17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

QUALITY ASSURANCE: SAMPLING

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

READING ASSIGNMENT NOV 16

Pohl, Klaus, and Andreas Metzger. "Software product line testing." Communications of the ACM 49, no. 12 (2006): 78-81.

Greiler, Michaela, Arie Van Deursen, and Margaret-Anne Storey. "Test confessions: a study of testing practices for plug-in systems." In Software Engineering (ICSE), 2012 34th International Conference on, pp. 244-254. IEEE, 2012

LEARNING GOALS

Understand the product-line specific QA challenges

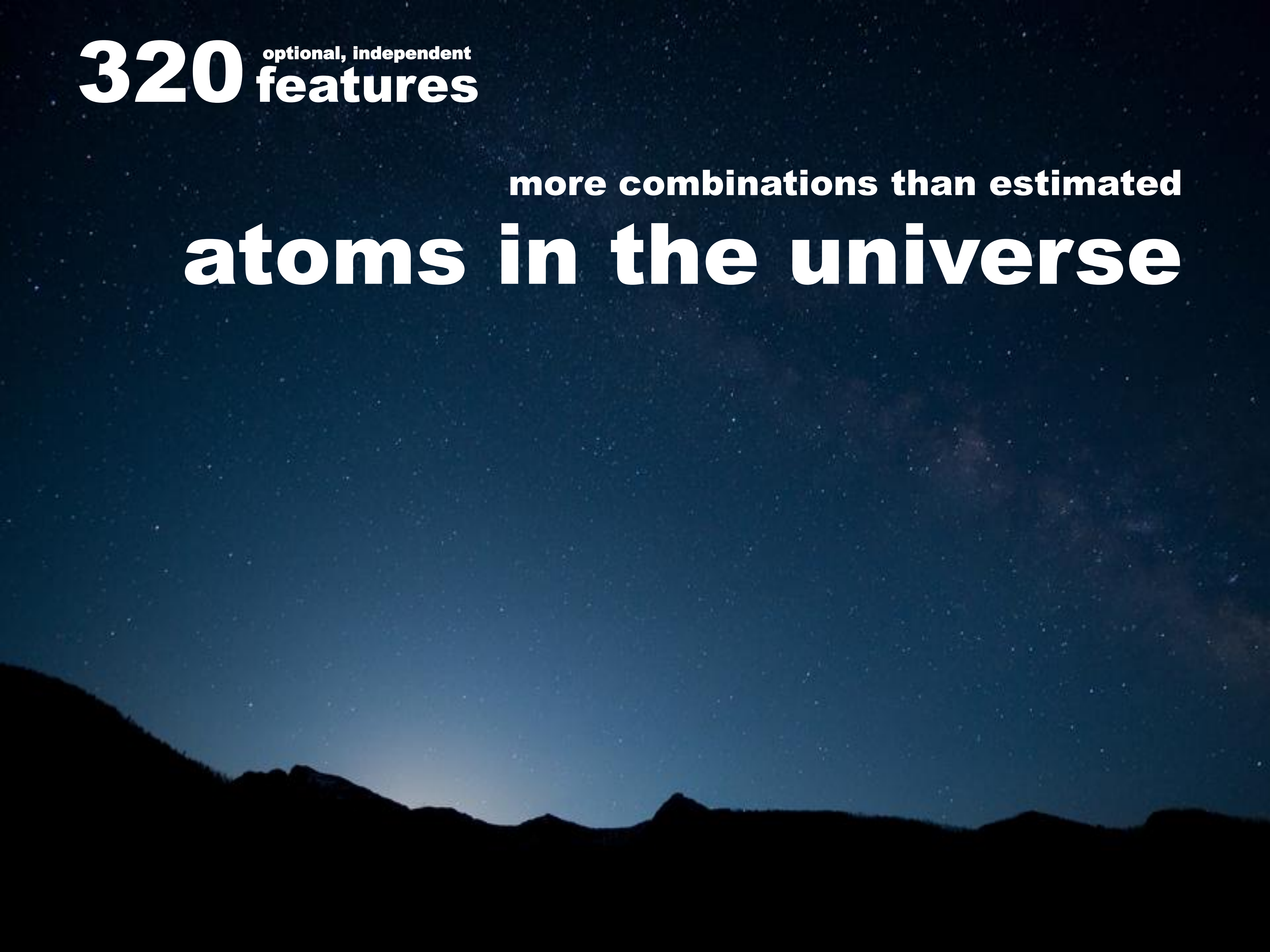
Differentiate and understand different sampling strategies, their limitations, and their tradeoffs

Select a suitable sampling strategy for a given project

EXPLOENTIAL CONFIGURATION SPACES

usually finite

but huge


320
optional, independent
features
more combinations than estimated
atoms in the universe

Configuration Bug

System exhibits bug when selecting multiple options in specific combination

```c
// Other definitions..
#ifdef SPLT
void png_handle_sPLT(){
    #ifdef POINTER
        png_sPLT_entryp p;
    #endif
    // Lines of code..
    #ifdef POINTER
        p = palette + i;
        p->red = *start++;
    #else
        p = new_palette;
        p[i].red = *start++;
    #endif
}
#endif
// More definitions..
```

