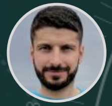
SAMSUNG

Mobile Security Team Warsaw, Poland

AUTHORS:



Bartosz Zator
b.zator@samsung.com

Adrian Nieć
a.niec@samsung.com

DPE SUMMIT

DPE in the Complex Low-Level System World

September 20 – 21, 2023

Introduction

Overview of the CAS system

A system that provides insight into how a S/W product is made and automates source code related operations

CAS

Code Aware Services

BAS
Build Awareness Service

Provides the build information gathered during product image creation

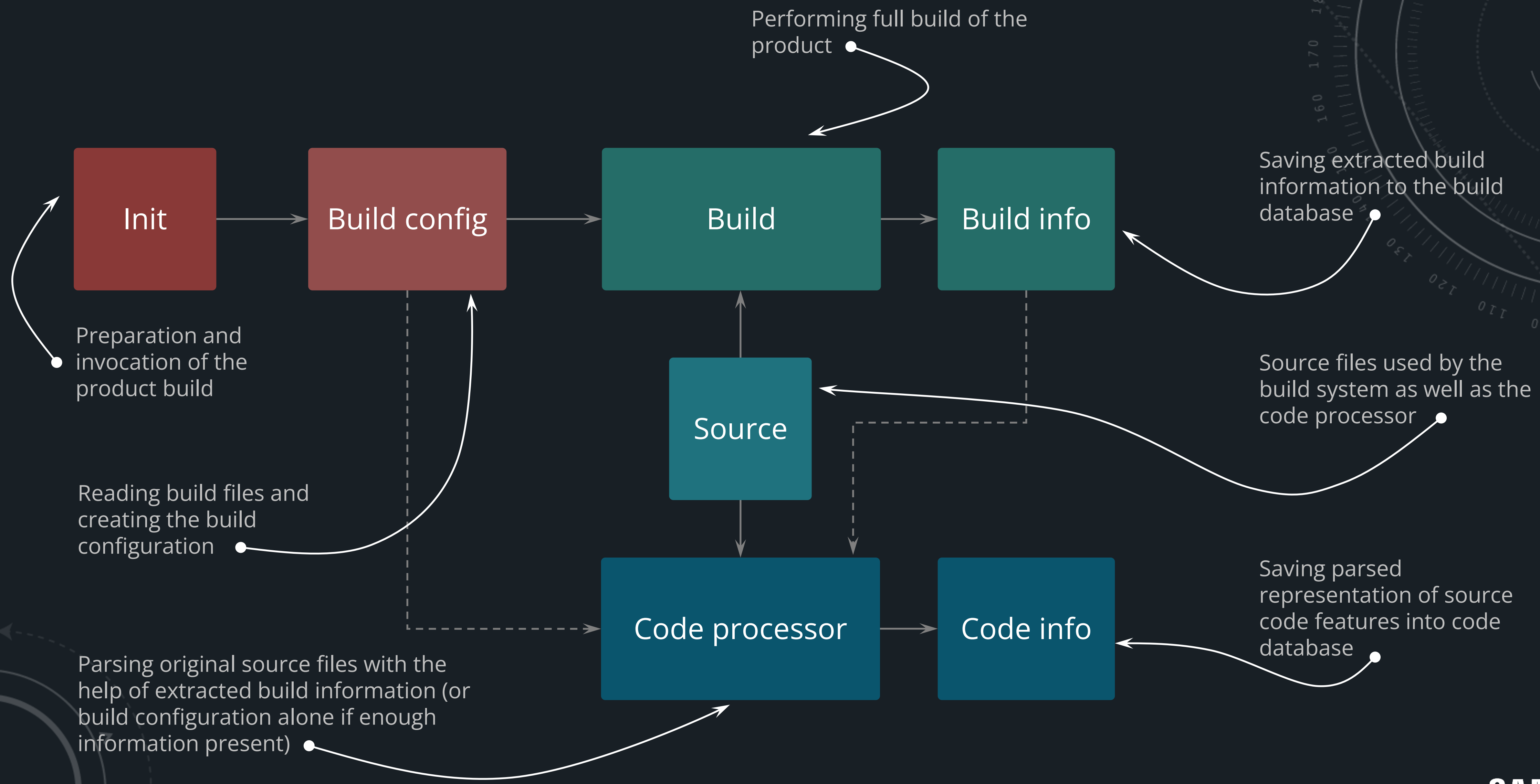
FTDB
Function/Type database

Transforms selected features from relevant source files into easily accessible format

Useful in various S/W engineering jobs: code search, IDE enhancement, test automation, build insight, code analysis etc.

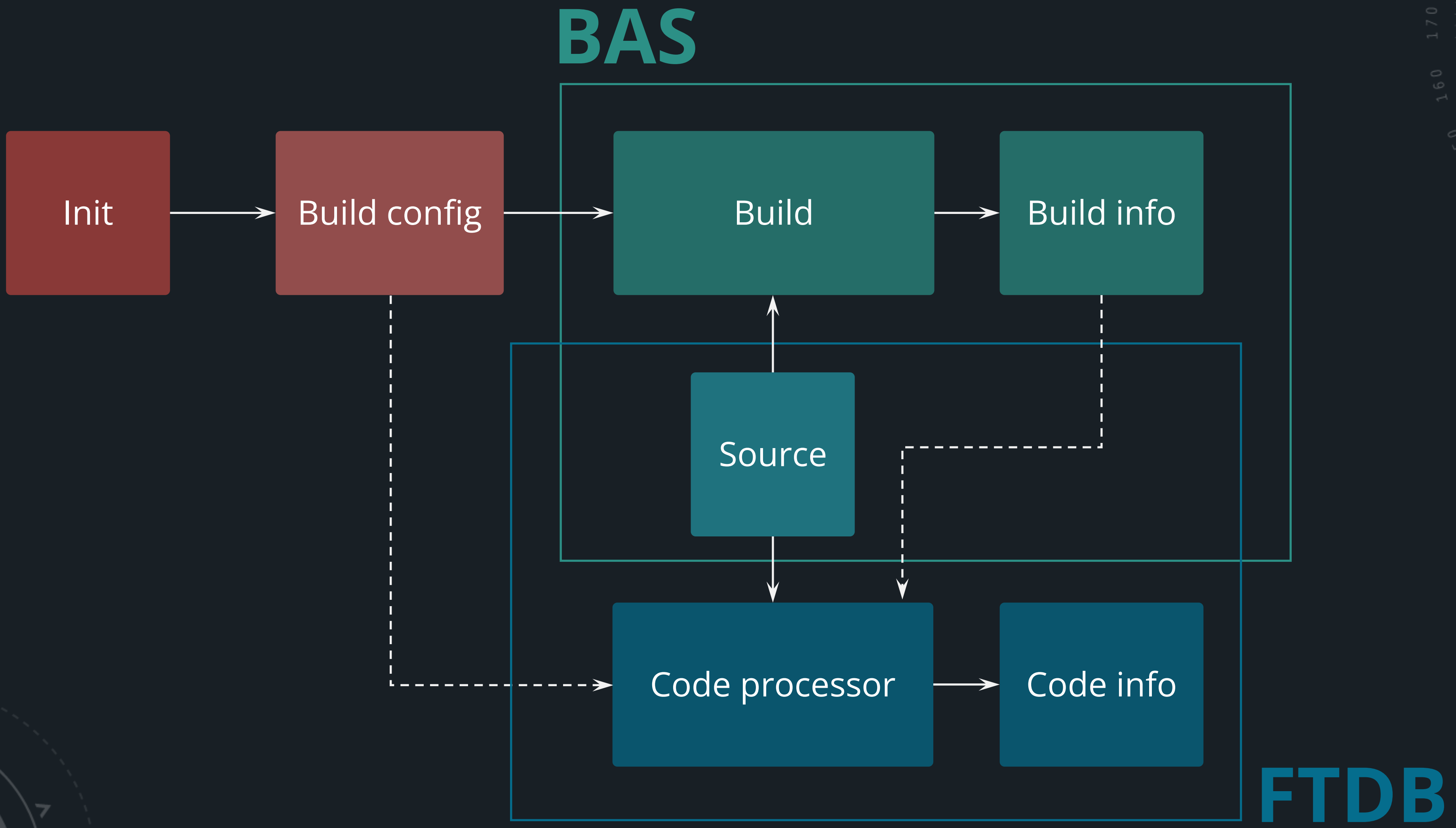
Main focus on usage in the Developer Productivity Engineering (DPE)

CAS: System overview



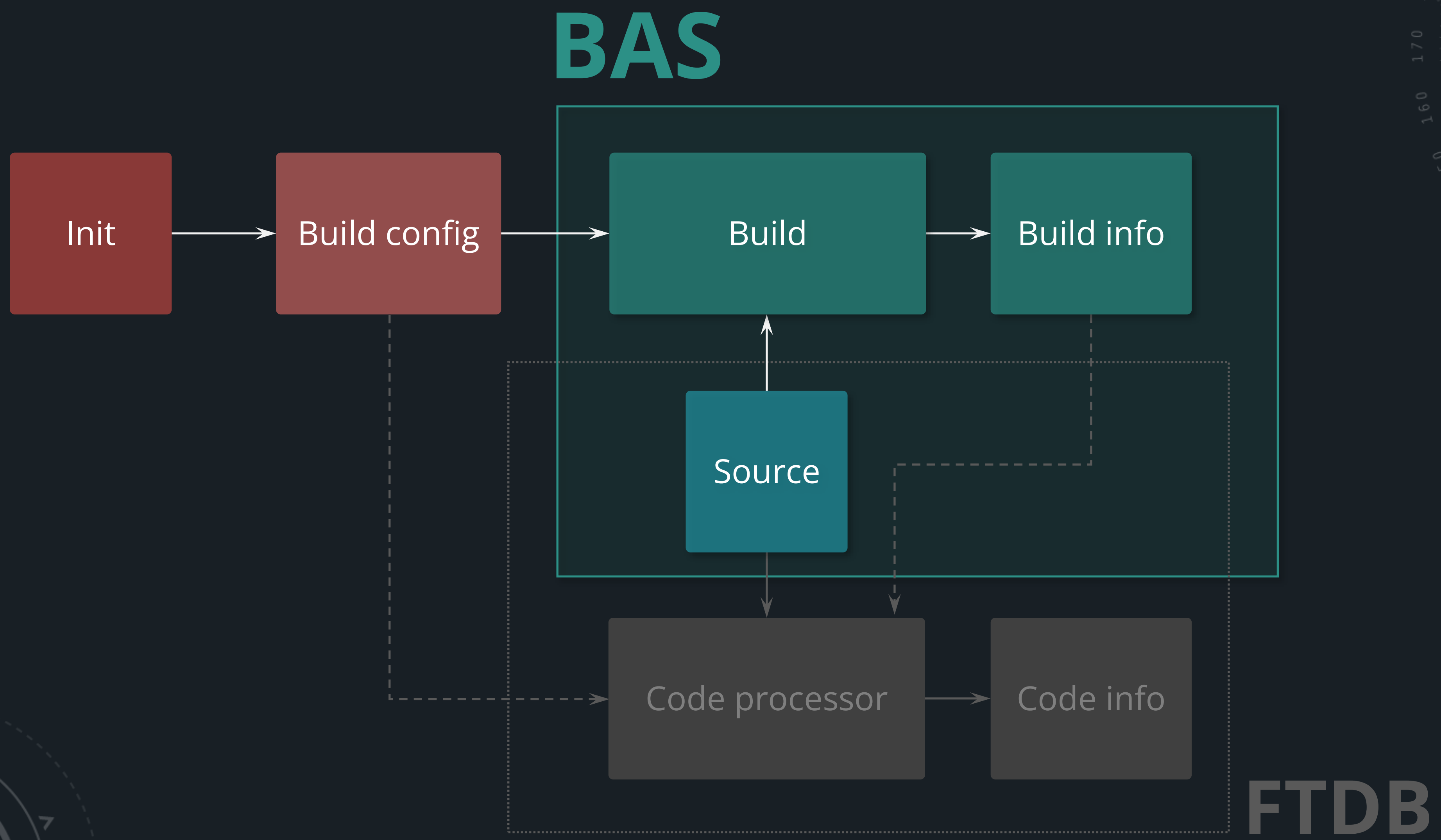
SAMSUNG

CAS: System overview



SAMSUNG

CAS: System overview



SAMSUNG

BAS: Build Awareness Service

SAMSUNG

BUILD AWARENESS SERVICES

Problem introduction

Extremely complex nature of software

System and framework layers of a modern mobile product

Many software layers

Bootloader, Linux Kernel, Modem, Native framework
Deveopers cannot handle S/W stack compexity without supporting tools

SAMSUNG

Problem introduction

Current tools that assist with S/W manipulation don't focus on a specific configuration

One S/W stack, many distinct products

S/W is highly configurable. Many flavors of one S/W stack.

