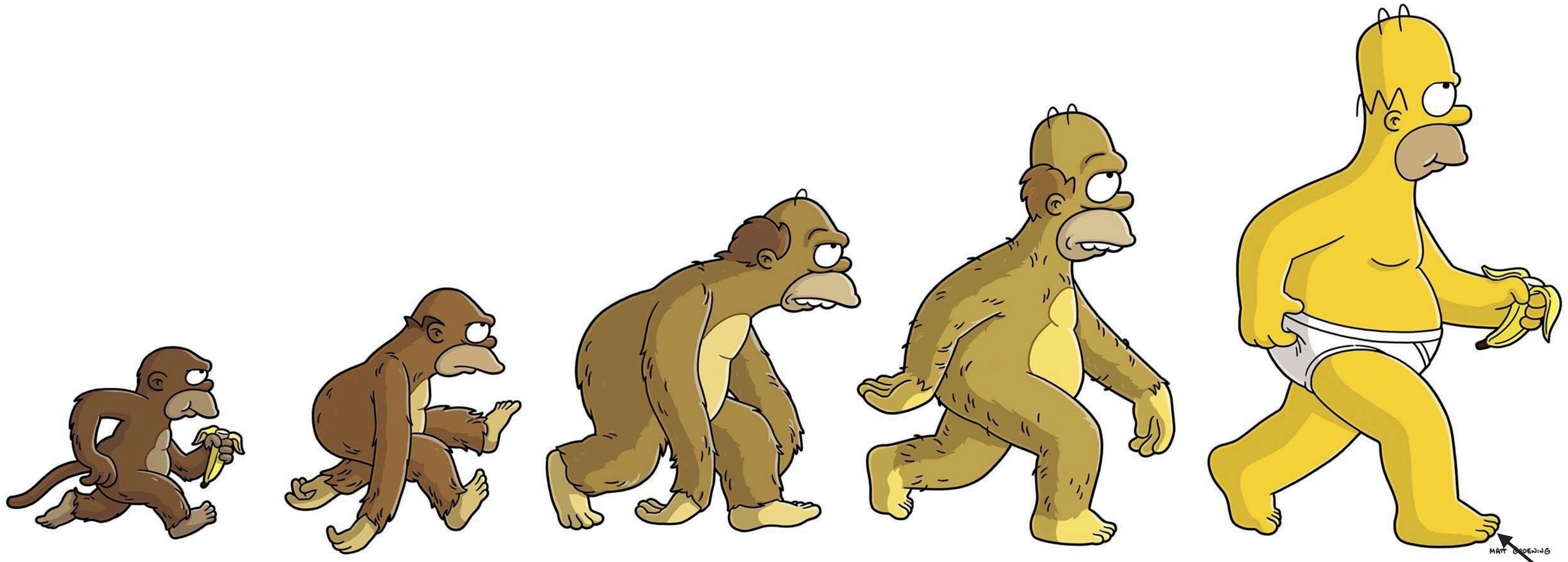


```
>>> sys.getsizeof({})
240
```

(3.6)



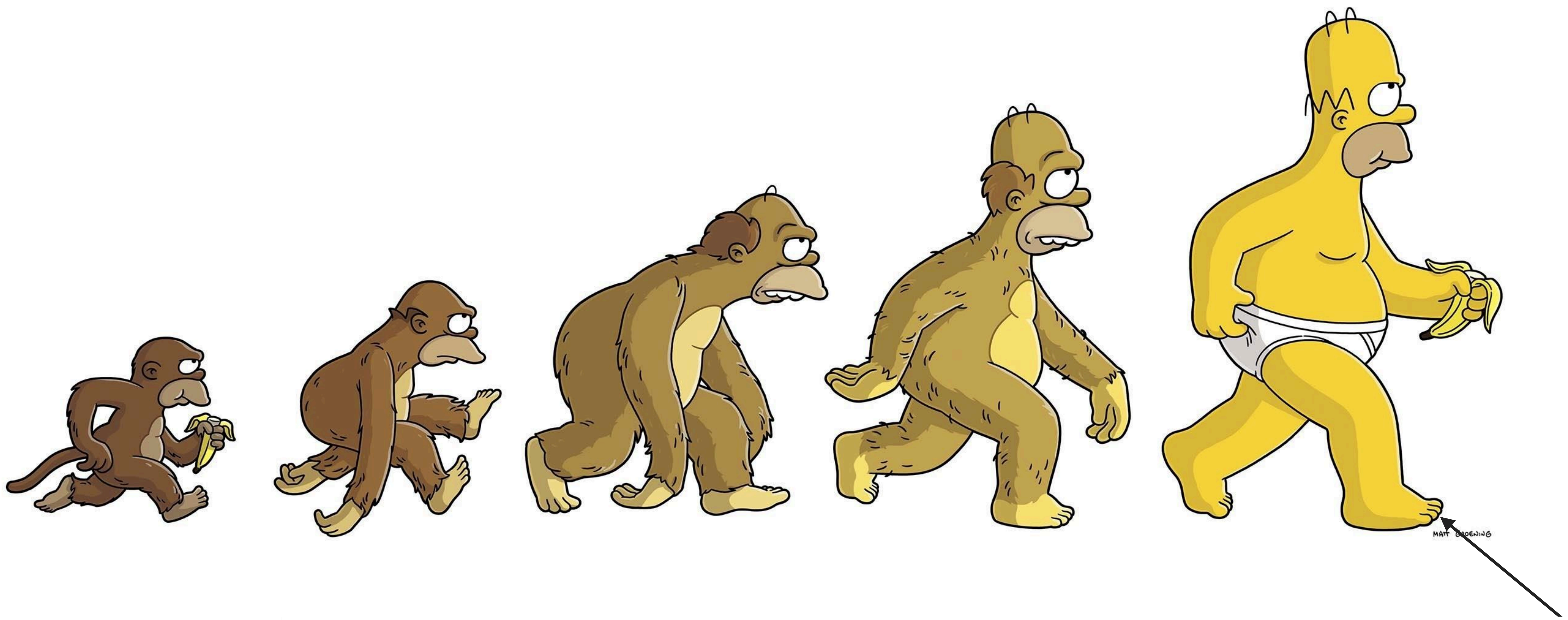
python™



python™

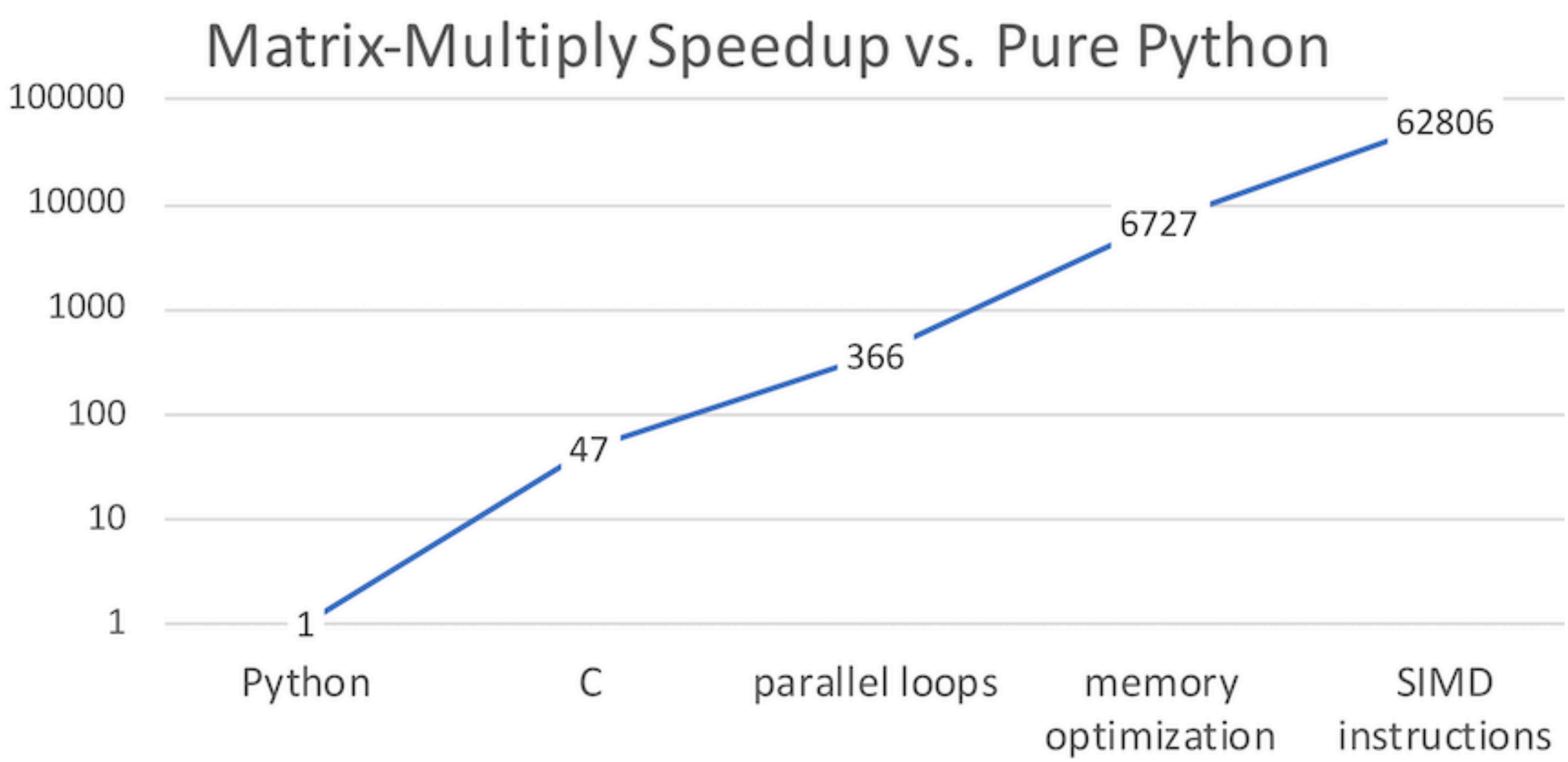
MATRIX-MATRIX MULTIPLY

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

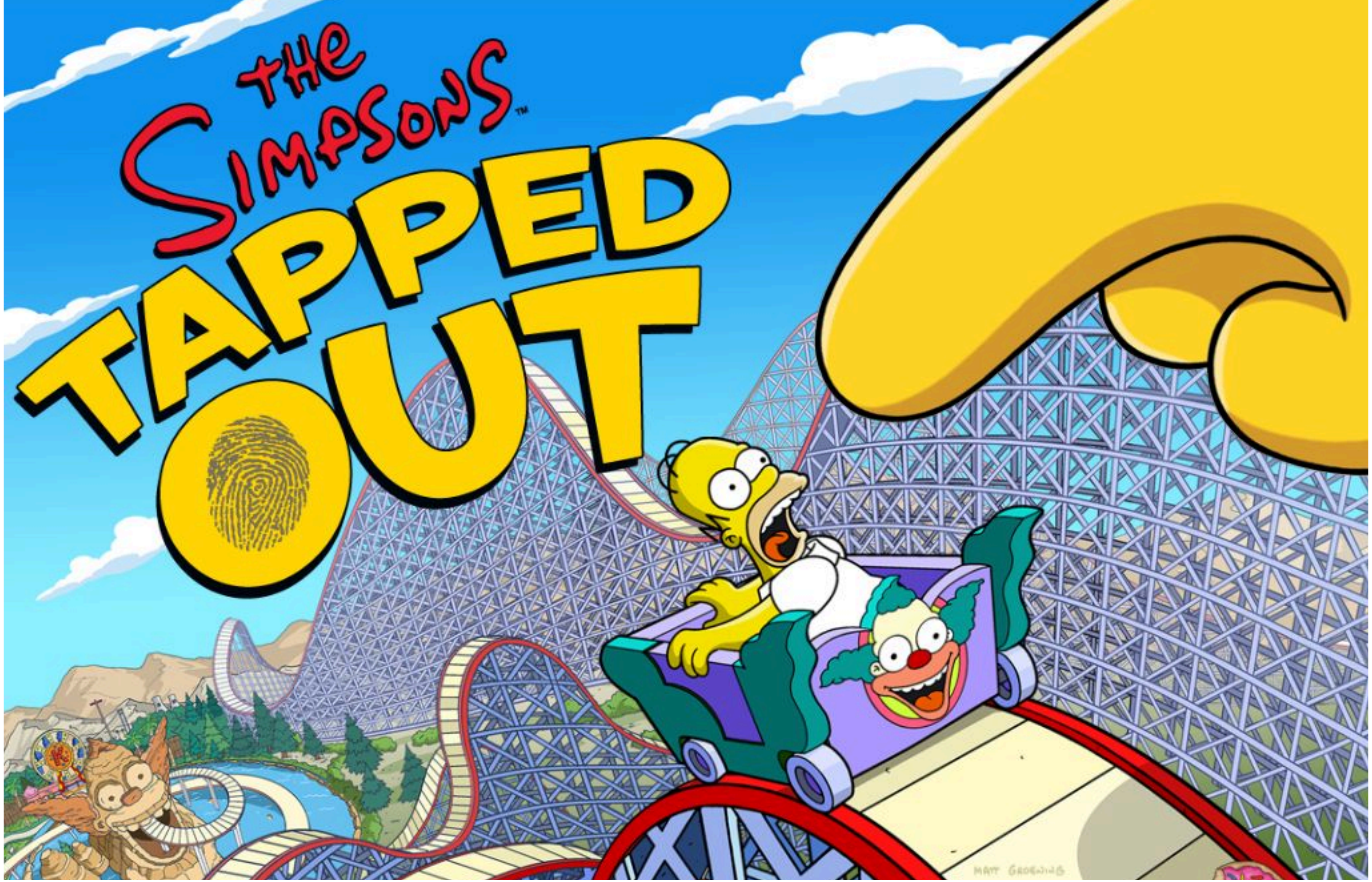


python™

MATRIX-MATRIX MULTIPLY

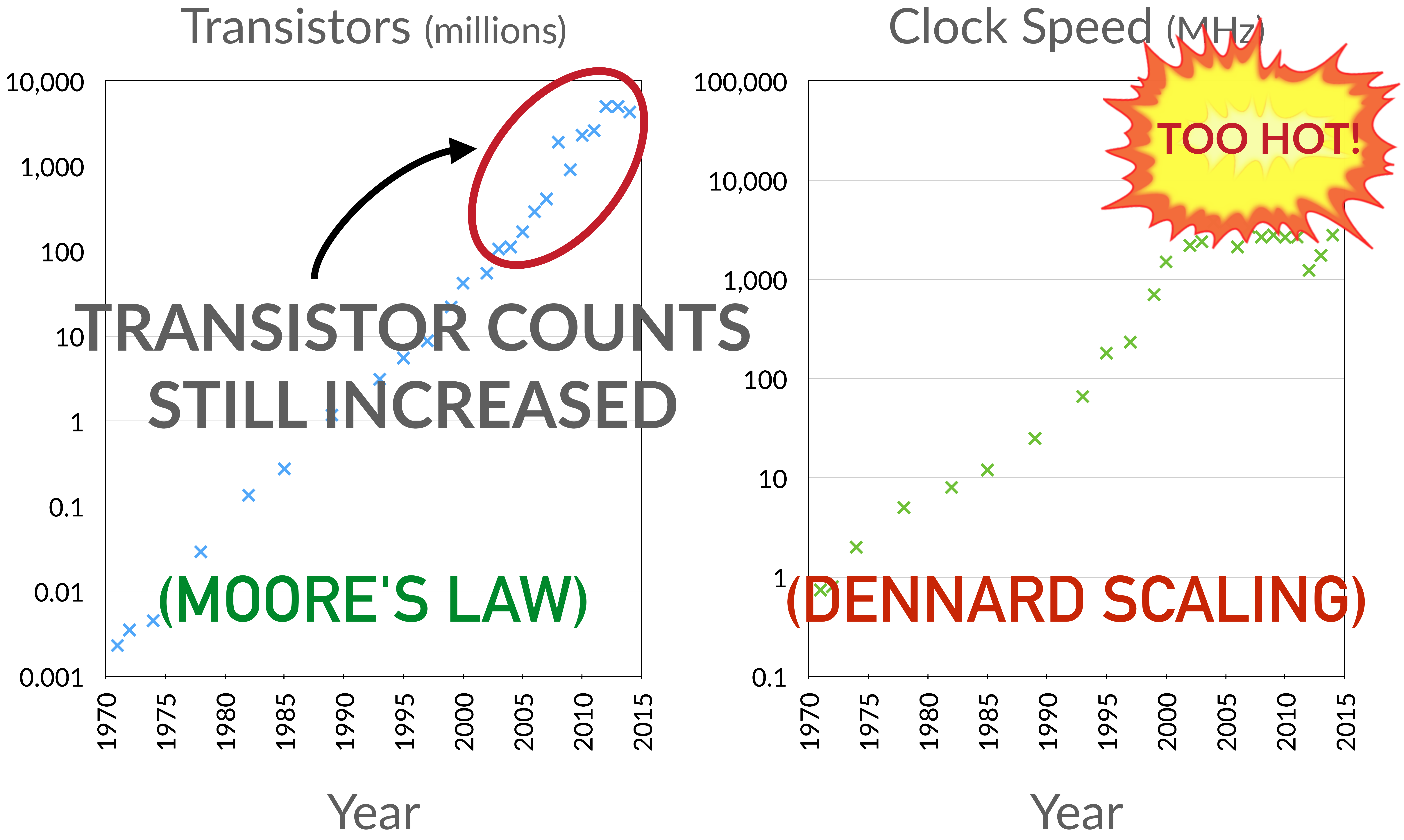


60,000X slowdown!



≈2010: THE RIDE IS OVER

The Ride Is Over



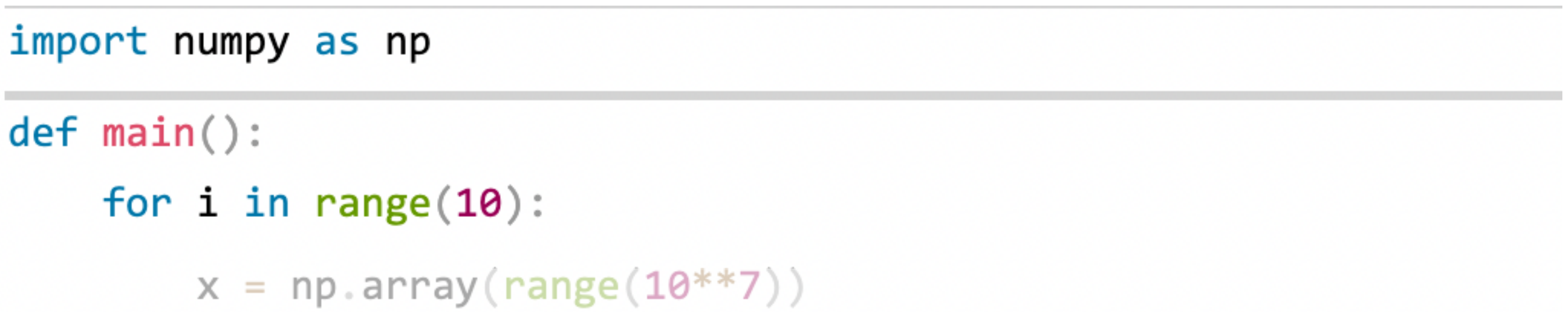
TRANSISTOR COUNTS STILL INCREASED

(MOORE'S LAW)

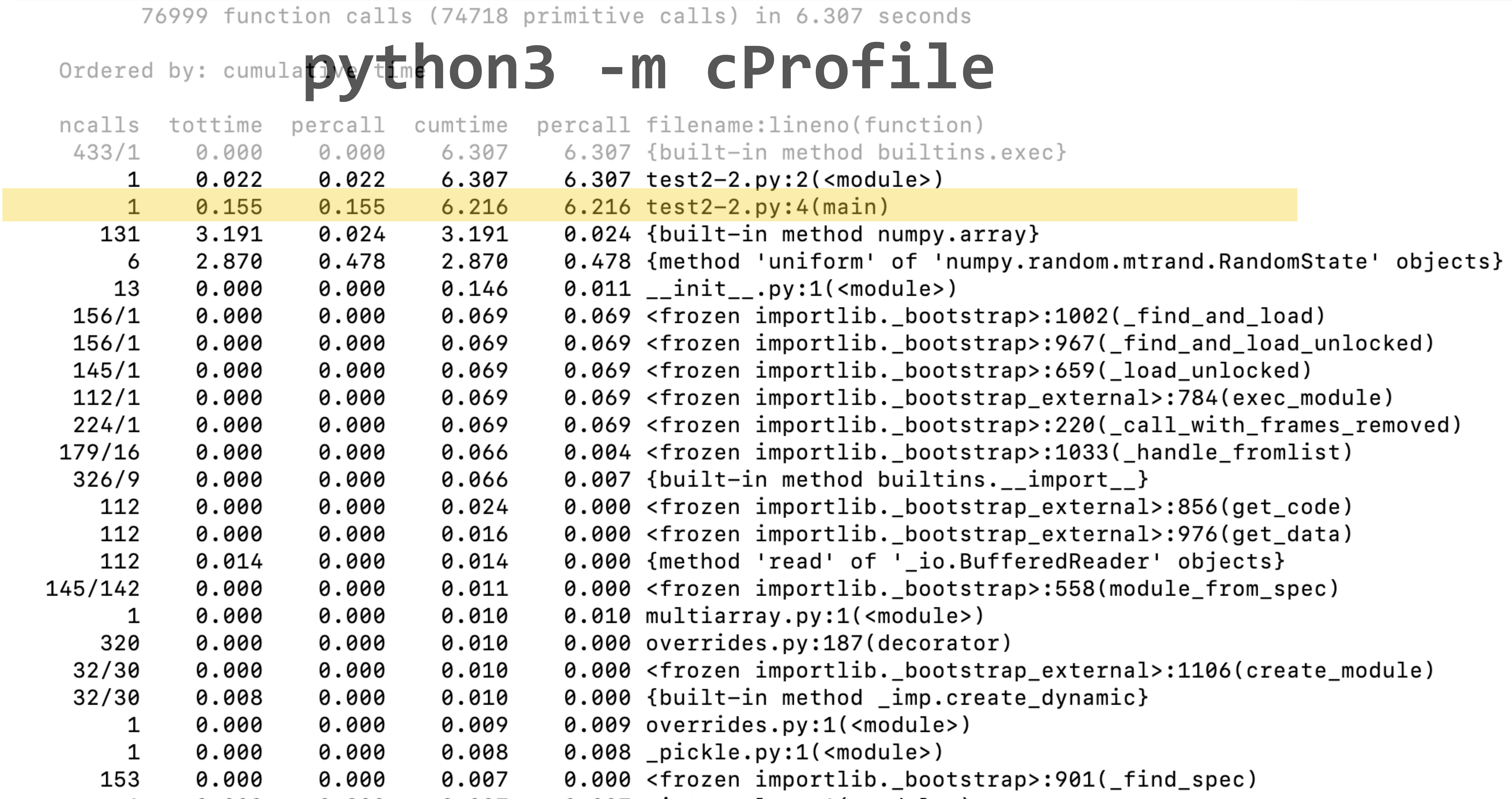
TOO HOT!

(DENNARD SCALING)

python3 -m cProfile



```python
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

   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”

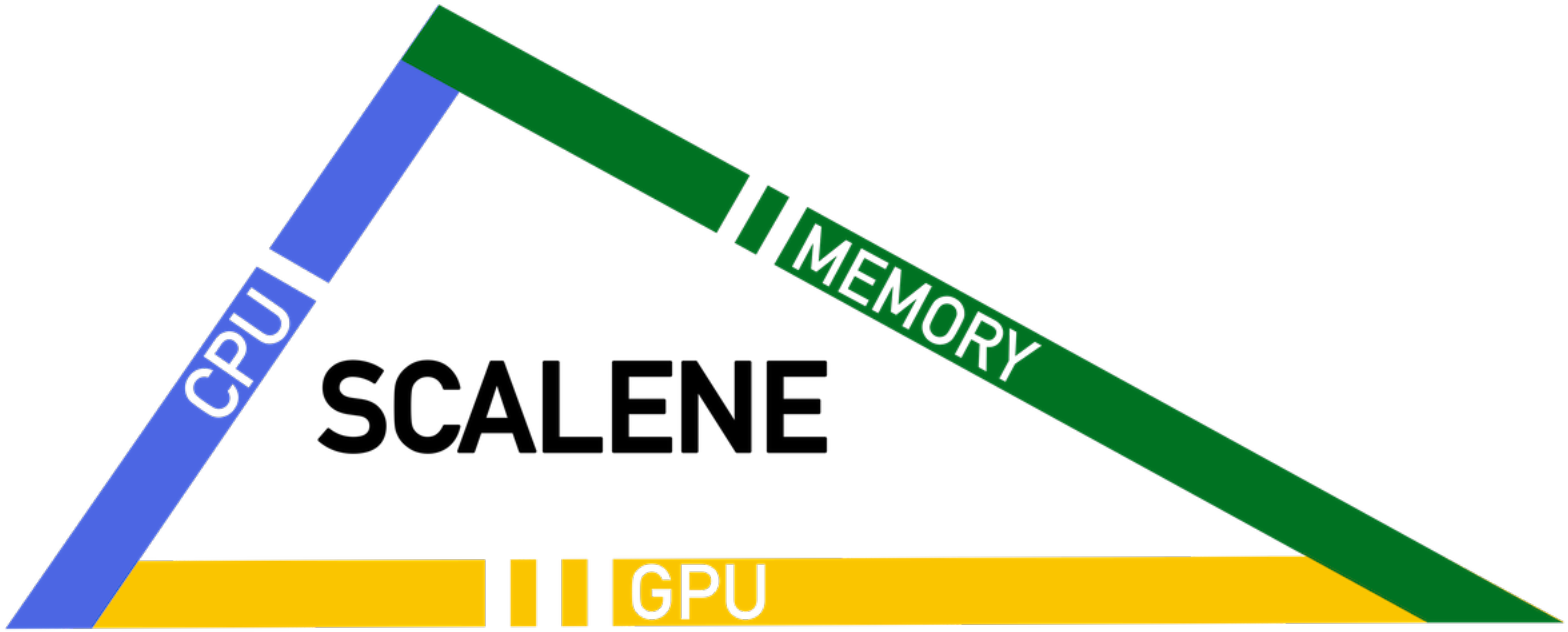
PYTHON

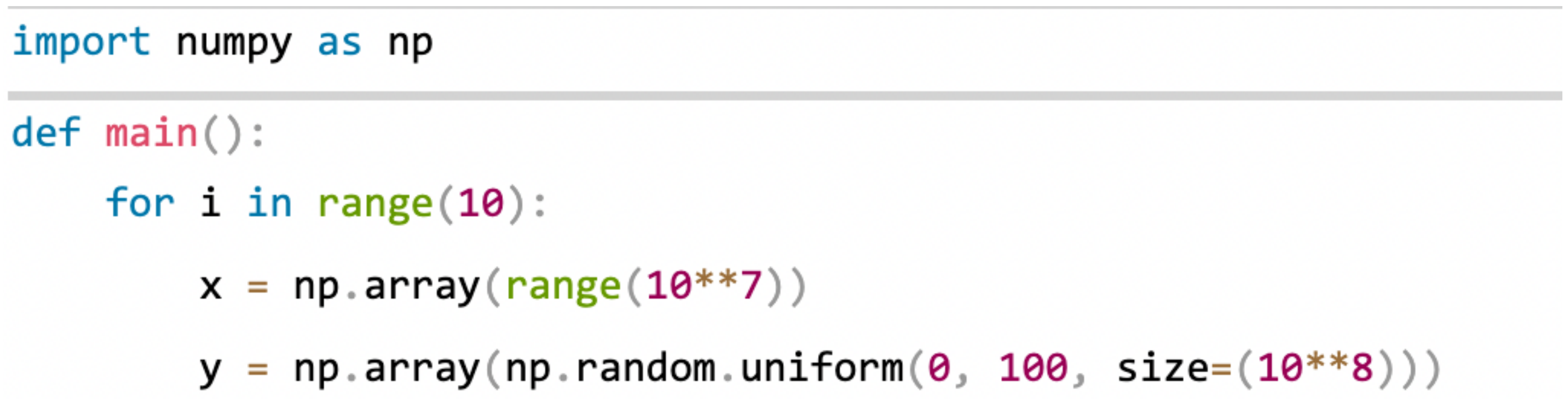
scikit

tensorflow

numpy

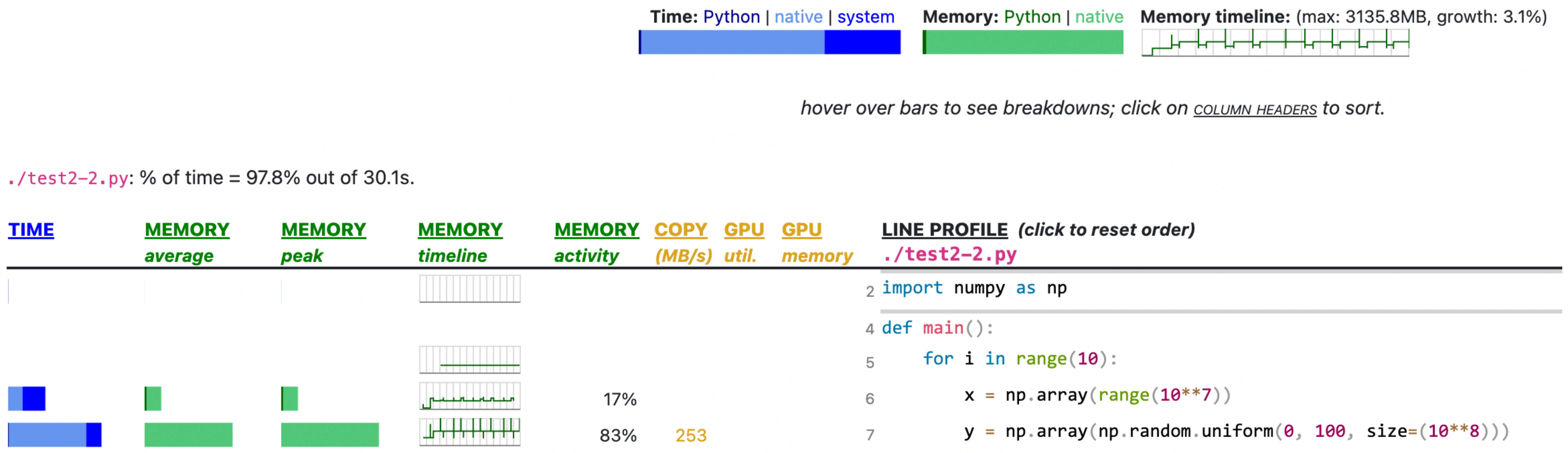
THE “C”


CPU
MEMORY
SCALENE
GPU



```python
import numpy as np
```

```python
def main():
    for i in range(10):
        x = np.array(range(10**7))
        y = np.array(np.random.uniform(0, 100, size=(10**8)))
```

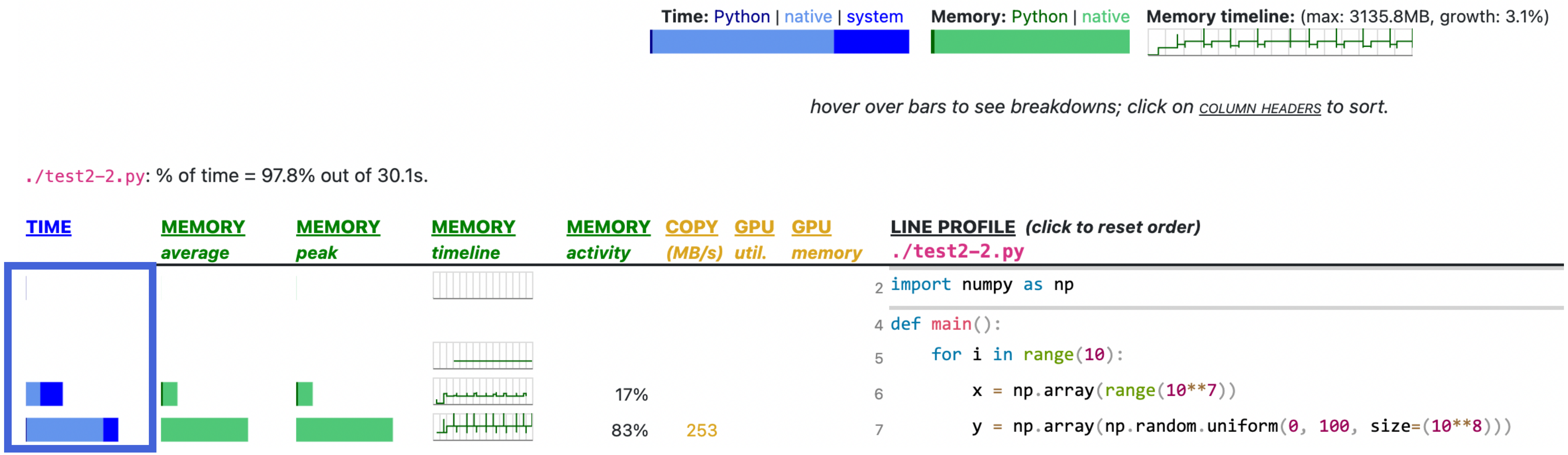


Time: Python | native | system   **Memory:** Python | native   **Memory timeline:** (max: 3135.8MB, growth: 3.1%)

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

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

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*	LINE PROFILE *(click to reset order)* ./test2-2.py
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				17%				`        x = np.array(range(10**7))`
				83%	253			`        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.

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

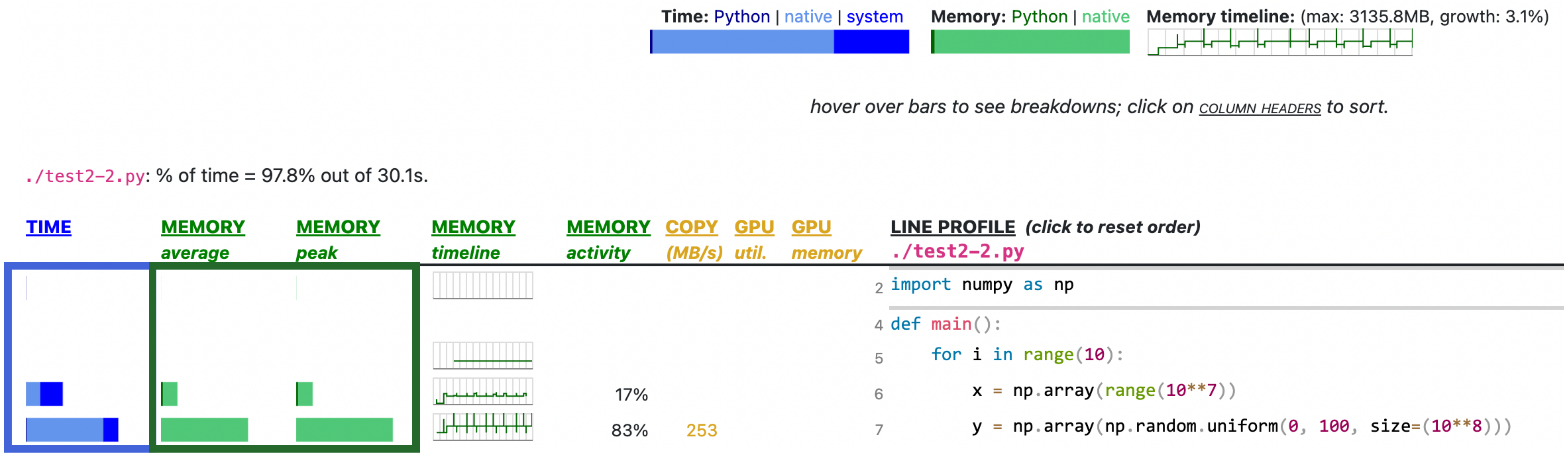
CPU

PYTHON

NATIVE

SYS%

CPU MEMORY SCALENE GPU



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.