Configuration 1

```c
#define SPLT
#define POINTER
```
✓

Configuration 2

```c
#undef SPLT
#define POINTER
```
✓

Configuration 3

```c
#define SPLT
#undef POINTER
```
⊗

Configuration 4

```c
#undef SPLT
#undef POINTER
```
✓

✓ Compilation succeed ⊗ Compilation error

```
void main() {
  int i = 0;
  int j = 10;
#ifdef PLUS
  i = i + 1;
#endif
#ifdef DIV
  j = j / i;
#endif
  printf(“%d - %d”, i, j);
}
```

DIV & !PLUS:

Floating point exception (core dumped)

Detecting Compilation Errors in Busybox Configurations

bunzip2.c, uncompress.c, … → check_signature16() → open_transformer.c

compiled if
BUNZIP2 || UNCOMPRESS || …

compiled if
FEATURE_COMPRESS_USAGE
MODINFO || RPM || GUNZIP ||
FEATURE_SEAMLESS_XZ || …

```
check_signature16,  init_transformer_aux_data, and open_transformer are
sometimes called in contexts where file open_transformer.c is not compiled.

The problem occurs if

undef CONFIG_FEATURE_COMPRESS_USAGE
undef CONFIG_FEATURE_SEAMLESS_XZ
undef CONFIG_MODINFO
undef CONFIG_FEATURE_SEAMLESS_BZ2
undef CONFIG_FEATURE_SEAMLESS_Z
undef CONFIG_RPM
undef CONFIG_GUNZIP
undef CONFIG_FEATURE_SEAMLESS_GZ
undef CONFIG_FEATURE_SEAMLESS_LZMA
undef CONFIG_INSMOD

and any of the following is activated:

CONFIG_UNXZ
CONFIG_BUNZIP2
CONFIG_UNCOMPRESS
```

11-way interaction. First repair attempt resulted in 26-way interaction.

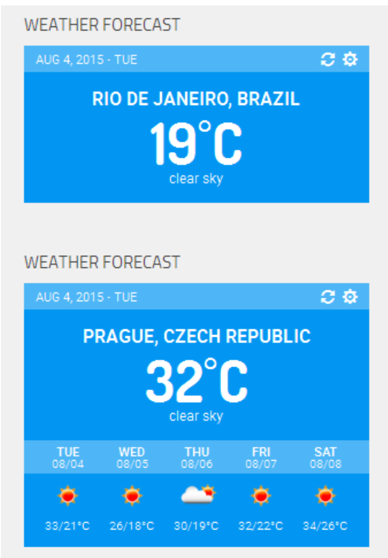


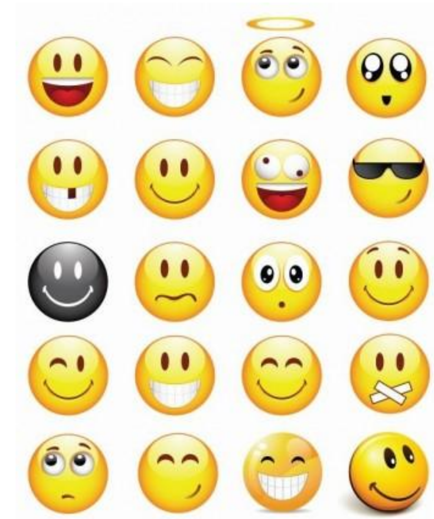
[:weather:]

:) 8-) ;-) …

Plugin

Plugin





Today's weather: [:weathe

Feature Interactions

Features designed in isolation
(divide and conquer)

Interact in intended and unintended ways
when composed

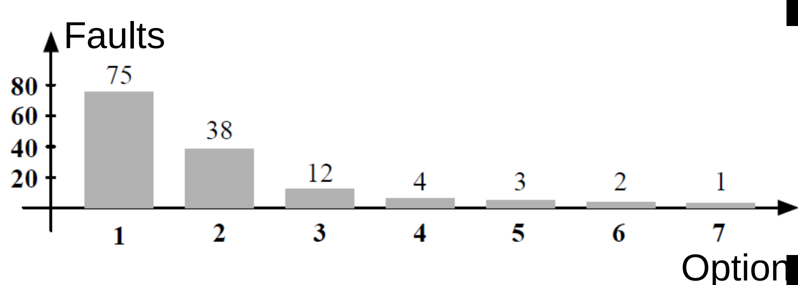
(Failure of compositionality
due to hidden underlying domain)

[Abal, ASE’14] [Medeiros’15]

Interaction Bugs in Practice

135 bugs across 24 open source systems

a	59
a & b	13
a & [illegible]	5
a & [illegible] & d & e	[illegible]
!a	6
!a & !b	8
!a & !b & !c	1
!a & !b & !c & !d	1
!a & !b & !c & !d [illegible] !f & !g	1
a xor b	17
(a & b) xor c	6
(a & b & c) xor d	3
a & b & !c	2
a & b & !c & !d [illegible] !f	1



What's the Specification?