One specific S/W configuration:

- ~44k source files
- ~10M lines of code
- ~4k linked binaries (libraries and executables)

A single source file can be compiled in many different ways

.c

```
#if defined MAX_RTT && (USE_STATIC==1 && not defined USE_EXCEPTIONS) || CLANG_VER>5
<...>
#elif USE_STATIC==1 && defined CONFIG_PRM && CLANG_VER>5
<...>
#elif USE_STATIC!=1 || !defined CLANG_VER
<...>
#else
<...>
#endif
```

N distinct preprocessor conditions => 2^N different S/W flavors

BUILD AWARENESS SERVICES

Build Information

Highly underestimated process in S/W engineering

S/W transformation and execution

Enormous amount of information extracted from the build process

S/W STACK

BUILD PROCESS [START]

Configuration

detailed configuration information (used source files, compilation switches, definitions etc.)

Structure

high level organization of S/W structure

Dependencies

dependencies between S/W components

Tools

build tools and resource usage

Commands

detailed build commands that help in problem solving

BUILD PROCESS [END]

RUNNING IMAGE

Project purpose: Extract the build information and make it readily available to developers

SAMSUNG

Approach to build tracing

Low-level tracing of the build process

Intercepting low-level OS primitives in the least intrusive manner possible

Process fully transparent to the ongoing build

WHAT WE NEED TO TRACE?

1. All opened files
during the build
(open syscalls)

2. All executed commands
during the build
(execve syscalls)

3. Auxiliary syscalls
(like pipes, rename etc.)

HOW TO TRACE THE BUILD?

~~LD_PRELOAD~~

~~ptrace~~

Linux kernel tracing infrastructure

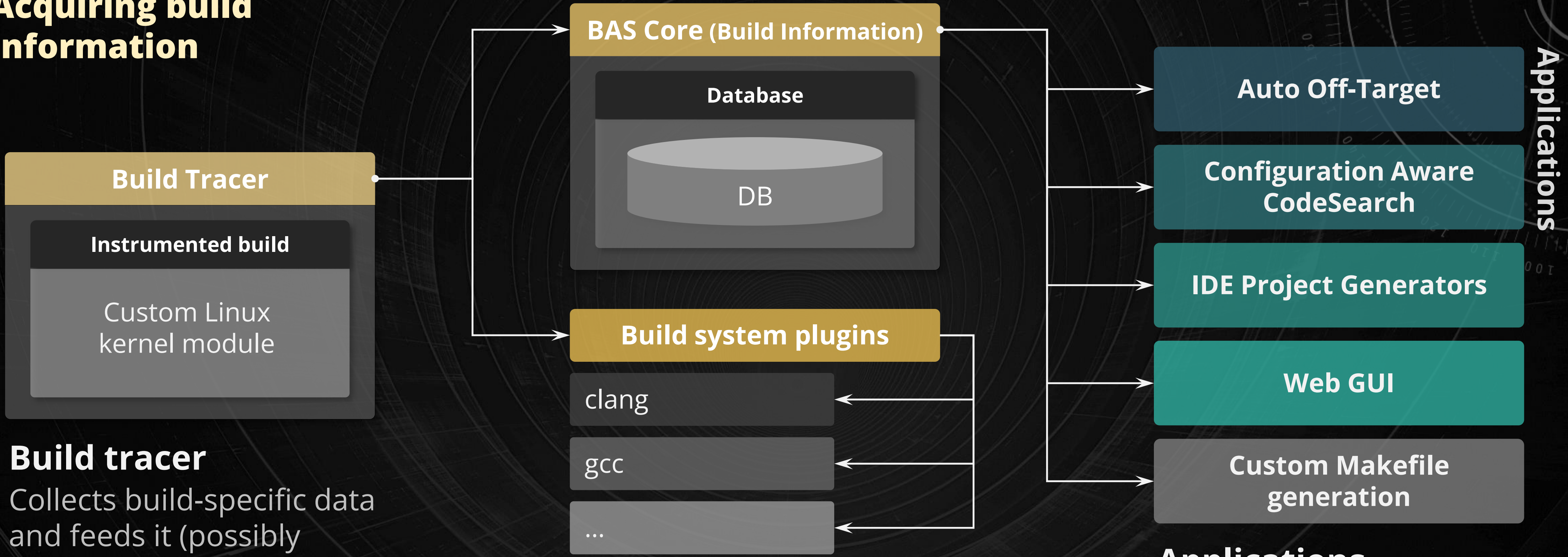
Full AOSP x86_64 build average overhead **~5%**

BUILD AWARENESS SERVICES

BAS Architecture

Build Awareness Service high-level overview

Acquiring build information



Build tracer

Collects build-specific data and feeds it (possibly post-processed) to the database for later retrieval

Core engine (service)

Service that reads data from database and serves it to connected clients

Applications

Specific applications/tools that use build information to operate efficiently

SAMSUNG

BAS: Examples

SAMSUNG

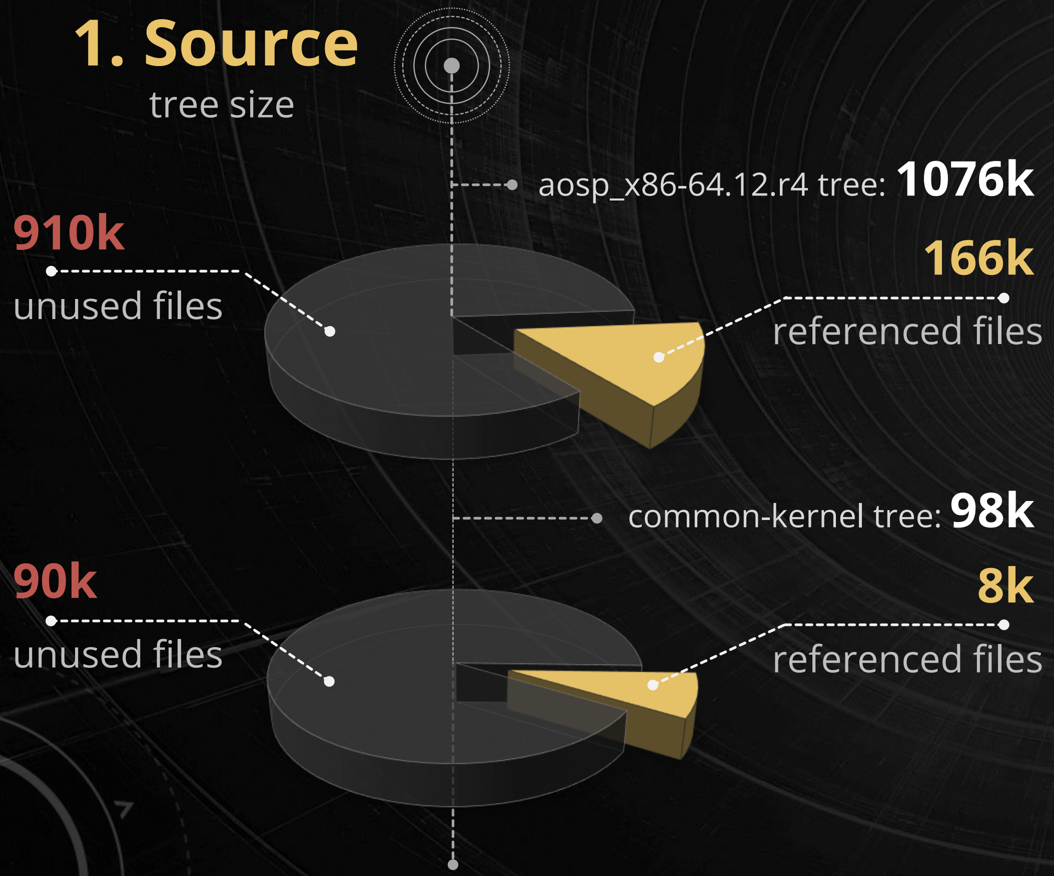
BAS EXAMPLES

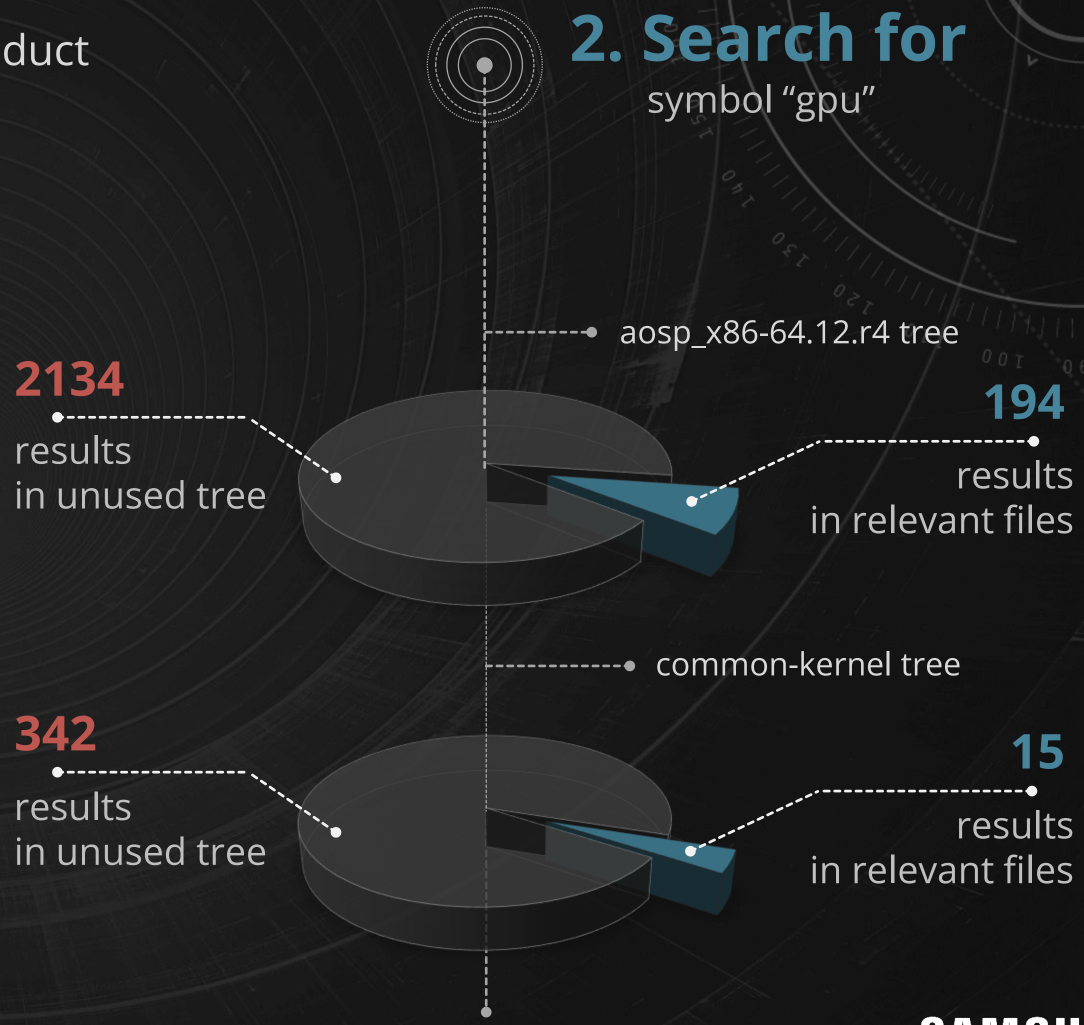
Code search improvements

Detailed code search only in relevant files

Acquiring the relevant file set

Getting a detailed subset of files used to create the final product
Getting a set of file dependencies for a specific module





SAMSUNG

BAS EXAMPLES

IDE indexing improvements

Setup the IDE to index the source code perfectly

Generating required project metadata

IDE needs exact compilation switches to index the sources properly (compilation database)
CAS can generate custom project description files for an IDE, e.g., Eclipse CDT

Original Linux kernel source tree	Linux linked kernel executable with relevant files and custom project description files
29,473 sources, **25,851** headers in **638** sec;	**2,578** sources, **5,094** headers in **105** sec;
8,279,064 declarations;	**1,274,685** declarations;
38,329,808 references;	**7,354,473** references;
854 unresolved inclusions;	**0** unresolved inclusions;
207,252 syntax errors;	**3,812** syntax errors;
1,645,350 unresolved names (**3.4%***)	**909** unresolved names (**0.011%***)

* Percentage of unresolved symbols in index

SAMSUNG

BAS EXAMPLES

Process visualization

Detailed presentation of the process tree executed during the build

Web-based process tree browser

Walking through entire process hierarchy of the build. Possibility to search for specific commands.

