Classifying Static Analysis False Positives by Learning from Alarm Review Data

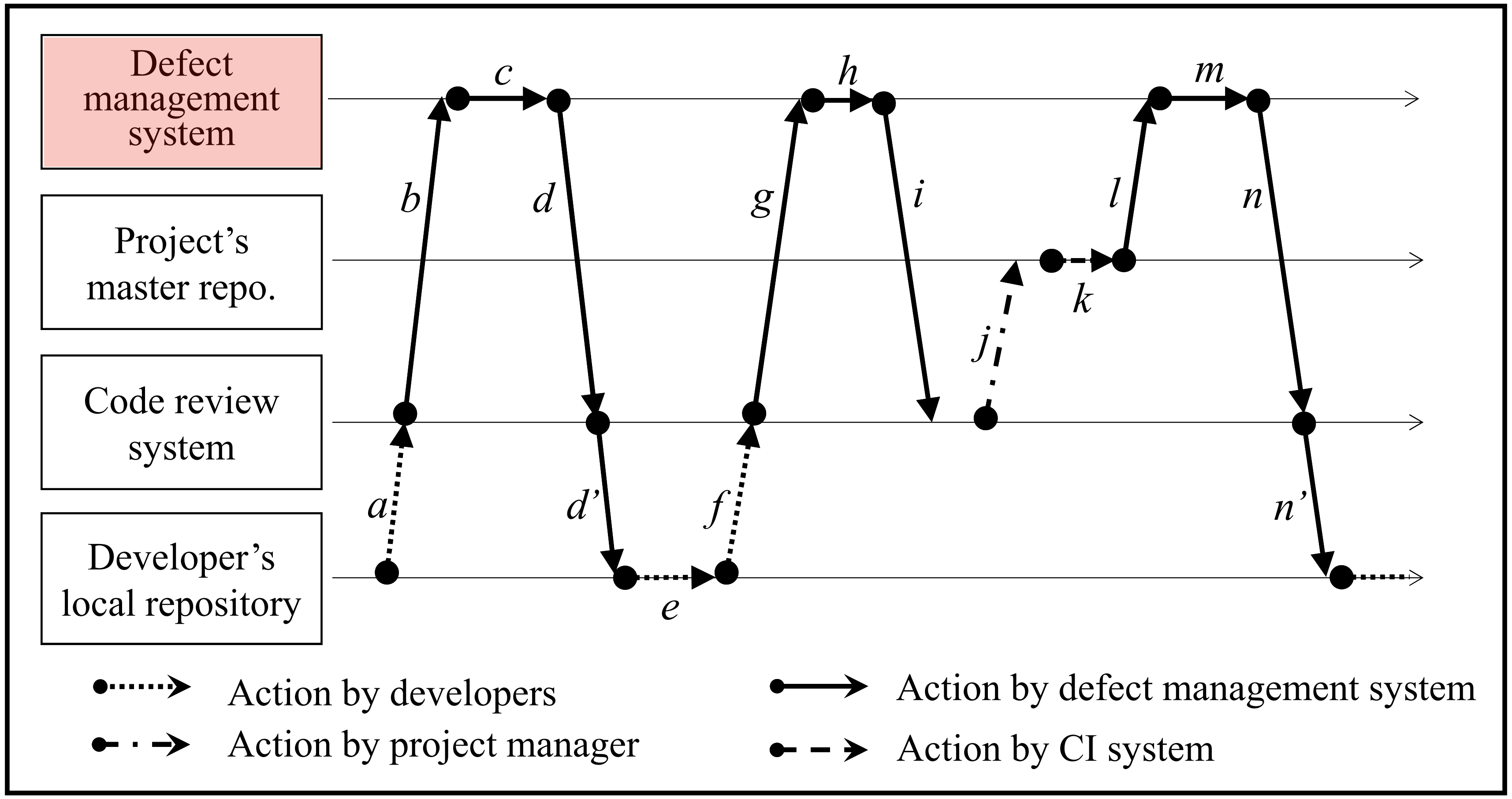
Seongmin Lee, Shin Yoo, Shin Hong
Jungbae Yi, Taeksu Kim, Chul-Joo Kim

HANDONG GLOBAL UNIVERSITY

KAIST

SAMSUNG

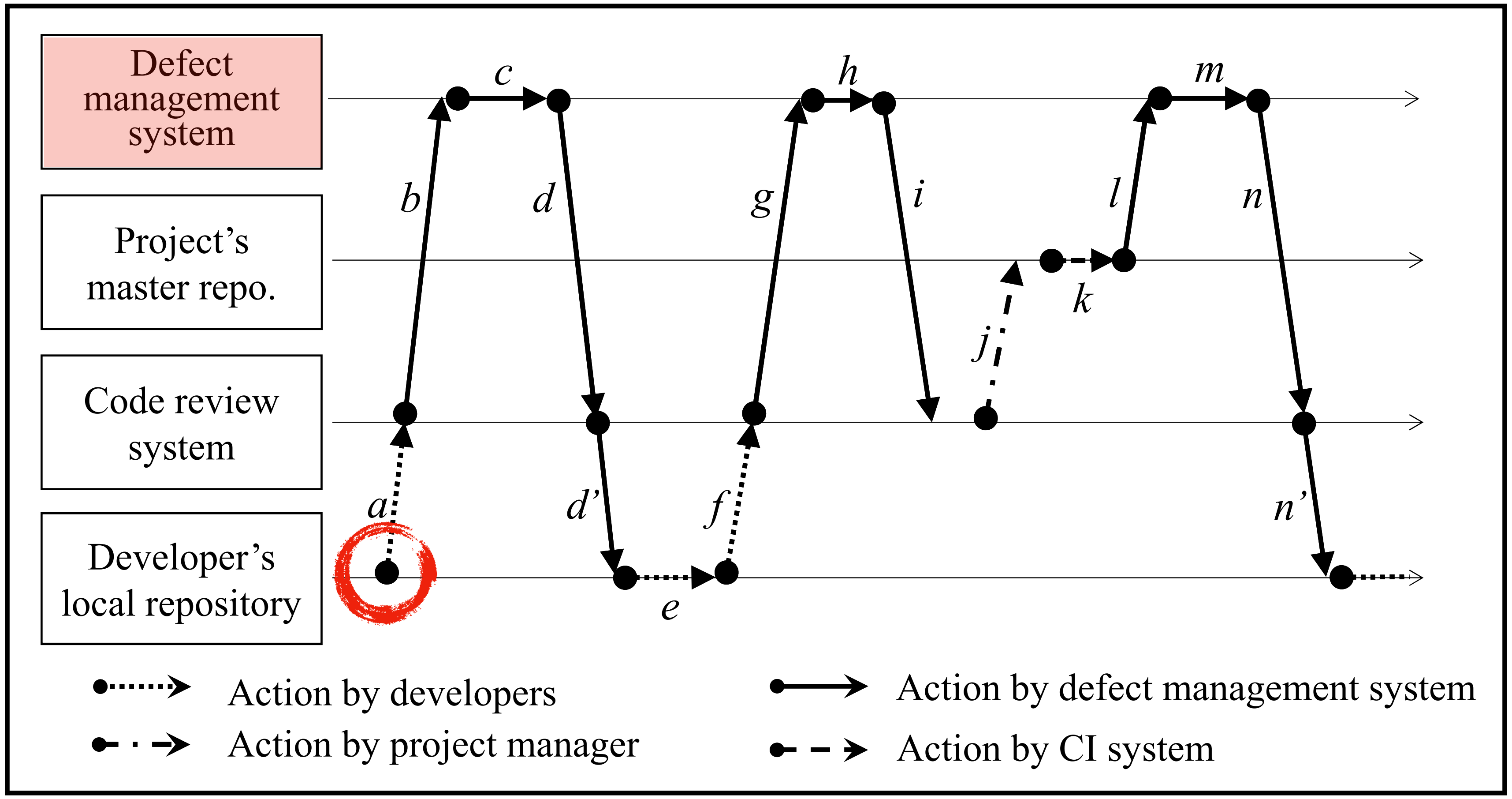
Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

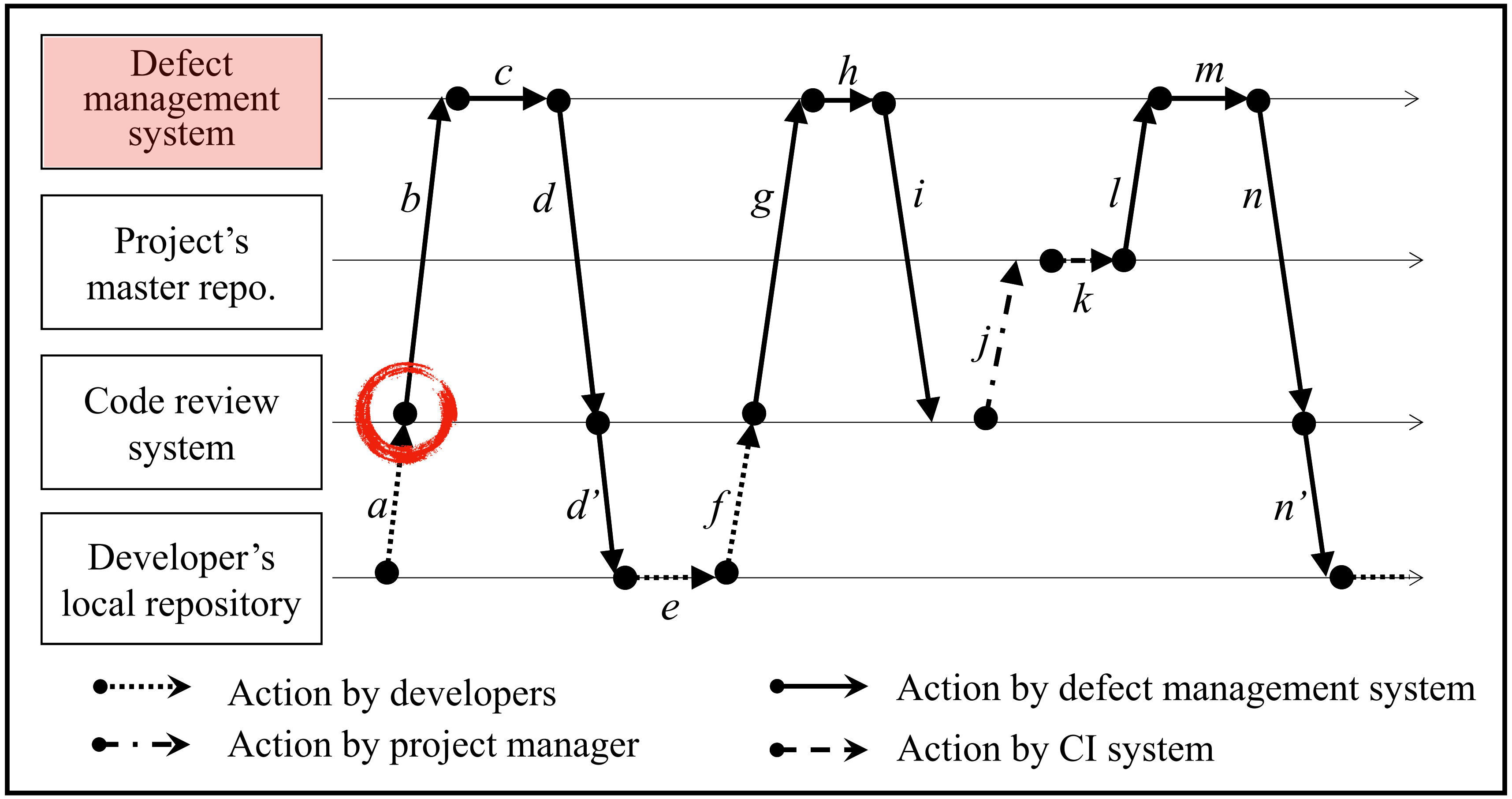
Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

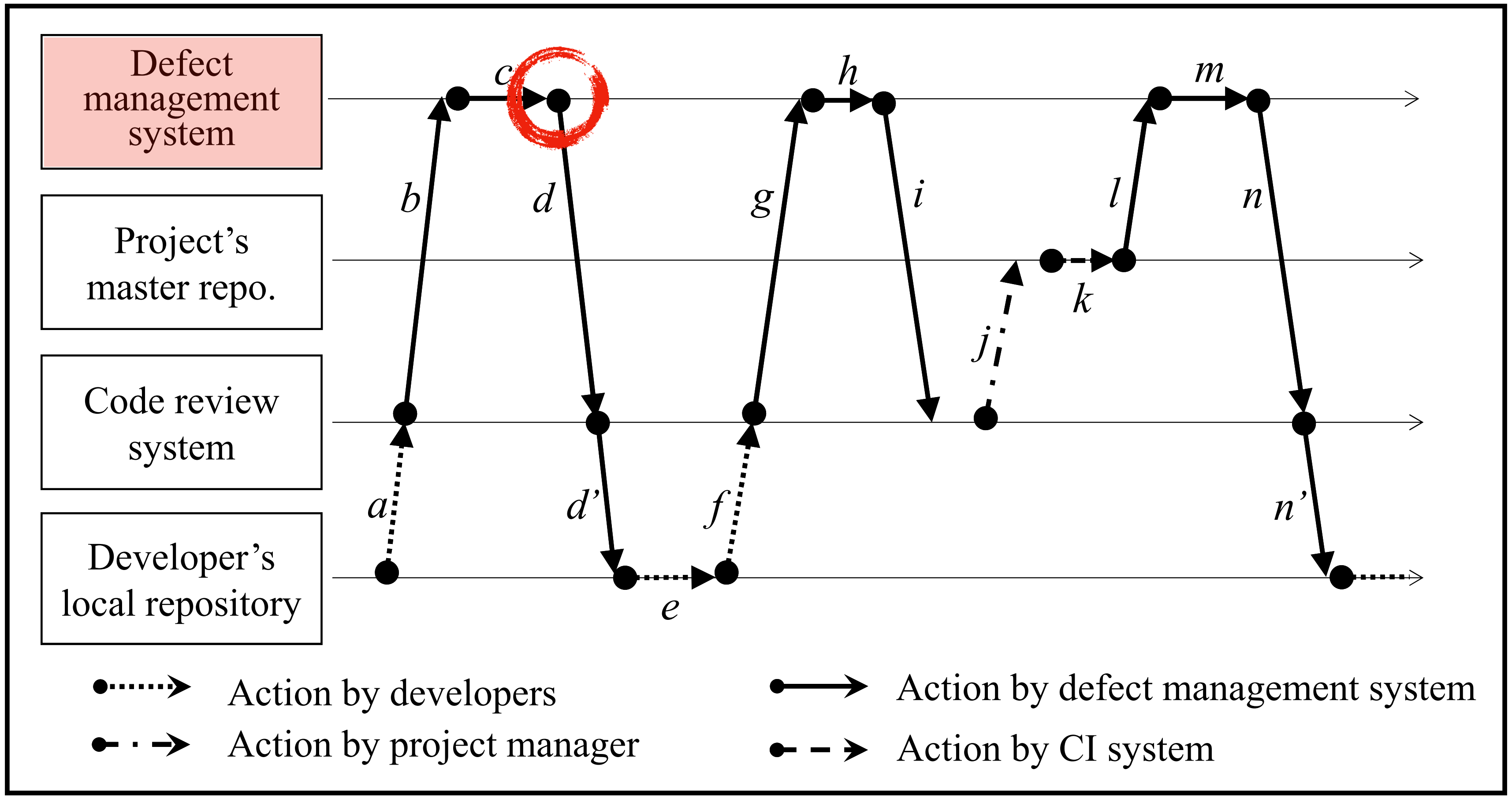
Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

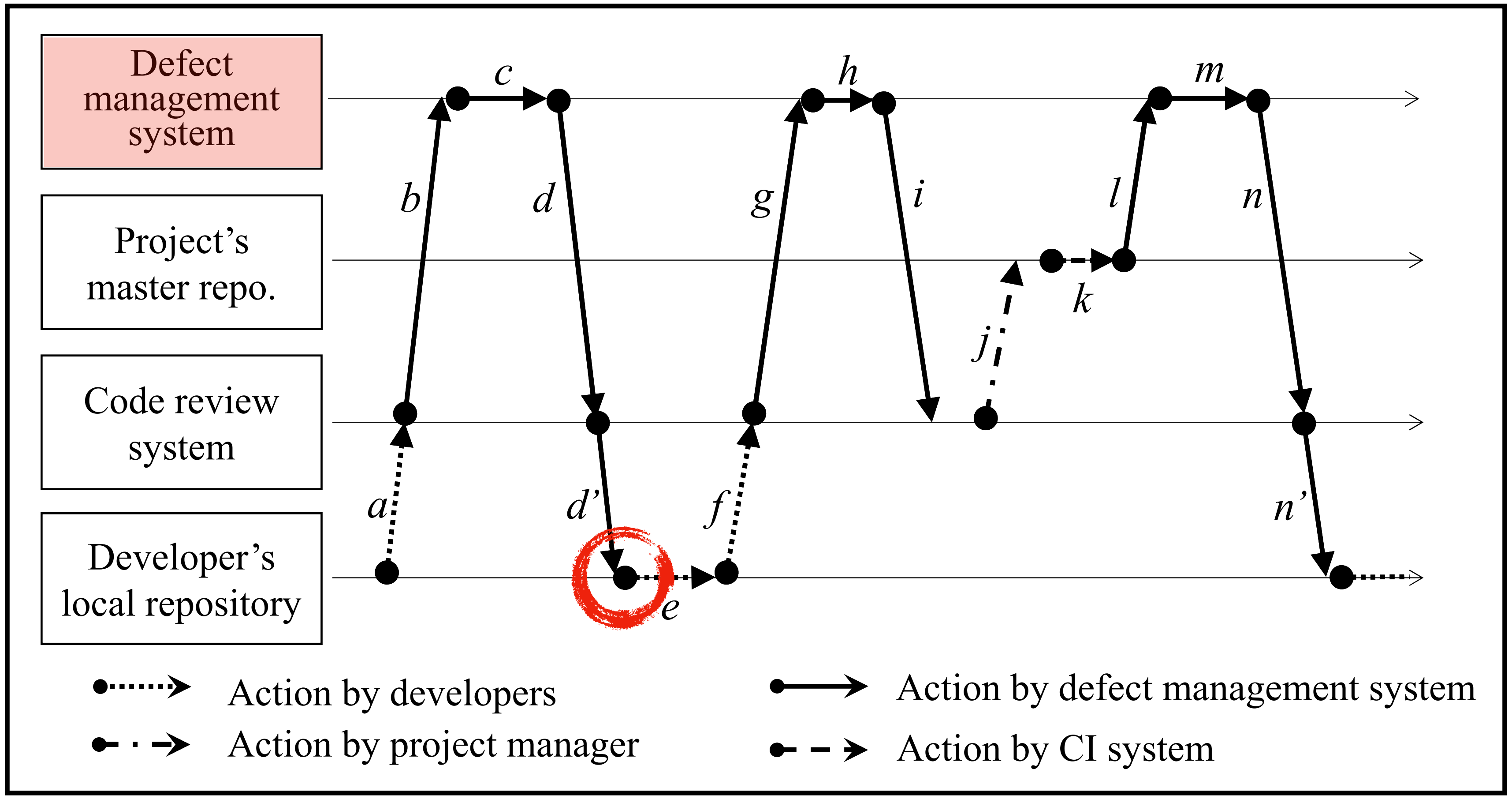
Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