```
TIME  MEMORY   MEMORY  MEMORY    MEMORY    COPY    GPU    GPU     LINE PROFILE (click to reset order)
      average  peak    timeline  activity  (MB/s)  util.  memory  ./test2-2.py
                                                                 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)))
```

CPU

PYTHON

NATIVE

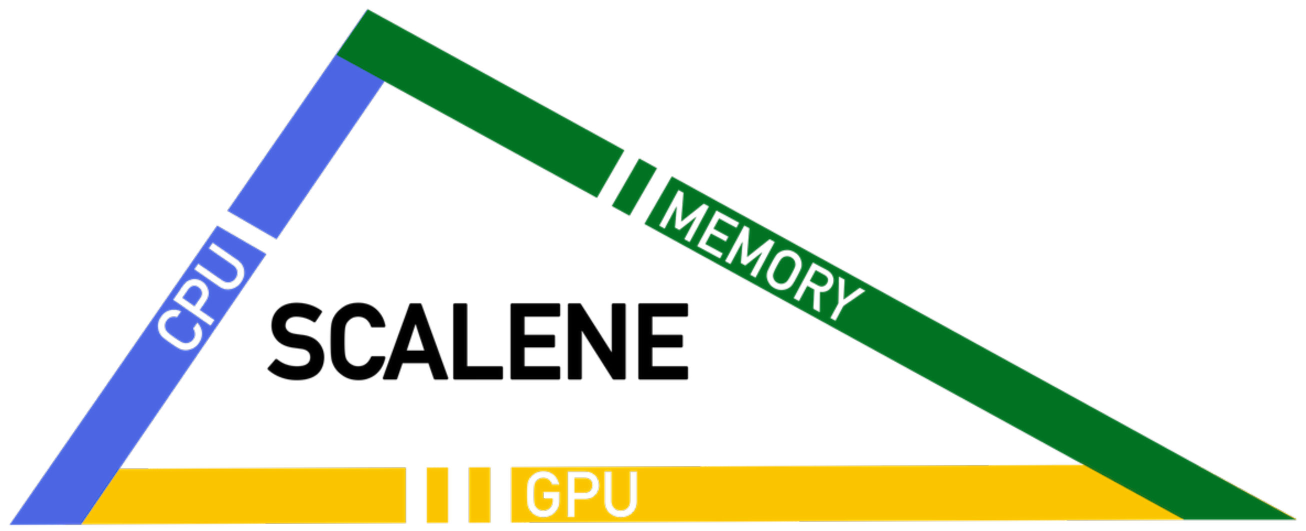
SYS%

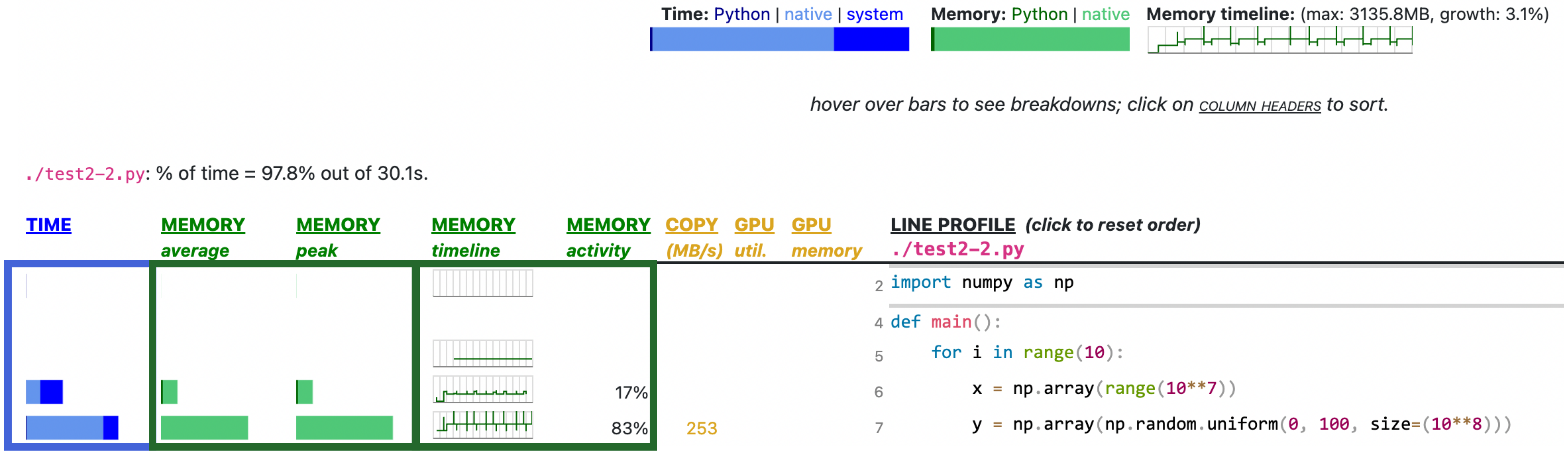
MEMORY

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.

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

CPU

PYTHON

NATIVE

SYS%

MEMORY

PYTHON

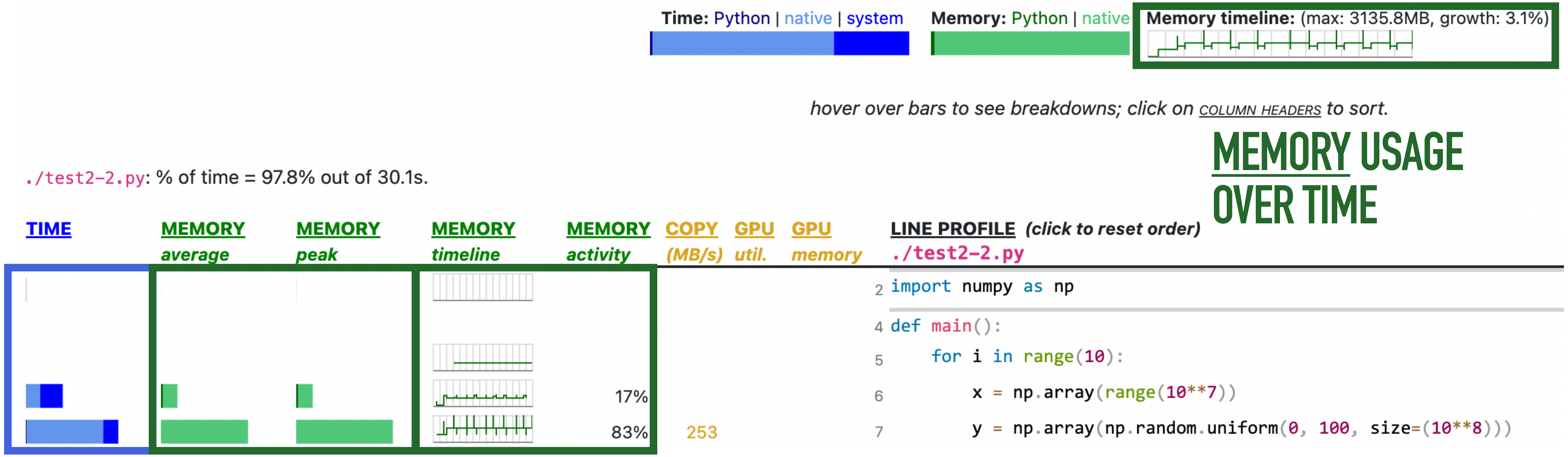
NATIVE

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.

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

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

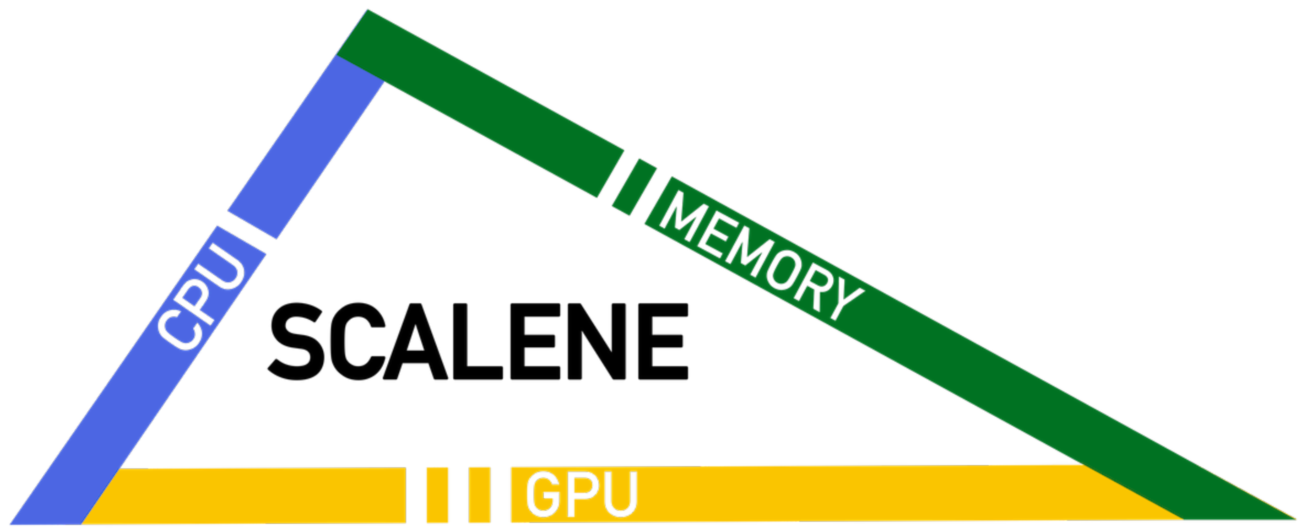
```
import numpy as np

def main():
    for i in range(10):
        x = np.array(range(10**7))
        y = np.array(np.random.uniform(0, 100, size=(10**8)))
```

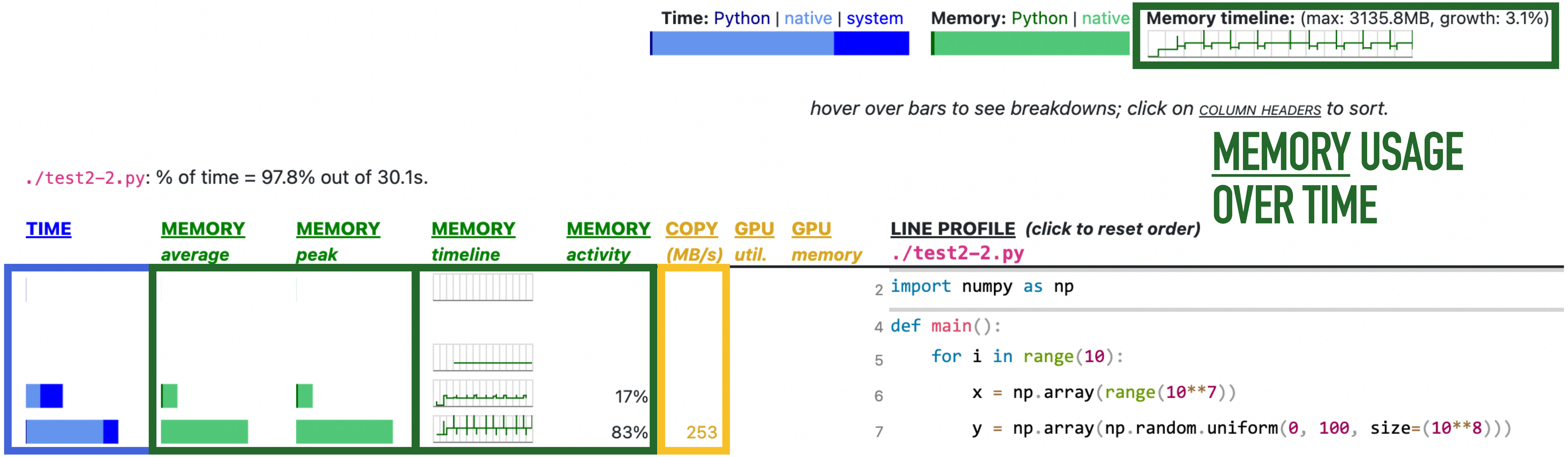
MEMORY USAGE OVER TIME

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.

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

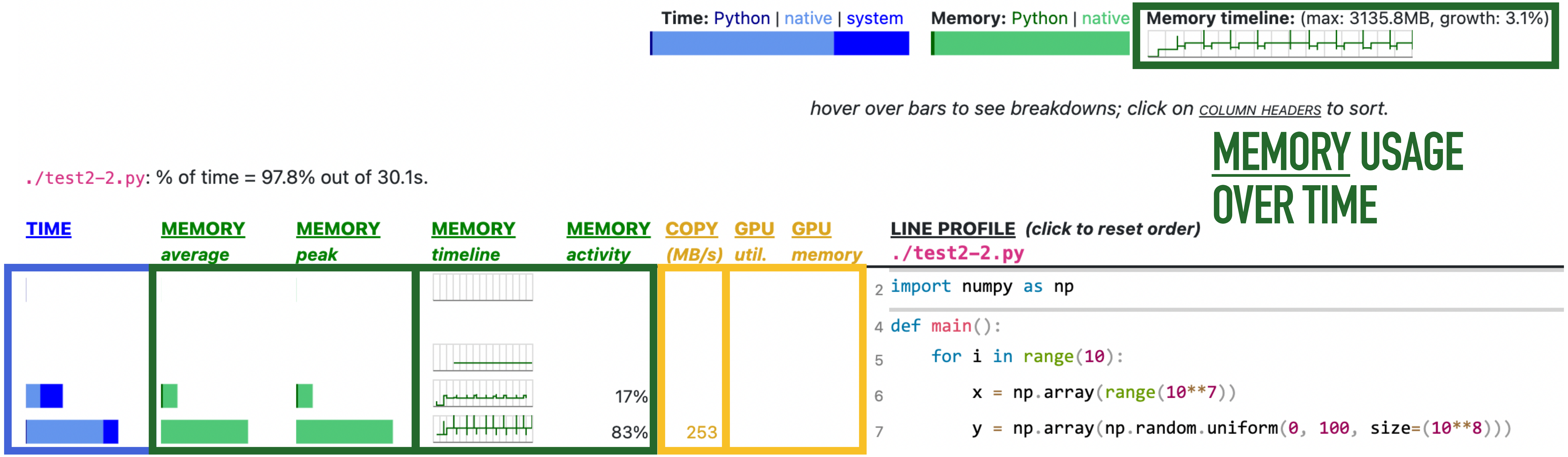
CPU
PYTHON
NATIVE
SYS%

MEMORY
PYTHON
NATIVE
AVERAGE & PEAK

MEMORY USAGE
OVER TIME, % OF MEM ALLOCATED

COPY VOLUME
(MB/s)

SCALENE



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.

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

MEMORY USAGE OVER TIME

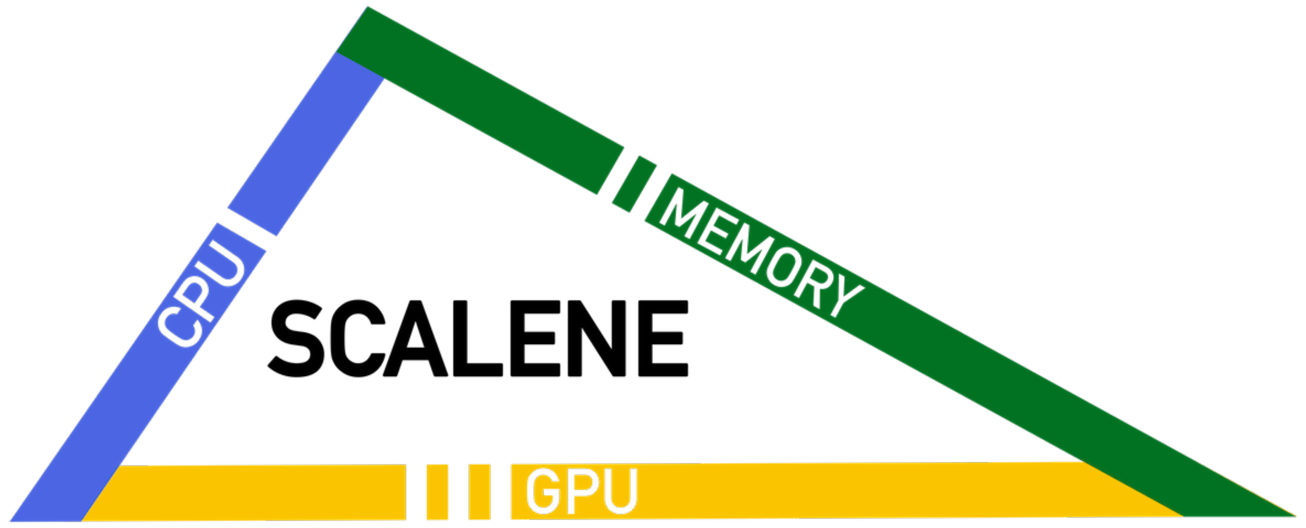
CPU
PYTHON
NATIVE
SYS%

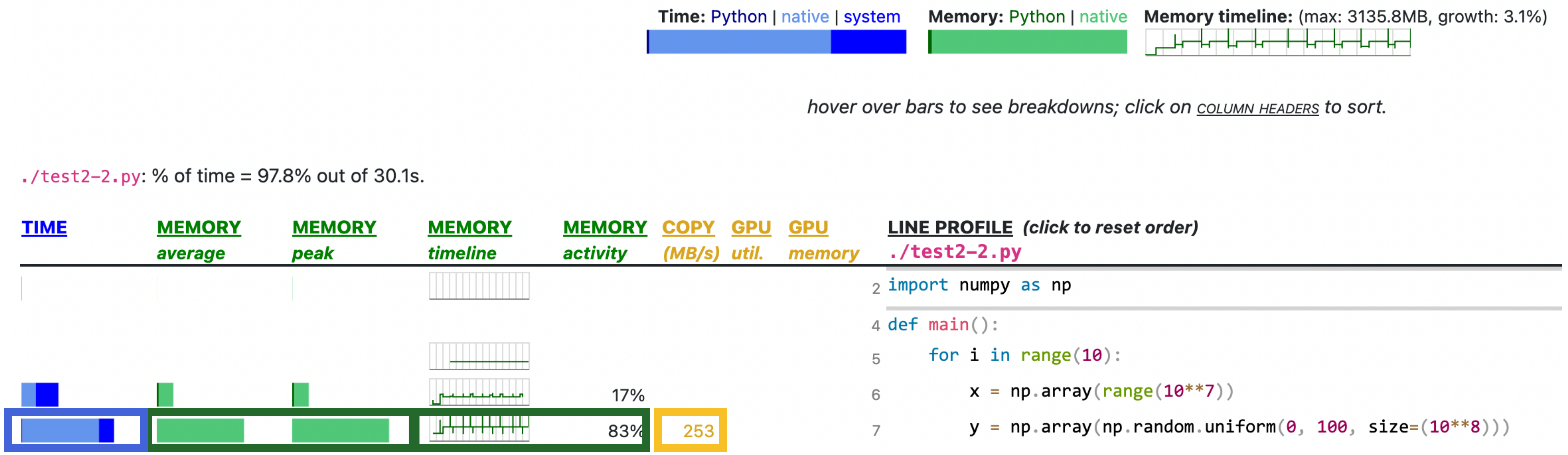
MEMORY
PYTHON
NATIVE
AVERAGE & PEAK

MEMORY USAGE
OVER TIME,
% OF MEM ALLOCATED

COPY VOLUME
(MB/s)

GPU
UTIL %,
PEAK MEMORY





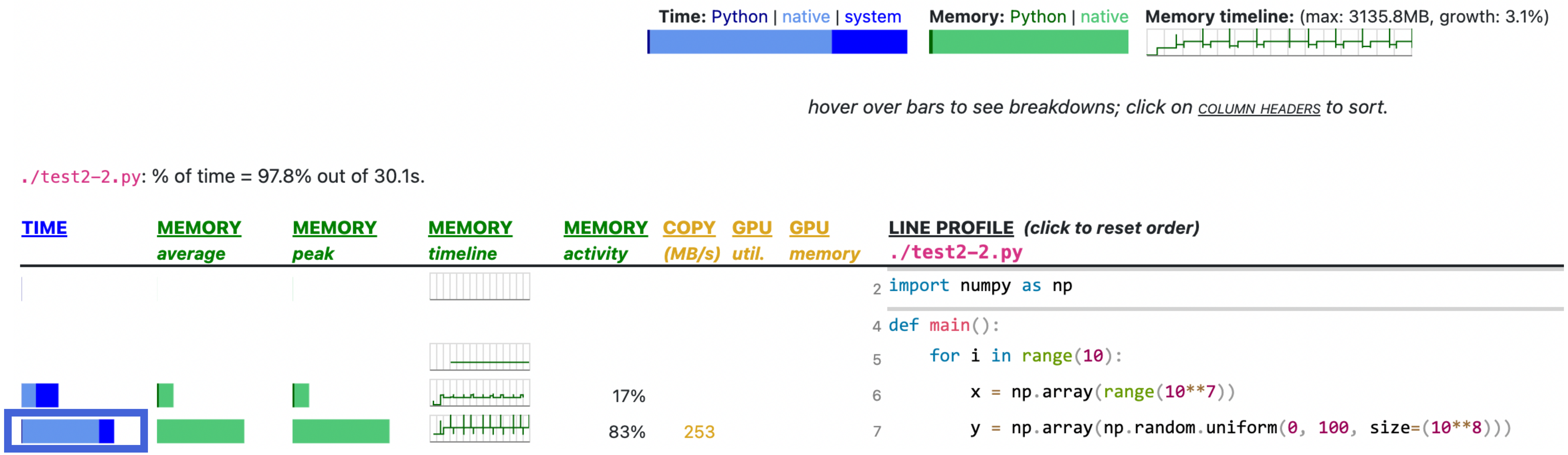
Time: Python | native | system **Memory:** Python | native **Memory timeline:** (max: 3135.8MB, growth: 3.1%)

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

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

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

SCALENE



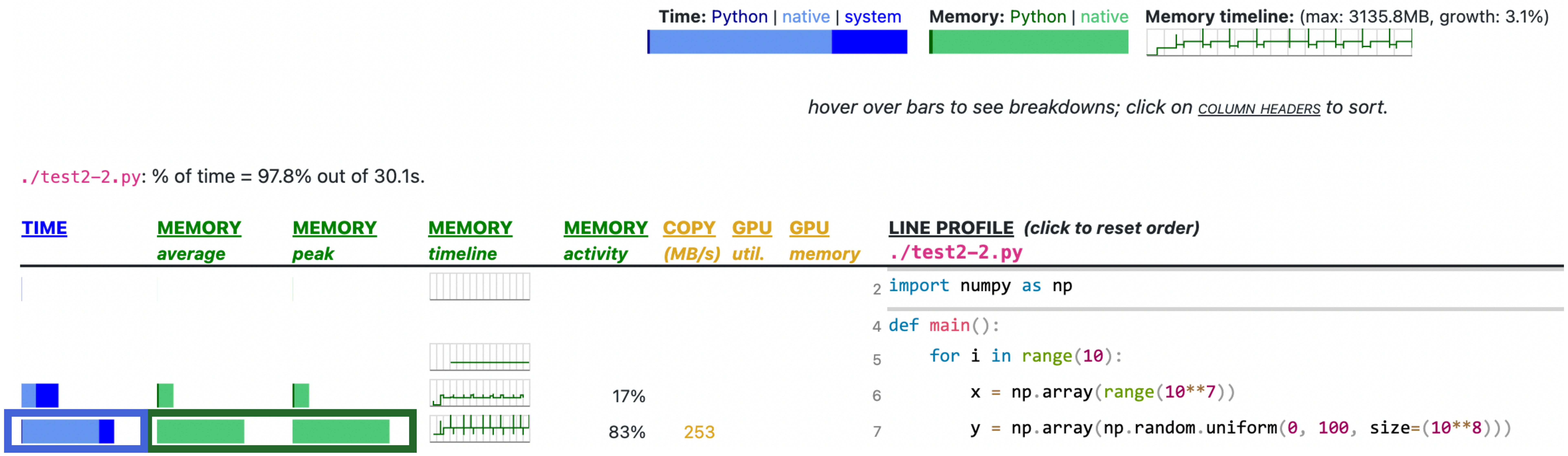
Time: Python | native | system **Memory:** Python | native **Memory timeline:** (max: 3135.8MB, growth: 3.1%)

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

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

TIME	MEMORY average	MEMORY peak	MEMORY timeline	MEMORY activity	COPY (MB/s)	GPU util.	GPU memory	LINE PROFILE (click to reset order) ./test2-2.py
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				17%				`        x = np.array(range(10**7))`
				83%	253			`        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%)

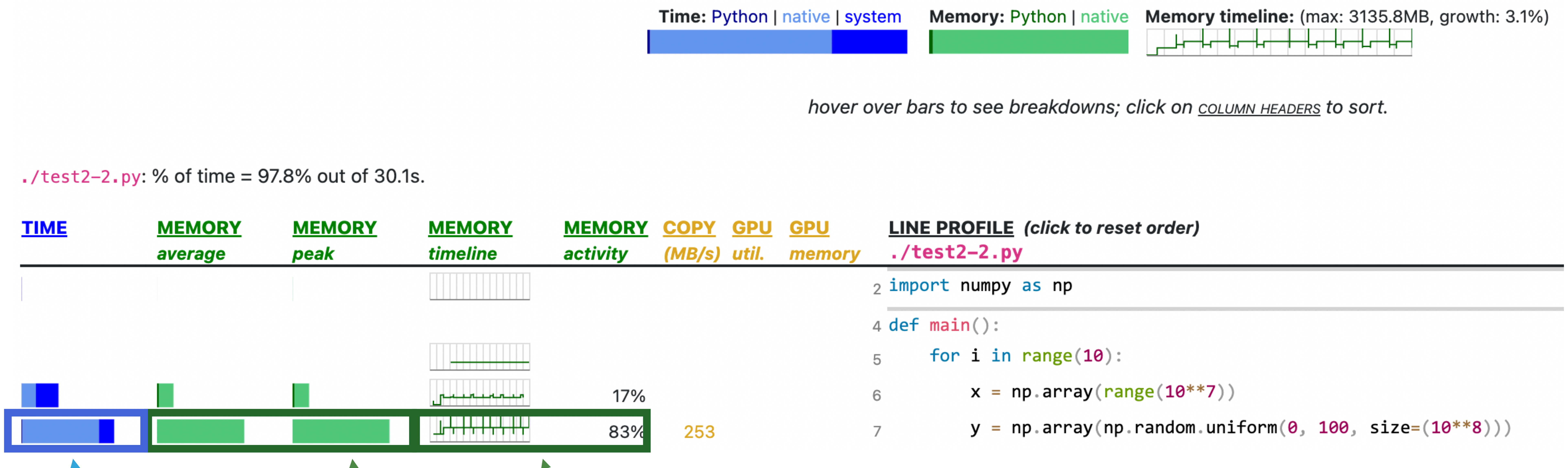
hover over bars to see breakdowns; click on COLUMN HEADERS *to sort.*

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

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*		**LINE PROFILE** *(click to reset order)* ./test2-2.py
								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.

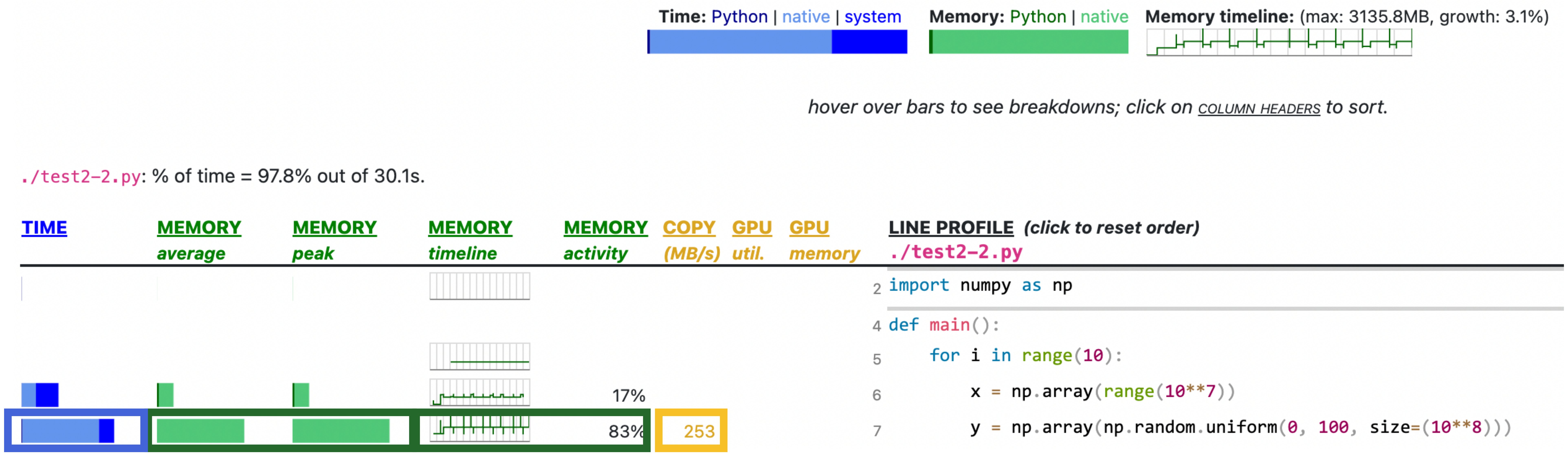
TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*	LINE PROFILE *(click to reset order)* ./test2-2.py
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				17%				`        x = np.array(range(10**7))`
				83%	253			`        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

SCALENE
CPU
MEMORY
GPU



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.

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

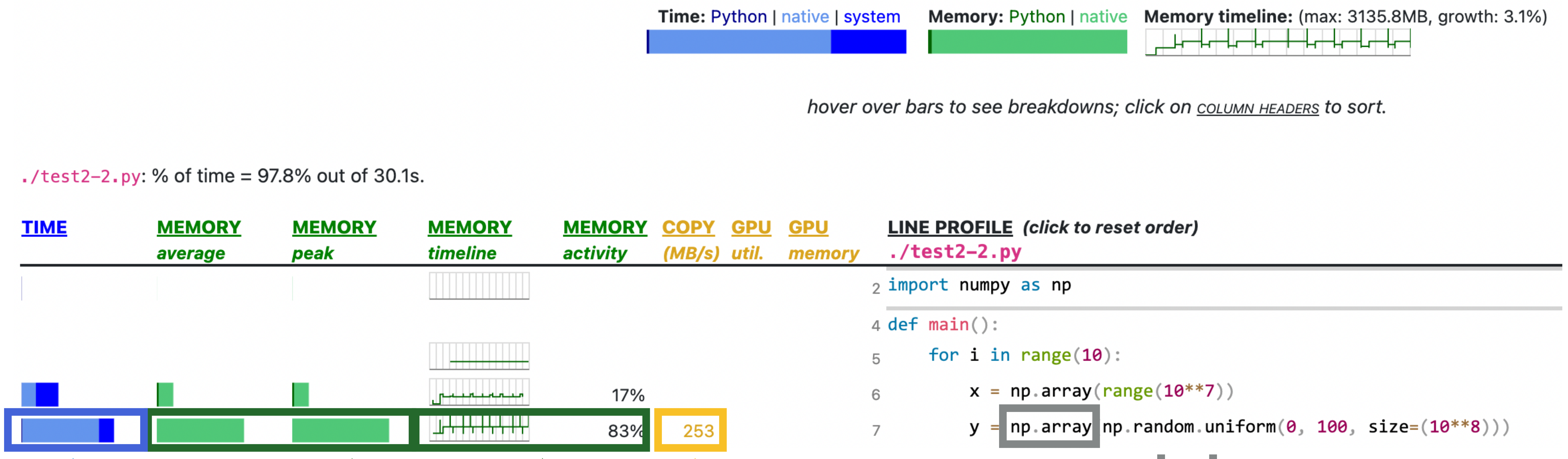
58% of runtime in native code

2GB allocated in native code

83% of memory activity "sawtooth" pattern

250MB/s copying

SCALENE



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.

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

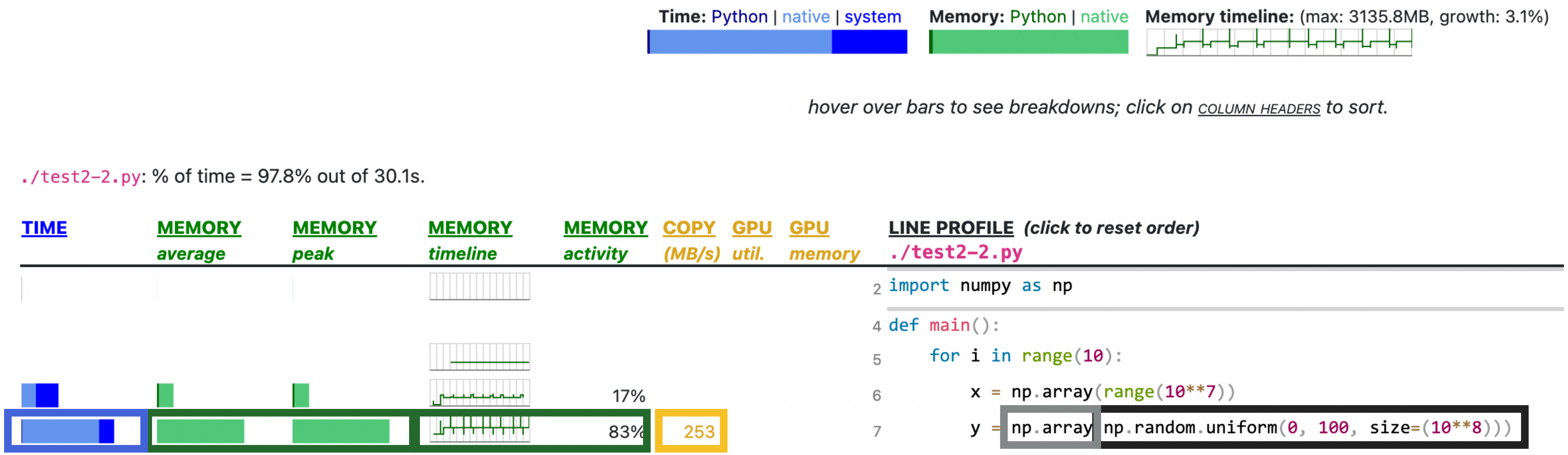
converts to numpy array

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.

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

