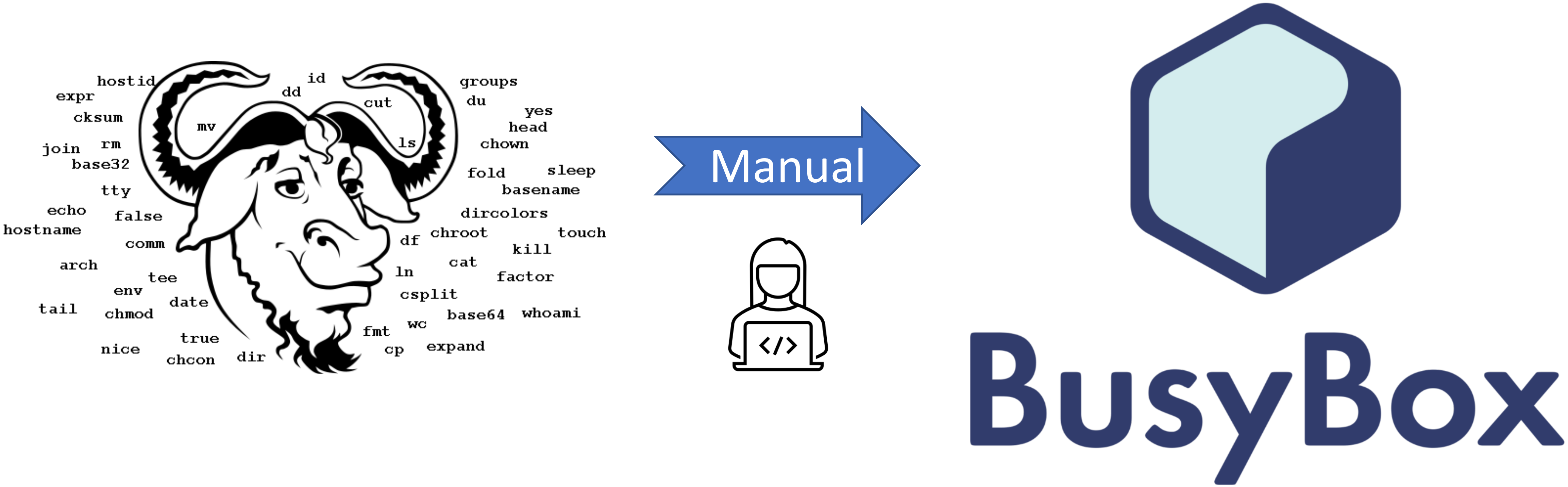
Lightweight, Multi-Stage, Compiler-Assisted Application Specialization (LMCAS)

Mohannad Alhanahnah and Somesh Jha

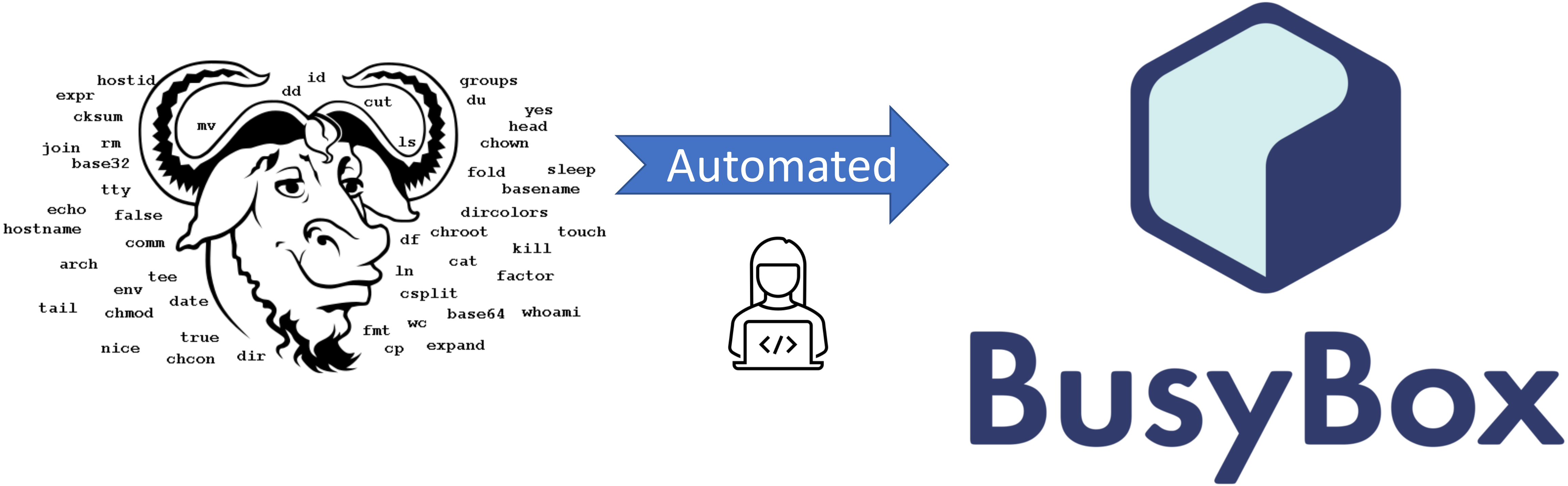
University of Wisconsin-Madison

Demands for Tiny & Specialized Utilities

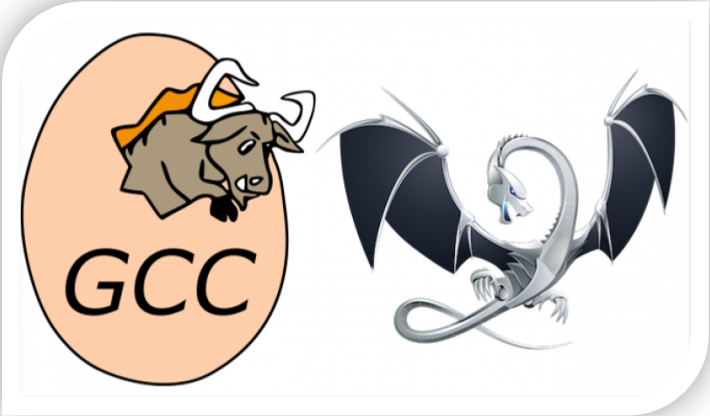
Tiny Utilities Generated Manually

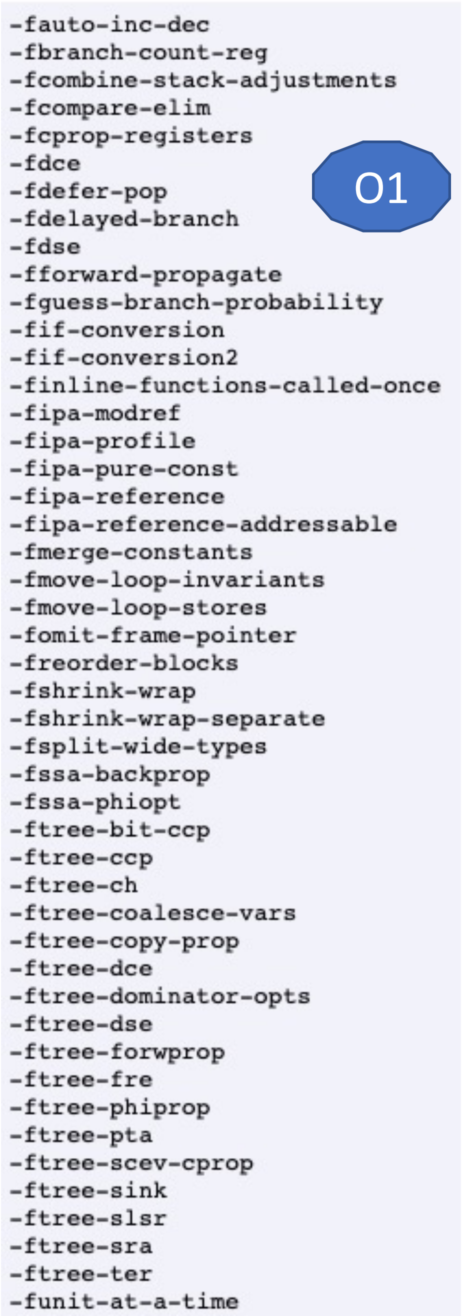


Our Goal


hostid
expr
cksum
join
rm
base32
tty
echo
false
hostname
comm
arch
tee
env
date
tail
chmod
true
nice
chcon
dir
mv
dd
id
cut
ls
groups
du
yes
head
chown
fold
sleep
basename
dircolors
df
chroot
touch
kill
cat
ln
factor
csplit
base64
whoami
fmt
wc
cp
expand
Automated
BusyBox

