Making Linux Fly
Towards a Certified Linux Kernel

Wentao Zhang

Tingxu Ren, Jinghao Jia, Darko Marinov, Tianyin Xu

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

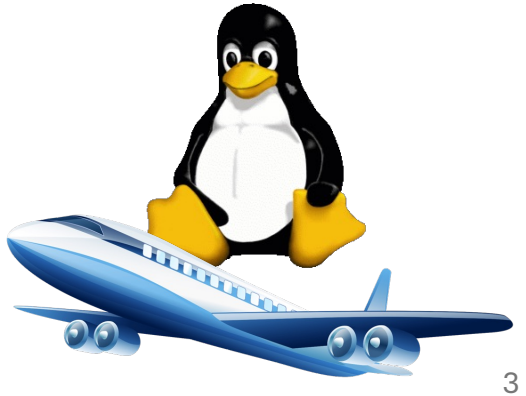
Overview

- Background
  - Software certification and DO-178C guidance
  - Modified Condition/Decision Coverage (MC/DC)
- First measurements of Linux kernel's MC/DC
- How it works: `llvm-cov` instrumentation
- Infrastructure for measuring Linux kernel's MC/DC
  - Toolchain
  - RFC patch: `kernel/llvm-cov/`
  - Kernel tests
- Demonstration
- Future work

Certifying Linux to DO-178C

- Goal: achieving DO-178C objectives 5 through 9 in Table A-7 for the Linux kernel
  - Objective 5: Test coverage of structure **(modified condition/decision)** is achieved
  - Objective 6: Test coverage of structure (decision coverage) is achieved
  - Objective 7: Test coverage of structure (statement coverage) is achieved
  - Objective 9: Verification of additional code, that cannot be traced to source code, is achieved
- This talk's focus: **enabling measurement** of **MC/DC** for Linux
  - Provide the tools to report MC/DC and ensure the tools' reliability
  - Complementary work: aiming at achieving high coverage

Kernel Testing & Dependability MC talk
Measuring and Understanding Linux Kernel Tests
Friday 12:00 PM, "Hall N2" (Austria Center)



