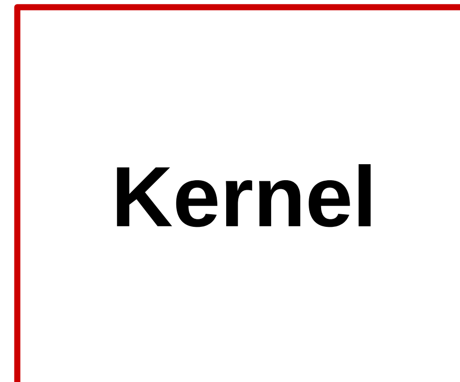
Verification of Linux Kernel without Loadable Kernel Modules

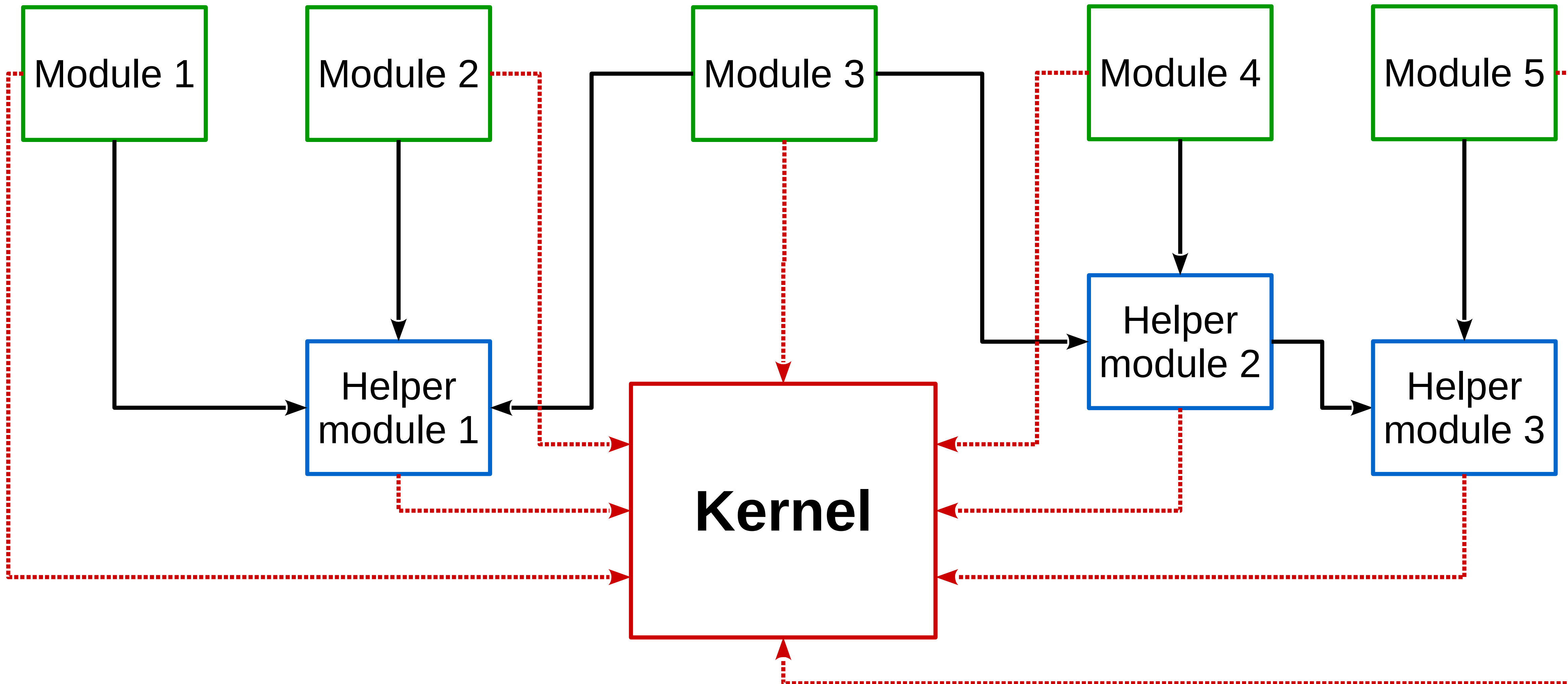
Evgeny Novikov and Ilja Zakharov
ISP RAS, Linux Verification Center