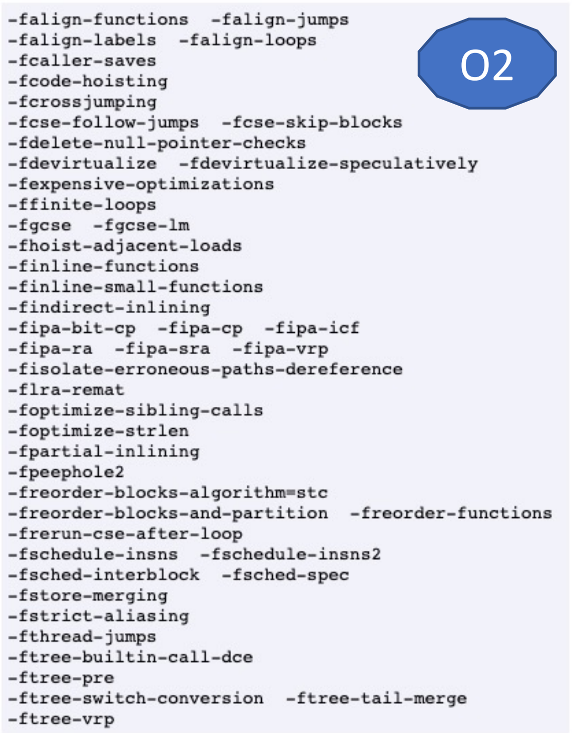
Compilers are Powerful





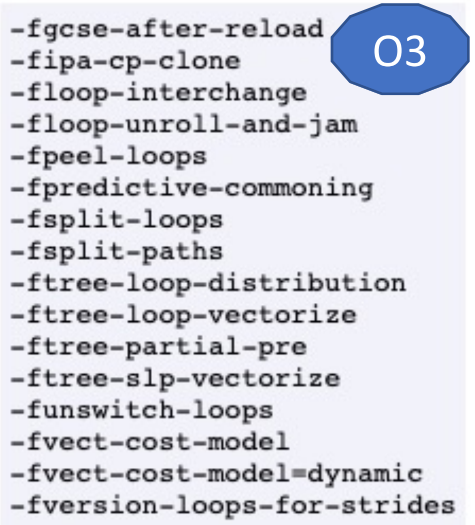
O1

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion
-fif-conversion2
-finline-functions-called-once
-fipa-modref
-fipa-profile
-fipa-pure-const
-fipa-reference
-fipa-reference-addressable
-fmerge-constants
-fmove-loop-invariants
-fmove-loop-stores
-fomit-frame-pointer
-freorder-blocks
-fshrink-wrap
-fshrink-wrap-separate
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-pta
-ftree-scev-cprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-ter
-funit-at-a-time
```



O2

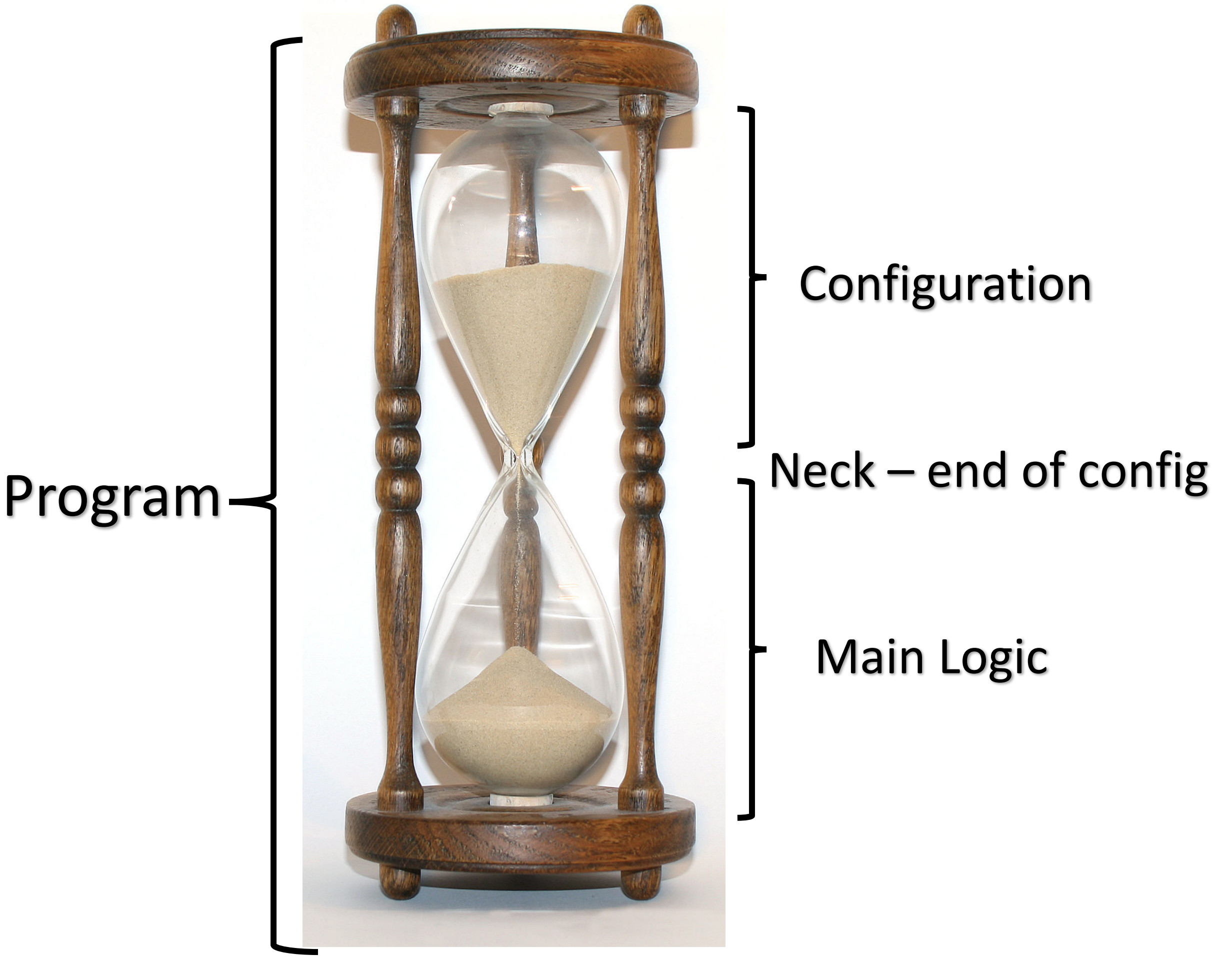
```
-falign-functions  -falign-jumps
-falign-labels  -falign-loops
-fcaller-saves
-fcode-hoisting
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize  -fdevirtualize-speculatively
-fexpensive-optimizations
-ffinite-loops
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-functions
-finline-small-functions
-findirect-inlining
-fipa-bit-cp  -fipa-cp  -fipa-icf
-fipa-ra  -fipa-sra  -fipa-vrp
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition  -freorder-functions
-frerun-cse-after-loop
-fschedule-insns  -fschedule-insns2
-fsched-interblock  -fsched-spec
-fstore-merging
-fstrict-aliasing
-fthread-jumps
-ftree-builtin-call-dce
-ftree-pre
-ftree-switch-conversion  -ftree-tail-merge
-ftree-vrp
```



O3

