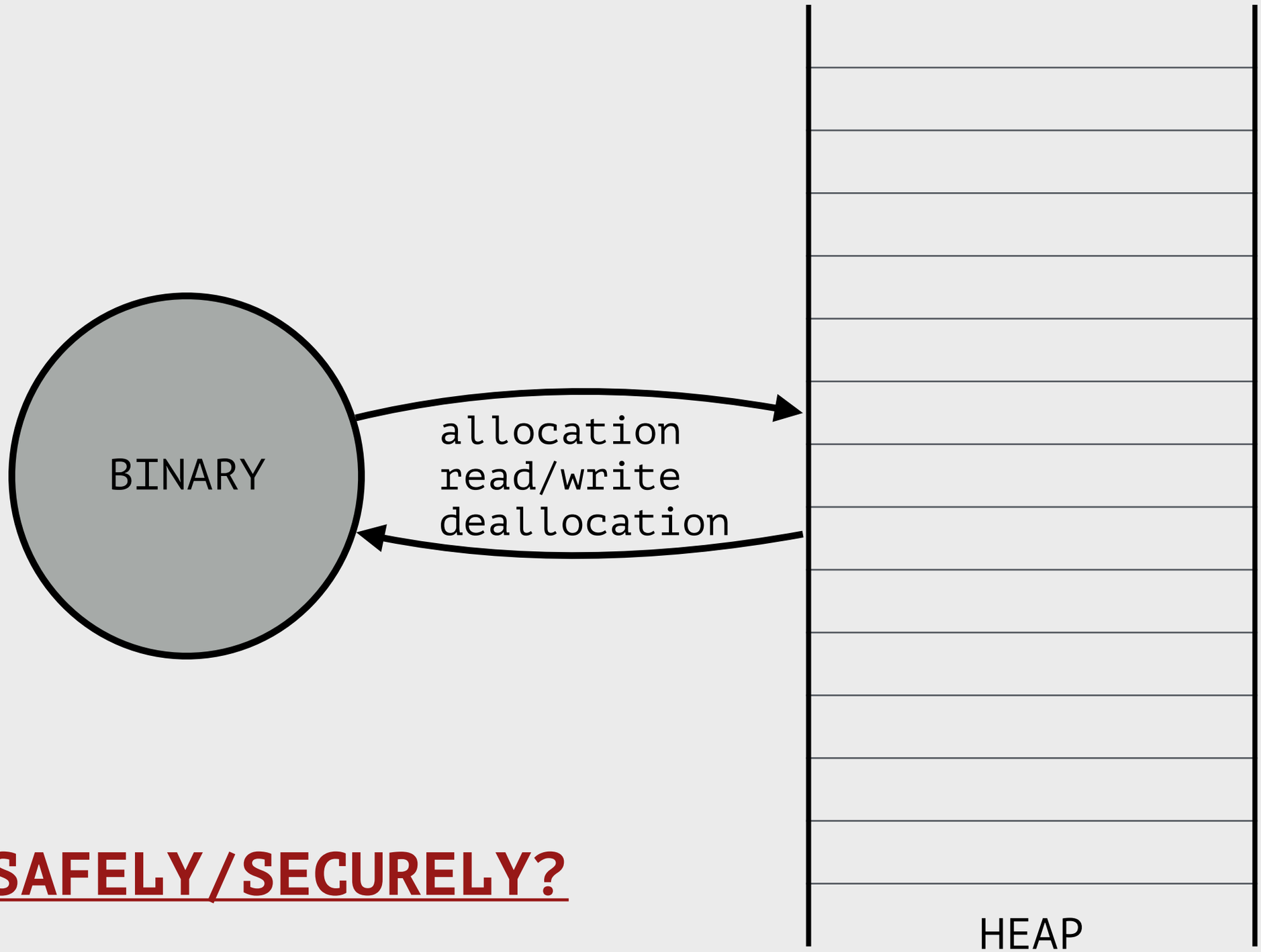
Metrics for runtime detection of allocators in binaries

August 14th, 2017

Franck de Goër - Roland Groz - Laurent Mounier

Grenoble, FRANCE



SAFELY/SECURELY?

SAFELY/SECURELY?