CPA&LDV'18, Moscow, September 26, 2018

Linux Kernel Architecture

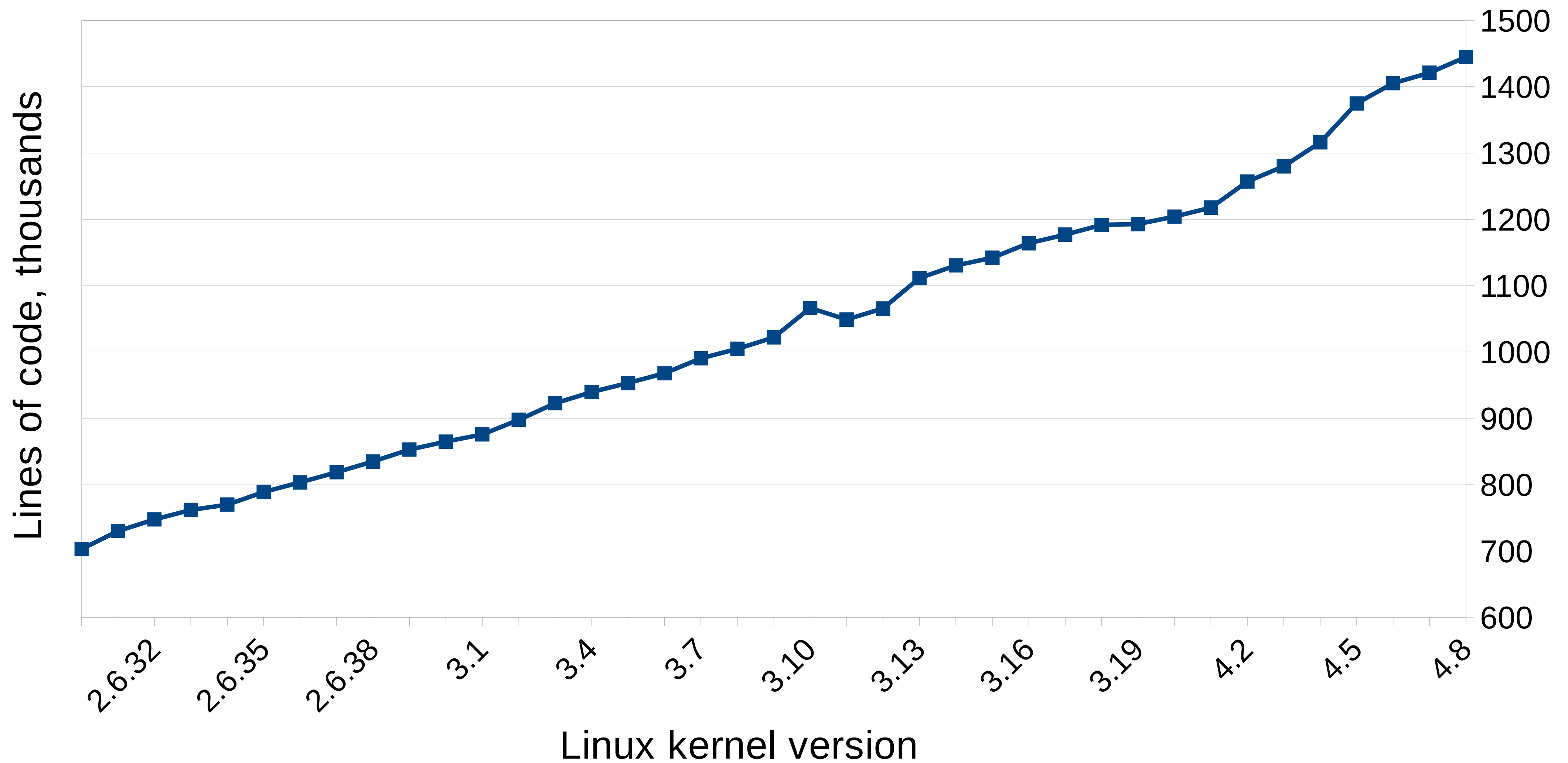


Linux Kernel Architecture



4373 modules
(Linux 3.14, x86_64, allmodconfig)

Size of Linux Kernel without Loadable Kernel Modules (Later – **Linux Kernel**)



Average size of modules is ~**2** KLOC

Challenge

Linux kernel operates on billions of devices
used by billions of people,

thus,

requirements for its functionality, security, reliability and performance are ones of the highest