```
-fgcse-after-reload
-fipa-cp-clone
-floop-interchange
-floop-unroll-and-jam
-fpeel-loops
-fpredictive-commoning
-fsplit-loops
-fsplit-paths
-ftree-loop-distribution
-ftree-loop-vectorize
-ftree-partial-pre
-ftree-slp-vectorize
-funswitch-loops
-fvect-cost-model
-fvect-cost-model=dynamic
-fversion-loops-for-strides
```

Disciplined Software Development



Program

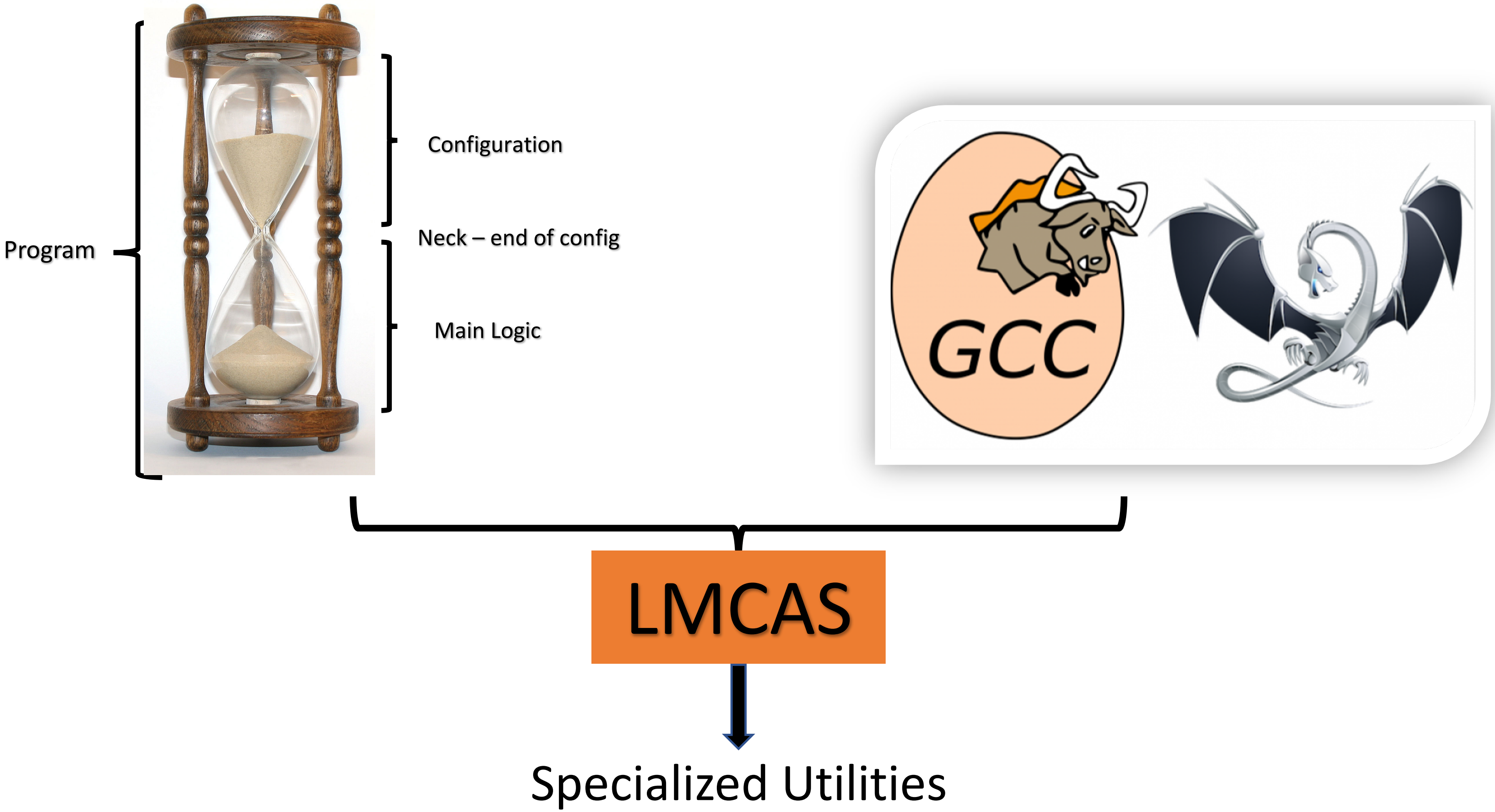
Configuration

Neck – end of config

Main Logic

Meinicke et al., Exploring differences and commonalities between feature flags and configuration options. ICSE-SEIP'20

LMCAS Pillars



Program

Configuration

Neck – end of config

Main Logic

LMCAS

Specialized Utilities

https://github.com/Mohannadcse/LMCAS_Demo

Agenda

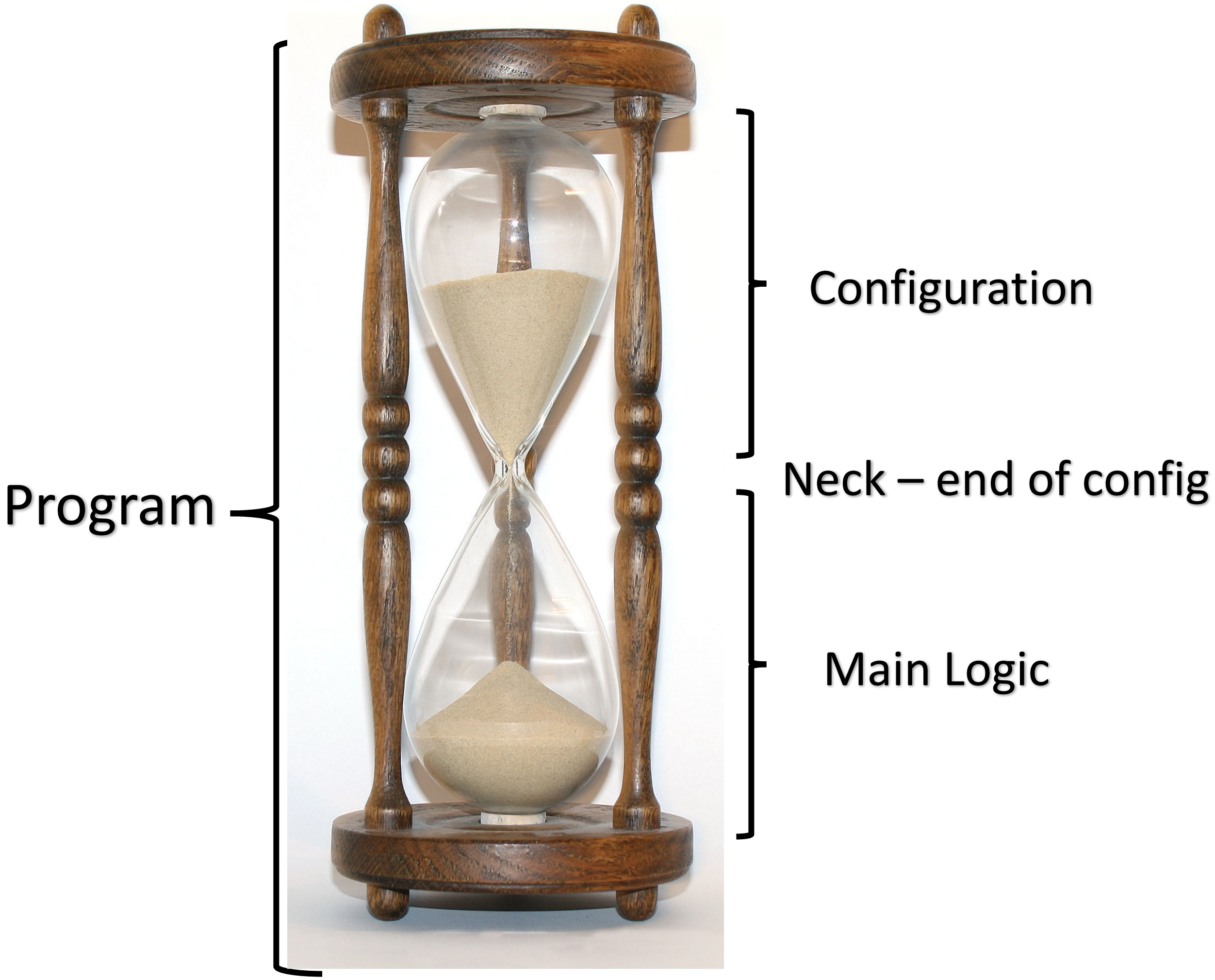
- Use Case