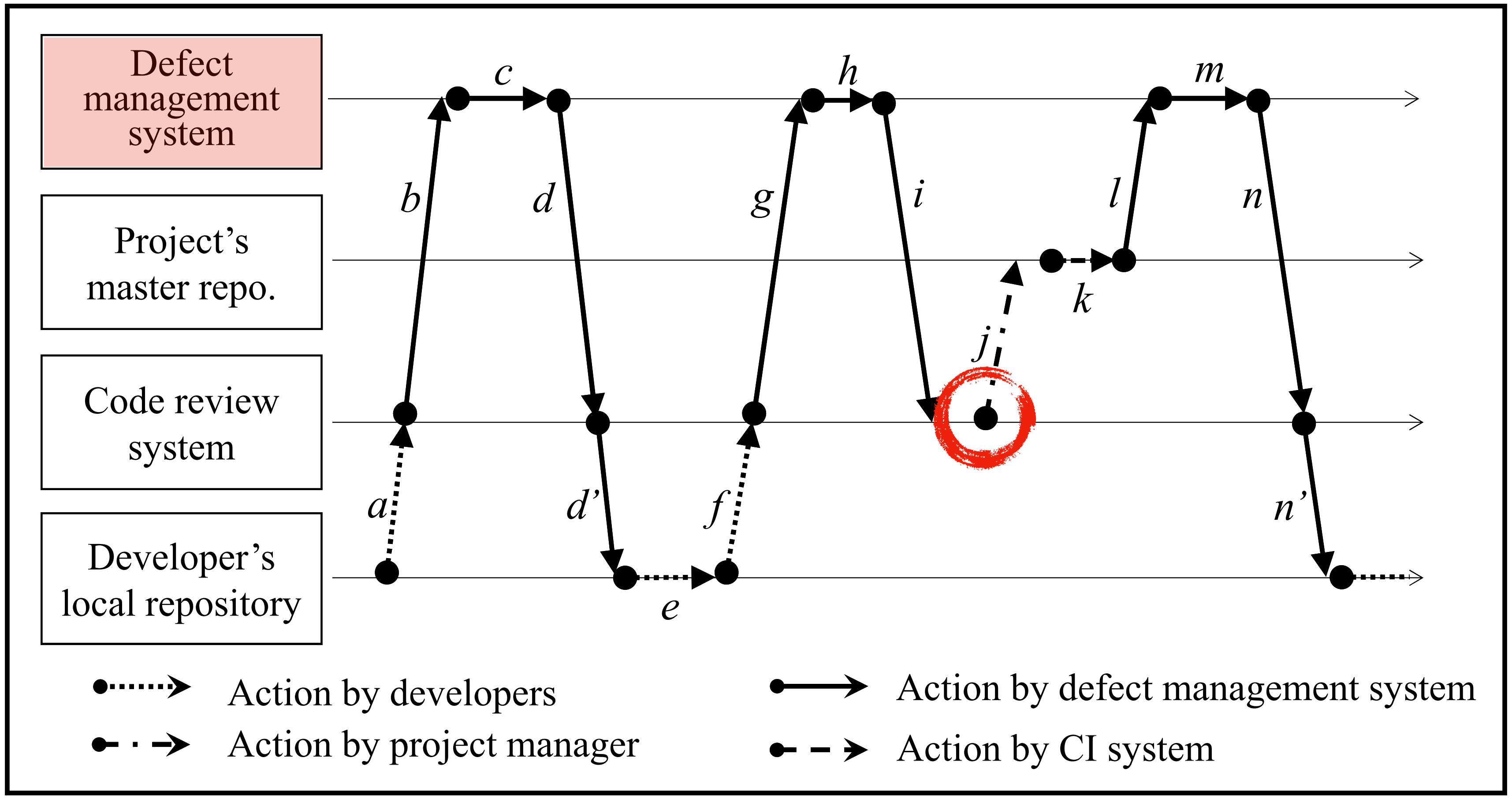
Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

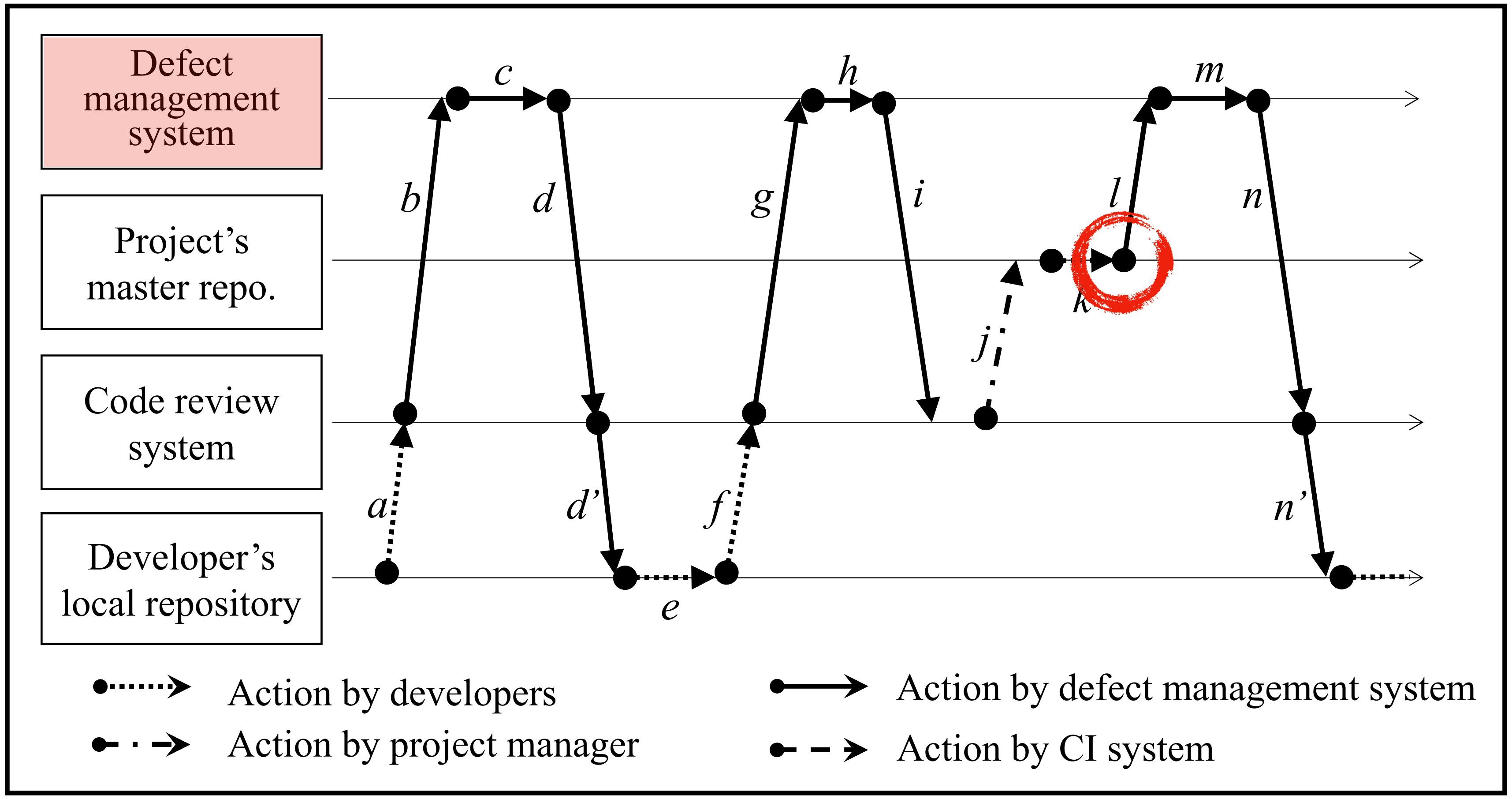
  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis
    - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

Static Analysis Practice in Samsung



Defect management system

Project's master repo.

Code review system

Developer's local repository

Action by developers

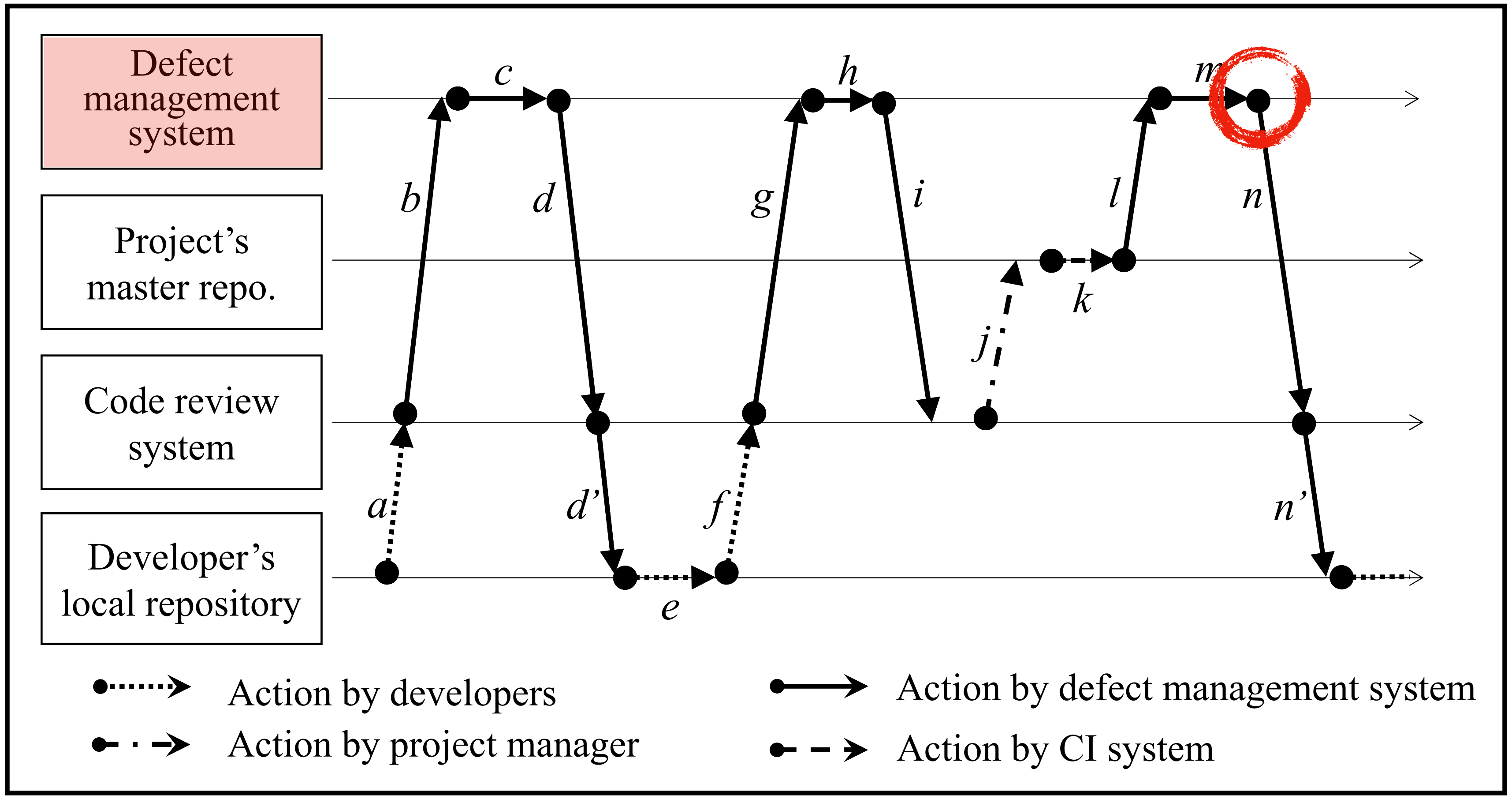
Action by project manager

Action by defect management system

Action by CI system

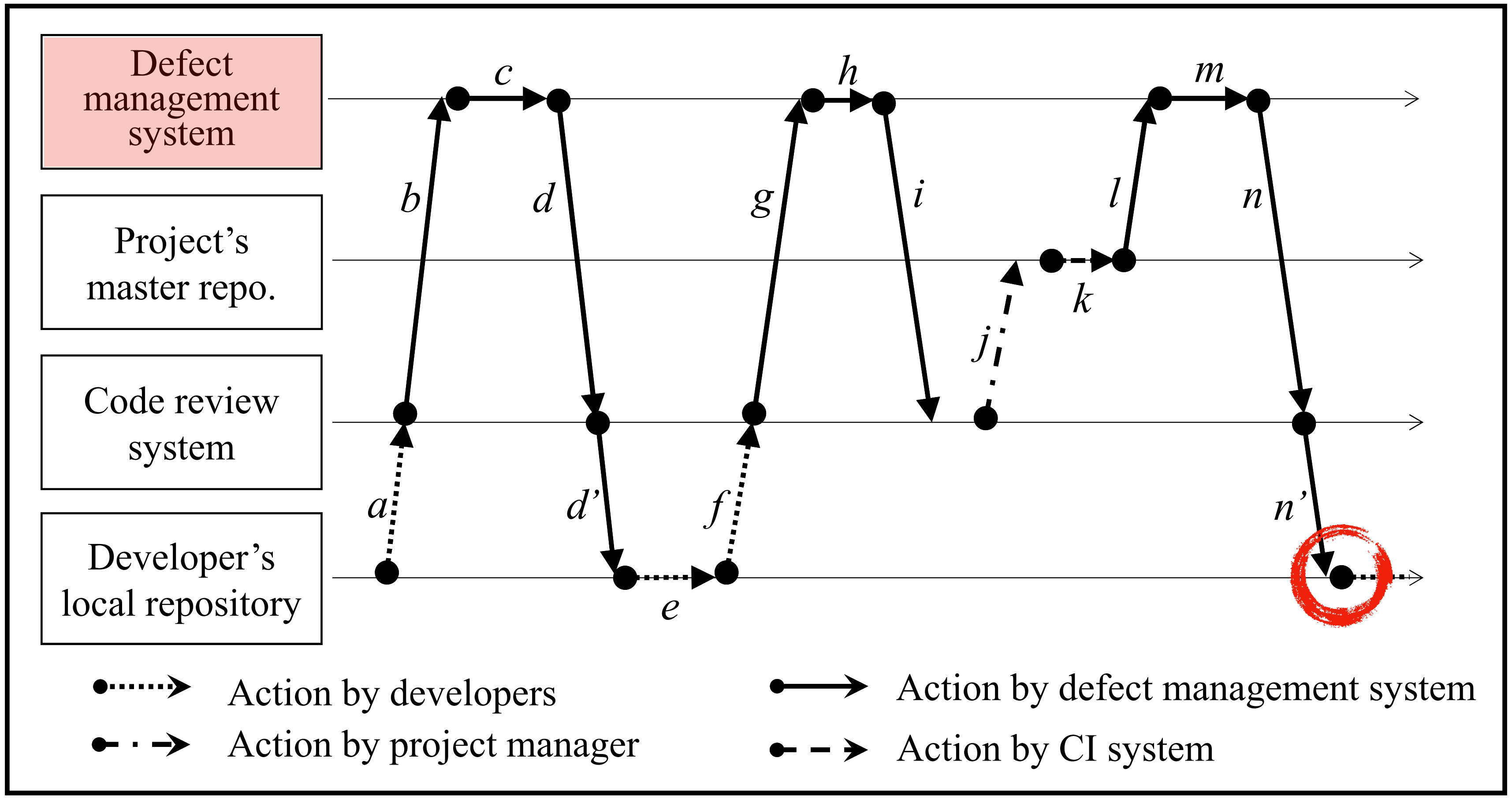
- Every project runs predefined static analysis checker per-commit and per-release basis
  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis
  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

Static Analysis Practice in Samsung



- Every project runs predefined static analysis checker per-commit and per-release basis

  - SVACE is tightly integrated in Continuous Integration pipeline. (E.g., Tizen uses 252 checkers over all its sub-projects)

Challenge: High Ratio of False Positive

- E.g., false positives in analyzing Tizen (sampled)

Category	Checker	FP ratio
API call sequence	MEMORY_LEAK.EX	36 %
	HANDLE_LEAK	44 %
	MEMORY_LEAK.STRUCT	27 %
	MEMORY_LEAK.STRDUP	36 %
	MEMORY_LEAK	43 %
	DOUBLE_FREE	32 %
Dataflow	DEREF_AFTER_NULL.EX	25 %
	DEREF_OF_NULL.EX	31 %
	TAINTED_INT.LOOP.MIGHT	50 %
	DEREF_AFTER_FREE.EX	48 %
Control flow	FALL_THROUGH	39 %
	UNREACHABLE_CODE	17 %
		Average: 35%

Challenge: High Ratio of False Positive

- E.g., false positives in analyzing Tizen (sampled)

Category	Checker	FP ratio
A se	MEMORY_LEAK.EX	36 %
	HANDLE_LEAK	44 %
	MEMORY_LEAK.STRUCT	27 %
	MEMORY_LEAK.STRDUP	36 %
	MEMORY_LEAK	43 %
	DOUBLE_FREE	32 %
flow	DEREF_AFTER_NULL.EX	25 %
	DEREF_OF_NULL.EX	31 %
	TAINTED_INT.LOOP.MIGHT	50 %
	DEREF_AFTER_FREE.EX	48 %
Control flow	FALL_THROUGH	39 %
	UNREACHABLE_CODE	17 %
		Average: 35%

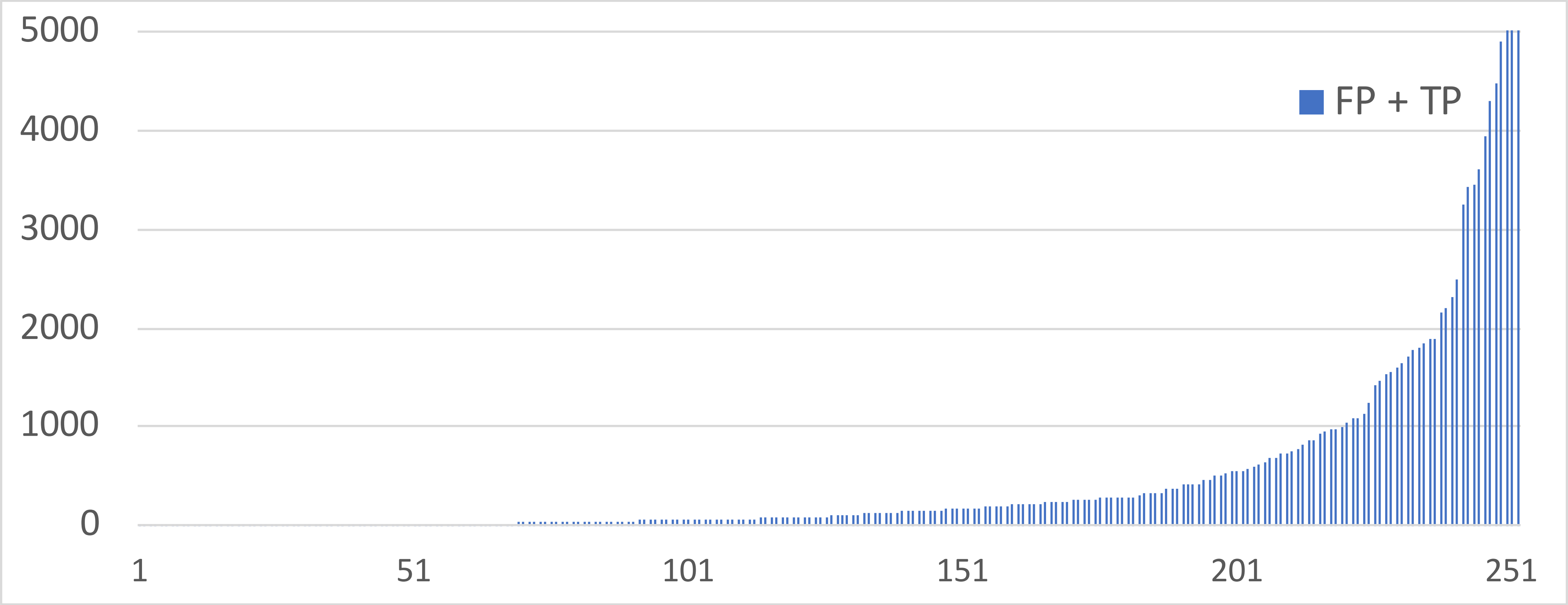
Challenge: High Ratio of False Positive

- E.g., false positives in analyzing Tizen (sampled)

Category	Checker	FP ratio
A se	MEMORY_LEAK.EX	36 %
	HANDLE_LEAK	44 %
	MEMORY_LEAK.STRUCT	27 %
	MEMORY_LEAK.STRDUP	36 %
	_LEAK	43 %
	REE	32 %
flow	ER_NULL.EX	25 %
	_NULL.EX	31 %
	D_INT.LOOP.MIGHT	50 %
	DEREF_AFTER_FREE.EX	48 %
Co	FALL_THROUGH	39 %
	UNREACHABLE_CODE	17 %
		Average: 35%

Chances: Developers' Feedback

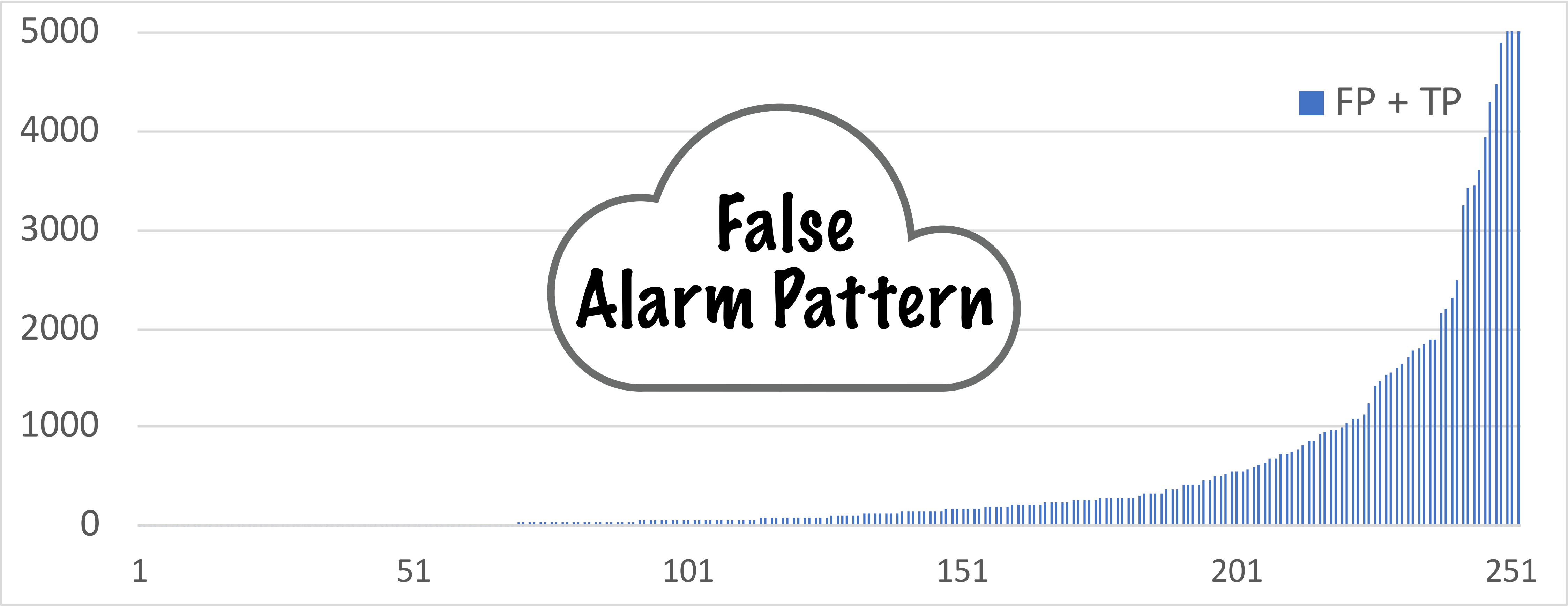
- From 2016, SVACE collects all target source code files, all alarms sent back to developers, and feedbacks (labels) from developers.
- E.g., 150k datapoints on the Tizen domain



Chances: Developers' Feedback

- From 2016, SVACE collects all target source code files, all alarms sent back to developers, and feedbacks (labels) from developers.

