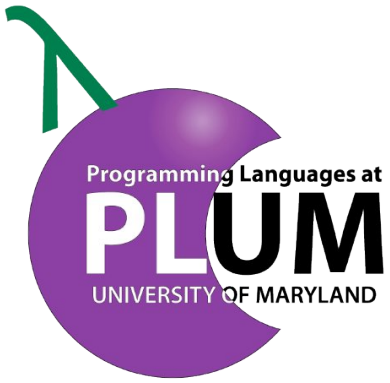
Dynamic Software Updating (DSU) on a Large Scale

Karla Saur



Kitsune: A Practical DSU System

- Whole-program updates for C
- Entirely standard compilation and tools

Previously with Kitsune:

Program	# Vers		LoC
memcached	3	(1.2.2–1.2.4)	4,181
vsftpd	14	(1.1.0–2.0.6)	12,202
redis	5	(2.0.0–2.0.4)	13,387
icecast	5	(2.2.0–2.3.1)	15,759
Tor	13	(0.2.1.18–0.2.1.30)	76,090

Kitsune: A Practical DSU System

- **Ease of Use:** Minimal per-update programmer work
- **Flexibility:** Supports natural program evolution, on-the-fly
- **Efficiency:** Quick update times, no steady-state overhead
- **Scalability:** Support large, complex programs

From snort.org:

> *Snort® is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection...*

Rules updated every few days

↳ Auto-reload partially built-in for rules only

Versions released every few months

↳ Requires full shutdown

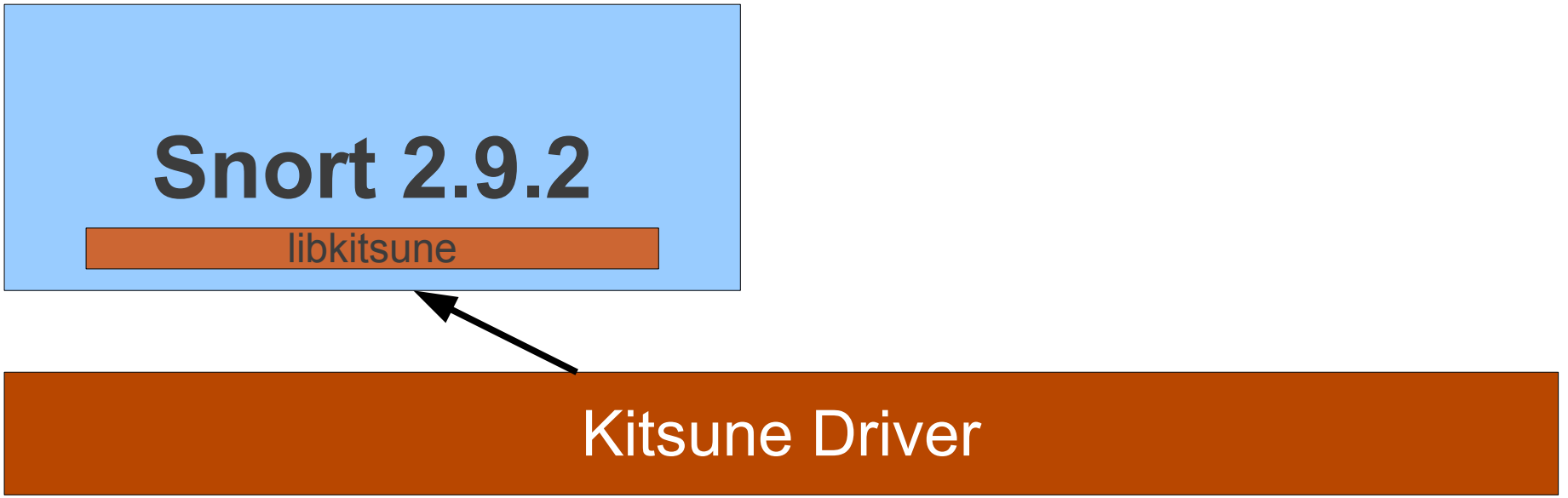
- DSU Challenges
  - Dynamic modules for protocols
  - Configuration file changes what goes where
  - ~215K lines of code in 2.9.2

Program	# Vers		LoC
memcached	3	(1.2.2–1.2.4)	4,181
vsftpd	14	(1.1.0–2.0.6)	12,202
redis	5	(2.0.0–2.0.4)	13,387
icecast	5	(2.2.0–2.3.1)	15,759
Tor	13	(0.2.1.18–0.2.1.30)	76,090
snort	4	(2.9.2–2.9.2.3)	214,703