Modified Condition/Decision Coverage

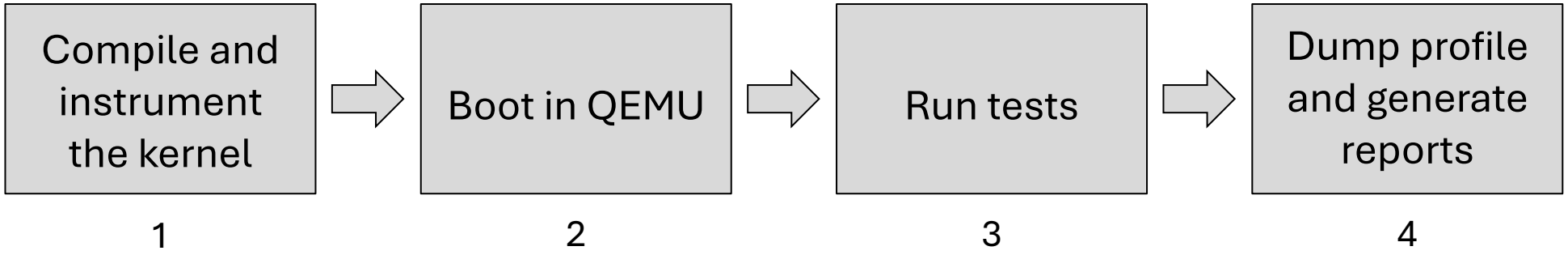
```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

- Cover all conditions in an expression
- At least two test vectors (True/False) to cover one condition
- Negating the condition also negates the decision outcome independently

Test vector	(x,y)	C_1 (x > 0)	C_2 (y > 0)	Decision outcome	Cover C_1	Cover C_2
0	(-3,-2)	False	-	False		
1	(4,-3)	True	False	False		*
2	(-1,2)	False	-	False	*	
3	(1,3)	True	True	True	*	*

Overview of Contributions

- We have built the *first* infrastructure for measuring Linux kernel's MC/DC
  - Integrating **open-source** solutions
- Key components
  - **Tool: Clang/LLVM** version >= 18
  - **Target:** Linux kernel mainline
  - **Tests: KUnit**, kselftest, LTP
- Steps:



Overall Coverage Report Including MC/DC

Coverage Report

Created: 2024-08-27 11:32

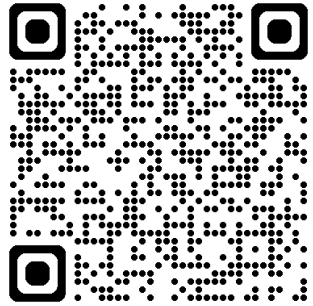
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	36.65% (2174/5931)	26.48% (18512/69901)	16.82% (7100/42208)	4.55% (143/3143)
block/	20.38% (292/1433)	14.51% (2842/19588)	8.76% (951/10858)	1.82% (17/934)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.92% (2170/12107)	13.25% (572/4316)	2.75% (11/400)
drivers/	13.87% (4531/32671)	11.09% (63031/568127)	7.53% (21449/284948	2.21% (592/26749)
fs/	21.02% (1906/9068)	13.57% (22231/163875)	9.02% (6798/75372)	2.25% (159/7063)
include/	26.12% (3565/13648)	20.28% (14397/70980)	16.36% (2626/16052)	6.21% (105/1692)
init/	58.62% (68/116)	46.18% (767/1661)	31.66% (202/638)	7.78% (7/90)
io_uring/	0.40% (3/757)	0.82% (99/12016)	0.11% (7/6326)	0.00% (0/655)
ipc/	9.39% (29/309)	5.33% (284/5326)	2.78% (57/2048)	0.00% (0/149)
kernel/	35.66% (3458/9696)	26.60% (34934/131331)	17.36% (11865/68346)	6.64% (422/6358)
lib/	48.36% (1299/2686)	39.28% (17893/45556)	26.87% (8735/32514)	15.41% (228/1480)
mm/	40.39% (1338/3313)	28.95% (15991/55237)	19.42% (5681/29254)	5.94% (167/2812)
net/	10.31% (1381/13401)	6.42% (17886/278474)	3.50% (5490/157004)	0.71% (113/15869)
security/	26.80% (369/1377)	14.55% (3419/23506)	10.89% (1308/12006)	1.45% (12/830)
sound/	6.48% (109/1681)	4.75% (1279/26942)	3.17% (416/13136)	0.63% (8/1264)
Totals	21.37% (20725/96983)	14.53% (215756/1484725)	9.70% (73258/755056	2.86% (1984/69490)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page for the whole kernel source tree

Simplest Example: 2-Condition Decision



```
218   11  if (!a)    /* num < 2 || size == 0 */
219    3    return;
220
221        /* called from 'sort' without swap function, let's pick the default */
222    8  if (swap_func == SWAP_WRAPPER && !((struct wrapper *)priv)->swap)
          MC/DC Decision Region (222:6) to (222:66)

          Number of Conditions: 2
             Condition C1 --> (222:6)
             Condition C2 --> (222:35)

          Executed MC/DC Test Vectors:

             C1, C2    Result
          1 { T,  F  = F      }
          2 { T,  T  = T      }

          C1-Pair: not covered
          C2-Pair: covered: (1,2)
          MC/DC Coverage for Expression  50.00%
223    6    swap_func = NULL;
224
225    8  if (!swap_func) {
226    6    if (is_aligned(base, size, 8))
```

lib/sort.c:sort_r

Example of a 6-Condition Decision

```
156  8.40k  for (; f < end; f++)
157  8.37k    if ((f->class == (u32) (dev->class >> f->class_shift) ||
158  8.37k         f->class == (u32) PCI_ANY_ID) &&
159  8.37k        (f->vendor == dev->vendor ||
160  6.70k         f->vendor == (u16) PCI_ANY_ID) &&
161  8.37k        (f->device == dev->device ||
162  2.09k         f->device == (u16) PCI_ANY_ID)) {

