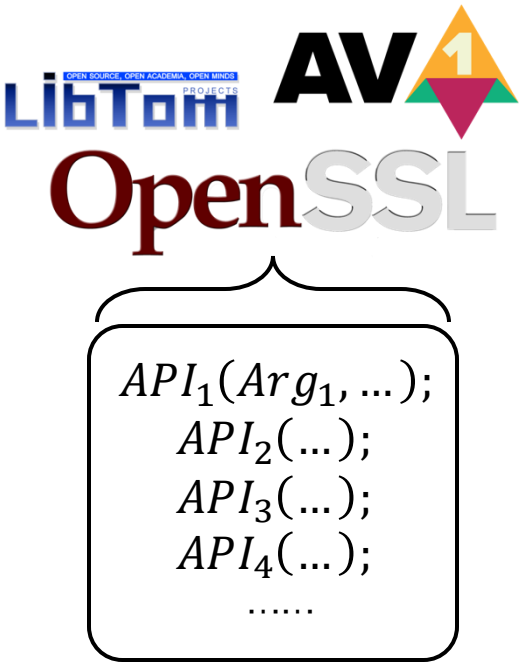
Automatic Library Fuzzing through API Relation Evolvement

Jiayi Lin*, Qingyu Zhang*, Junzhe Li*, Chenxin Sun*, Hao Zhou^, Changhua Luo*, Chenxiong Qian*

*The University of Hong Kong, ^The Hong Kong Polytechnic University

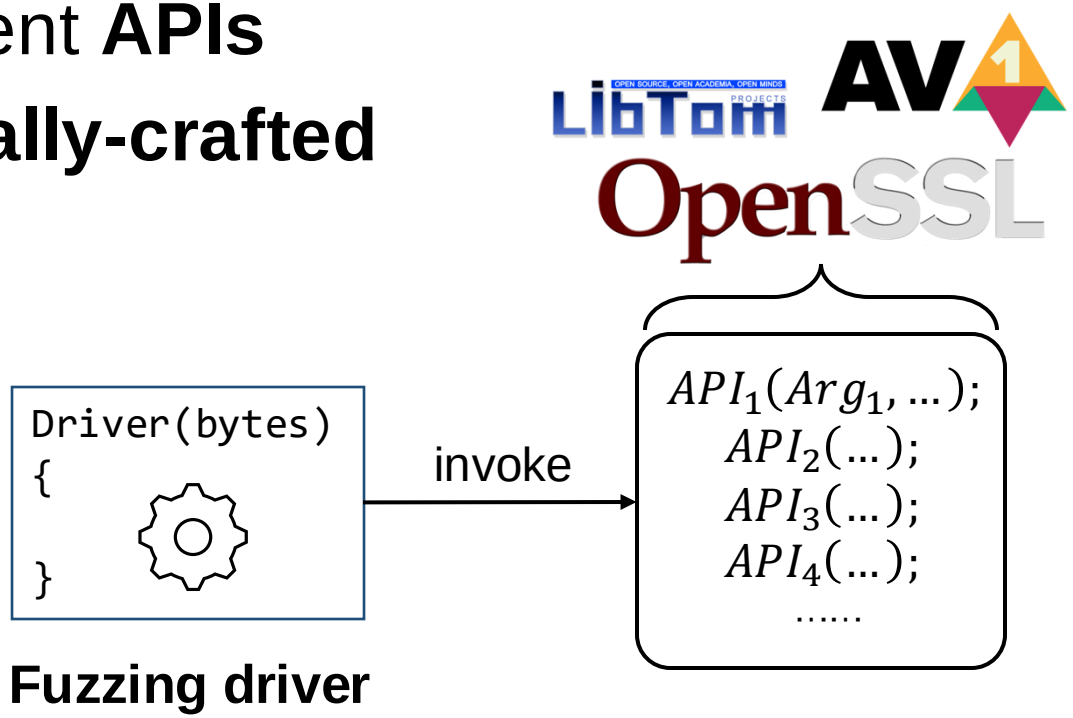
Background – Library Fuzzing

- **Libraries** expose interdependent **APIs**



Background – Library Fuzzing

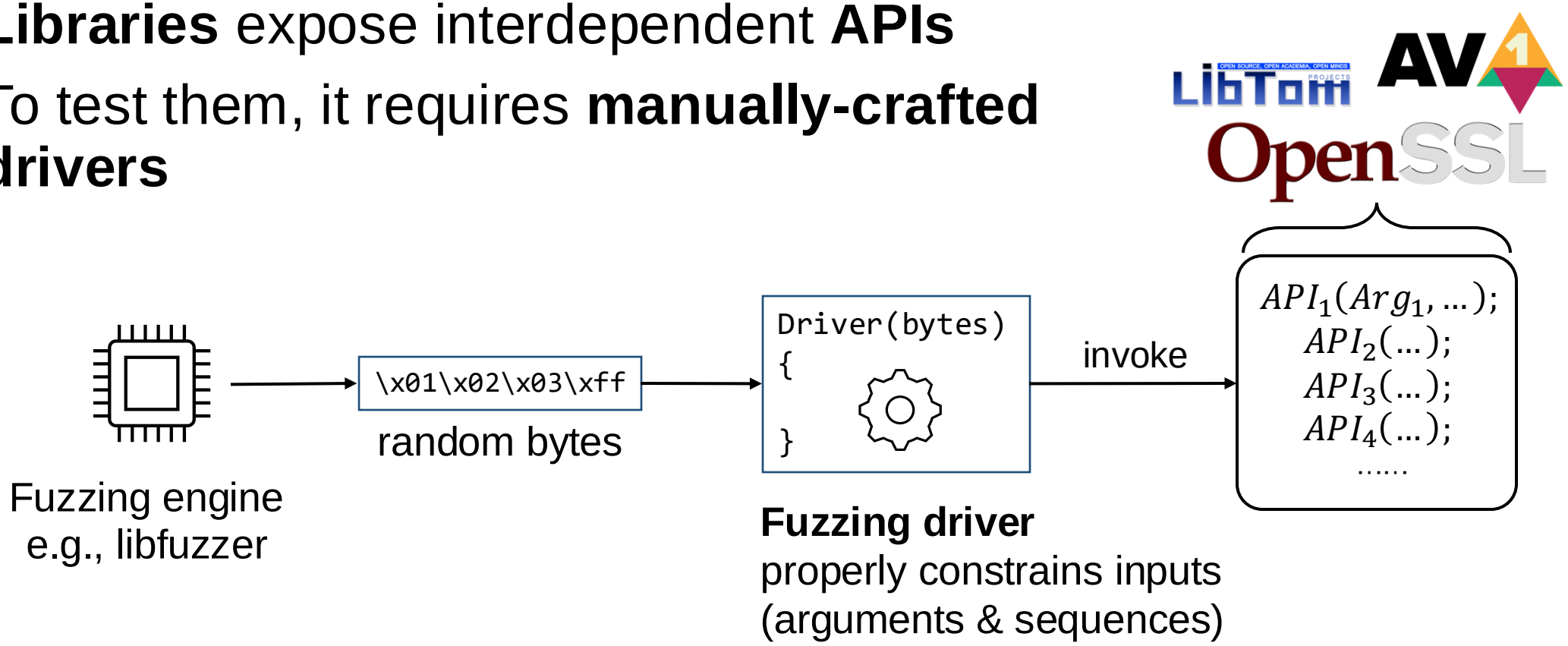
- **Libraries** expose interdependent **APIs**
- To test them, it requires **manually-crafted drivers**



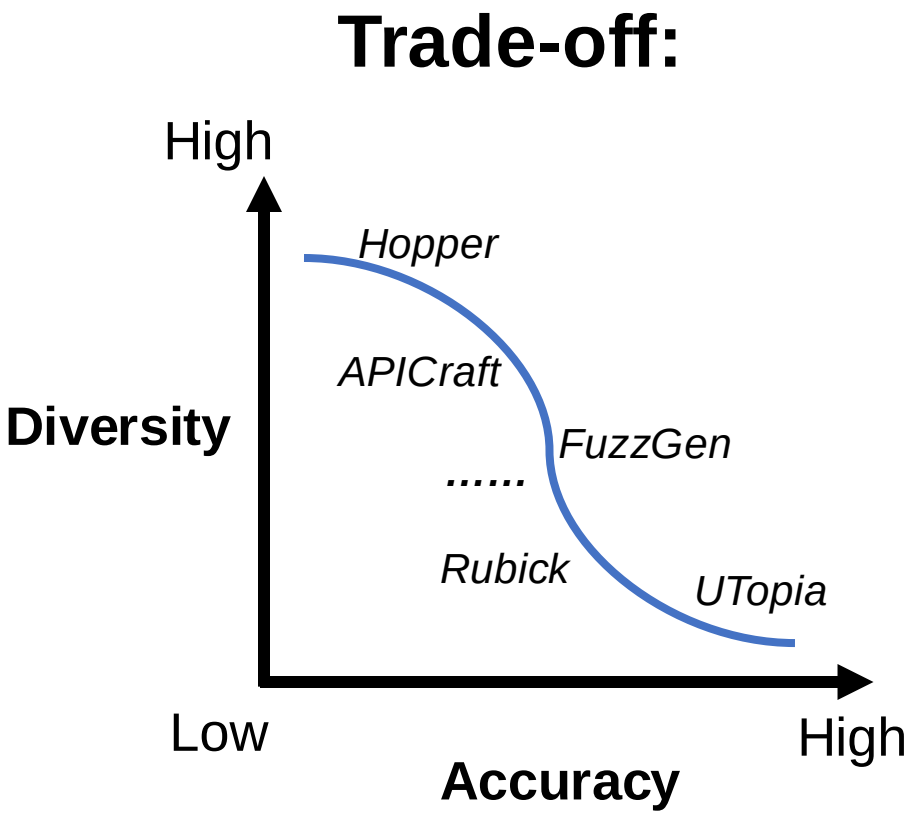
Fuzzing driver

Background – Library Fuzzing

- **Libraries** expose interdependent **APIs**
- To test them, it requires **manually-crafted drivers**