```
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)))
```

converts to numpy array

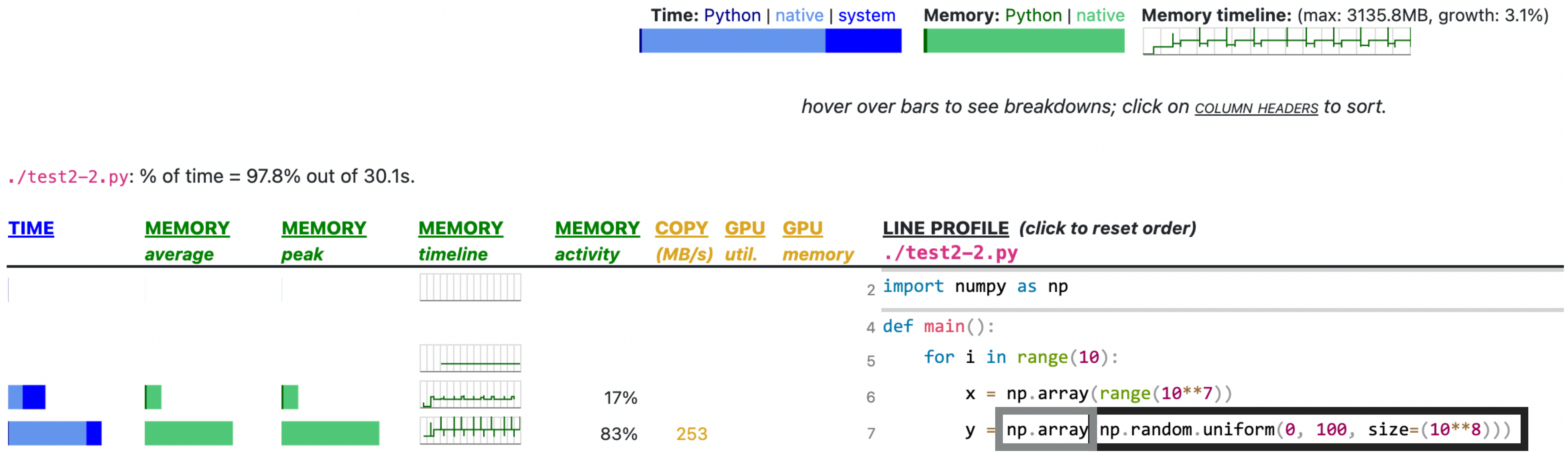
already a numpy array!

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.

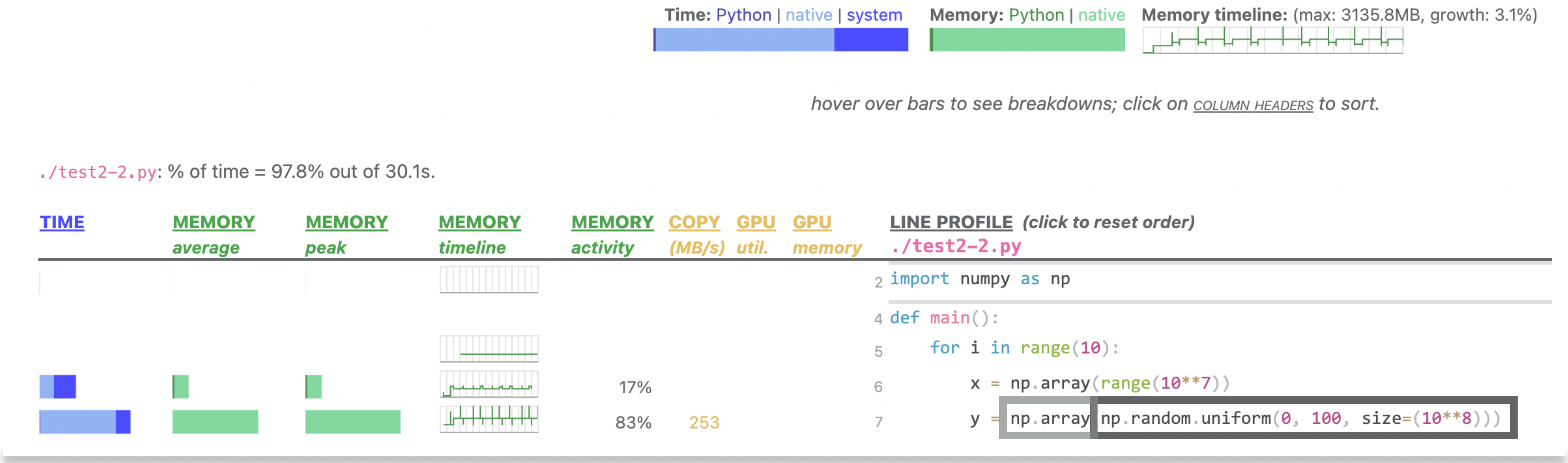
TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*	**LINE PROFILE** *(click to reset order)* `./test2-2.py`
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				17%				`        x = np.array(range(10**7))`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

CPU

MEMORY

SCALENE

GPU

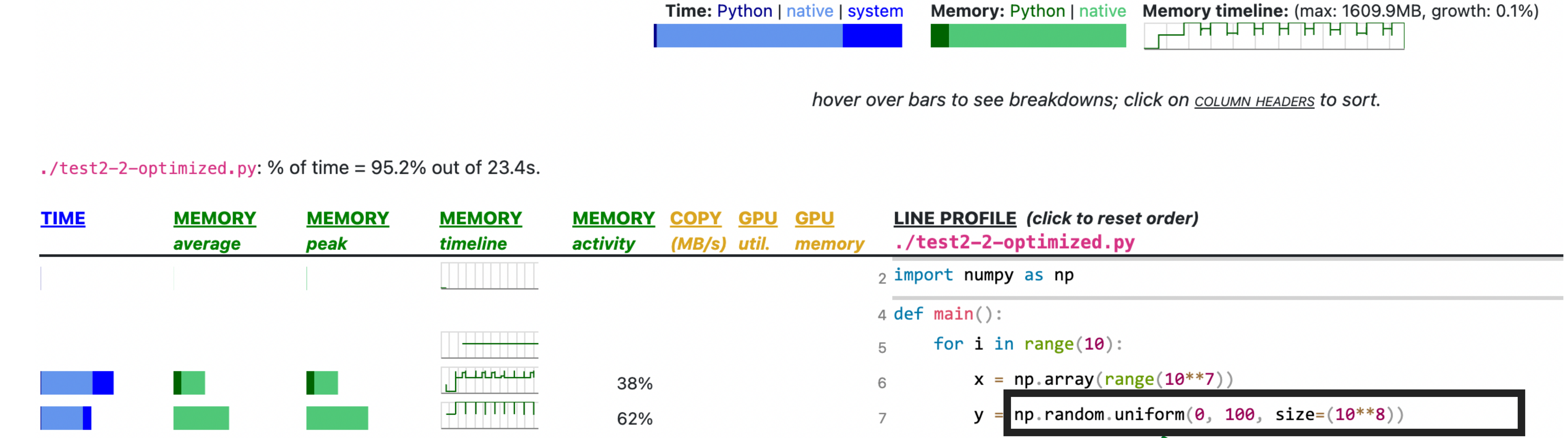


Time: Python | native | system **Memory:** Python | native **Memory timeline:** (max: 3135.8MB, growth: 3.1%)

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

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

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*		LINE PROFILE *(click to reset order)* `./test2-2.py`
								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)))`

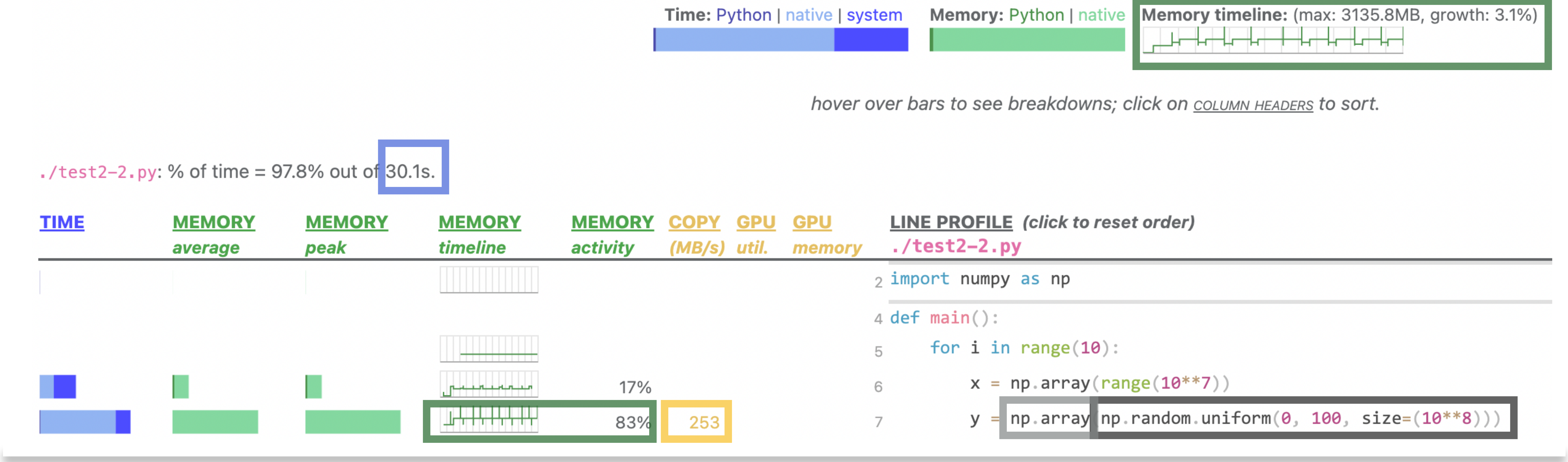


Time: Python | native | system **Memory:** Python | native **Memory timeline:** (max: 1609.9MB, growth: 0.1%)

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

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

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*		LINE PROFILE *(click to reset order)* `./test2-2-optimized.py`
								2	`import numpy as np`
								4	`def main():`
								5	`    for i in range(10):`
				38%				6	`        x = np.array(range(10**7))`
				62%				7	`        y = np.random.uniform(0, 100, size=(10**8))`



Time: Python | native | system

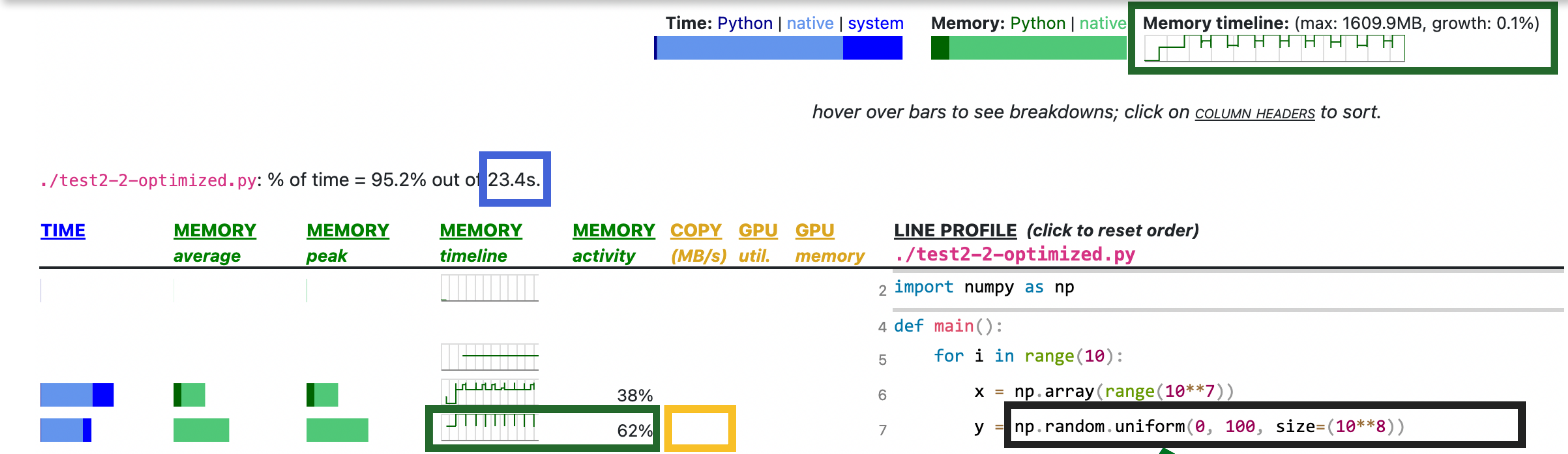
Memory: Python | native

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.

TIME	MEMORY average	MEMORY peak	MEMORY timeline	MEMORY activity	COPY (MB/s)	GPU util.	GPU memory	LINE PROFILE *(click to reset order)* ./test2-2.py
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				17%				`        x = np.array(range(10**7))`
				83%	253			`        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.

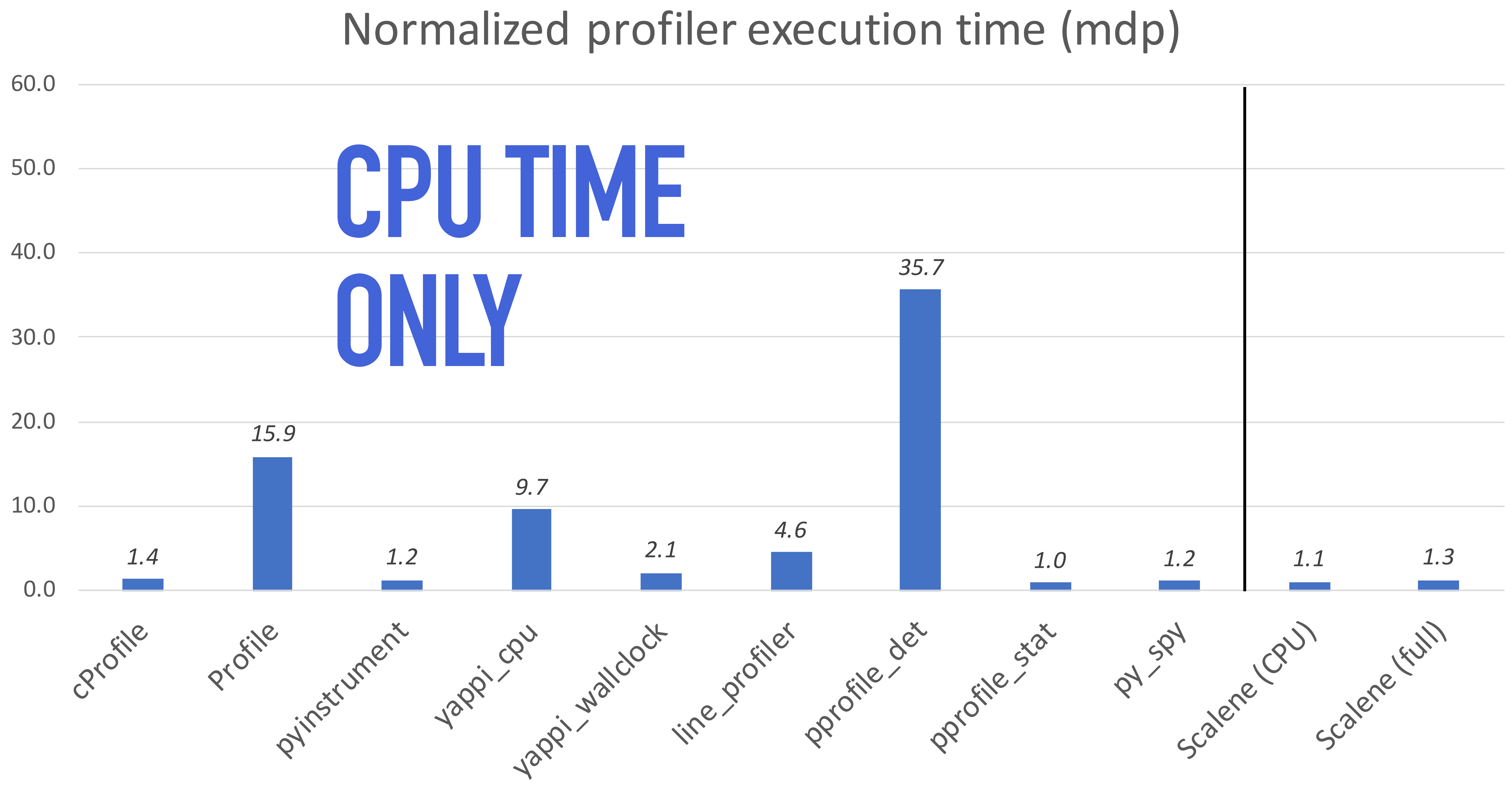
TIME	MEMORY average	MEMORY peak	MEMORY timeline	MEMORY activity	COPY (MB/s)	GPU util.	GPU memory	LINE PROFILE *(click to reset order)* ./test2-2-optimized.py
								`import numpy as np`
								`def main():`
								`    for i in range(10):`
				38%				`        x = np.array(range(10**7))`
				62%				`        y = np.random.uniform(0, 100, size=(10**8))`

CPU

MEMORY

SCALENE

GPU



memory_profiler: ~300X slower
(from 5s to 20 minutes!)

Time: Python | native | system

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

NATIVE VS.
PYTHON TIME

LOW-OVERHEAD
MEMORY PROFILING

python™

SAMPLING
-BASED
PROFILING

SAMPLING -BASED PROFILING

foo

function	time
foo	**1**

SAMPLING-BASED PROFILING

python™ python™

foo

bar

function	time
foo	1
bar	**1**

SAMPLING-BASED PROFILING

foo

bar

bar

function	time
foo	1
bar	**2**

SAMPLING -BASED PROFILING

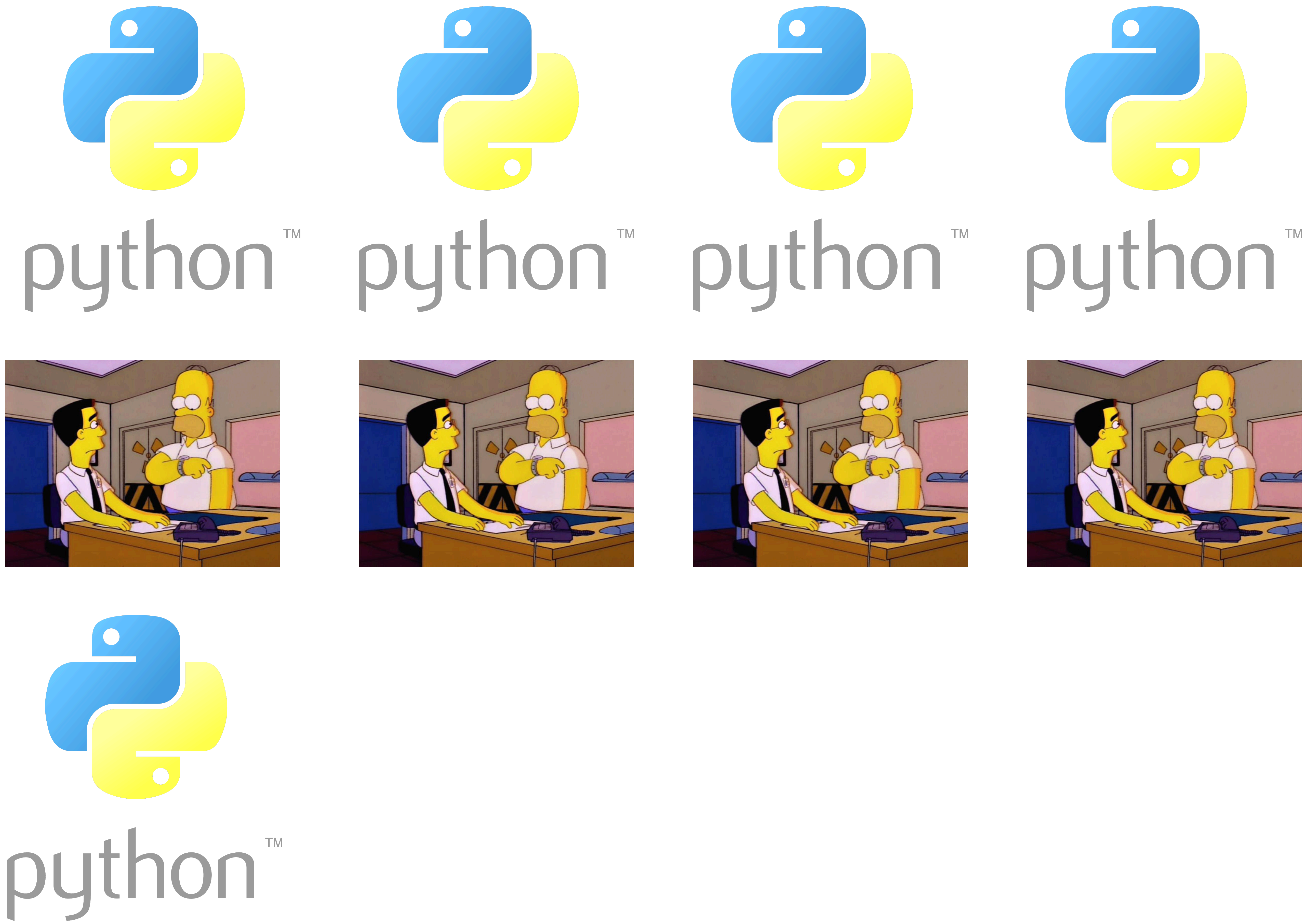
foo

bar

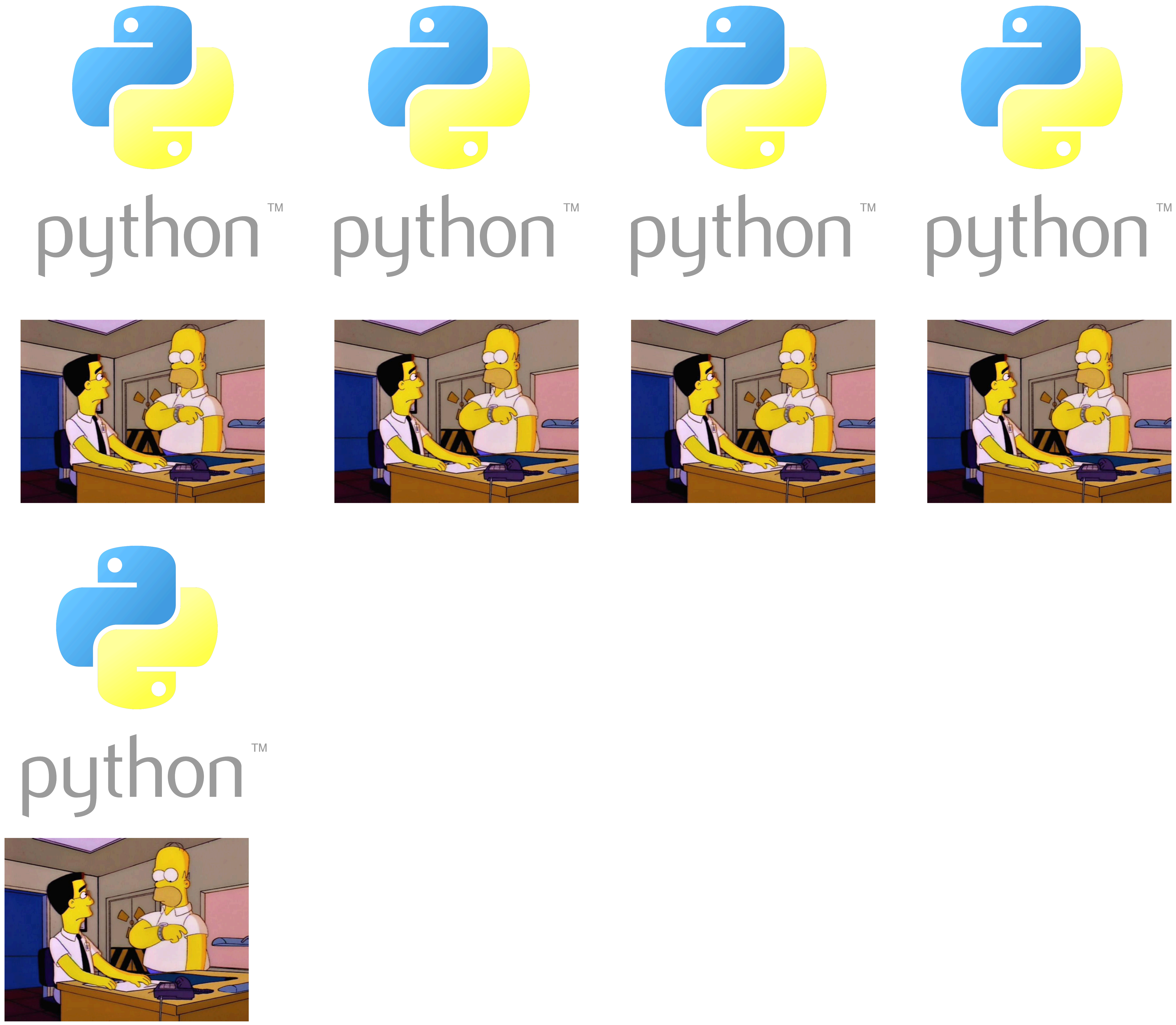
bar

foo

function	time
foo	2
bar	2



DEFERRED SIGNAL DELIVERY



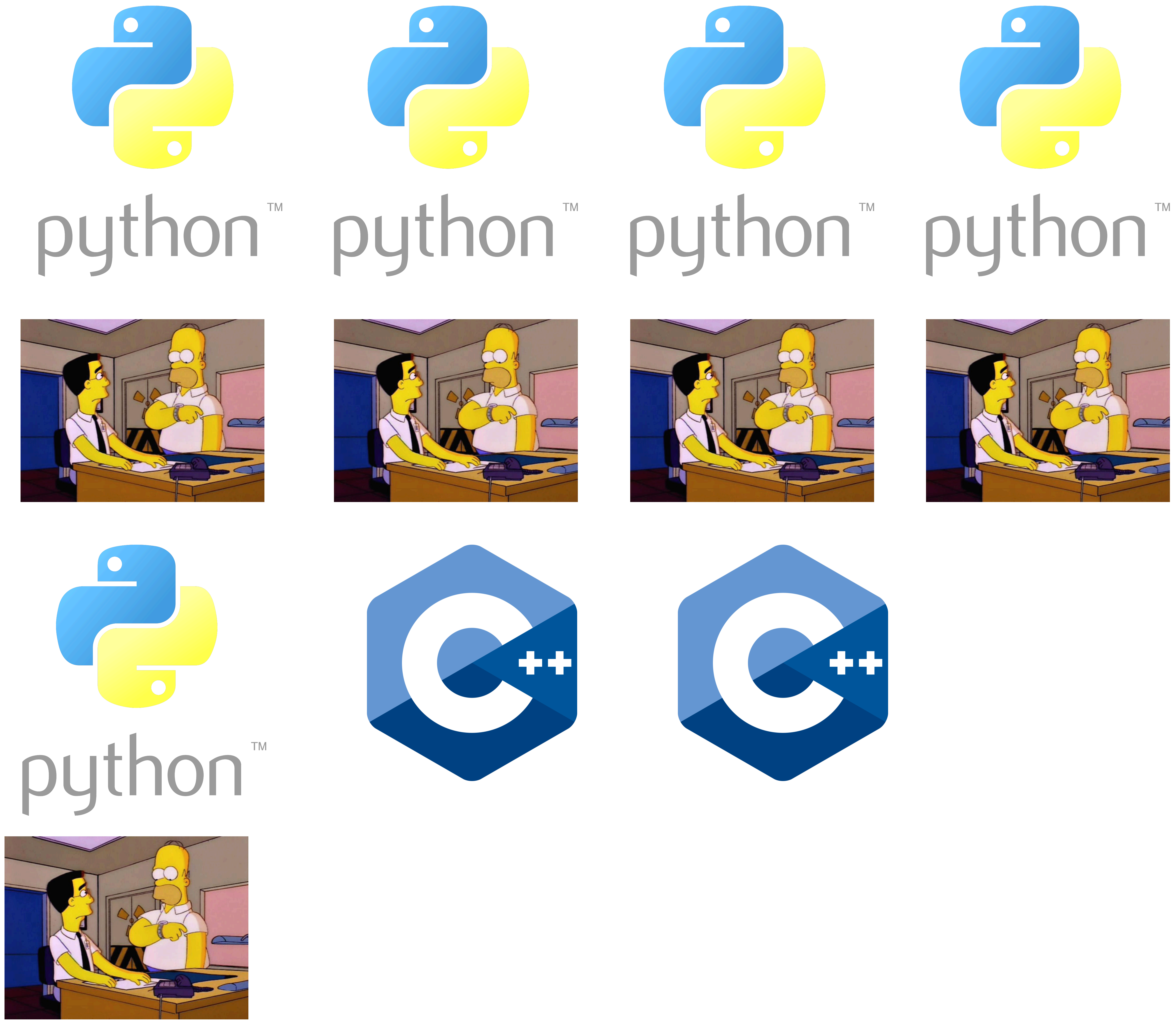
DEFERRED SIGNAL DELIVERY

function	time
foo	**1**

python™
python™
python™
python™
python™

DEFERRED SIGNAL DELIVERY

function	time
foo	1



DEFERRED SIGNAL DELIVERY

function	time
foo	1

python
python
python
python
python
python

DEFERRED SIGNAL DELIVERY

function	time
foo	2

DEFERRED SIGNAL DELIVERY

function	time
foo	2

DEFERRED
SIGNAL
DELIVERY

(pprofile)

function	time
foo	2

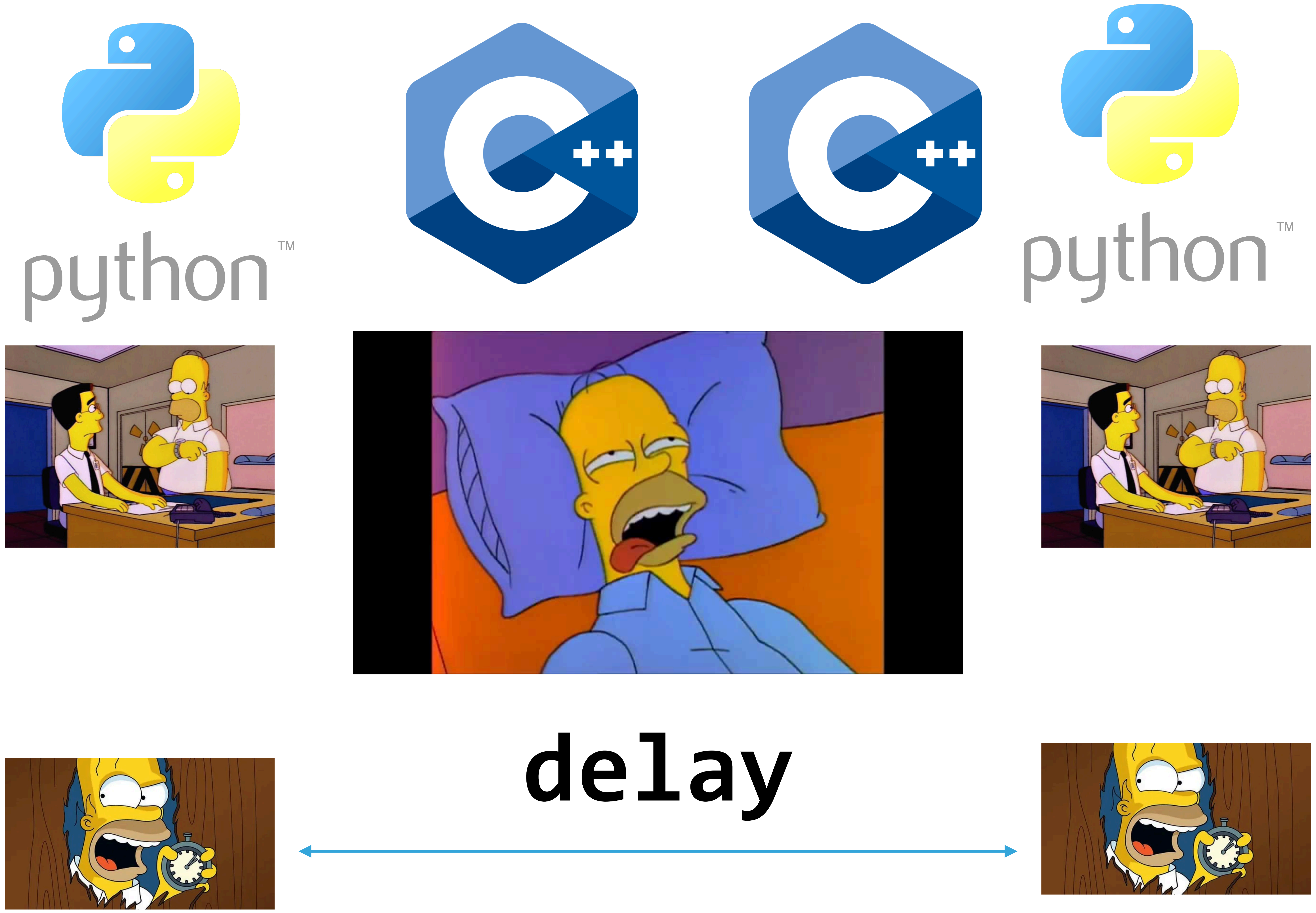
(VIRTUAL TIME)

INFERRING EXECUTION TIME

delay

(VIRTUAL TIME)

INFERRING EXECUTION TIME



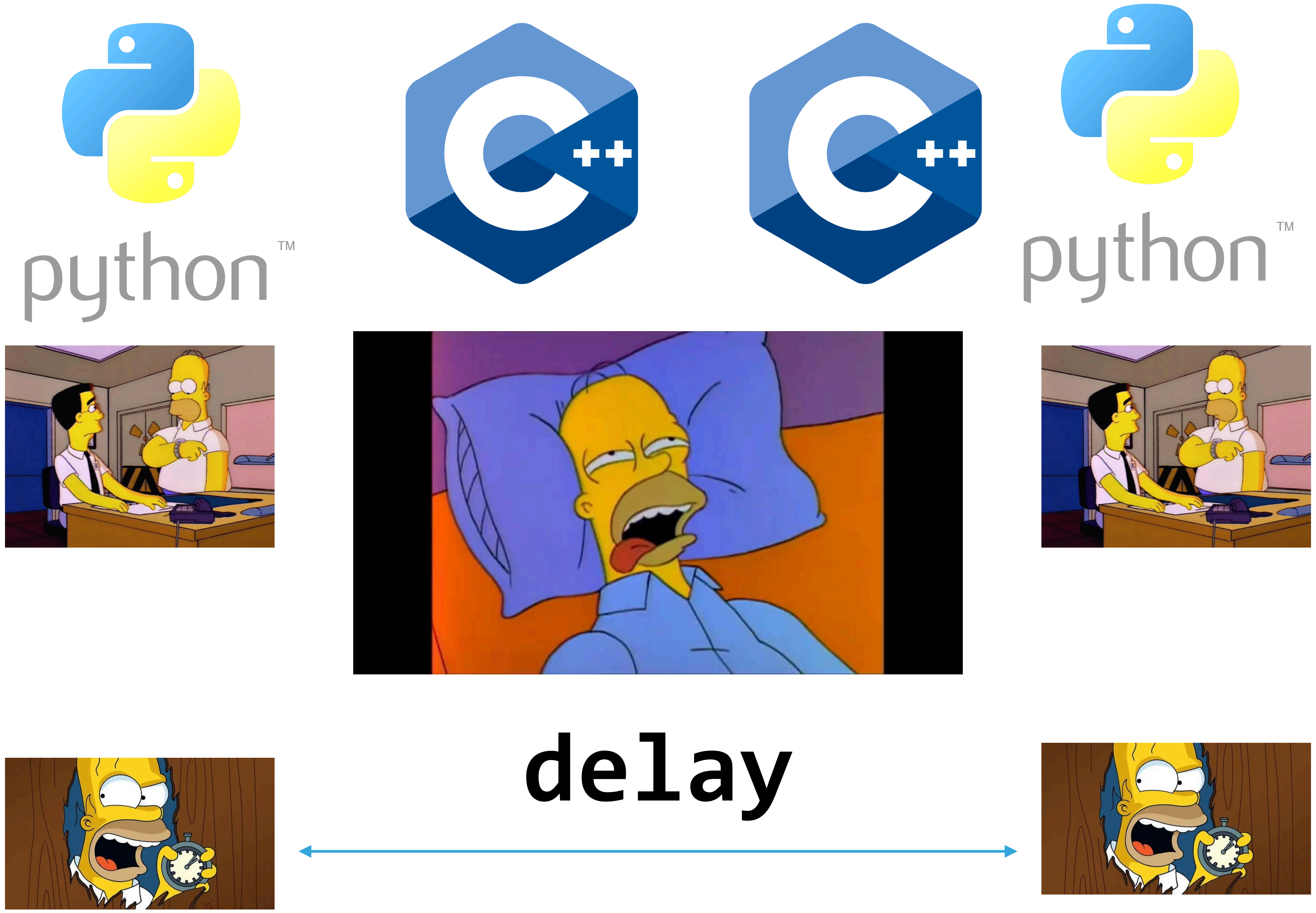
delay

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

function	time
foo	2
bar	2

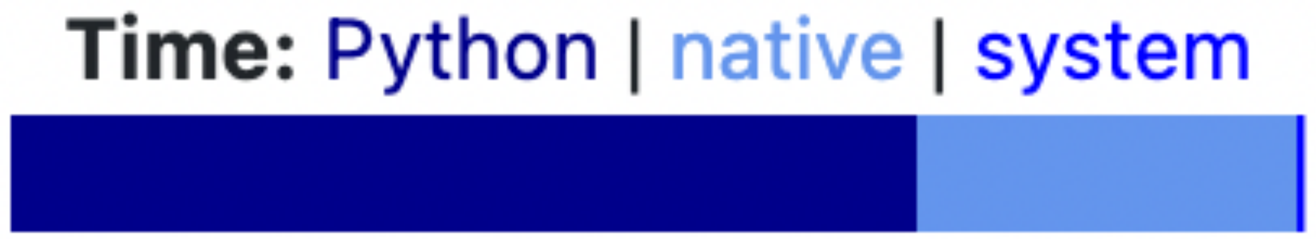
(VIRTUAL TIME)

INFERRING EXECUTION TIME



delay

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



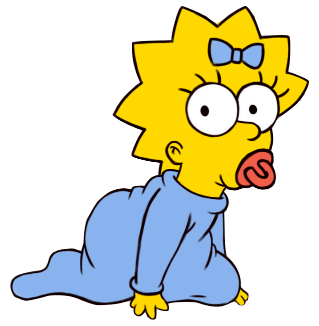
Time: Python | native | system

function	time
foo	2
bar	2

memory_profiler:

memory_profiler: ~300x slower

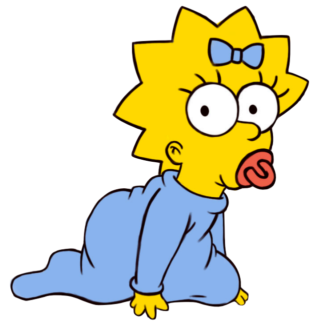
memory_profiler: ~300x slower



record `malloc` info

tracks every `malloc/free`

memory_profiler: ~300x slower

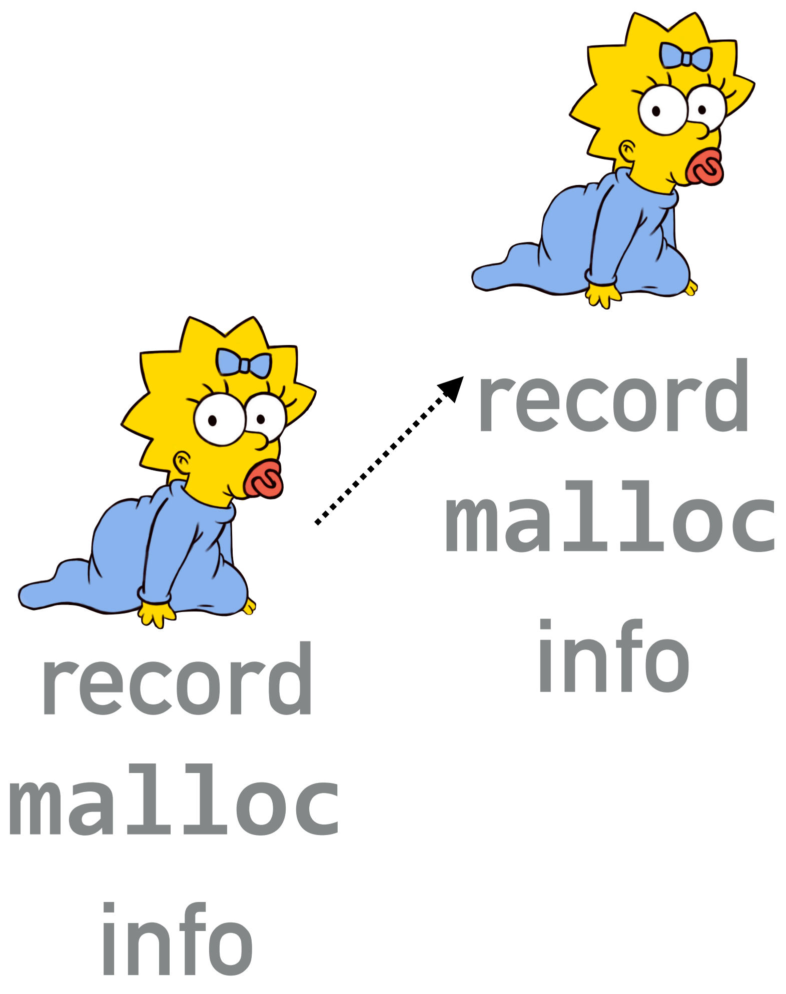


record `malloc` info

tracks every `malloc/free`

invokes `getrusage!`

memory_profiler: ~300x slower

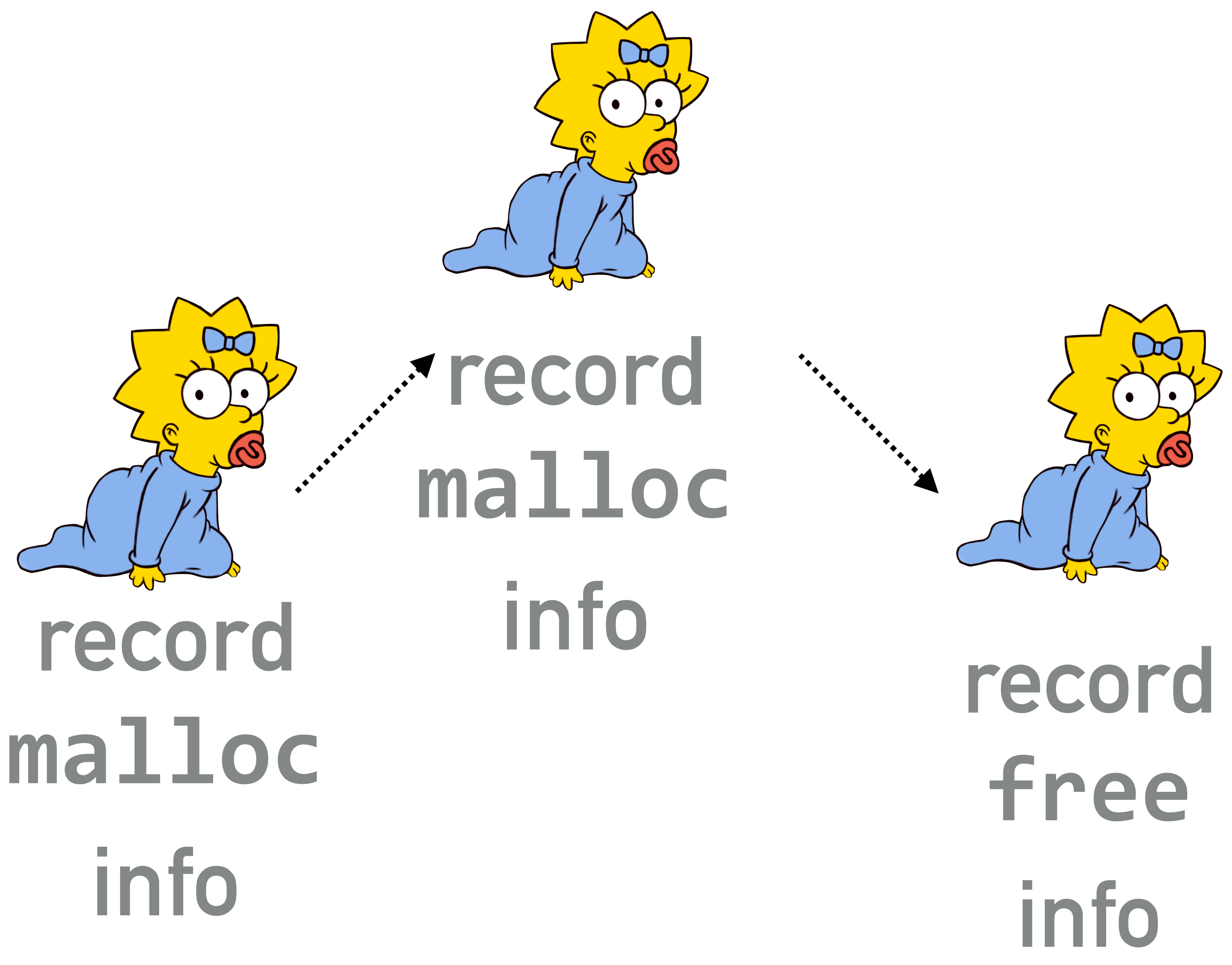


record `malloc` info

record `malloc` info

tracks every `malloc/free`

memory_profiler: ~300x slower



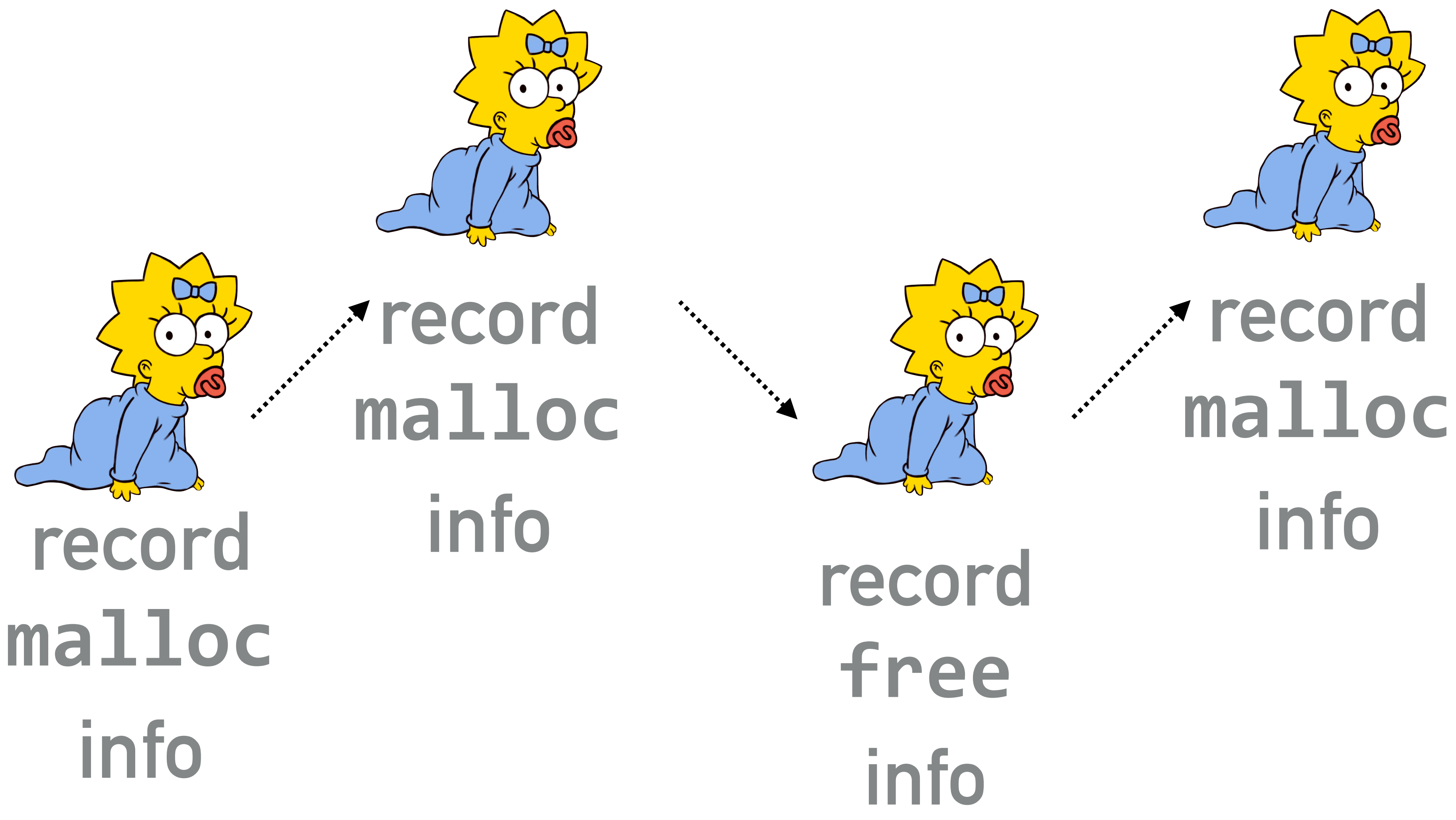
record `malloc` info

record `malloc` info

record `free` info

tracks every `malloc`/`free`

memory_profiler: ~300x slower



record `malloc` info

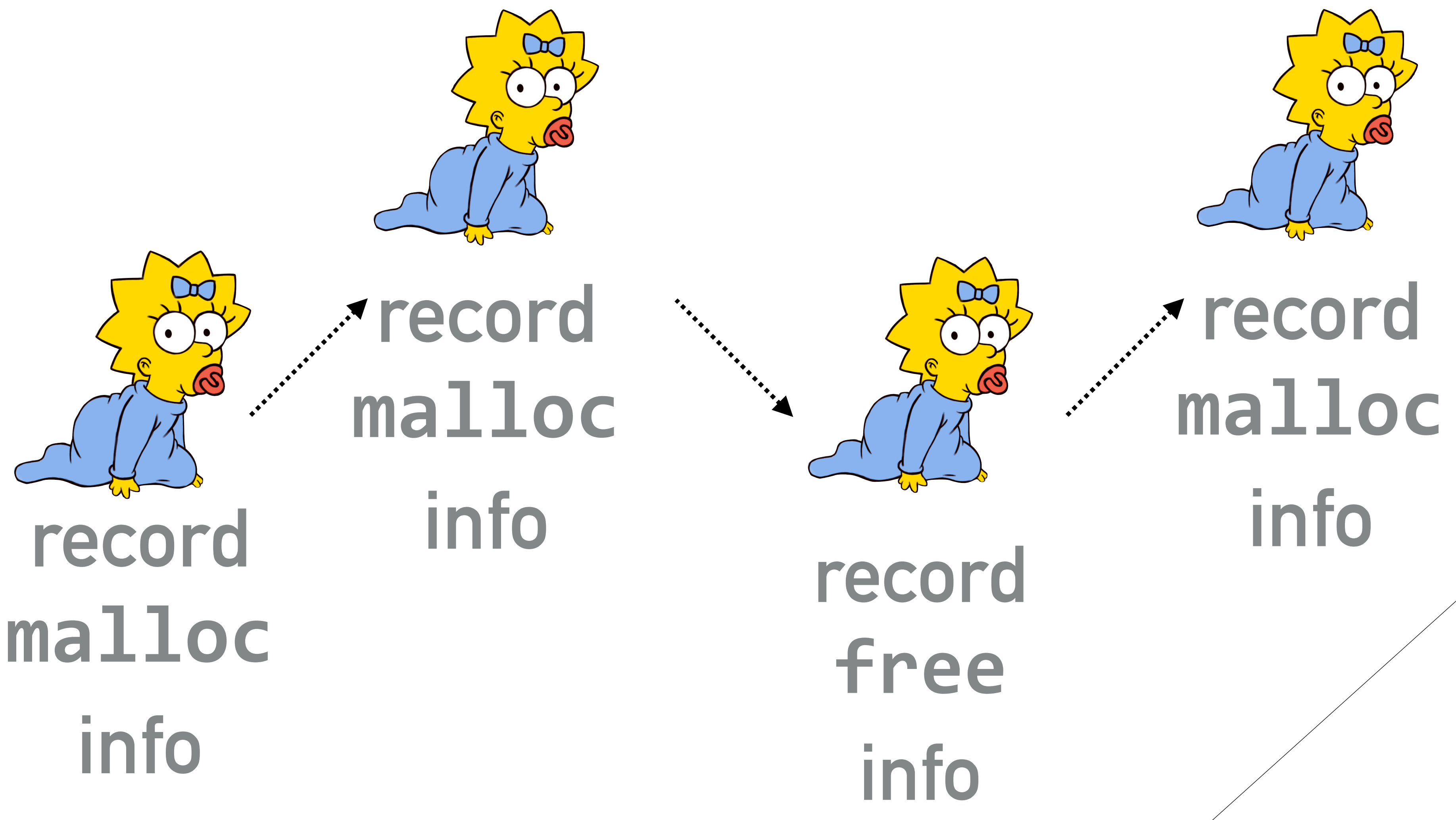
record `malloc` info

record `free` info

record `malloc` info

tracks every `malloc/free`

memory_profiler: ~300x slower



record malloc info

record malloc info

record free info

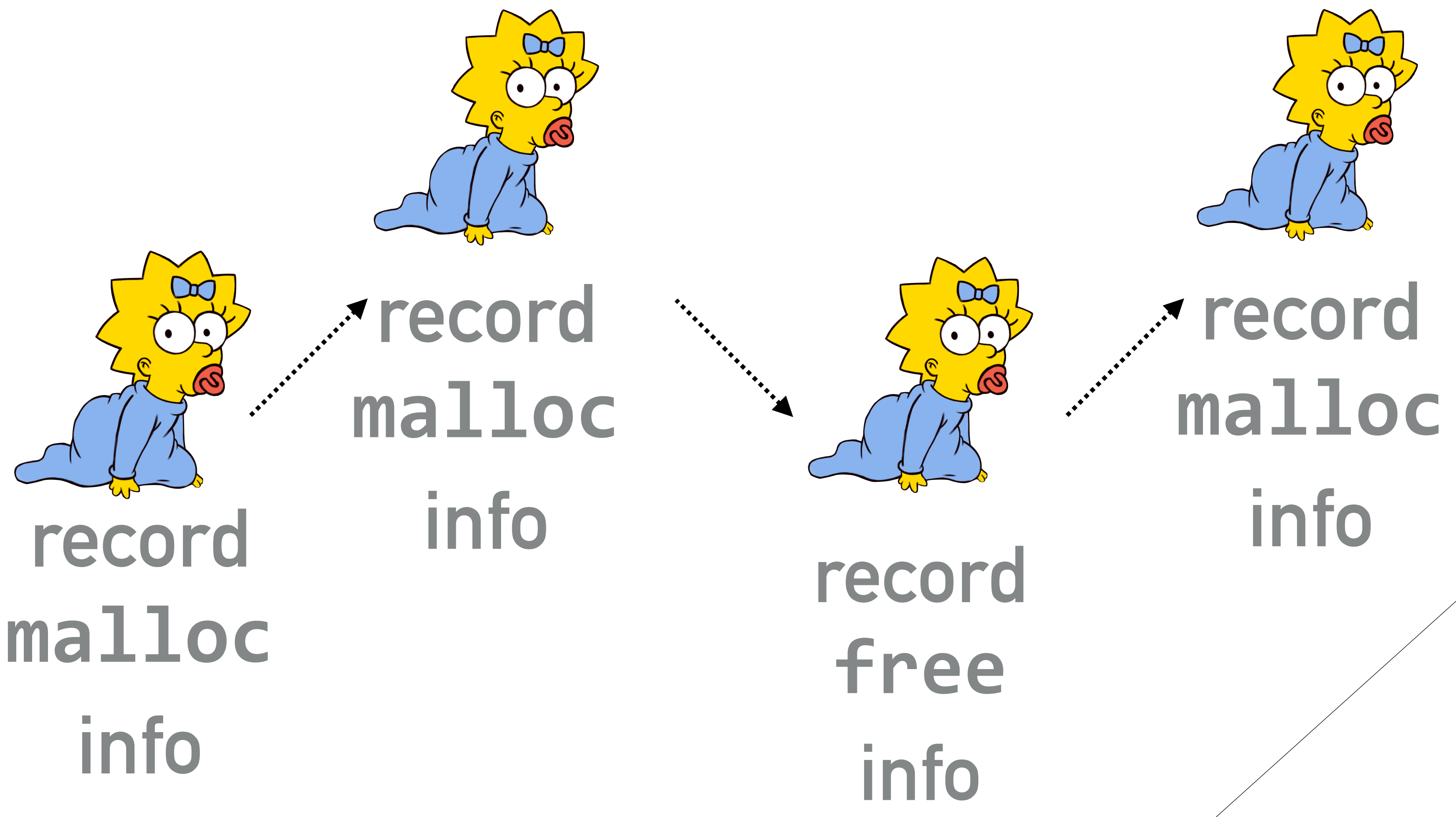
record malloc info

tracks every malloc/free

SCALENE

threshold-based sampling

memory_profiler: ~300x slower



record `malloc` info

record `malloc` info

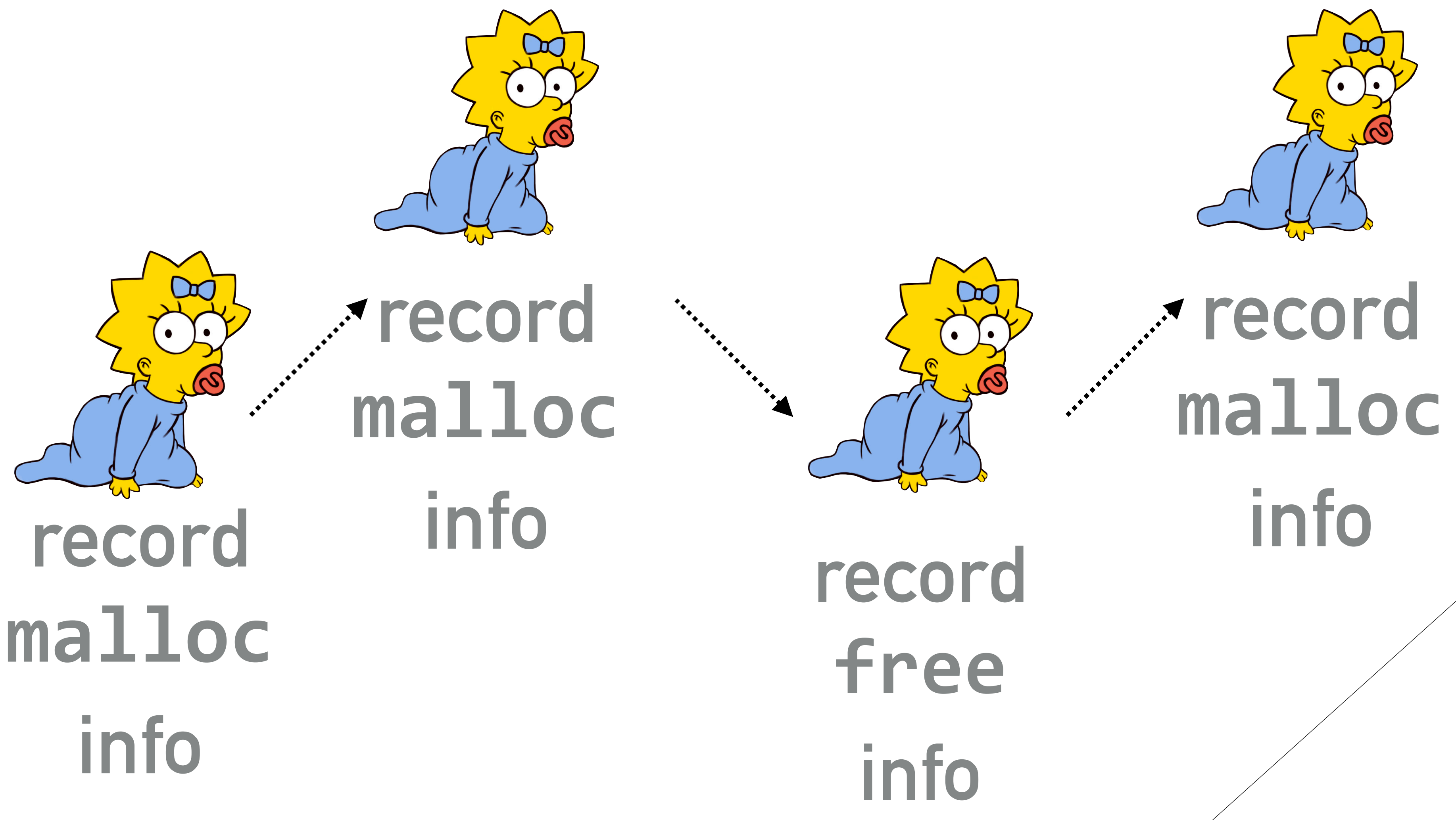
record `free` info

record `malloc` info

tracks every `malloc/free`

<u>threshold-based</u> sampling

memory_profiler: ~300x slower



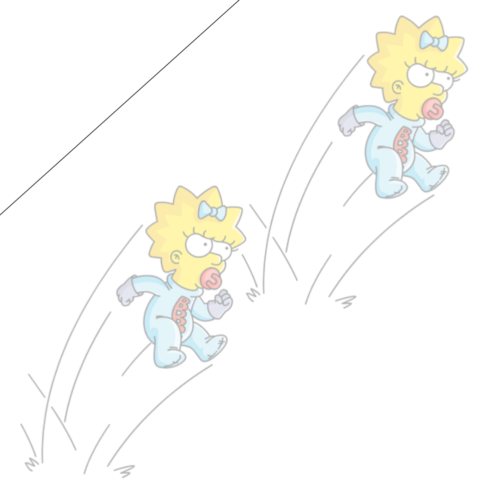
record malloc info

record malloc info

record free info

record malloc info

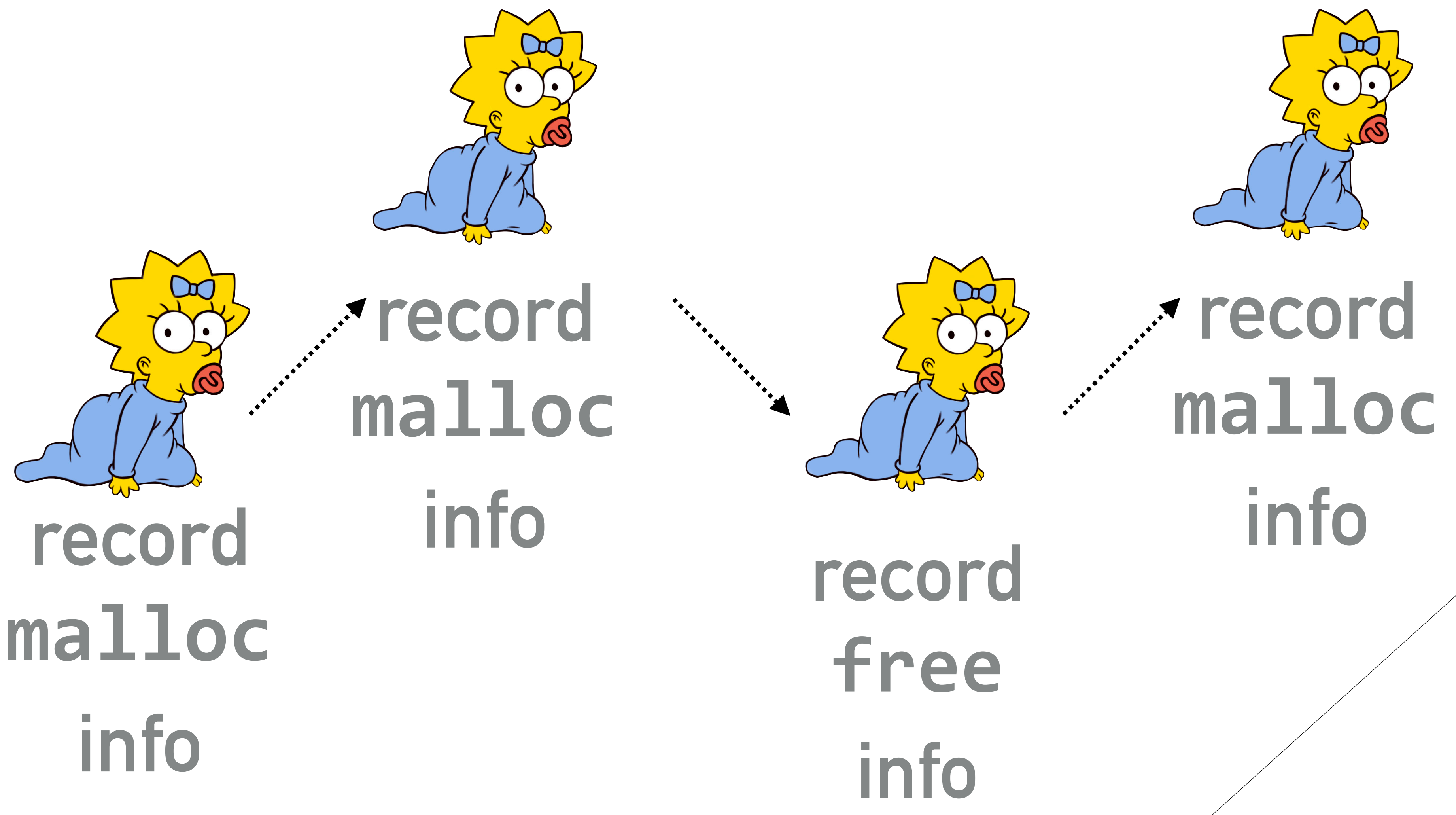
tracks every malloc/free



SCALENE

threshold-based sampling

memory_profiler: ~300x slower



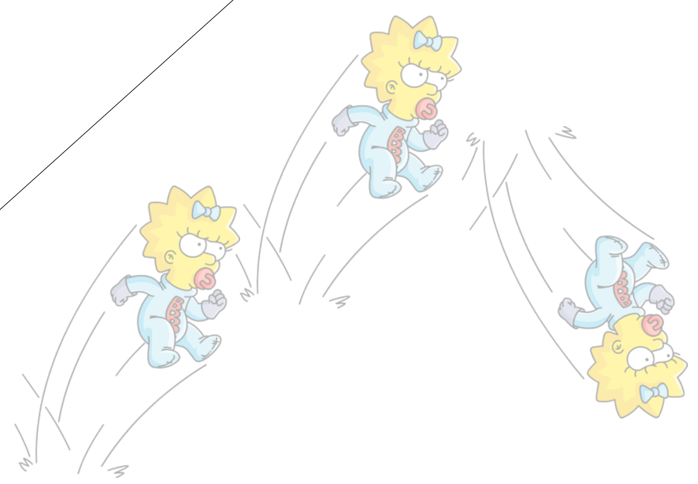
record malloc info

record malloc info

record free info

record malloc info

tracks every malloc/free

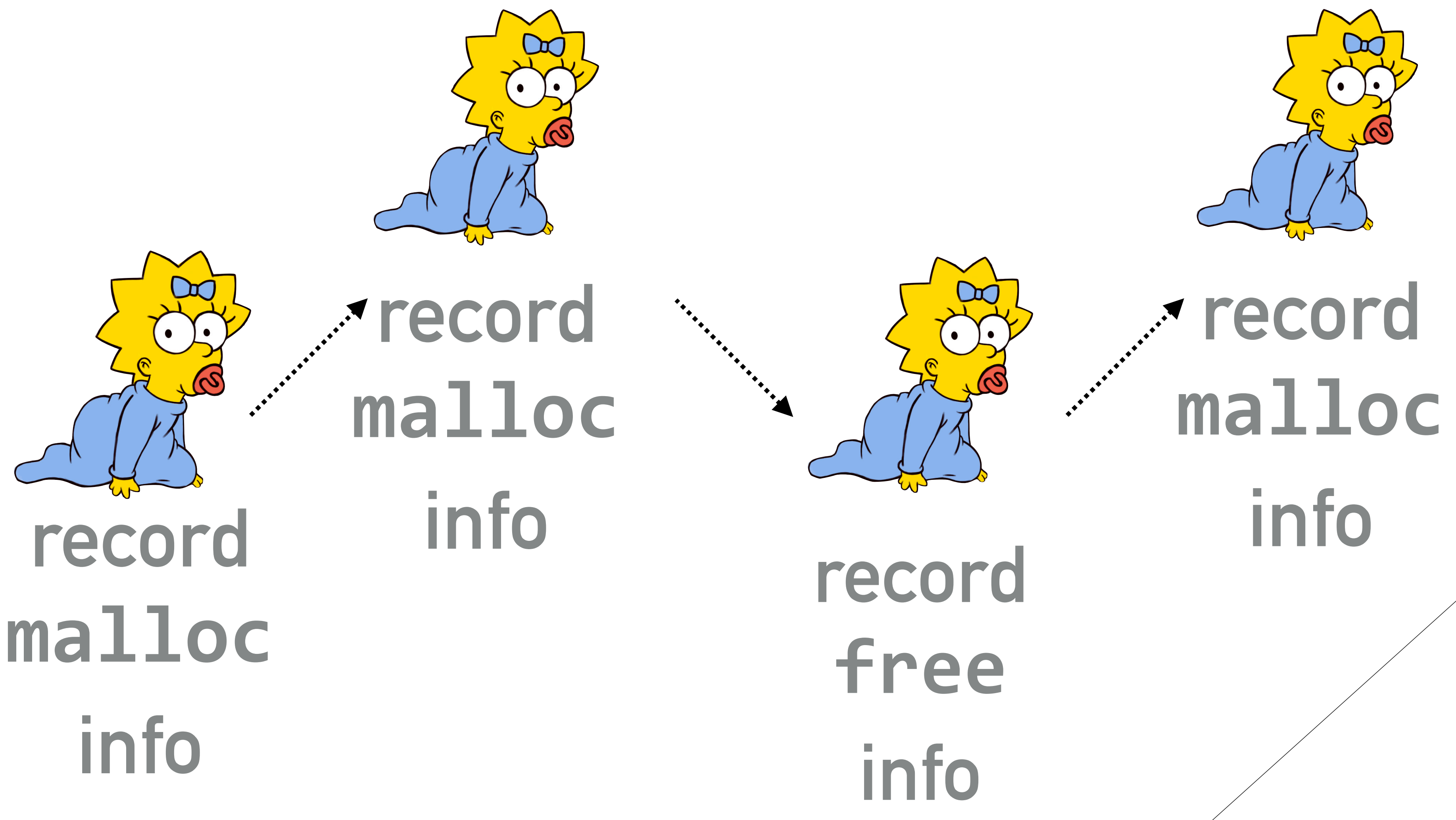


SCALENE

CPU MEMORY GPU

threshold-based sampling

memory_profiler: ~300x slower



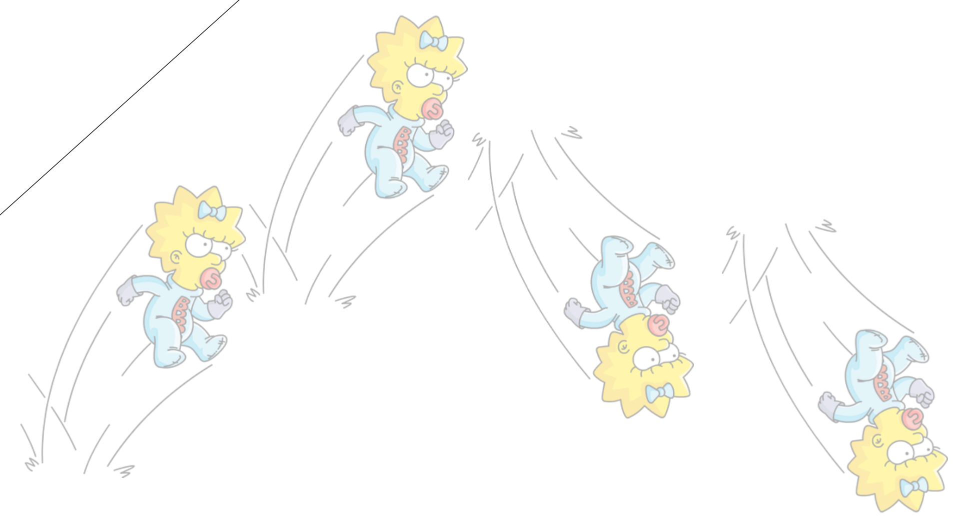
record malloc info

record malloc info

record free info

record malloc info

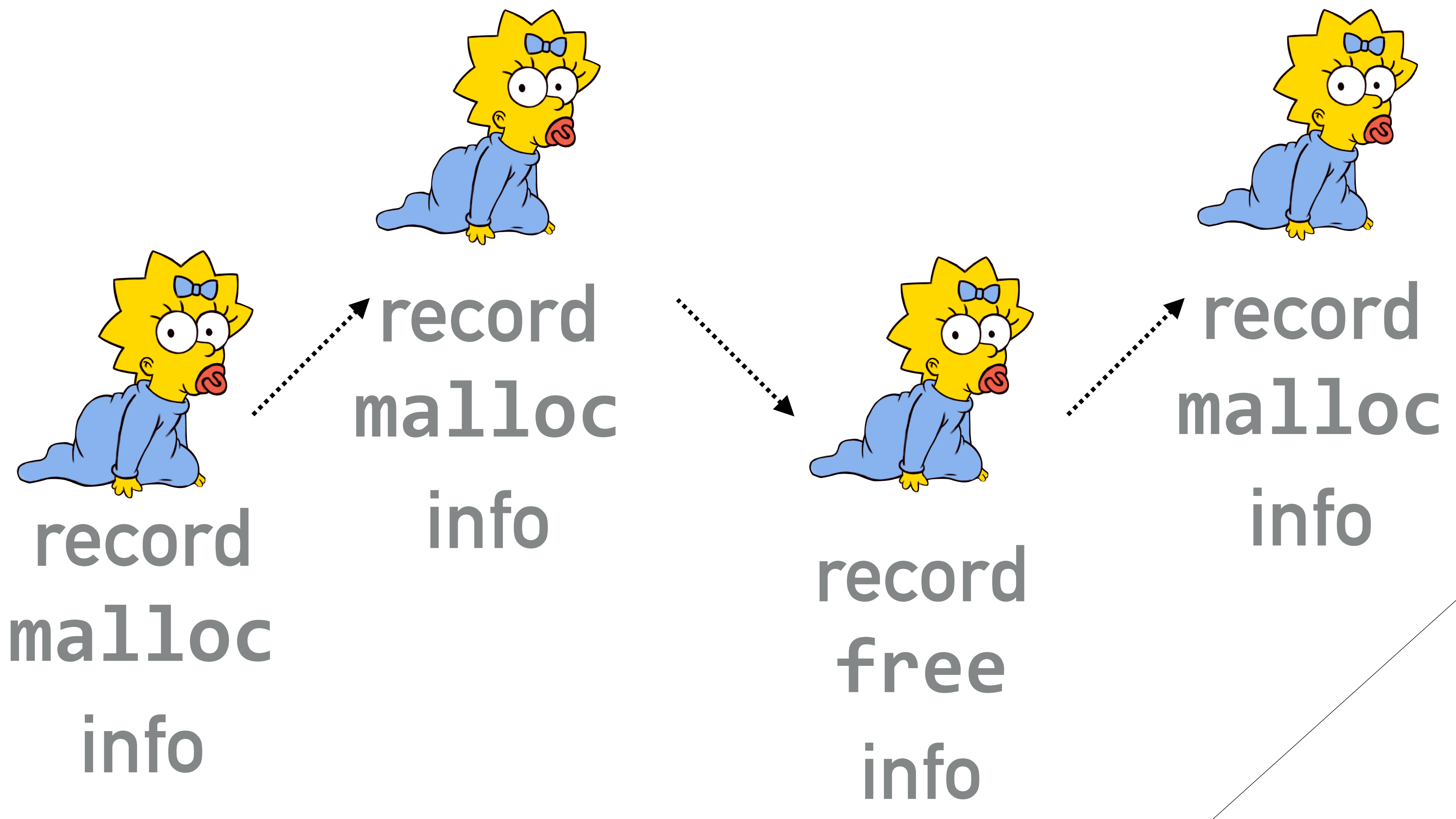
tracks every malloc/free



SCALENE

threshold-based sampling

memory_profiler: ~300x slower



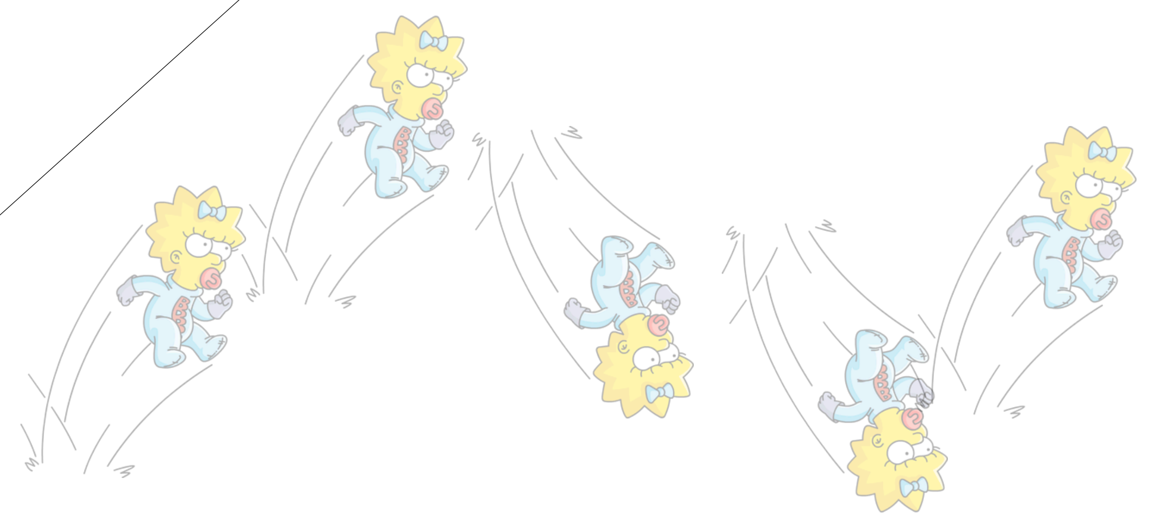
record malloc info

record malloc info

record free info

record malloc info

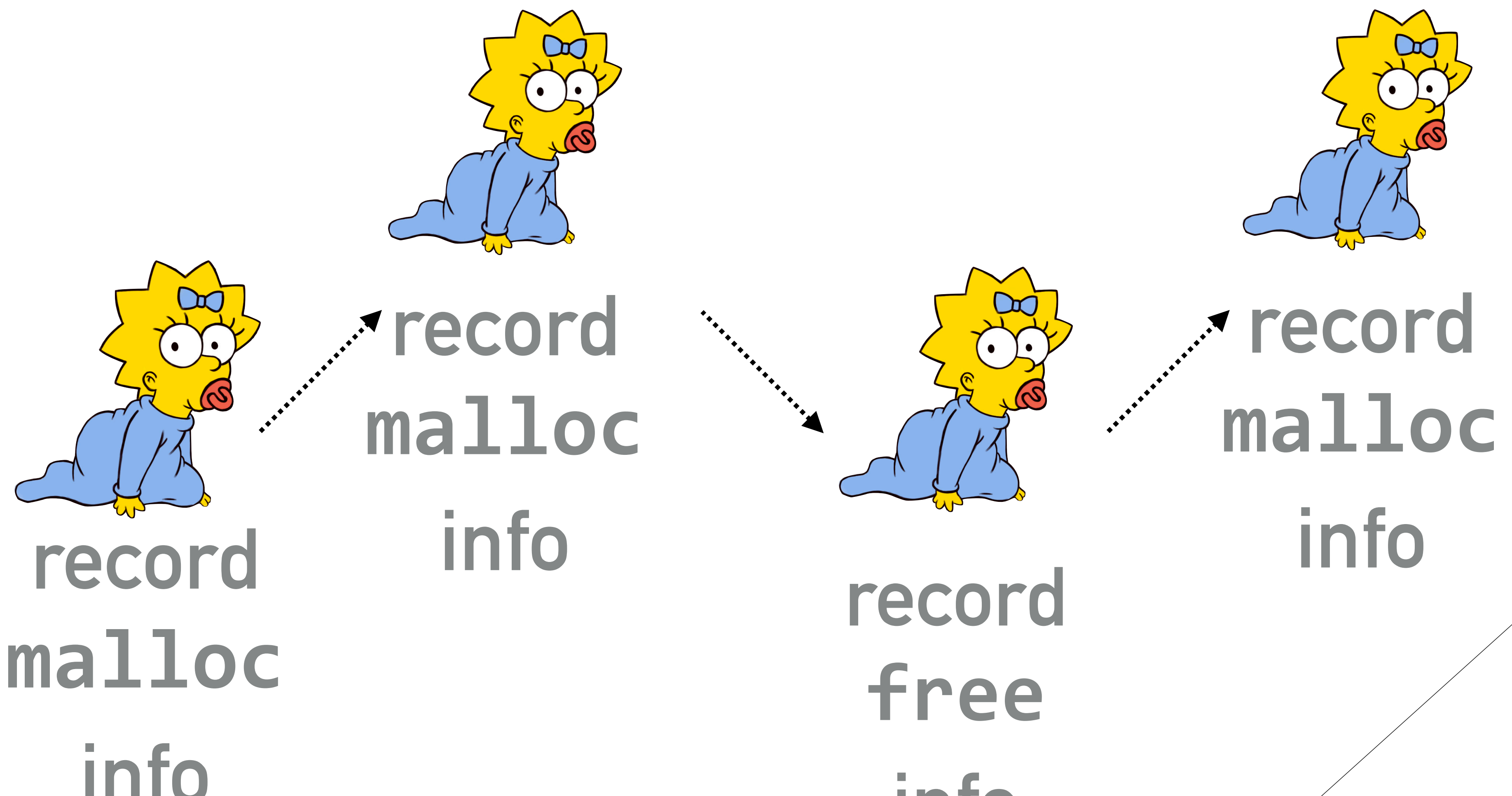
tracks every malloc/free



SCALENE

threshold-based sampling

memory_profiler: ~300x slower



record malloc info

record malloc info

record free info

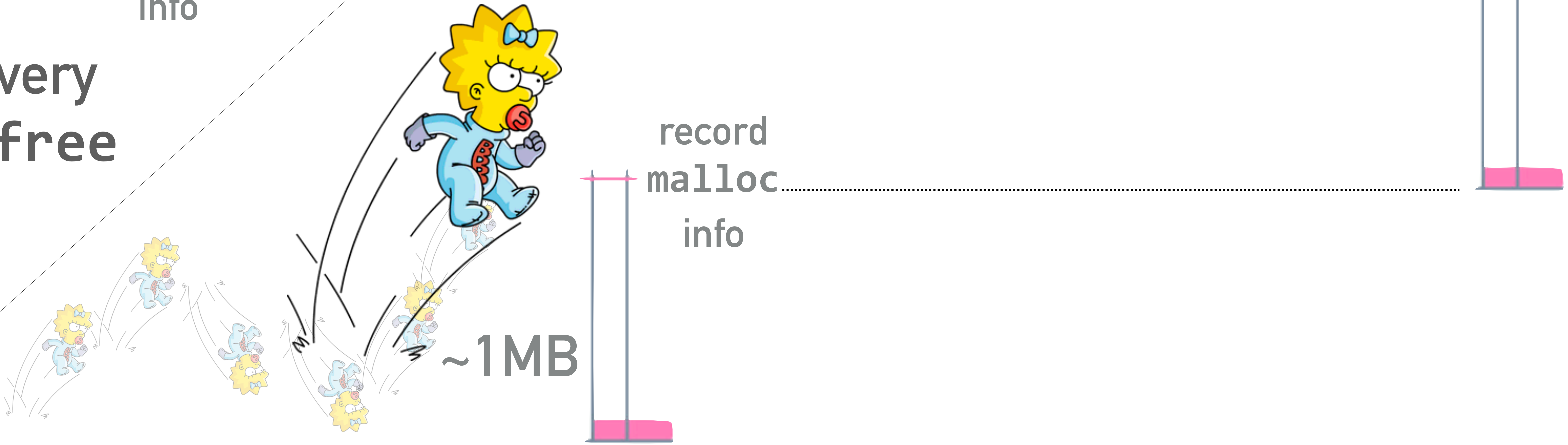
record malloc info

tracks every `malloc/free`

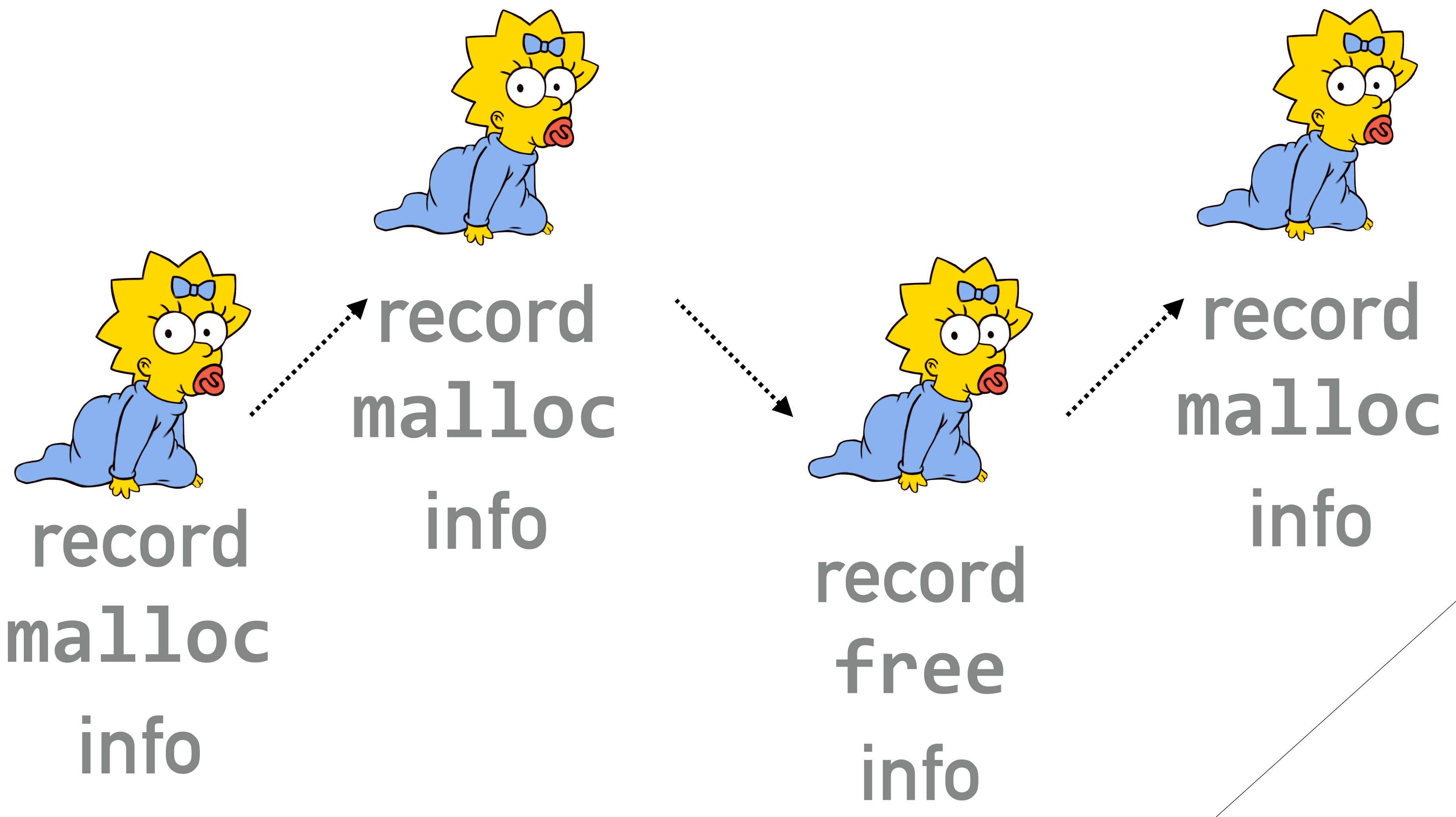
SCALENE

threshold-based sampling

only tracks every Δ ≥ 1MB



memory_profiler: ~300x slower



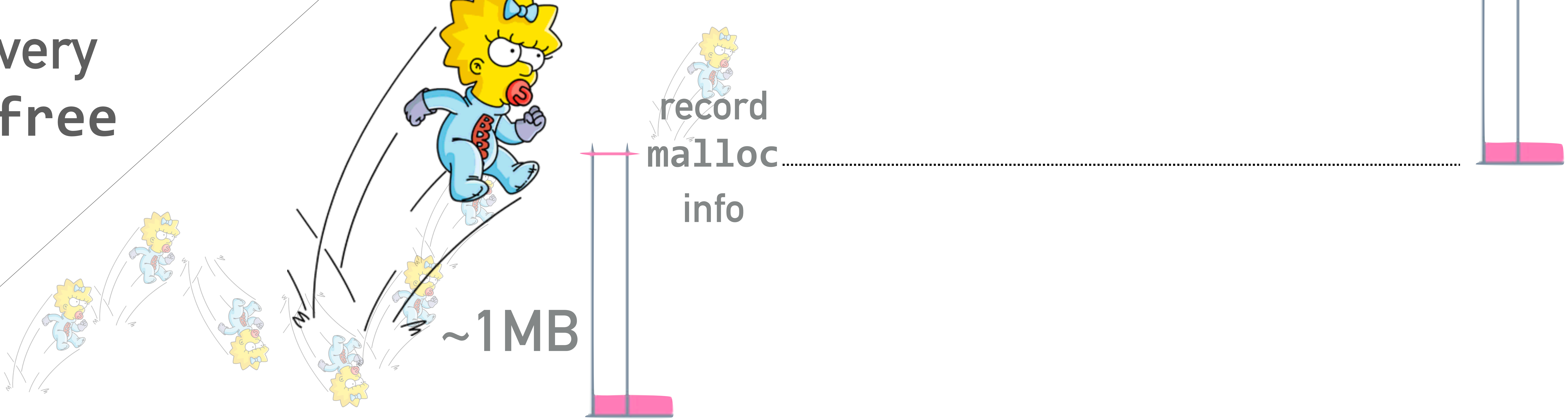
tracks every `malloc/free`

SCALENE

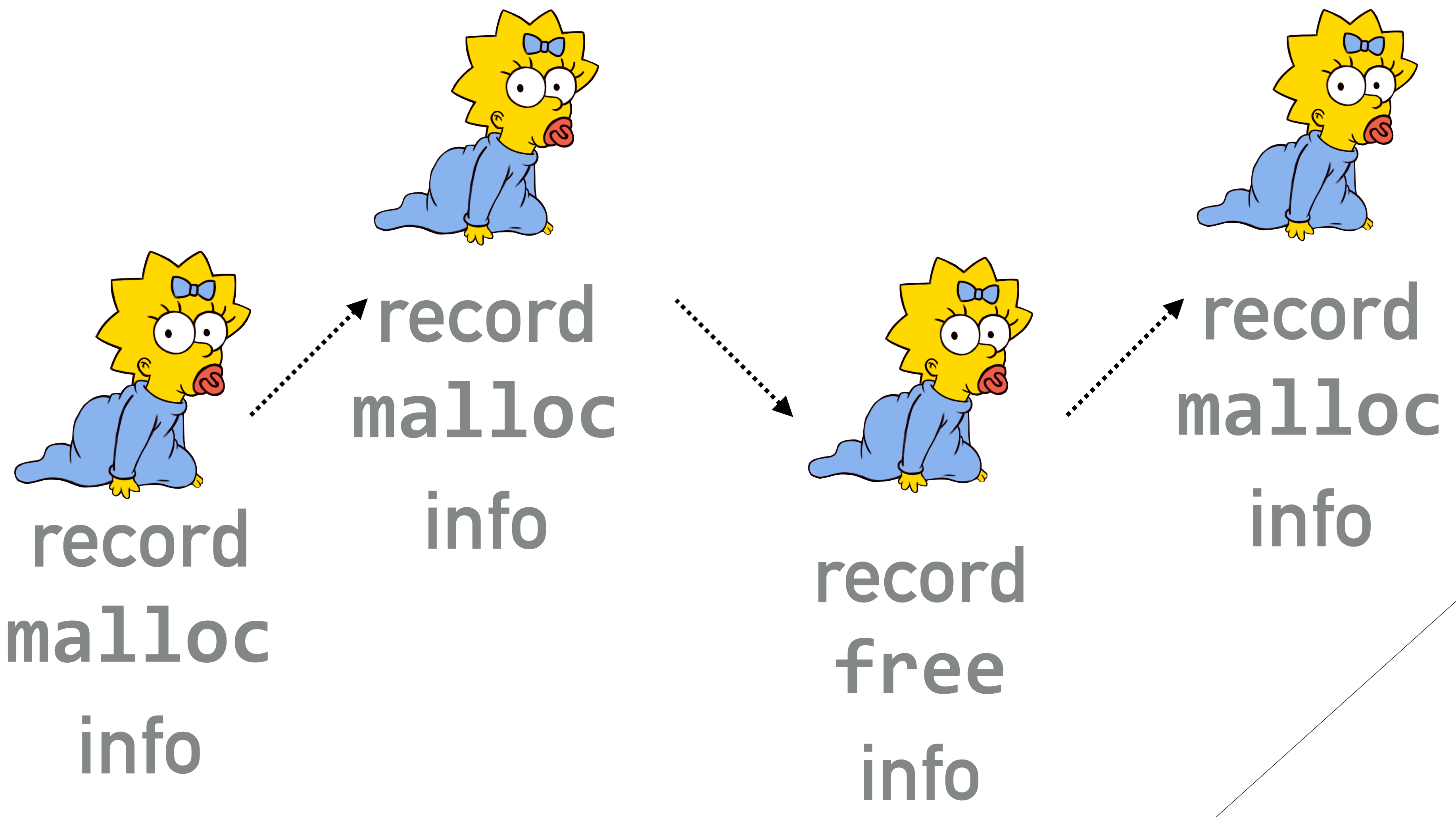
CPU MEMORY GPU

threshold-based sampling

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



memory_profiler: ~300x slower



record malloc info

record malloc info

record free info

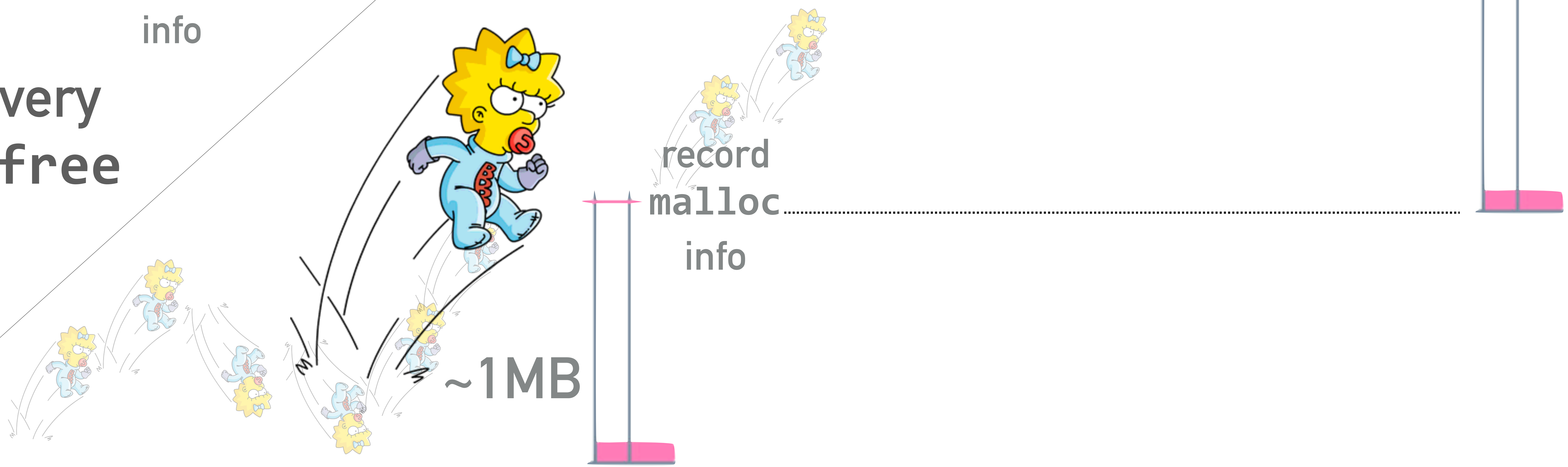
record malloc info

tracks every `malloc/free`

SCALENE

threshold-based sampling

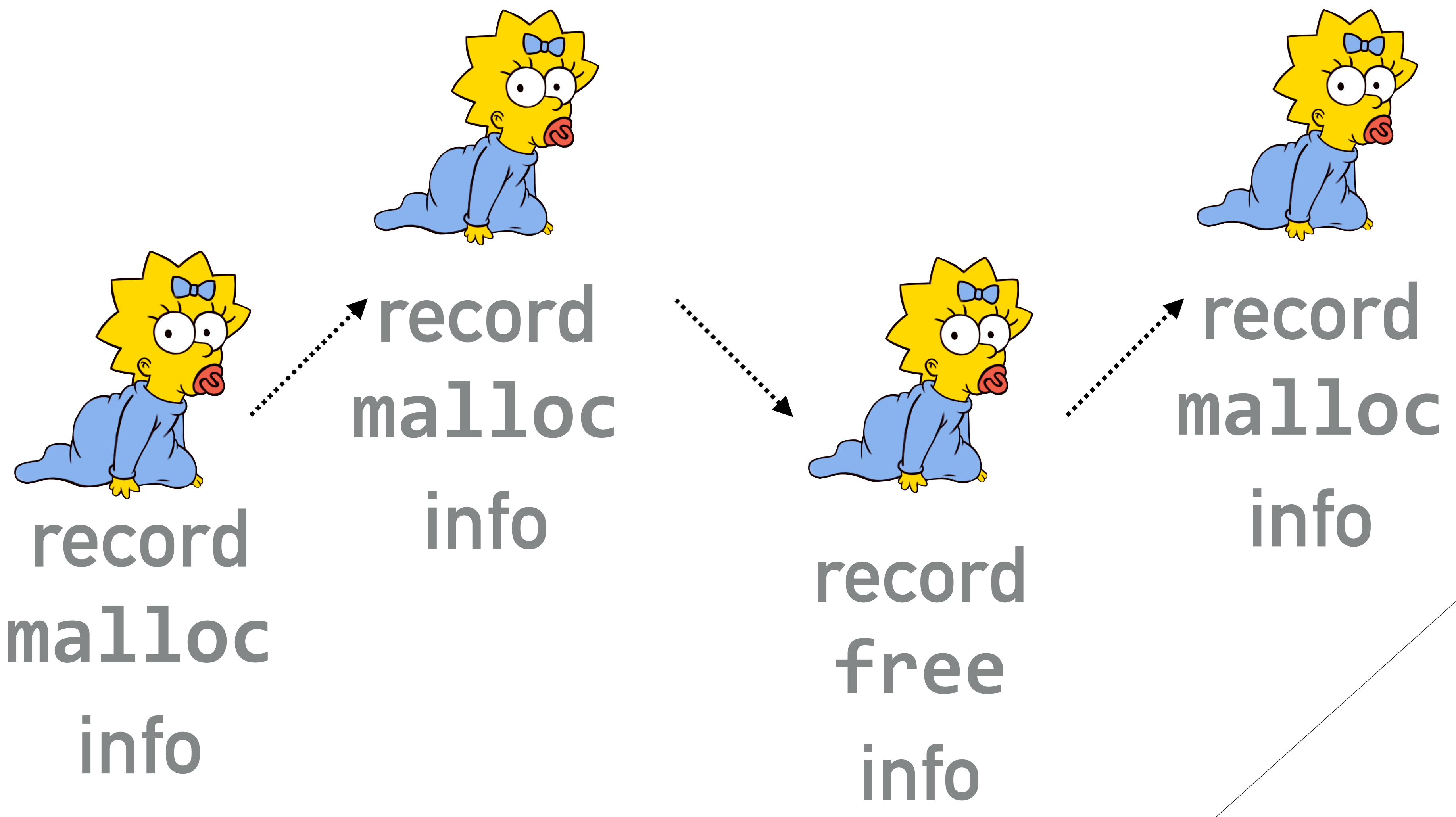
only tracks every Δ ≥ 1MB



record malloc info

~1MB

memory_profiler: ~300x slower

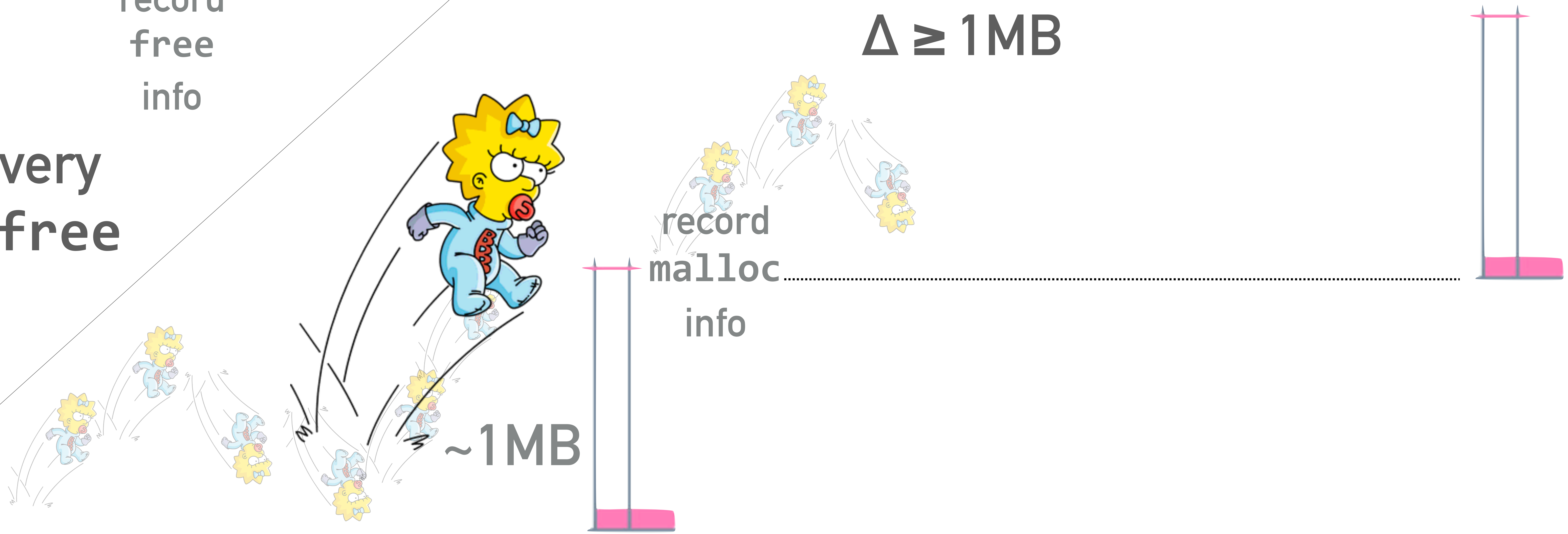


tracks every
`malloc/free`

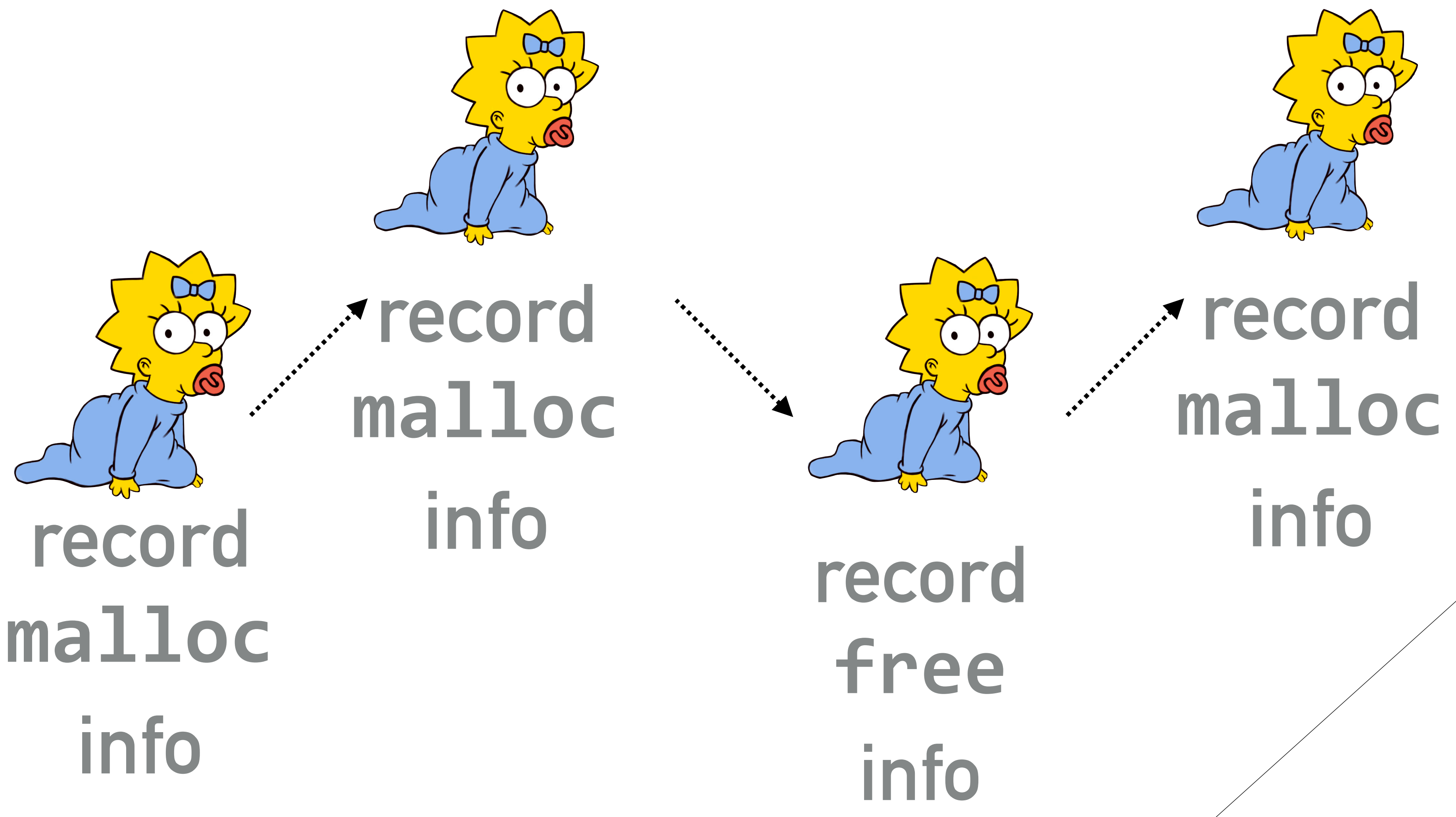
SCALENE

threshold-based sampling

only tracks every
Δ ≥ 1MB



memory_profiler: ~300x slower



record malloc info

record malloc info

record free info

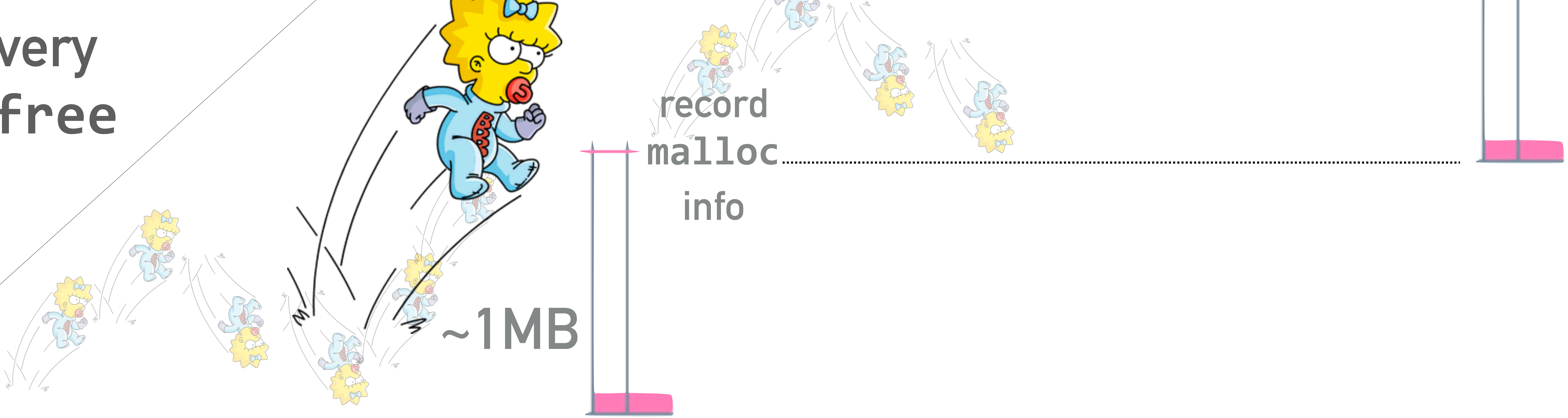
record malloc info

tracks every malloc/free

CPU MEMORY SCALENE GPU

threshold-based sampling

only tracks every Δ ≥ 1MB



~1MB

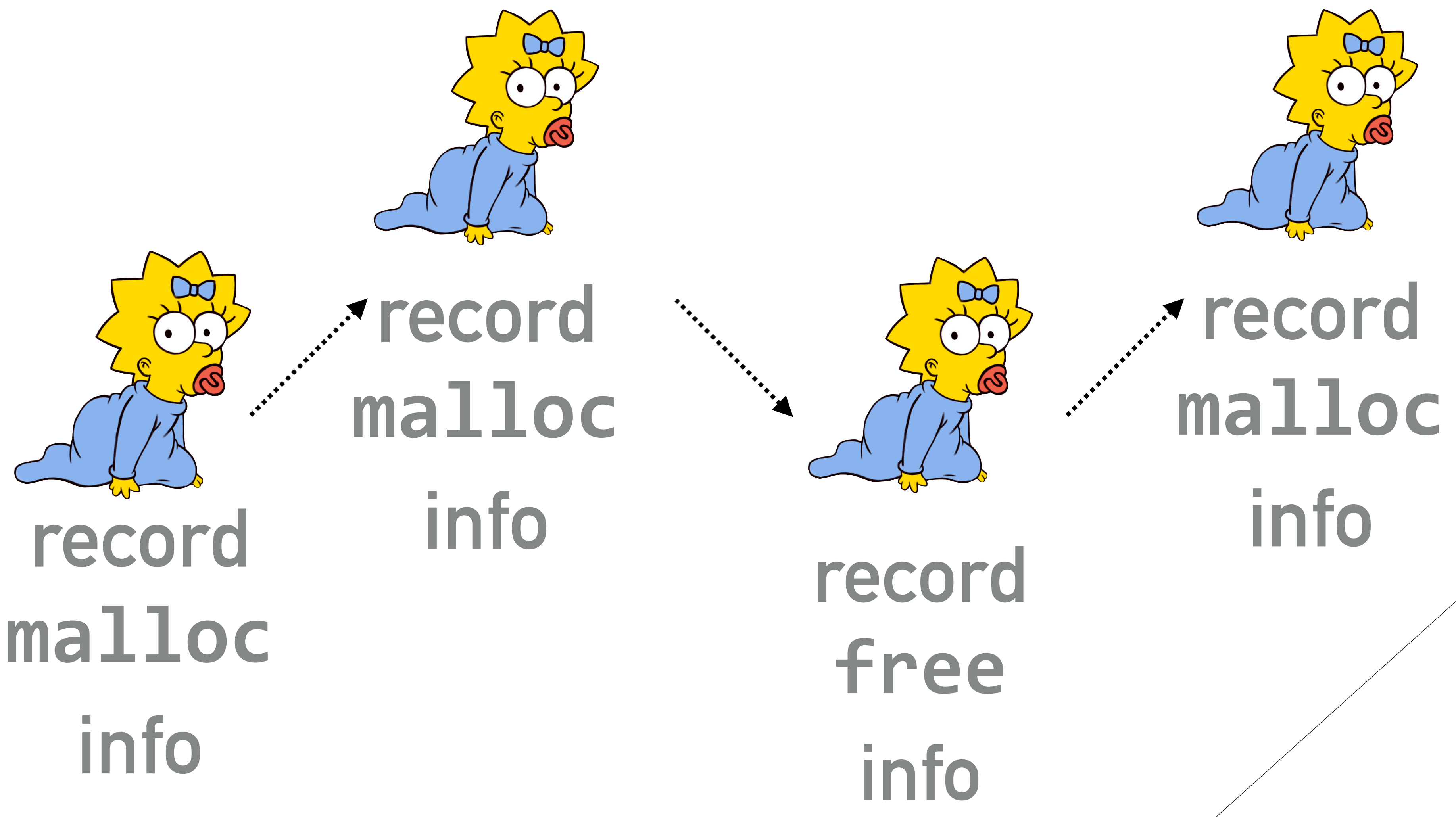
record malloc info

memory_profiler: ~300x slower

SCALENE

CPU MEMORY GPU

threshold-based sampling



record malloc info

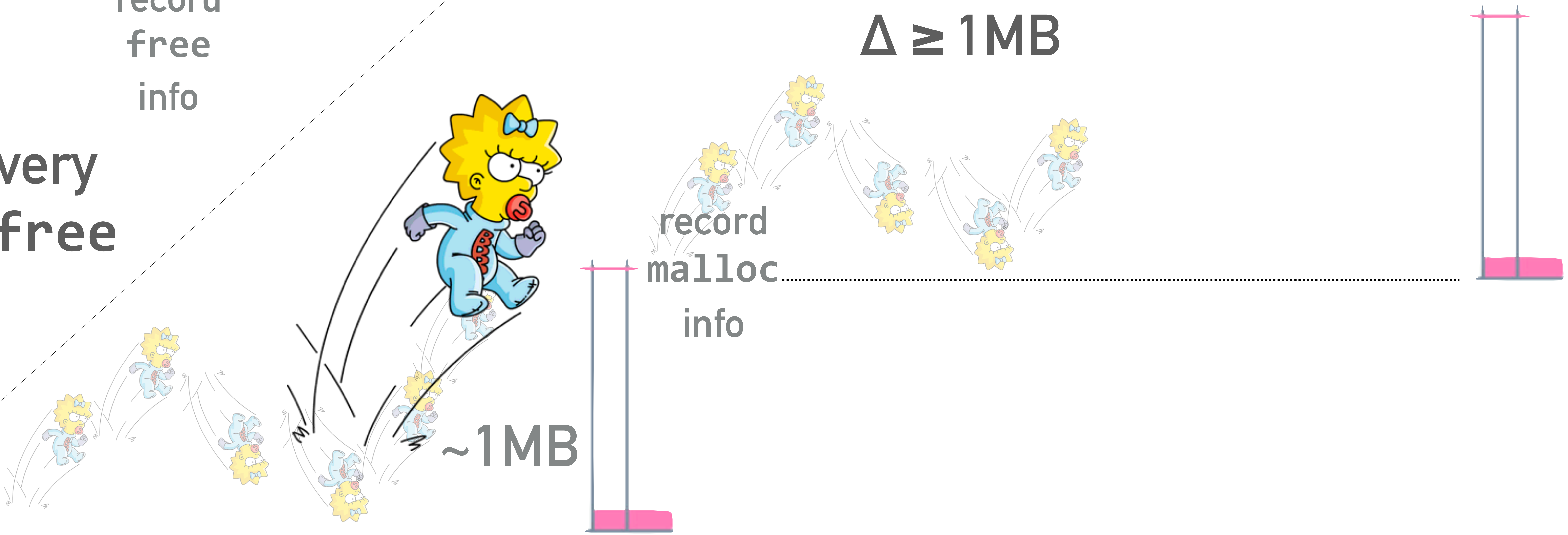
record malloc info

record free info

record malloc info

tracks every `malloc/free`

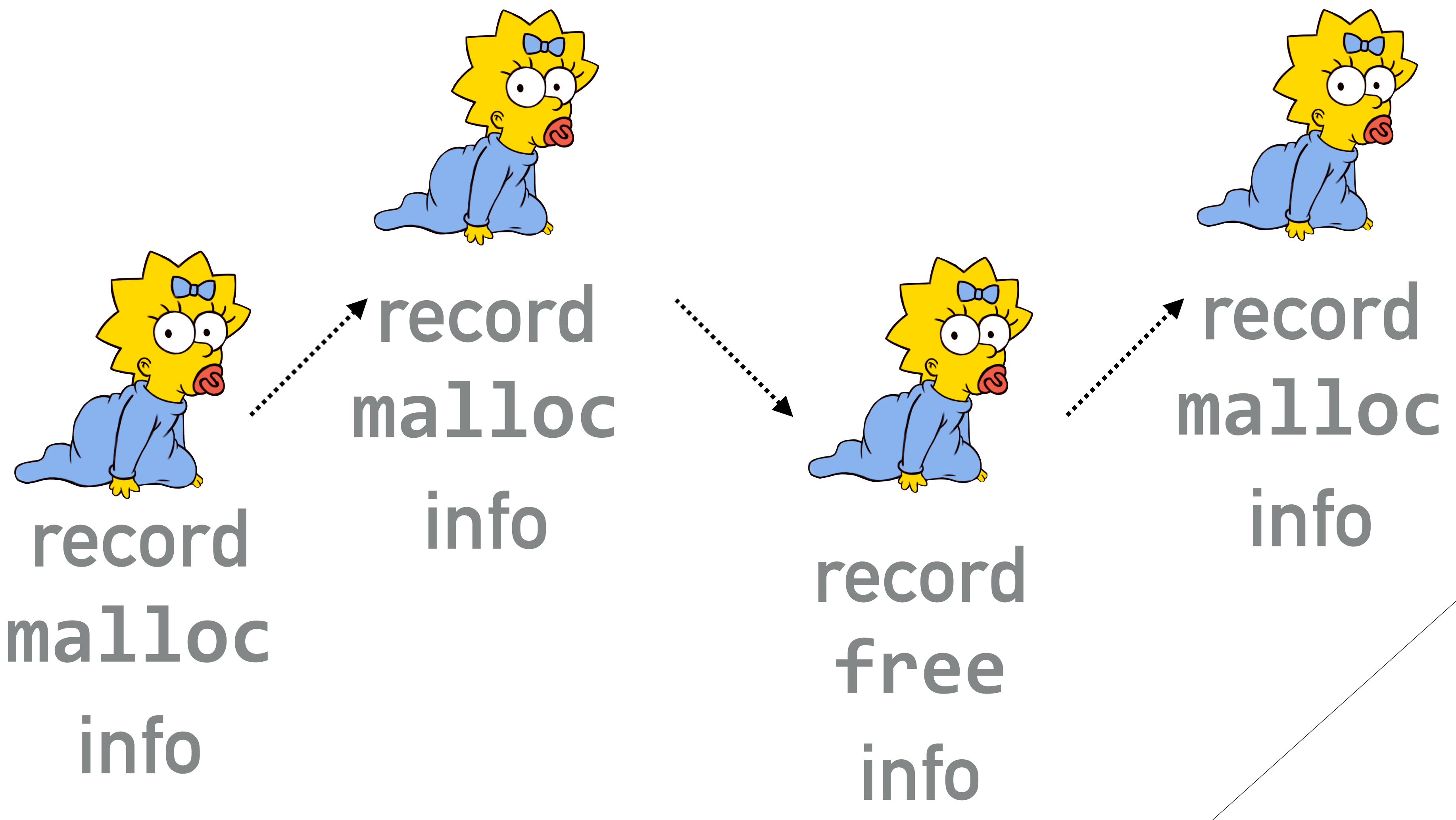
only tracks every Δ ≥ 1MB



record malloc info

~1MB

memory_profiler: ~300x slower



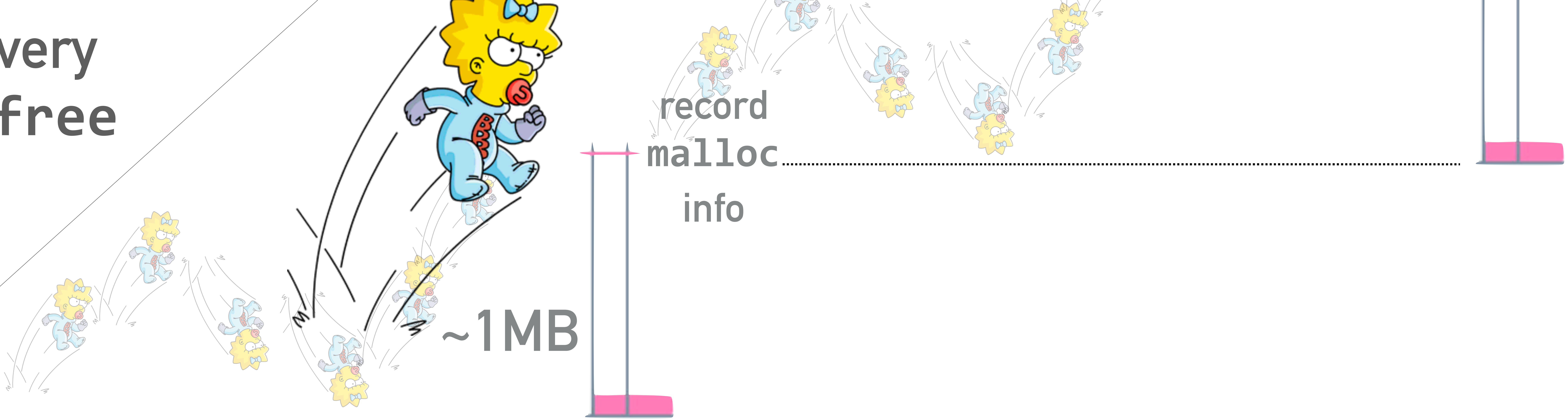
tracks every `malloc/free`

SCALENE

CPU MEMORY GPU

threshold-based sampling

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



memory_profiler: ~300x slower

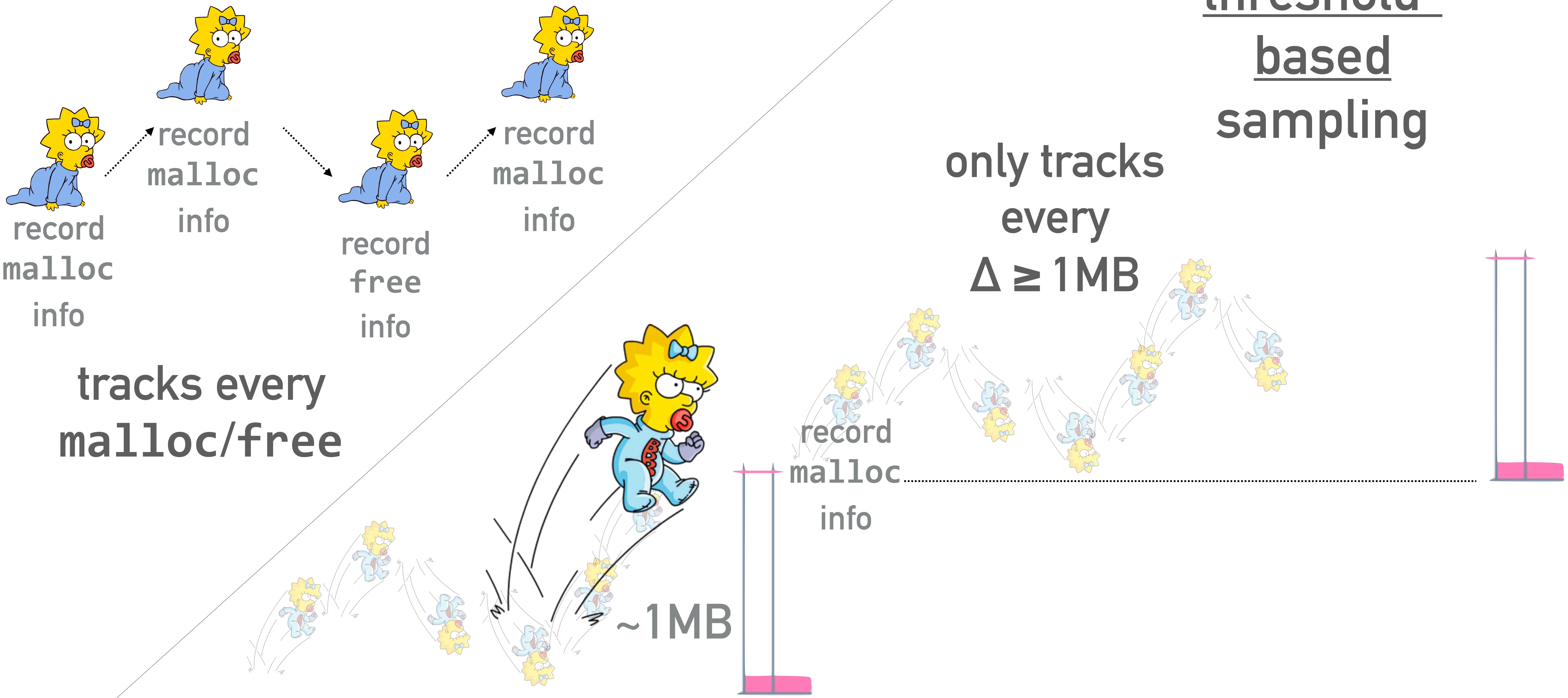
SCALENE

CPU

MEMORY

GPU

threshold-based sampling



record malloc info

record malloc info

record free info

record malloc info

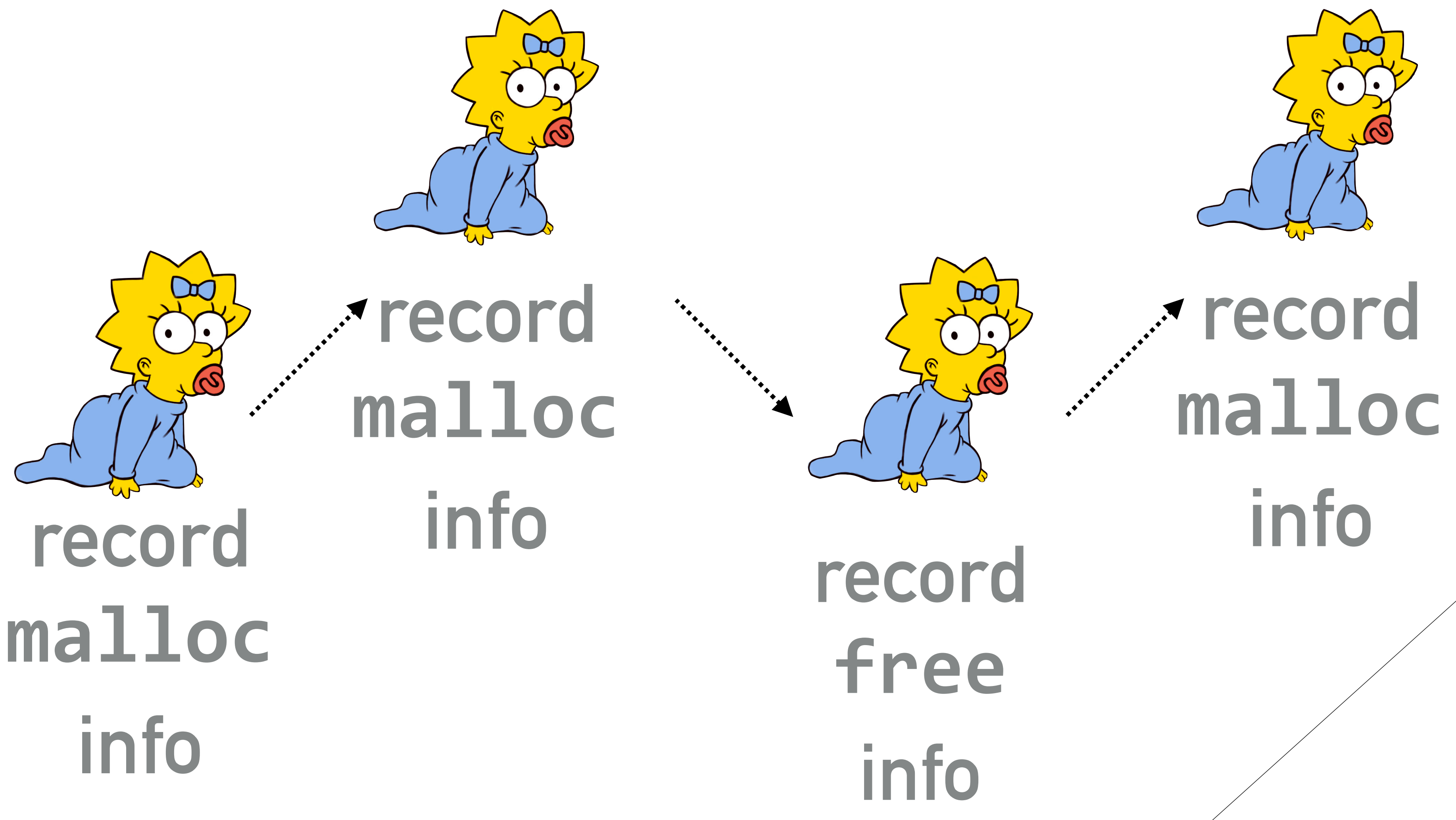
tracks every `malloc/free`

only tracks every Δ ≥ 1MB

~1MB

record malloc info

memory_profiler: ~300x slower



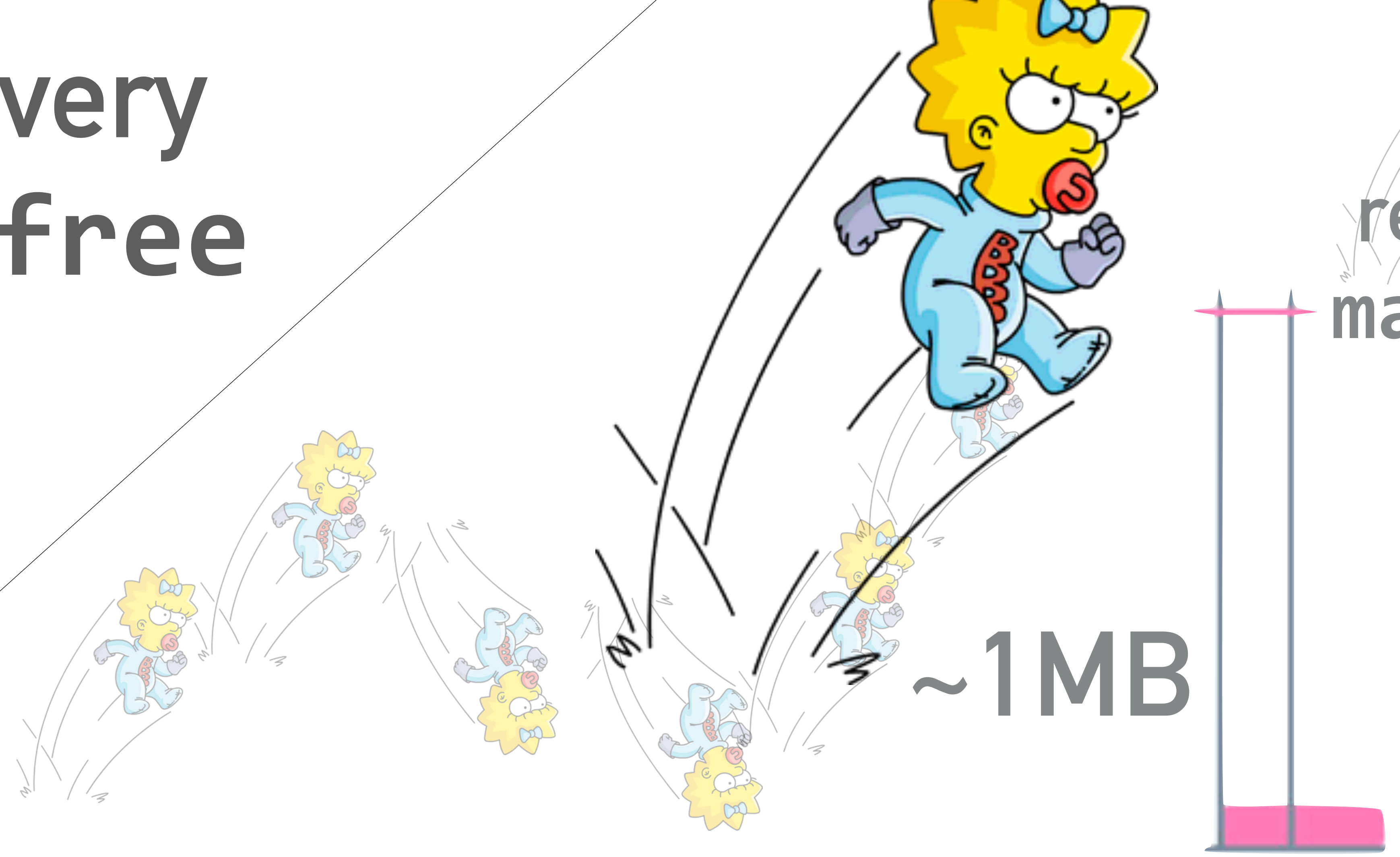
tracks every
`malloc/free`

SCALENE

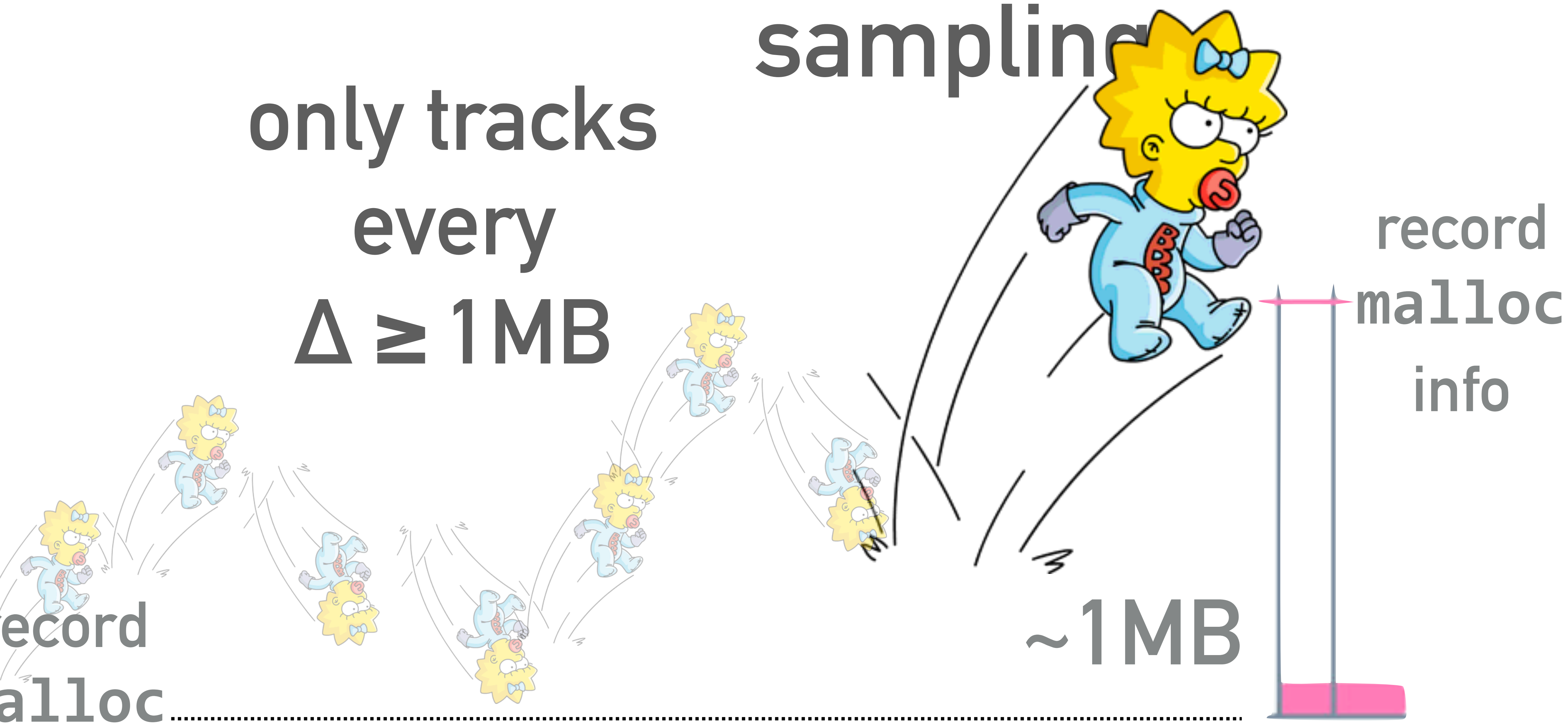
CPU MEMORY GPU

threshold-based sampling

only tracks every
Δ ≥ 1MB



record
malloc
info

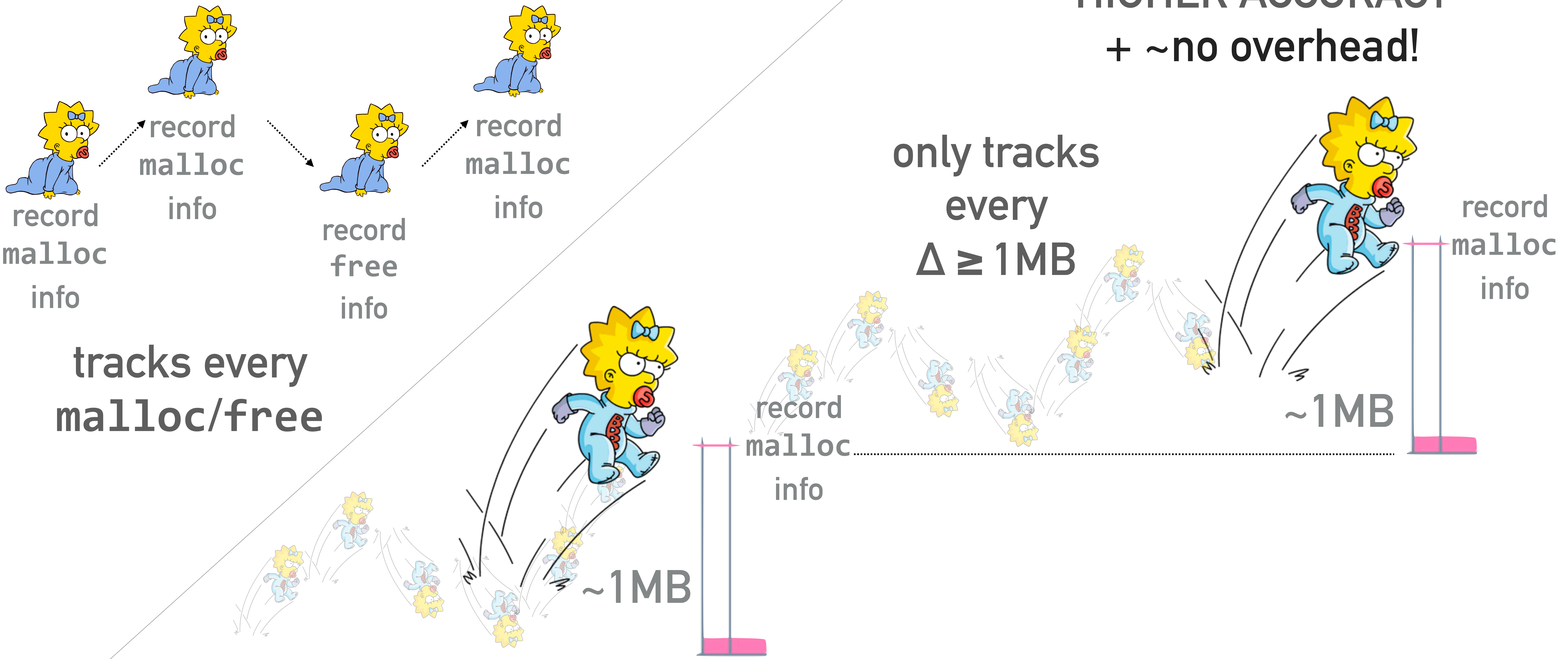


memory_profiler: ~300x slower

SCALENE

HIGHER ACCURACY

+ ~no overhead!



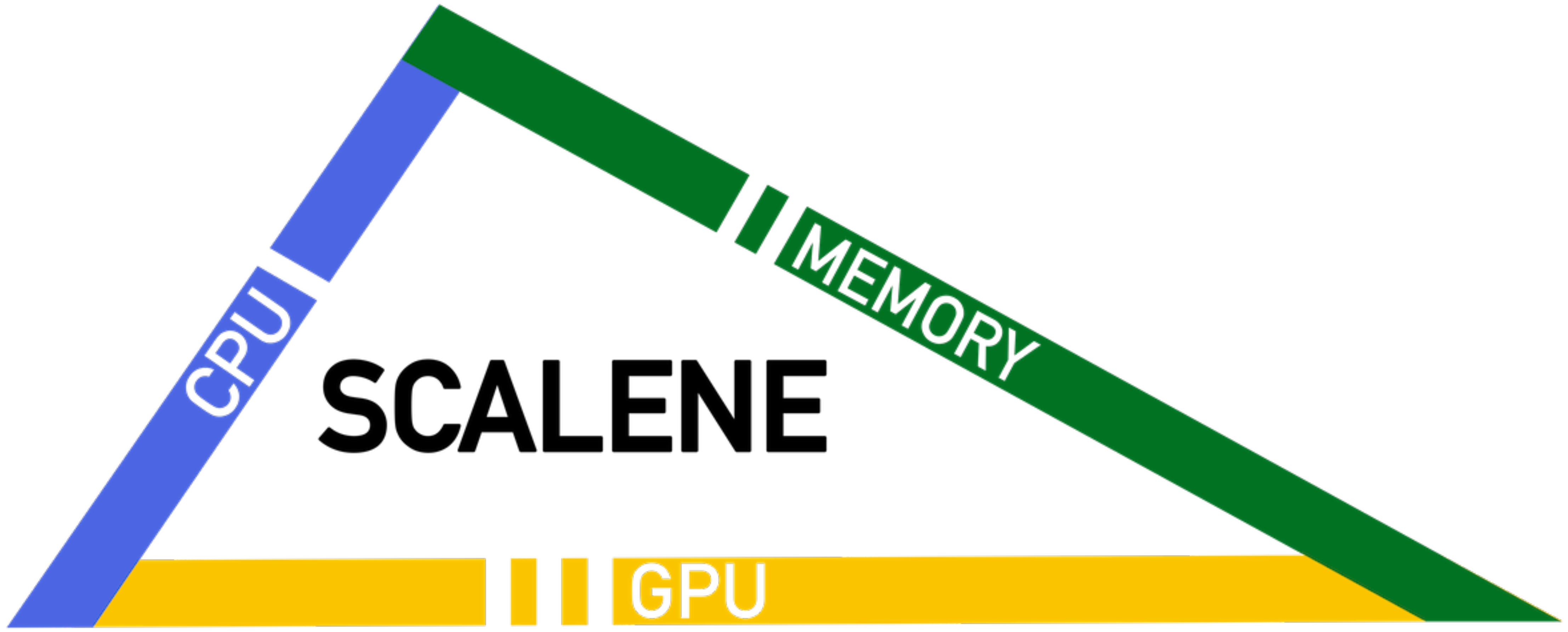
tracks every `malloc/free`

only tracks every Δ ≥ 1MB

reports leak volume (MB/s) per line

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY *(MB/s)*	GPU *util.*	GPU *memory*	LINE PROFILE *(click to reset order)* leaky/test-leaky.py
							562	3 `for i in range(10000000):`
			possible leak	100%	430		562	4 `    leak.leak("I want to run\nI want to hide\nI want to tear down the walls`

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



CPU

PYTHON

NATIVE

SYS%

MEMORY

PYTHON

NATIVE

AVERAGE & PEAK

MEMORY USAGE

OVER TIME, % OF MEM ALLOCATED

COPY VOLUME

(MB/s)

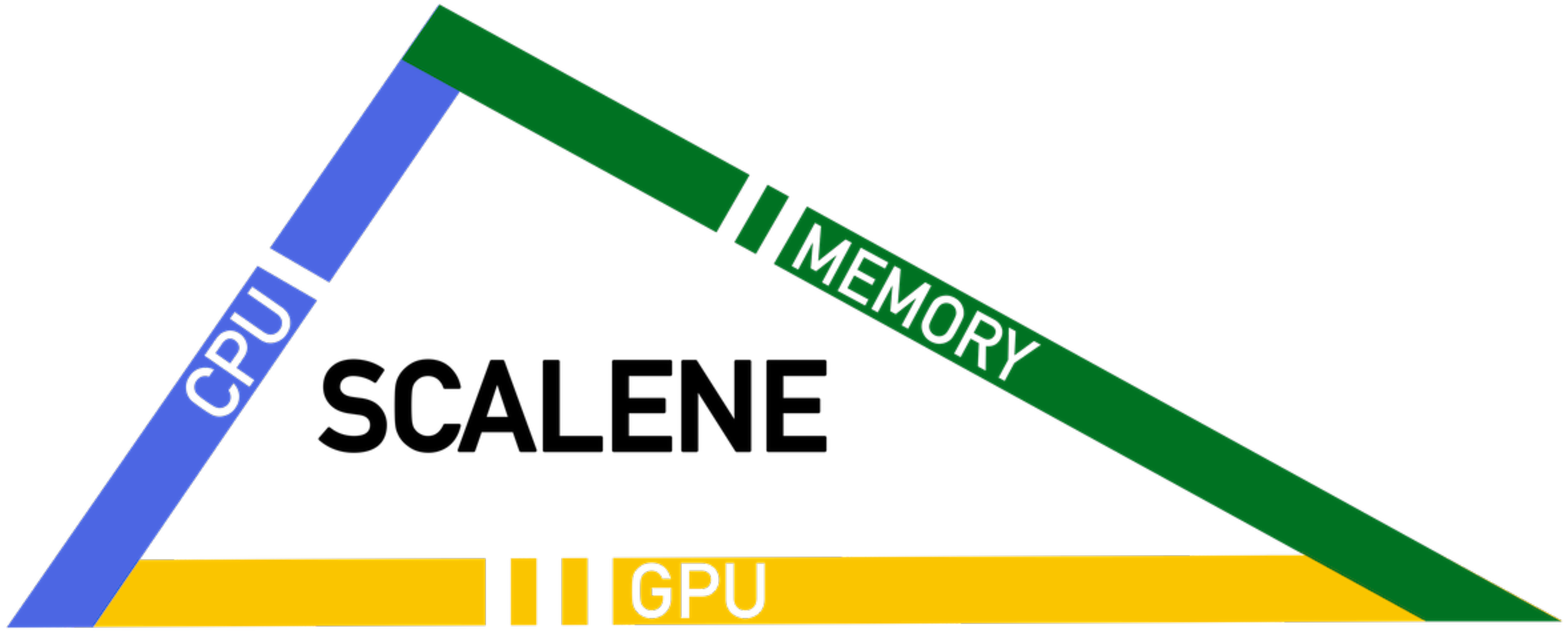
GPU

UTIL %, PEAK MEMORY