MC/DC Decision Region (157:7) to (162:38)

Number of Conditions: 6
   Condition C1 --> (157:8)
   Condition C2 --> (158:8)
   Condition C3 --> (159:8)
   Condition C4 --> (160:8)
   Condition C5 --> (161:8)
   Condition C6 --> (162:8)

Executed MC/DC Test Vectors:

   C1, C2, C3, C4, C5, C6    Result
1 { F,  F,  -,  -,  -,  -  = F      }
2 { F,  T,  F,  F,  -,  -  = F      }
3 { F,  T,  T,  -,  F,  F  = F      }
4 { T,  -,  F,  F,  -,  -  = F      }
5 { F,  T,  T,  -,  F,  T  = T      }
6 { F,  T,  T,  -,  T,  -  = T      }
7 { T,  -,  F,  T,  F,  T  = T      }
8 { T,  -,  T,  -,  F,  T  = T      }

C1-Pair: covered: (1,7)
C2-Pair: covered: (1,5)
C3-Pair: covered: (2,5)
C4-Pair: covered: (4,7)
C5-Pair: covered: (3,6)
C6-Pair: covered: (3,5)
MC/DC coverage for expression: 100.00%

163  15       void (*hook)(struct pci_dev *dev);
```

drivers/pci/quirks.c:
pci_do_fixups

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

⇨
File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the curly brace is entered?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
⇨ File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the decision is evaluated?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
⇨ File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the 1st condition is evaluated?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
⇨ File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the 2nd condition is evaluated?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
⇨ Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

The same location also has a branch region

Branch region: How many times the 1st condition is evaluated to “true” and “false” respectively?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#     #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

The same location also has a branch region

Branch region: How many times the 2nd condition is evaluated to “true” and “false” respectively?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
⇨ File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the 1st return statement is executed?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
⇨ File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the 2nd return statement is executed?

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

```
int foo(int x, int y) {
    if ((x > 0) && (y > 0))
        return 1;
    return 0;
}
```

```
$ clang -Xclang -dump-coverage-mapping

File 0, 1:23 -> 5:2 = #0
File 0, 2:9 -> 2:27 = #0
File 0, 2:9 -> 2:16 = #0
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
File 0, 2:20 -> 2:27 = #2
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
Gap,File 0, 2:28 -> 3:9 = #1
File 0, 3:9 -> 3:17 = #1
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Four independent counters are maintained. Others can be derived from these four.

Illustration of `llvm-cov` Instrumentation

- Instrumentation adds counters

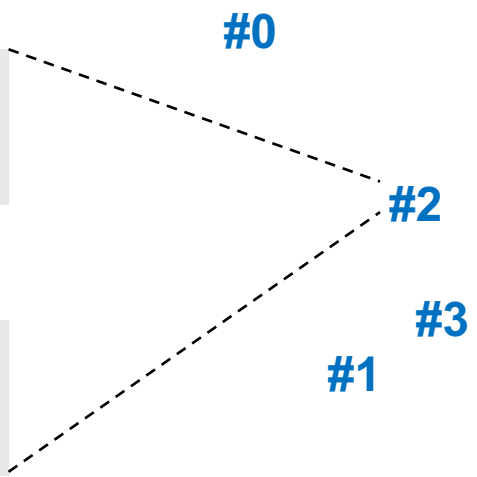
```
$ clang -Xclang -dump-coverage-mapping
```

LLVM IR

```
%pgocount = load i64, ptr @__profc_foo(int, int), align 8
%0 = add i64 %pgocount, 1
store i64 %0, ptr @__profc_foo(int, int), align 8
```

x86-64

```
mov     rax, qword ptr [rip + .L__profc_foo(int, int)]
add     rax, 1
mov     qword ptr [rip + .L__profc_foo(int, int)], rax
```