Existing **Automatic** Library Fuzzing



Existing **Automatic** Library Fuzzing

Input Diversity:

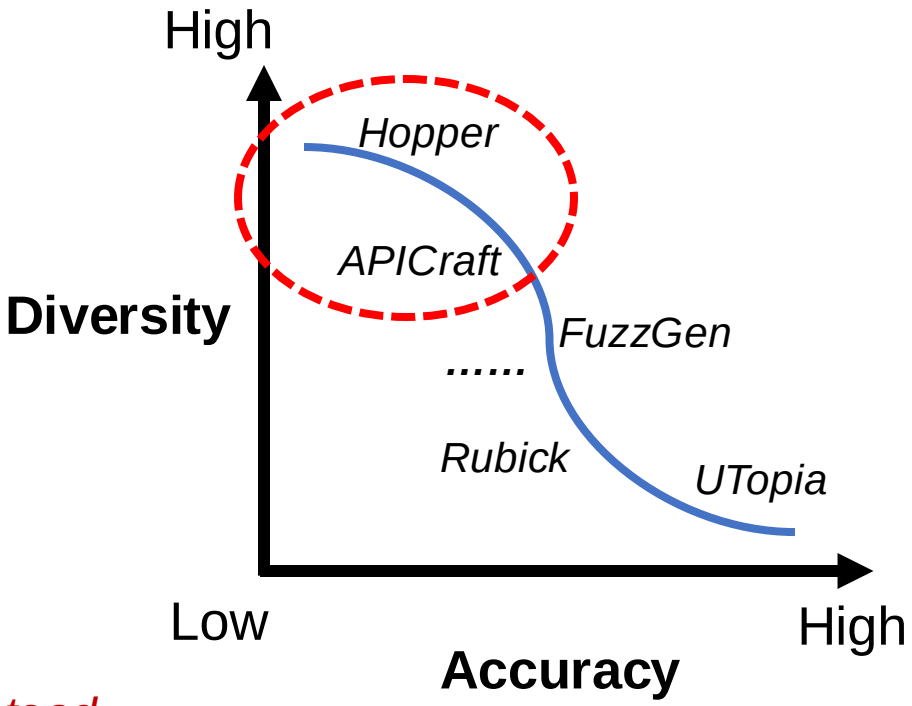
- Wider argument value ranges
- Mutable API sequence
- More API coverage

- Erroneous argument / sequence usage

*Crashing due to **API misuse** instead of real bugs*

e.g., `memcpy(a, b, 0x12341234)`

Trade-off:



Existing **Automatic** Library Fuzzing

Trade-off:

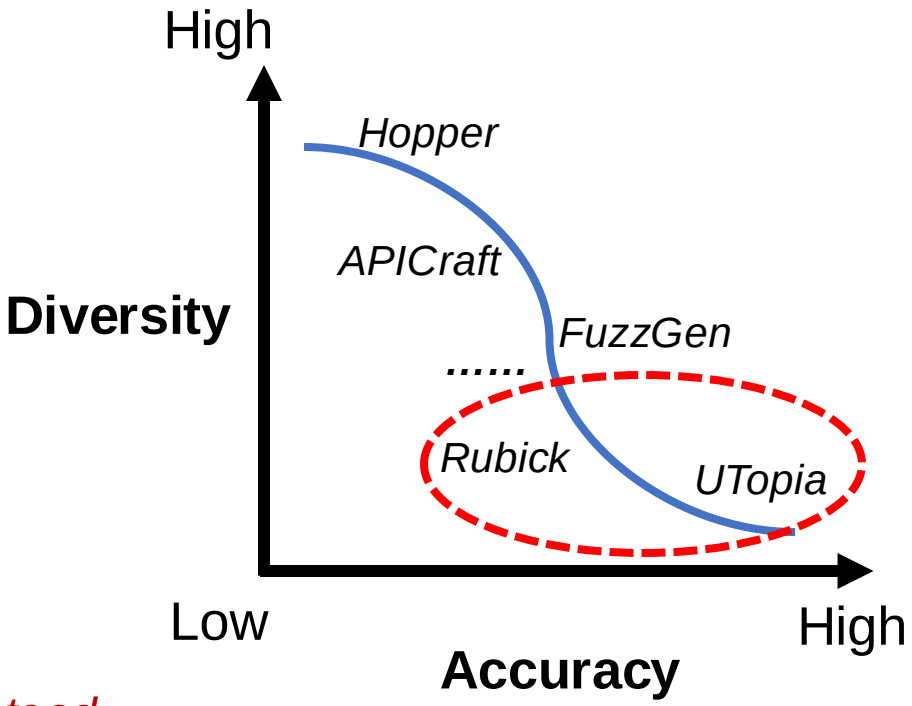
Input Diversity:

- Wider argument value ranges
- Mutable API sequence
- More API coverage

- Erroneous argument / sequence usage

*Crashing due to **API misuse** instead of real bugs*

e.g., `memcpy(a, b, 0x12341234)`



API Usage Accuracy:

- Correct constant / value range usage
- Correct control / data dependencies

- Limited API numbers
- Fixed API sequence

Existing **Automatic** Library Fuzzing

Input Diversity:

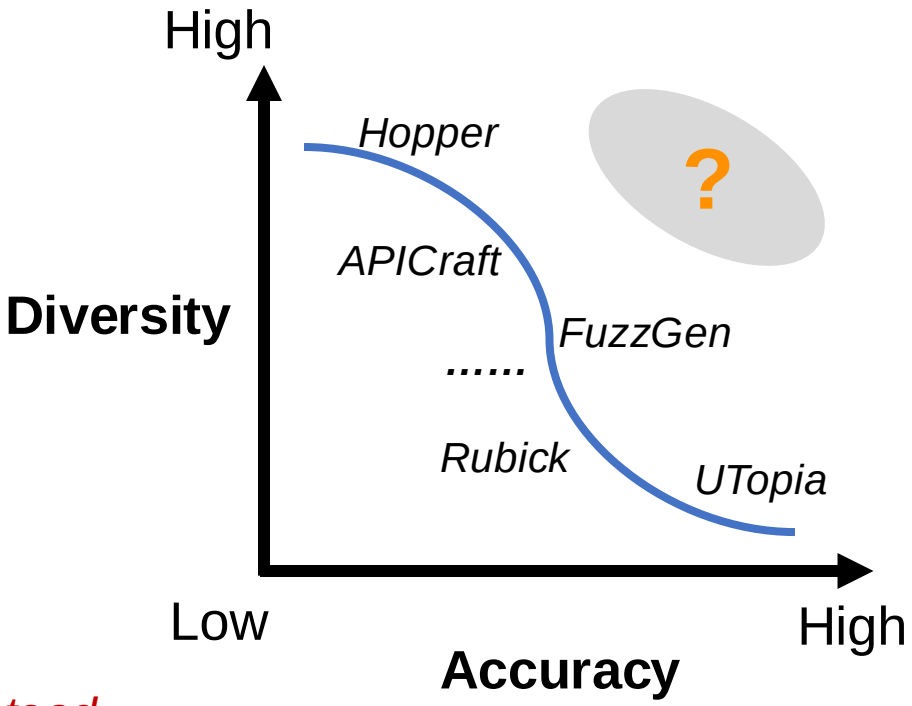
- Wider argument value ranges
- Mutable API sequence
- More API coverage

- Erroneous argument / sequence usage

*Crashing due to **API misuse** instead of real bugs*

e.g., `memcpy(a, b, 0x12341234)`

Trade-off:



API Usage Accuracy:

- Correct constant / value range usage
- Correct control / data dependencies

- Limited API numbers
- Fixed API sequence

Our Method – NEXZZER

Challenges:

- Cover wider input space
  - API numbers & value ranges & sequence orders
- Learn accurate API usage
  - argument constraints & sequence dependencies
- Filter API misuse
  - save manual post-processing or triage

Our Method – NEXZZER

Challenges:

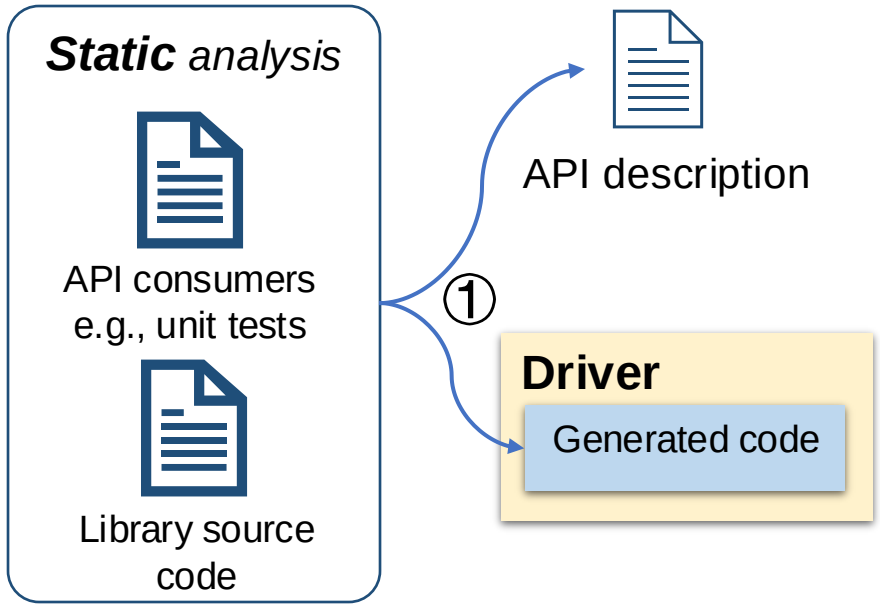
- Cover wider input space
  - API numbers & value ranges & sequence orders
- Learn accurate API usage
  - argument constraints & sequence dependencies
- Filter API misuse
  - save manual post-processing or triage

Solutions:

- A modular & scalable fuzzing driver architecture
- Static analysis & dynamic learning
- Rule-based automatic filtering strategies