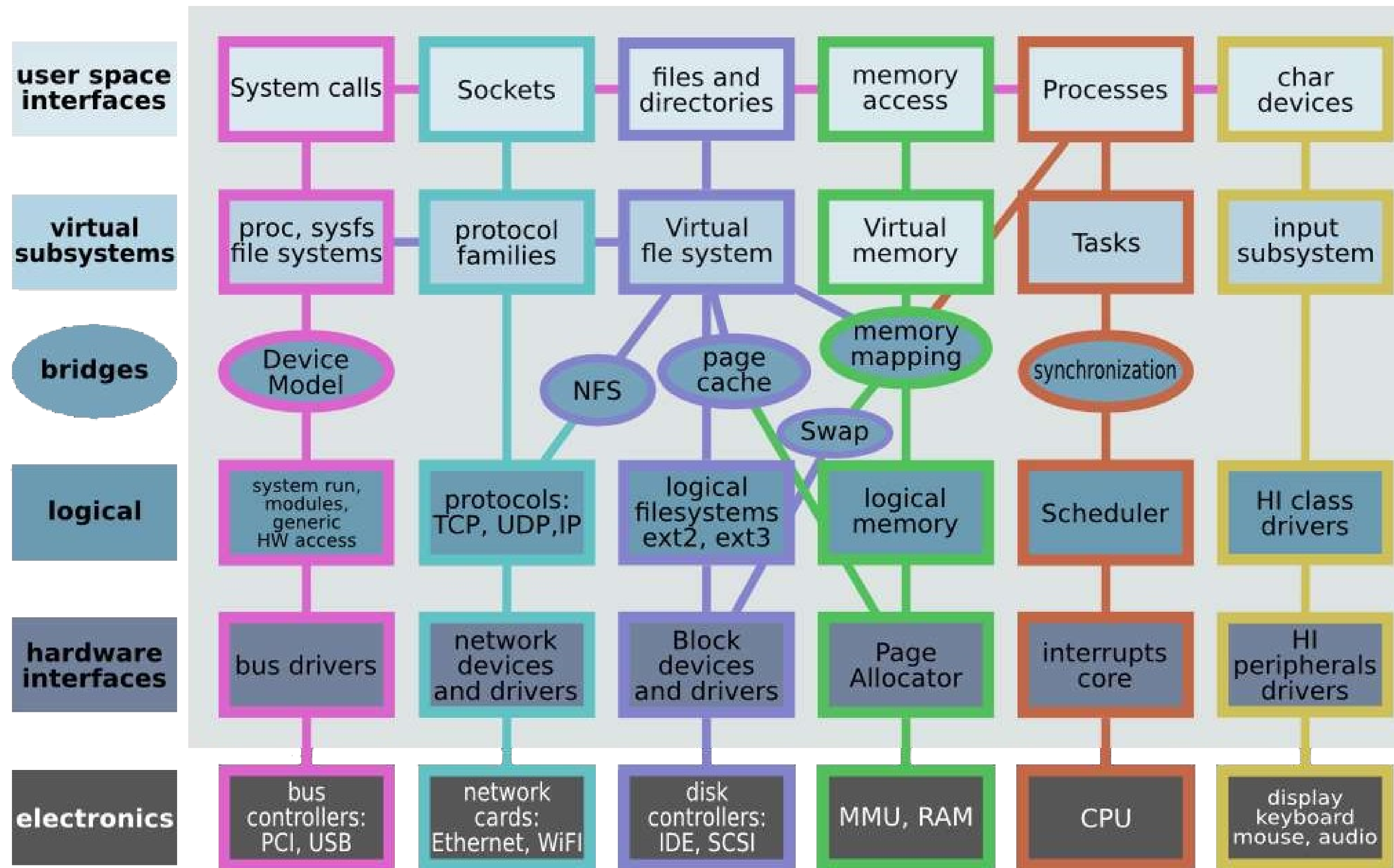
Related Work

- Using special programming languages, tools and hardware
  - Helps just in some cases
- Code review, testing, static analysis
  - Does not aim at detecting all violations of checked requirements
- Deductive verification
  - Needs too much human efforts (~1 man-year per 1 KLOC)
- **Software verification**
  - Seems to be the only appropriate approach for scalable heavy-weight formal verification of software