Counters under the hood: memory read, add by one and memory write

Infrastructure for Measuring Linux MC/DC

- **Tool:** Clang/LLVM version >= 18
  - We helped test the tool as early adopters and fixed/reported bugs
- **Target:** Linux kernel mainline
  - We built the necessary kernel support to export the coverage profile
  - Results shown are for `v6.11-rc5`
- **Tests:** KUnit, kselftest and LTP
  - Results shown are for KUnit

Toolchain

- MC/DC feature for Clang/LLVM was implemented and merged into mainline in January 2024
  - Built on top of Source-based Code Coverage [1]
  - Mostly contributed by Alan Phipps from Texas Instruments
  - Utilizing bitmaps to track test vectors
- Included in releases >= 18.1.0 since March 2024
- We are actively testing and verifying Clang/LLVM MC/DC
  - We are among the first to test this implementation (and our target is very unique!)
  - Collaborating with the upstream, we fixed/reported a few bugs (see next slides)

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html

Our Contributions to LLVM

ID	Title	Status
#80952	[llvm-cov][CoverageView] minor fix/improvement to HTML and text coverage output	Merged
#82464	[clang][CodeGen] Keep processing the rest of AST after encountering unsupported MC/DC expressions	Merged
#86998	[clang][CoverageMapping] "Assertion `AfterLoc.isValid()` failed" during compiling `switch` within statement expressions	Merged
#87000	[llvm-cov][MC/DC] "Branch not found in Decisions" when handling complicated macros	Merged
#92216	[llvm-cov][MC/DC] "Branch not found in Decisions" when handling variadic macros	Confirmed
#95831	[clang][CoverageMapping] Assertion fails when headers included in function bodies	Reported
#96016	[llvm-cov] let text mode divider honor `--show-branch-summary --show-region-summary` etc	Merged
#97385	[llvm-cov][MC/DC] "Out-of-bounds Bit access." when run with binary profile correlation	Confirmed
#101241	[CoverageMapping] fail to evaluate "constant folded" conditions at compile time	Confirmed
…	…	…

Our Contributions to LLVM

- Important bugs in LLVM MC/DC found on Linux

Reduced kernel code

```
struct Foo foo = {
    .field1 = ({
        switch (123) {
        case 123:
            break;
        }
        456;
    }),
};
```

- #86998 (and fix in #89564)
- Exposed by `fs/coredump.c`
- A non-standard C syntax

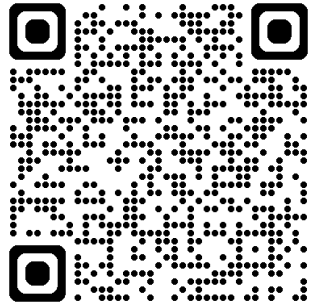
Reduced kernel code

```
#define FOO(x) foo_##x

int a, foo_b;

if (a && FOO(b)) { ... }
```

- #87000 (and fix in #89869)
- Exposed by `drivers/iommu/intel/perfmon.c`
- Complicated macros

- Even with these fixes, Clang/LLVM coverage does **not** work out-of-the-box to measure code coverage of the Linux kernel

Kernel Support for Linux MC/DC



- Challenge: export in-memory counters and bitmaps
  - Straightforward for user-space applications: just write to a file at the same directory as the executable
  - In a freestanding environment, like OS kernels: no concept of “current directory”
  - Solution: write to **a pseudo file system** instead
    - Also the practice of `kernel/gcov/`
- Implementation of `kernel/llvm-cov/`
  - Kbuild support
  - Debugfs interface and profile serialization
    - Reuse part of patch by Sami Tolvanen et al. “pgo: add clang's Profile Guided Optimization infrastructure patches” [1] with different goals: performance optimization vs. **precise coverage** for high assurance
  - RFC “Enable measuring the kernel's Source-based Code Coverage and MC/DC with Clang” [2]

[1] https://lore.kernel.org/lkml/20210407211704.367039-1-morbo@google.com/
[2] https://lore.kernel.org/lkml/20240824230641.385839-1-wentaoz5@illinois.edu/