- Introducing LMCAS
- Questions
- Demo

Survey

- Do you follow Disciplined Software Development?

YES □

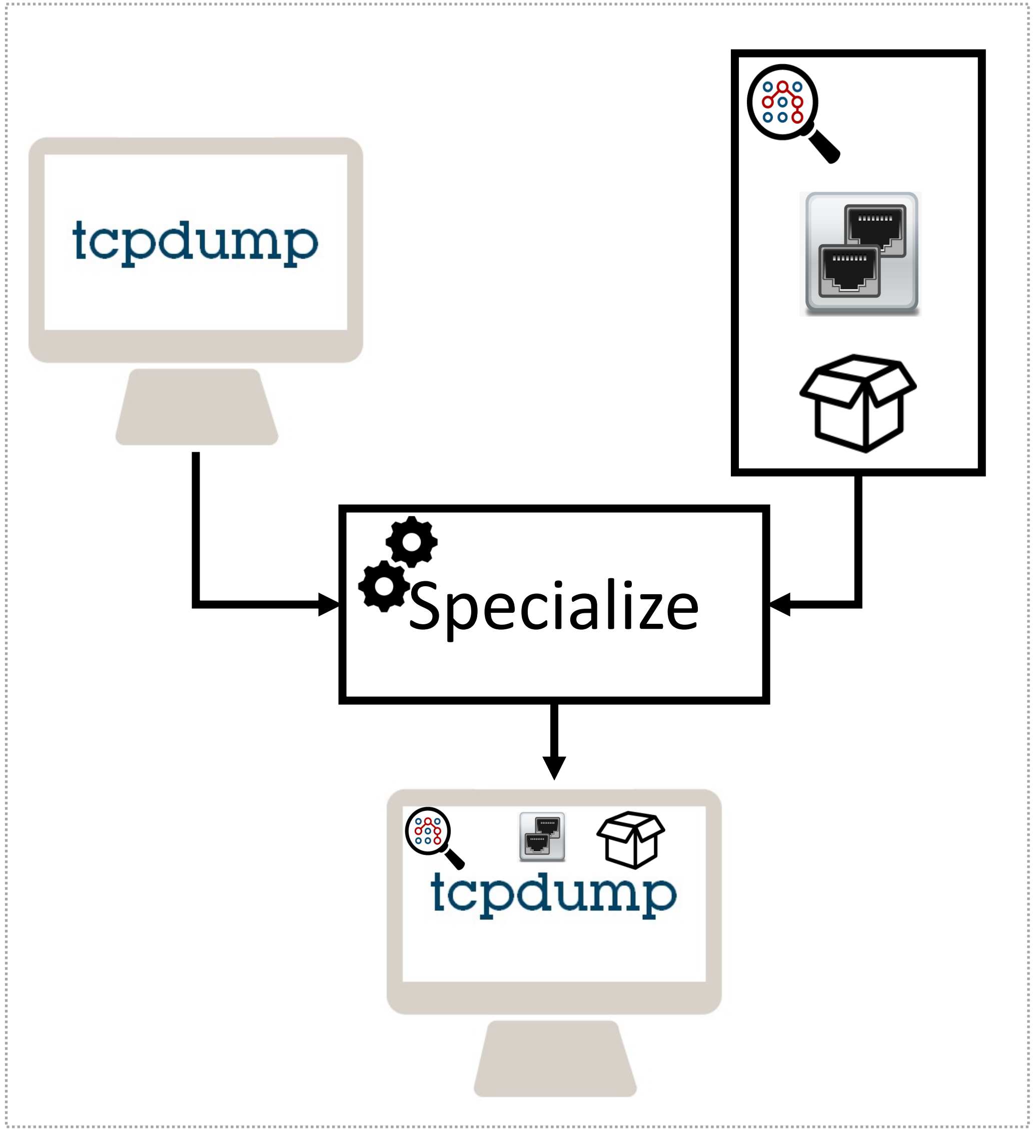
NO □



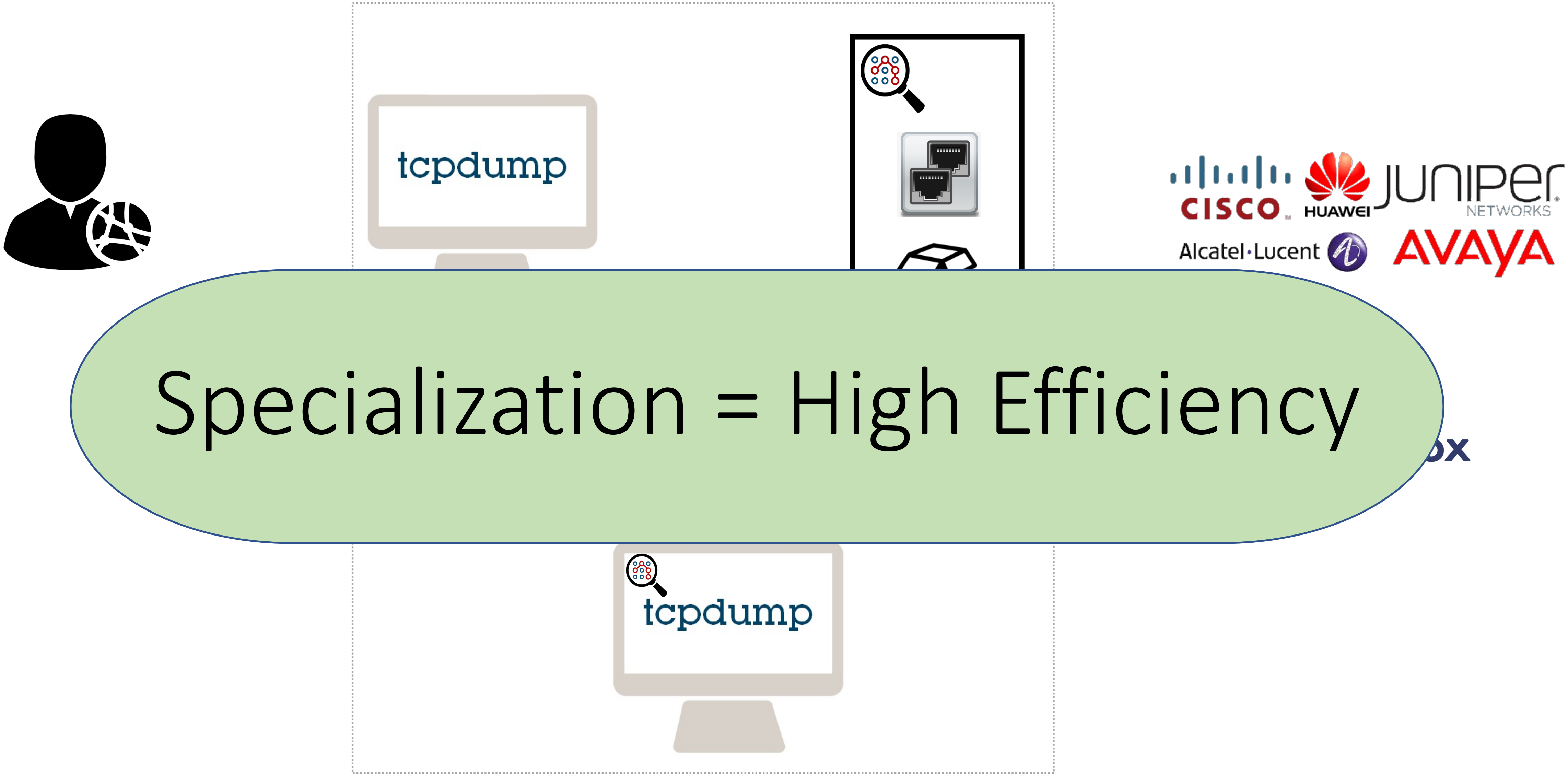
Agenda

- Use Case
- Introducing LMCAS
- Questions
- Demo

Use Case (Network Monitoring)



Use Case (Network Monitoring)



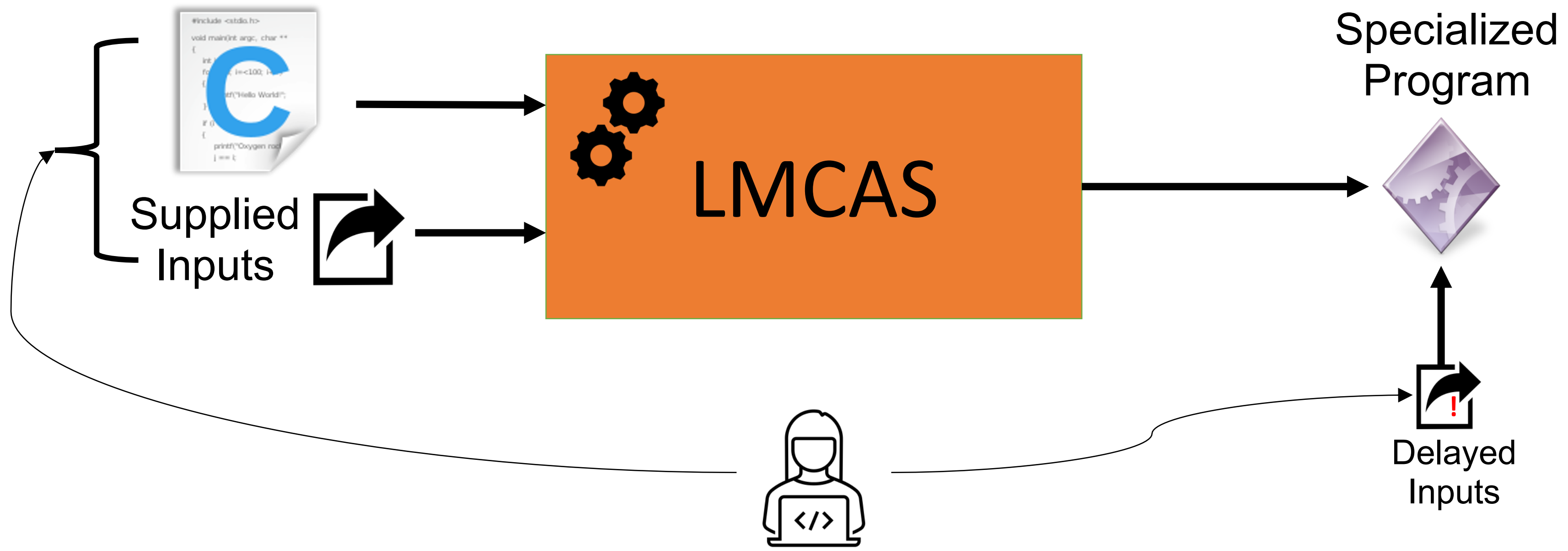
Specialization = High Efficiency

Other Use Cases

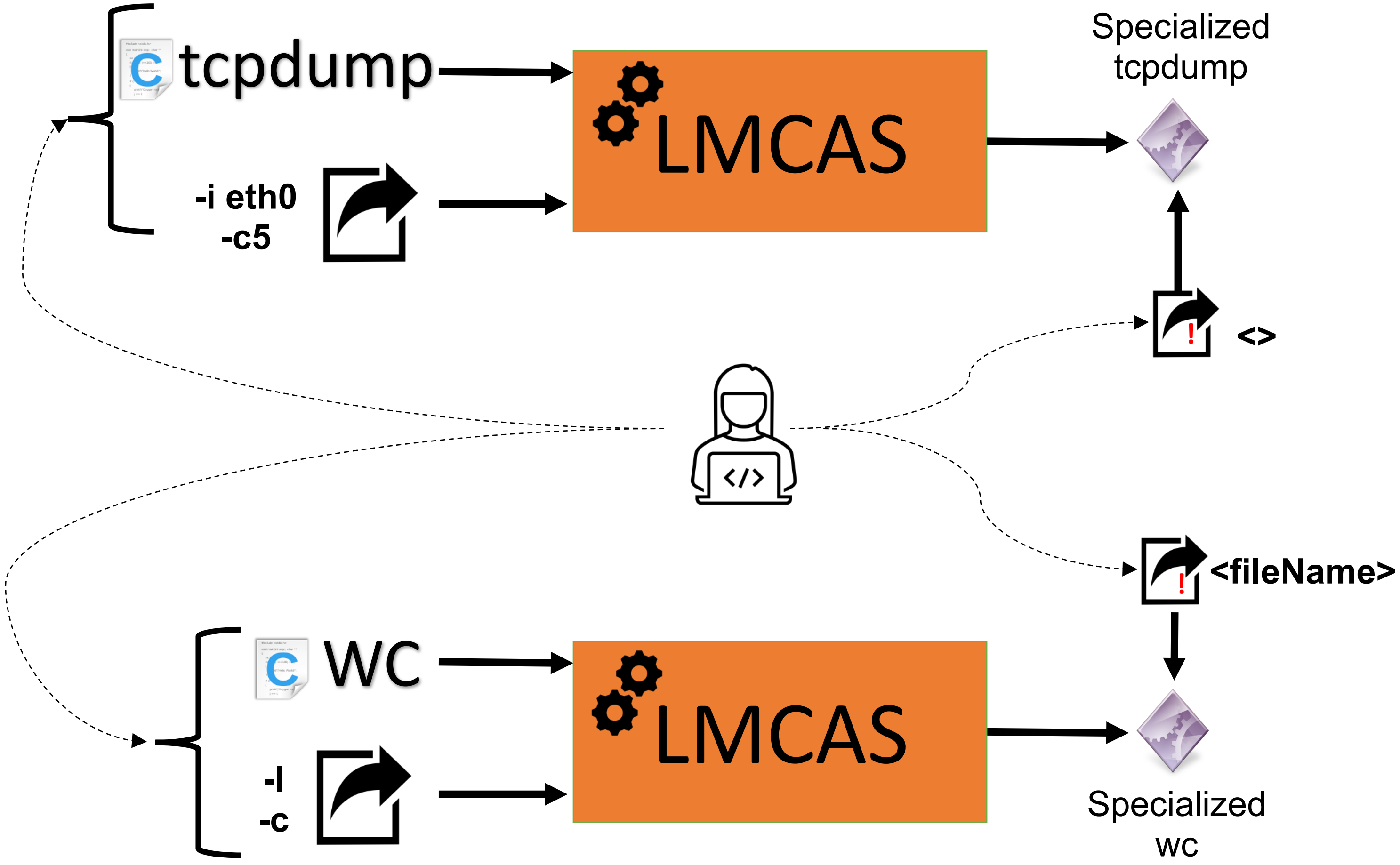
Agenda

- Use Case
- Introducing LMCAS
- Questions
- Demo

LMCAS Workflow (Partial Evaluation)

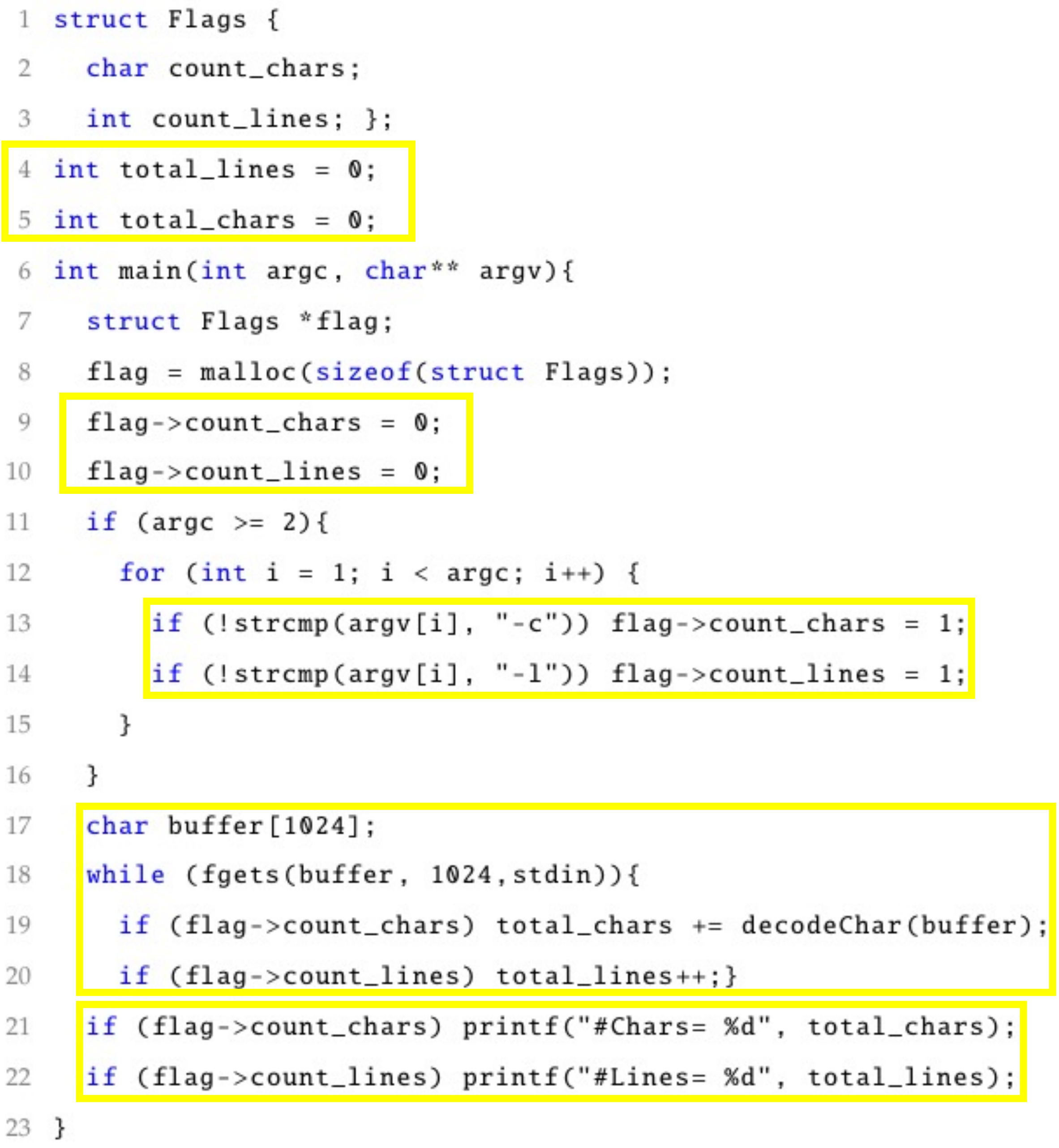


LMCAS Workflow - Examples



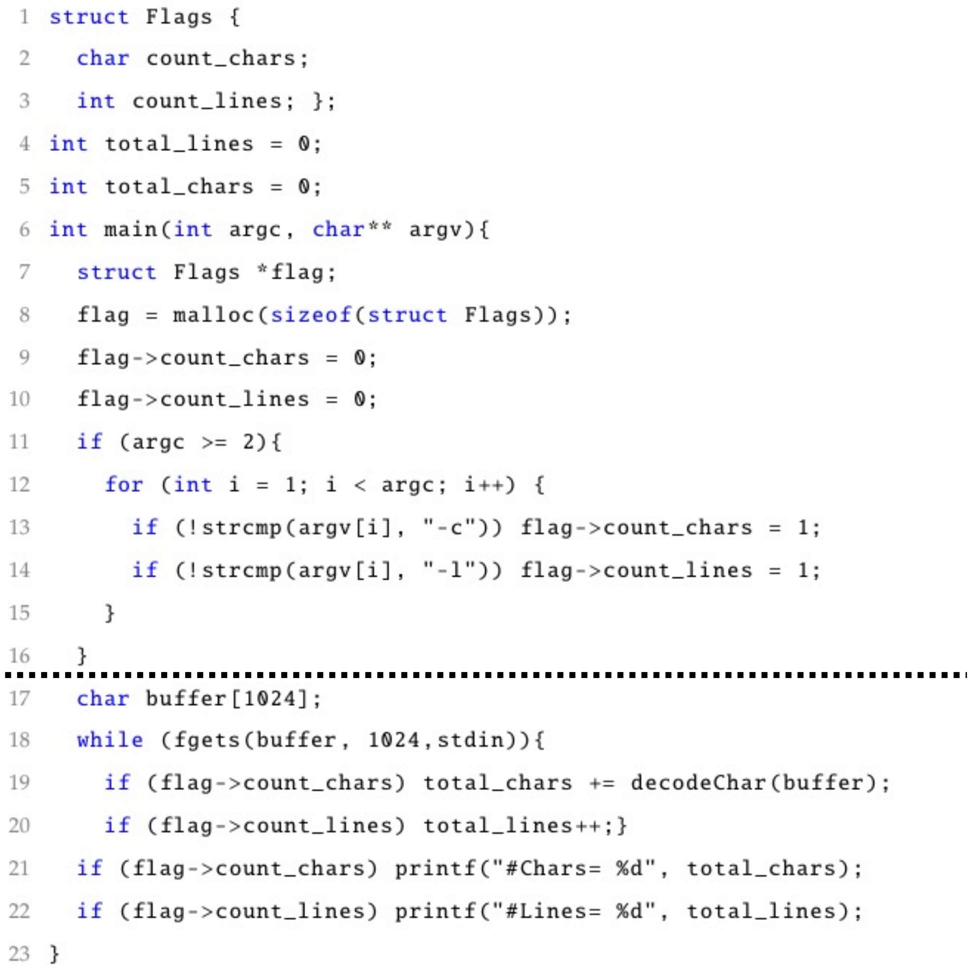
Illustrative Example

- scaled-down version of the *wc* utility
    1. Line count
    2. Char count

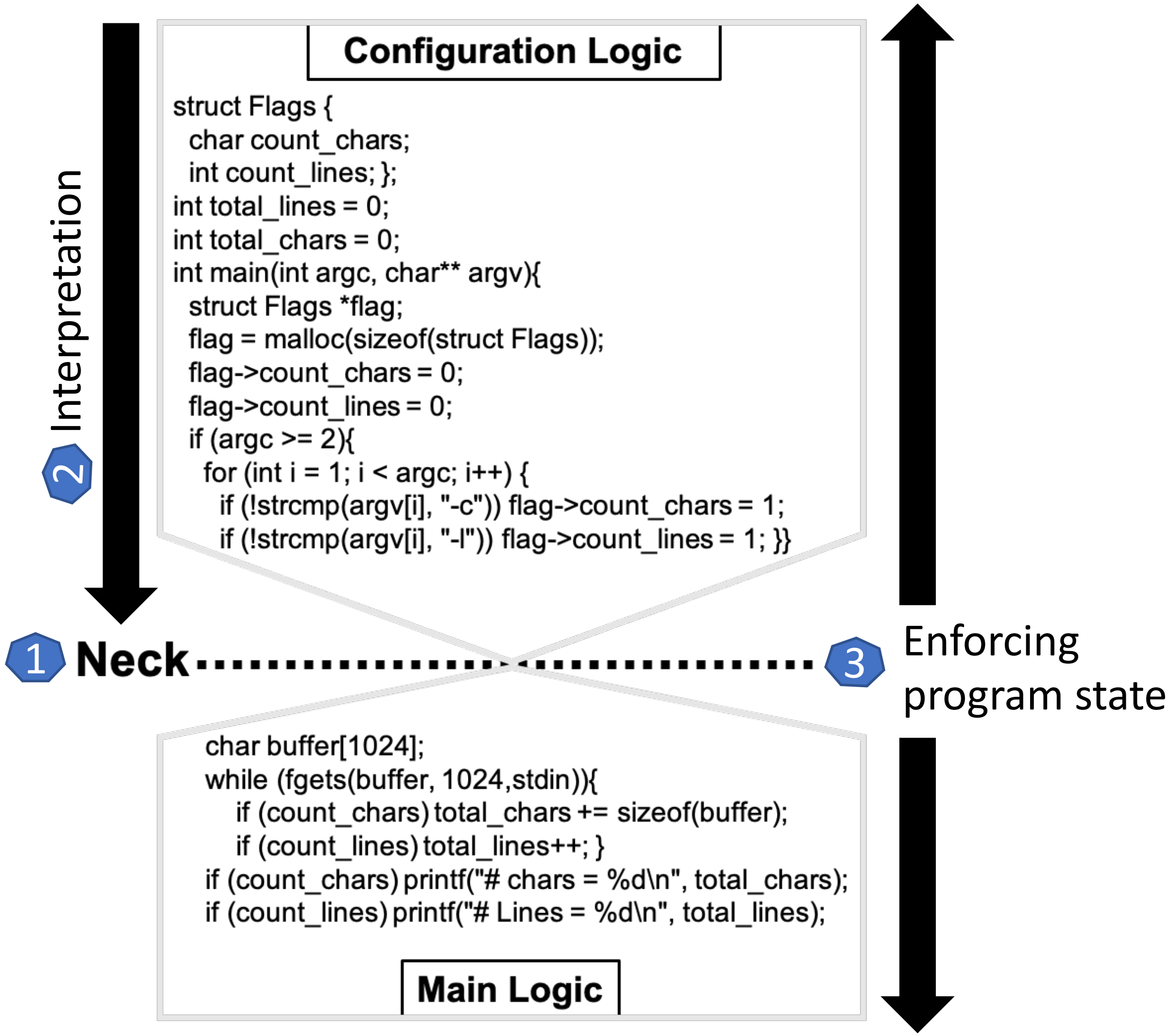