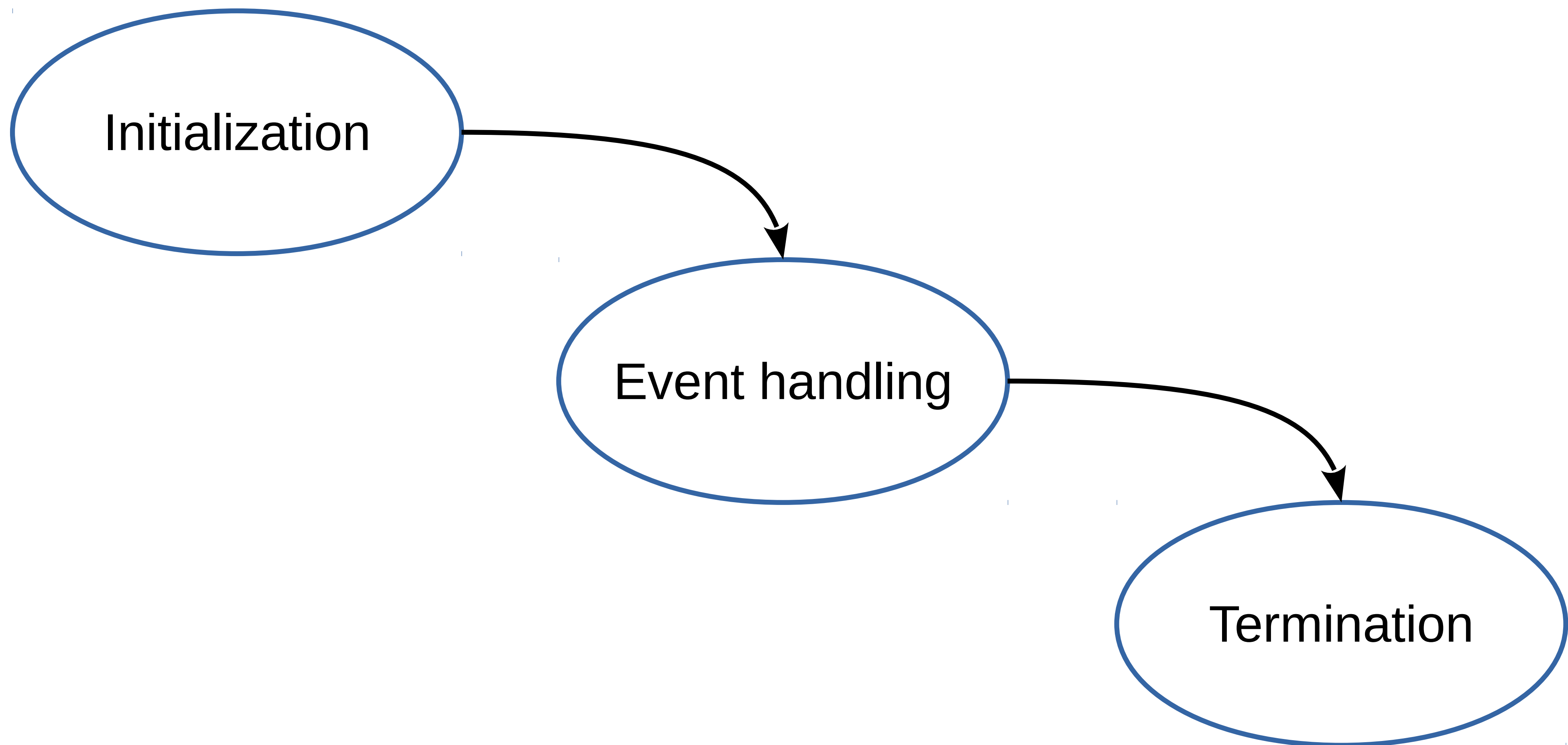
Outline

- Background
  - Linux kernel subsystems
  - Software verification tools
  - Klever software verification framework
- Verification of Linux kernel
  - Decomposing Linux kernel into subsystems
  - Verifying Linux kernel subsystems together with device drivers
  - Generating environment models for Linux kernel subsystems
  - Checking requirements for Linux kernel subsystems
  - Improving verification results
- Implementation and evaluation