- E.g., 150k datapoints on the Tizen domain



Checker 1. **HANDLE_LEAK**

- `HANDLE_LEAK` reports a warning for a pair of statements in a function <*X*, *Y*> if
  1. *X* acquires a resource (e.g., `fopen`) and stores the handler to a local var. *V*,
  2. *Y* follows X in an execution path where *V* does not escape to global, and
  3. *Y* eliminates the handler by overwriting *V* or by deallocating *V* (i.e., `return`)

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 3367 cases (15.4%)
  - True alarms: 18485 cases (84.6%)

```
func() {
  int fd = open(…);   // acquire
  ...
  if (feof(fd) == true)
     return;          // release
}
```

True alarm

```
func() {
  int fd = open(…);    // acquire
  if (fd < 0) {
    error();
    return;        // not released
... }
```

False alarm

Checker 1. HANDLE_LEAK

- HANDLE_LEAK reports a warning for a pair of statements in a function <*X*, *Y*> if
  1. *X* acquires a resource (e.g., fopen) and stores the handler to a local var. *V*,
  2. *Y* follows X in an execution path where *V* does not escape to global, and
  3. *Y* eliminates the handler by overwriting *V* or by deallocating *V* (i.e., return)

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 3367 cases (15.4%)
  - True alarms: 18485 cases (84.6%)

```
func() {
  int fd = open(…);  // acquire
  ...
  if (feof(fd) == true)
     return;          // release
}
```

True alarm

```
func() {
  int fd = open(…);   // acquire
  if (fd < 0) {
    error();
    return;       // not released
... }
```

False alarm

Checker 1. HANDLE_LEAK

- `HANDLE_LEAK` reports a warning for a pair of statements in a function <*X*, *Y*> if

1. *X* acquires a resource (e.g., `fopen`) and stores the handler to a local var. *V*,
2. *Y* follows X in an execution path where *V* does not escape to global, and
3. *Y* eliminates the handler by overwriting *V* or by deallocating *V* (i.e., `return`)

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 3367 cases (15.4%)
  - True alarms: 18485 cases (84.6%)

```
func() {
  int fd = open(…);  // acquire
  ...
  if (feof(fd) == true)
     return;          // release
}
```

True alarm

```
func() {
  int fd = open(…);   // acquire
  if (fd < 0) {
    error();
    return;       // not released
... }
```

False alarm

Checker 2. **FALL_THROUGH**

- FALL_THROUGH reports a warning for a `case` block if there may be a path that possibly exits the block without taking a `break` statement.

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 2709 cases (13%)
  - True alarms: 18265 cases (87%)

```
switch (z) {
 case 1:
  if (e == 1)
   break;
  else if (e == 2)
   break;  // else break missing
 case 2:
   …
```

True alarm

```
switch (z) {
  case ‘x’:                  // intended
  case ‘y’:                  // fall
  case ‘z’:                  // through
     x_or_y_or_z = 1;

  case ‘a’:
    …
```

False alarm

Checker 2. **FALL_THROUGH**

- FALL_THROUGH reports a warning for a `case` block if there may be a path that possibly exits the block without taking a `break` statement.

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 2709 cases (13%)
  - True alarms: 18265 cases (87%)

```
switch (z) {
 case 1:
  if (e == 1)
   break;
  else if (e == 2)
   break;  // else break missing
 case 2:
   …
```

True alarm

```
switch (z) {
  case ‘x’:            // intended
  case ‘y’:            // fall
  case ‘z’:            // through
     x_or_y_or_z = 1;

  case ‘a’:
   …
```

False alarm

Checker 2. **FALL_THROUGH**

- FALL_THROUGH reports a warning for a `case` block if there may be a path that possibly exits the block without taking a `break` statement.

- Warning review data (collected from Tizen in July 2017)
  - False alarms: 2709 cases (13%)
  - True alarms: 18265 cases (87%)

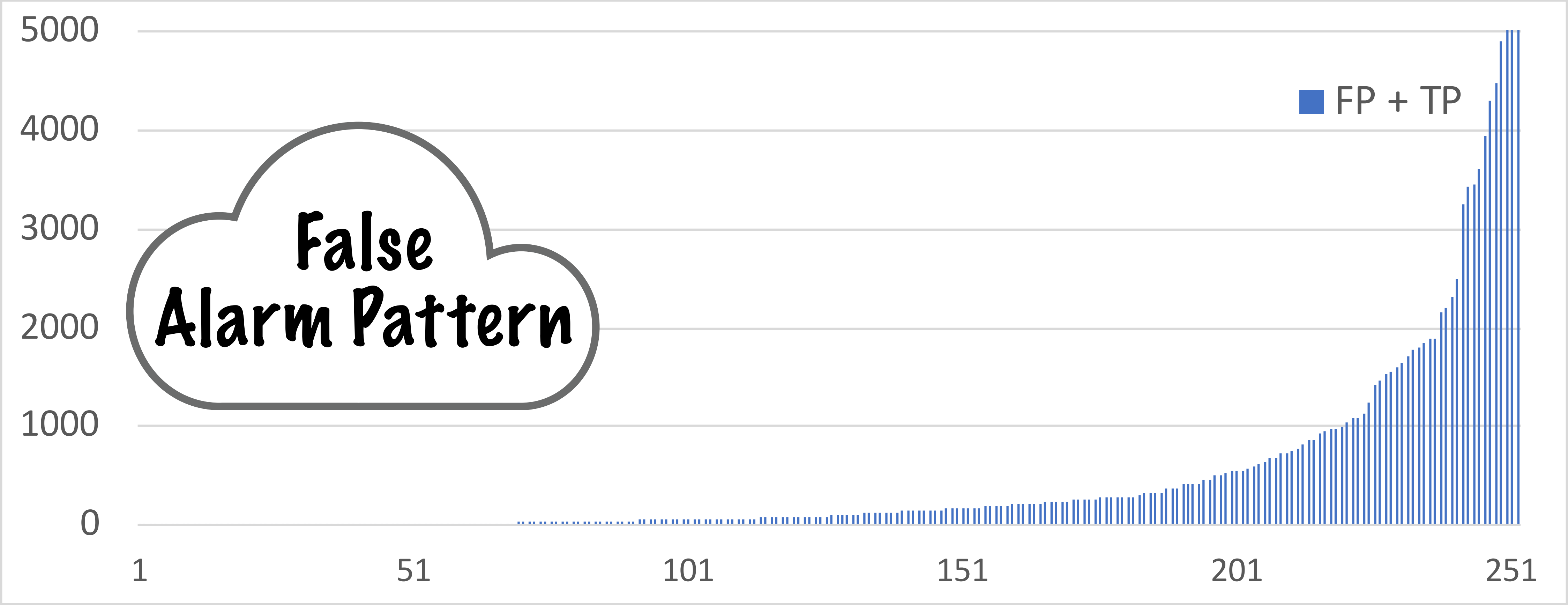
```
switch (z) {
 case 1:
  if (e == 1)
   break;
  else if (e == 2)
   break;  // else break missing
 case 2:
   …
```

True alarm

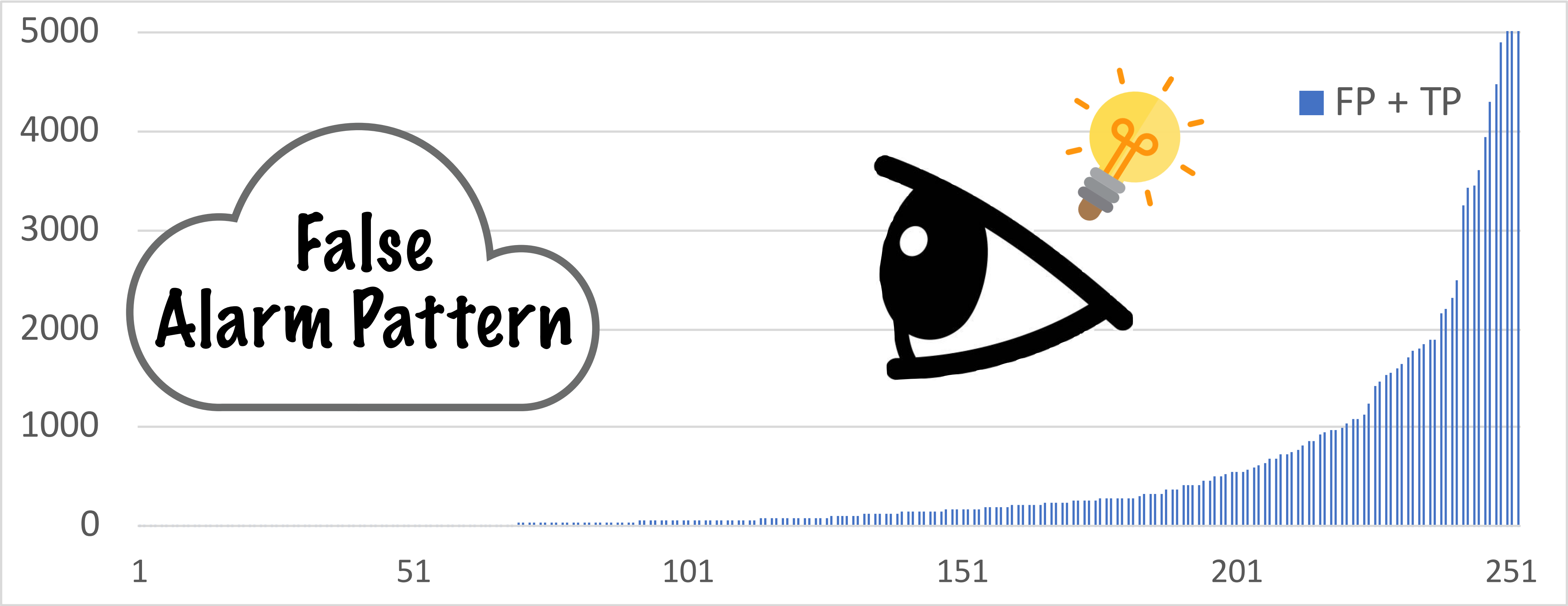
```
switch (z) {
  case ‘x’:                 // intended
  case ‘y’:                 // fall
  case ‘z’:                 // through
     x_or_y_or_z = 1;

  case ‘a’:
   …
```

False alarm

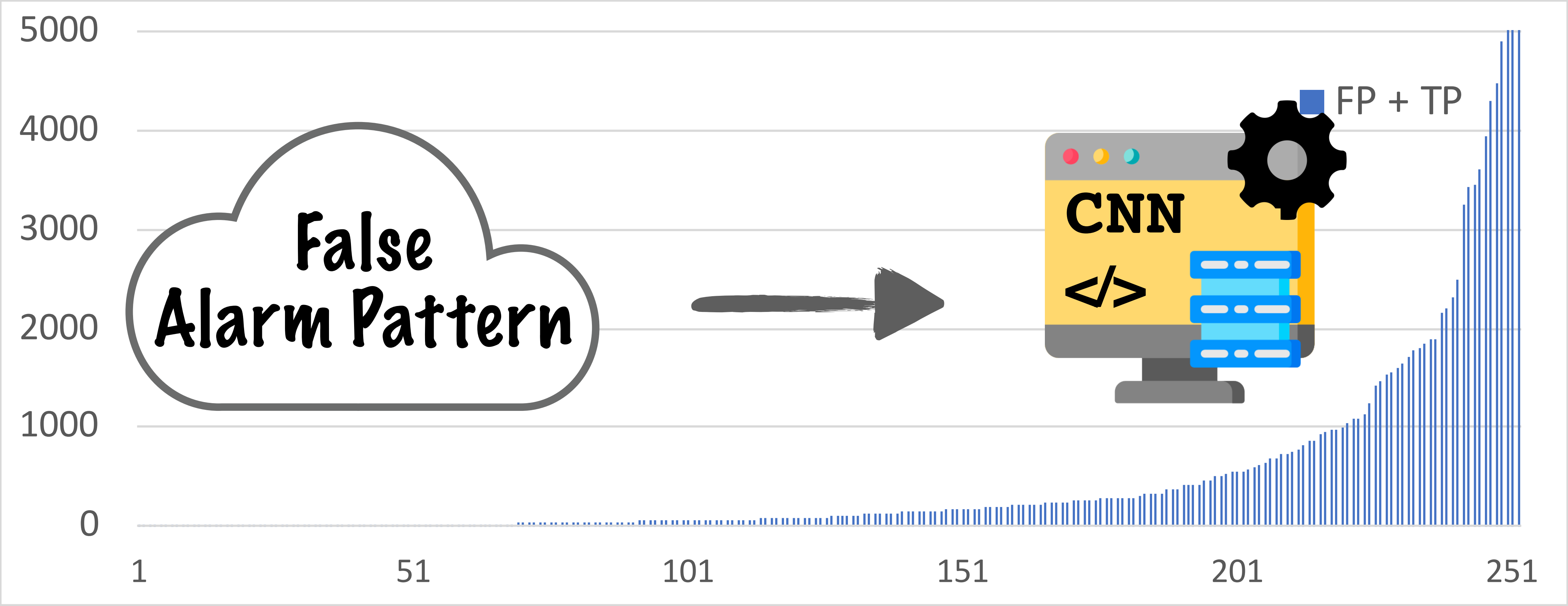
Learn the lexical pattern



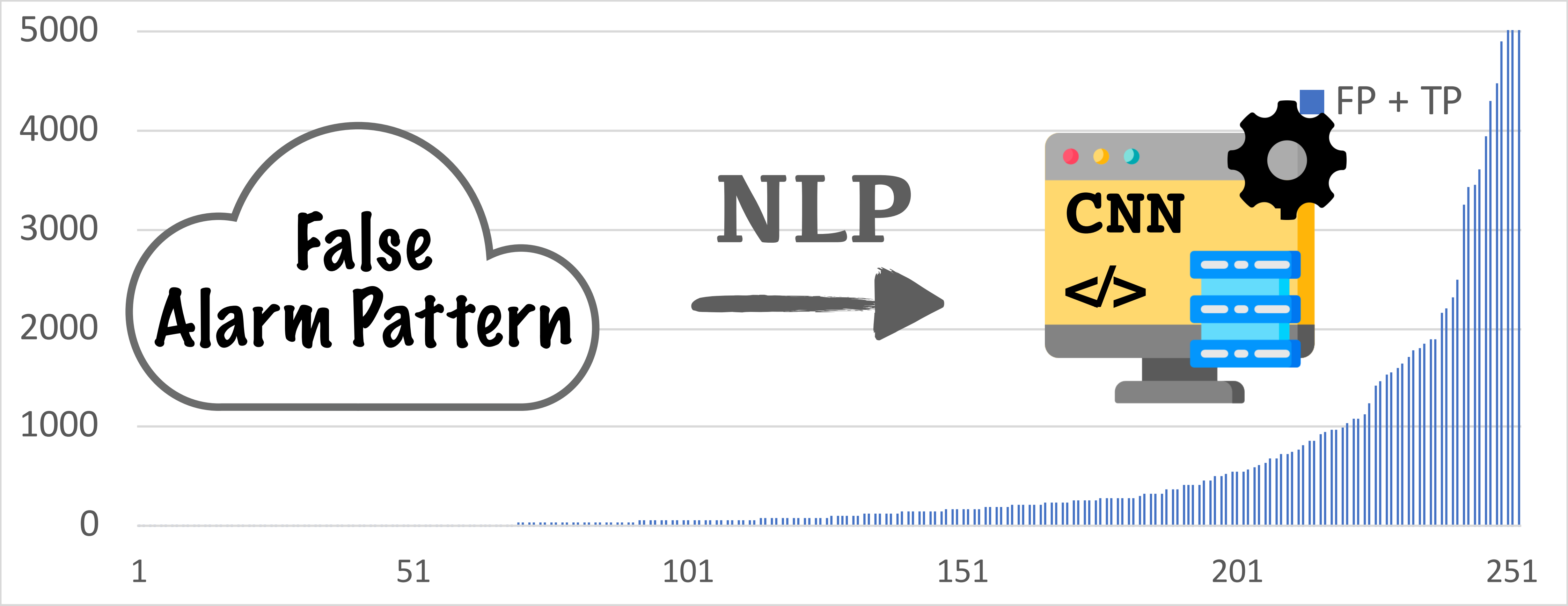
Learn the lexical pattern



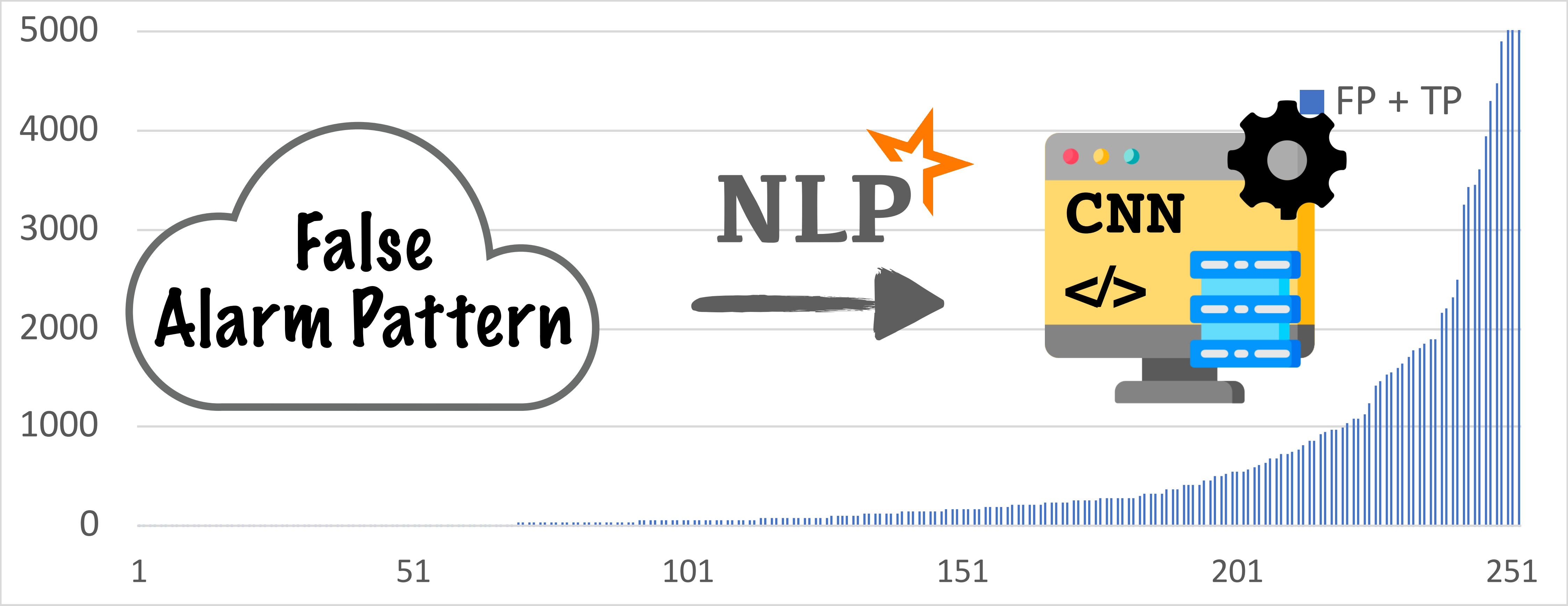
Learn the lexical pattern



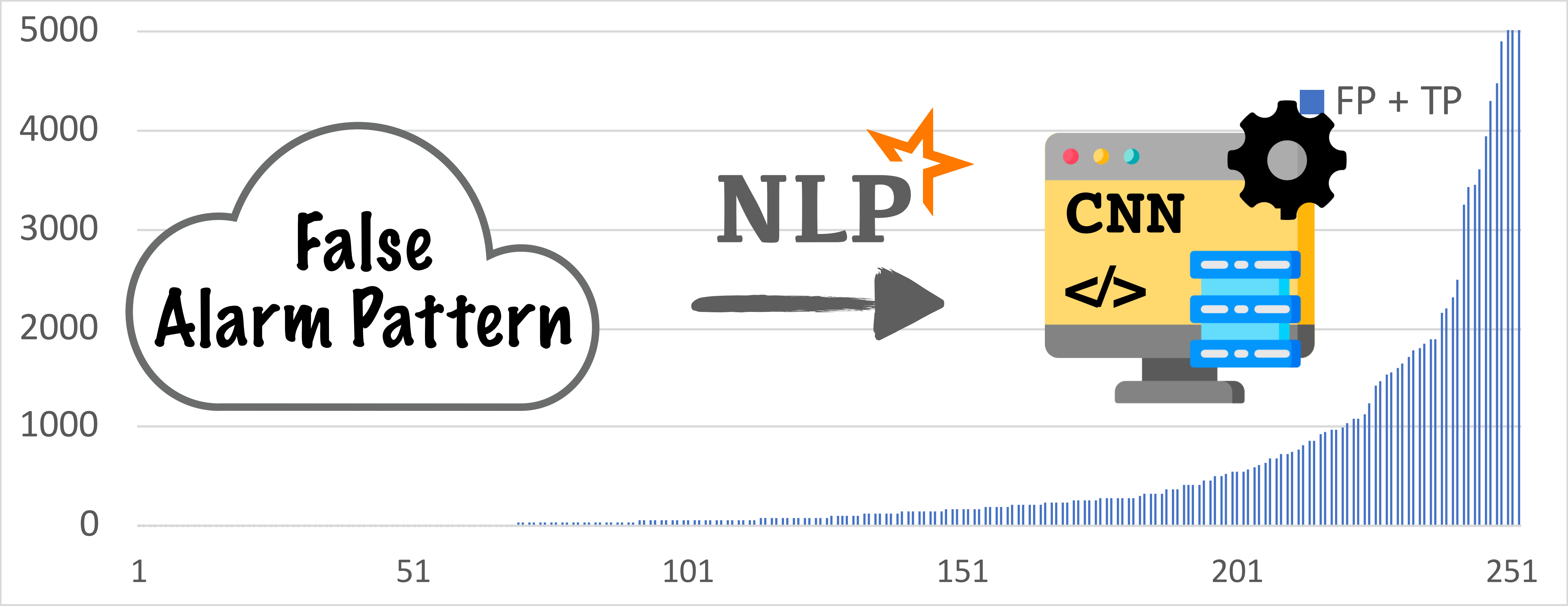
Learn the lexical pattern



Learn the lexical pattern



Learn the lexical pattern



No feature extraction is needed!