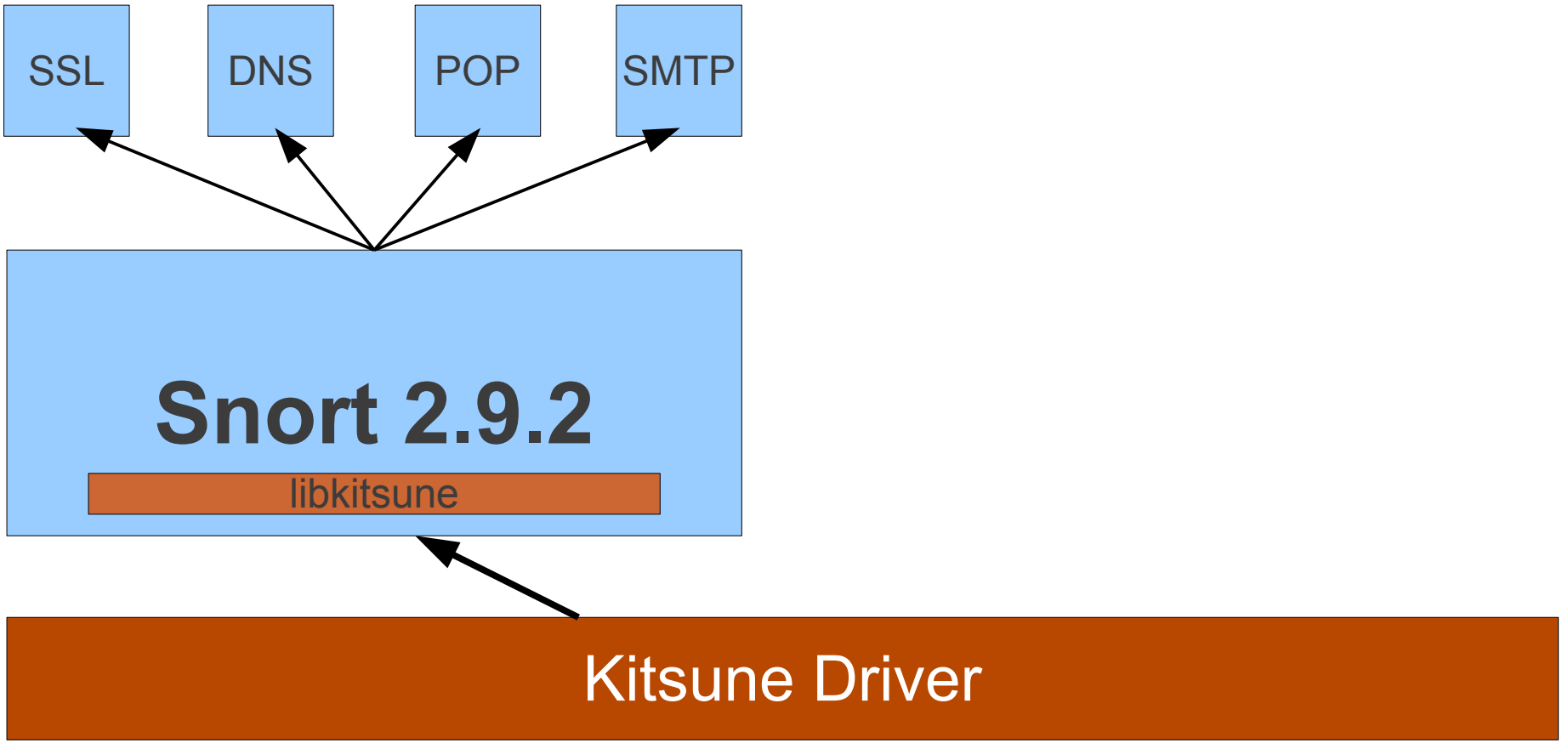
Kitsune and Snort

Kitsune Driver

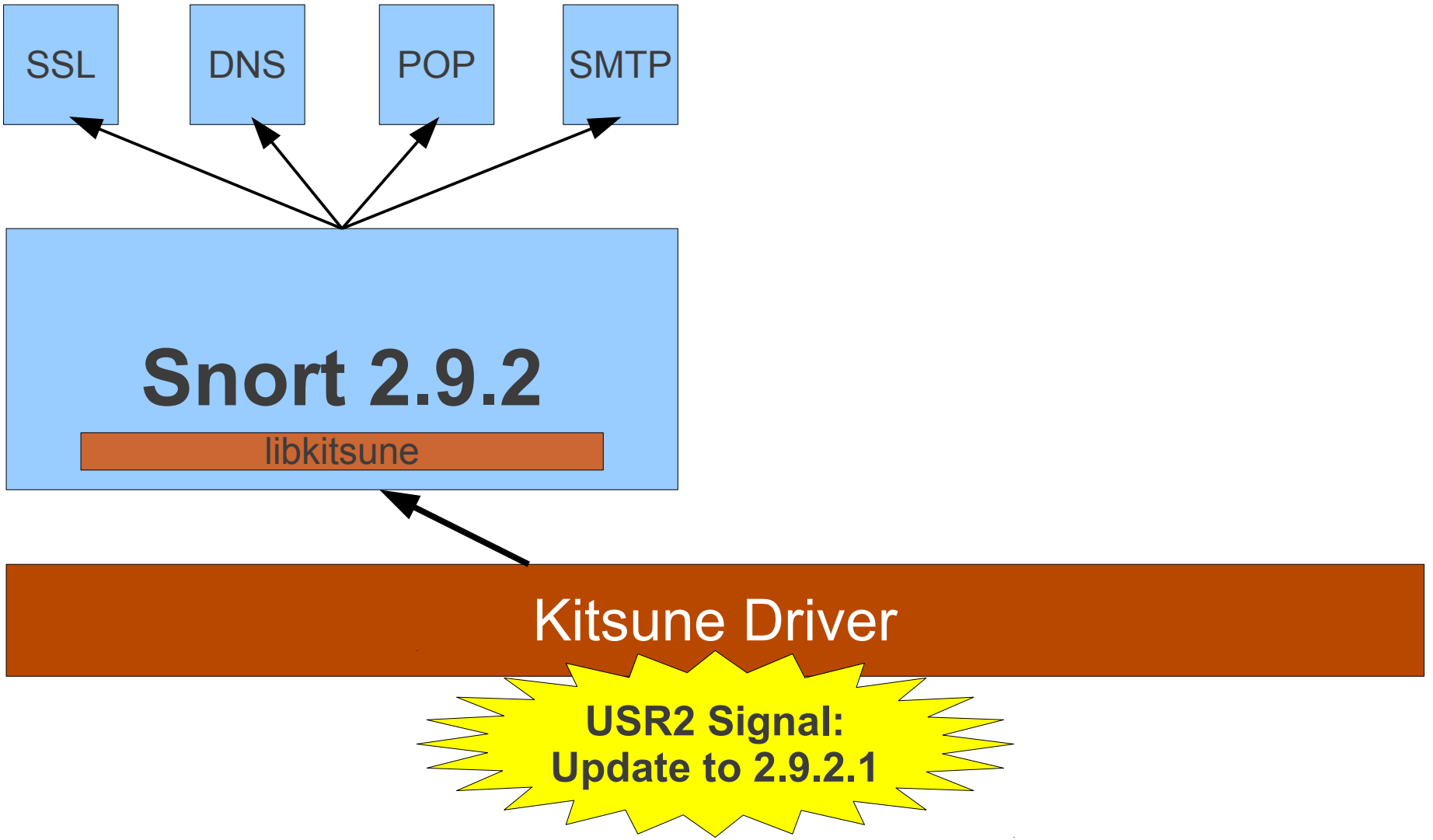
Kitsune and Snort



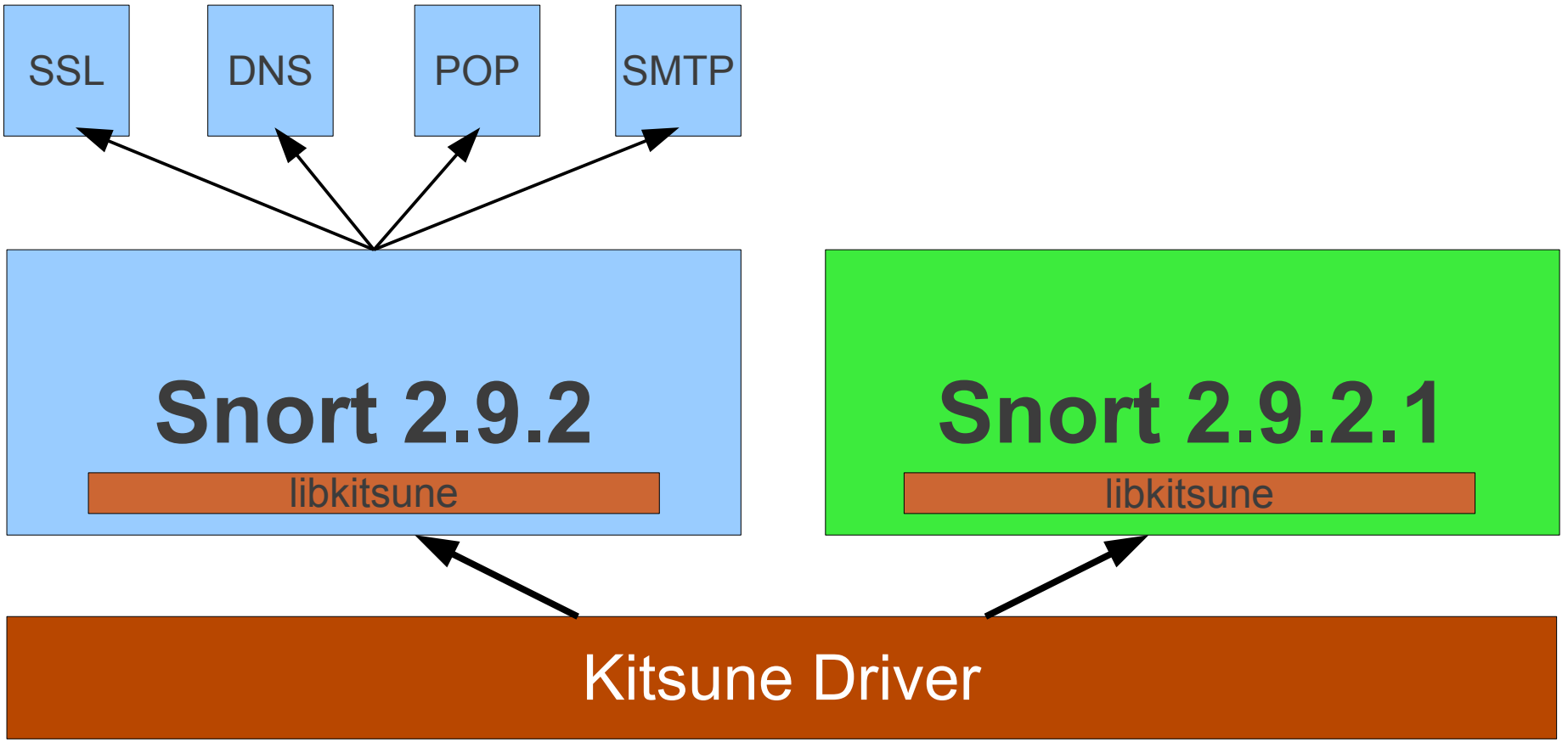