Typically **global** property x for every program

Syntactically correct, well-typed

Absence of double-free vulnerabilities

Returns positive number for parameter 3

Terminates within 10 seconds

Challenge is checking all configurations

e.g., $\forall p \in PL: p \vDash x$

Brute-Force

Product Configuration

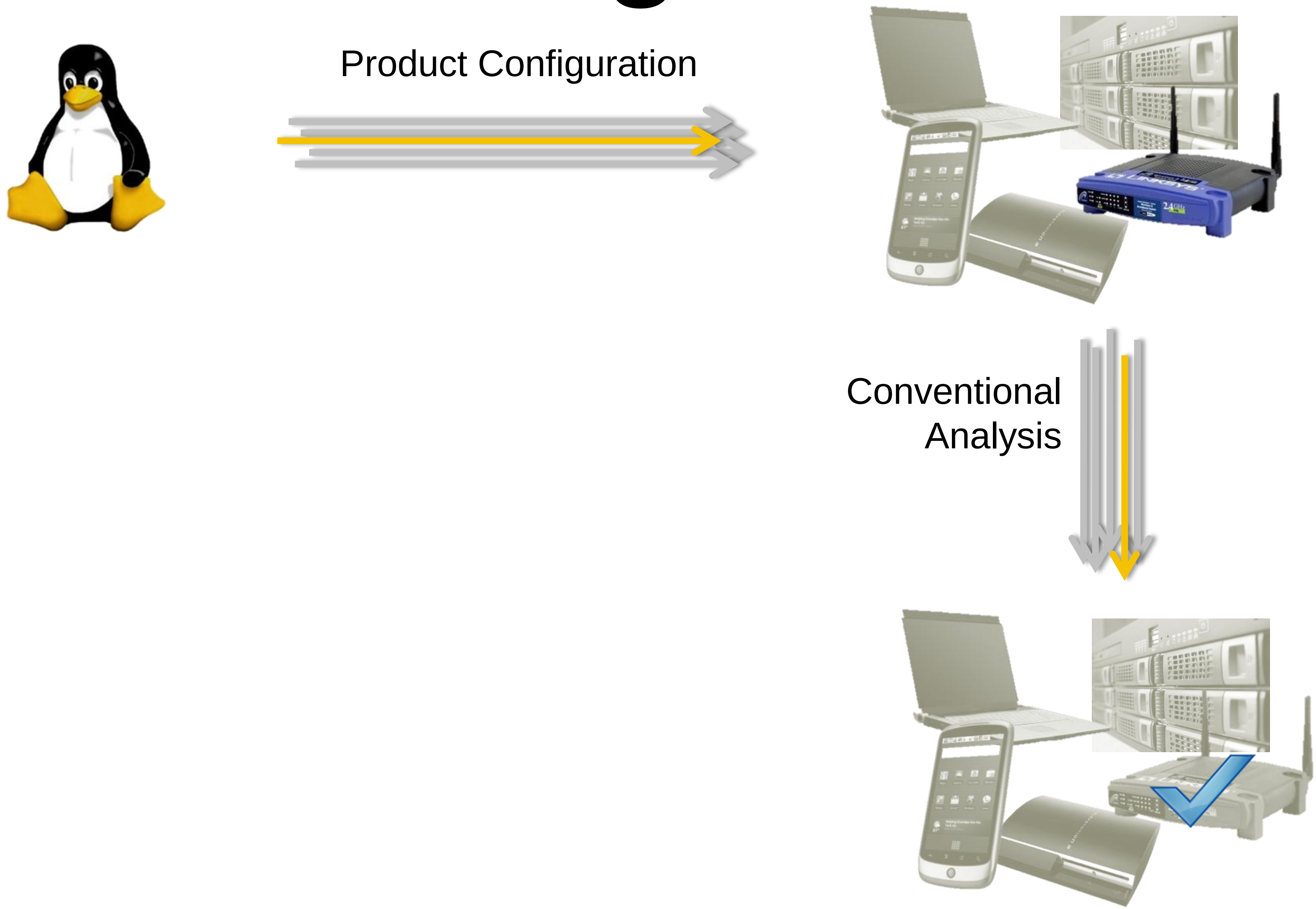
Conventional Analysis

320 optional, independent features

more combinations than estimated

atoms in the universe

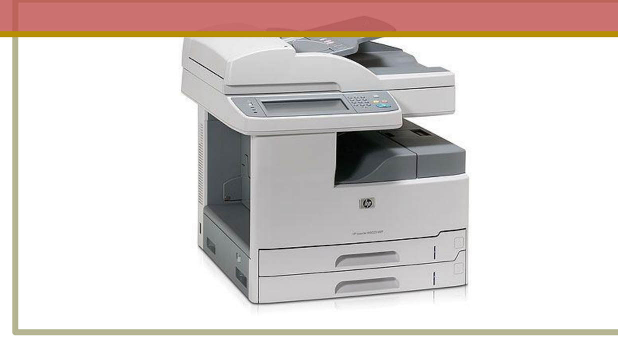
One Configuration



Checking **Products**