Our Method – NEXZZER

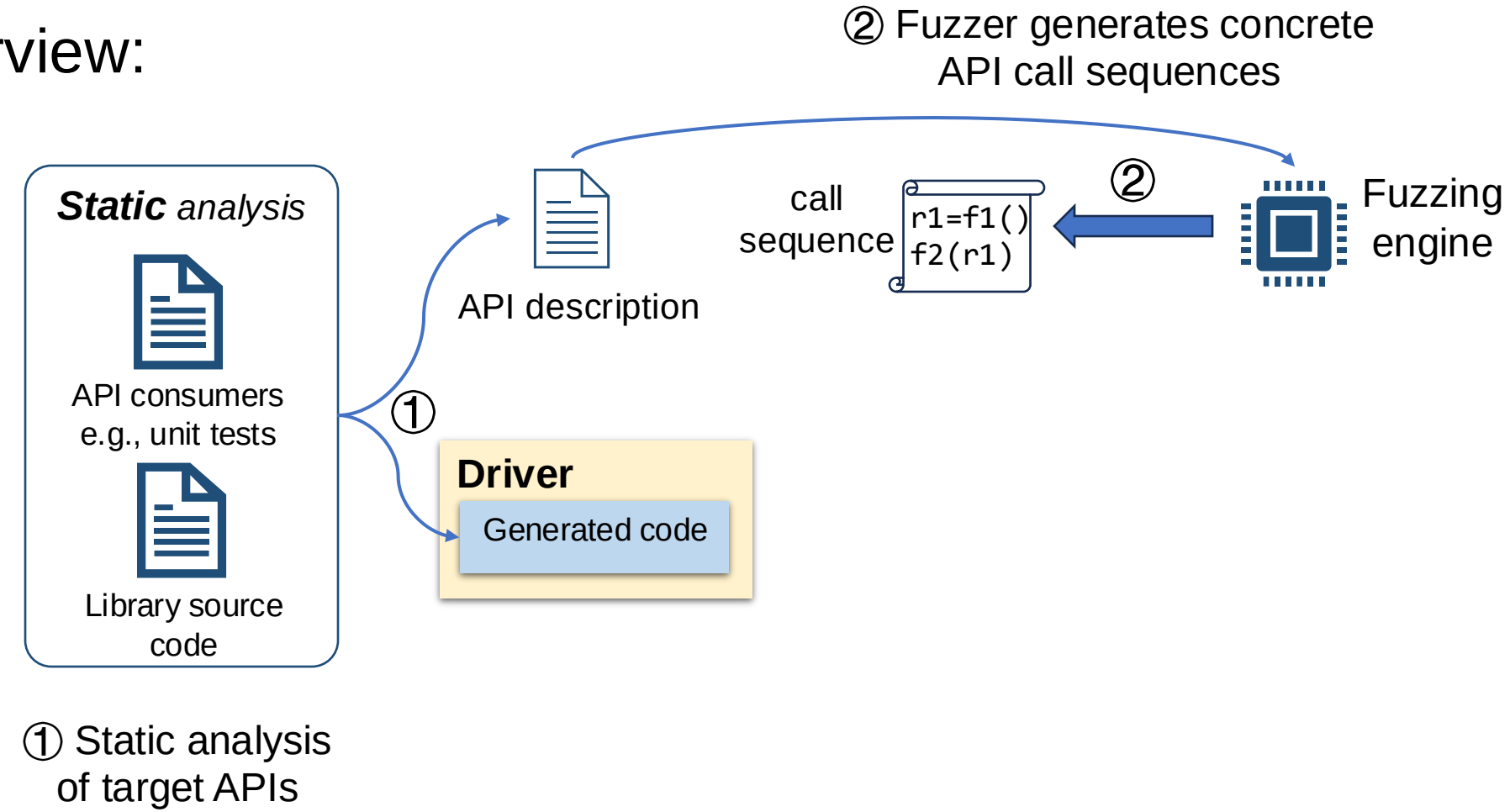
Overview:



① Static analysis of target APIs

Our Method – NEXZZER

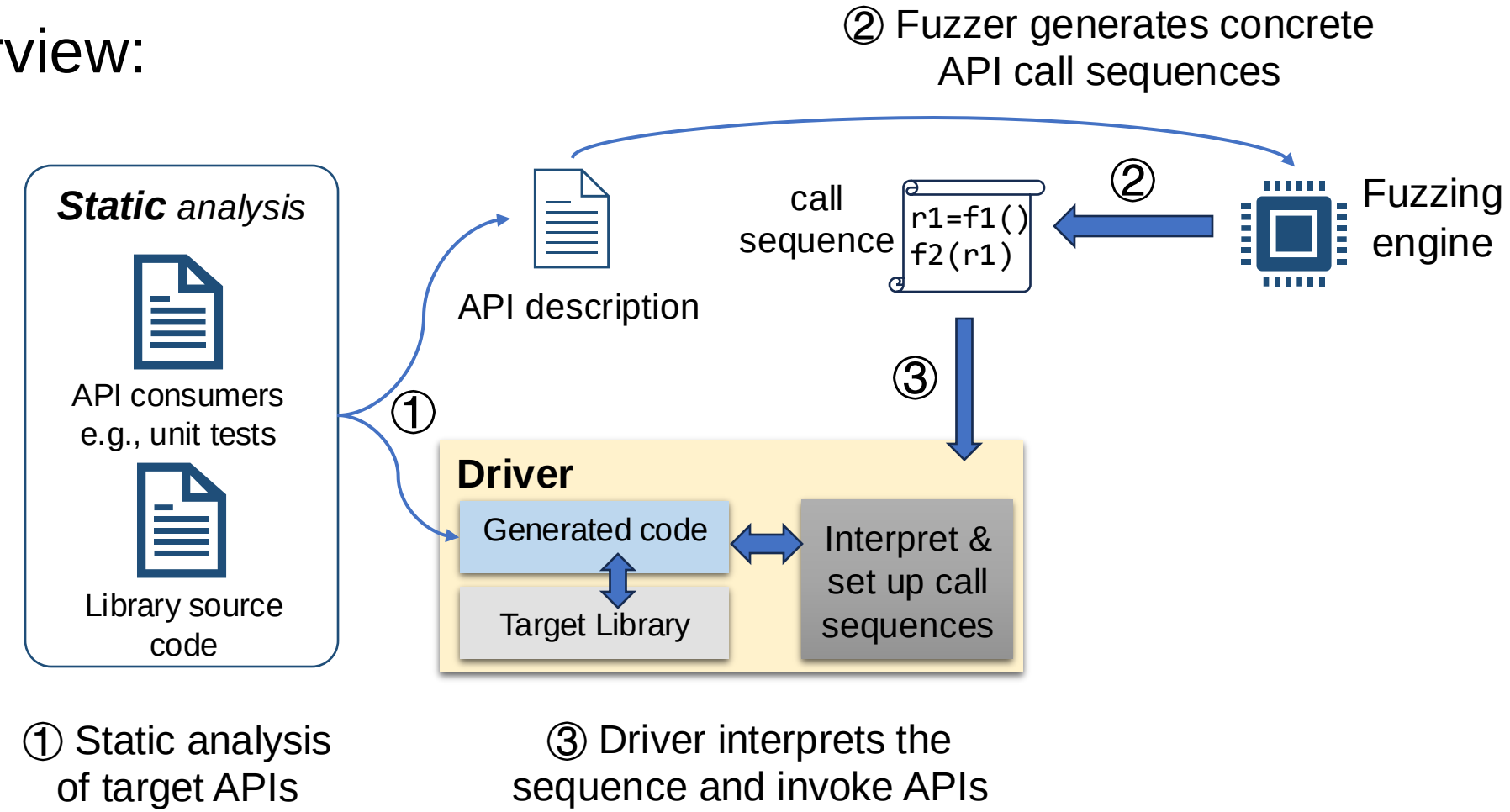
Overview:



① Static analysis
of target APIs

Our Method – NEXZZER

Overview:

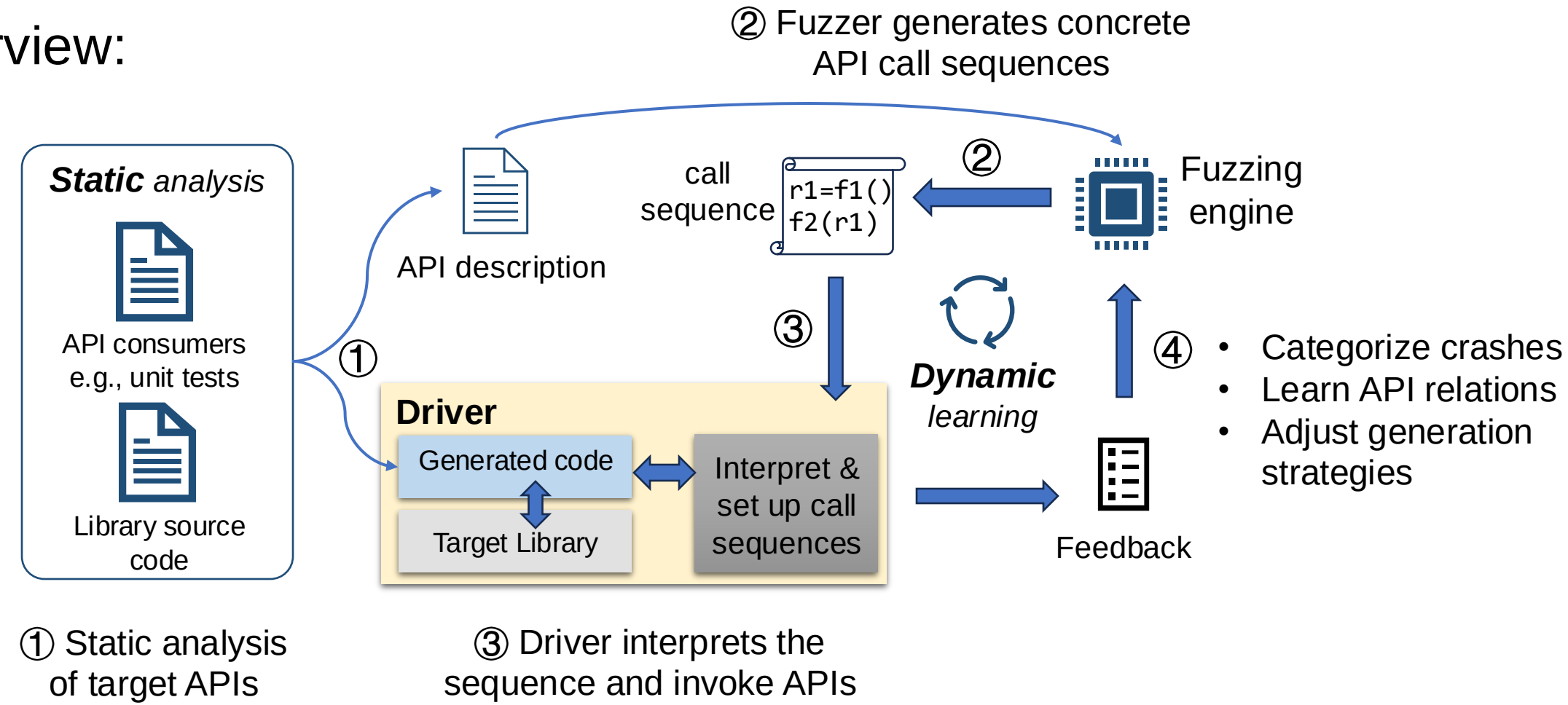


① Static analysis of target APIs

③ Driver interprets the sequence and invoke APIs

Our Method – NEXZZER

Overview:



NEXZZER's Modular Driver Architecture

Monolithic

Tasks:

assign arguments (int, char*, …)

randomize and constrain sequences

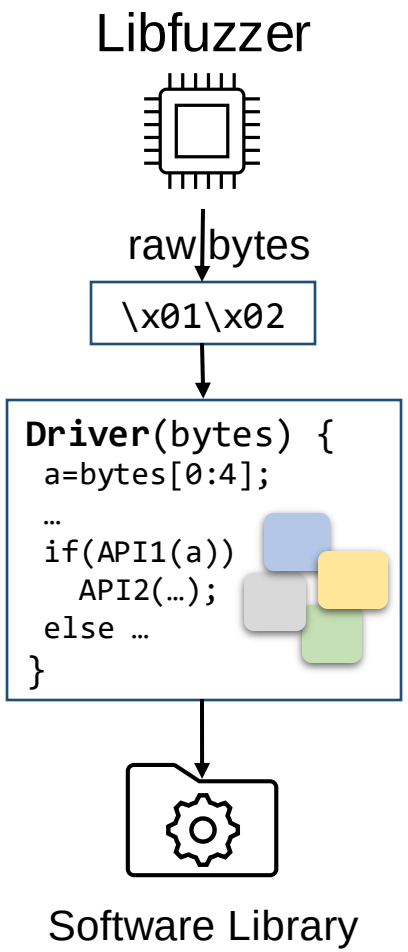
ensure dependencies & signify orders

invoke APIs

Modular

NEXZZER's Modular Driver Architecture

Monolithic



Tasks:

assign arguments (int, char*, …)

randomize and constrain sequences

ensure dependencies & signify orders

invoke APIs

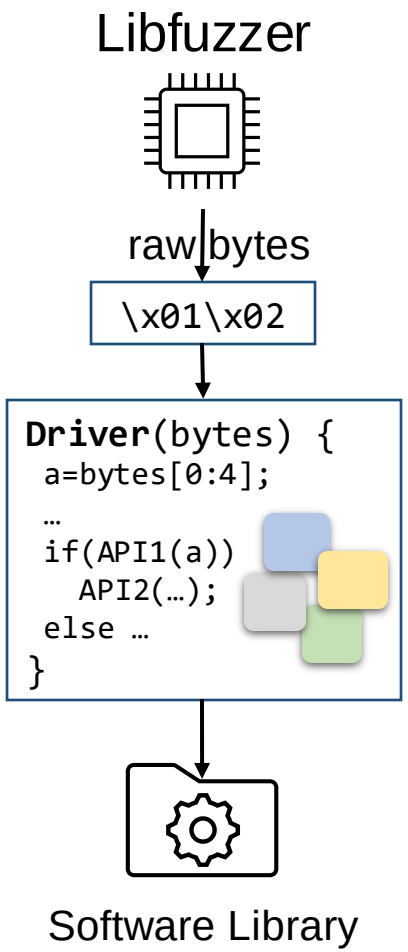
Modular

NEXZZER's Modular Driver Architecture

Synthesized Driver:

- *target-specific*
- *hard to scale or automatize*
- *error-prone*
- *require manual fix*

Monolithic



Tasks:

assign arguments (int, char*, …)

randomize and constrain sequences

ensure dependencies & signify orders

invoke APIs

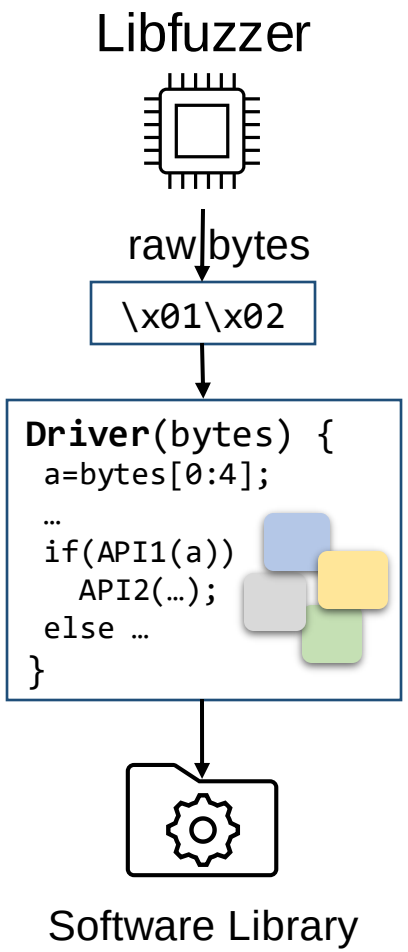
Modular

NEXZZER's Modular Driver Architecture

Synthesized Driver:

- *target-specific*
- *hard to scale or automatize*
- *error-prone*
- *require manual fix*

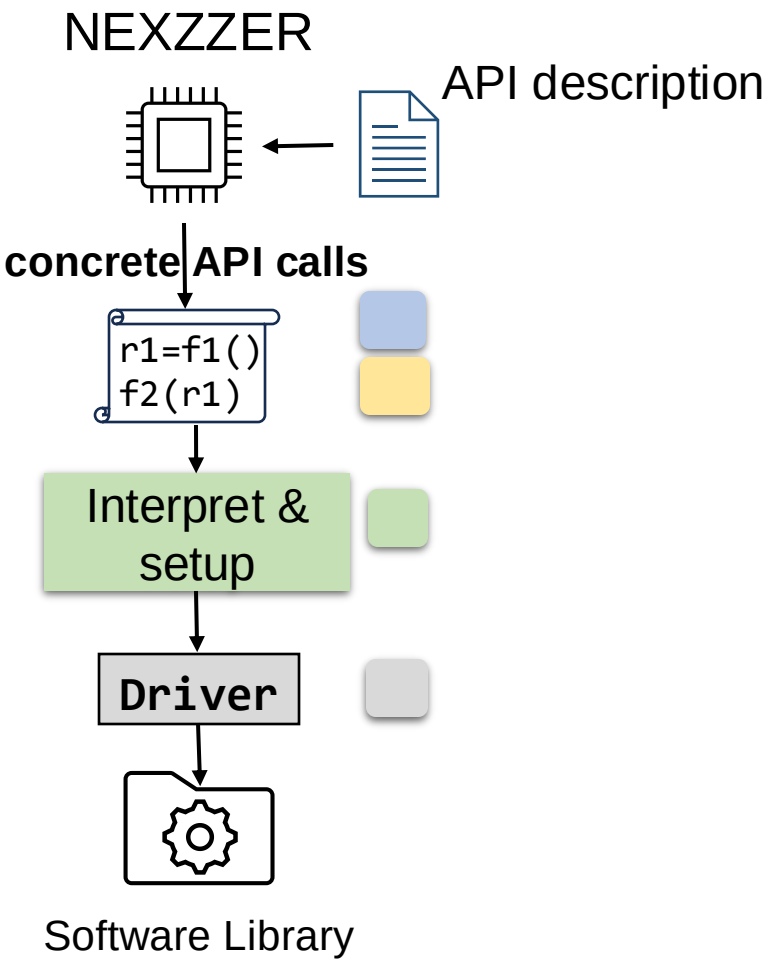
Monolithic



Tasks:

- assign arguments (int, char*, …)
- randomize and constrain sequences
- ensure dependencies & signify orders
- invoke APIs

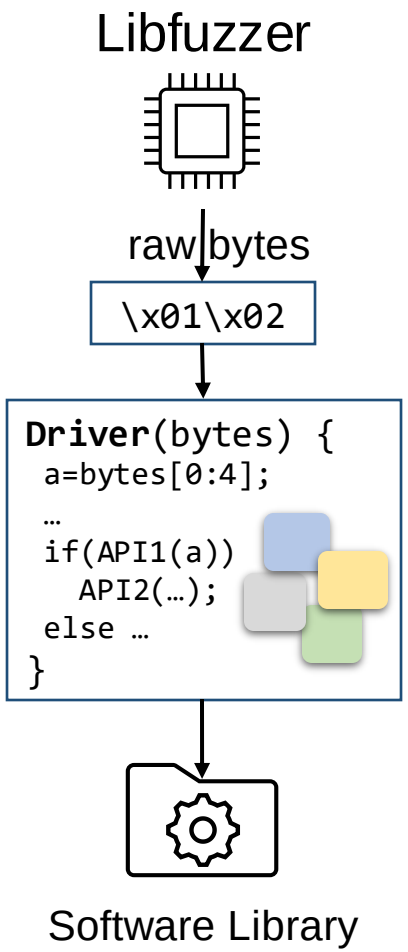
Modular



NEXZZER's Modular Driver Architecture

Synthesized Driver:
- *target-specific*
- *hard to scale or automatize*
- *error-prone*
- *require manual fix*