```
% pip install -U scalene
```

downloads 715k | downloads/month 27k

GitHub.com/plasma-umass/scalene



CPU
PYTHON
NATIVE
SYS%

MEMORY
PYTHON
NATIVE
AVERAGE & PEAK

MEMORY USAGE
OVER TIME, % OF MEM ALLOCATED

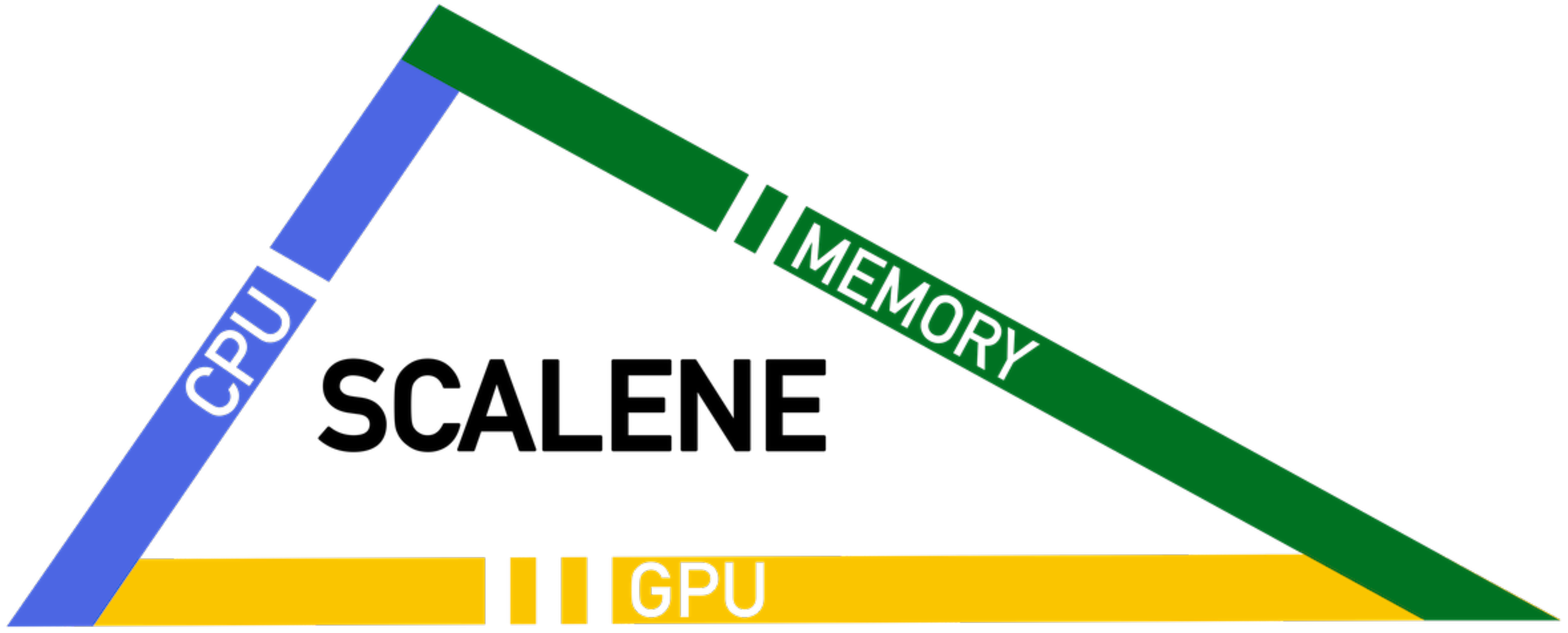
COPY VOLUME
(MB/s)

GPU
UTIL %, PEAK MEMORY