2000 Features
100 Printers
30 New Printers per Year

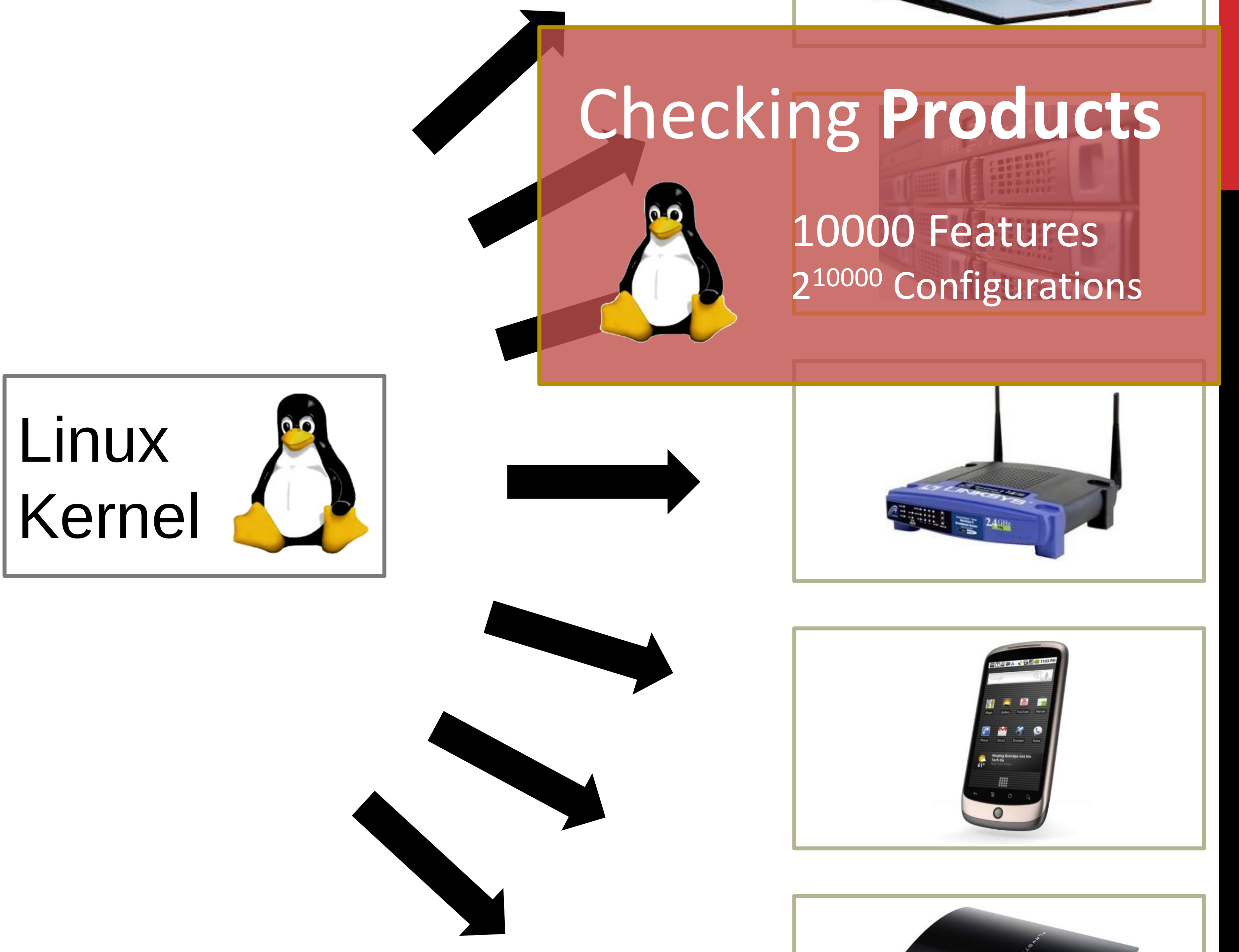
Printer
Firmware



Checking **Products**

10000 Features

2^{10000} Configurations



AUTOMATION?

Frameworks vs Components

Implications for QA?

Sampling

Product Configuration

Conventional
Analysis

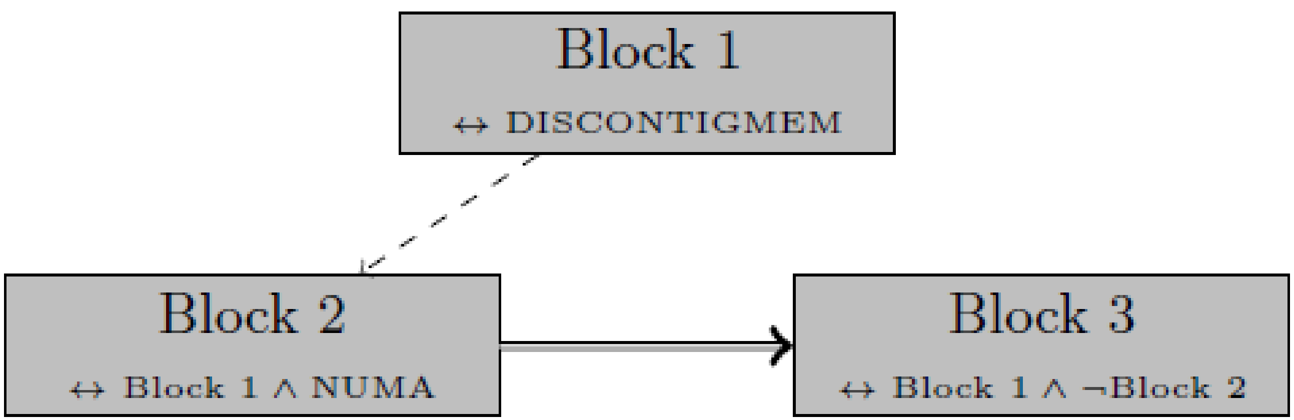
Check random configurations
Check representative configurations
Check maximum configuration
Check config. used by customers
Combinatorial interaction testing
(pairwise, etc)
Code coverage heuristics