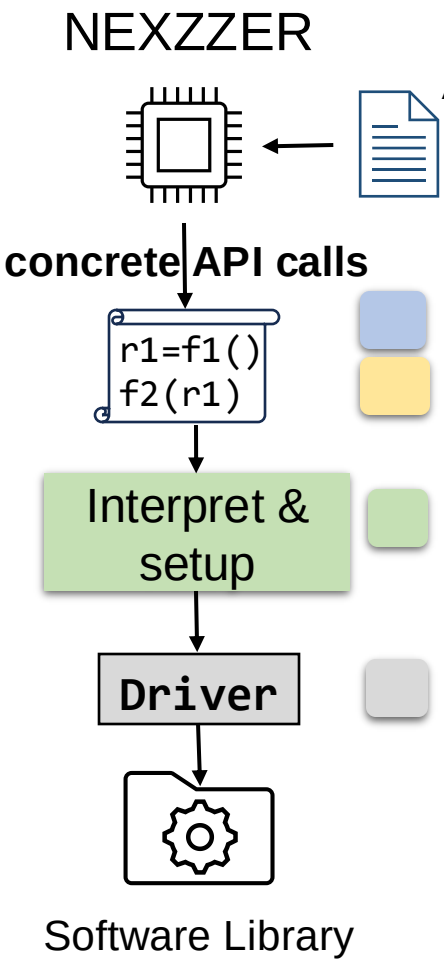
Monolithic



Tasks:

- assign arguments (int, char*, …)
- randomize and constrain sequences
- ensure dependencies & signify orders
- invoke APIs

Modular

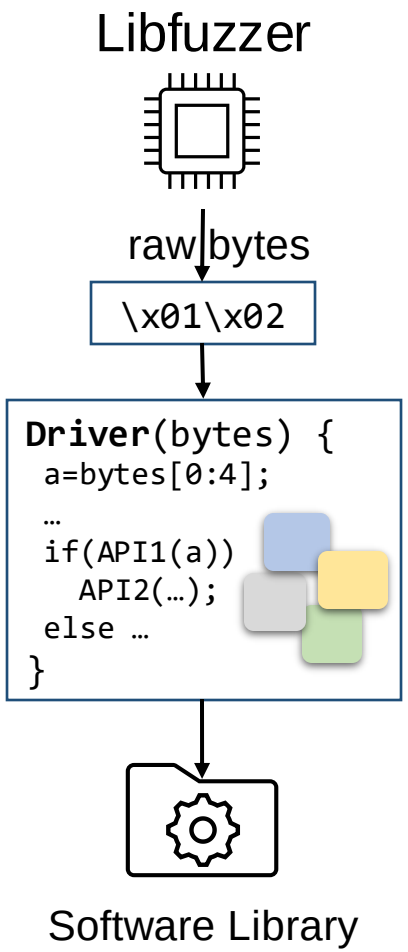


- *easy to automatize*
- *easy to adjust API usage & strategies*

NEXZZER's Modular Driver Architecture

Synthesized Driver:
- *target-specific*
- *hard to scale or automatize*
- *error-prone*
- *require manual fix*

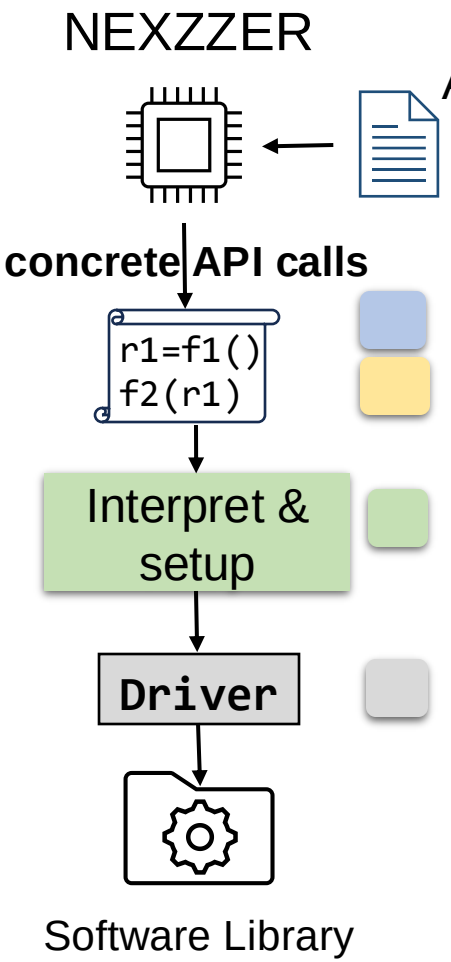
Monolithic



Tasks:

- assign arguments (int, char*, …)
- randomize and constrain sequences
- ensure dependencies & signify orders
- invoke APIs

Modular

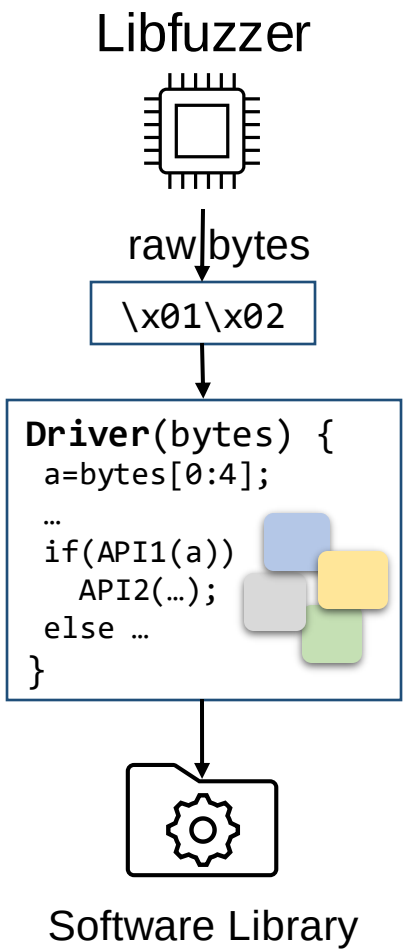


- *easy to automatize*
- *easy to adjust API usage & strategies*
- *target-agnostic*

NEXZZER's Modular Driver Architecture

Synthesized Driver:
- *target-specific*
- *hard to scale or automatize*
- *error-prone*
- *require manual fix*

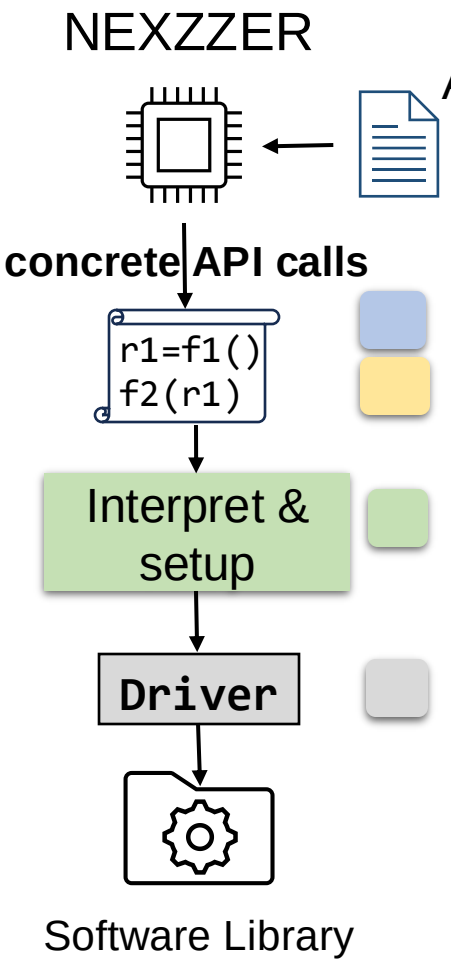
Monolithic



Tasks:

- assign arguments (int, char*, …)
- randomize and constrain sequences
- ensure dependencies & signify orders
- invoke APIs

Modular



- *easy to automatize*
- *easy to adjust API usage & strategies*
- *target-agnostic*

Synthesized Driver:
- *easy to synthesize*
- *scalable*

API Usage Learning

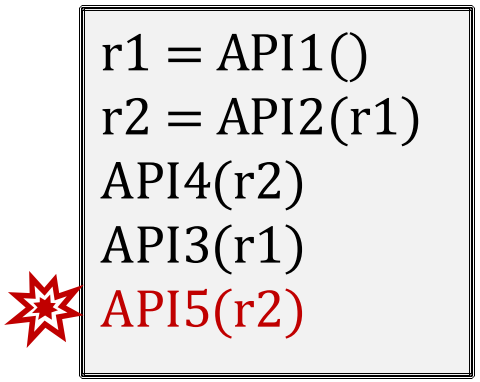
1. Static consumer analysis: backward data-flow slicing to collect attributes
   - constants, dependencies, …

API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - tentatively mutate API arguments to learn constraints
   - tentatively minimize API sequences to learn relations

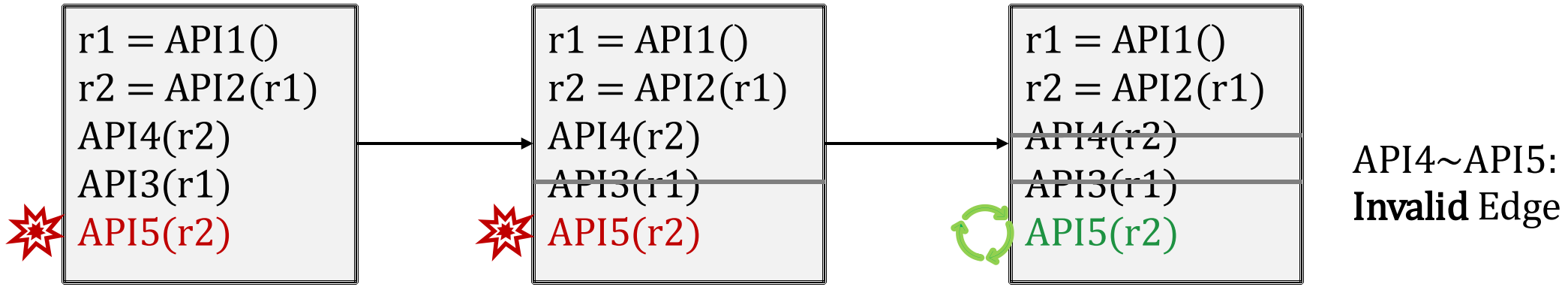
API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - tentatively mutate API arguments to learn constraints
   - tentatively minimize API sequences to learn relations