```
% pip install -U scalene
```

downloads 715k downloads/month 27k

GitHub.com/plasma-umass/scalene



CPU

PYTHON

NATIVE

SYS%

MEMORY

PYTHON

NATIVE

AVERAGE & PEAK

MEMORY USAGE

OVER TIME, % OF MEM ALLOCATED

COPY VOLUME

(MB/s)

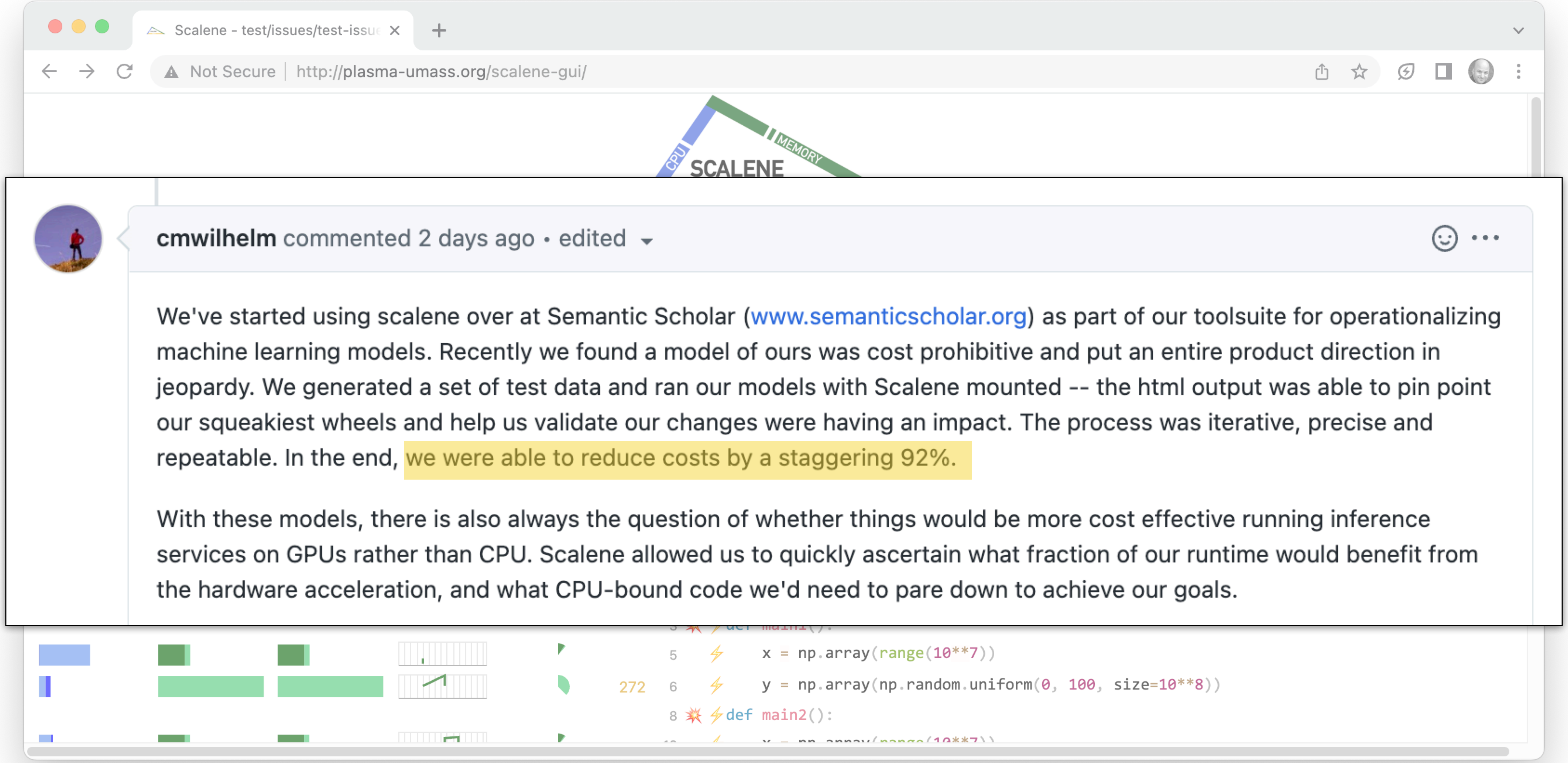
GPU

UTIL %, PEAK MEMORY