CODE COVERAGE

```
#if defined(CONFIG_ARCH_FPGA11107)
/* fpga cpu/bus are currently 30 times slower so
     scale frequency as well to slow down Linux's
     sense of time */
[...]
#define TIMER3_FREQUENCY_KHZ (tmrHw_HIGH_FREQUENCY_HZ
     /1000 * 30)
#else
[...]
#define TIMER3_FREQUENCY_KHZ (tmrHw_HIGH_FREQUENCY_HZ
     /1000)
#endif


static struct clk sp804_timer3_clk = {
        .name = "sp804-timer-3",
        .type = CLK_TYPE_PRIMARY,
        .mode = CLK_MODE_XTAL,
        .rate_hz = TIMER3_FREQUENCY_KHZ * 1000,
};
```

```
#ifdef CONFIG_DISCONTIGMEM                          Block 1
static inline int pfn_to_nid(unsigned long
    pfn)
{
#ifdef CONFIG_NUMA                                  Block 2
   return((int) physnode_map[(pfn) /
       PAGES_PER_ELEMENT]);
#else                                               Block 3
   return 0;
#endif
}
#endif                                              Block 1
```

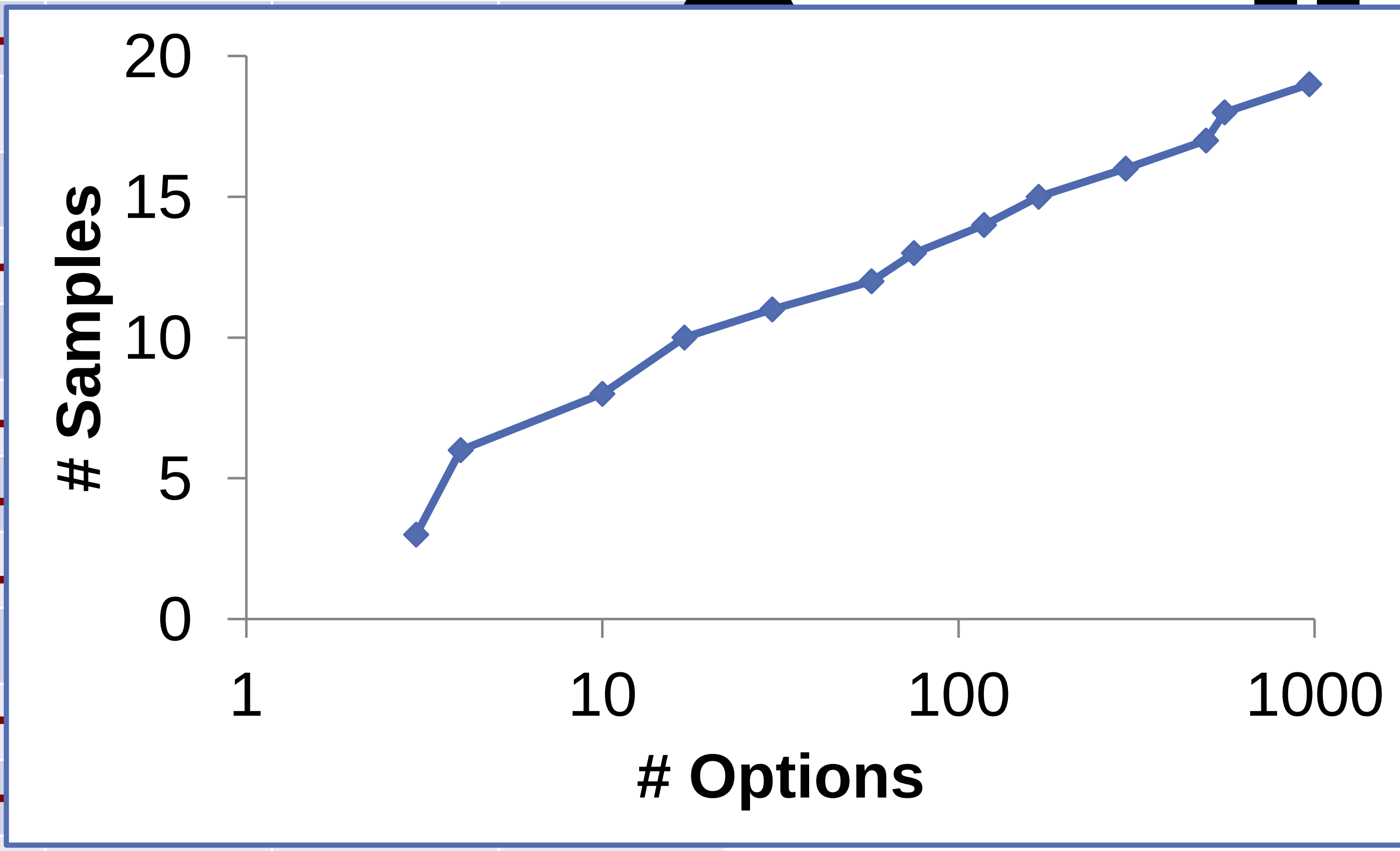