Makefile

```
all:
        @gcc -Wall -c myfile.c -o myfile.o
        @g++ -o myapp myfile.o
```

Process tree viewer

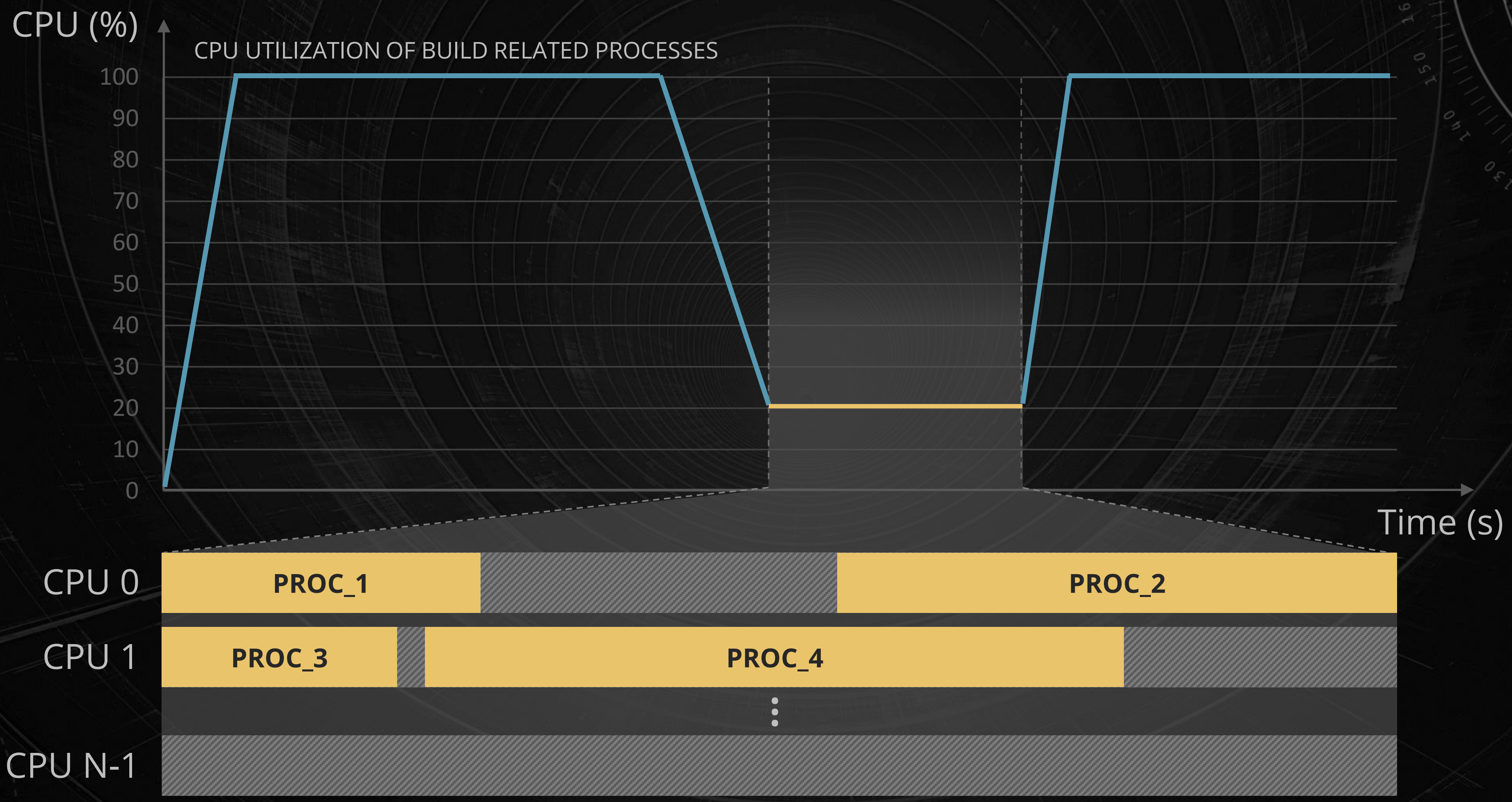
```
[make]
  └── [gcc -Wall -c myfile.c -o myfile.o]
        ├── [cc1 -quiet -imultiarch x86_64-linux-gnu myfile.c -quiet -dumpbase myfile.c (...) -o /tmp/ccYEYDEJ.s]
        └── [as --64 -o myfile.o /tmp/ccYEYDEJ.s]
  └── [g++ -o myapp myfile.o]
        └── [collect2 (...) myfile.o -lstdc++ -lm -lgcc_s -lgcc -lc -lgcc_s -lgcc (...)]
              └── [ld (...) myfile.o -lstdc++ -lm -lgcc_s -lgcc -lc -lgcc_s -lgcc (...)]
```

BAS EXAMPLES

Execution time measurements

Information about CPU scheduling of build related processes

Investigation of build system serialization bottlenecks



SAMSUNG

BAS EXAMPLES

Build dependency analysis

Detailed presentation of file dependencies between low-level product build components

Web-based dependency tree browser

Walking through the file dependencies of the build products. Easily accessible reverse dependency mappings.

Dependency graph viewer

```
[frameworks/av/media/mediaserver/mediaserver]
  frameworks/av/media/mediaserver/main_mediaserver.cpp
  frameworks/av/include/media/MediaExtractorPluginApi.h
  [frameworks/av/media/libmediaplayerservice/libmediaplayerservice.so]
    frameworks/av/media/libmediaplayerservice/ActivityManager.cpp
    frameworks/av/media/libmediaplayerservice/ActivityManager.h
    [frameworks/av/drm/libdrmframework/libdrmframework.so]
      frameworks/av/drm/libdrmframework/DrmManagerClient.cpp
      frameworks/av/drm/libdrmframework/include/DrmManagerClientImpl.h
      [system/core/libutils/libutils.so]
        system/core/libutils/Threads.cpp
        system/core/libutils/Unicode.cpp
        [bionic/libc/libc.so]
  [system/core/libutils/libutils.so]
  [bionic/libc/libc.so]
```

BAS EXAMPLES

Custom build script generation

Automatic preparation of build scripts for customizable, partial tree product rebuilds

Re-using original build information to enhance productivity

Selective clang static analysis; partial/incremental builds of selected functionality; hooking into compilation process

Makefile

```
.PHONY: all
.PHONY: cmd_0
.PHONY: cmd_1
(...)

all: cmd_0 cmd_1 ...
        @echo Done!

cmd_0:
        @echo "CMD 0"
        @(cd ${ROOT_DIR}/android && $(ADDITIONAL_OPTS_PREFIX)prebuilts/jdk/jdk11/linux-x86/bin/javac -J-Xmx2048M
-J-XX:OnError=cat\ hs_err_pid%p.log -J-XX:CICompilerCount=6 -J-XX:+UseDynamicNumberOfGCThreads -J-
XX:+TieredCompilation -J-XX:TieredStopAtLevel=1 -Xmaxerrs 9999999 -encoding UTF-8 -sourcepath -g -
XDskipDuplicateBridges=true -XDstringConcat=inline -proc:none -Xlint:-dep-ann --
system=out/soong/.intermediates/build/soong/java/core-libraries/core-public-stubs-system-
modules/android_common/system -classpath
out/soong/.intermediates/frameworks/base/hwbinder.stubs/android_common/turbine-combined/hwbinder.stubs.jar ...
$(ADDITIONAL_OPTS_POSTFIX))

(...)
```

Hooking plug-in invocation into the command

SAMSUNG

BAS command line tool

Command line utility for getting information from the BAS database

Querying the build database by using a dedicated Linux tool

Examples of commands that support build productivity tools

.bash — Getting **the list of all used original source files** during the build

```bash

cas ref_files --filter=[exists=FILE,source_root=true]or[exists=DIR,source_root=true]

```

.bash — Generating **project description files for Eclipse CDT** for Linux kernel modules

```bash

cas linked_modules --filter=[path=*/vmlinux,type=wc]or[path=*.ko,type=wc] deps_for\
--ide=eclipse --skip-pattern=".*vmlinux$" --skip-objects --skip-linked

```

.bash — Generating **custom Makefile with all java compiler invocations**

```bash

cas binaries --filter=[bin=*/javac,type=wc] --commands --generate --makefile --all

```

BAS EXAMPLES

BAS web-API access

Remote access to the BAS database through the web-based protocols

Querying the build database by using a well defined web API

Examples of web queries that support build productivity tools

.bash — Getting **the list of all used original source files** during the build

```bash
cas ref_files --filter=[exists=FILE,source_root=true]or[exists=DIR,source_root=true]
```

https://bas/ref_files?filter=[exists=1,source_root=1]or[exists=2,source_root=1]

.bash — Generating **project description files for Eclipse CDT** for Linux kernel modules

```bash
cas linked_modules --filter=[path=*/vmlinux,type=wc]or[path=*.ko,type=wc] deps_for\
--ide=eclipse --skip-pattern=".*vmlinux$" --skip-objects --skip-linked
```

https://bas/linked_modules?filter=[path=*/vmlinux,type=wc]or[path=*.ko,type=wc]&deps_for&ide=eclipse&skip-pattern=".*vmlinux$"&skip-objects=true&skip-linked=true

.bash — Generating **custom Makefile with all java compiler invocations**

```bash
cas binaries --filter=[bin=*/javac,type=wc] --commands --generate --makefile --all
```

https://bas/binaries?filter=[*javac,type=wc]&commands=true&generate=true&makefile=true&all=true

SAMSUNG

BAS EXAMPLES

BAS Python API

Getting information from the BAS database using dedicated Python API

Querying the build database by writing custom Python programs

For all header files used during the build of the Linux kernel executable
get a list of compiled files that included each of these headers