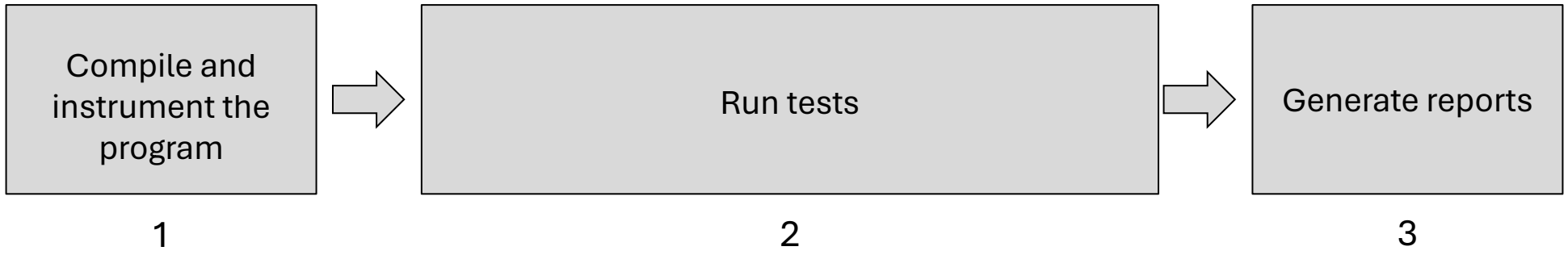
Exercise Various Kernel Testing Techniques

- Coverage report with different kernel test harnesses:

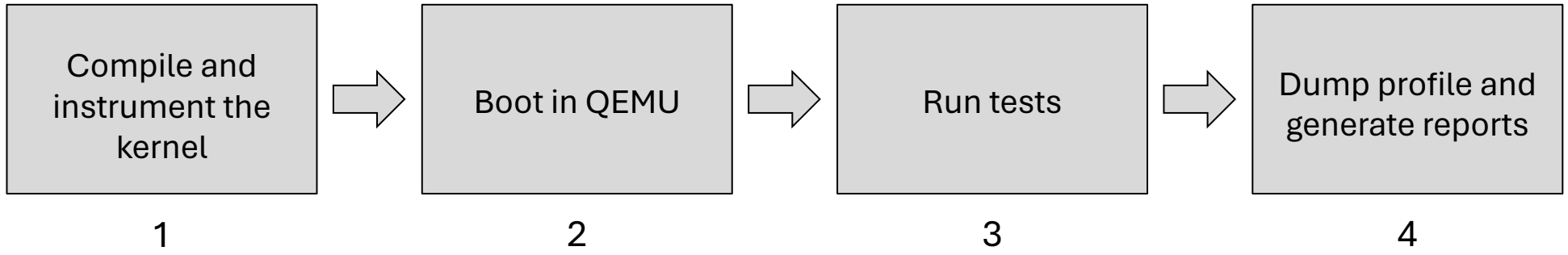
Kernel test harnesses	Function coverage	Line coverage	Branch coverage	MC/DC
Boot	28.05%	19.80%	13.61%	4.58%
Boot + KUnit	30.60% ↑2.55pp	22.06% ↑2.26pp	15.62% ↑2.01pp	5.23% ↑0.65pp
Boot + KUnit + Kselftest + LTP	39.29% ↑8.69pp	29.95% ↑7.89pp	22.29% ↑6.67pp	9.68% ↑4.45pp