Kitsune and Snort



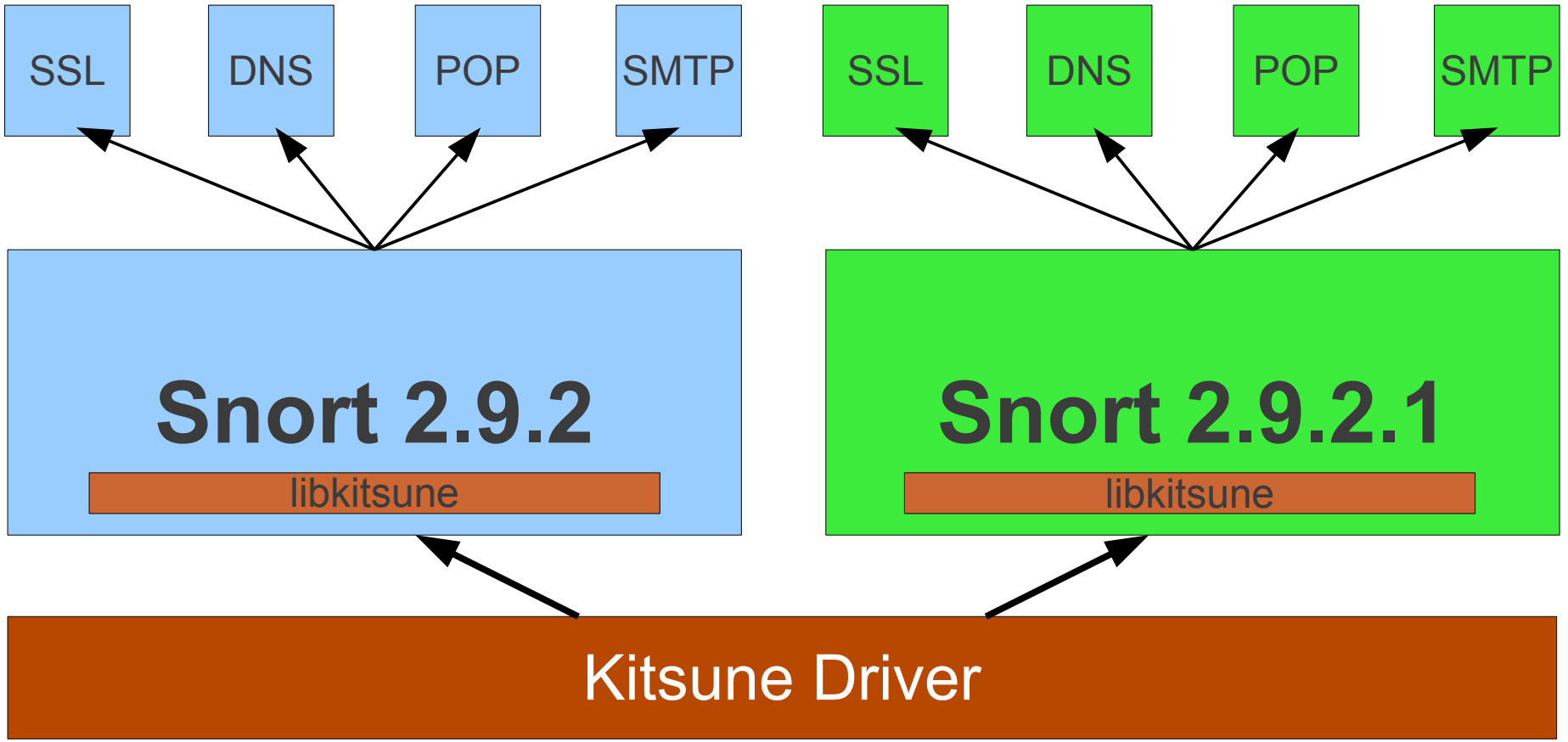
Updating


SSL
DNS
POP
SMTP
Snort 2.9.2
libkitsune
Kitsune Driver
USR2 Signal:
Update to 2.9.2.1