.py

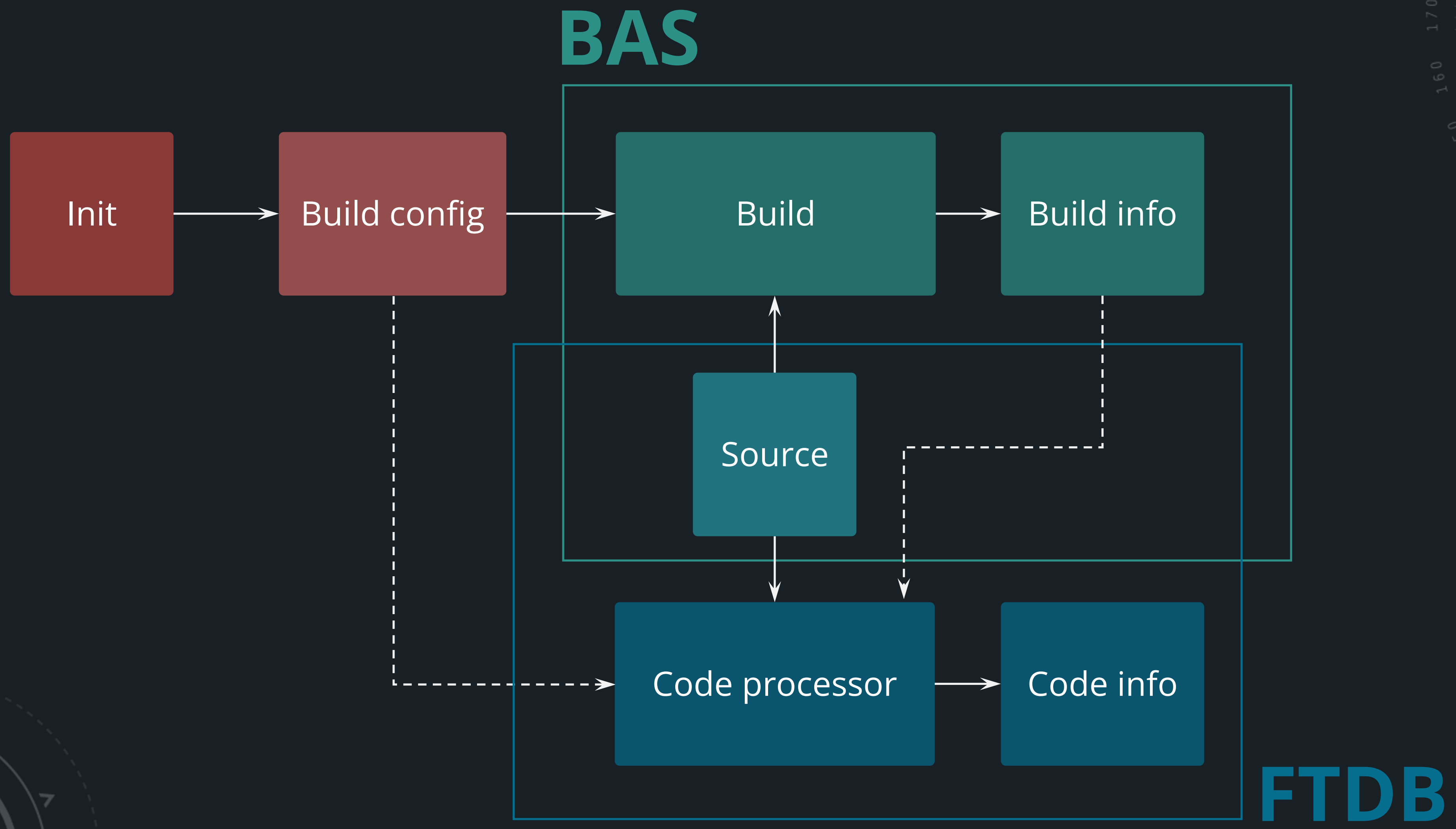
```
import libetrace

nfsdb = libetrace.nfsdb()
nfsdb.load(".nfsdb.img",quiet=True)
nfsdb.load_deps(".nfsdb.deps.img",quiet=True)

hmap = {}

vmlinux = [e[0].path for e in nfsdb.linked_modules()
    if e[0].path.endswith("vmlinux")][0]
for d in nfsdb.mdeps(vmlinux):
    cdeps = list()
    if d.is_compiled():
        ce = d.opaque
        while ce:
            cdeps+=[u.path for u in ce.opens_with_children]
            ce = ce.next
        for fn in cdeps:
            if fn.startswith(nfsdb.source_root) and fn.endswith(".h"):
                if fn in hmap:
                    hmap[fn].add(d.opaque.compilation_info.files[0].path)
                else:
                    hmap[fn] = {d.opaque.compilation_info.files[0].path}
```

Linux kernel header file map

Linux kernel header file	List of sources that use this header
include/linux/pagewalk.h	mm/mincore.c
	mm/madvise.c
	mm/mprotect.c
	fs/proc/task_mmu.c
(…)	(…)

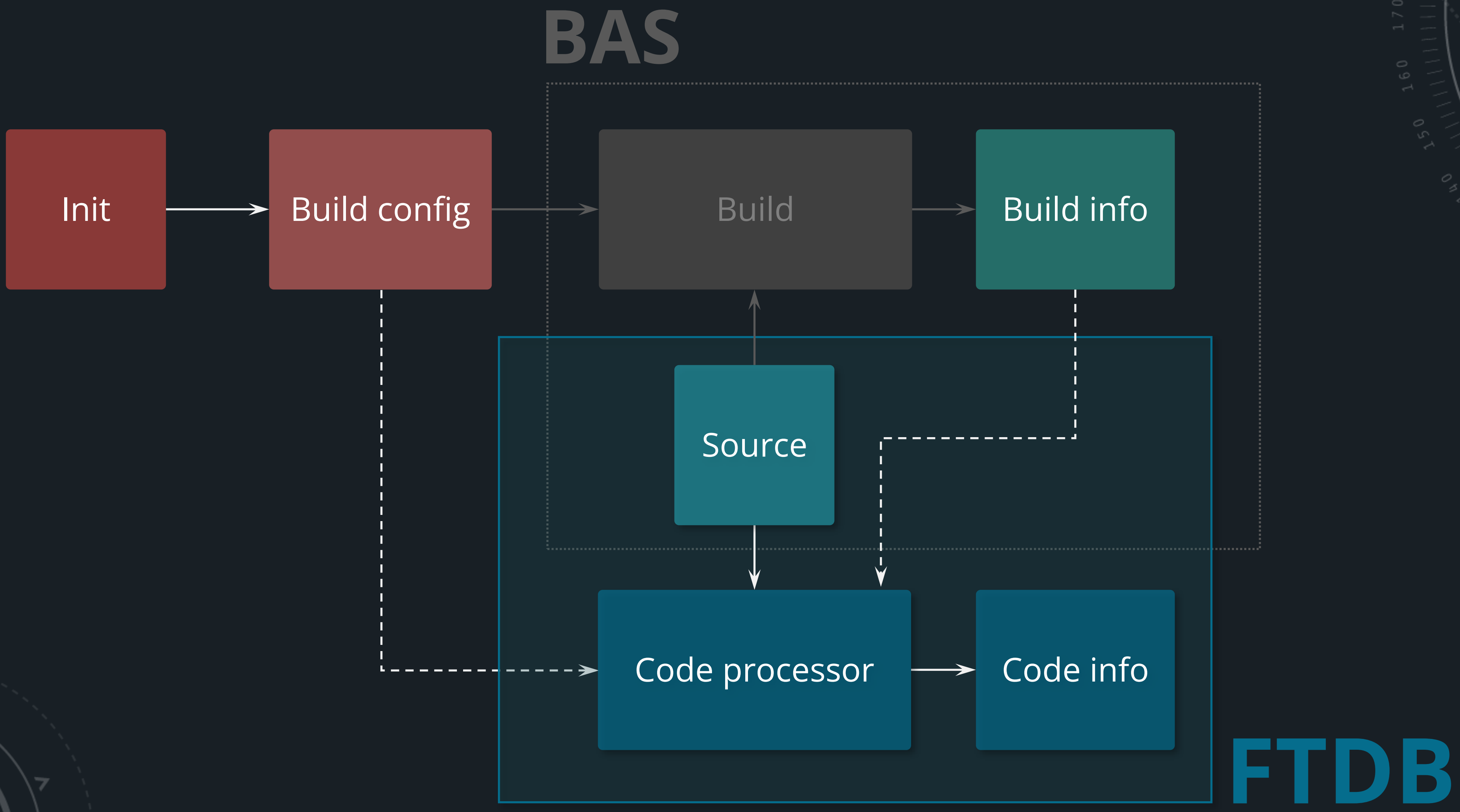
SAMSUNG

CAS: System overview



SAMSUNG

CAS: System overview



FTDB: Function Type DB

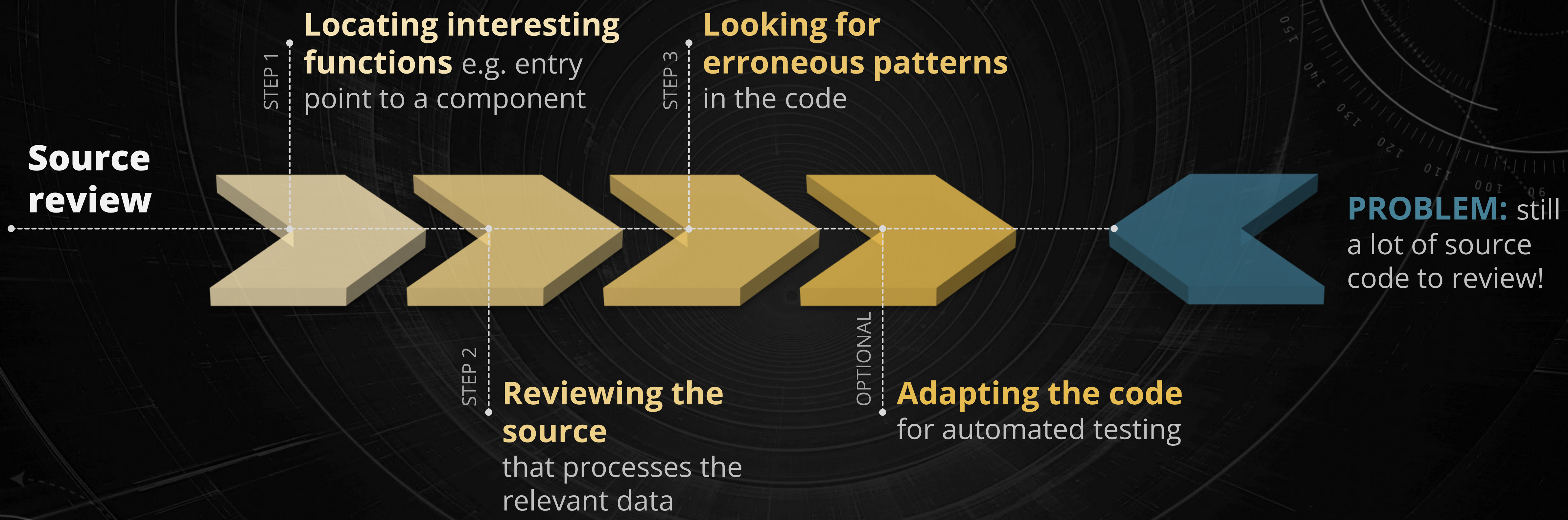
SAMSUNG

FUNCTION TYPE DB

Problem introduction

Automating source code review process

Approach to source review of low-level OS components



Need a way to automate and make the code review process more scalable

SAMSUNG

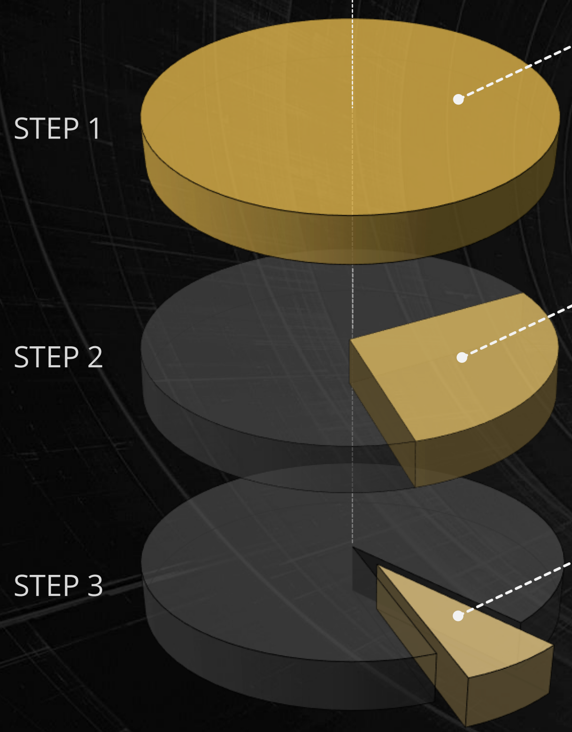
FUNCTION TYPE DB

Code review automation

Writing tools that operate on parsed representation of source code

Automating software engineers workflow

Source review



Source tree

Entire source tree of the product

Used sources

Relevant source files for a given configuration

Selected source features

parsed representation in **JSON**

Experienced Software Engineer

Transforming know-how into automation scripts

Vulnerable pattern recognition

Data filtering

Visualization and presentation

Need for a parsed source representation in a simple format (JSON)

SAMSUNG

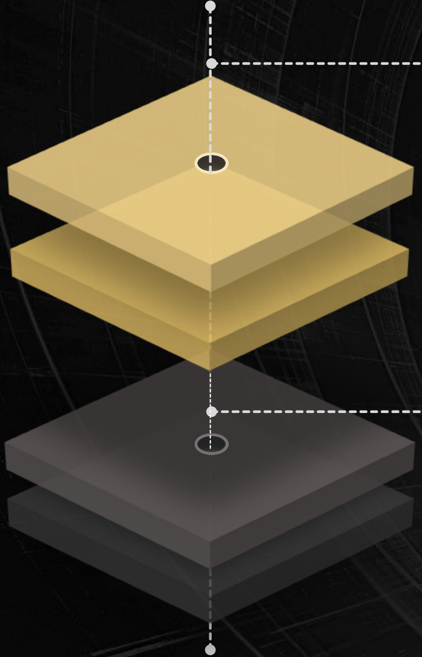
FUNCTION TYPE DB

Parsing the source code

Transform the source into easily accessible representation

Source code parsing

Possibilities

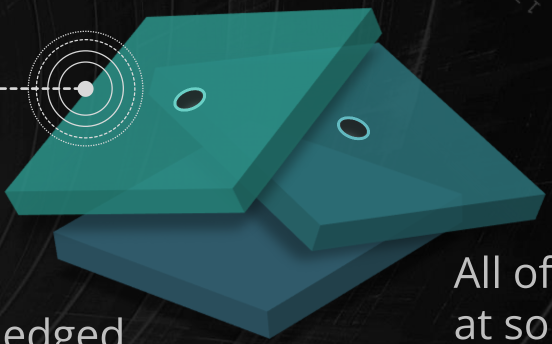


Regular expressions

Custom parsers

Outcome

C++ extremely difficult to parse (grammar heavily relies on context)