Data gathering

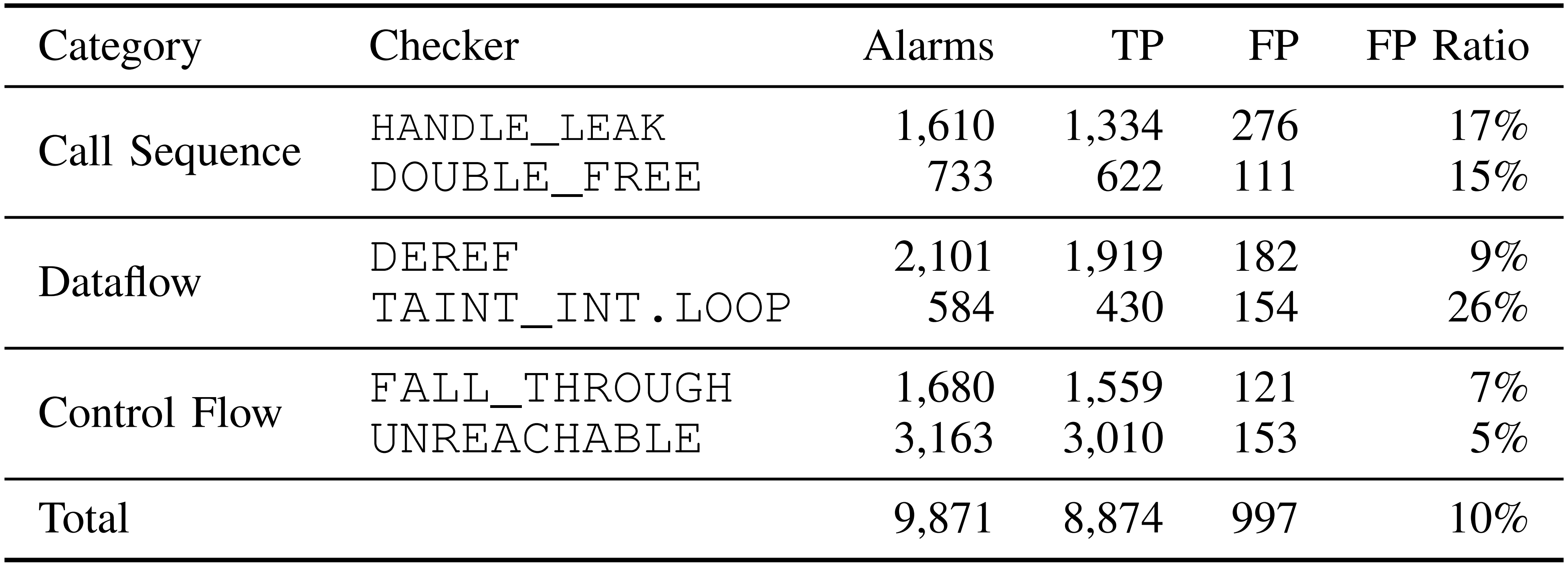
- **Picked 12 checkers.** These checkers
  - are used for Tizen,
  - check important properties,
  - have many alarms,
  - are motivated to high false alarm ratio.
- **Data cleaning**
  - Remove noisy, duplicated data & Normalize data
  - 150K → 9.8K datapoints (580-2100 datapoint per checker)
- **Label transformation**
  - {Confirmed, Won't fix, Fixed, Undecided, False positive}

    → {0, 1}

Target Static Analysis Checkers

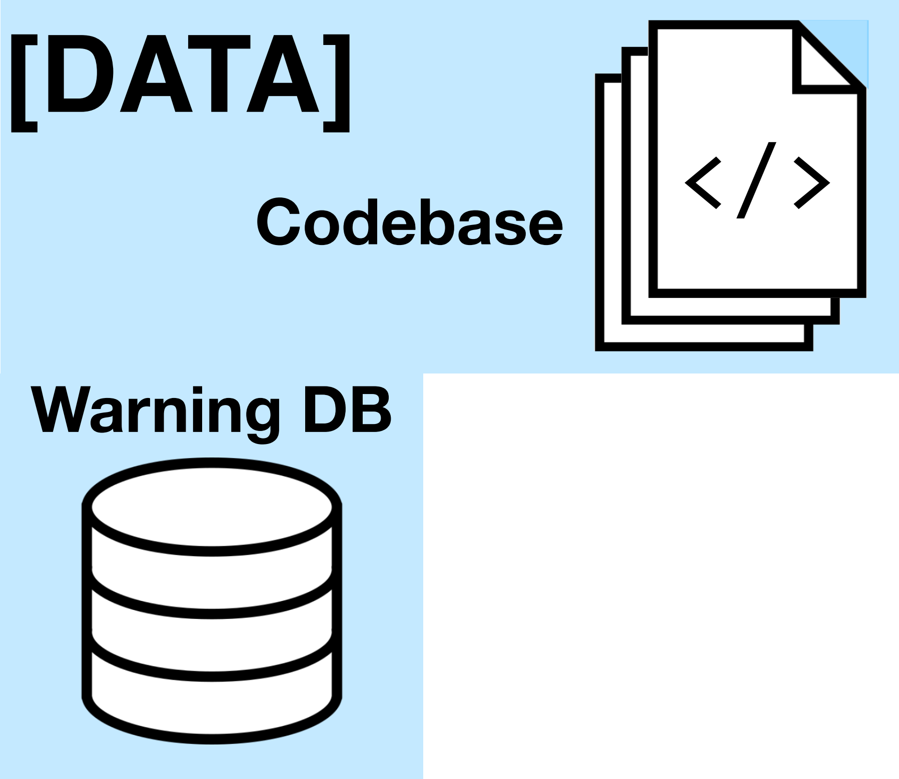
Category	Checker	TP	FP	FP ratio
API call sequence	MEMORY_LEAK.EX	2496	1391	36 %
	HANDLE_LEAK	1552	1203	44 %
	MEMORY_LEAK.STRUCT	548	203	27 %
	MEMORY_LEAK.STRDUP	376	214	36 %
	MEMORY_LEAK	293	220	43 %
	DOUBLE_FREE	271	126	32 %
Dataflow	DEREF_AFTER_NULL.EX	408	134	25 %
	DEREF_OF_NULL.EX	345	157	31 %
	TAINTED_INT.LOOP.MIGHT	129	131	50 %
	DEREF_AFTER_FREE.EX	133	123	48 %
Control flow	FALL_THROUGH	309	196	39 %
	UNREACHABLE_CODE	941	187	17 %

Target Static Analysis Checkers



Category	Checker	Alarms	TP	FP	FP Ratio
Call Sequence	`HANDLE_LEAK`	1,610	1,334	276	17%
	`DOUBLE_FREE`	733	622	111	15%
Dataflow	`DEREF`	2,101	1,919	182	9%
	`TAINT_INT.LOOP`	584	430	154	26%
Control Flow	`FALL_THROUGH`	1,680	1,559	121	7%
	`UNREACHABLE`	3,163	3,010	153	5%
Total		9,871	8,874	997	10%

Overall Process

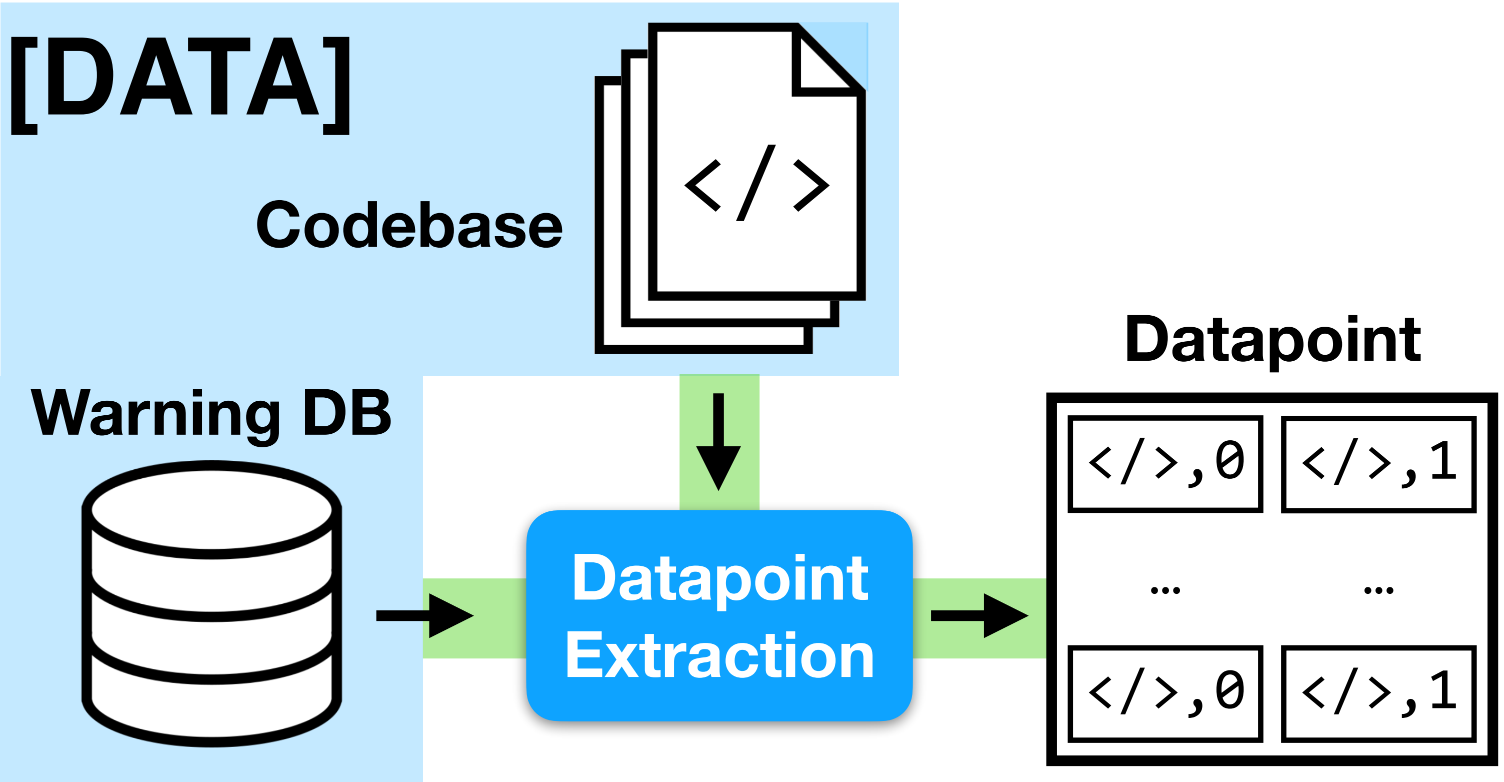


[DATA]