[**memory leaks**]

> every **allocated** block is **freed**

[**heap overflows**]

> access within the bounds of **allocated** blocks

[**use-after-frees**]

> no access after a block is **freed**

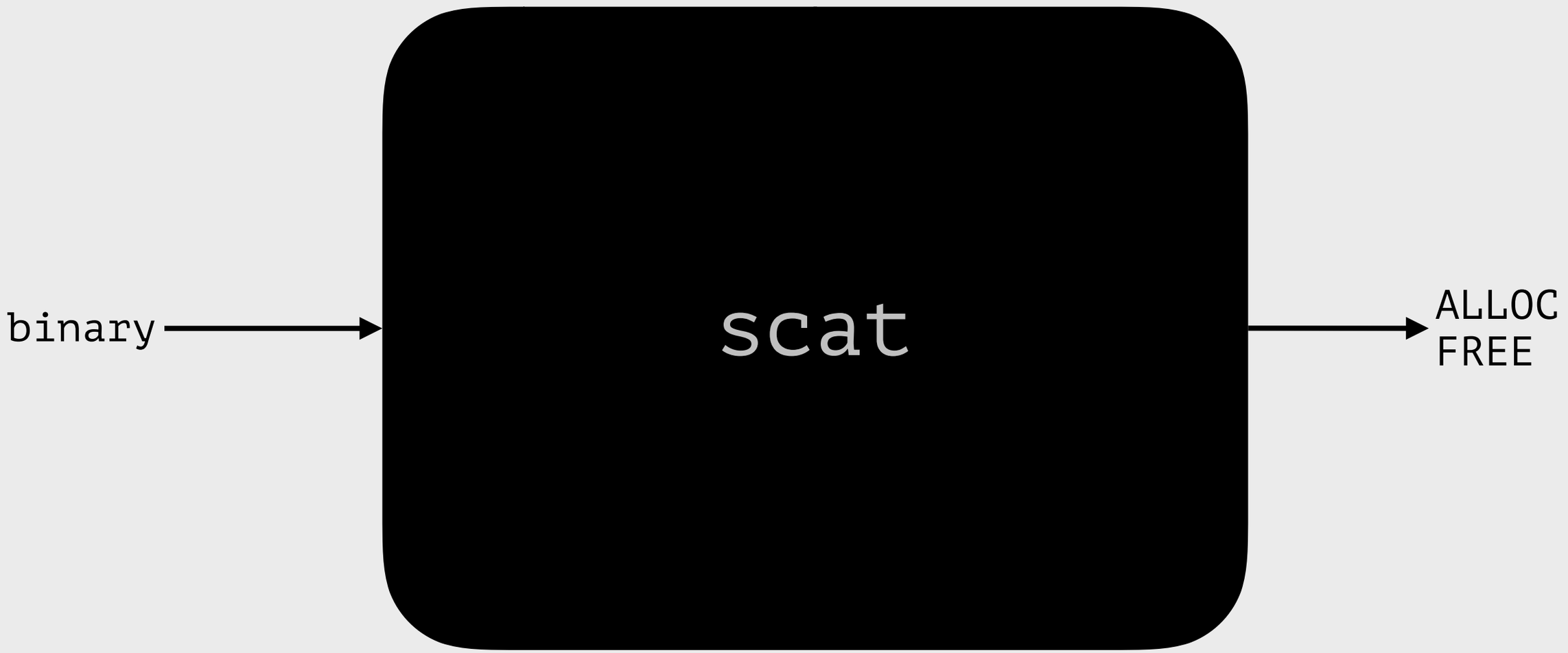
WHO ALLOCATES MEMORY?