Fully fledged preprocessor is needed

All of this breaks at some point

Solution

Use real compiler (clang) to parse the source code

Clang **parses C/C++/Objective-C**

Easy to hook into the compiler internals

SAMSUNG

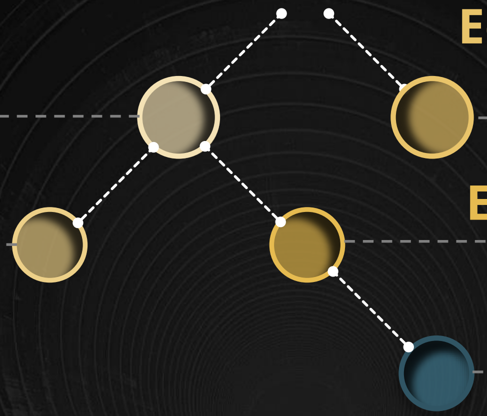
FUNCTION TYPE DB

Abstract Syntax Tree (AST)

Easily accessible representation of the original source code

Main feature

Abstract Syntax Tree



Tree like structure

C++ class for each tree node

Equivalent representation of source file

Easy to traverse

Can be used to regenerate source code if needed

.C

```c
struct A {
    int q;
};
int fun(struct A* pA) {
    return (int)pA;
}
int main(void) {
    struct A x;
    return fun(&x);
};
```

```
CompoundStmt 0x2fba510
`-ReturnStmt 0x2fba4f8
  `-CStleCastExpr 0x2fba4d0 'int' <PointerToIntegral>
    `-ImplicitCastExpr 0x2fba4b8 'struct A *' <LValueToRValue>
      `-DeclRefExpr 0x2fba478 'struct A *' lvalue ParmVar 0x2fba290 'pA' 'Struct A *'
CompoundStmt 0x2fba878
|-DeclStmt 0x2fba738
| `-VarDecl 0x2fba6d8 used x 'struct A':'struct A'
`-ReturnStmt 0x2fba860
  `-CallExpr 0x2fba830 'int'
    |-ImplicitCastExpr 0x2fba818 'int (*)(struct A *)' <FunctionToPointerDecay>
    | `-DeclRefExpr 0x2fba750 'int (struct A *)' Function 0x2fba398 'fun' 'int (struct A *)'
    `-UnaryOperator 0x2fba7a0 'struct A *' prefix '&'
      `-DeclRefExpr 0x2fba778 'struct A':'struct A' lvalue Var 0x2fba6d8 'x''struct A':'struct A'
```

FUNCTION TYPE DB

Function information in JSON

Intermediate format for source code features

Code extraction

Extracting various function attributes, e.g.: name, source attributes, argument information, call information, referenced types, body, source literals, argument taints, selected expressions, referenced variables, etc.

.c

```c
static long v4l2_ioctl(
        struct file *filp, unsigned int cmd, unsigned long arg)
{
        struct video_device *vdev = video_devdata(filp);
        int ret = -ENODEV;

        if (vdev->fops->unlocked_ioctl) {
                if (video_is_registered(vdev))
                        ret = vdev->fops->unlocked_ioctl(filp, cmd, arg);
        } else
                ret = -ENOTTY;

        return ret;
}
```

.json


```json
{
        "name": "v4l2_ioctl",
        "id": 3,
        "fid": 0,
        "fids": [ 0 ],
        "nargs": 3,
        "variadic": false,
        "linkage": "internal",
        "attributes": [ ],
        "body": "static long v4l2_ioctl...",
        "location": "drivers/media/v4l2-core/v4l2-dev.c:353",
        "literals":{
                "integer": [ 19, 25 ],
        },
        "calls": [ 2, 1 ],
        "funrefs": [  1, 2 ],
        "refs": [  12, 1, 4, 10, 13, 14, 16 ],
        "types": [ 9, 10, 1, 4 ],
        "locals": [
                {
                        "id": 0,
                        "name": "filp",
                        "parm": true,
                        "type": 10,
                        "static": false,
                        "used": true,
                        "location":
                                "drivers/media/v4l2-core/v4l2-dev.c:37:"
                },
        ], }
```


SAMSUNG

FUNCTION TYPE DB

Function information in JSON

Intermediate format for source code features

Code extraction

Extracting various function attributes, e.g.: name, source attributes, argument information, call information, referenced types, body, source literals, argument taints, selected expressions, referenced variables, etc.

.c

```c

static long v4l2_ioctl (
        struct file *filp, unsigned int cmd, unsigned long arg)
{
        struct video_device *vdev = video_devdata(filp);
        int ret = -ENODEV;

        if (vdev->fops->unlocked_ioctl) {
                if (video_is_registered(vdev))
                        ret = vdev->fops->unlocked_ioctl(filp, cmd, arg);
        } else
                ret = -ENOTTY;

        return ret;
}

```

.json


```json
{
        "name": "v4l2_ioctl",
        "id": 3,
        "fid": 0,
        "fids": [ 0 ],
        "nargs": 3,
        "variadic": false,
        "linkage": "internal",
        "attributes": [ ],
        "body": "static long v4l2_ioctl...",
        "location": "drivers/media/v4l2-core/v4l2-dev.c:353",
        "literals":{
                "integer": [ 19, 25 ],
        },
        "calls": [ 2, 1 ],
        "funrefs": [  1, 2 ],
        "refs": [  12, 1, 4, 10, 13, 14, 16 ],
        "types": [ 9, 10, 1, 4 ],
        "locals": [
                {
                        "id": 0,
                        "name": "filp",
                        "parm": true,
                        "type": 10,
                        "static": false,
                        "used": true,
                        "location":
                                "drivers/media/v4l2-core/v4l2-dev.c:37:"
                },
        ], }
```

FUNCTION TYPE DB

Function information in JSON

Intermediate format for source code features

Code extraction

Extracting various function attributes, e.g.: name, source attributes, argument information, call information, referenced types, body, source literals, argument taints, selected expressions, referenced variables, etc.

.c

```c

static long v4l2_ioctl(
        struct file *filp, unsigned int cmd, unsigned long arg)
{
        struct video_device *vdev = video_devdata(filp);
        int ret = -ENODEV;

        if (vdev->fops->unlocked_ioctl) {
                if (video_is_registered(vdev))
                        ret = vdev->fops->unlocked_ioctl(filp, cmd, arg);
        } else
                ret = -ENOTTY;

        return ret;
}
```

.json


```json
{
        "name": "v4l2_ioctl",
        "id": 3,
        "fid": 0,
        "fids": [ 0 ],
        "nargs": 3,
        "variadic": false,
        "linkage": "internal",
        "attributes": [ ],
        "body": "static long v4l2_ioctl...",
        "location": "drivers/media/v4l2-core/v4l2-dev.c:353",
        "literals":{
                "integer": [ 19, 25 ],
        },
        "calls": [ 2, 1 ],
        "funrefs": [ 1, 2 ],
        "refs": [ 12, 1, 4, 10, 13, 14, 16 ],
        "types": [ 9, 10, 1, 4 ],
        "locals": [
                {
                        "id": 0,
                        "name": "filp",
                        "parm": true,
                        "type": 10,
                        "static": false,
                        "used": true,
                        "location":
                                "drivers/media/v4l2-core/v4l2-dev.c:37:"
                },
        ], }
```

FUNCTION TYPE DB

Function information in JSON

Intermediate format for source code features

Code extraction

Extracting various function attributes, e.g.: name, source attributes, argument information, call information, referenced types, body, source literals, argument taints, selected expressions, referenced variables, etc.

.c

```c

static long v4l2_ioctl(
        struct file *filp, unsigned int cmd, unsigned long arg)
{
        struct video_device *vdev = video_devdata(filp);
        int ret = -ENODEV;

        if (vdev->fops->unlocked_ioctl) {
                if ( video_is_registered(vdev) )
                        ret = vdev->fops->unlocked_ioctl(filp, cmd, arg);
        } else
                ret = -ENOTTY;

        return ret;
}
```

.json


```json
{
        "name": "v4l2_ioctl",
        "id": 3,
        "fid": 0,
        "fids": [ 0 ],
        "nargs": 3,
        "variadic": false,
        "linkage": "internal",
        "attributes": [ ],
        "body": "static long v4l2_ioctl...",
        "location": "drivers/media/v4l2-core/v4l2-dev.c:353",
        "literals":{
                "integer": [ 19, 25 ],
        },
        "calls": [ 2, 1 ],
        "funrefs": [ 1, 2 ],
        "refs": [ 12, 1, 4, 10, 13, 14, 16 ],
        "types": [ 9, 10, 1, 4 ],
        "locals": [
                {
                        "id": 0,
                        "name": "filp",
                        "parm": true,
                        "type": 10,
                        "static": false,
                        "used": true,
                        "location":
                                "drivers/media/v4l2-core/v4l2-dev.c:37:"
                },
        ], }
```


SAMSUNG

FUNCTION TYPE DB

Function information in JSON

Intermediate format for source code features

Code extraction

Extracting various function attributes, e.g.: name, source attributes, argument information, call information, referenced types, body, source literals, argument taints, selected expressions, referenced variables, etc.

.c

```c

static long v4l2_ioctl(
      struct file *filp, unsigned int cmd, unsigned long arg)
{
      struct video_device *vdev = video_devdata(filp);
      int ret = -ENODEV;

      if (vdev->fops->unlocked_ioctl) {
            if (video_is_registered(vdev))
                  ret = vdev->fops->unlocked_ioctl(filp, cmd, arg);
      } else
            ret = -ENOTTY;

      return ret;
}
```

.json


```json
{
      "name": "v4l2_ioctl",
      "id": 3,
      "fid": 0,
      "fids": [ 0 ],
      "nargs": 3,
      "variadic": false,
      "linkage": "internal",
      "attributes": [ ],
      "body": "static long v4l2_ioctl...",
      "location": "drivers/media/v4l2-core/v4l2-dev.c:353",
      "literals":{
            "integer": [ 19, 25 ],
      },
      "calls": [ 2, 1 ],
      "funrefs": [  1, 2 ],
      "refs": [  12, 1, 4, 10, 13, 14, 16 ],
      "types": [ 9, 10, 1, 4 ],
      "locals": [
            {
                  "id": 0,
                  "name": "filp",
                  "parm": true,
                  "type": 10,
                  "static": false,
                  "used": true,
                  "location":
                        "drivers/media/v4l2-core/v4l2-dev.c:37:"
            },
      ], }
```


SAMSUNG

FTDB: Examples

SAMSUNG

FTDB EXAMPLES

FTDB Python API

Getting information from the FTDB database using dedicated Python API

Querying the code database by writing custom Python programs

Looking for 'memcpy' invocations with current function parameters passed to the 'memcpy' as its 3rd argument

libftdb.ex.py

```python

import libftdb

ftdb = libftdb.ftdb()
ftdb.load('db.img',quiet=True)

memcpy_ids = [f.id for f in list(ftdb.funcs) +
    list(ftdb.funcdecls) if f.name=='memcpy']

for f in ftdb.funcs:
    for i,cid in enumerate(f.calls):
        if cid in memcpy_ids:
            # 'memcpy' argument of index 2
            arg2 = f.derefs[f.call_info[i].args[2]]
            if arg2.offsetrefs[0].kindname=='parm':
                vid = arg2.offsetrefs[0].id
                print ("%s -> memcpy(...,%s)"%
                    (f.name,f.locals[vid].name))
            if arg2.offsetrefs[0].kindname=='member':
                ME = f.derefs[arg2.offsetrefs[0].id]
                if ME.offsetrefs[0].kindname=='parm':
                    vid = ME.offsetrefs[0].id
                    print ("%s -> memcpy(...,%s.$)"%
                        (f.name,f.locals[vid].name))
```