```c
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
    }
  }
  char buffer[1024];
  while (fgets(buffer, 1024,stdin)){
    if (flag->count_chars) total_chars += decodeChar(buffer);
    if (flag->count_lines) total_lines++;}
  if (flag->count_chars) printf("#Chars= %d", total_chars);
  if (flag->count_lines) printf("#Lines= %d", total_lines);
}
```

LMCAS Approach



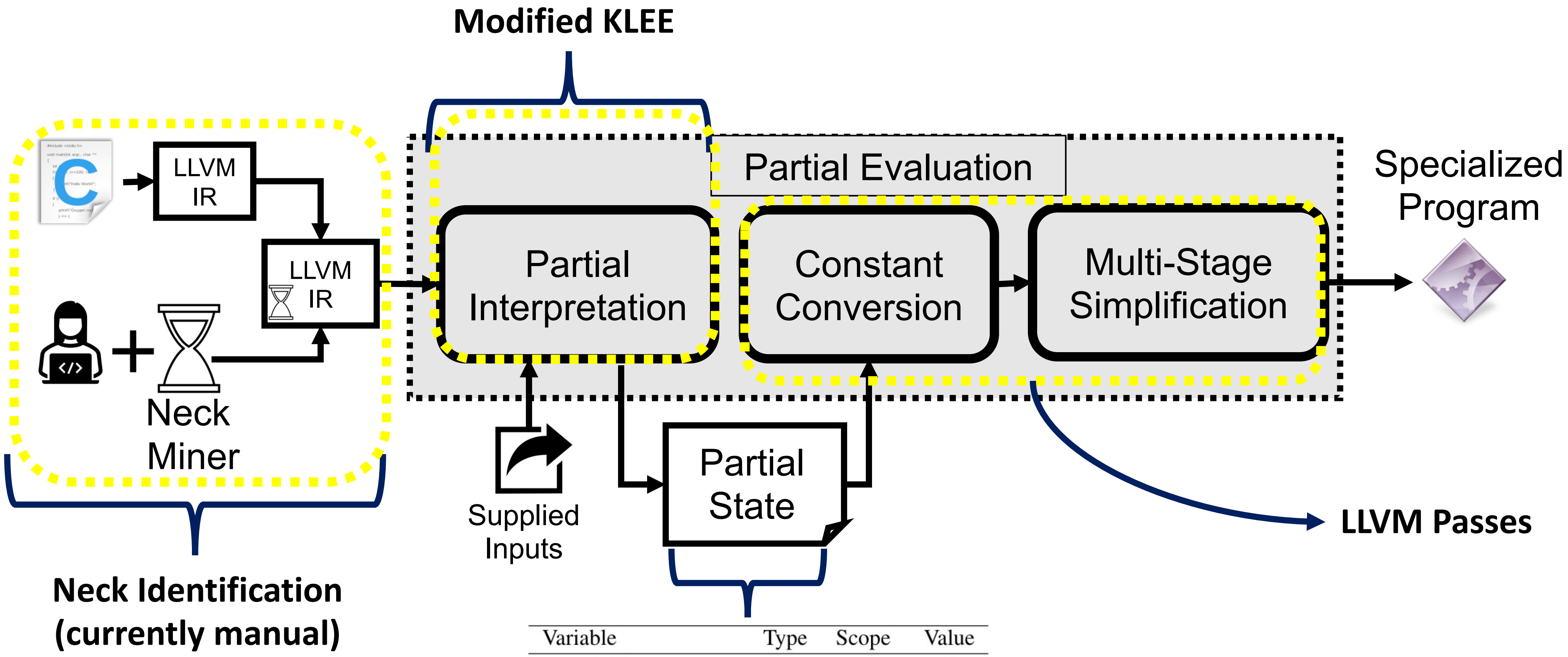
```
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
    }
  }
  char buffer[1024];
  while (fgets(buffer, 1024,stdin)){
    if (flag->count_chars) total_chars += decodeChar(buffer);
    if (flag->count_lines) total_lines++;}
  if (flag->count_chars) printf("#Chars= %d", total_chars);
  if (flag->count_lines) printf("#Lines= %d", total_lines);
}
```