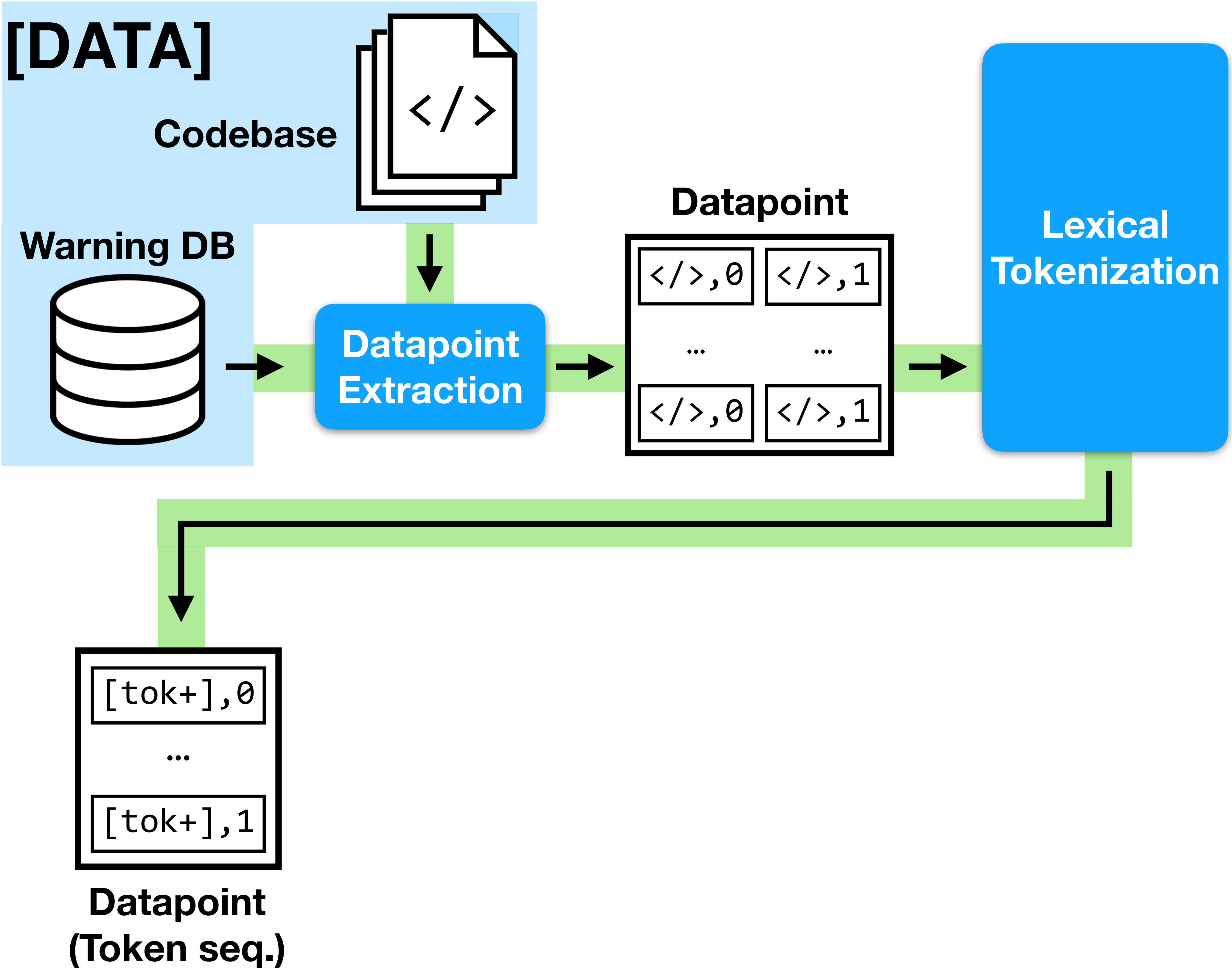
Codebase

Warning DB

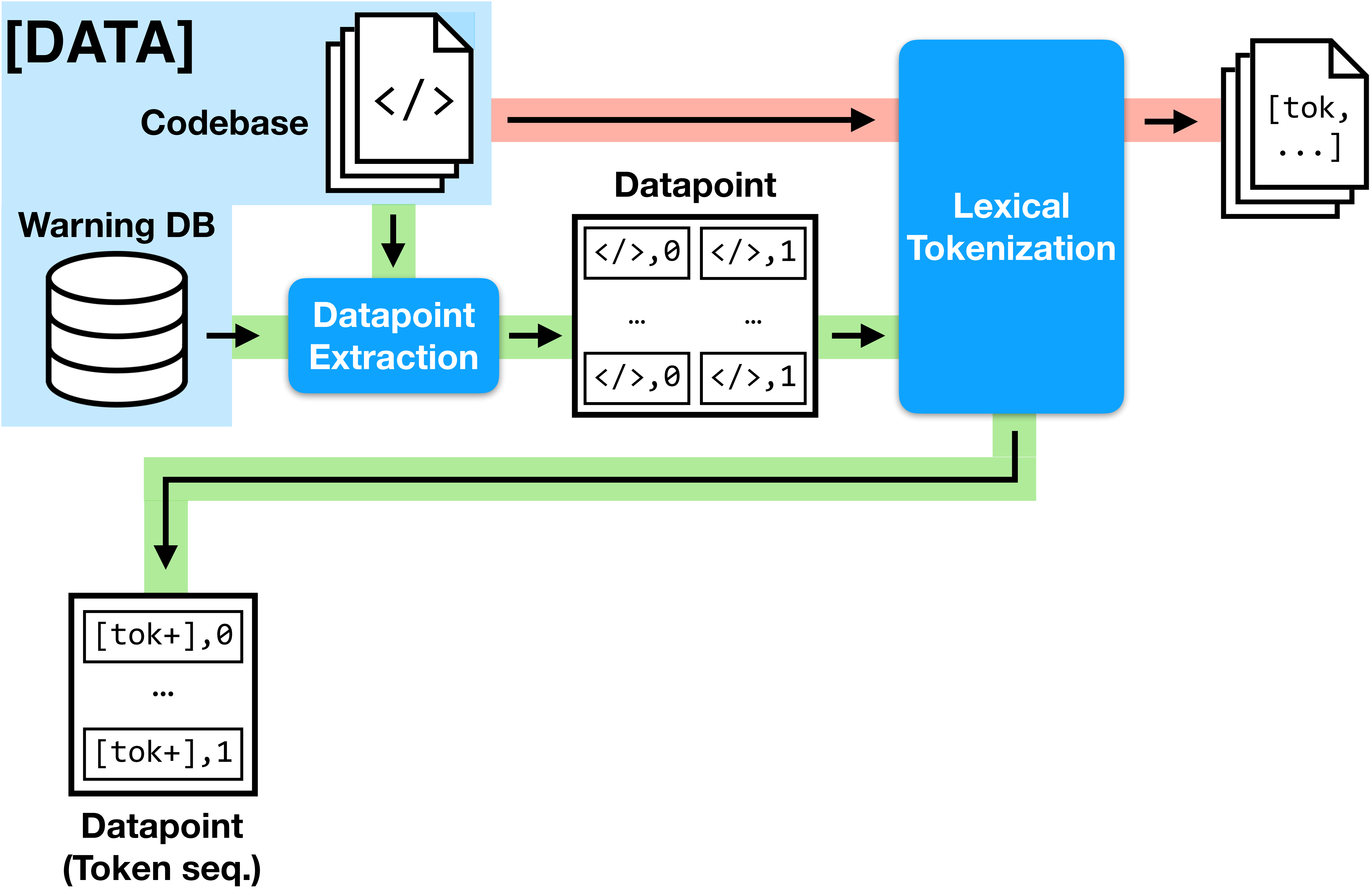
Overall Process



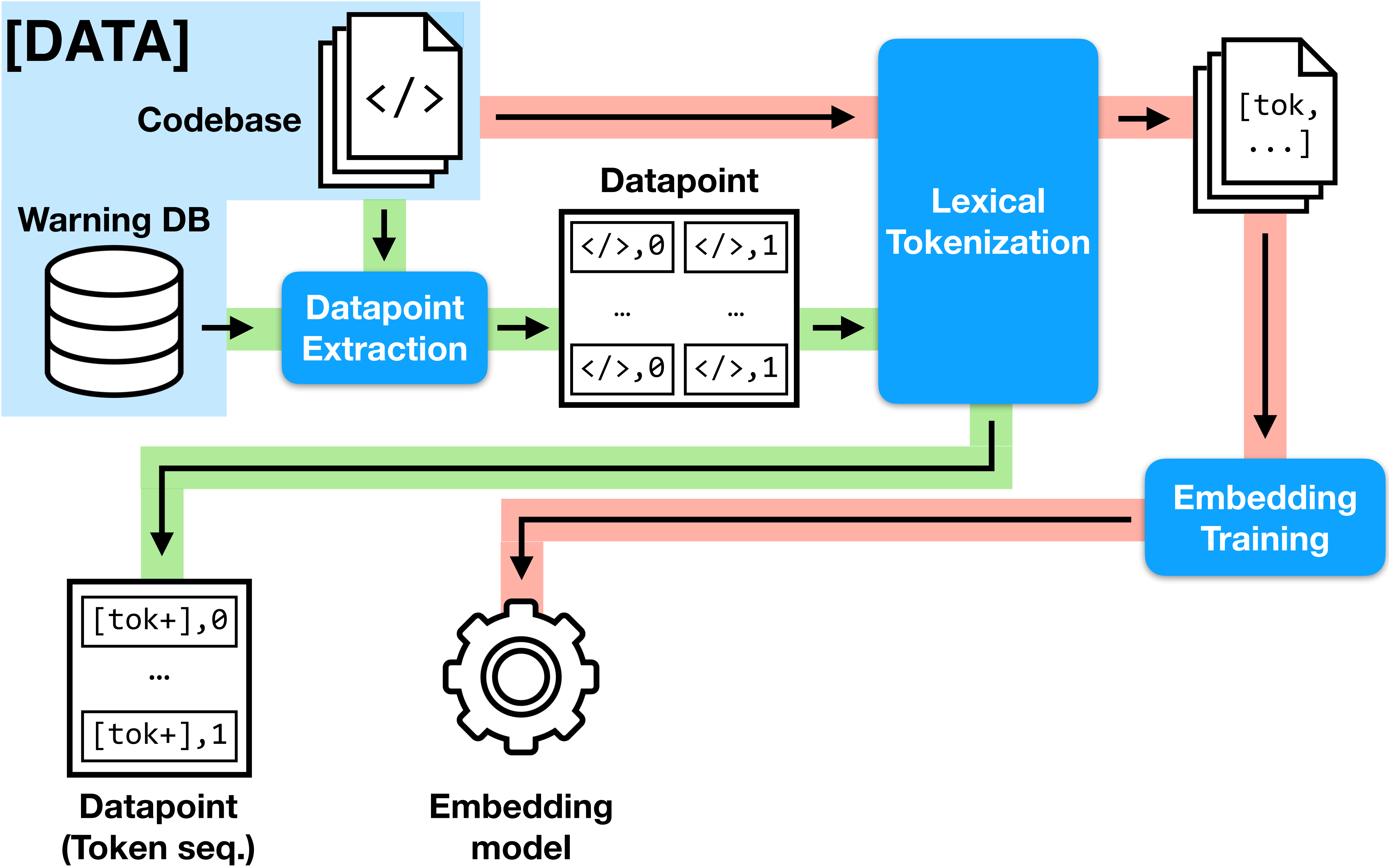
Overall Process



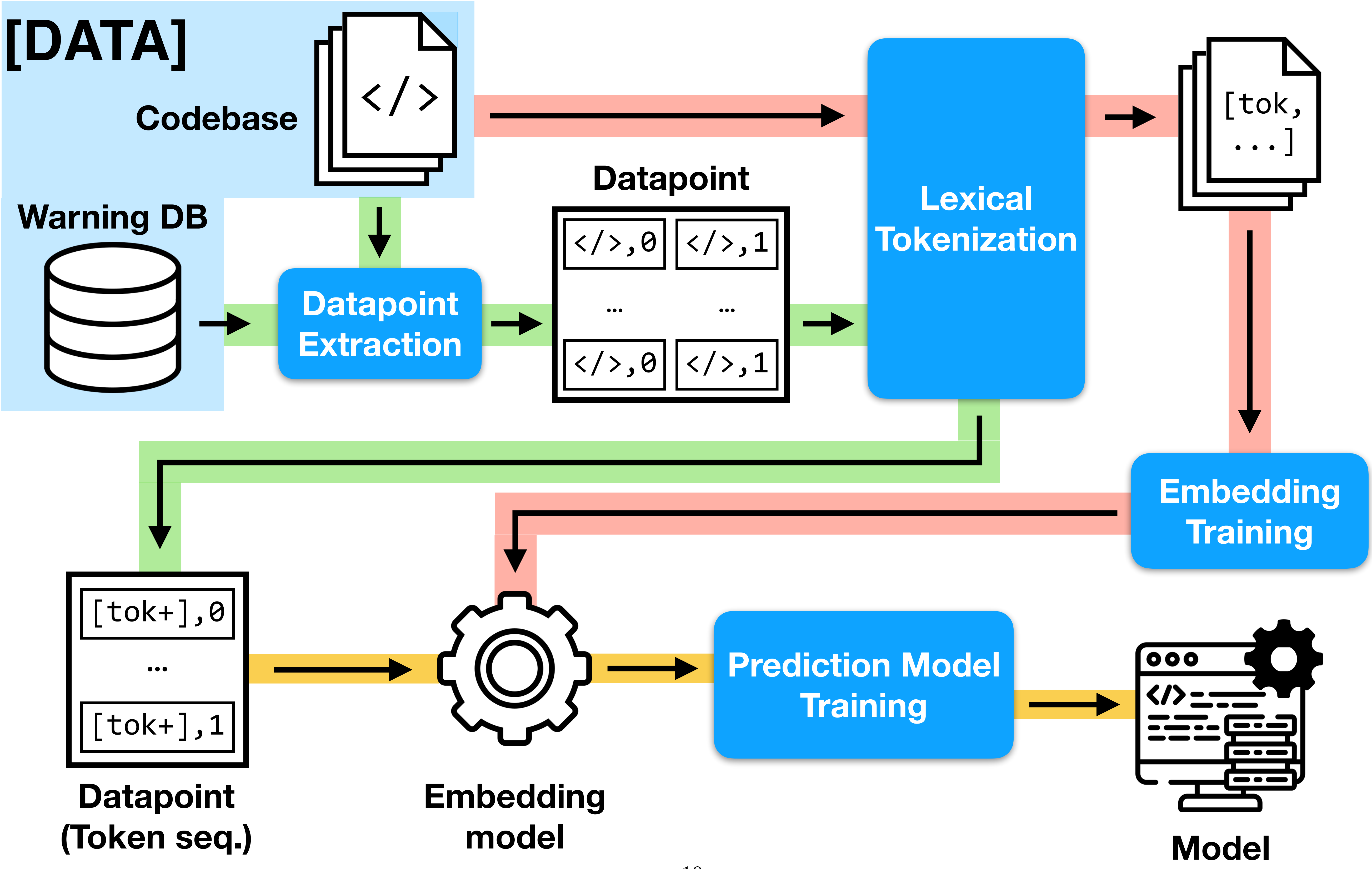
Overall Process



Overall Process



Overall Process



1. Datapoint Extraction

- A datapoint is a data representation of an alarm review case.
- A datapoint is defined to contain raw code data to support the alarm review.
- Each checker has a data point definition scheme that combines code snippets related to the warning trace.

- Cases
  - HANDLE_LEAK: 10 lines from the resource acquire point to the leak-point
  - FALL_THROUGH: 20 lines surrounding the exit-point of a case block

```
01  func() {
02    int fd = open(...); // acquire
...
06    if (x < y)
...
21    x = y
...
24    if (feof(fd) == true)
25      return;             // release
```

02-06

21-25

HANDLE_LEAK

```
01  switch (z) {
...
07    case ‘x’:         // intended
08    case ‘y’:         // fall
09    case ‘z’:         // through
10       x_or_y_or_z = 1;

12    case ‘a’:
...
```

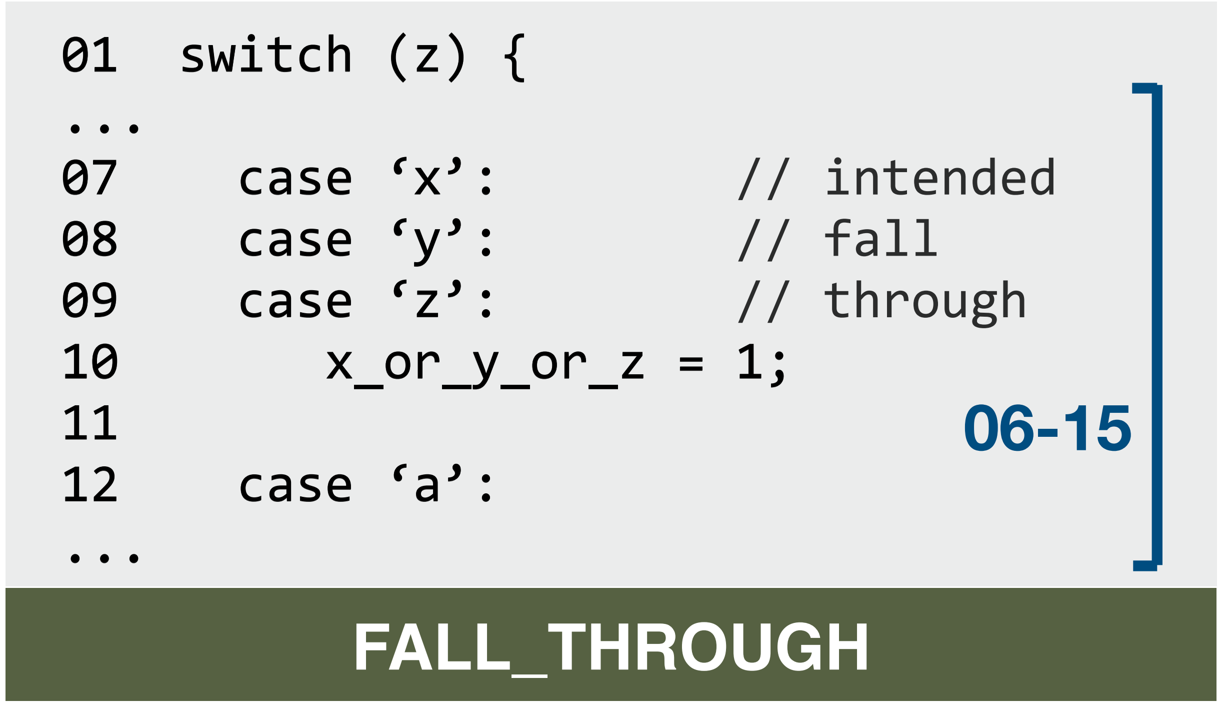
06-15

FALL_THROUGH

1. Datapoint Extraction

- A datapoint is a data representation of an alarm review case.
- A datapoint is defined to contain raw code data to support the alarm review.
- Each checker has a data point definition scheme that combines code snippets related to the warning trace.

- Cases
  - `HANDLE_LEAK`: 10 lines from the resource acquire point to the leak-point
  - `FALL_THROUGH`: 20 lines surrounding the exit-point of a case block



```
01  switch (z) {
...
07    case ‘x’:          // intended
08    case ‘y’:          // fall
09    case ‘z’:          // through
10       x_or_y_or_z = 1;
11
12    case ‘a’:
...
```

FALL_THROUGH

2. Lexical Tokenization

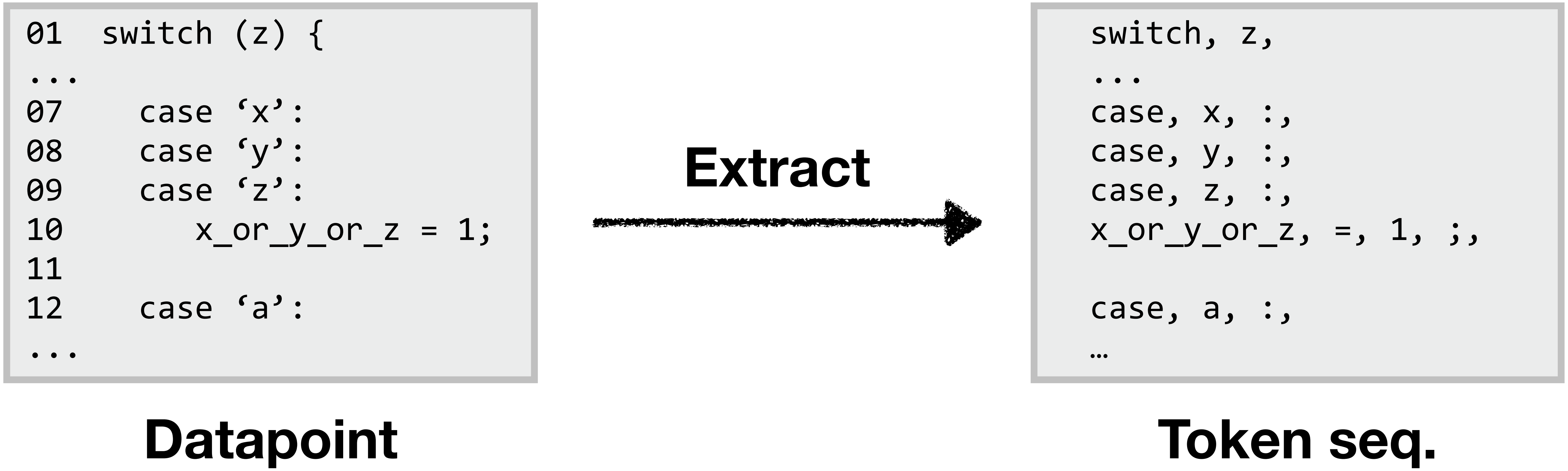
- Input: Datapoint, Output: Token sequence

```
switch (z) {
...
  case ‘x’:
  case ‘y’:
  case ‘z’:
     x_or_y_or_z = 1;

  case ‘a’:
...
```

Datapoint

2. Lexical Tokenization

- Input: Datapoint, Output: Token sequence
  1. Extract tokens (e.g. Identifier, operator, number) from datapoint



```
01  switch (z) {
...
07    case ‘x’:
08    case ‘y’:
09    case ‘z’:
10       x_or_y_or_z = 1;
11
12    case ‘a’:
...
```

Datapoint

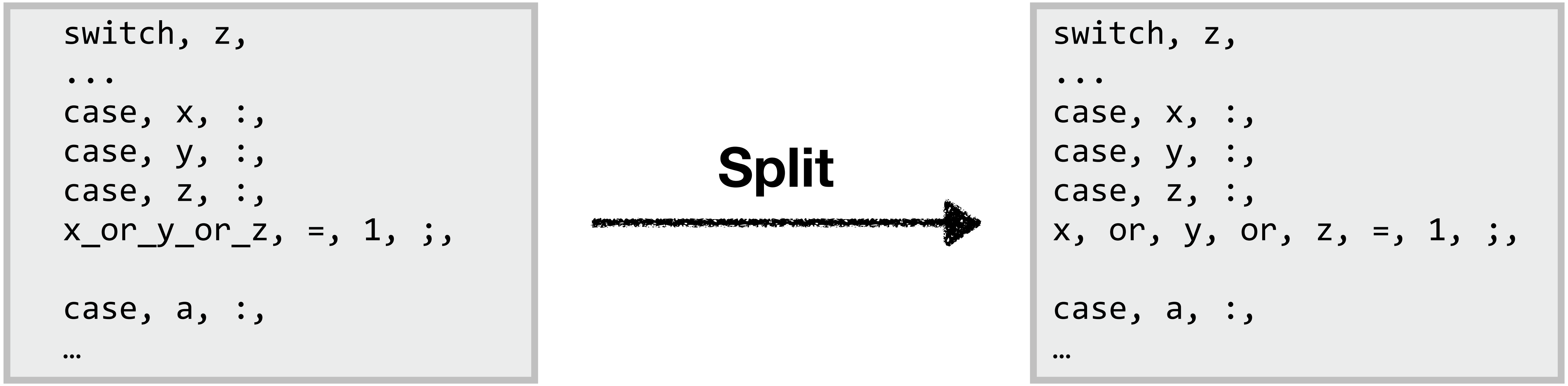
Extract

```
switch, z,
...
case, x, :,
case, y, :,
case, z, :,
x_or_y_or_z, =, 1, ;,

case, a, :,
…
```

Token seq.

2. Lexical Tokenization

- Input: Datapoint, Output: Token sequence
  1. Extract tokens (e.g. Identifier, operator, number) from datapoint
  2. Split `camelCase` and `snake_case` tokens



```
switch, z,
...
case, x, :,
case, y, :,
case, z, :,
x_or_y_or_z, =, 1, ;,

case, a, :,
…
```

Split

```
switch, z,
...
case, x, :,
case, y, :,
case, z, :,
x, or, y, or, z, =, 1, ;,

case, a, :,
…
```

Token seq.

Token seq. (splt ver.)

2. Lexical Tokenization

- Input: Datapoint, Output: Token sequence
  1. Extract tokens (e.g. Identifier, operator, number) from datapoint
  2. Split `camelCase` and `snake_case` tokens
  3. Insert special tokens (e.g. `NEWLINETOK` in FALL_THROUGH)

```
switch, z,
...
case, x, :,
case, y, :,
case, z, :,
x, or, y, or, z, =, 1, ;,

case, a, :,
…
```

Token seq. (splt ver.)

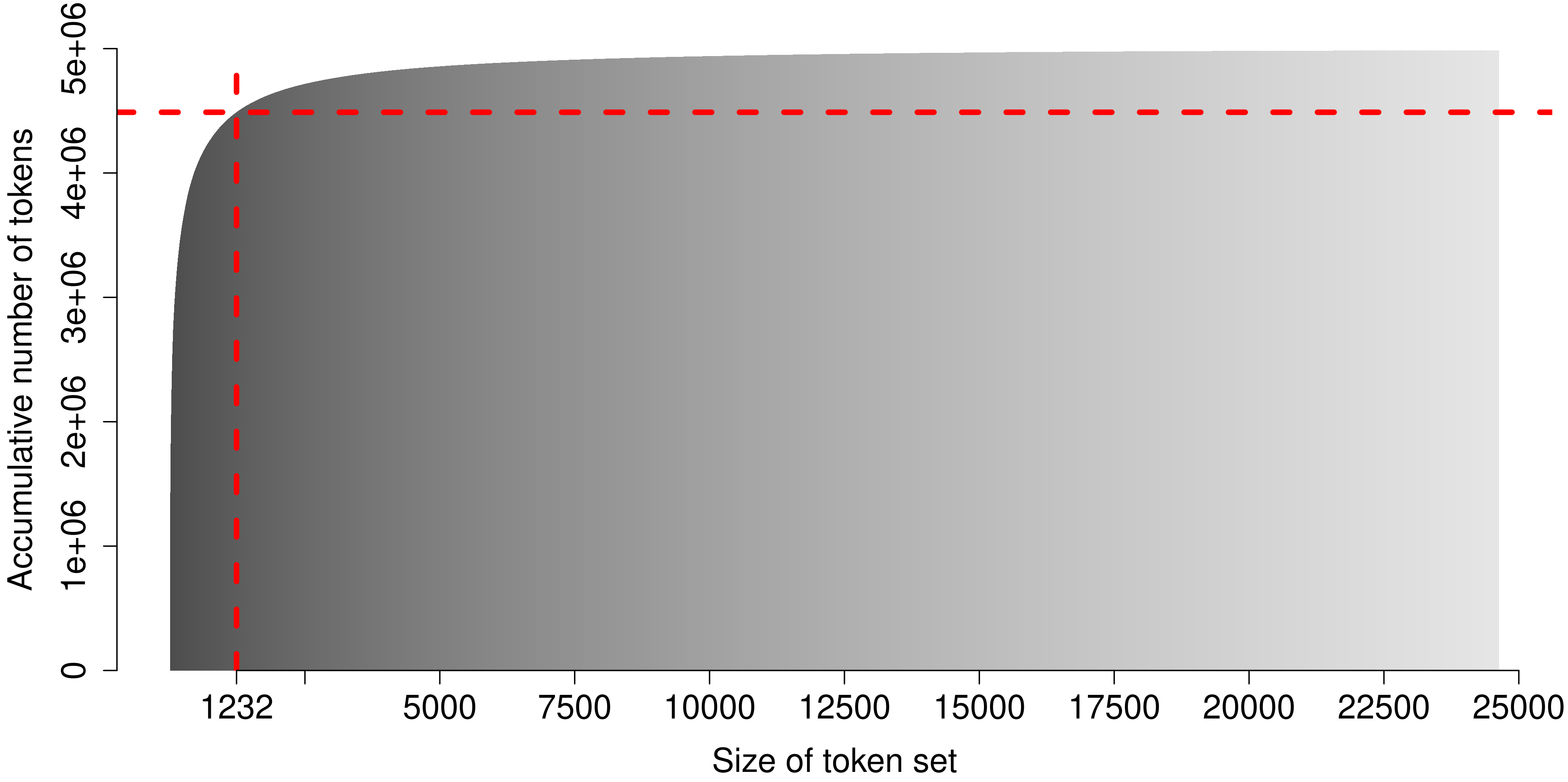
Add tokens →

```
switch, z,
...
case, x, :,
case, y, :,
case, z, :,
x, or, y, or, z, =, 1, ;,
NEWLINETOK,
case, a, :,
…
```

Token seq. (final ver.)