STEP #1

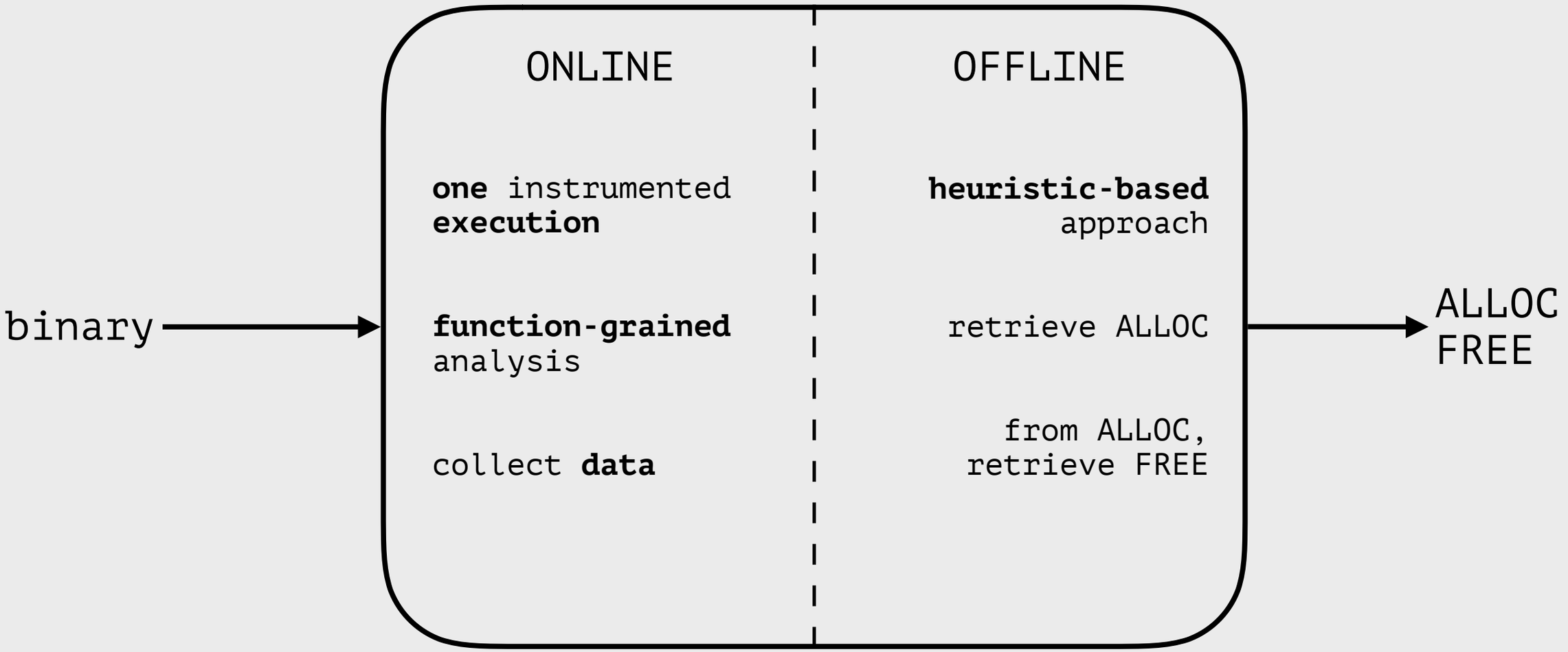
retrieve ALLOC and FREE

STEP #1 - retrieve ALLOC/FREE

overview | requirements | online | offline



STEP #1 - retrieve ALLOC/FREE



STEP #1 - retrieve ALLOC/FREE

REQUIREMENT	PROVIDER
function instrumentation	PIN
function parameters	ABI
function undertyped prototypes	scat

STEP #1 - retrieve ALLOC/FREE


```
ADDR malloc(NUM);

NUM free(ADDR);

NUM add(NUM, NUM);

NUM strnlen(ADDR, NUM);

…
```

STEP #1 - retrieve ALLOC/FREE

Instrument each call and return, and log:

[*] concrete value of each parameter

[*] identifier of the function being called or returning

[*] identifier of the function that called fid

STEP #1 - retrieve ALLOC/FREE

UNDERTYPED PROTOTYPE	INSTRUMENTATION?
ADDR malloc(NUM);	yes
NUM free(ADDR);	yes
NUM add(NUM, NUM);	no
NUM strnlen(ADDR, NUM);	yes
…	

STEP #1 - retrieve ALLOC/FREE

MAIN HEURISTICS

[**ALLOC**]