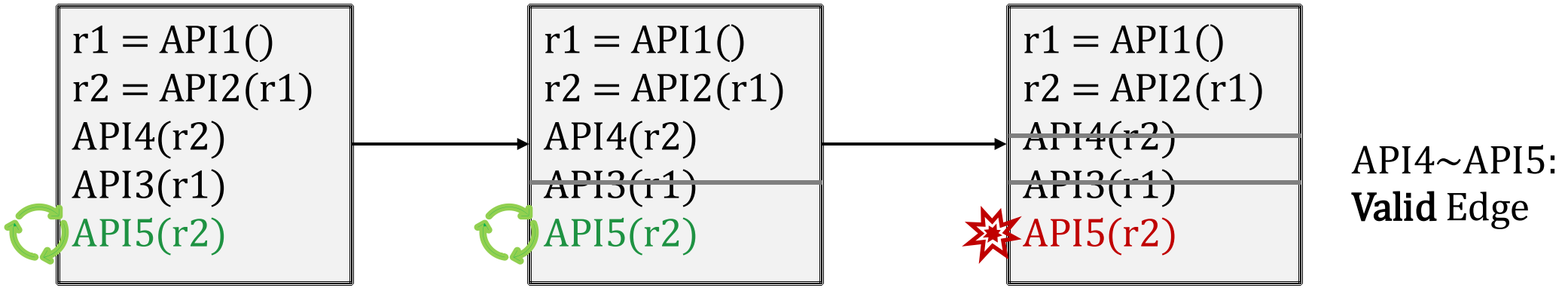
API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - tentatively mutate API arguments to learn constraints
   - tentatively minimize API sequences to learn relations



API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - tentatively mutate API arguments to learn constraints
   - tentatively minimize API sequences to learn relations



API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - e.g.,

```
APIGraph::Invalid_Edges:
  BN_free::arg0       -> BN_add::arg0
  BN_free::arg0       -> BN_sub::arg0
  BN_generate_prime   -> BN_add_word
  ...
APIGraph::Valid_Edges:
  EVP_DecryptInit::arg0      -> EVP_DecryptUpdate::arg0
  RSA_generate_key_ex::arg0 -> RSA_private_decrypt::arg3
```

API Usage Learning

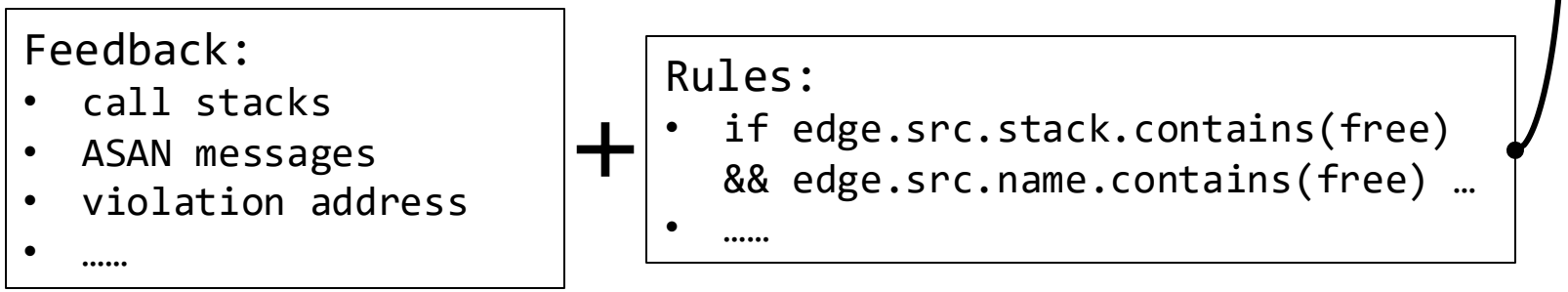
1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - e.g.,
```
APIGraph::Invalid_Edges:
    BN_free::arg0       -> BN_add::arg0
    BN_free::arg0       -> BN_sub::arg0
    BN_generate_prime   -> BN_add_word
    ...
```
3. Heuristic filtering rules to categorize them & update call sequence generation strategies

API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
   - e.g., **APIGraph::Invalid_Edges:**

```
BN_free::arg0      -> BN_add::arg0        Use-After-Free misuse
BN_free::arg0      -> BN_sub::arg0
BN_generate_prime  -> BN_add_word         Suspicious Bug
...
```

3. Heuristic filtering rules to categorize them & update call sequence generation strategies



API Usage Learning

1. Static consumer analysis: backward data-flow slicing to collect attributes
2. Dynamic learning
3. Heuristic filtering rules to categorize them & update call sequence generation strategies

- *Please refer to the paper for more technical details:*
  - *learning argument constraints*
  - *concrete filtering rules*

Evaluation

- RQ1: How effective is NEXZZER when comparing with existing works
- RQ2: How effective is NEXZZER’s different components

Evaluation – RQ1

- Among 18 libraries, NEXZZER found 27 new bugs within 24 hours

Targets	Type	Function	Status	ID
OpenSSL	Integer Overflow	BN_mod_exp_mont_consttime	Fixed	4378e
OpenSSL	Integer Overflow	BN_bntest_rand	Fixed	23704
OpenSSL	Integer Overflow	ASN1_BIT_STRING_set_bit	Fixed	20719
OpenSSL	Integer Overflow	ASN1_BIT_STRING_get_bit	Fixed	20719
OpenSSL	Undefined Behavior	BN_GF2m_mod_inv	Reported	19826
OpenSSL	Integer Overflow	RC2_ofb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	RC2_cfb64_encrypt	Confirmed	22986
OpenSSL	Integer Overflow	TXT_DB_create_index	Confirmed	22986
OpenSSL	Integer Overflow	TXT_DB_get_by_index	Confirmed	22986
OpenSSL	Stack Overflow	CAST_ofb64_encrypt	Confirmed	22986
OpenSSL	Integer Overflow	CAST_cfb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	DES_ede3_ofb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	DES_ofb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	DES_cfb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	DES_ede3_cfb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	RC5_32_ofb64_encrypt	Confirmed	22986
OpenSSL	Stack Overflow	RSA_padding_add_PKCS1_type_1	Confirmed	22986
OpenSSL	Stack Overflow	BF_ofb64_encrypt	Confirmed	22986
libxml2	Null Pointer Dereference	xmlCopyNode	Confirmed	463
Relic	Integer Overflow	bn_grow	Fixed	CVE-2023-36326
Relic	Heap Overflow	bn_get_prime	Fixed	CVE-2023-36327
libTom	Integer Overflow	mp_grow	Fixed	CVE-2023-36328
libpcre2	Heap Overflow	pcre2_match_data_create	Fixed	CVE-2023-29822
libGMP	Integer Overflow	mpz_nextprime	Fixed	CVE-2022-46386
lcms	Buffer Overflow	cmsCreateExtendedTransform	Fixed	46355
libopus	Stack Overflow	opus_decode	Reported	329
tesseract	Integer Overflow	TessBaseAPIAdaptToWordStr	Reported	4299

Evaluation – RQ1

- Comparison with existing works: OSS-Fuzz, FuzzGen, UTopia, Hopper

	Fuzzers	OpenSSL	tesseract	jsonnet	leveldb	uriparser	libvpx	libaom	libTom	libGMP	relic	libxml2	libpcre2	lcms	cJSON	libopux	libgsm	libavc	libhevc
#UAPI	BASELINE	606	9	5	15	17	17	13	33	59	54	59	14	10	6	4	5	1	1
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	5	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	12	7	1	1
	UTOPIA	N/A	356	6	91	24	43	109	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
	NEXZZER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
Code Coverage	BASELINE	13215(4%)	17013(9%)	1408(6%)	2247(8%)	7351(4%)	**2592(5%)**	**8329(8%)**	776(4%)	2817(6%)	2388(3%)	9541(8%)	**7737(4%)**	2222(3%)	621(1%)	1332(2%)	213(1%)	**4525(4%)**	**5658(4%)**
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	1378(5%)	6028(6%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1033(1%)	209(3%)	4119(3%)	3574(3%)
	UTOPIA	N/A	3951(2%)	1825(4%)	**2668(3%)**	7465(2%)	1728(6%)	7047(4%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	17989(15%)	19400(10%)	2037(11%)	1000(10%)	7312(5%)	1461(8%)	7202(9%)	1142(4%)	5872(4%)	3490(6%)	5718(10%)	6544(10%)	1701(2%)	863(2%)	**4418(6%)**	312(0%)	33(0%)	21(0%)
	NEXZZER	**26984(9%)**	**20315(12%)**	**3434(5%)**	1877(9%)	**7835(4%)**	2467(13%)	7381(13%)	**1326(6%)**	**6450(4%)**	**4192(5%)**	**14195(9%)**	6825(7%)	**3306(3%)**	**865(2%)**	4021(4%)	**324(0%)**	4202(5%)	4733(4%)
#UVul	BASELINE	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0
	UTOPIA	N/A	1	0	0	0	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	2	0	0	0	0	0	0	0	0	0	0	1	1	**1**	0	0	0	0
	NEXZZER	**18**	**1**	0	0	0	0	0	**1**	**1**	**2**	**1**	**1**	**1**	0	**1**	0	0	0