Linux Kernel Subsystems



Linux Kernel Subsystems Operation



Initialization of Linux Kernel Subsystems

- Each subsystem defines one or more initialization functions
- Startup function *start_kernel()* initializes the most vital subsystems first of all
- Most of subsystems are initialized in accordance with their *levels* specified via macros taking initialization function names as arguments, e.g. Linux 3.14 has 19 such the levels
- Some subsystems initialize other ones

Event Handling in Linux Kernel Subsystems

- Subsystems define and register callbacks for handling events
- Subsystems define helper functions invoked during handling events by other subsystems and loadable kernel modules

Termination of Linux Kernel Subsystems

- Subsystems operate until normal or abnormal reboot
- There are no exit functions
- Subsystems do not perform final clean up

Software verification tools

- Capable to check industrial programs of thousands or dozens of thousands of lines of code in size
- Need rather accurate environment models
- Allow to check various requirements (usually non-functional ones)

Klever software verification framework

- Is designed for checking various GNU C programs
- Includes specifications allowing:
  - to generate rather accurate environment models for invoking most popular device driver APIs
  - to check various requirements in device drivers

Outline

- Background
  - Linux kernel subsystems
  - Software verification tools
  - Klever software verification framework
- Verification of Linux kernel
  - Decomposing Linux kernel into subsystems
  - Verifying Linux kernel subsystems together with device drivers
  - Generating environment models for Linux kernel subsystems
  - Checking requirements for Linux kernel subsystems
  - Improving verification results
- Implementation and evaluation

Decomposing Linux Kernel into Subsystems

- Treat all source files from specified directories built into Linux kernel as subsystems and add/remove individual source files by hand
  - Simple update of configuration for new versions of Linux kernel
  - Allow obtaining quite compact subsystems

Verifying Linux Kernel Subsystems together with Device Drivers

- Verify each subsystem with all device drivers that use its interfaces one by one
  - Too much time for verification but all possible interaction scenarios are covered
- Select those device drivers that increase function coverage in the best way
  - Compromise between verification time and quality

Generating Environment Models for Linux Kernel Subsystems

- Generator for initializing subsystems and device drivers
  - Needs specifications relating subsystems initialization levels and functions
- Generator for invoking callbacks (the same as for device drivers)
  - Reuses relevant environment model specifications for device drivers
  - Needs subsystem specific specifications
- Modeling remaining environment
  - Extending intermediate representation of environment model
  - Developing models for vital undefined functions

Checking Requirements for Linux Kernel Subsystems

- Check those requirements that are checked for device drivers and relevant for subsystems:
  - Rules of correct usage of the Linux kernel API
  - Memory safety
  - Concurrency safety
- Adjust requirement specifications
  - Do not check final state

Improving Verification Results

- Until obtaining reasonable coverage and acceptable number of false alarms one needs step by step:
  - to adjust tool configurations describing target subsystems and device drivers verified together with them
  - to refine environment model and requirement specifications

Outline

- Background
  - Linux kernel subsystems
  - Software verification tools
  - Klever software verification framework
- Verification of Linux kernel
  - Decomposing Linux kernel into subsystems
  - Verifying Linux kernel subsystems together with device drivers
  - Generating environment models for Linux kernel subsystems
  - Checking requirements for Linux kernel subsystems
  - Improving verification results
- Implementation and evaluation