.h

```c

void *memcpy(void *dest, const void * src, size_t size);

```

.c

```c

static struct cfg80211_beacon_data *
cfg80211_beacon_dup(struct cfg80211_beacon_data *beacon) {

    (...)
    memcpy(pos, beacon->head, beacon->head_len);
    (...)
}
```

.bash

```bash
$ python3 libftdb.ex.py | wc -l

744
```

SAMSUNG

FTDB EXAMPLES

FTDB Python API

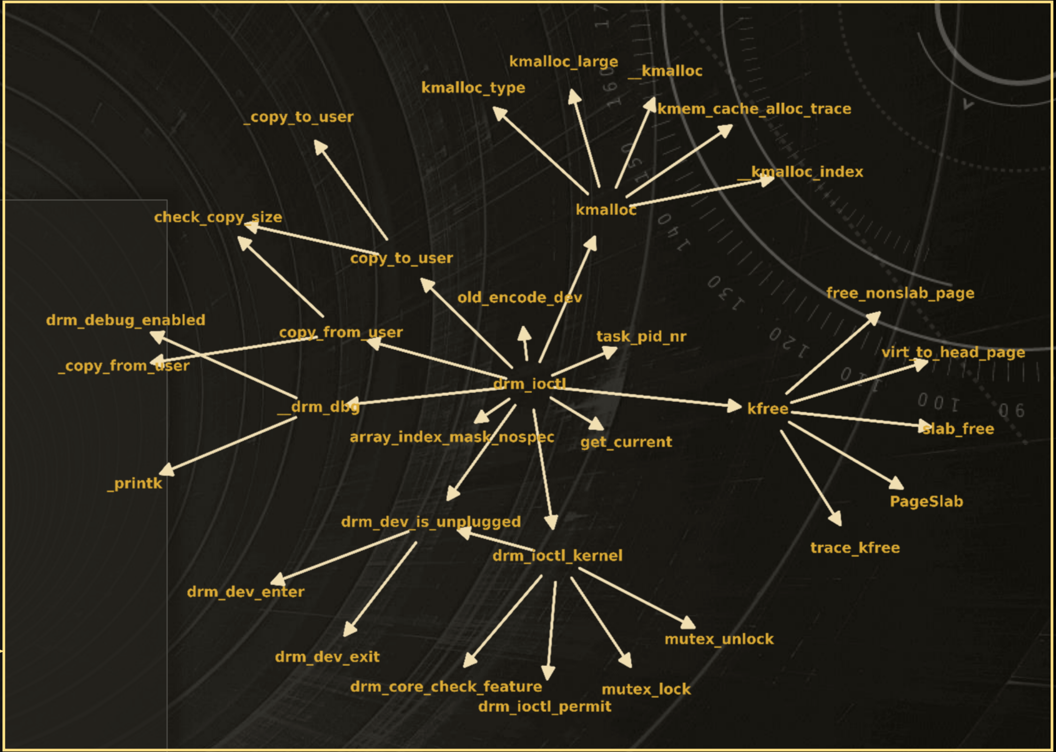
Getting information from the FTDB database using dedicated Python API

Querying the code database with Python programs

Generating call graph of a given function with provided depth

libftdb.graph.py

```python

import libftdb

ftdb = libftdb.ftdb()
ftdb.load('db.img',quiet=True) # change: ftdb image

func_name = 'drm_ioctl'
depth = 2

func_id = ftdb.funcs.entry_by_name(func_name)[0].id
func_frontier = {func_id}

call_graph = {}

while depth:
    new_frontier = set()
    for func_id in func_frontier:
        callees = {callee for callee in
ftdb.funcs.entry_by_id(func_id)['calls'] if ftdb.funcs.contains_id(callee)}
        call_graph[func_id] = set(callees)
        new_callees = {callee for callee in callees if callee not in
call_graph}
        new_frontier.update(new_callees)
    func_frontier = new_frontier
    depth -= 1
```



Call graph of function 'drm_ioctl' with a given depth = 2

SAMSUNG

FTDB EXAMPLES

Improved code review

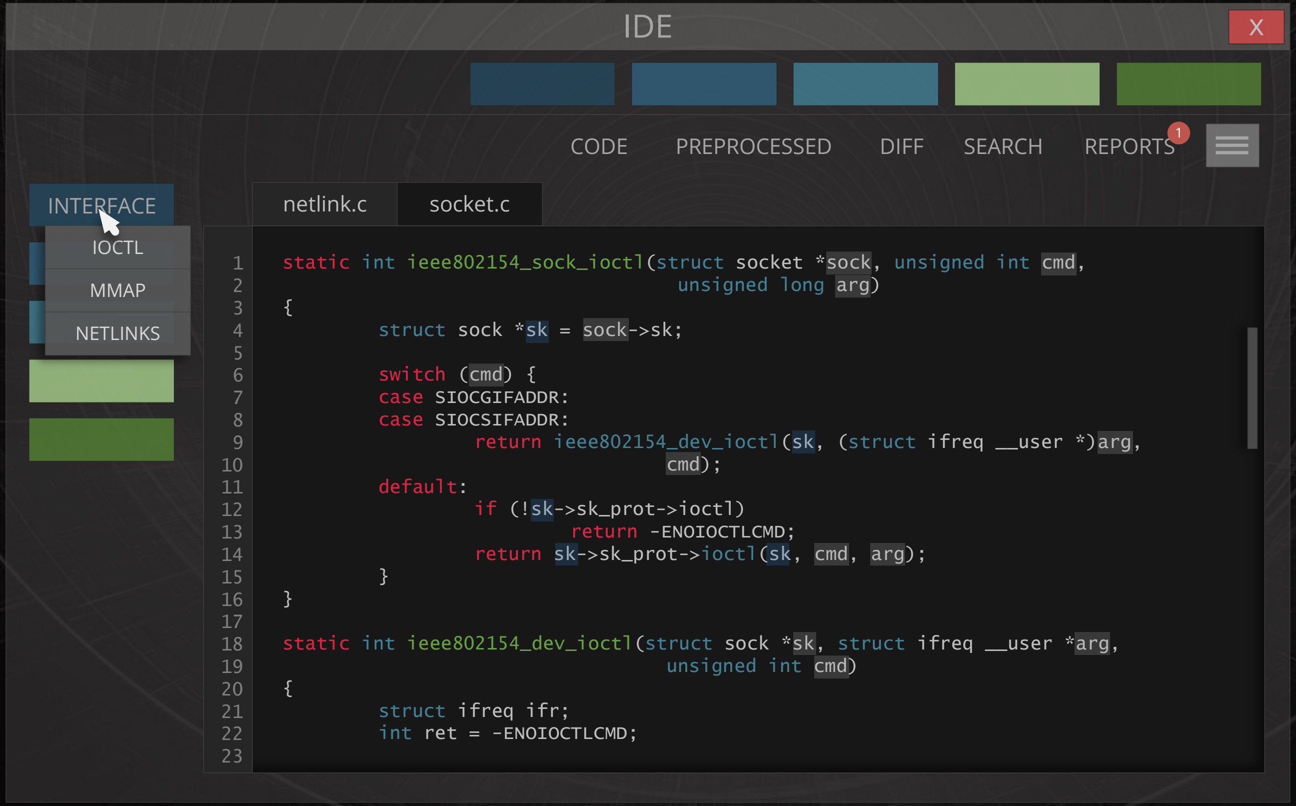
Automated features to support source code review process

Code review environment

Presentation layer to the end user (software engineer)

IDE like presentation (web rendering, IDE plugins)- original code, preprocessed code, code diffs, argument taints, etc.

Searching for potentially erroneous patterns with heuristics usage of dangerous functions on the call hierarchy, cyclomatic complexity, memory usage patterns



Sorting by the probability of error

Automatic extraction of interesting functions (e.g. entry points)

Search engine for custom source code queries

Code review support features possible based solely on the **FTDB**

SAMSUNG

Structure aware test data

Automatic preparation of a test data with a specific structure

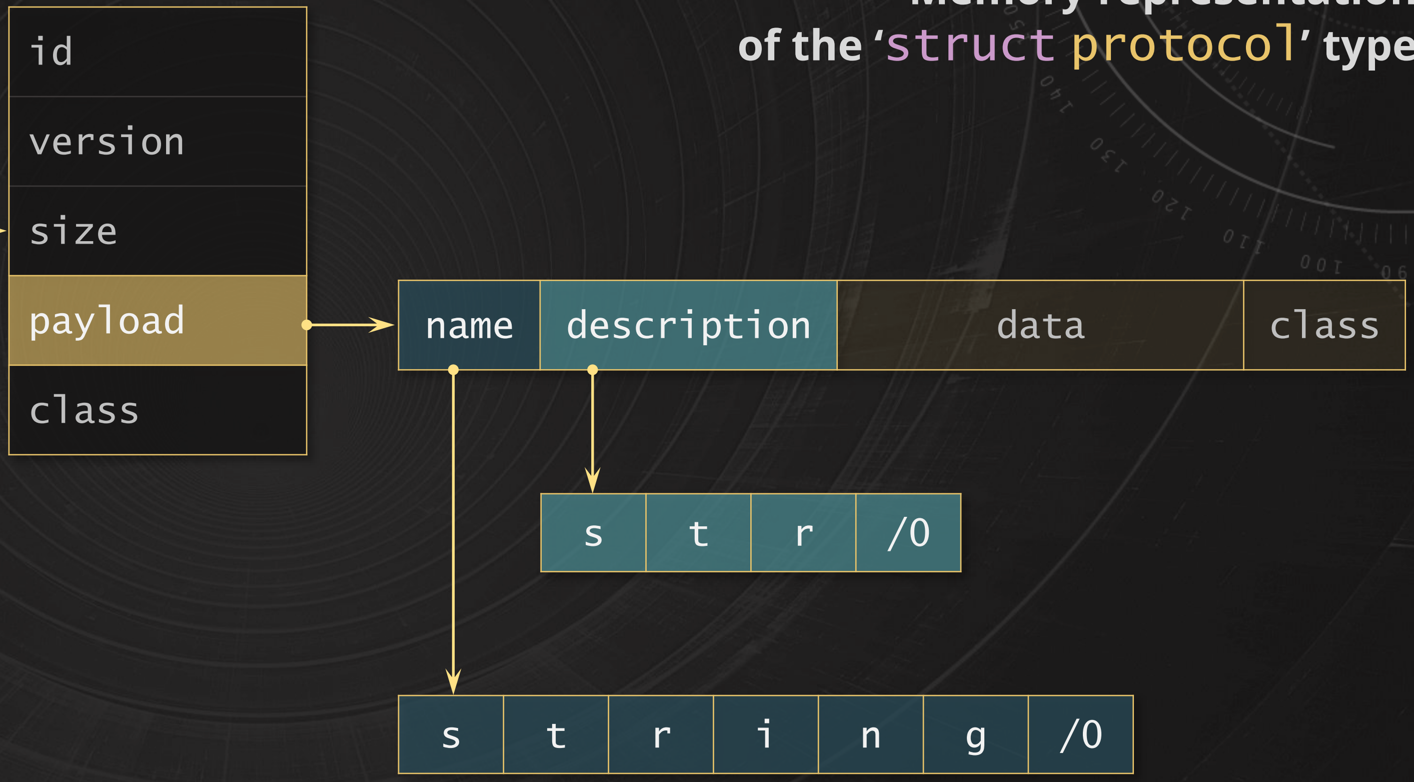
Testing packet processing application

Preparing data packets with proper structure for application to operate on

.h

```
struct protocol {
   int id;
   char version;
   unsigned long size;
   void* payload;
   enum proto_class class;
};

struct protocol_data {
   char* name;
   char* description;
   unsigned char data[40];
   enum proto_class class;
};
```

Memory representation of the ‘struct protocol’ type



AUTO OFF-TARGET

On-Target vs Off-Target

Introducing off-target testing methodology

Example: testing message parser of the WLAN driver

Extracting complex S/W component running on a custom hardware

Setup a test WLAN network

Send test messages over the air to the device

When the device crashes capture logs from the device (if any), start offline code analysis

Reboot the device and **repeat**

Difficult test setup for numerous complex embedded systems

On-Target vs Off-Target

Introducing off-target testing methodology

Example: testing message parser in the WLAN driver

Extracting complex S/W component running on a custom hardware

Testing Off-Target

Prepare the test harness for the parser function

Use the available toolchain: gdb, coverage, etc.