```
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1; }}
```

```
char buffer[1024];
while (fgets(buffer, 1024,stdin)){
    if (count_chars) total_chars += sizeof(buffer);
    if (count_lines) total_lines++; }
if (count_chars) printf("# chars = %d\n", total_chars);
if (count_lines) printf("# Lines = %d\n", total_lines);
```

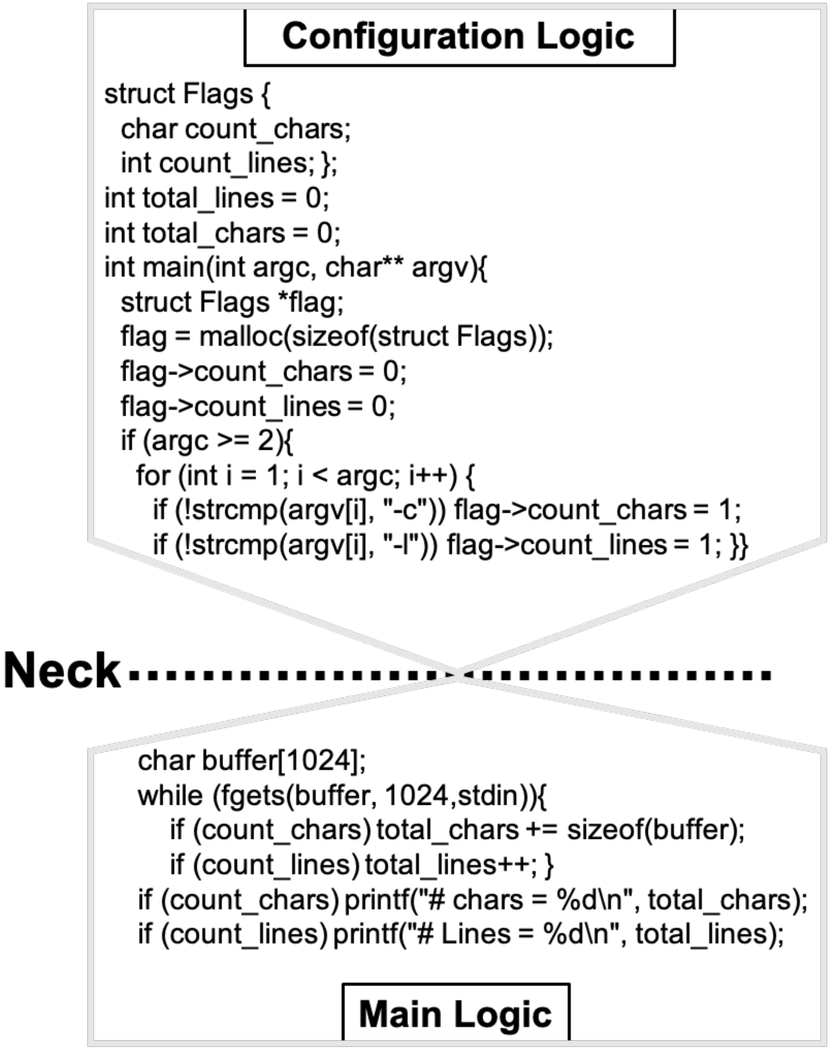
LMCAS Architecture



Variable	Type	Scope	Value
total_lines	int	Global	0
total_chars			0
flag->count_lines	int	Local	1
flag->count_chars	char		0

Neck Properties

- Neck should be an articulation point in the CFG:
  - Dominator of main logic nodes
  - Always executed
    - Reachable from the entry
  - Executed once
    - Outside any loop structure



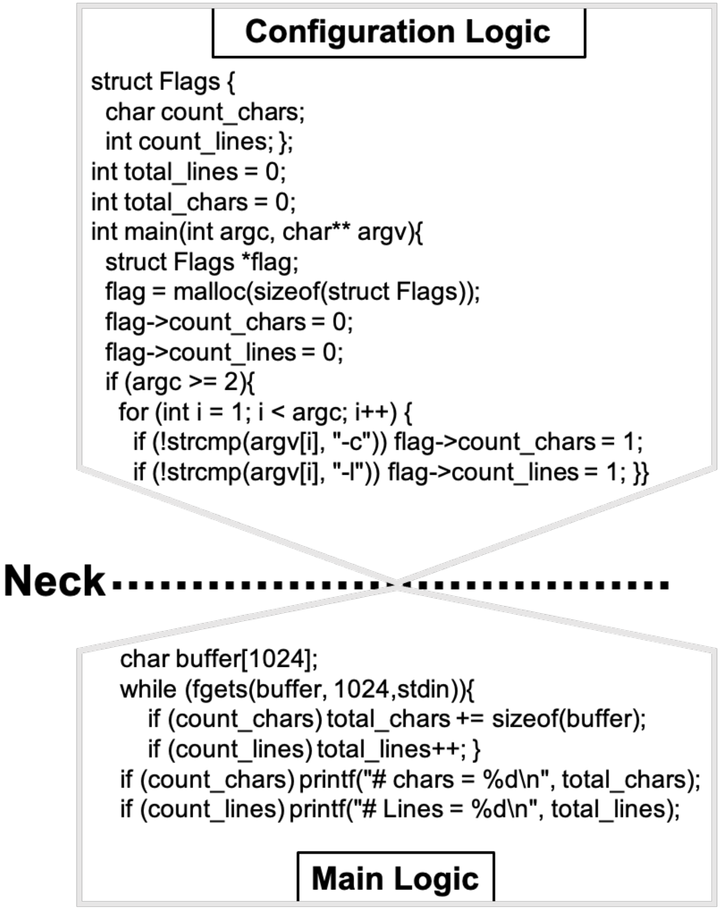
```
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1; }}
```