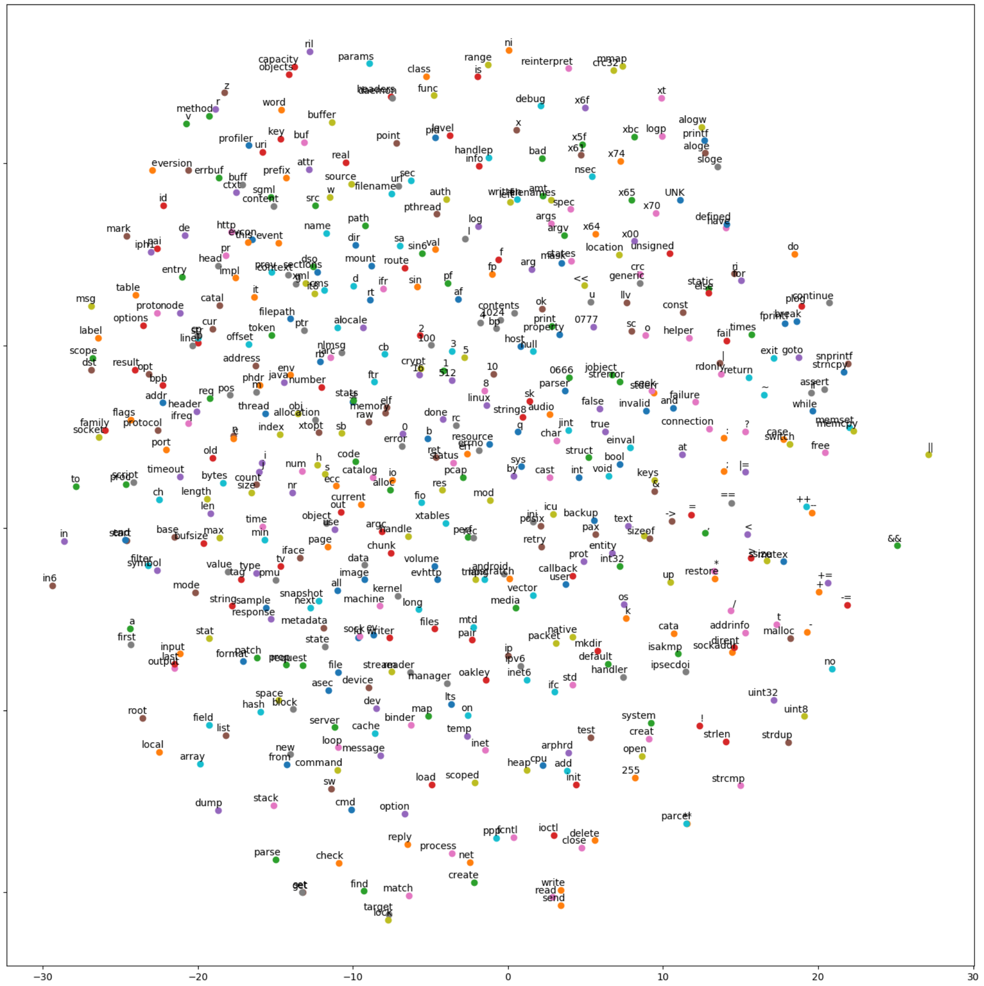
2.1. Define Vocabulary

- Select a small amount of frequent words, and remove all other infrequent words
  - to avoid overfitting
  - to reduce computational cost

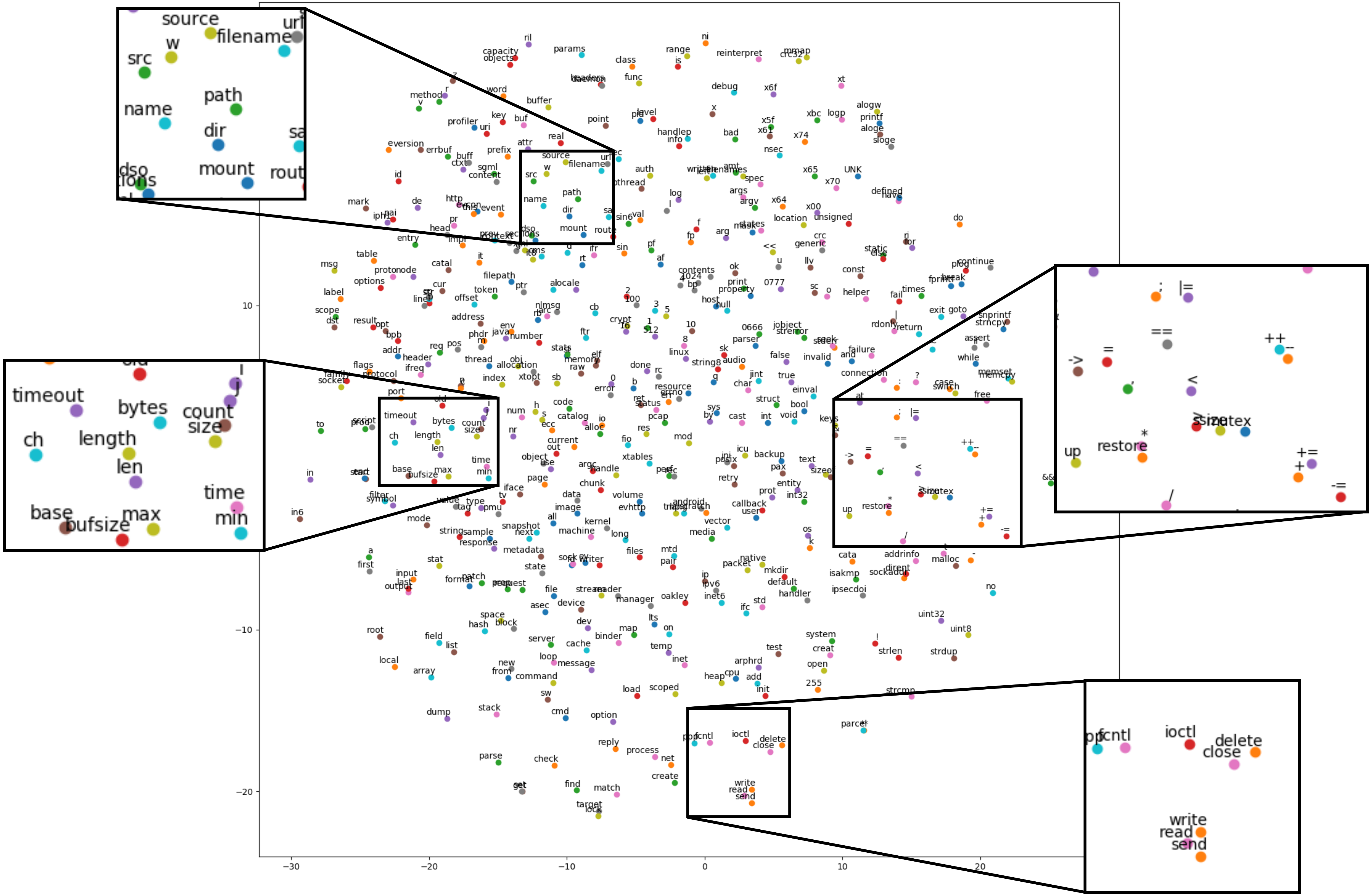


3. Word2Vec + CNN

- Word2Vec
    - Predictive modeling for learning vector embedding of words in a given corpus.
    - The semantic and syntactic patterns can be reproduced using vector arithmetic.
    - Two method: CBOW, Skip-gram
    - The hidden layer represents the embedding.



- Skip-gram
    - Input: target word
      Output: context(surrounding words)

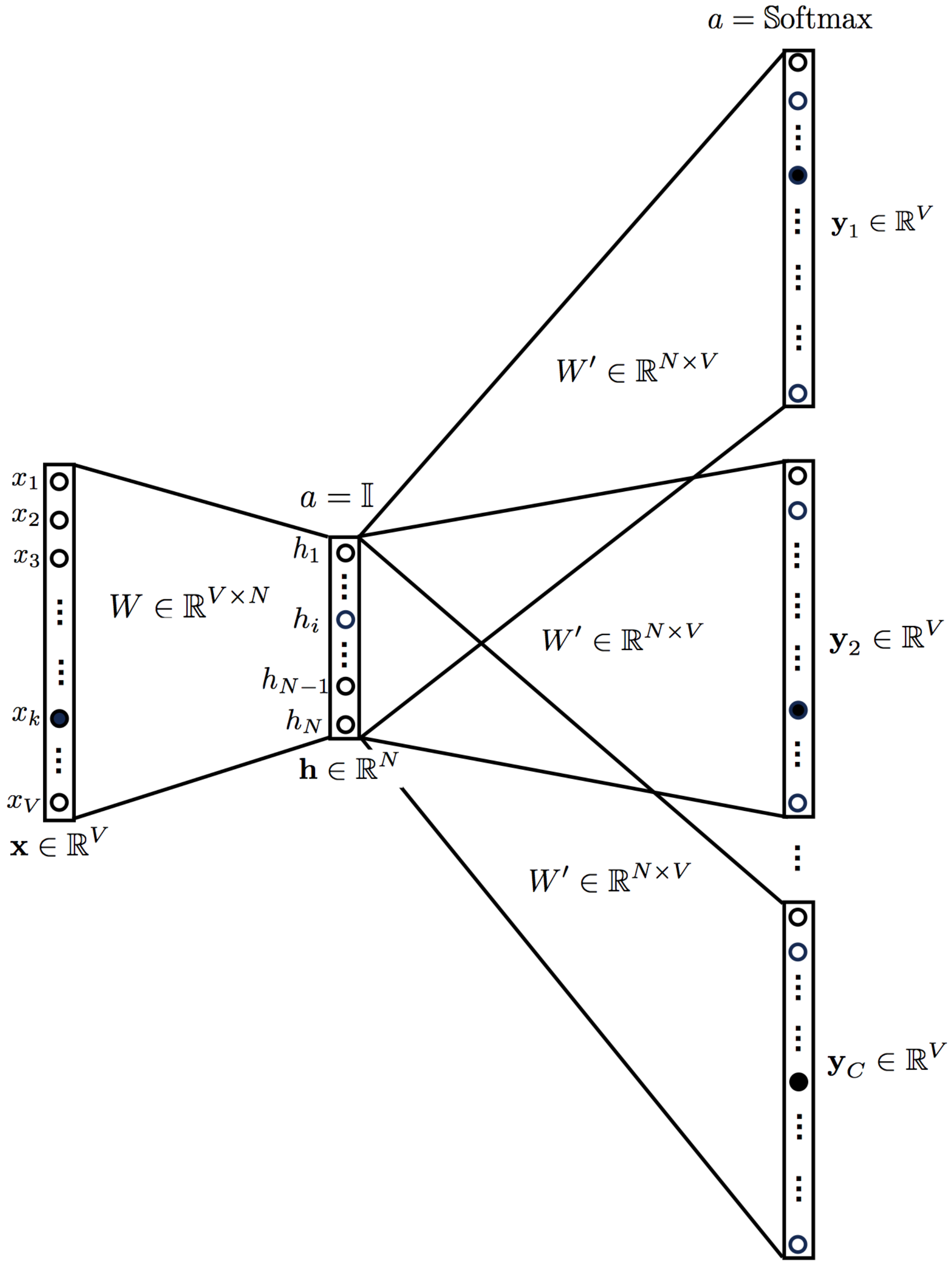


−30
−20
−10
0
10
20
10
0
−10
−20
source
url
filename
w
src
path
name
dir
mount
dso
tions
rout
sa
timeout
bytes
count
size
ch
length
len
time
base
bufsize
max
min
old
;
|=
==
++
--
->
=
<
sizeof
mutex
*
restore
up
+=
+
-=
/
fcntl
ioctl
delete
close
write
read
send
ril
params
capacity
objects
class
range
is
ni
reinterpret
mmap
crc32
debug
x6f
xt
method
r
v
word
buffer
headers
daemon
func
profiler
uri
key
buf
point
pla
level
handlep
info
bad
x5f
x61
xbc
logp
alogw
printf
aloge
sloge
eversion
errbuf
buff
ctxt
prefix
attr
real
x74
nsec
id
sgml
content
auth
pthread
written
filenames
amt
spec
x65
x70
UNK
defined
have
mark
iphdr
pai
de
http
this
event
log
args
argv
x64
x00
unsigned
do
entry
head
pr
impl
context
sections
val
sin
f
fp
arg
states
mask
location
crc
generic
static
else
ri
for
table
msg
it
catal
gms
fd
ifr
rt
pf
af
<<
u
llv
const
pkg
continue
proto
node
filepath
ptr
alocale
contents
1024
bp
ok
print
property
0777
sc
o
helper
fail
times
fprintf
break
options
label
cur
token
2
100
3
5
host
null
exit
goto
snprintf
strncpy
scope
dst
result
opt
address
offset
nlmsg
cb
crypt
16
512
1
10
0666
jobject
strerror
rdonly
return
bpb
phdr
java
number
ftr
8
linux
parser
seek
assert
addr
reg
pos
m
stats
elf
sk
false
invalid
and
failure
while
flags
header
thread
allocation
obj
memory
raw
done
rc
string8
audio
true
connection
memset
memcpy
family
protocol
ifreq
socket
index
xtopt
sb
0
b
resource
q
jint
char
case
switch
port
errno
err
einval
free
h
code
ret
status
struct
bool
at
num
catalog
sys
by
cast
int
void
keys
to
script
prot
ecc
alloc
io
res
nr
fio
mod
current
out
icu
posix
ini
backup
text
sizep
object
use
xtables
argc
handle
perf
fc
retry
pax
in
start
card
page
chunk
entity
int32
iface
data
volume
filter
symbol
tv
value
type
tag
pmu
image
evhttp
android
trap
attach
callback
user
os
in6
mode
all
kernel
vector
k
string
sample
snapshot
next
long
media
response
machine
metadata
a
first
stat
files
mtd
pair
cata
addrinfo
malloc
input
last
output
patch
format
prop
request
state
sock
fd
writer
native
packet
mkdir
isakmp
sockaddr
dirent
file
stream
reader
manager
oakley
ip
ipv6
inet6
default
handler
ipsecdoi
no
asec
device
std
ifc
space
hash
block
lts
uint32
root
field
list
dev
binder
map
on
system
uint8
server
cache
temp
creat
!
strlen
strdup
local
loop
message
inet
test
array
new
from
command
arphrd
open
heap
cpu
add
init
load
scoped
255
sw
strcmp
dump
stack
cmd
option
parcel
pop
reply
process
parse
check
net
create
get
set
find
match
target
lock

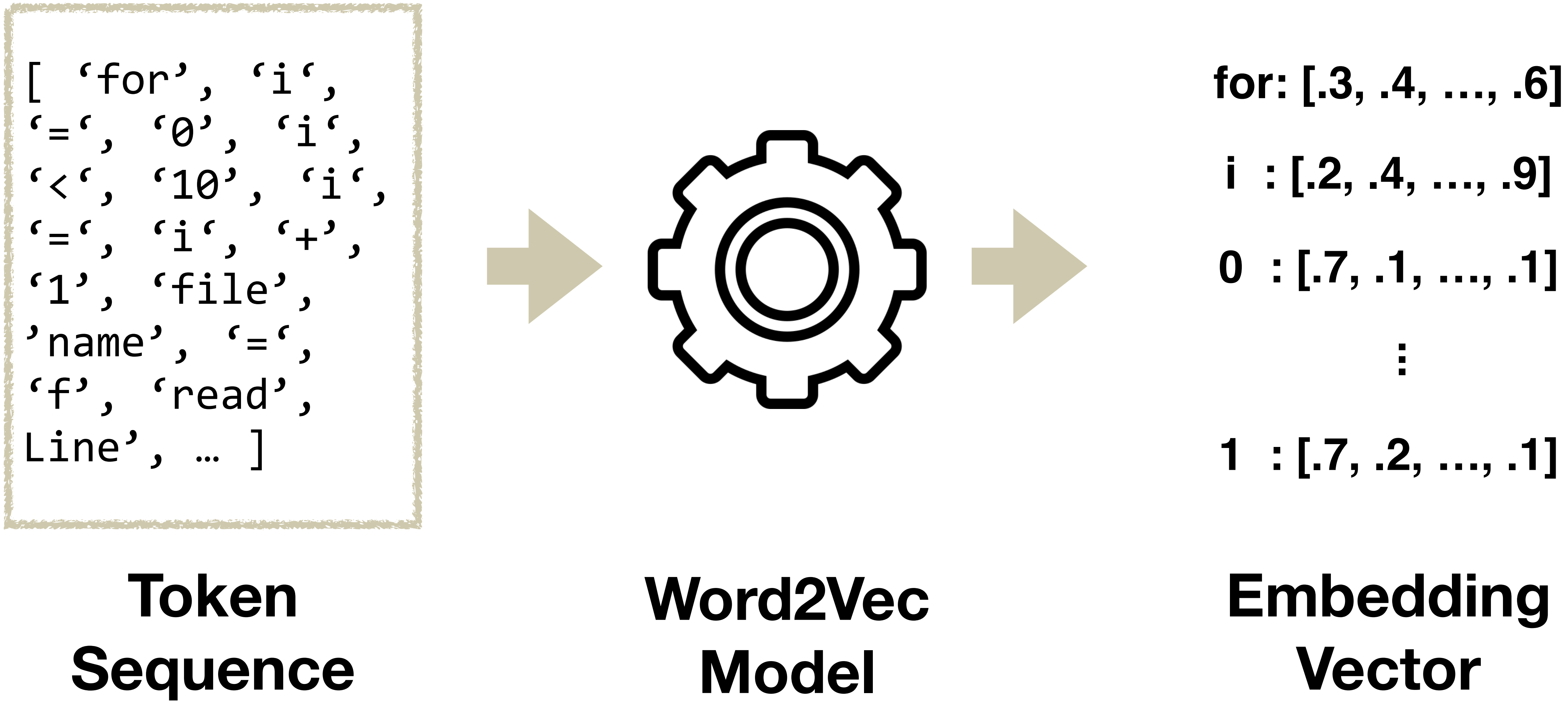
3. Word2Vec + CNN

- Word2Vec
  - Predictive modeling for learning vector embedding of words in a given corpus.
  - The semantic and syntactic patterns can be reproduced using vector arithmetic.
  - Two method: CBOW, Skip-gram
  - The hidden layer represents the embedding.