Evaluation – RQ1

- Comparison with existing works: OSS-Fuzz, FuzzGen, UTopia, Hopper

	Fuzzers	OpenSSL	tesseract	jsonnet	leveldb	uriparser	libvpx	libaom	libTom	libGMP	relic	libxml2	libpcre2	lcms	cJSON	libopux	libgsm	libavc	libhevc
#UAPI	BASELINE	606	9	5	15	17	17	13	33	59	54	59	14	10	6	4	5	1	1
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	5	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	12	7	1	1
	UTOPIA	N/A	356	6	91	24	43	109	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
	NEXZZER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
Code Coverage	BASELINE	13215(4%)	17013(9%)	1408(6%)	2247(8%)	7351(4%)	**2592(5%)**	**8329(8%)**	776(4%)	2817(6%)	2388(3%)	9541(8%)	**7737(4%)**	2222(3%)	621(1%)	1332(2%)	213(1%)	**4525(4%)**	**5658(4%)**
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	1378(5%)	6028(6%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1033(1%)	209(3%)	4119(3%)	3574(3%)
	UTOPIA	N/A	3951(2%)	1825(4%)	**2668(3%)**	7465(2%)	1728(6%)	7047(4%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	17989(15%)	19400(10%)	2037(11%)	1000(10%)	7312(5%)	1461(8%)	7202(9%)	1142(4%)	5872(4%)	3490(6%)	5718(10%)	6544(10%)	1701(2%)	863(2%)	**4418(6%)**	312(0%)	33(0%)	21(0%)
	NEXZZER	**26984(9%)**	**20315(12%)**	**3434(5%)**	1877(9%)	**7835(4%)**	2467(13%)	7381(13%)	**1326(6%)**	**6450(4%)**	**4192(5%)**	**14195(9%)**	6825(7%)	**3306(3%)**	**865(2%)**	4021(4%)	**324(0%)**	4202(5%)	4733(4%
#UVul	BASELINE	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0
	UTOPIA	N/A	1	0	0	0	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	2	0	0	0	0	0	0	0	0	0	0	1	1	**1**	0	0	0	0
	NEXZZER	**18**	**1**	0	0	0	0	0	**1**	**1**	**2**	**1**	**1**	**1**	0	**1**	0	0	0

- **~48%** more code coverage

Evaluation – RQ1

- Comparison with existing works: OSS-Fuzz, FuzzGen, UTopia, Hopper

	Fuzzers	OpenSSL	tesseract	jsonnet	leveldb	uriparser	libvpx	libaom	libTom	libGMP	relic	libxml2	libpcre2	lcms	cJSON	libopux	libgsm	libavc	libhevc
#UAPI	BASELINE	606	9	5	15	17	17	13	33	59	54	59	14	10	6	4	5	1	1
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	5	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	12	7	1	1
	UTOPIA	N/A	356	6	91	24	43	109	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
	NEXZZER	3834	138	46	68	90	24	35	93	594	270	114	27	274	78	54	36	1	1
Code Coverage	BASELINE	13215(4%)	17013(9%)	1408(6%)	2247(8%)	7351(4%)	**2592(5%)**	**8329(8%)**	776(4%)	2817(6%)	2388(3%)	9541(8%)	**7737(4%)**	2222(3%)	621(1%)	1332(2%)	213(1%)	**4525(4%)**	**5658(4%)**
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	1378(5%)	6028(6%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1033(1%)	209(3%)	4119(3%)	3574(3%)
	UTOPIA	N/A	3951(2%)	1825(4%)	**2668(3%)**	7465(2%)	1728(6%)	7047(4%)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	17989(15%)	19400(10%)	2037(11%)	1000(10%)	7312(5%)	1461(8%)	7202(9%)	1142(4%)	5872(4%)	3490(6%)	5718(10%)	6544(10%)	1701(2%)	863(2%)	**4418(6%)**	312(0%)	33(0%)	21(0%)
	NEXZZER	**26984(9%)**	**20315(12%)**	**3434(5%)**	1877(9%)	**7835(4%)**	2467(13%)	7381(13%)	**1326(6%)**	**6450(4%)**	**4192(5%)**	**14195(9%)**	6825(7%)	**3306(3%)**	**865(2%)**	4021(4%)	**324(0%)**	4202(5%)	4733(4%)
#UVul	BASELINE	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0
	FUZZGEN	N/A	N/A	N/A	N/A	N/A	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0
	UTOPIA	N/A	1	0	0	0	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	HOPPER	2	0	0	0	0	0	0	0	0	0	0	1	1	**1**	0	0	0	0
	NEXZZER	**18**	**1**	0	0	0	0	0	**1**	**1**	**2**	**1**	**1**	**1**	0	**1**	0	0	0

- **~48%** more code coverage
- **19** more bugs

Evaluation – RQ1

- Comparison with existing works: OSS-Fuzz, FuzzGen, UTopia, Hopper

Targets		NEXZZER										HOPPER				UTopia			
		ValueSet				APIEdge			Unknown Crash			Total	#Inf	#FP	#TP (BUG)	Total	#FP (Driver)	#TP (Driver)	#Bug
		Total	#L	#N	#R	Total	#Mis-DU	#I	Total	#FP	#TP (BUG)								
OpenSSL	P1	236	8	219	9	217	209	8	30	28	2	806	110	696	0	N/A	N/A	N/A	N/A
	P2	255	11	223	21	286	268	18	31	30	1	828	335	493	0	N/A	N/A	N/A	N/A
	P3	236	8	218	10	313	303	10	36	35	1	134	31	103	0	N/A	N/A	N/A	N/A
	P4	203	2	196	5	237	225	12	26	23	3	230	60	170	0	N/A	N/A	N/A	N/A
	P5	210	12	195	3	227	203	24	29	28	1	40	24	16	0	N/A	N/A	N/A	N/A
	P6	84	0	75	9	95	72	23	14	13	1	70	38	32	0	N/A	N/A	N/A	N/A
	P7	233	35	182	16	198	169	29	44	42	9	300	81	217	2	N/A	N/A	N/A	N/A
tesseract		95	6	89	0	47	37	10	29	27	1	86	9	76	1	246	111	135	0
jsonnet		29	0	29	0	16	7	9	4	3	0	22	4	18	0	15	0	15	0
leveldb		53	10	41	2	57	48	9	8	8	0	36	23	13	0	157	71	86	0
uriparser		8	3	3	2	60	49	11	7	7	0	93	46	47	0	80	61	19	0
libvpx		48	0	43	5	59	54	5	2	2	0	9	2	7	0	17	14	3	0
libaom		90	**5**	69	16	30	25	5	15	15	0	10	4	6	0	109	37	72	0
cJSON		73	16	57	0	39	39	0	7	6	0	11	9	1	1	N/A	N/A	N/A	N/A
lcms		206	8	198	0	261	260	1	12	9	1	106	2	103	1	N/A	N/A	N/A	N/A
libTom		271	14	176	81	345	320	25	13	12	1	194	106	88	0	N/A	N/A	N/A	N/A
Relic		260	33	161	66	443	396	47	43	41	2	2	2	0	0	N/A	N/A	N/A	N/A
libGMP		397	117	227	53	320	309	11	41	40	1	737	227	510	0	N/A	N/A	N/A	N/A
libopus		74	5	62	7	38	36	2	11	10	1	125	21	103	1	N/A	N/A	N/A	N/A
libpcre2		101	8	92	1	77	71	6	6	4	1	213	30	182	1	N/A	N/A	N/A	N/A
libxml2		95	14	78	3	169	161	8	31	30	1	11	0	11	0	N/A	N/A	N/A	N/A
libgsm		5	0	4	1	8	8	0	0	0	0	0	0	0	0	N/A	N/A	N/A	N/A
libavc		5	0	1	4	5	0	5	1	0	0	0	0	0	0	N/A	N/A	N/A	N/A
libhevc		5	0	1	4	4	0	4	0	0	0	0	0	0	0	N/A	N/A	N/A	N/A
Total		3272	315	2639	318	3551	3269	282	440	413	27	4063	1164	2892	7	624	294	330	0

*filtered **93%** misuse, while another work filtered 47%*

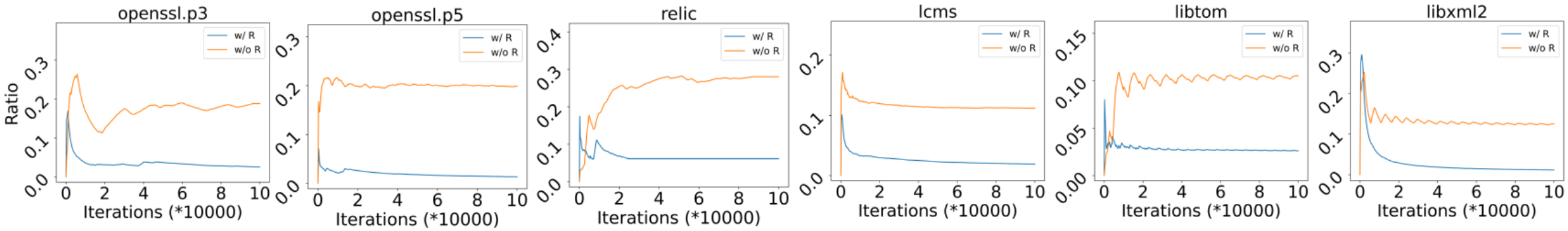
Evaluation – RQ2

- Ablation study
  - API usage from static consumer analysis: **~10%** more cov & **8** more bugs
  - API usage from dynamic learning: **~25%** more cov & **9** more bugs

Evaluation – RQ2

- Ablation study
  - API usage from static consumer analysis: **~10%** more cov & **8** more bugs
  - API usage from dynamic learning: **~25%** more cov & **9** more bugs

e.g., API misuse ratio with relation learning VS without relation learning:



The decreasing trends indicate that NEXZZER learns correct API relations to gradually generate more effective API call sequences