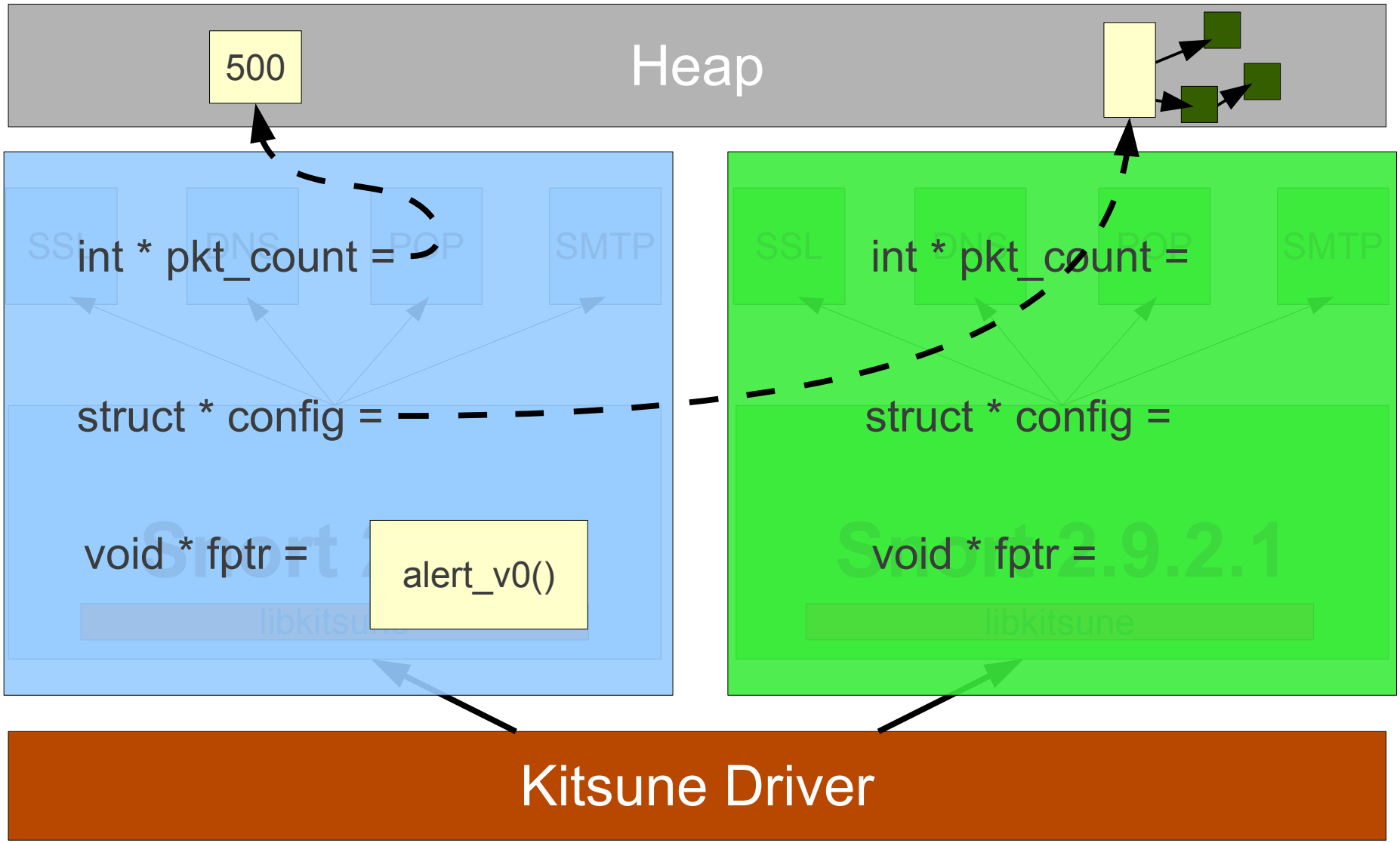
Updating: Load New



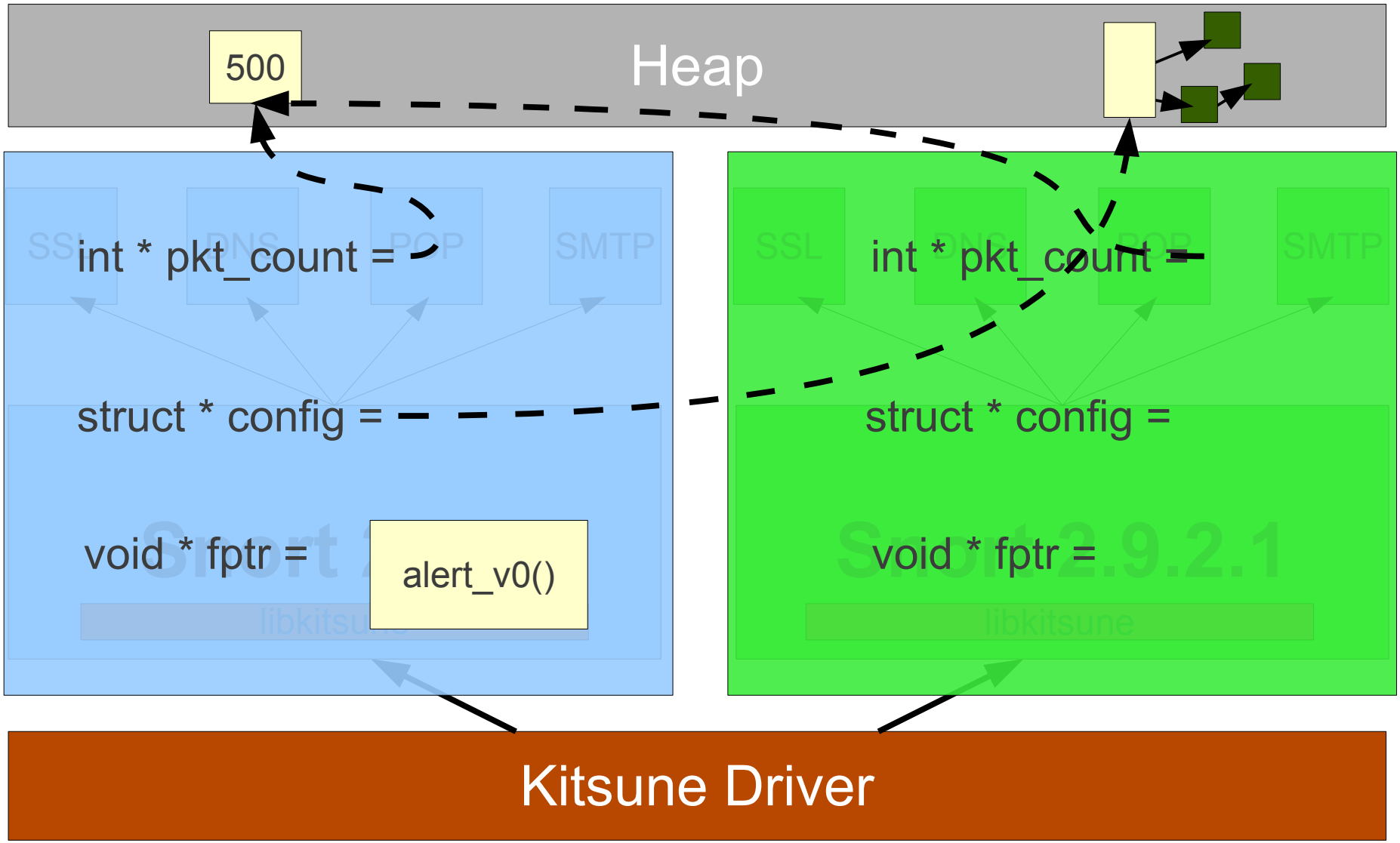
Updating: Load New