```
% pip install -U scalene
```

downloads 715k downloads/month 27k

GitHub.com/plasma-umass/scalene


Scalene - test/issues/test-issu
Not Secure
http://plasma-umass.org/scalene-gui/
CPU
MEMORY
SCALENE
272

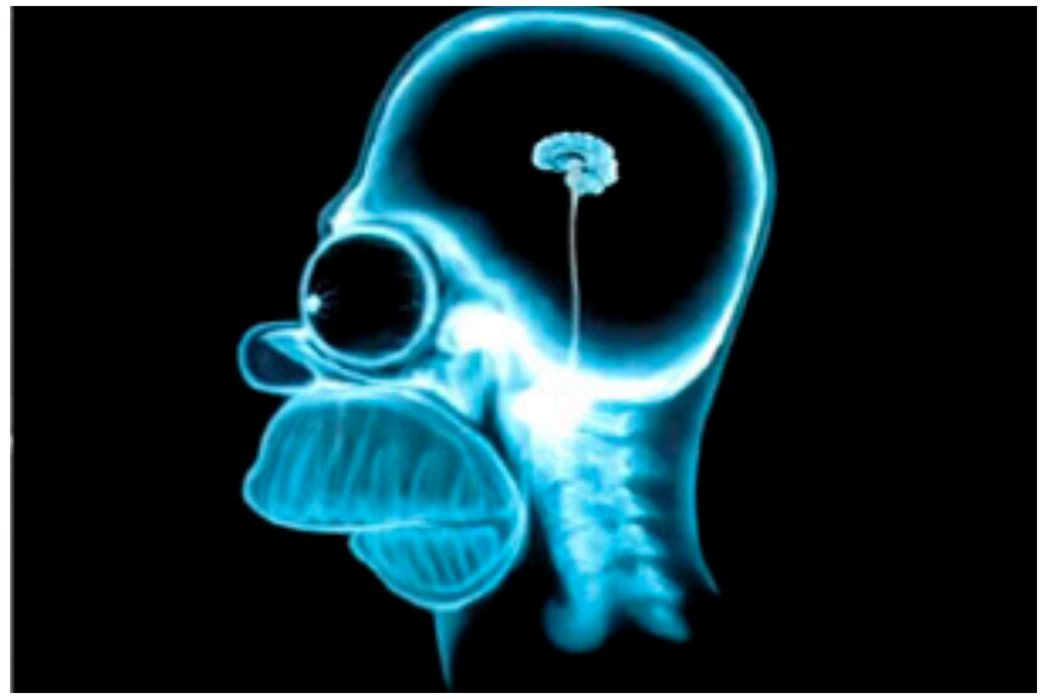
cmwilhelm commented 2 days ago • edited

We've started using scalene over at Semantic Scholar (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.