Fuzz the harness on a powerful development machine

BUG FOUND

Easier and faster testing in a native development environment

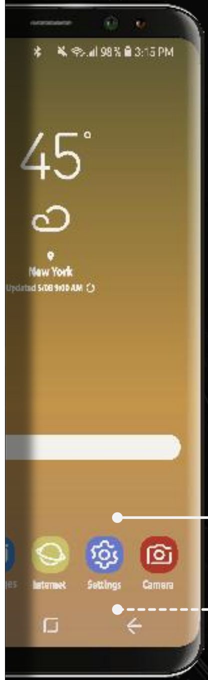
AUTO OFF-TARGET

Overview of the process

Introducing off-target testing methodology

Extracting parts of the source from one target to another

Extracting complex S/W component running on a custom hardware



On-Target

Step 1:
Extract from mobile

Interesting parts of source code with all required dependencies

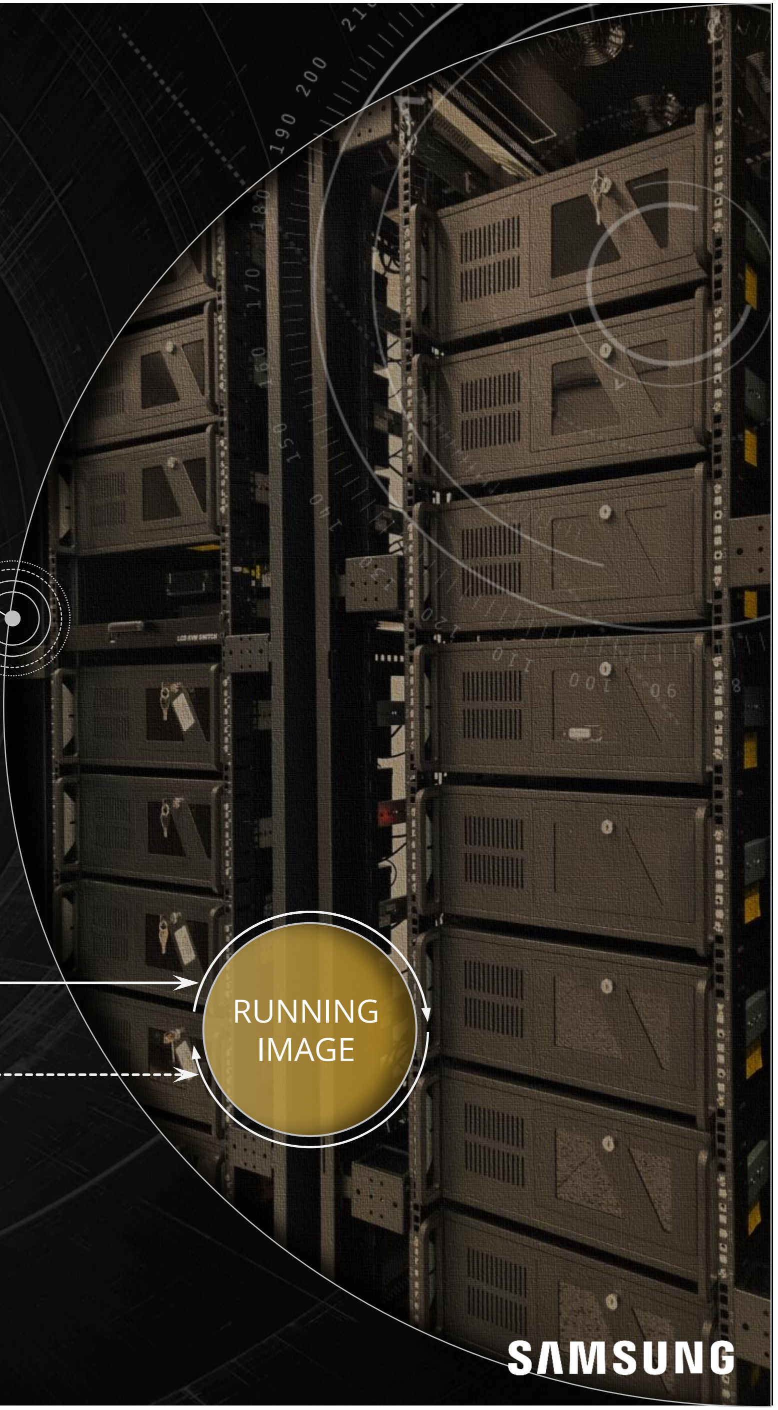
Off-Target

Step 2:
Compile and run on a server

Source code testing **MANY TIMES FASTER** on large servers than on a mobile device

TARGET SOURCE

RUNNING IMAGE



From mobile device to dev machine for most targets:
manual labor for several days

Mostly manual process until NOW!

SAMSUNG

AUTO OFF-TARGET

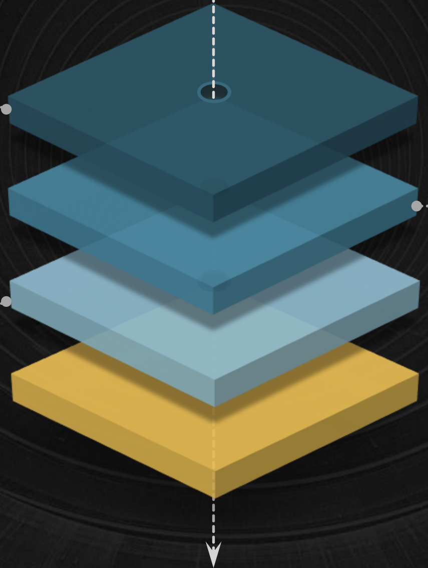
AoT: Automatic Off-Target

Automating the process of the OT creation

Overview

NOW: automation of laborious work of preparing OT
Project: https://github.com/samsung/auto_off_target
Paper: https://dl.acm.org/doi/10.1145/3551349.3556915
Talk: https://www.youtube.com/watch?v=Xzn_kmtW3_c

Automatic Off-Target



AoT can **automatically extract**

parts of the source code (e.g., a driver), compile and test on a development machine

AoT is an **automated off-target**

testing approach

Applications

- Unit test-like testing for complex software
- Easy debugging
- Quick compilation

Main point

AoT operates entirely on code information available in a **FTDB** and is fully independent from the original source code tree

SAMSUNG

KFLAT

KFLAT intro

Selective memory serialization for C structures

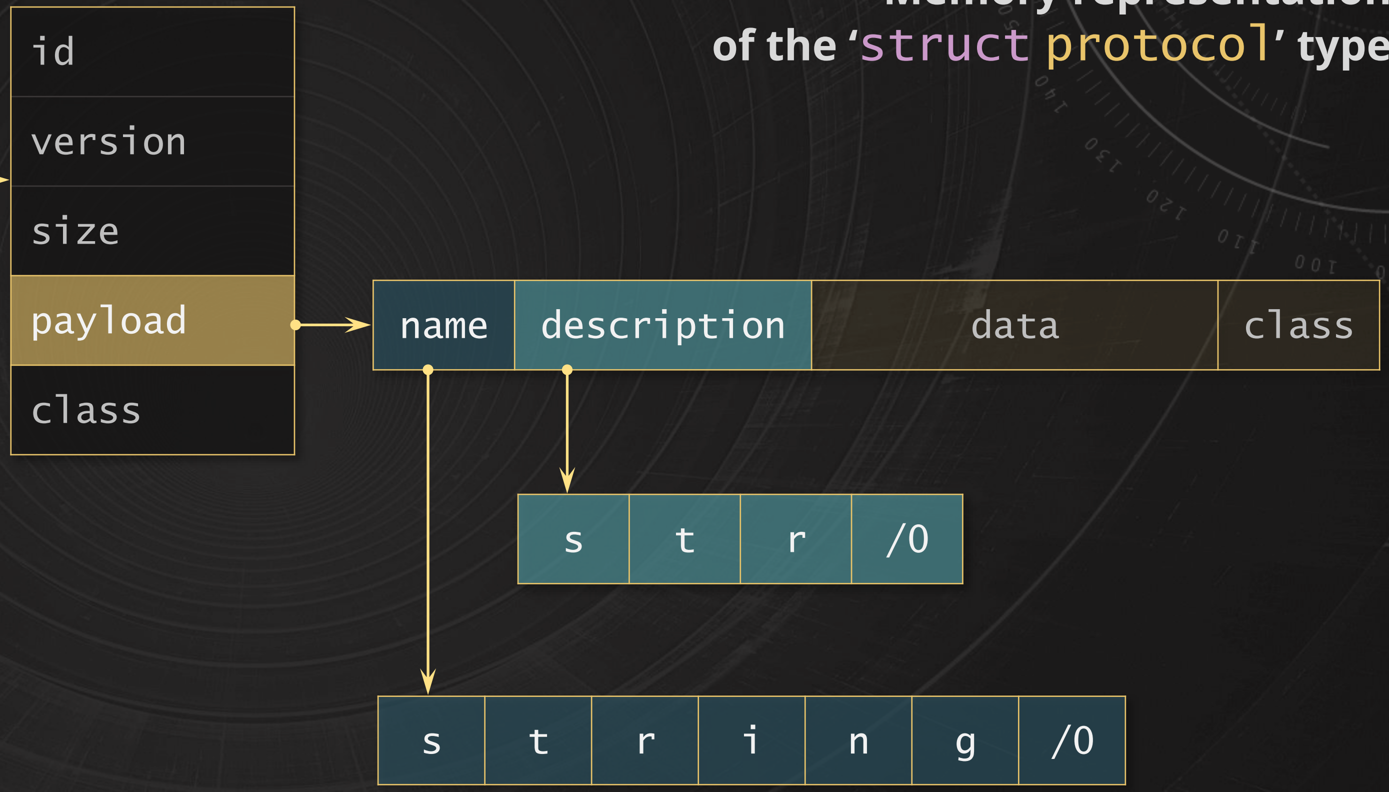
Packet processing application

Reminder of the memory structure of the protocol data

.h

```
struct protocol {
    int id;
    char version;
    unsigned long size;
    void* payload;
    enum proto_class class;
};

struct protocol_data {
    char* name;
    char* description;
    unsigned char data[40];
    enum proto_class class;
};
```

Memory representation of the 'struct protocol' type



SAMSUNG

KFLAT

KFLAT intro

Selective memory serialization for C structures

Packet processing application

Memory serialization and restoration of the protocol data

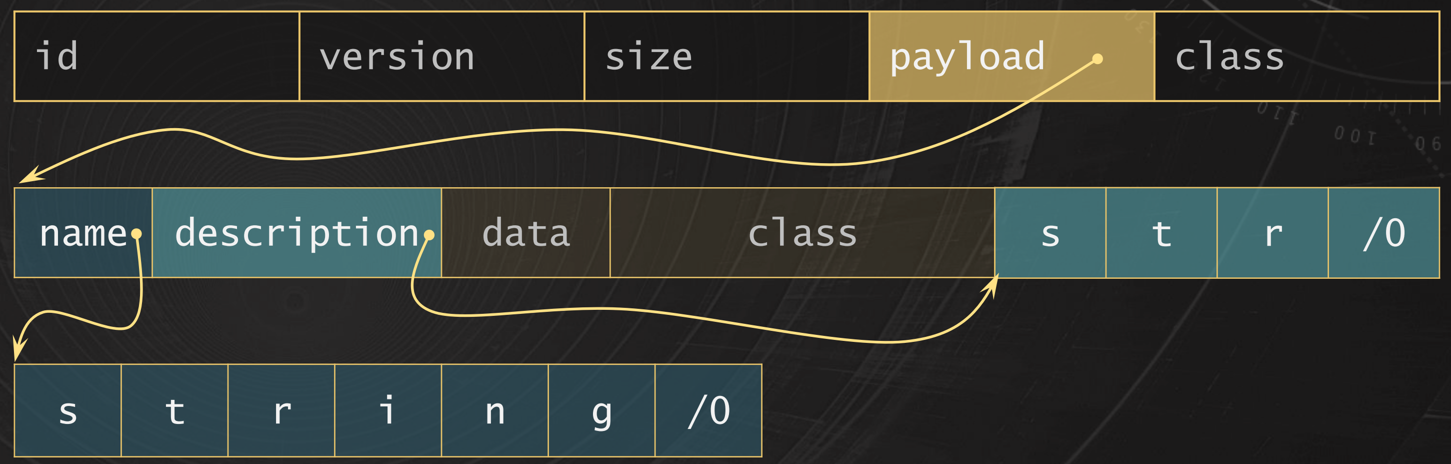
.h

```c

struct protocol {
    int id;
    char version;
    unsigned long size;
    void* payload;
    enum proto_class class;
};

struct protocol_data {
    char* name;
    char* description;
    unsigned char data[40];
    enum proto_class class;
};