```
    char buffer[1024];
    while (fgets(buffer, 1024,stdin)){
        if (count_chars) total_chars += sizeof(buffer);
        if (count_lines) total_lines++; }
    if (count_chars) printf("# chars = %d\n", total_chars);
    if (count_lines) printf("# Lines = %d\n", total_lines);
```

Partial Interpreter

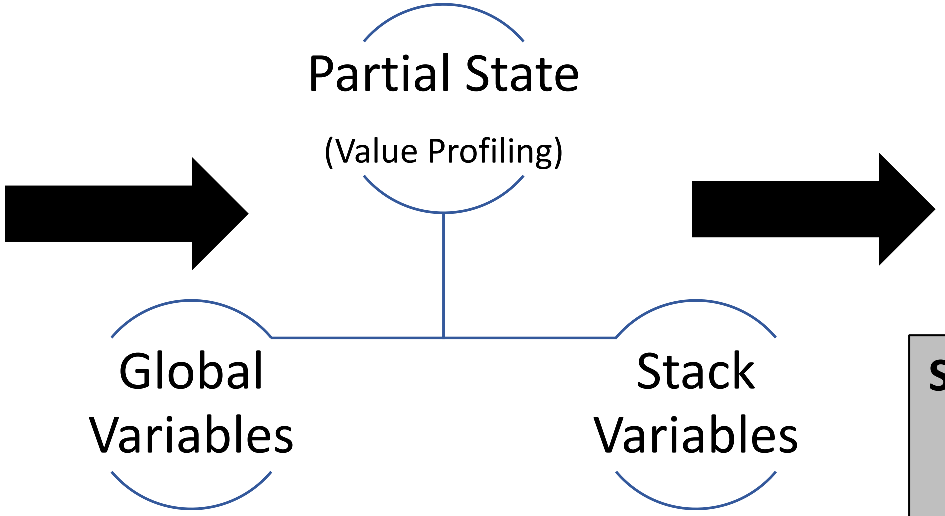
Modified KLEE

Program Partial State



```
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1; }}
```

```
    char buffer[1024];
    while (fgets(buffer, 1024,stdin)){
        if (count_chars) total_chars += sizeof(buffer);
        if (count_lines) total_lines++; }
    if (count_chars) printf("# chars = %d\n", total_chars);
    if (count_lines) printf("# Lines = %d\n", total_lines);
```



Partial State

(Value Profiling)

Global Variables

Stack Variables

Variable	Type	Scope	Value
total_lines	int	Global	0
total_chars			0
flag->count_lines	int	Local	1
flag->count_chars	char		0

Stored as txt files:

-globals=gbls.txt

-primitive stack=primitiveLocals.txt

-struct stack=customizedLocals.txt

-ptr to struct=ptrToStructLocals.txt

-ptr to primitive=ptrToPrimitiveLocals.txt

-string Vars=stringVars.txt

-nested structs=nestedStructLocals.txt

Partial Interpreter Implementation

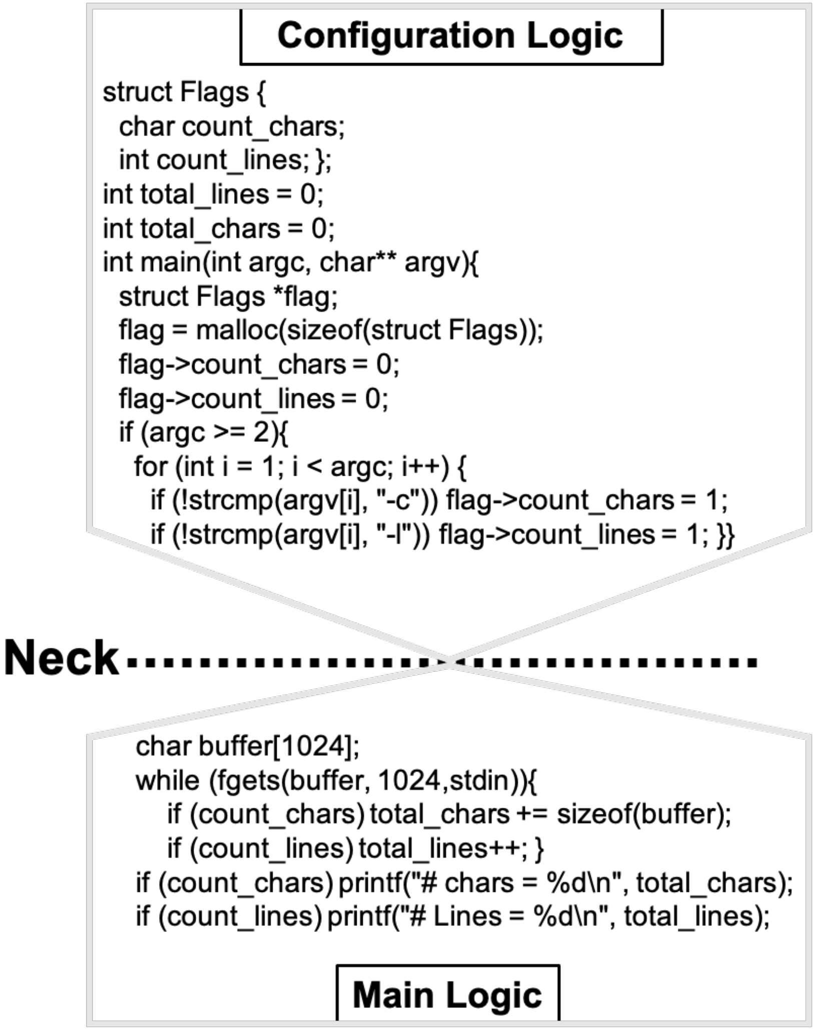
- KLEE 2.1
- LLVM 10.0

Compiler-Assisted Optimizations

Constant Conversion

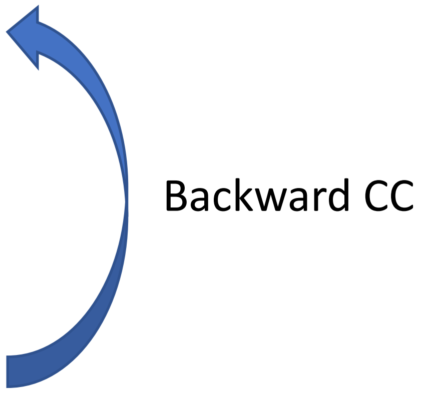
Multi-stage Simplifications

Constant Conversion (CC)



```
struct Flags {
  char count_chars;
  int count_lines; };
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1; }}
```

```
    char buffer[1024];
    while (fgets(buffer, 1024,stdin)){
        if (count_chars) total_chars += sizeof(buffer);
        if (count_lines) total_lines++; }
    if (count_chars) printf("# chars = %d\n", total_chars);
    if (count_lines) printf("# Lines = %d\n", total_lines);
```

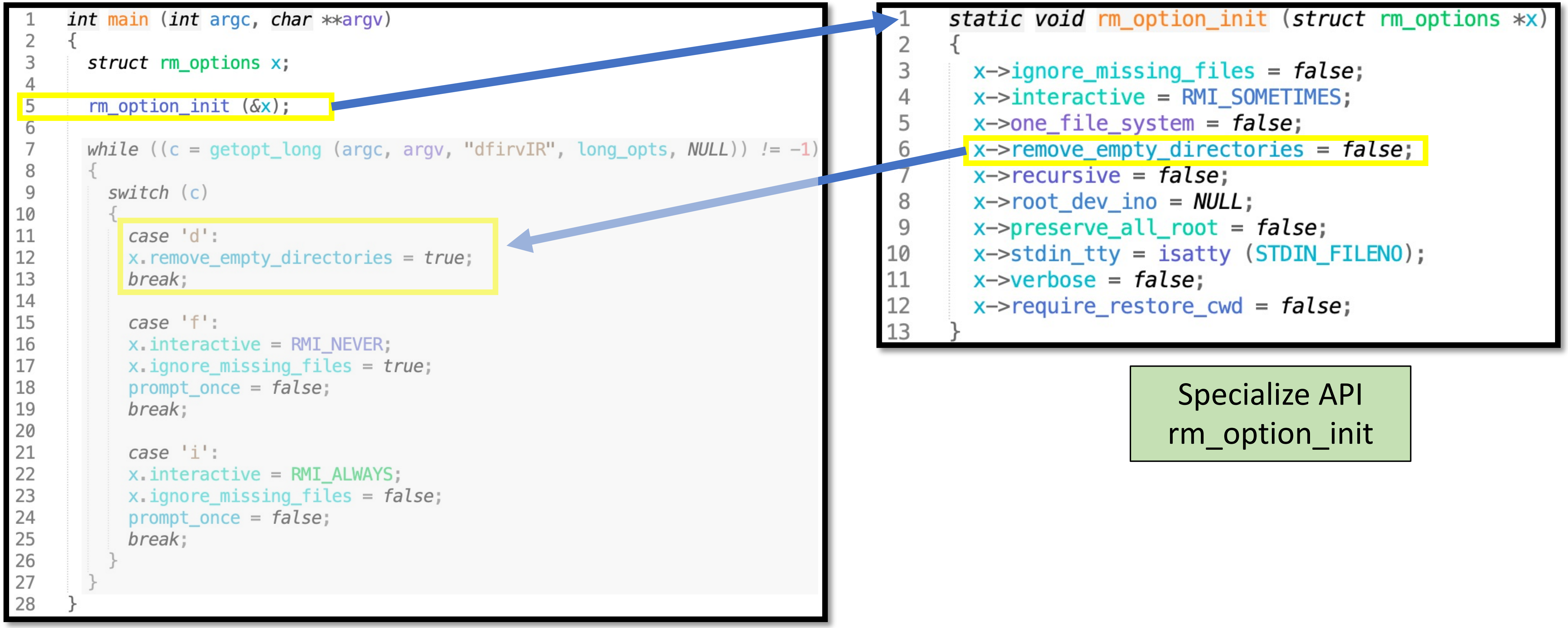


Variable	Type	Scope	Value
`total_lines`	`int`	Global	0
`total_chars`			0
`flag->count_lines`	`int`	Local	1
`flag->count_chars`	`char`		0

Constant Conversion
LLVM Pass

-globals=**gbls.txt**
-plocals=**primitiveLocals.txt**
-clocals=**customizedLocals.txt**
-ptrStructlocals=**ptrToStructLocals.txt**
-ptrToPrimLocals=**ptrToPrimitiveLocals.txt**
-stringVars=**stringVars.txt**
-nestedStrcts=**nestedStructLocals.txt**

CC Pre-neck



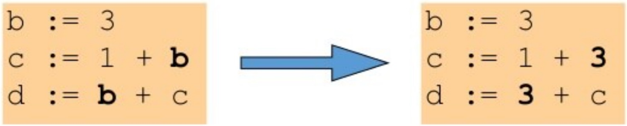
```
int main (int argc, char **argv)
{
  struct rm_options x;

  rm_option_init (&x);

  while ((c = getopt_long (argc, argv, "dfirvIR", long_opts, NULL)) != -1)
  {
    switch (c)
    {
      case 'd':
      x.remove_empty_directories = true;
      break;

      case 'f':
      x.interactive = RMI_NEVER;
      x.ignore_missing_files = true;
      prompt_once = false;
      break;

      case 'i':
      x.interactive = RMI_ALWAYS;
      x.ignore_missing_files = false;
      prompt_once = false;
      break;
    }
  }
}
```