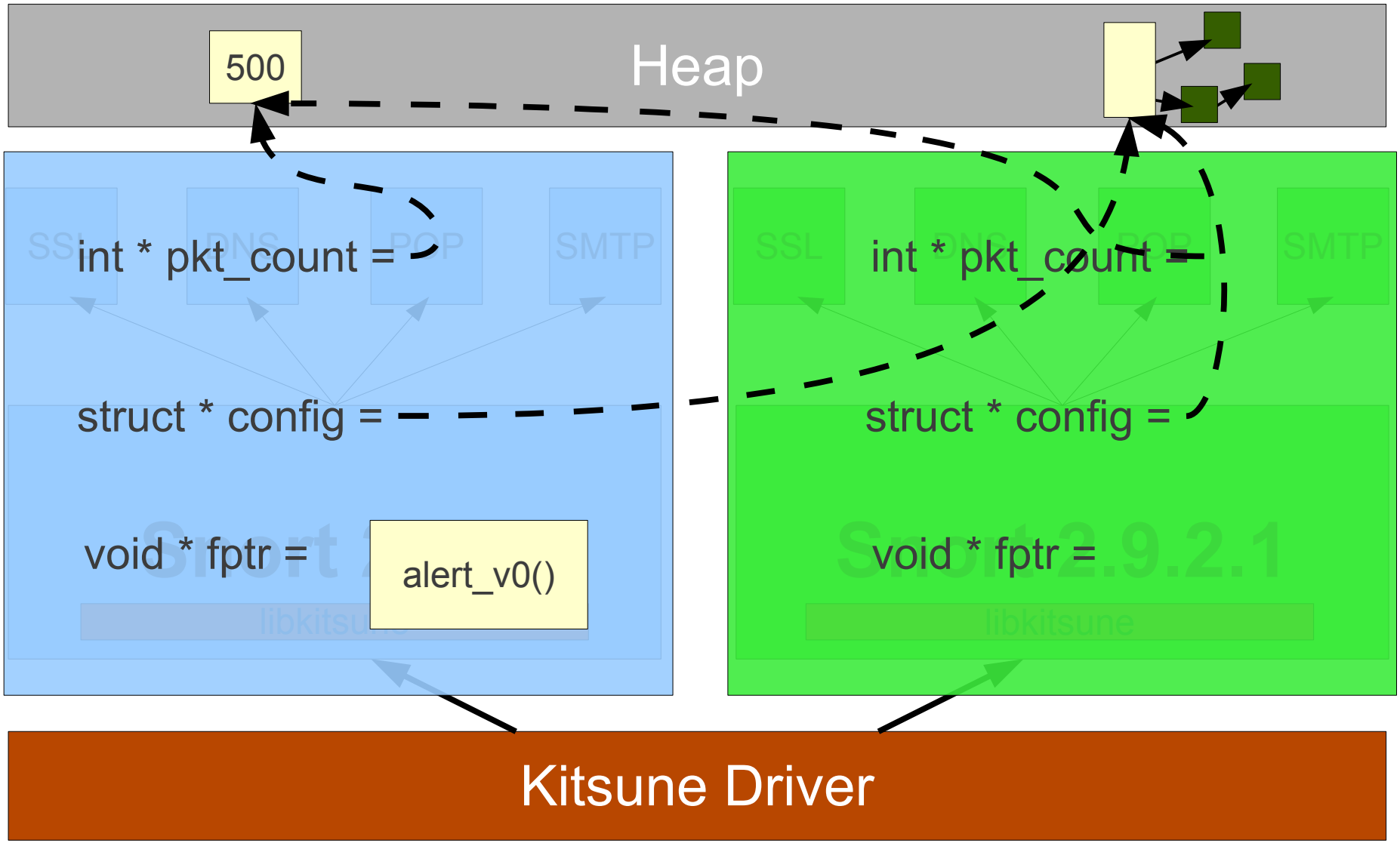
Updating: State Transfer



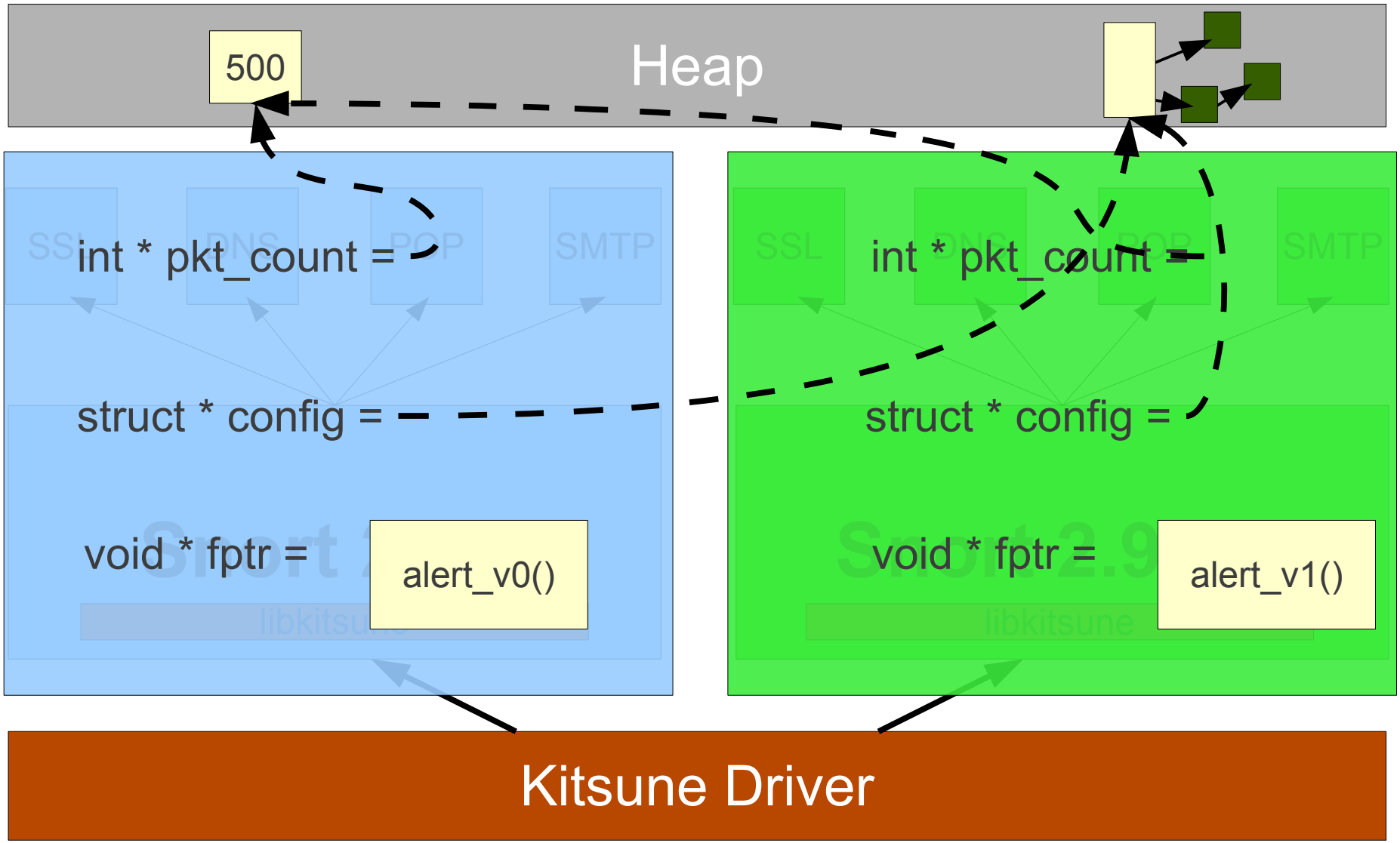
Updating: State Transfer