Evaluation

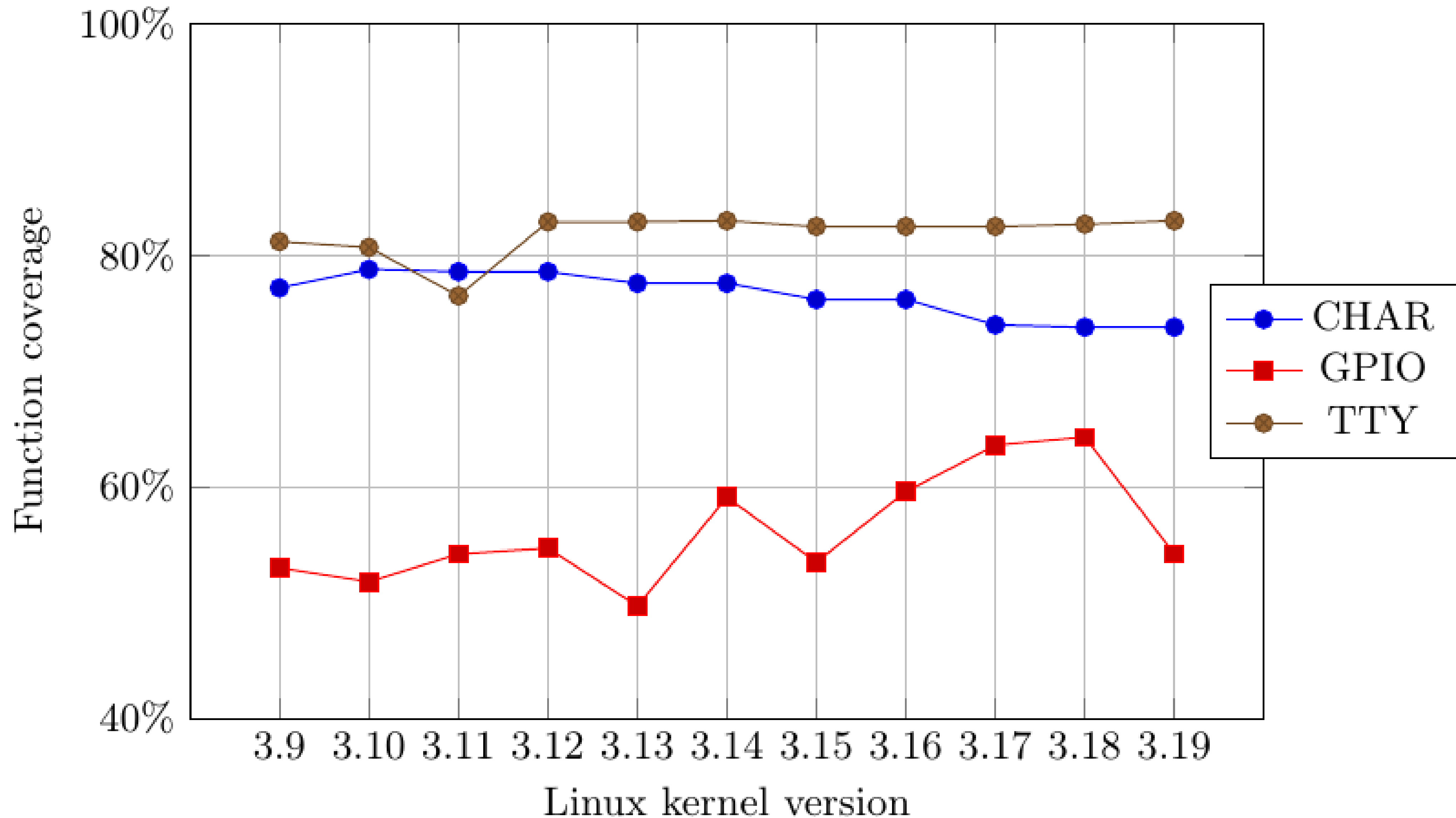
- Klever Git branch *kernel-verification*
  - default specifications and tool configurations
- CPAchecker Subversion revision *trunk:27583*
  - configuration *ldv-bam* for reachability
  - configuration *smg-ldv* for memory satety
  - 15 minutes of CPU time and 10 GB of memory per each verification task
- Linux kernel
  - architecture x86_64
  - configuration *allmodconfig*

Target Subsystems (Linux 3.9 – 3.19)