ALLOC produces a high number of new addresses

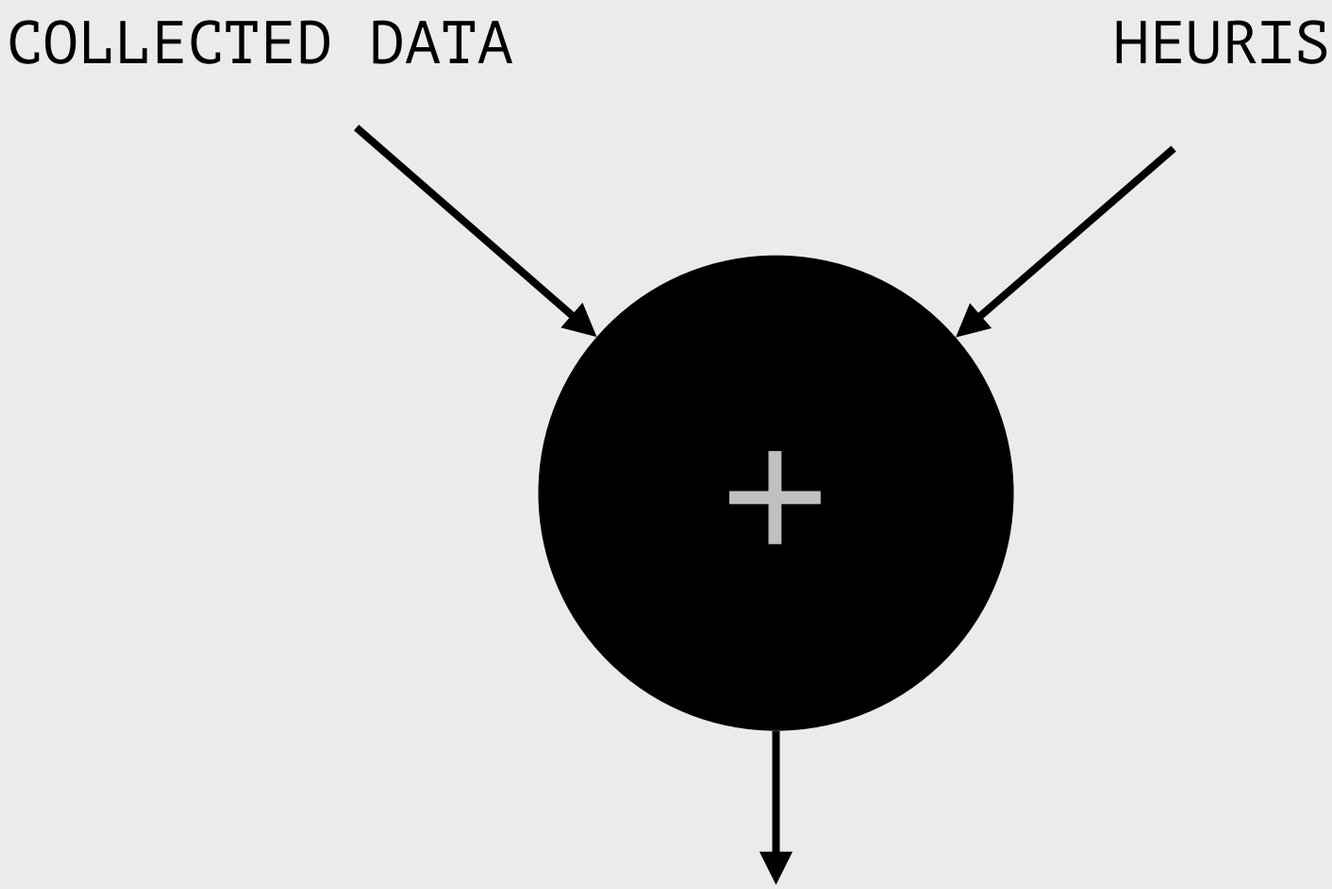
[**FREE**]

FREE takes, as a parameter, a high number of values that were output by a previous call to ALLOC

STEP #1 - retrieve ALLOC/FREE

ALLOC	**[production]**	ALLOC produces a high number of new addresses
	[**diversity**]	ALLOC is called by multiple functions
FREE		FREE takes, as a parameter, a high number of values that were output by a previous call to ALLOC
	[**last accessor**]	FREE is the last function to access an allocated block
	[**diversity**]	FREE is called by multiple functions