Pairwise Sampling

F1	F2	F3	F4
on	on	on	on
on	on	on	off
on	on	off	on
on	on	off	off
on	off	on	on
on	off	on	off
on	off	off	on
on	off	off	off
off	on	on	on
off	on	on	off
off	on	off	on
off	on	off	off
off	off	on	on
off	off	on	off
off	off	off	on
off	off	off	off

Pairwise

F1	F2	F3	F4
~~on~~	~~on~~	~~on~~	~~on~~
on	on	on	off
~~on~~			
on			
on			
~~on~~			
on			
~~on~~			
~~off~~			
~~off~~			
off			
~~off~~			
~~off~~			
off	off	on	off
~~off~~	~~off~~	~~off~~	~~on~~
~~off~~	~~off~~	~~off~~	~~off~~



http://math.nist.gov/coveringarrays/

Firefox | style toward clarity and grace - Google S... | meinicke/VarexJ · GitHub | tartler_12_osr.pdf (application/pdf Object)

math.nist.gov/coveringarrays/ipof/tables/table.2.2.html

Table for CA(2,*k*,2)

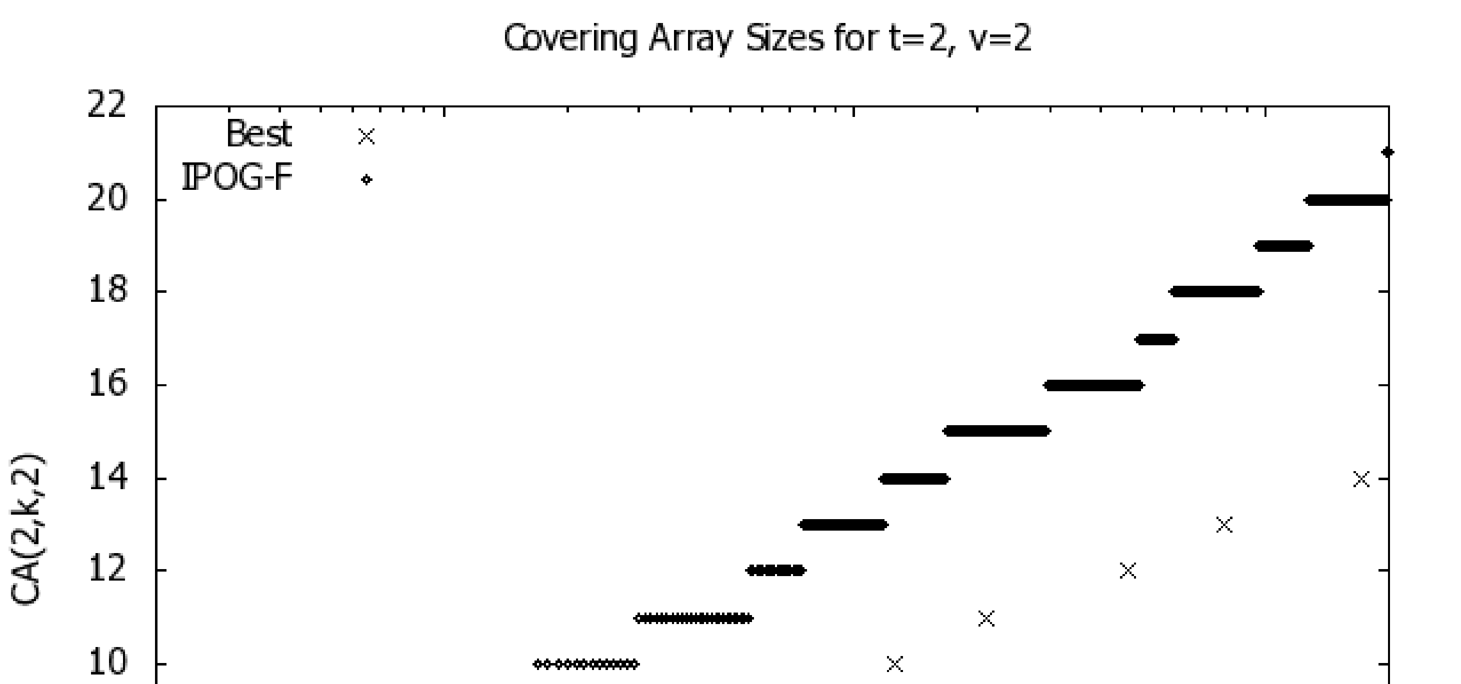
Covering array sizes are given for each *k* listed, where *t*=2 and *v*=2. The arrays, as generated by the IPOG-F program, are given as links t