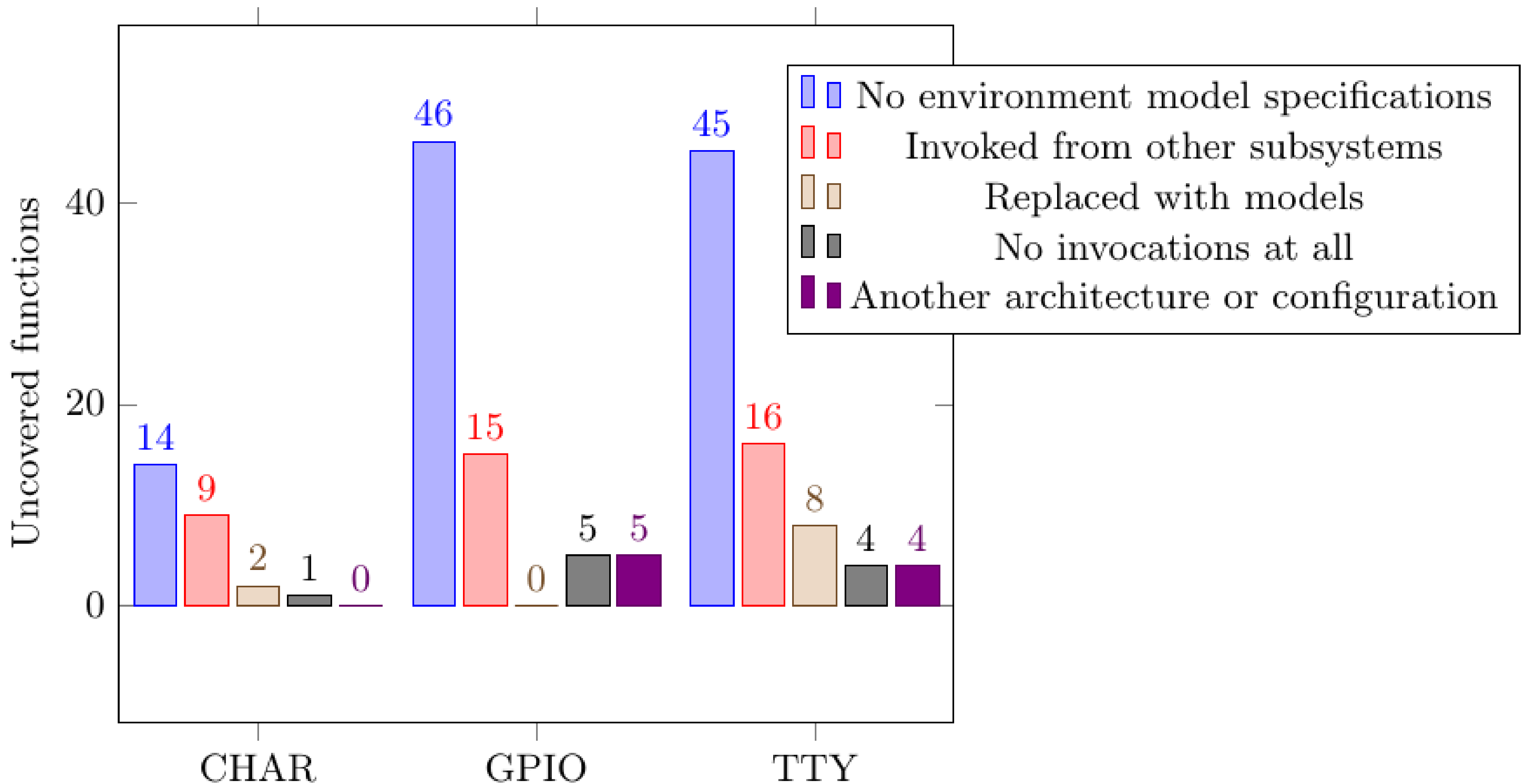
Subsystem name	Directory	Source files	Lines of code
Character Devices Support (*CHAR*)	drivers/char	5	4194
General-Purpose I/O (*GPIO*)	drivers/gpio	6	4472
Terminal Devices Support (*TTY*)	drivers/tty	11	12129

Subsystem name	Source files added/removed	Lines of code added/removed
CHAR	+0/-1 (+0%/-20%)	+950/-712 (+23%/-17%)
GPIO	+2/-3 (+33%/-50%)	+5074/-3079 (+113%/-69%)
TTY	+1/-0 (+9%/-0%)	+4012/-3221 (+33%/-27%)