STEP #1 - retrieve ALLOC/FREE



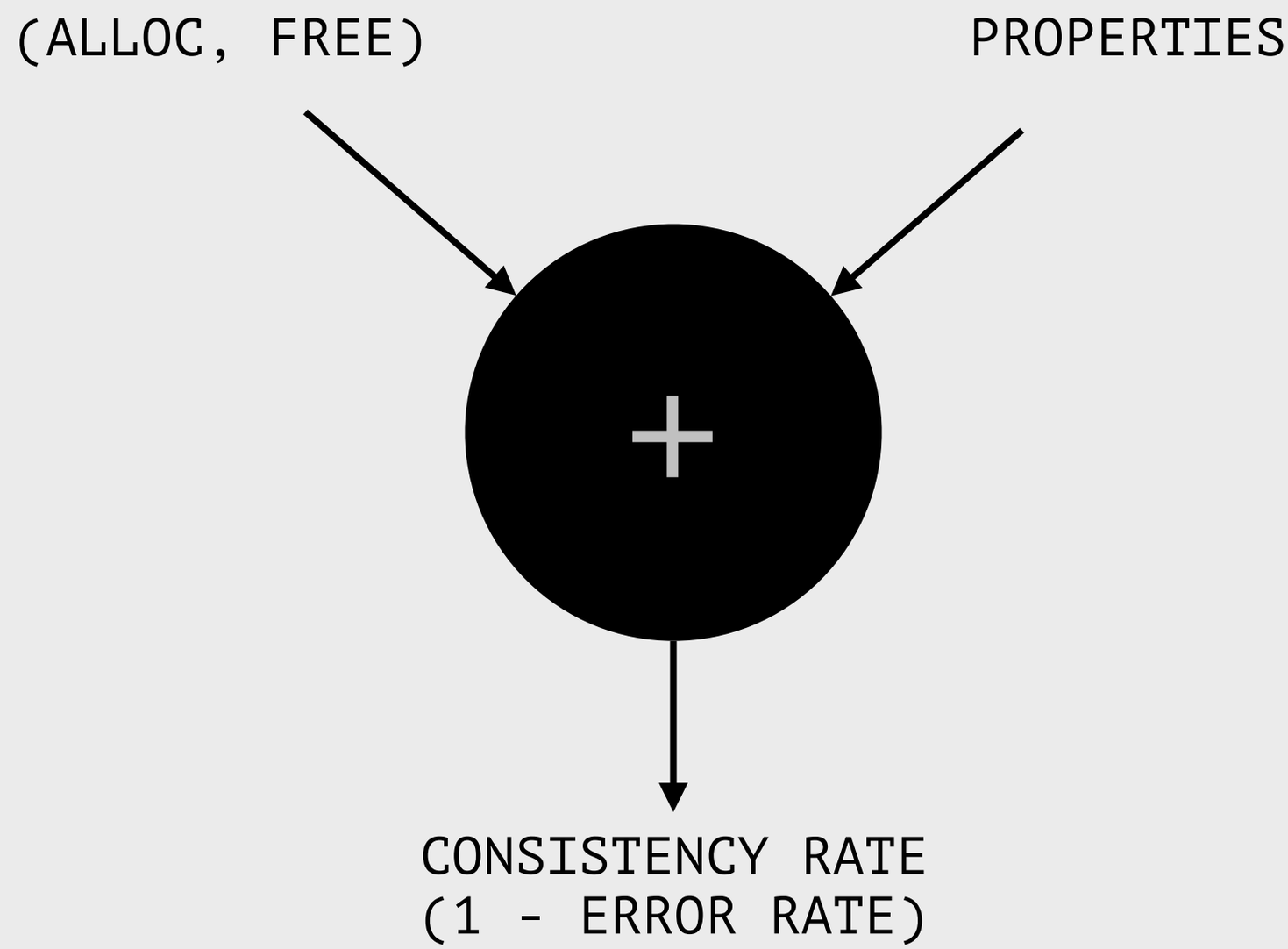

```
ALLOC_0, (FREE_00, FREE_01, …)
ALLOC_1, (FREE_10, FREE_11, …)
                …
ALLOC_N, (FREE_N0, FREE_N1, …)
```

STEP #2

evaluate consistency

STEP #2 - evaluate consistency

overview | properties | error rate



STEP #2 - evaluate consistency

[**ALLOC**]

ALLOC should not output an allocated address

[**FREE**]

FREE should only occur on allocated addresses

STEP #2 - evaluate consistency

overview | properties | error rate

ERRORS = ERROR(ALLOC) + ERROR(FREE)