Discussion

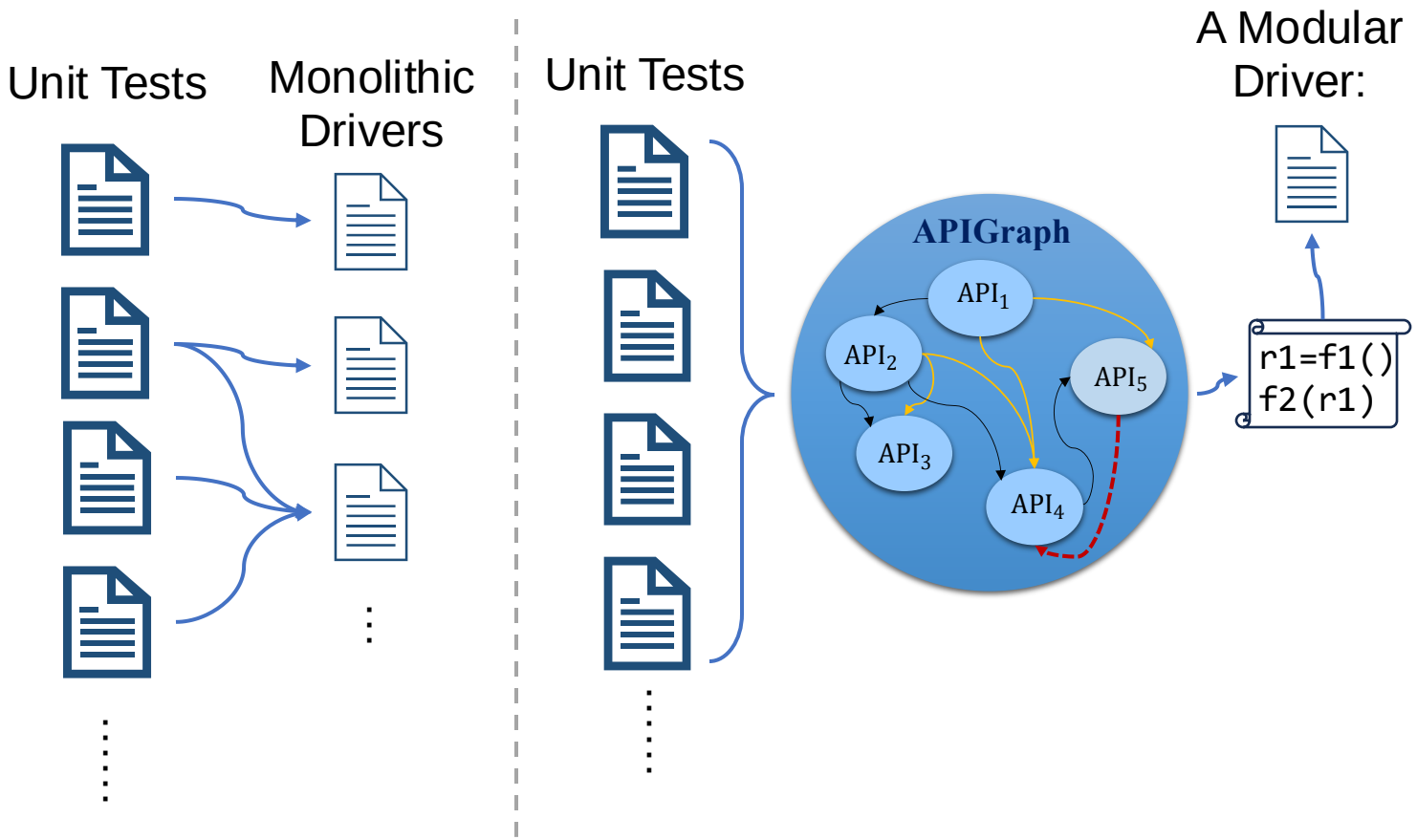
- Most bugs come from uncovered APIs
  - OpenSSL has thousands of APIs, oss-fuzz drivers only cover hundreds
- Some interesting bugs come from API sequence mutation
  - Manual drivers can't trigger a bug by: *BN_bin2bn* → *BN_mod_exp_mont_consttime*
  - NEXZZER can by mutating sequences: *BN_set_bit* → *BN_mod_exp_mont_consttime*
- Limitations
  1. Modeling of more complex API usage like multi-dimensional arrays' lengths
  2. Limited C++ support of the synthesized driver

Thank you!

- Q & A

Appendix – Modular Driver

- Scalability issue:




```
int DriverEntry(uint8_t *raw_api_seq) {
  APISeq api_seq = Interpret_and_setup(api_seq);
  for (int i = 0; i < api_seq.len(); i++) {
    Transfer_dependencies(api_seq);
    switch (api_seq.apis[i].idx) {
    // -------- Synthesized code starts --------
      case 0:
        BN_set_bit(api_seq.apis[i].args[0], ...);
        break;
      case 1:
        dep_1 = BN_new();
      ...
    // -------- Synthesized code ends ----------
    }
  }
  Collect_feedback(api_seq);
}
```

Appendix – Ablation Study

TABLE III: Consumer Analysis Results and Ablation Study

Targets		Consumer Results and Ablation								Relation Ablation						
		Consumer				Coverage			ΔBug	Coverage			FP Ratio			ΔBug
		#F	#N	#E*	#E	w/o C	NEXZZER	ΔCov		w/o R	NEXZZER	ΔCov	w/o R	NEXZZER	ΔFP	
OpenSSL	**P1**	296	568	16607	20852	9408	10904	**+15.90%**	2	7784	10904	**+28.61%**	0.2583	0.0377	**-85.40%**	1
	P2	547	951	11773	17346	11967	12358	**+3.27%**	0	9267	12358	**+25.01%**	0.1965	0.0289	**-85.29%**	0
	P3	255	634	2917	7642	12506	15076	**+20.55%**	0	9796	15076	**+35.02%**	0.1884	0.0254	**-86.51%**	0
	P4	262	625	4341	6638	10357	13983	**+35.01%**	0	9575	13983	**+31.52%**	0.1188	0.0309	**-73.98%**	1
	P5	117	423	3390	4892	9652	10352	**+7.25%**	0	9414	10352	**+9.06%**	0.1985	0.0122	**-93.85%**	0
	P6	24	254	794	1603	4987	5869	**+17.69%**	1	5397	5869	**+8.04%**	0.0416	0.0085	**-79.56%**	1
	P7	200	539	1439	3078	9011	9149	**+1.53%**	0	8537	9149	**+6.68%**	0.1188	0.0208	**-82.49%**	0
libpcre2		40	78	165	834	6511	6825	**+4.82%**	1	6171	7266	**+15.07%**	0.0512	0.0069	**-86.52%**	1
tesseract†		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	16925	20315	**+20.03%**	0.2018	0.0679	**-66.35%**	2
jsonnet		6	48	106	141	3091	3434	**+11.10%**	1	2410	3434	**+42.49%**	0.1313	0.0127	**-90.33%**	0
leveldb		87	92	8	204	1810	1877	**+3.70%**	0	1612	1877	**+16.44%**	0.0152	0.0075	**-50.66%**	0
uriparser		3	70	71	179	7313	7835	**+7.14%**	0	6637	7835	**+18.05%**	0.1341	0.0881	**-34.30%**	0
libxml2		21	1182	16255	22972	13489	14195	**+5.23%**	1	10072	14195	**+28.86%**	0.1246	0.0079	**-93.65%**	0
cJSON		19	79	2459	2763	865	865	0.00%	0	686	865	**+20.69%**	0.1837	0.0237	**-87.09%**	0
lcms		4	287	3382	4120	3011	3306	**+9.80%**	2	2649	3306	**+19.87%**	0.1138	0.0187	**-83.56%**	2
libTom		19	180	3872	5640	1298	1374	**+5.86%**	0	948	1374	**+31.00%**	0.1045	0.0722	**-30.90%**	1
Relic		59	361	420	2538	3877	4192	**+8.12%**	0	3296	4192	**+22.08%**	0.2795	0.0348	**-87.54%**	0
libGMP		87	594	1339	7495	6019	6450	**+7.16%**	0	3201	6450	**+50.37%**	0.2205	0.0568	**-74.24%**	0
libopus		4	67	49	245	2965	3276	**+10.49%**	0	2197	3276	**+32.93%**	0.2249	0.0933	**-58.51%**	0
libvpx		4	37	108	218	2290	2467	**+7.73%**	0	1940	2467	**+19.40%**	0.0001	0.0001	**0.00%**	0
libaom		4	35	113	201	6889	7381	**+7.14%**	0	5642	7381	**+23.56%**	0.0001	0.0001	**0.00%**	0
libgsm		4	35	16	35	318	324	**+2.99%**	0	292	301	**+2.99%**	0.0001	0.0001	**0.00%**	0
libavc		2	53	0	280	33	4202	**+12633%**	0	3438	4202	**+18.18%**	0.0001	0.0001	**0.00%**	0
libhevc		2	29	0	135	46	4733	**+10189%**	0	4371	4733	**+7.64%**	0.0001	0.0001	**0.00%**	0

#F: the number of consumer files; **#N**: the number of (possibly duplicated) API call nodes in consumers; **#E***: the number of Def-Use Edges by type-matching; **#E**: the number of all edges after consumer analysis;
w/o C: running NEXZZER without the results from consumer analysis (i.e., only use type-matching edges); **w/o R**: running NEXZZER without the relation learning to constrain the mutator;
†: We did not find consumers using the C APIs of *tesseract* in its repository.

Appendix – API Usage Learning Workflow

Algorithm 3: A Fuzzing Iteration

Input: APIGraph G, Seed Corpus C.
▷ E: learned edges.
▷ X: learned argument constraints.
▷ S: current seed.
$S \leftarrow mutation\_generation\,(APIGraph, C)$
$feedback \leftarrow execute\,(S)$
$idx \leftarrow feedback.target\_idx$
$SS \leftarrow G.node_{idx}.SeedSpaces$
if $duplicate\,(S,\ idx,\ SS,\ feedback)$ **then**
  return
$S, E \leftarrow InterAPI\_relation\_learning\,(G,\ S,\ idx)$
$X \leftarrow IntraAPI\_relation\_learning\,(S,\ idx)$
$SS.update\,(S, X, E)$
for $e \in$ *RULES* **do**
  $check\_misuse\,(e, S, SS)$
end
$G.update\,(SS)$