Back to menu

Change *t*: Increase

Change *v*: Increase

k	CA(2,*k*,2)	Filesize
3	4	1.0K
4	6	1.0K
5	6	1.0K
6	6	1.0K
7	6	1.0K
8	6	1.0K
9	6	1.0K
10	8	1.0K
11	8	1.0K



Sampling

Product Configuration

Conventional
Analysis

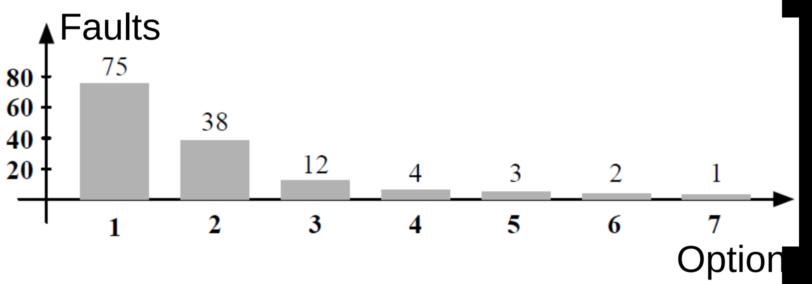
Check random configurations
Check representative configurations
Check maximum configuration
Check config. used by customers
Combinatorial interaction testing
(pairwise, etc)
Code coverage heuristics

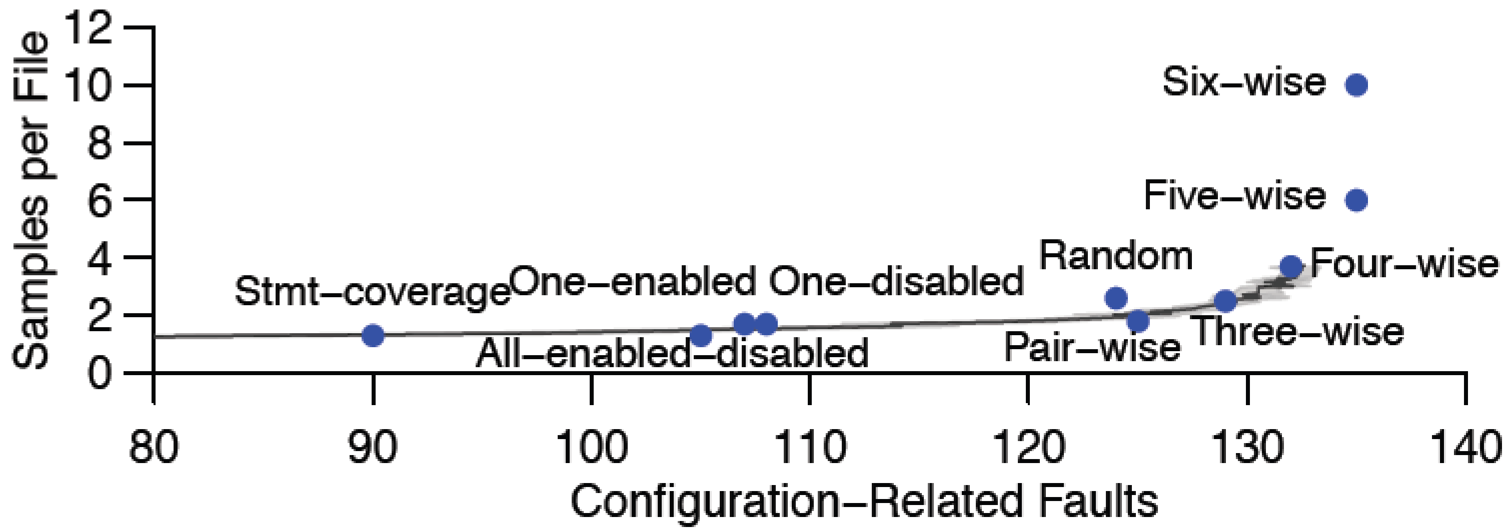
a	59
a & b	13
a & b & c	5
a & b & c & d & e	1
!a	16
!a & !b	8
!a & !b & !c	1
!a & !b & !c & !d	1
!a & !b & !c & !d & !e & !f & !g	1
a xor b	17
(a & b) xor c	6
(a & b & c) xor d	3
a & b & !c	2
a & b & !c & !d & !e & !f	1

[Abal, ASE'14] [Medeiros'15]