ERROR RATE = ERRORS / (CALLS(ALLOC) + CALLS(FREE))

ERROR RATE < 0.05 => likely an allocator

STEP #2 - evaluate consistency

2 sources of error

- **reallocation**

- misimplementation (missing calls and returns)

PRELIMINARY RESULTS

It’s time to know if it really works…

Preliminary results

retrieve ALLOC/FREE | consistency rate

ALLOCATOR	EXPERIMENTAL STATE
standard libc (malloc, free)	reasonnable
LD_PRELOAD custom	reasonnable
embedded custom	early stage

Preliminary results

retrieve ALLOC/FREE

consistency rate

program	time (in s)	ALLOC/FREE	error rate
mupdf	12.91	✓/✓	8.22×10^{-2}
pdflatex	56.26	✓/✓	5.98×10^{-4}
readelf	3.21	✓/✓	3.60×10^{-4}
base64	0.882	✓/✓	0.00
cat	0.372	✓/✓	0.00
cp	2.852	✓/✓	3.22×10^{-4}
head	0.477	✓/✓	0.00
id	0.611	✓/✓	0.00
paste	0.930	✓/✓	0.00
pinky	0.631	✓/✓	0.00
tail	0.478	✓/✓	0.00
uname	0.388	✓/×	3.89×10^{-1}

Table 1: Detection of ALLOC and FREE on mupdf, pdflatex, readelf and several coreutils programs which use libc allocator

Preliminary results

retrieve ALLOC/FREE

consistency rate

	ALLOC/FREE	error rate	online (in s)	offline (in s)
b2sum	✓/✓	0	0.474	0.209/0.219
base32	✓/✓	0	0.561	0.121/0.119
base64	✓/✓	0	0.529	0.123/0.361
cat	✓/✓	0	0.48	0.177/0.199
chgrp	✓/✓	0.000397	0.521	0.143/0.132
chmod	✓/✓	0.000408	0.389	0.154/0.169
chown	✓/✓	5.15e-05	2.44	8.58/9.24
comm	✓/×	0.724	0.43	0.723/0.336
cp	✓/✓	0.000936	1.47	0.749/0.553
csplit	✓/✓	0	0.972	0.513/0.881
cut	✓/×	0.753	0.377	0.13/0.19
df	✓/×	0.27	0.677	0.677/0.658
dir	✓/×	0.0227	0.43	0.916/0.12
dircolors	✓/✓	0	0.407	0.244/0.291
du	✓/✓	0.000446	0.706	0.598/0.596
fmt	✓/✓	0	0.483	0.496/0.605
fold	✓/✓	0	0.569	0.129/0.134
ginstall	✓/✓	0	2.35	0.529/0.54
groups	✓/×	0.743	0.511	0.296/0.375
head	✓/✓	0	0.545	0.14/0.408
id	✓/×	0.743	0.533	0.282/0.373
join	✓/×	0.5	0.487	0.483/0.329
logname	✓/✓	0	0.657	0.314/0.439
ls	✓/×	0.0495	0.854	0.662/0.59
pinky	✓/✓	0	0.62	0.179/0.194
pr	✓/✓	0	0.652	0.504/0.289
rm	✓/✓	0.000198	20.5	0.268/0.268
shred	✓/✓	0	1.68	0.344/0.527
shuf	✓/×	0.949	0.431	0.109/0.179
sort	✓/✓	0	0.682	0.397/0.243
split	✓/✓	0	0.542	0.153/0.141
stat	✓/✓	0	0.791	0.203/0.198