```
static void rm_option_init (struct rm_options *x)
{
  x->ignore_missing_files = false;
  x->interactive = RMI_SOMETIMES;
  x->one_file_system = false;
  x->remove_empty_directories = false;
  x->recursive = false;
  x->root_dev_ino = NULL;
  x->preserve_all_root = false;
  x->stdin_tty = isatty (STDIN_FILENO);
  x->verbose = false;
  x->require_restore_cwd = false;
}
```

Specialize API rm_option_init

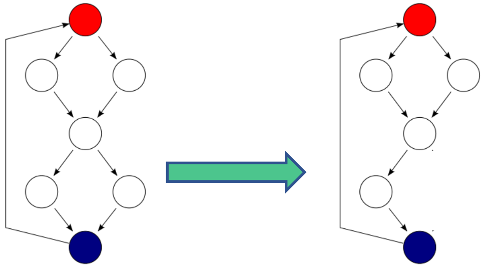
rm GNU Coreutils

Multi-stage Simplifications



```
b := 3
c := 1 + b
d := b + c
```

```
b := 3
c := 1 + 3
d := 3 + c
```



Constant Propagation

- Standard LLVM pass

Simplifying CFG

- Standard LLVM Pass

Cleaning up

- Customized LLVM Pass

Cleaning up

Removing unused functions

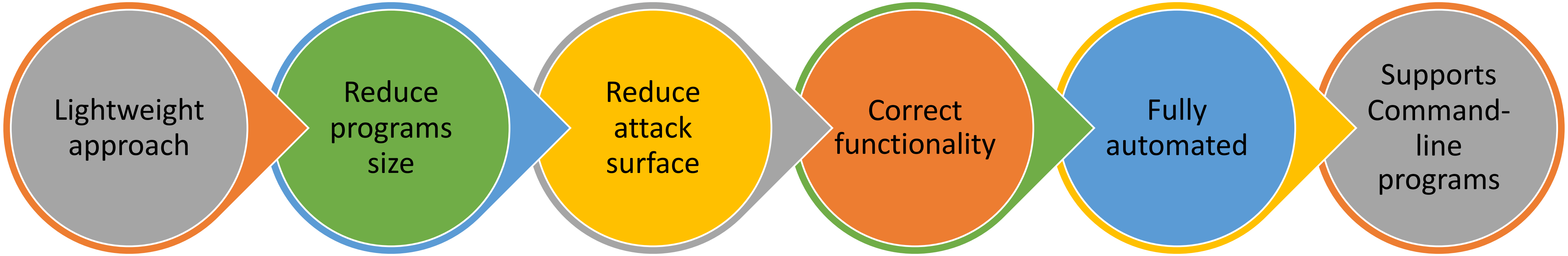
- Iterate the Call Graph
- Check the number of users of each function
- Remove the function if number of users is zero
  - function pointer won’t be removed

Removing unused variables in the remained functions

- Local and global

We used users() LLVM API

LMCAS Power



Lightweight approach

Reduce programs size

Reduce attack surface

Correct functionality

Fully automated

Supports Command-line programs

- No test cases/heuristics are required

- Binary
- Functions
- Basic blocks
- Instructions

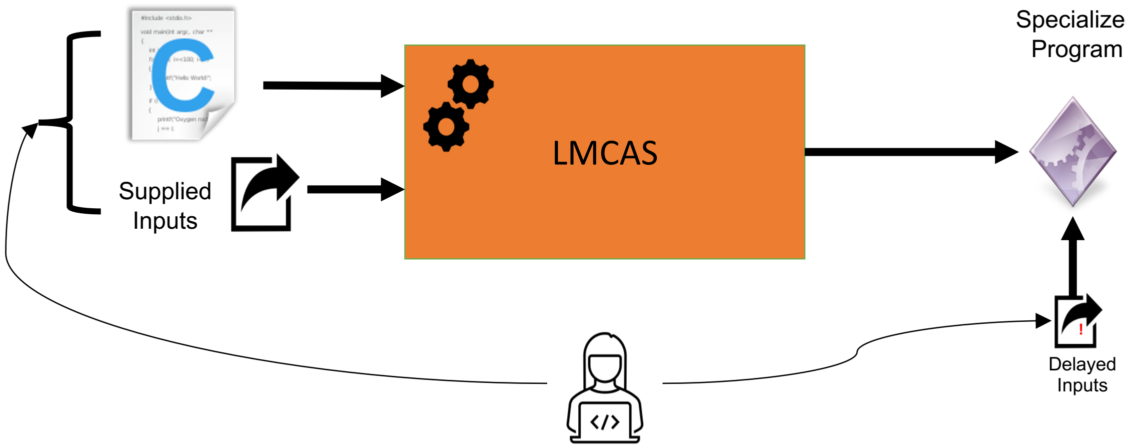
- Eliminate known CVEs
- Reduce gadgets

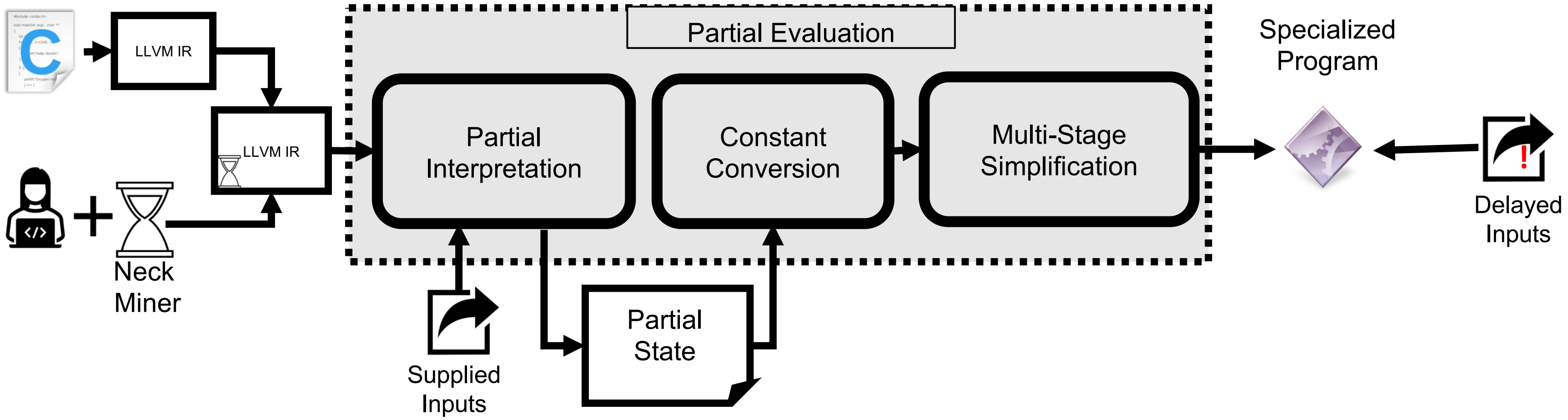
- Except the neck
- Ongoing work

Agenda

- Use Case
- Introducing LMCAS
- Questions
- Demo

LMCAS - Summary





Agenda

- Use Case
- Introducing LMCAS
- Questions
- Demo

Demo Roadmap

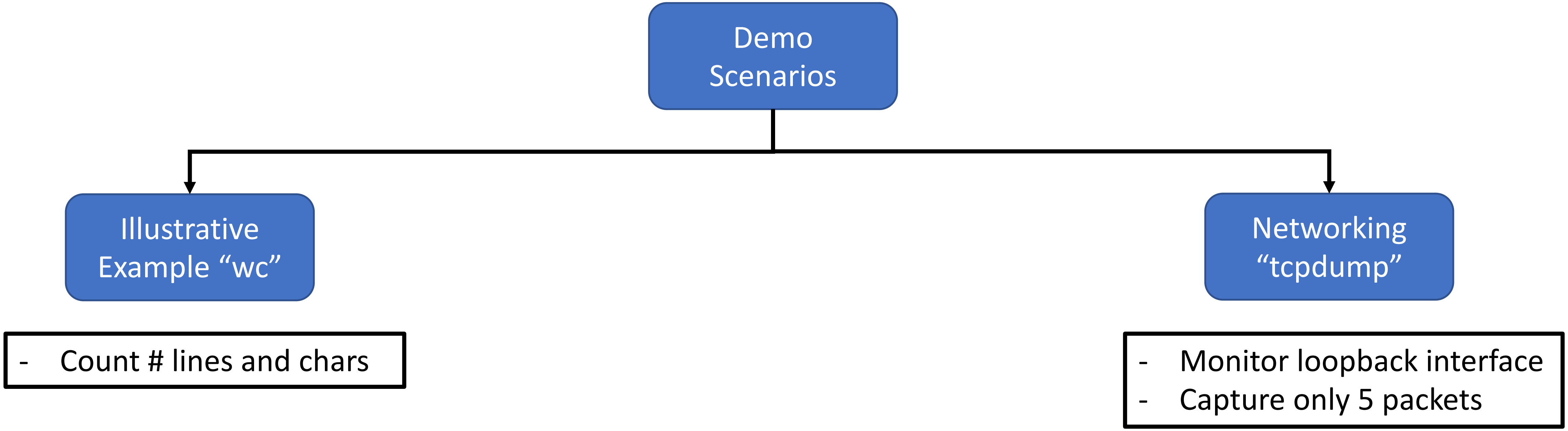
Two debloating scenarios

See size reduction statistics

See gadgets reduction statistics

Verify the functionality of the debloated program

Debloating Scenarios



Demo Scenarios

Illustrative Example “wc”

- Count # lines and chars

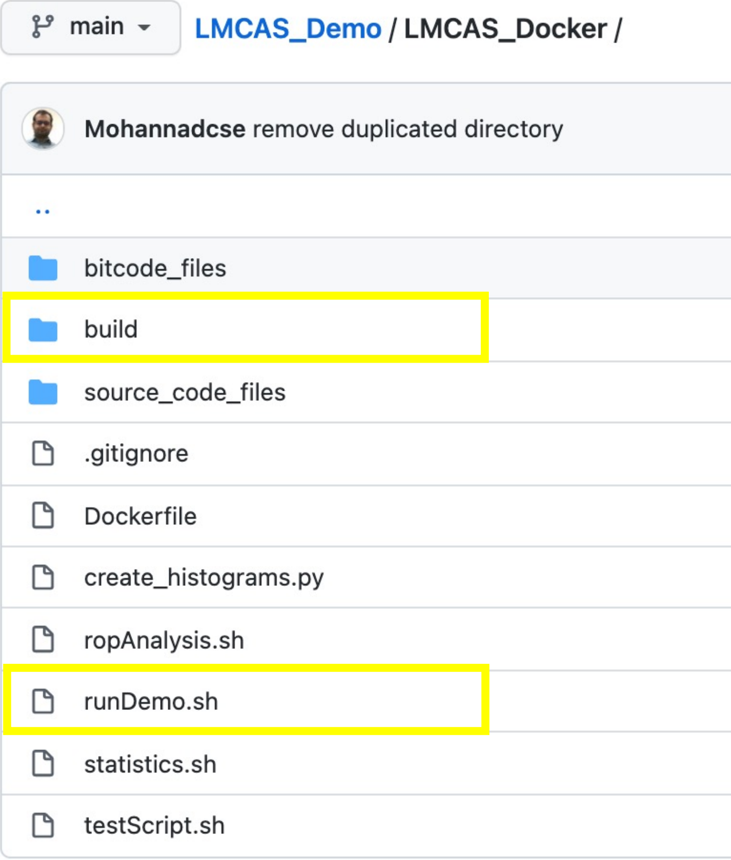
Networking “tcpdump”

- Monitor loopback interface
- Capture only 5 packets

LMCAS Docker and programs used in the demo are available at: https://github.com/Mohannadcse/LMCAS_Demo

18 Programs

LMCAS Demo Repo

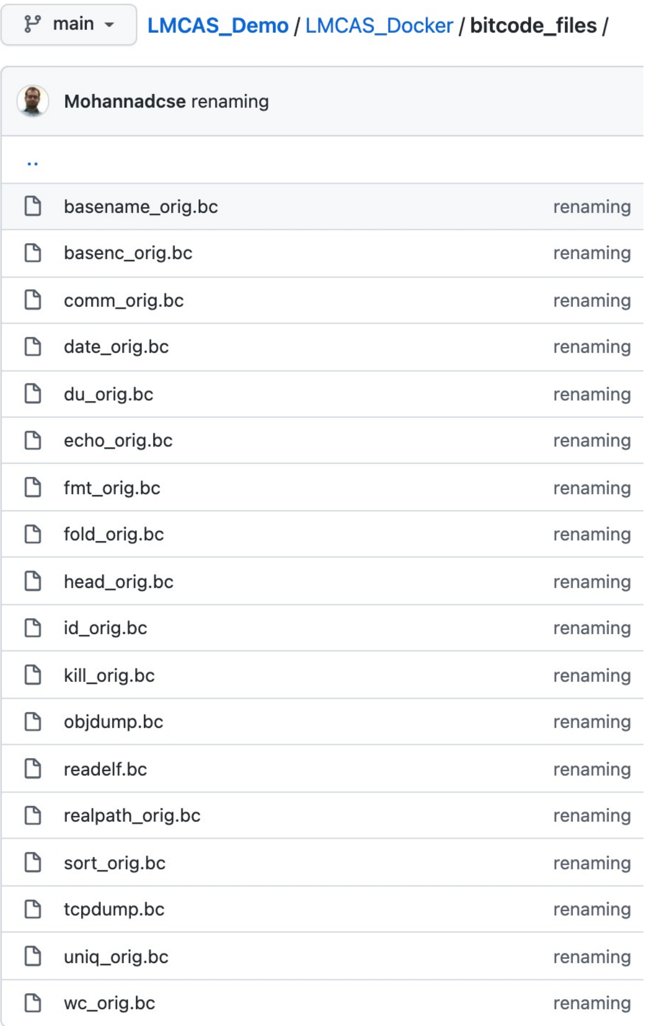


main LMCAS_Demo / LMCAS_Docker /

Mohannadcse remove duplicated directory

..
bitcode_files
build
source_code_files
.gitignore
Dockerfile
create_histograms.py
ropAnalysis.sh
runDemo.sh
statistics.sh
testScript.sh

WLLVM



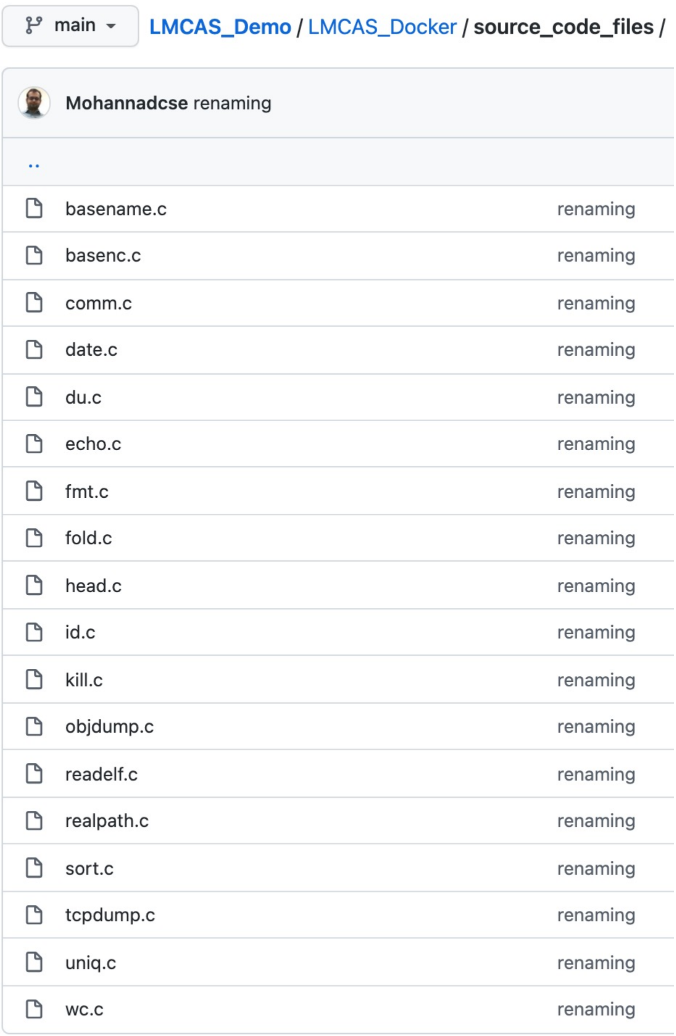
main LMCAS_Demo / LMCAS_Docker / bitcode_files /

Mohannadcse renaming

..

basename_orig.bc	renaming
basenc_orig.bc	renaming
comm_orig.bc	renaming
date_orig.bc	renaming
du_orig.bc	renaming
echo_orig.bc	renaming
fmt_orig.bc	renaming
fold_orig.bc	renaming
head_orig.bc	renaming
id_orig.bc	renaming
kill_orig.bc	renaming
objdump.bc	renaming
readelf.bc	renaming
realpath_orig.bc	renaming
sort_orig.bc	renaming
tcpdump.bc	renaming
uniq_orig.bc	renaming
wc_orig.bc	renaming

klee_dump_memory



main LMCAS_Demo / LMCAS_Docker / source_code_files /

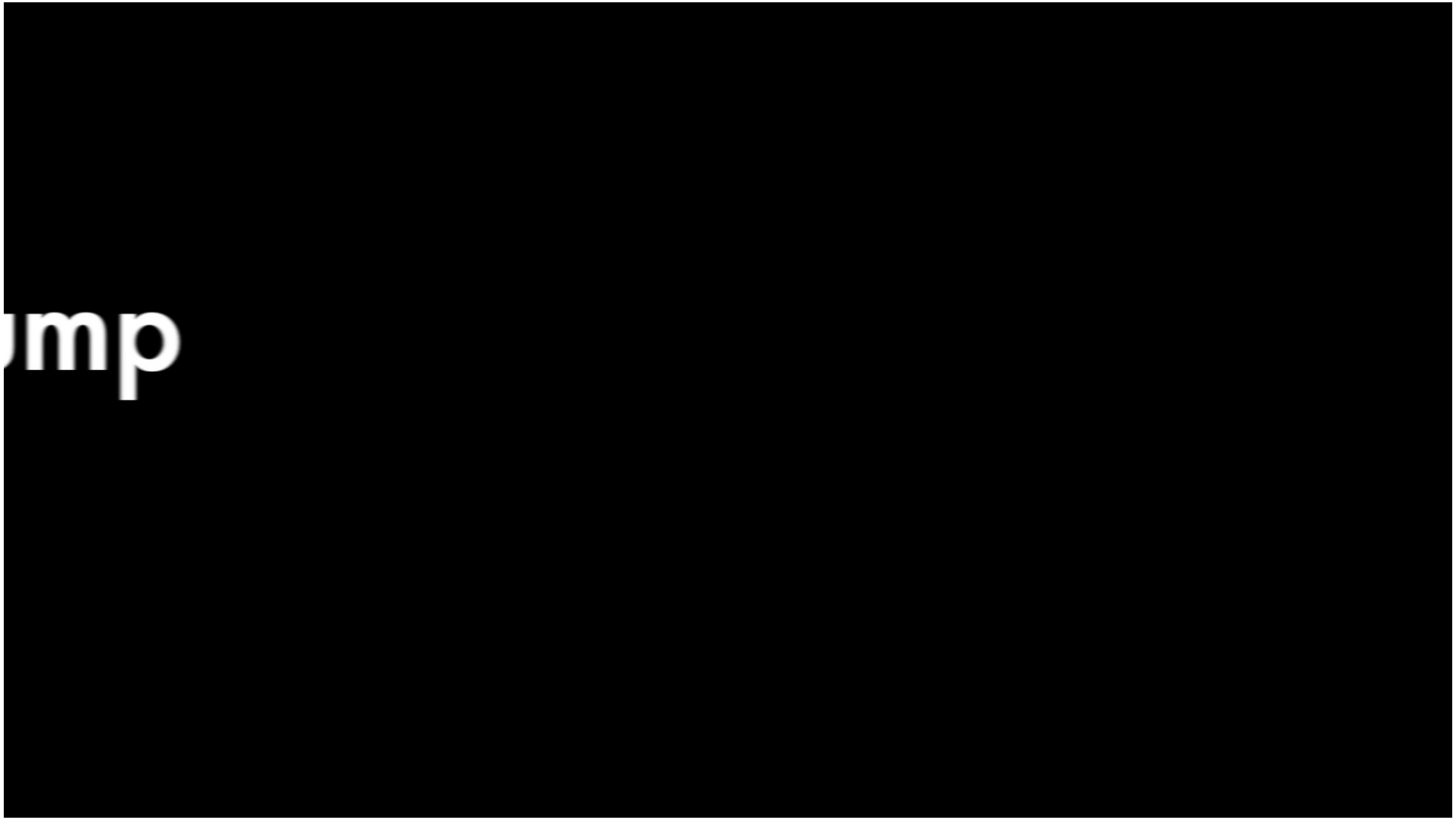
Mohannadcse renaming

..

basename.c	renaming
basenc.c	renaming
comm.c	renaming
date.c	renaming
du.c	renaming
echo.c	renaming
fmt.c	renaming
fold.c	renaming
head.c	renaming
id.c	renaming
kill.c	renaming
objdump.c	renaming
readelf.c	renaming
realpath.c	renaming
sort.c	renaming
tcpdump.c	renaming
uniq.c	renaming
wc.c	renaming

Specializing GNU wc

Specializing tcpdump



Summary

LMCAS specializes programs

- Compiler Optimizations
- Disciplined Implementation

LMCAS Benefits

- Lightweight approach
- Reducing attack surface
- Reducing program size

Ongoing project

- Automating the neck identification
- Leveraging data flow analysis
- Covering more real-world use cases

More Info

https://github.com/Mohannadcse/LMCAS_Demo

arXiv https://arxiv.org/abs/2109.02775

✉ mohannad@cs.wisc.edu

Malhanahnah