```
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=10**8))
def main2():
```

writing your code in Python



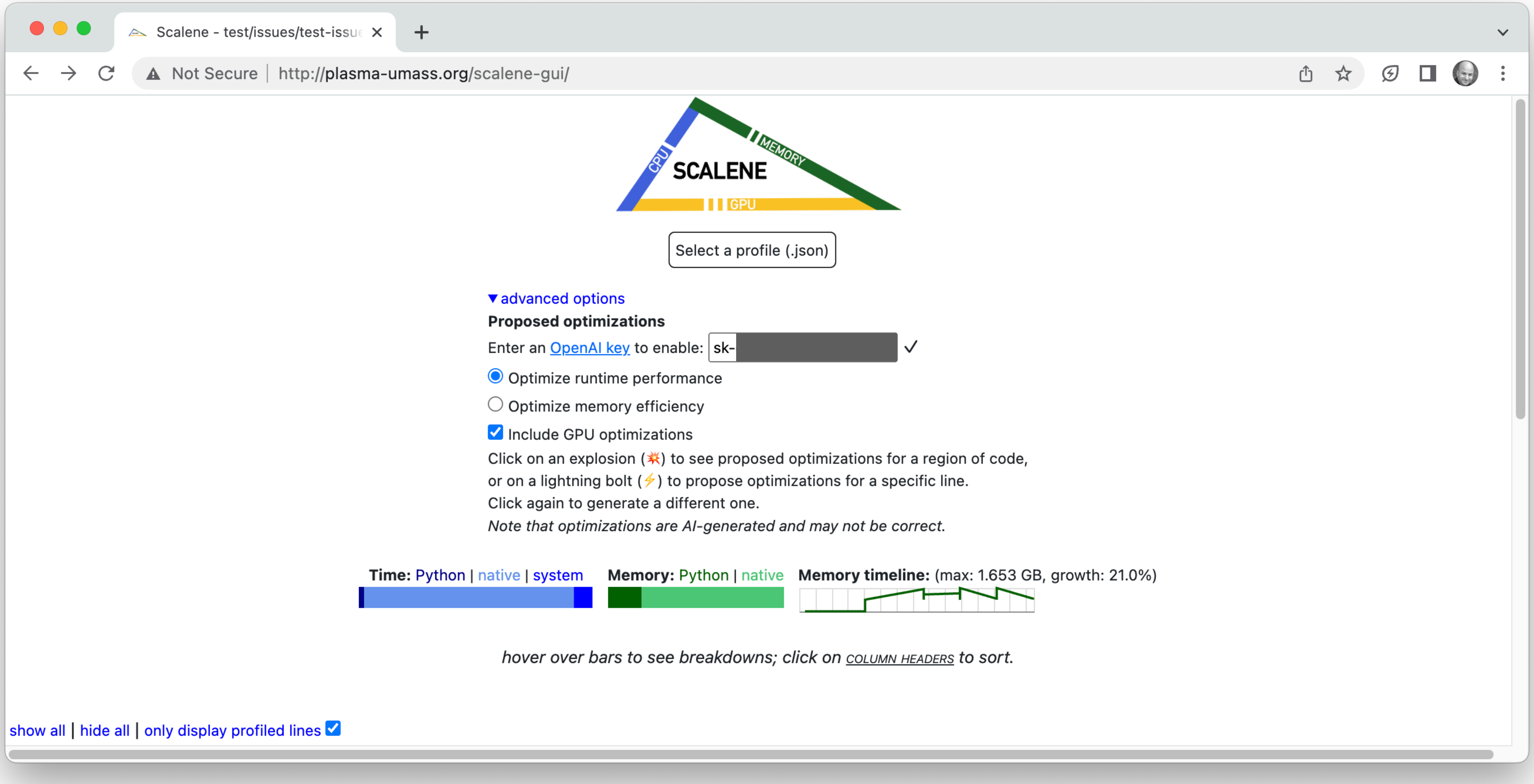
writing your code in Python

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-issu
Not Secure
http://plasma-umass.org/scalene-gui/
CPU
MEMORY
SCALENE
GPU

Select a profile (.json)

▾ advanced options

Proposed optimizations

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

● Optimize runtime performance

○ Optimize memory efficiency

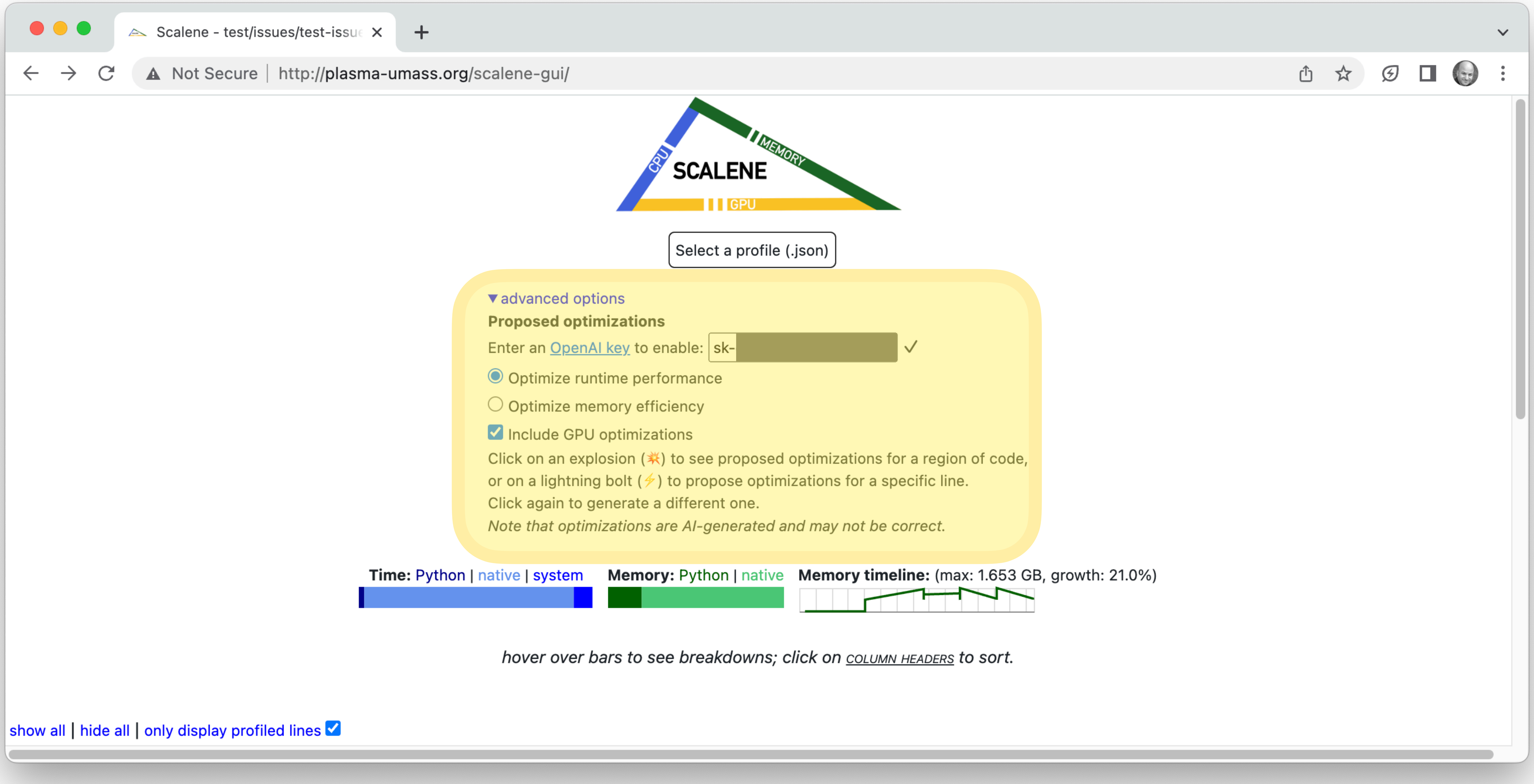
☑ Include GPU optimizations

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-issu
Not Secure
http://plasma-umass.org/scalene-gui/
CPU
MEMORY
GPU
SCALENE

Select a profile (.json)

▾ advanced options

Proposed optimizations

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

● Optimize runtime performance

○ Optimize memory efficiency

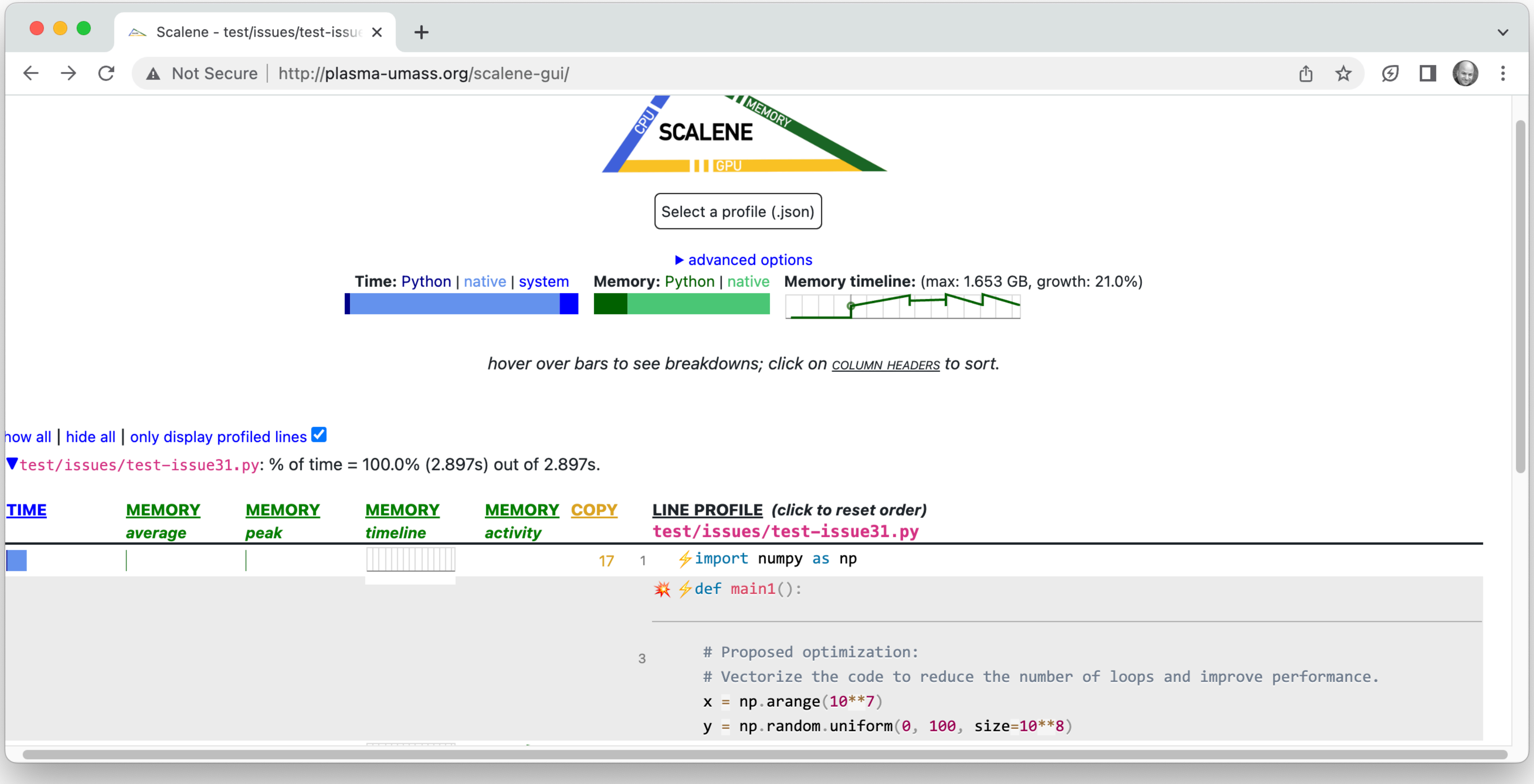
☑ Include GPU optimizations

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 ☑



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

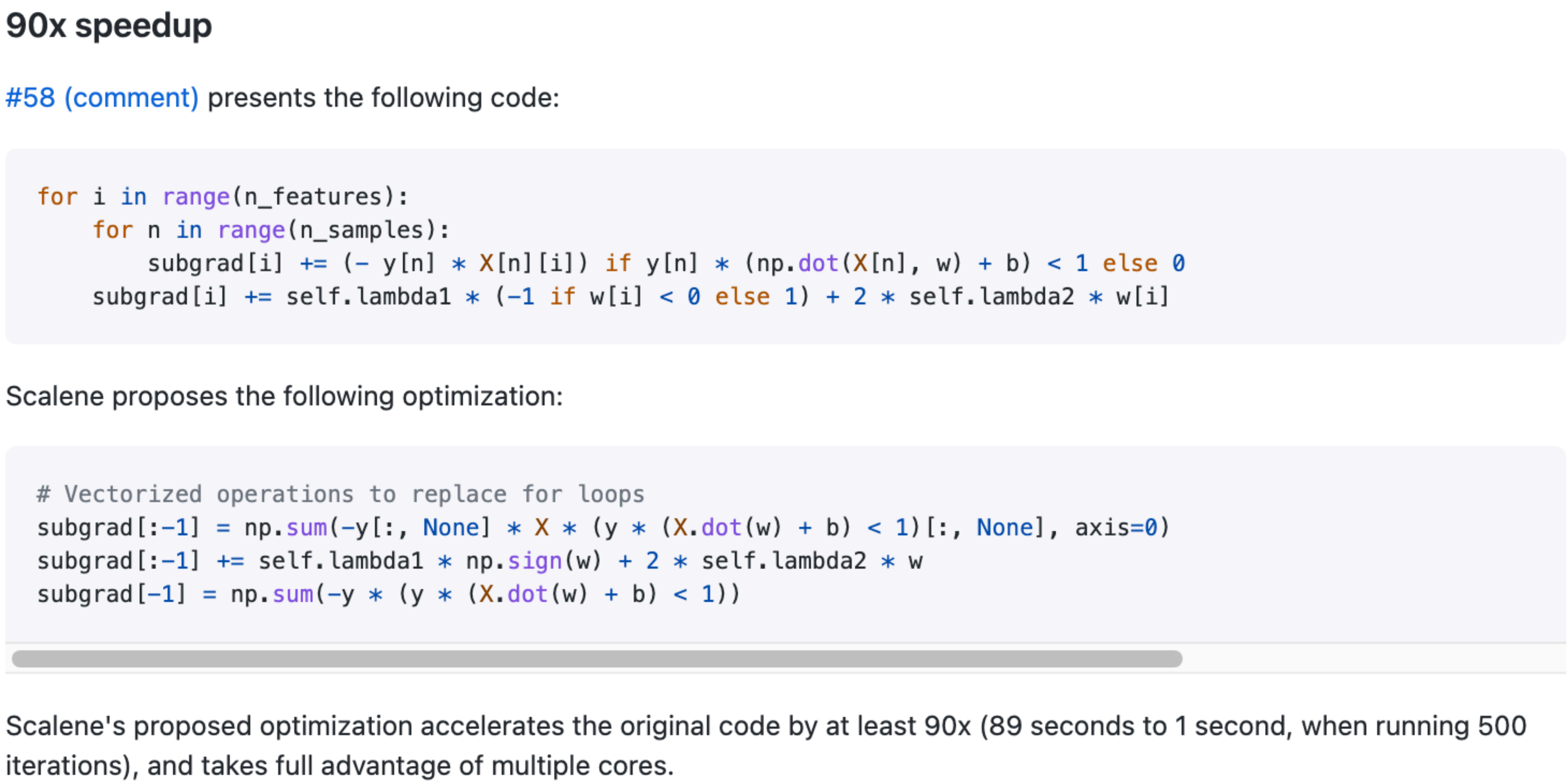
hover over bars to see breakdowns; click on COLUMN HEADERS to sort.

how all | hide all | only display profiled lines ☑

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

TIME	MEMORY *average*	MEMORY *peak*	MEMORY *timeline*	MEMORY *activity*	COPY		LINE PROFILE *(click to reset order)* test/issues/test-issue31.py
					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

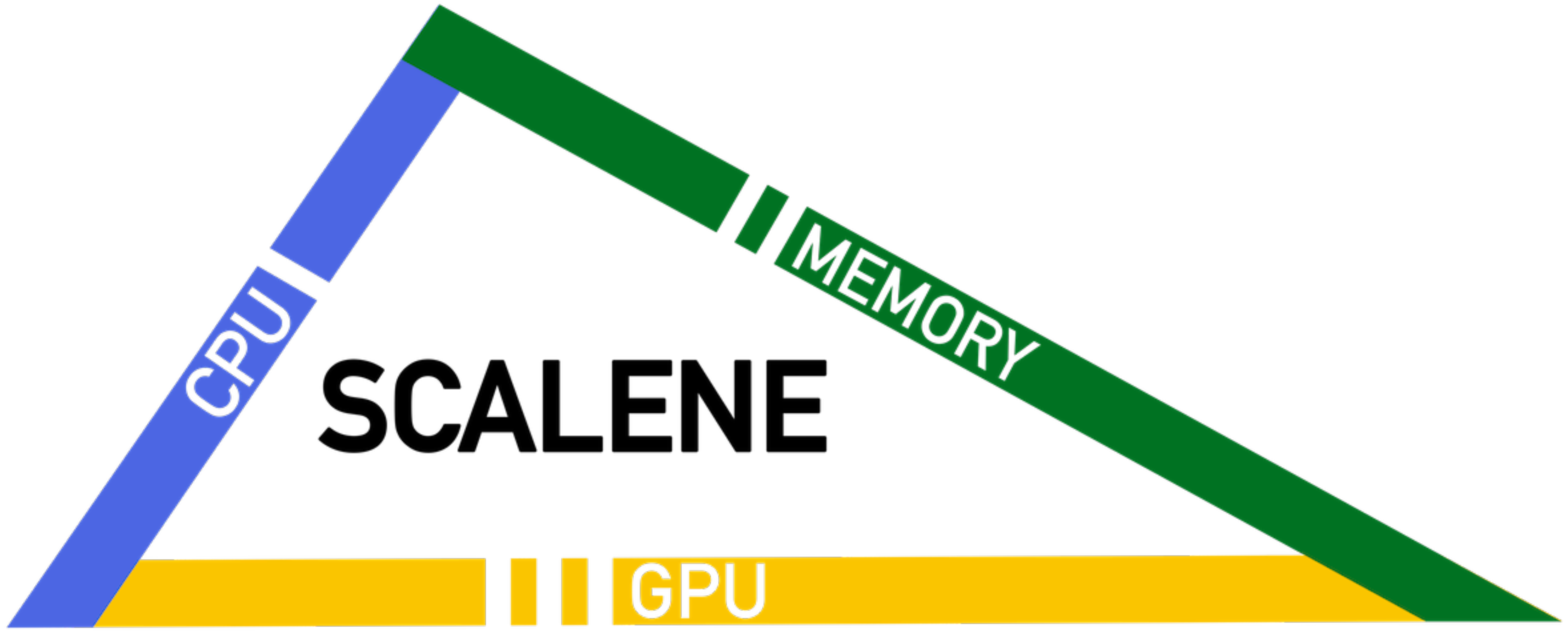
#58 (comment) presents the following code:

```
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.



CPU

PYTHON

NATIVE

SYS%

MEMORY

PYTHON

NATIVE

AVERAGE & PEAK

MEMORY USAGE

OVER TIME, % OF MEM ALLOCATED

COPY VOLUME

(MB/s)

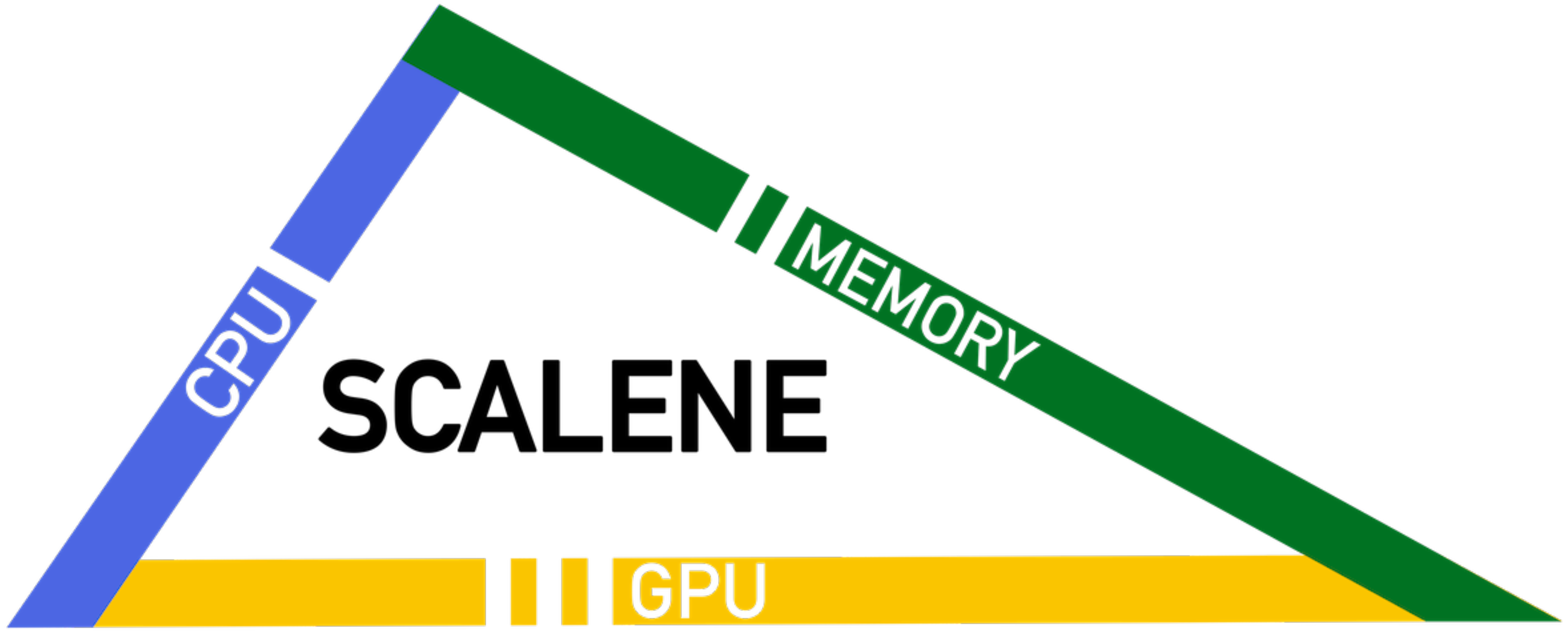
GPU

UTIL %, PEAK MEMORY

```
% pip install -U scalene
```

downloads 715k | downloads/month 27k

GitHub.com/plasma-umass/scalene



AI powered optimizations!

CPU

PYTHON

NATIVE

SYS%

MEMORY

PYTHON

NATIVE

AVERAGE & PEAK

MEMORY USAGE

OVER TIME, % OF MEM ALLOCATED

COPY VOLUME

(MB/s)

GPU

UTIL %, PEAK MEMORY

```
% pip install -U scalene
```

downloads	715k

downloads/month	27k

GitHub.com/plasma-umass/scalene

python™