```

.c

```c
struct protocol my_proto;
init_my_proto(&my_proto);
// (...)
FOR_ROOT_POINTER(&my_proto,
    FLATTEN_STRUCT(protocol, &my_proto);
);
```

GENERATES



Continuous memory fragment

READS

.c

```c
CUnflatten unflatten = unflatten_init();
struct protocol* my_proto =
    (const struct protocol*) unflatten_root_pointer_next(unflatten);
```

SAMSUNG

KFLAT recipes

Example of recipes for simple structure definition

KFLAT recipes

Telling the KFLAT engine how to serialize a given structure, i.e., what every pointer member is exactly pointing to

.c

```c

struct protocol {
     int id;
     char version;
     unsigned long size;
     void* payload;
     enum proto_class class;
};

struct protocol_data {
     char* name;
     char* description;
     unsigned char data[40];
     enum proto_class class;

```

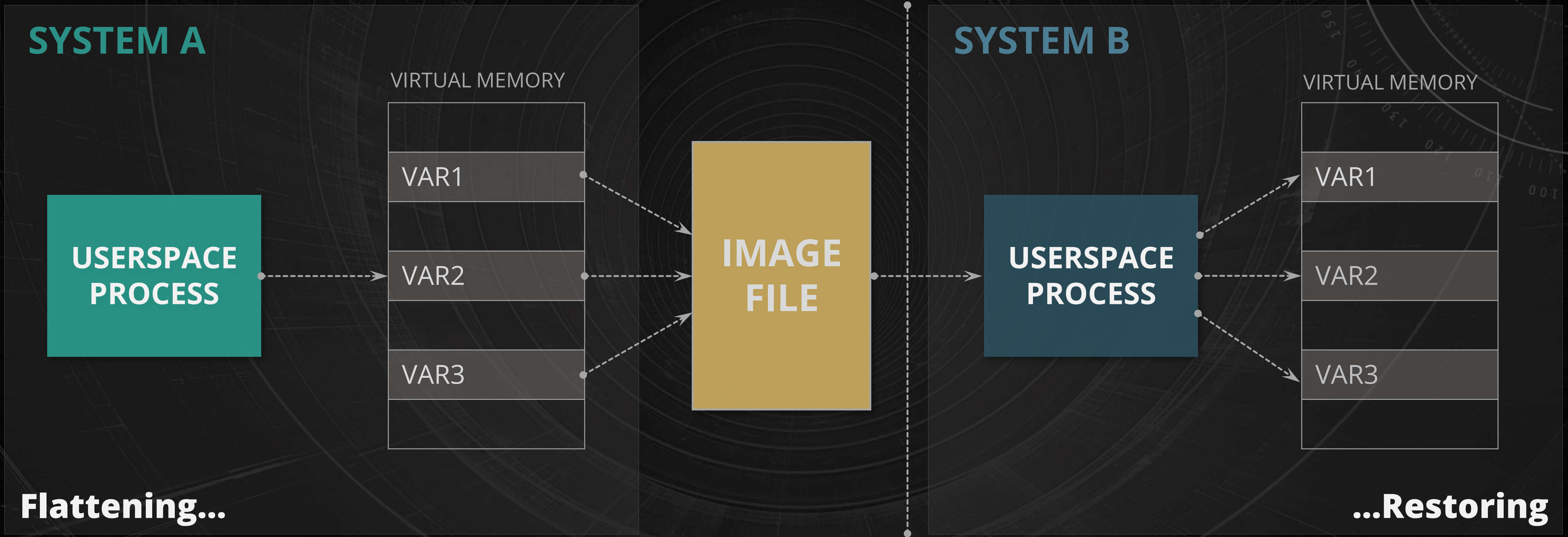
```c
FUNCTION_DEFINE_FLATTEN_STRUCT(protocol,
     AGGREGATE_FLATTEN_STRUCT(protocol_data,payload);
);

FUNCTION_DEFINE_FLATTEN_STRUCT(protocol_data,
     AGGREGATE_FLATTEN_STRING(name);
     AGGREGATE_FLATTEN_STRING(description);
);
```

KFLAT usage

Applying KFLAT to serialize user-space process memory variables

Caching computed memory of the Linux process

Memory serialization of source code variables for the user-space process



Useful for creating the cache of large memory structures which can be read/mapped quickly in other process

Example: large build system creating cache of parsed makefiles

KFLAT usage

Applying KFLAT to serialize kernel-space process memory variables

Help in Linux kernel debugging

.h

```
FUNCTION_DECLARE_FLATTEN_STRUCT(task_struct);

FUNCTION_DEFINE_FLATTEN_STRUCT(task_struct,
    AGGREGATE_FLATTEN_STRUCT_SHIFTED(task_struct,tasks.prev,-offsetof(struct task_struct,tasks));
    AGGREGATE_FLATTEN_STRUCT_SHIFTED(task_struct,tasks.next,-offsetof(struct task_struct,tasks));
);
```

```
## Found 236 tasks
T[1:1], cpu: 0, prio: 120, comm: init, flags: 1077936384, utime: 84000000, stime: 1989530070
T[2:2], cpu: 1, prio: 120, comm: kthreadd, flags: 2129984, utime: 0, stime: 12000000
T[3:3], cpu: 0, prio: 100, comm: rcu_gp, flags: 69238880, utime: 0, stime: 0
T[4:4], cpu: 0, prio: 100, comm: slub_flushwq, flags: 69238880, utime: 0, stime: 0
T[5:5], cpu: 0, prio: 100, comm: netns, flags: 69238880, utime: 0, stime: 0
T[7:7], cpu: 0, prio: 100, comm: kworker/0:0H, flags: 69238880, utime: 0, stime: 0
T[9:9], cpu: 0, prio: 100, comm: mm_percpu_wq, flags: 69238880, utime: 0, stime: 0
(…)
```

Image size produced: **1.5MB**

Problem: very large number of recipes to prepare for dependent types
Number of structures directly reachable from **struct task_struct: ~3000**

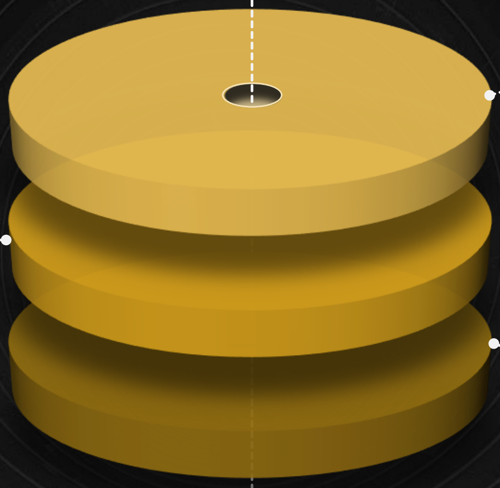
KFLAT: Summary

Selective memory serialization for the Linux system

Overview

Project: https://github.com/samsung/kflat
Talk: https://www.youtube.com/watch?v=Ynunpuk-Vfo

KFLAT



KFLAT can **serialize**

selected C variables and their dependencies

Recipes are required

to precisely describe the format of the data to dump

Applications

- State initialization of the Linux kernel Off-Target applications
- Linux kernel snapshots and debugging
- Memory caching of user space applications

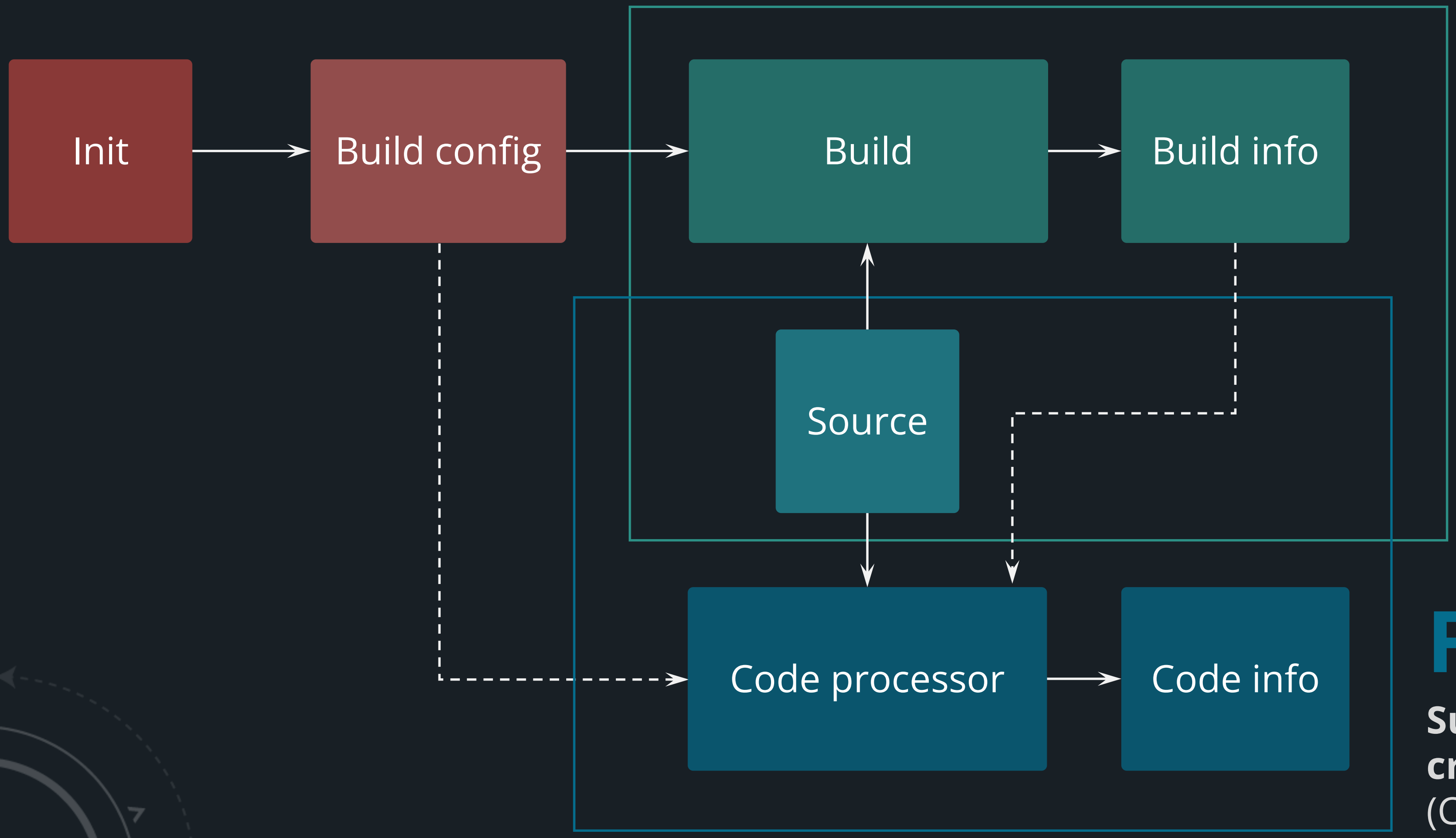
Main point

KFLAT can use information from the FTDB to automate preparation of the recipes that describe the layout of memory to be serialized

CAS: System overview

BAS

Linux OS system utility (build system agnostic)
Special understanding of C/C++ compilation/linking
(easily extensible for other special process classes)



FTDB

Support for code database creation for C programs only
(C++ implementation is on its way)

SAMSUNG

Conclusion

DPE in the Complex Low-Level System World

Key takeaways regarding CAS

1. Large **S/W systems are extremely complex** these days
2. Detailed **build information helps you navigate through this complexity**
3. **CAS allows you to acquire detailed build information** from a complex S/W system allowing radical improvement of S/W engineering tools and processes
4. **Source verification** tasks related to the code are **difficult to automate** due to inherent complexity of C/C++
5. **CAS creates database of code information extracted from C code** (C++ to be added) which opens new areas of possibilities for automation
6. **CAS improves automated testing and the review** of C code in practice
7. **CAS forms a basis for other S/W Engineering tools** that increase developers productivity
8. **You are the one to find new ways of using it!**

CAS is open source

Feedback from the community highly appreciated: b.zator@samsung.com
or create issue/PR at: https://github.com/samsung/CAS

SAMSUNG