Preliminary results

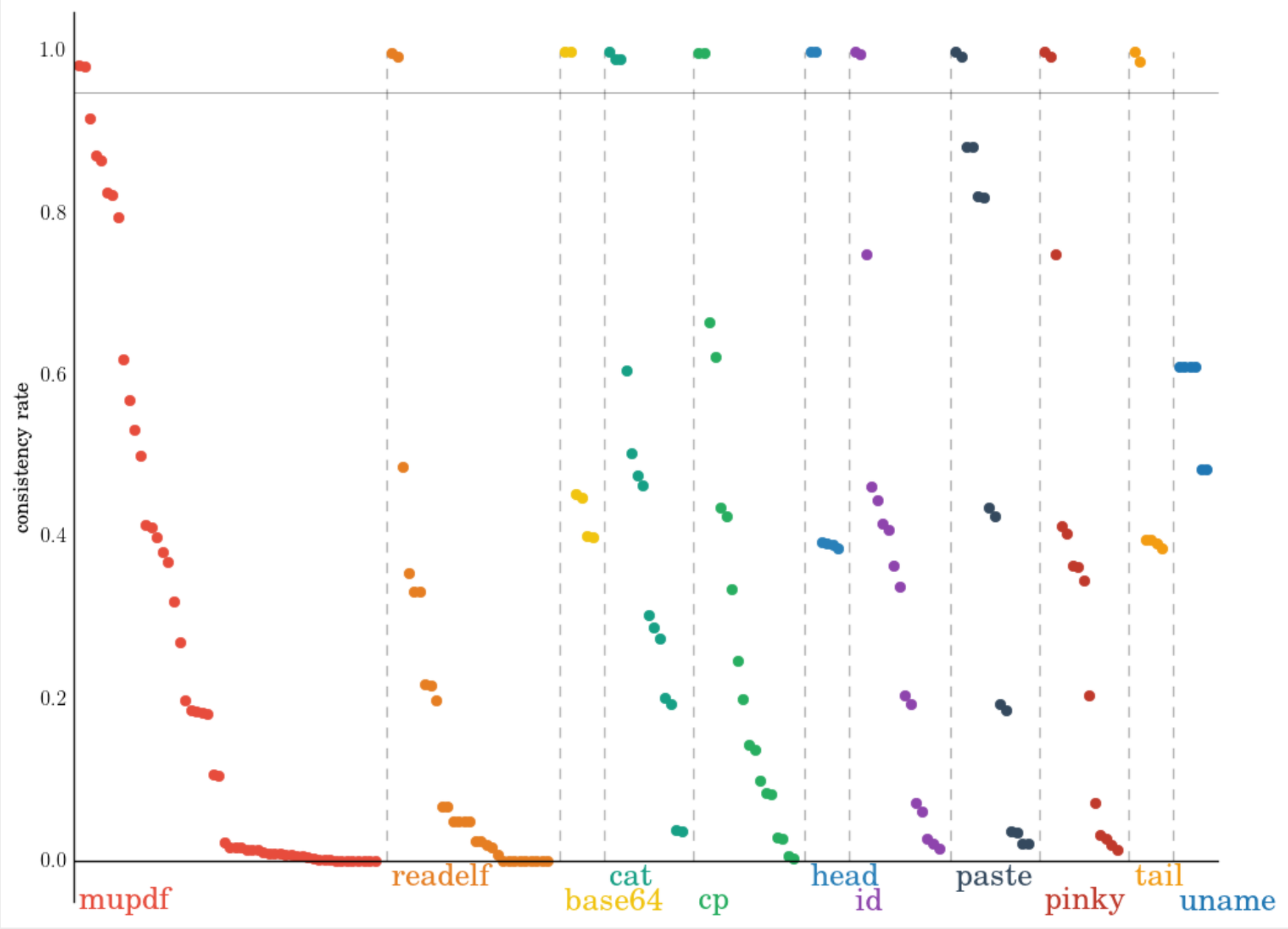
retrieve ALLOC/FREE | consistency rate

PROGRAM	ALLOC/FREE
jasper	ok/ok
openssl	ok/ok
jansson	ok/ok
git	n.c./n.c.
opusenc	ok/ok