- Skip-gram
  - Input: target word
    Output: context(surrounding words)


$a = \mathbb{S}\text{oftmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
x_1
x_2
x_3
x_k
x_V
$\mathbf{x} \in \mathbb{R}^V$
$W \in \mathbb{R}^{V \times N}$
$a = \mathbb{I}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

3. Word2Vec + CNN



```
[ ‘for’, ‘i‘,
‘=‘, ‘0’, ‘i‘,
‘<‘, ‘10’, ‘i‘,
‘=‘, ‘i‘, ‘+’,
‘1’, ‘file’,
’name’, ‘=‘,
‘f’, ‘read’,
Line’, … ]
```

Token Sequence

Word2Vec Model

for: [.3, .4, …, .6]

i : [.2, .4, …, .9]

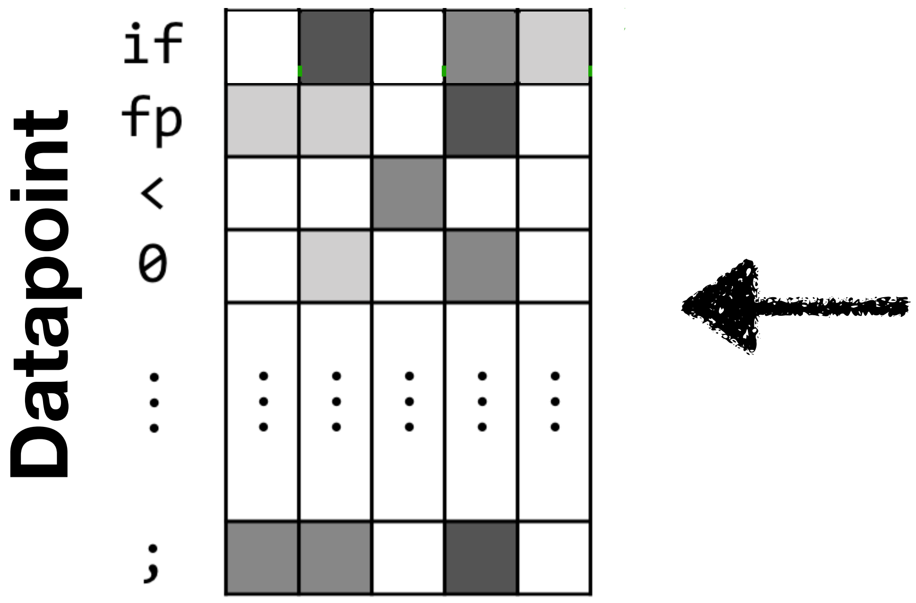
0 : [.7, .1, …, .1]

⋮

1 : [.7, .2, …, .1]

Embedding Vector

3. Word2Vec + CNN



for: [.3, .4, …, .6]

i : [.2, .4, …, .9]

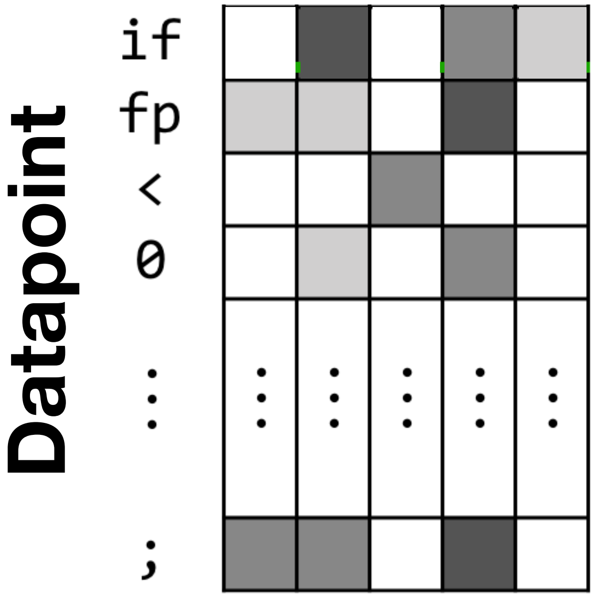
0 : [.7, .1, …, .1]

⋮

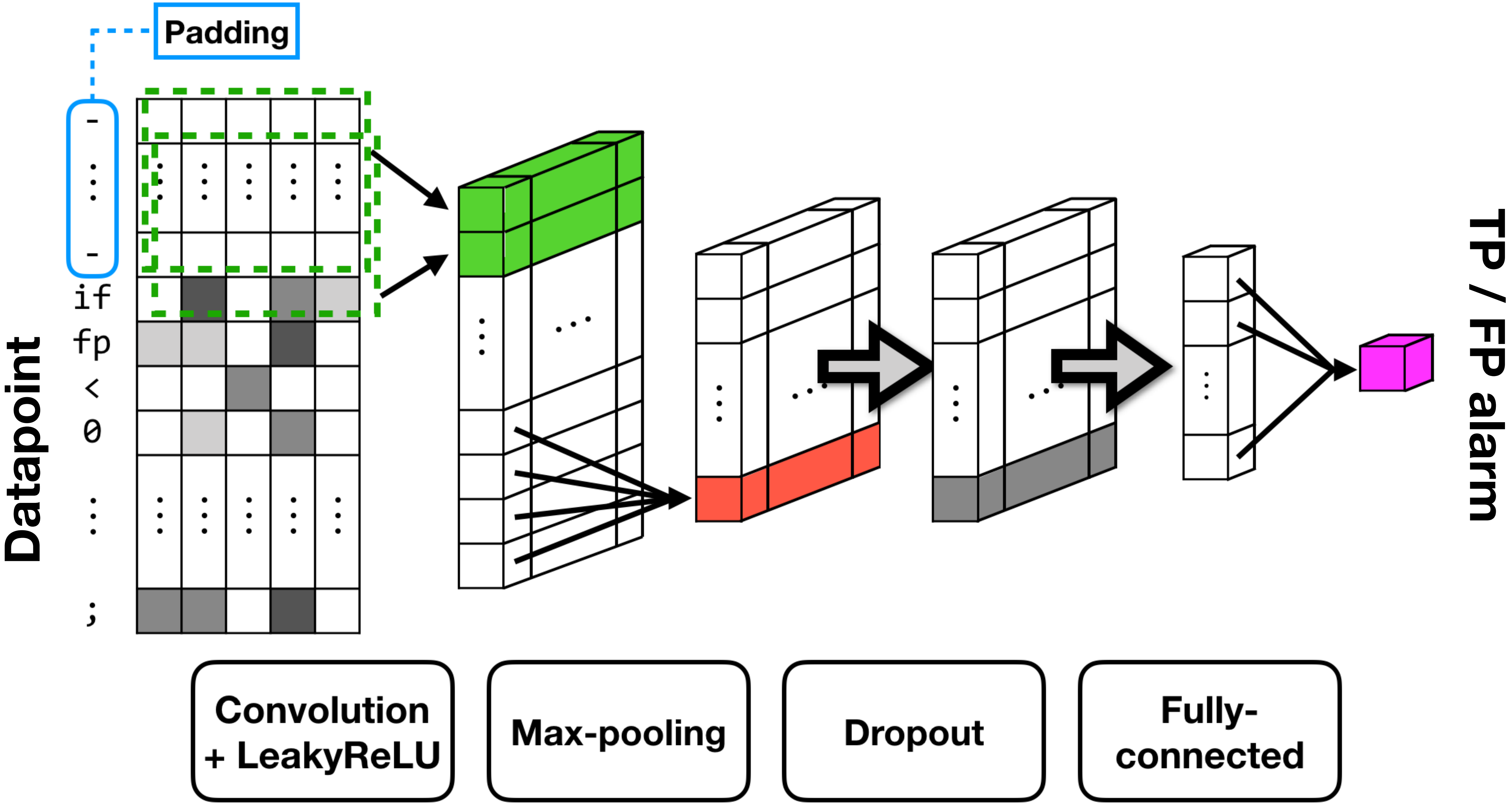
1 : [.7, .2, …, .1]

Embedding Vector

3. Word2Vec + CNN



3. Word2Vec + CNN



Convolution + LeakyReLU

Max-pooling

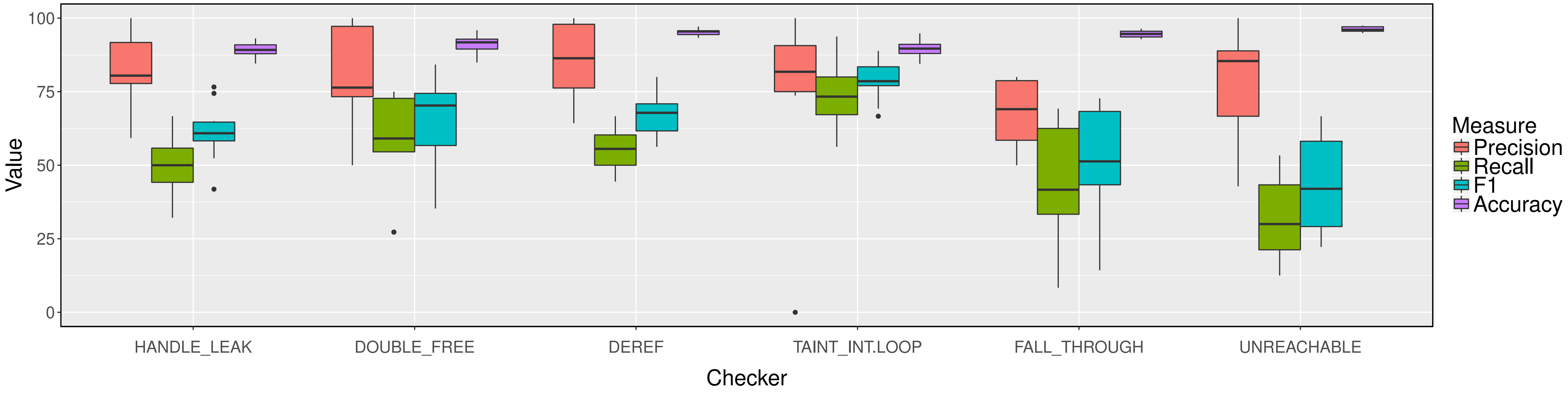
Dropout

Fully-connected

Experiment Setup

- Model configurations
  - An embedding size of 128 with **Word2Vec** implemented using Tensorflow
  - We trained the **CNN** classifier for 150 epochs, using the mini-batch size of 10 with Keras.
- Environment
  - Ubuntu 14.04 LTS, running on Intel Core i7-6700K with 32GB RAM
  - The TensorFlow backend used NVidia CUDA 8.0, running on NVidia GTX1080 GPU with 12GB memory
- Evaluation
  - 10-fold cross validation

Result



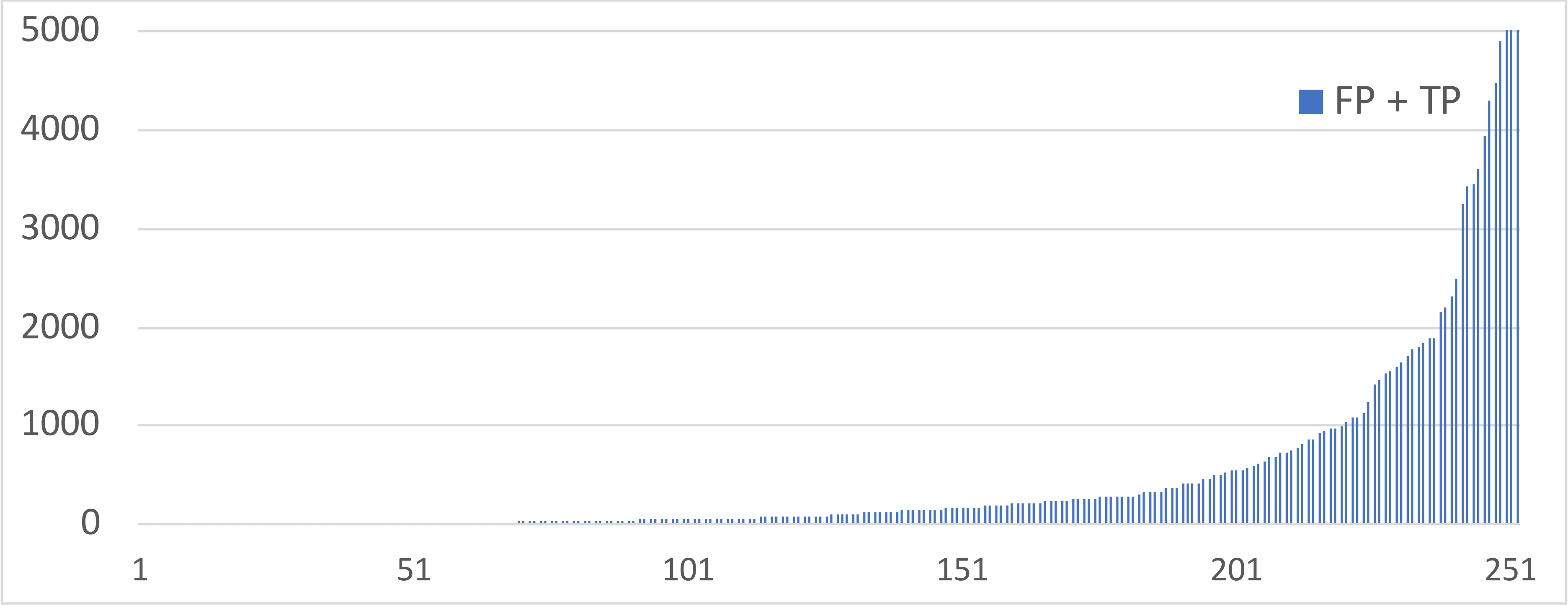
Average precision: **79.72%**, average recall: **51.09%**

Average F1: **61.18%**, average accuracy: **92.64%**

Discussion

- Data, data, data…
  - overfitting
  - collaboration
  - open data

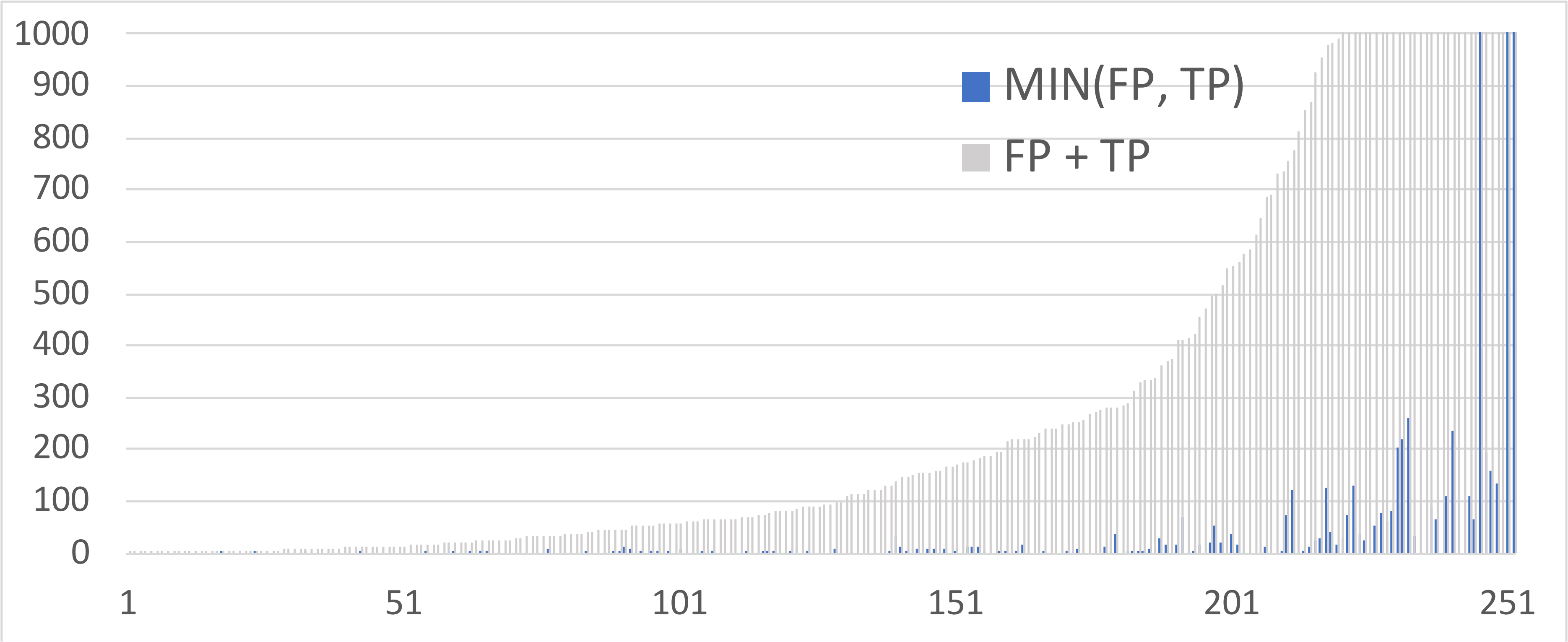
- Where to put the classifier in CI pipeline?
  - filtering static analyzer output
  - assisting review
  - assisting audit



Discussion

- Data, data, data…
  - overfitting
  - collaboration
  - open data

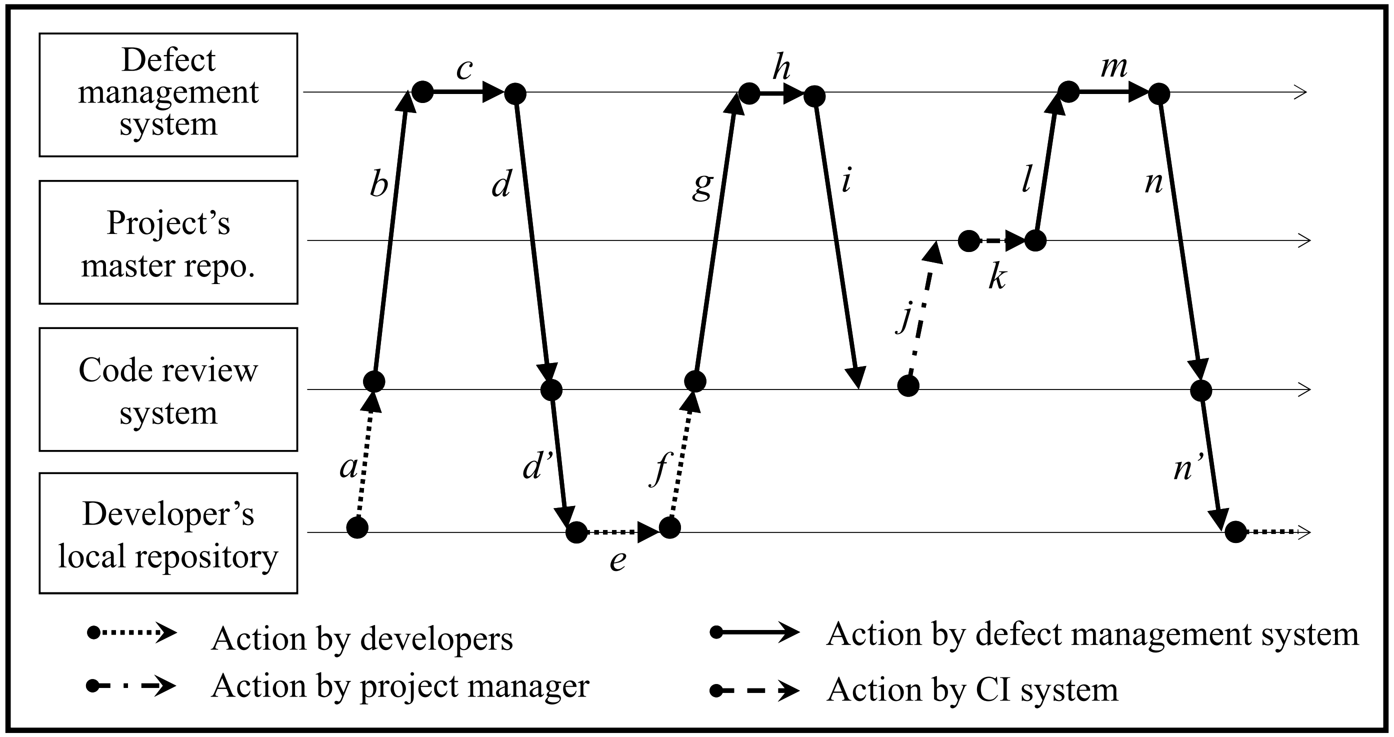
- Where to put the classifier in CI pipeline?
  - filtering static analyzer output
  - assisting review
  - assisting audit

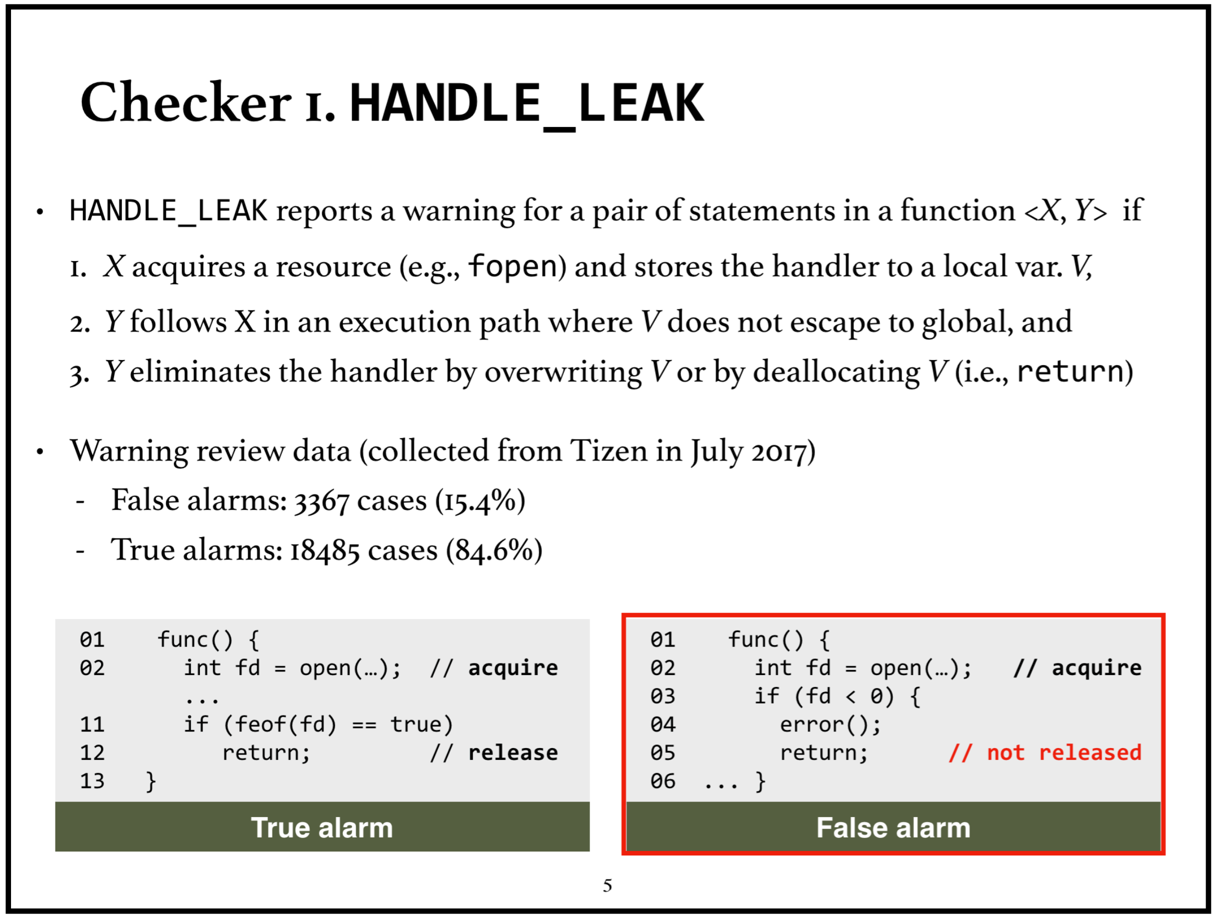


Discussion

- Data, data, data…
  - overfitting
  - collaboration
  - open data

- Where to put the classifier in CI pipeline?
  - filtering static analyzer output
  - assisting review
  - assisting audit