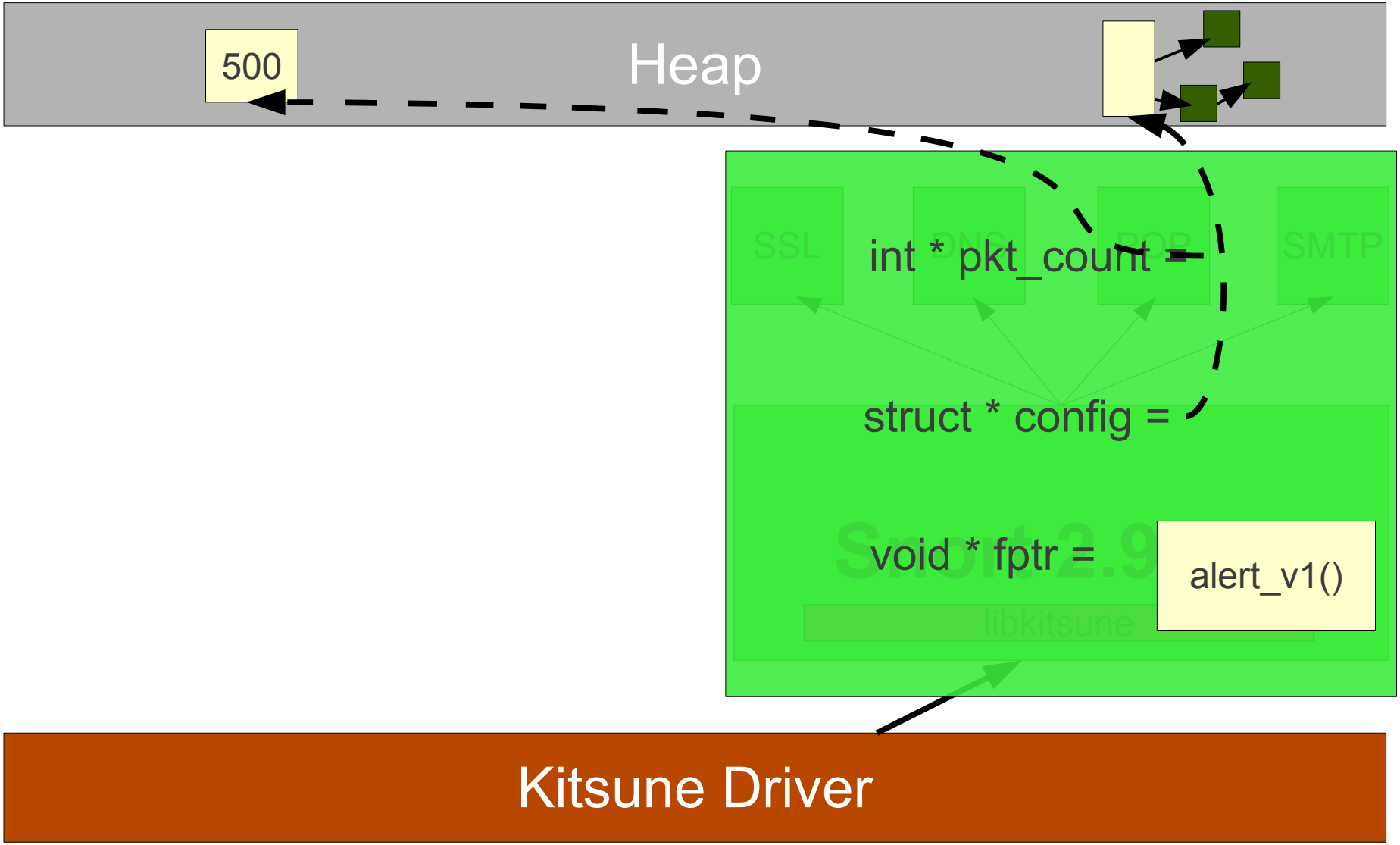
Updating: State Transfer



Updating: State Transfer



Updating: Cleanup



Programmer Obligations:

Quick Overview

```
1  char *snort_conf_dir = NULL;
2  SnortConfig *snort_conf = NULL;
3  int main(){
4
5
6          SnortInit();
7
8          //read from libpcap
9          PacketLoop();
10 }
11 void SnortInit(){
12         //initialize main SnortConfig structure
13 }
14 void PacketLoop(){
15         while(!exit){
16
17                 //wait for packet
18                 pkt = read(libpcap);
19                 send_to_Decoder(pkt);
20         }
21 }
```

Programmer Obligations:

Update Points

```
1  char *snort_conf_dir = NULL;
2  SnortConfig *snort_conf = NULL;
3  int main(){
4
5
6          SnortInit();
7
8          //read from libpcap
9          PacketLoop();
10 }
11 void SnortInit(){
12         //initialize main SnortConfig structure
13 }
14 void PacketLoop(){
15         while(!exit){
16                 kitsune_update();
17                 //wait for packet
18                 pkt = read(libpcap);
19                 send_to_Decoder(pkt);
20         }
21 }
```

Programmer Obligations:

Control Migration

```
 1  char *snort_conf_dir = NULL;
 2  SnortConfig *snort_conf = NULL;
 3  int main(){
 4
 5      if(!kitsune_is_updating()){  // only perform initialization if not updating
 6          SnortInit();
 7      }
 8      //read from libpcap
 9      PacketLoop();
10  }
11  void SnortInit(){
12      //initialize main SnortConfig structure
13  }
14  void PacketLoop(){
15      while(!exit){
16          kitsune_update();
17          //wait for packet
18          pkt = read(libpcap);
19          send_to_Decoder(pkt);
20      }
21  }
```

Programmer Obligations:

Data Migration

```
 1 char *snort_conf_dir = NULL; //automigrated
 2 SnortConfig *snort_conf = NULL; //automigrated
 3 int main(){
 4       kitsune_do_automigrate();  // perform state transfer here if updating
 5       if(!kitsune_is_updating()){  // only perform initialization if not updating
 6           SnortInit();
 7       }
 8       //read from libpcap
 9       PacketLoop();
10 }
11 void SnortInit(){
12       //initialize main SnortConfig structure
13 }
14 void PacketLoop(){
15       while(!exit){
16             kitsune_update();
17             //wait for packet
18             pkt = read(libpcap);
19             send_to_Decoder(pkt);
20       }
21 }
```

Migrating and Transforming State

The *xfgen* tool:

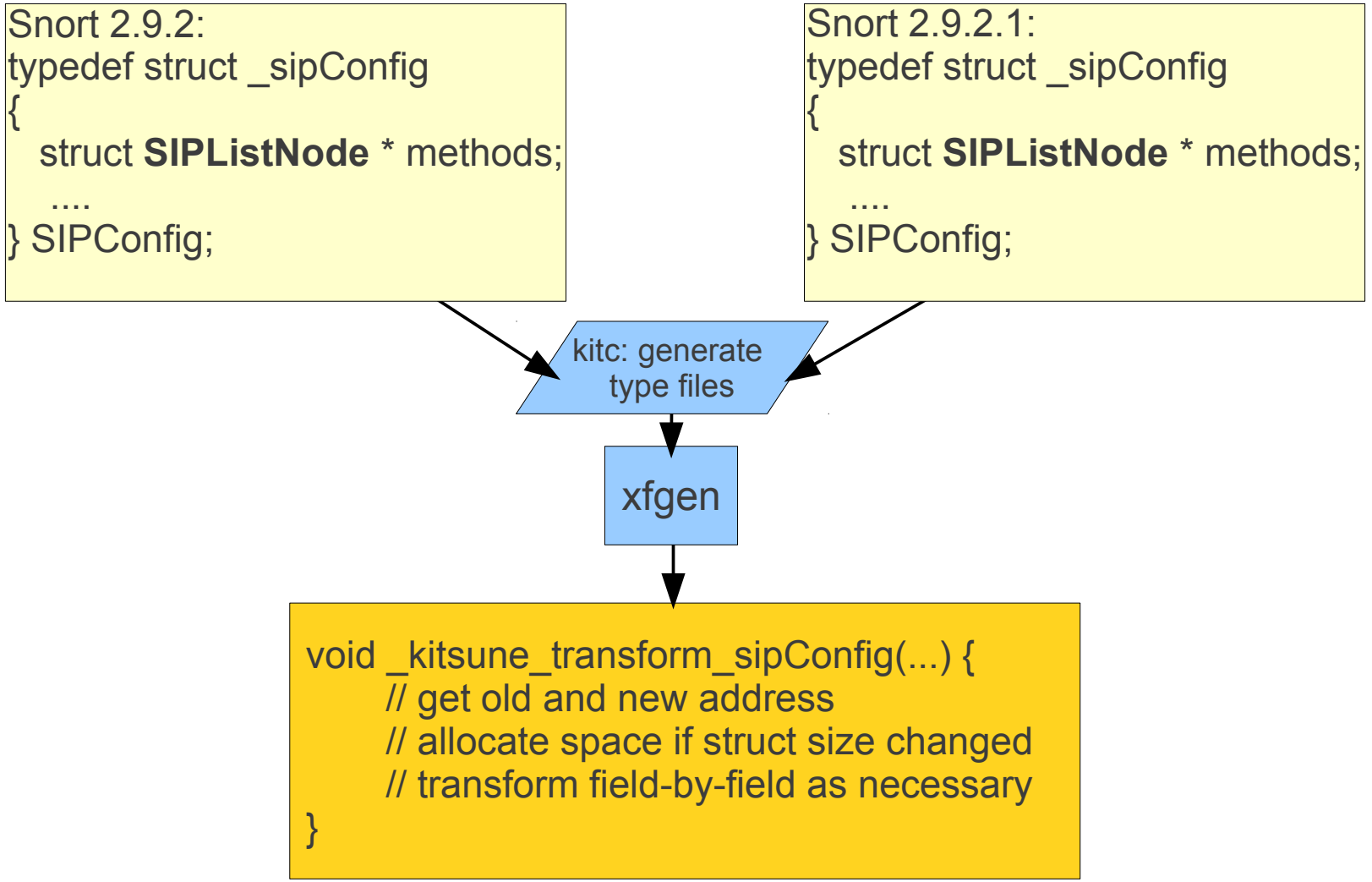
- Generates migration code based on old and new program version
- Asks for programmer input as necessary