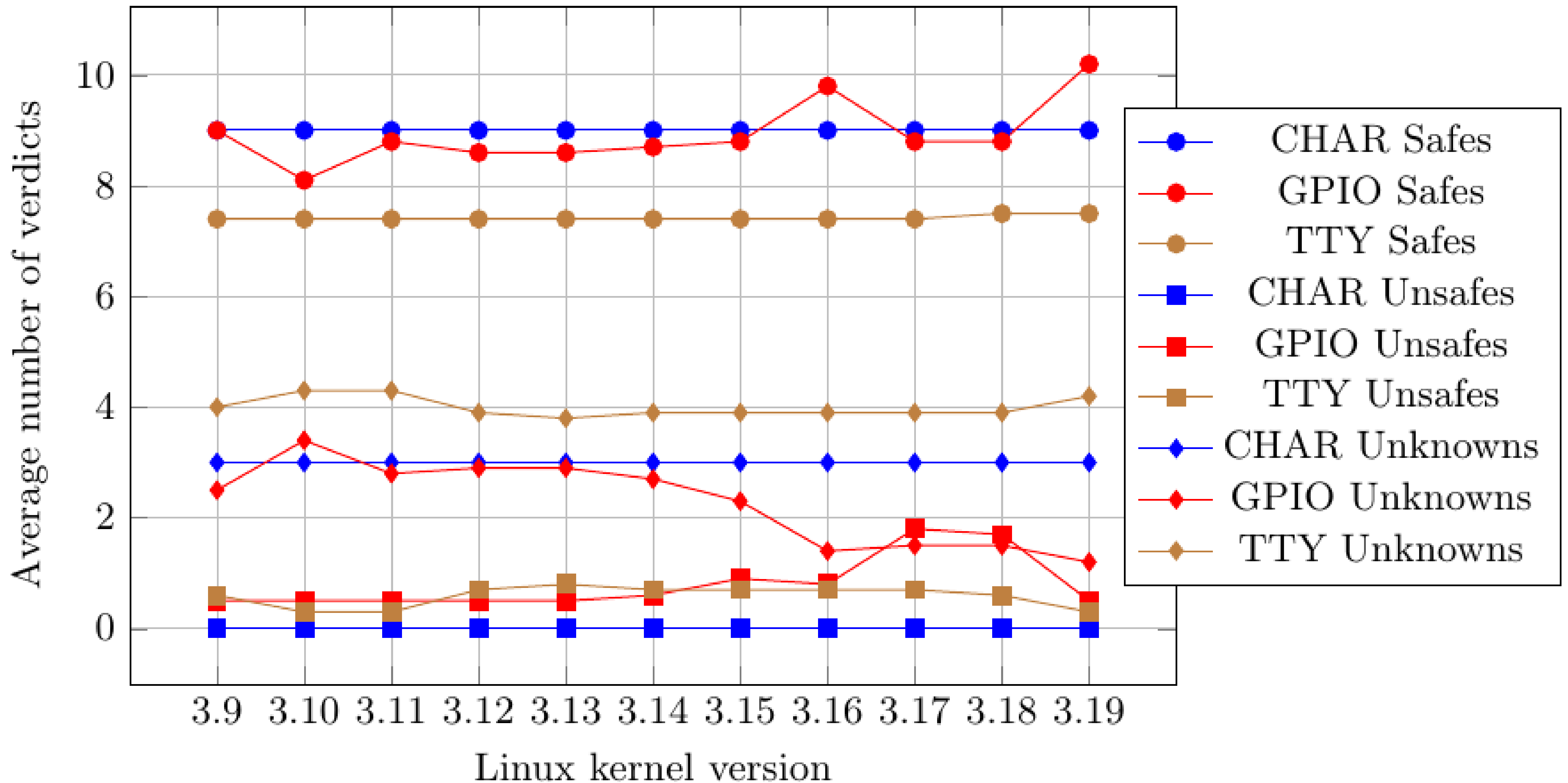
Function Coverage for Target Subsystems



Reasons of Absence of Function Coverage for Target Subsystems (Linux 3.14)



Average Number of Verdicts for Target Subsystems (12 Requirement Specifications)



Detecting Known Faults in Target Subsystems

Subsystem name	Commit hash	Requirements specification	Detection status
CHAR	08d2d00b291e b5325a02aa84 61c6375d5523	generic:memory generic:memory generic:memory	✗(another architecture) ✓(extra source files) ✗(another configuration)
GPIO	e9595f84a627 00acc3dc2480	generic:memory linux:kernel:locking:spinlock	✓(extra source files) ✓
TTY	b216df538481 07584d4a356e 1d9e689c934b	generic:memory linux:kernel:module generic:memory	✗(needs specification) ✓(dead code) ✗(too complex)

Conclusion

- We developed a new method that:
  - enables rather thorough checking and finding hard-to-detect faults for subsystems of various versions of the Linux kernel
  - does not require considerable efforts for configuring tools and developing specifications
- We could detect:
  - one fault in GPIO Linux kernel subsystem
  - 2 unreported faults in Linux kernel device drivers
  - 4 of 8 known faults after slight adjustment
- There is room for improvement primarily by means of developing specifications

Questions?

Novikov E. and Zakharov I. Verification of Operating System Monolithic Kernels without Extensions. Proceedings of the 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, 2018