Checker 1. HANDLE_LEAK

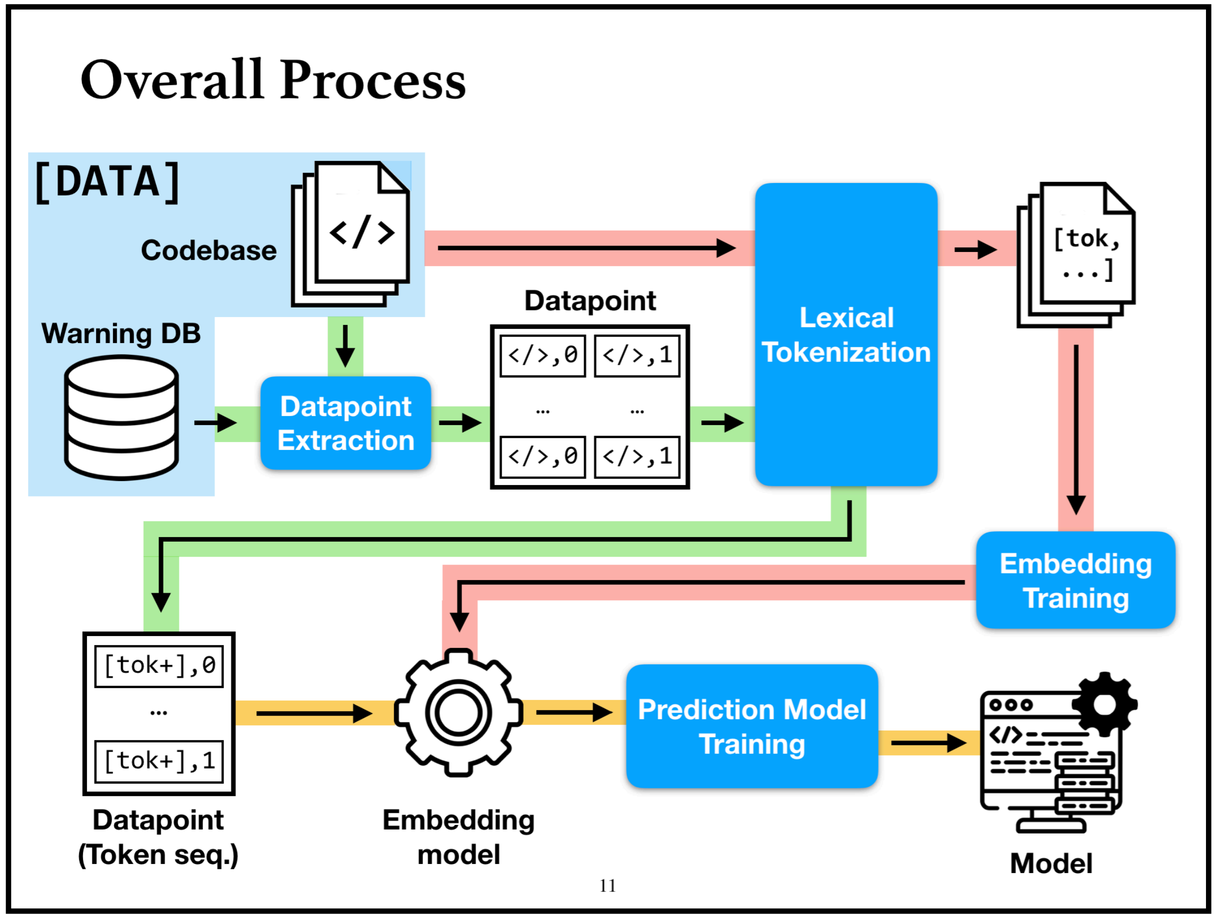
- HANDLE_LEAK reports a warning for a pair of statements in a function <*X*, *Y*> if
  1. *X* acquires a resource (e.g., fopen) and stores the handler to a local var. *V*,
  2. *Y* follows X in an execution path where *V* does not escape to global, and
  3. *Y* eliminates the handler by overwriting *V* or by deallocating *V* (i.e., return)
- Warning review data (collected from Tizen in July 2017)
  - False alarms: 3367 cases (15.4%)
  - True alarms: 18485 cases (84.6%)

```
func() {
  int fd = open(…);  // acquire
  ...
  if (feof(fd) == true)
     return;          // release
}
```

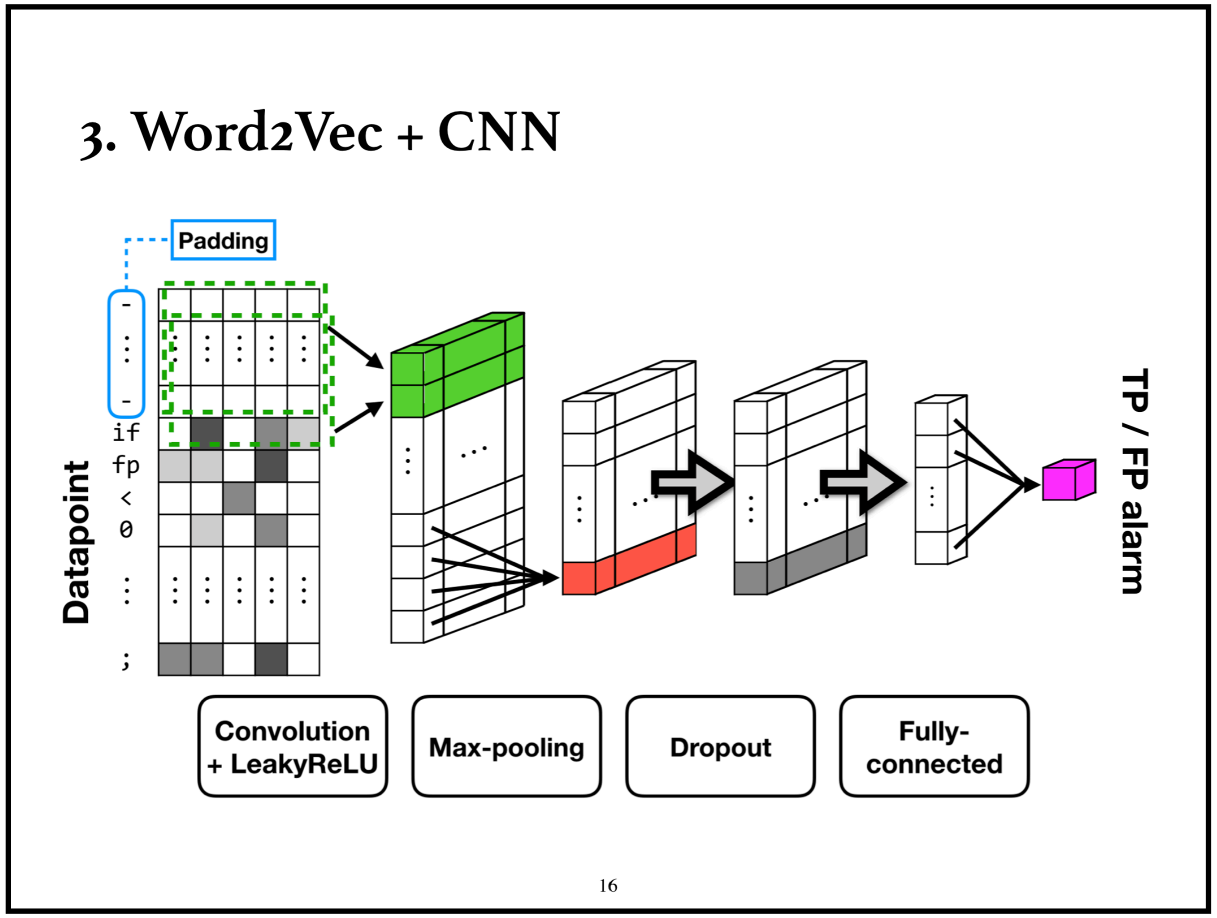
True alarm

```
func() {
  int fd = open(…);    // acquire
  if (fd < 0) {
    error();
    return;        // not released
... }
```

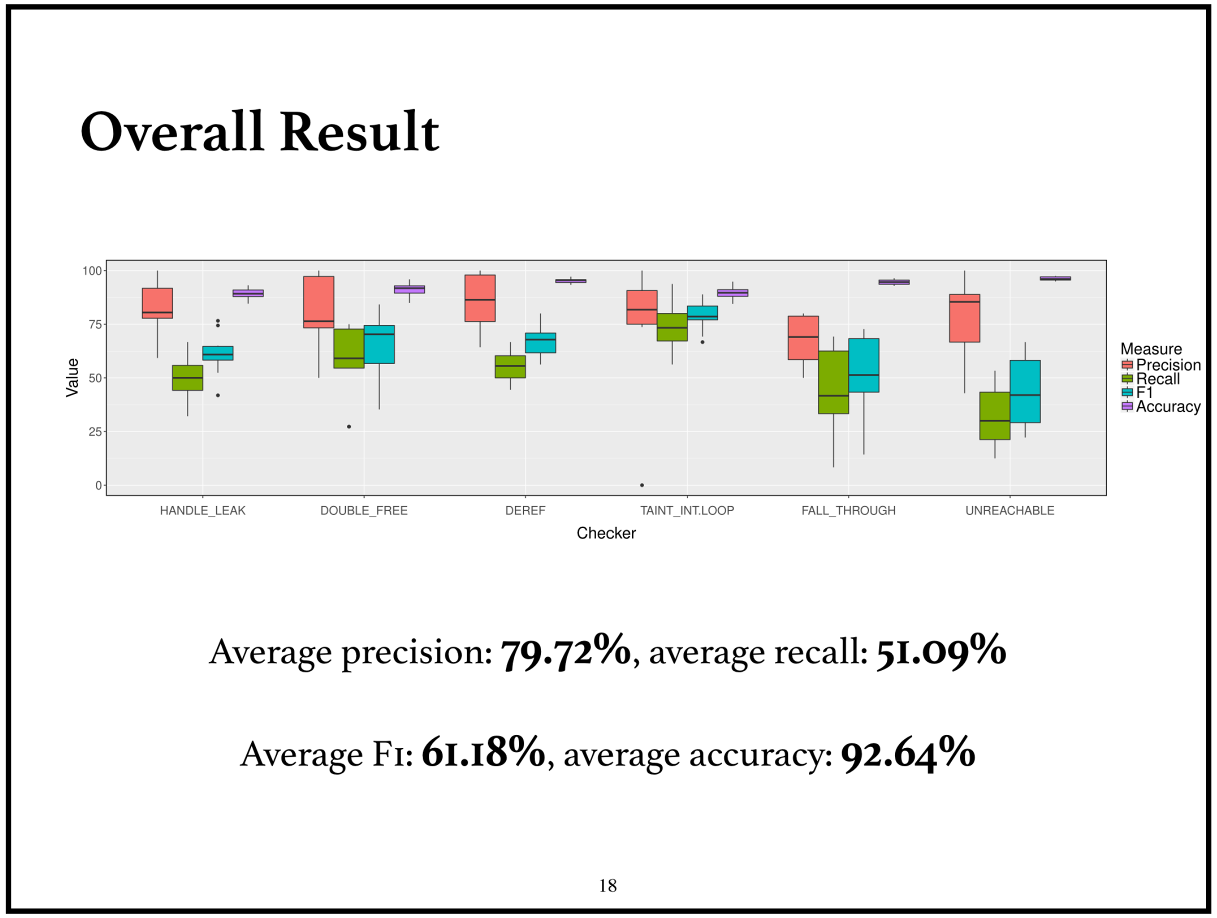
False alarm



Overall Process



3. Word2Vec + CNN



Overall Result

Average precision: **79.72%**, average recall: **51.09%**

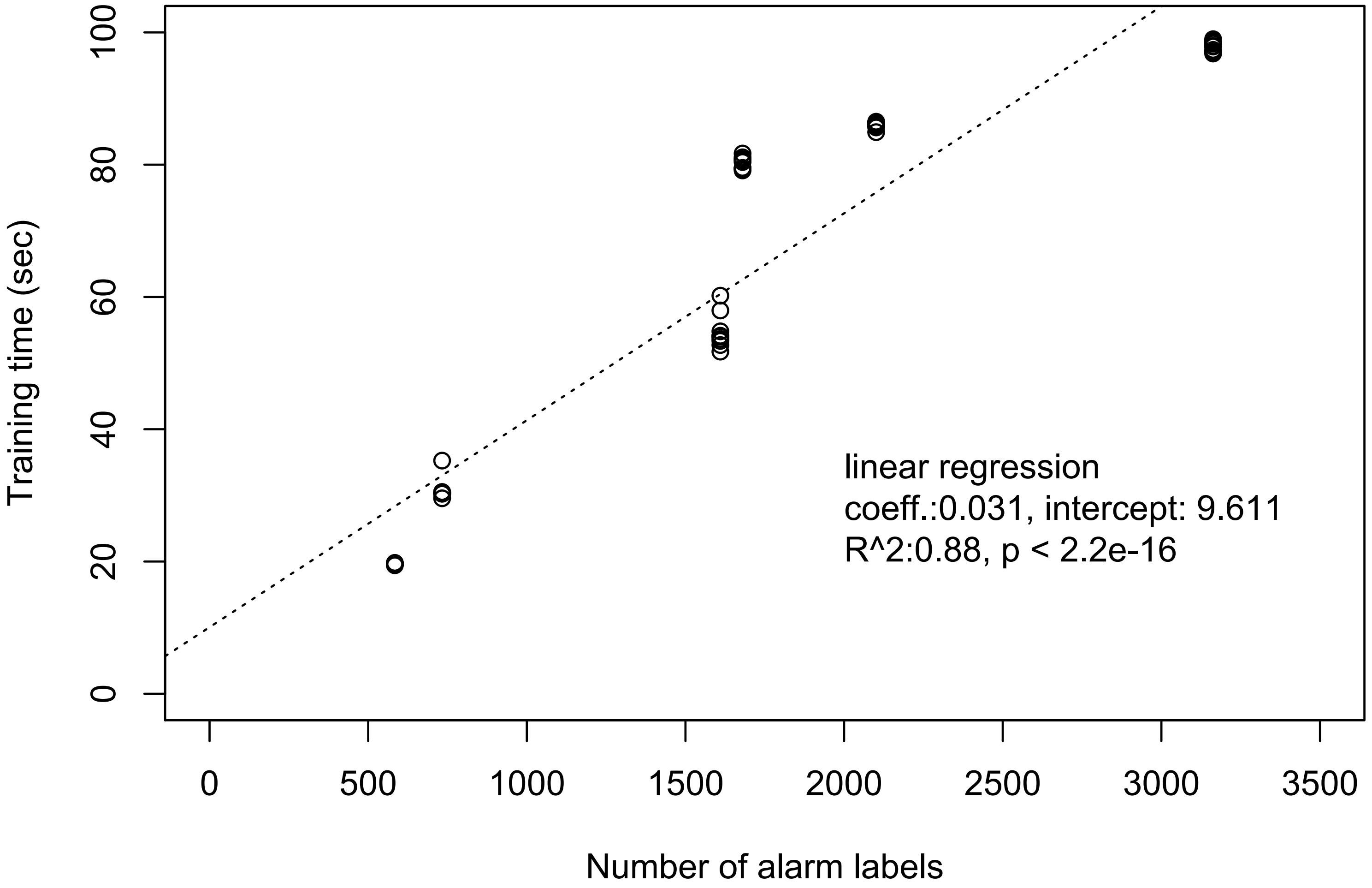
Average F1: **61.18%**, average accuracy: **92.64%**

Appendix A. Detailed results

TABLE II: Average accuracy results of ten-fold cross validation for 6 checkers

Checker	Precision		Recall		F1		Accuracy		Avg. # of Predicted / Actual	
	Mean	Var.	Mean	Var.	Mean	Var.	Mean	Var.	TP Alarms	FP Alarms
HANDLE_LEAK	81.80%	186.65	49.74%	90.54	61.24%	90.06	89.27%	7.15	143.9 / 133.4	17.1 / 27.6
DOUBLE_FREE	79.39%	293.09	57.50%	289.36	64.84%	229.52	90.99%	10.57	65.0 / 62.2	8.3 / 11.1
DEREF	85.70%	144.97	55.53%	53.56	66.87%	48.30	95.24%	1.08	198.1 / 191.9	12.0 / 18.2
TAINT_INT.LOOP	85.98%	101.06	73.95%	137.50	78.66%	47.64	89.50%	9.38	44.9 / 43.0	13.5 / 15.4
FALL_THROUGH	67.99%	108.47	44.42%	332.34	52.28%	293.16	94.64%	1.43	160.3 / 155.9	7.7 / 12.1
UNREACHABLE	77.48%	399.67	31.41%	216.05	43.20%	290.30	96.20%	0.84	310.0 / 301.0	6.3 / 15.3
Average	79.72%	-	51.09%	-	61.18%	-	92.64%	-	-	-

Appendix B. Scalability



The training time increases linearly as the number of alarms increases.

All training finished within 100 seconds.

Appendix C. Vocabulary size

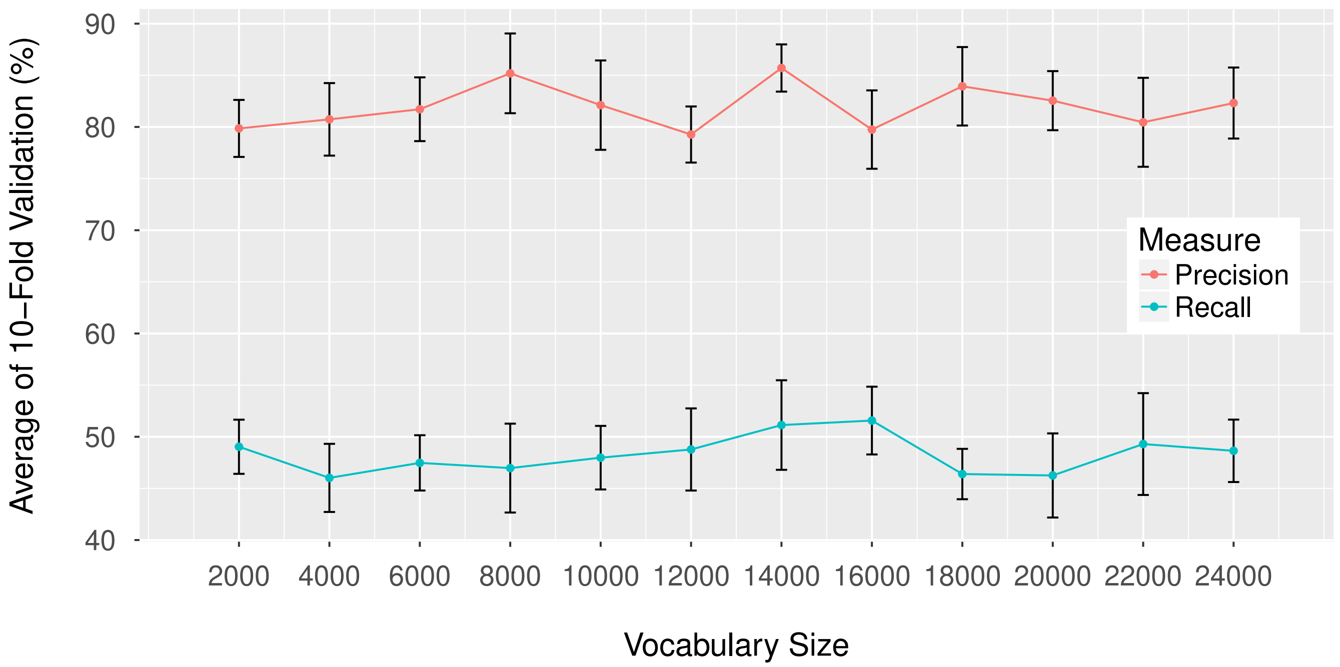


Fig. 13: Change of average cross validation precision and recall of `HANDLE_LEAK` classifier with varying vocabulary sizes: reducing the vocabulary size does not significantly damage the results of training.

While there are fluctuations, the precision level does not drop much below 80%, while maintaining similar levels of recall values.