Demo

- `llvm-cov` with user space programs



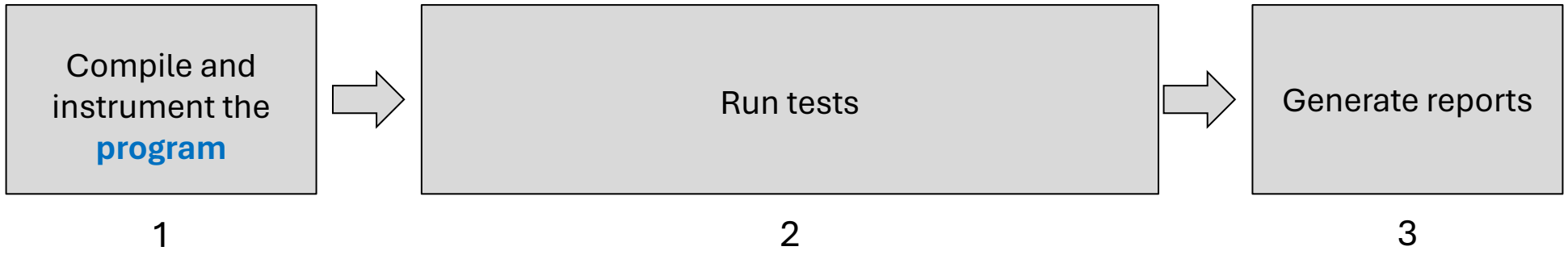
- `llvm-cov` with Linux kernel



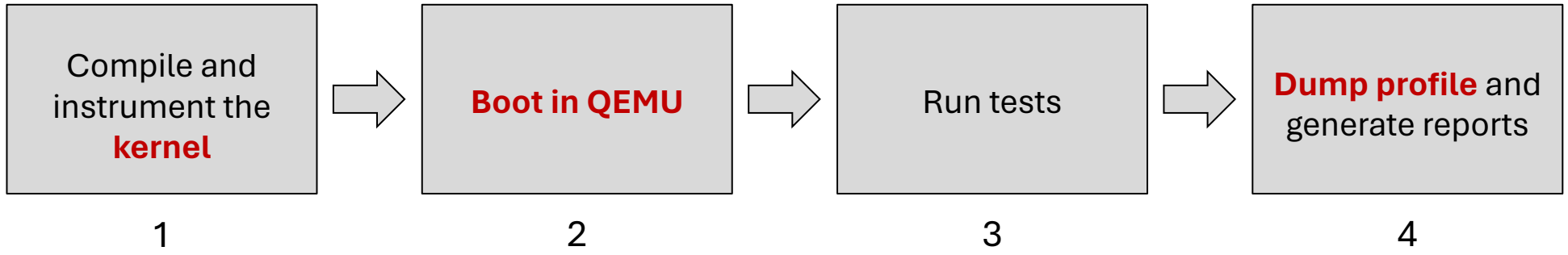
Demo

User space-specific
Kernel-specific

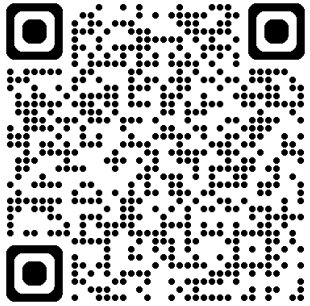
- `llvm-cov` with user space programs



- `llvm-cov` with Linux kernel



Summary and Future Work



Visit links in slides and give feedback

- We can measure MC/DC of Linux kernel
  - Tested with different kernel branches
  - RFC “Enable measuring the kernel's Source-based Code Coverage and MC/DC with Clang”

Next steps

- Test the tools more thoroughly and keep improving them
  - Check reliability and accuracy of the current implementation
  - Improve the presentation of data in the report
  - Compare with proprietary tools like VectorCAST
  - DO-330 Tool Qualification for `llvm-cov`
- Other objectives for certifying Linux
  - Data coupling and control coupling coverage (DO-178C objective 8)
  - Object coverage (DO-178C objective 9)

Acknowledgement

- **Open-source community**
  - LLVM developers: Alan Phipps, @chapuni, @ZequanWu, @ornata, @hanickadot, @MaskRay…
  - GCC developers: Jørgen Kvalsvik, Andrew Pinski, Alejandro Colomar…
  - Kernel developers: Sami Tolvanen, Bill Wendling, Dmitry Vyukov…
  - ELISA community
- Experiments are largely run on **CloudLab**
- UIUC work is supported with funding from **The Boeing Company**