Preliminary results

retrieve ALLOC/FREE

consistency rate



Preliminary results

retrieve ALLOC/FREE

consistency rate

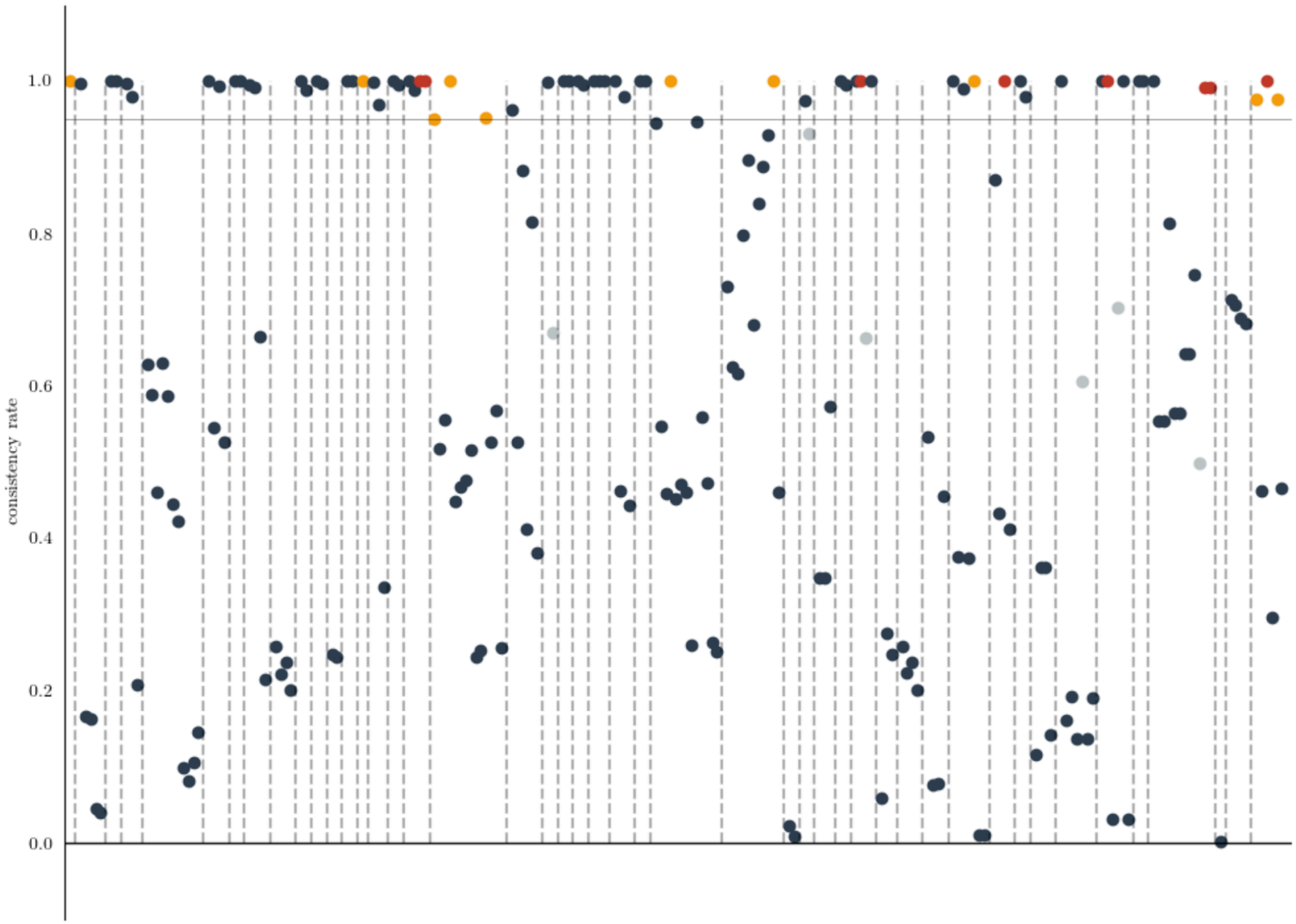


Figure 9.7: Relevancy of the consistency rate to detect wrong allocators

LAST WORDS

one or two more things…

Last words

contributions | limitations | future work

[**detection**]

> a lightweight heuristic-based **passive** approach to retrieve ALLOC/FREE?

[**metrics**]

> a consistency criterion to evaluate a potential allocator

[**open-source**]

> both code and experimental sets are available on GH

Last words

contributions | limitations | future work

[**benchmark**]

> more experiments are required

PB: oracle/ground truth?

[**hypothesis**]

> the main one: asumption on ALLOC & FREE prototypes

[**multiple allocators**]

> retrieve all/retrieve a hierarchy

Last words

contributions | limitations | future work

[**validation**]

> more experiments, especially custom allocators

[**large set of binaries**]

> automated analysis of a large set of binaries

[**use cases**]

> memory use/vulnerability detections/etc.

Metrics for runtime detection of allocators in binaries

August 14th, 2017

Franck de Goër - Roland Groz - Laurent Mounier

Grenoble, FRANCE