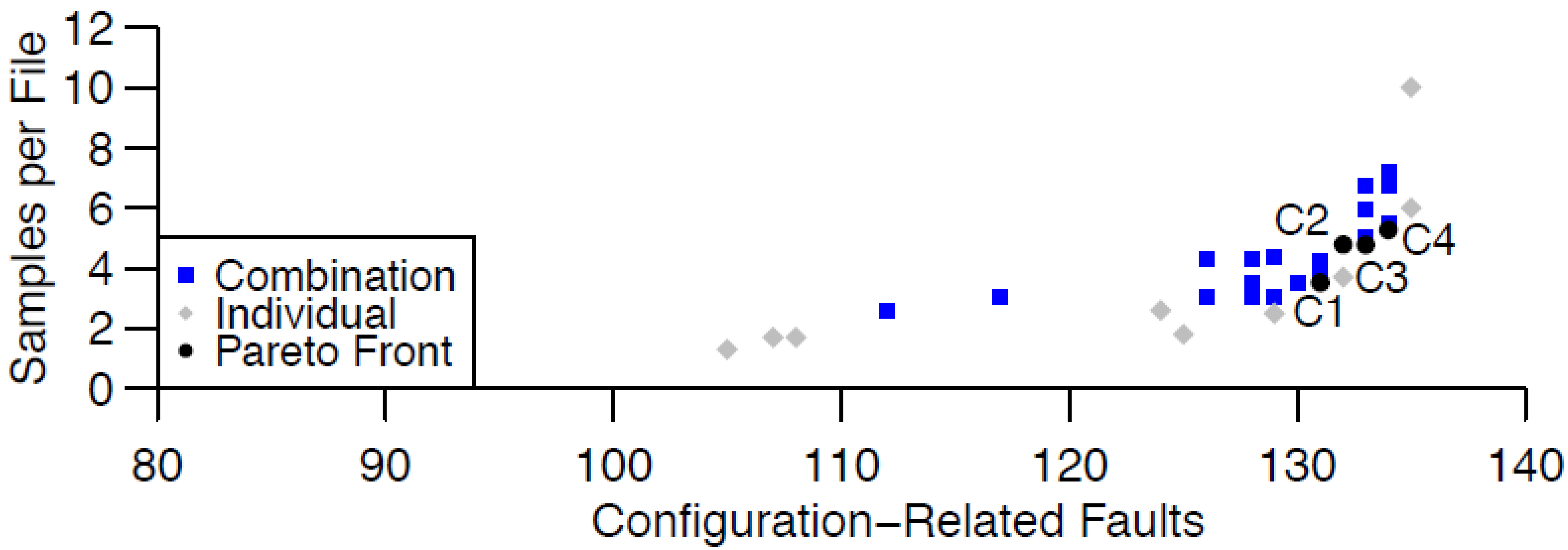
Interaction Bugs in Practice

135 bugs across 24 open source systems





Sampling Algorithm	Faults	Samples
Statement-coverage	90	1.3
All-enabled-disabled	105	1.3
One-enabled	107	1.7
One-disabled	108	1.7
Random	124	2.6
Pair-wise	125	1.8
Three-wise	129	2.5
Four-wise	132	3.7
Five-wise	135	6.0
Six-wise	135	10.0



Sampling Algorithm
C1 Pair-wise and one-disabled
C2 One-enabled, one-disabled and statement-coverage
C3 One-enabled, one-disabled and all-enabled-disabled
C4 One-enabled, one-disabled and pair-wise

	Faults	**Samples**		**Faults**	**Samples**
C1	131	3.5	C2	132	4.8
C3	133	4.8	C4	134	5.3

CHALLENGES FOR SAMPLING

Constraints

Build system information

Global vs local analysis

Header files

Algorithms	Constraints			Global Analysis			Header Files			Build System		
	Faults	Configs	Rank	Faults	Configs	Rank	Faults	Configs	Rank	Faults	Configs	Rank
Pair-wise	33 ↓	30 ⇑	5	–	–	–	39 =	936 ⇑	4	33 ↓	2.8 ↑	4
Three-wise	–	–	–	–	–	–	43 =	1,218 ⇑	5	42 ↓	3.9 ↑	5
Four-wise	–	–	–	–	–	–	45 =	1,639 ⇑	7	45 =	5.7 ↑	8
Five-wise	–	–	–	–	–	–	–	–	–	47 =	8.3 ↑	9
Six-wise	–	–	–	–	–	–	–	–	–	47 =	12 ↑	10
Most-enabled-disabled	23 ↓	1.4 =	1	27 =	1.4 =	1	27 =	1.4 =	1	26 ↓	1.4 ↑	2
One-enabled	30 ↑	1.1 ↓	3	31 ↑	7,943 ⇑	3	31 ↑	890 ⇑	6	20 ↓	2.3 ↑	7
One-disabled	38 ↓	1.1 ↓	4	39 =	7,943 ⇑	2	39 =	890 ⇑	3	39 =	2.3 ↑	3
Random	39 ↓	4.1 =	6	29 ⇓	8,123 ⇑	4	40 ↓	17.2 ⇑	2	41 =	4.2 ↑	6
Stmt-coverage	32 ↑	4.1 ↑	2	–	–	–	–	–	–	25 =	1.3 ↑	1

Some algorithms do not scale, indicated using dashes (–). We use ↑ and ↓ to represent small changes in the number of faults and size of sample set, as compared to our first study. Furthermore, we use ⇑ and ⇓ to represent larger changes.

Summary

Using a global analysis, we can potentially detect non-modular faults that span multiple files; it causes an explosion in the number of considered configuration options that leads to large sample sets; too large for t-wise and statement-coverage.

Summary

When incorporating header files, there is a potential to detect additional faults from header files; but a difficult setup; and much larger sample sets (if feasible at all), leading to ranking changes.

Summary

When considering constraints, we face an essential reduction of false positives; but high costs for generating sample sets, which are often not optimal or unique; it is infeasible for three-wise and higher.

Summary

When including build-system information, we face a difficult analysis, a few more configuration options to consider, but no significant changes.