Generated code:

- Has transformer functions for each heap element, which together form a full-heap traversal

xfgen

The majority of transformation functions are generated by xfgen with no additional work:



xfgen

Generated functions form a complete heap traversal:

Snort *Code:*

```
SIPConfig *sip_eval_config;

typedef struct _sipConfig
{
     .....
    SIPListNode * methods;
} SIPConfig;

typedef struct _sipListNode
{
       ....
      struct _sipListNode* nextm;
} SIPListNode;
```

xfgen *Generated Code:*

```
void _kitsune_transform_sipConfig(...){
       ....

}
```

xfgen

Generated functions form a complete heap traversal:

Snort *Code:*

```
SIPConfig *sip_eval_config;

typedef struct _sipConfig
{
    .....
   SIPListNode * methods;
} SIPConfig;

typedef struct _sipListNode
{
      ....
     struct _sipListNode* nextm;
} SIPListNode;
```

xfgen *Generated Code:*

```
void _kitsune_transform_sipConfig(...){
        ....
        _kitsune_transform_sipListNode(...);
}

void _kitsune_transform_sipListNode(...){
        ....
        _kitsune_transform_sipListNode(...);
}
```

xfgen rules:
Transforming Values

Identify the actual type of void*:

```
typedef struct _SnortConfig
{
    RunMode run_mode;
    int run_mode_flags;
    int run_flags;
    void* daq_vars;
    ....
} SnortConfig;
```

```
typedef struct _StringVector{
    char** v;
    unsigned n;
} StringVector;

void ConfigDaqVar(SnortConfig *sc, char *args){
    StringVector * sv = malloc(sizeof(StringVector));
    sc->daq_vars = sv;
}
```

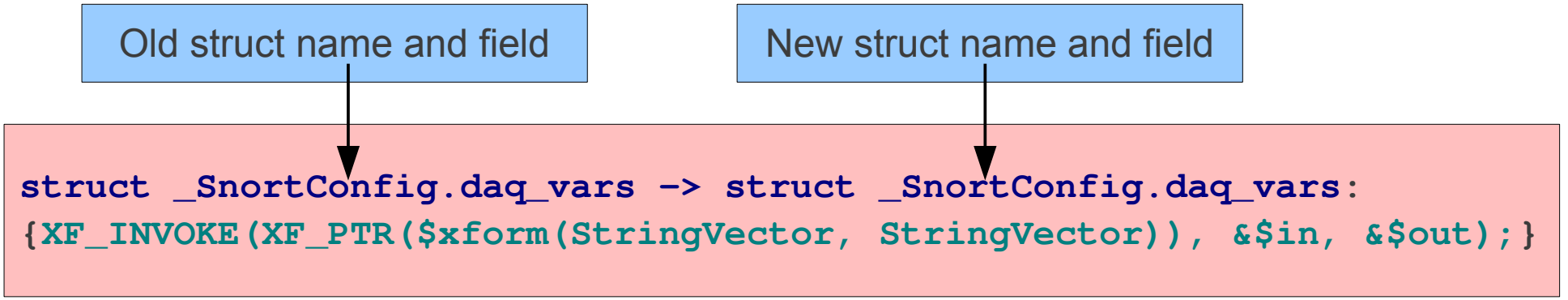
xfgen rules:
Transforming Values

Transform *old* variable or values to new:

```
struct _SnortConfig.daq_vars -> struct _SnortConfig.daq_vars:
{XF_INVOKE(XF_PTR($xform(StringVector, StringVector)), &$in, &$out);}
```

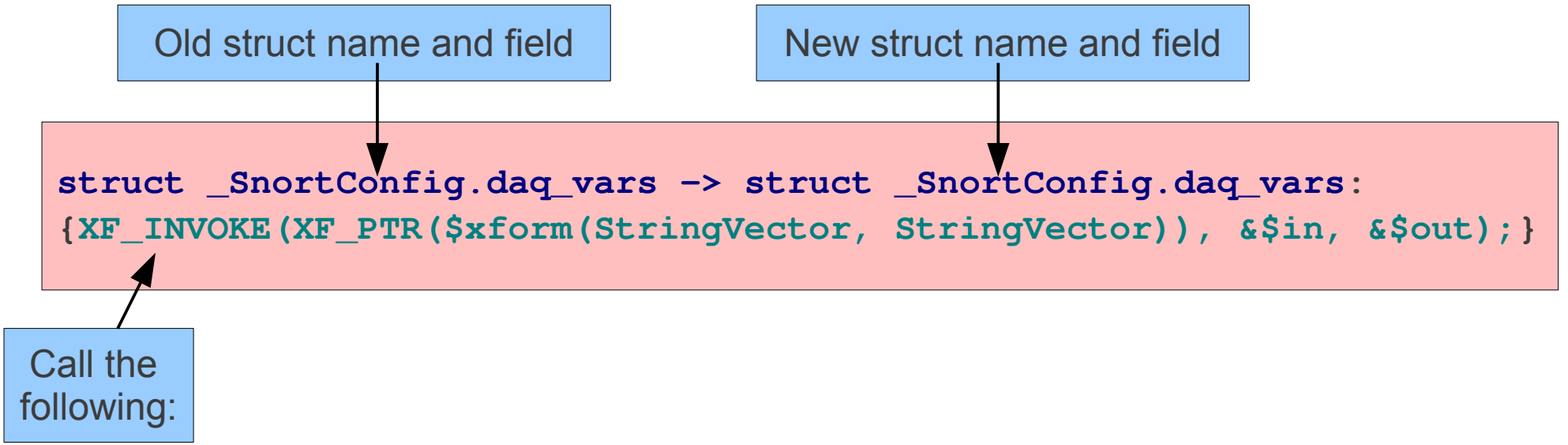
xfgen rules:
Transforming Values

Transform *old* variable or values to new:



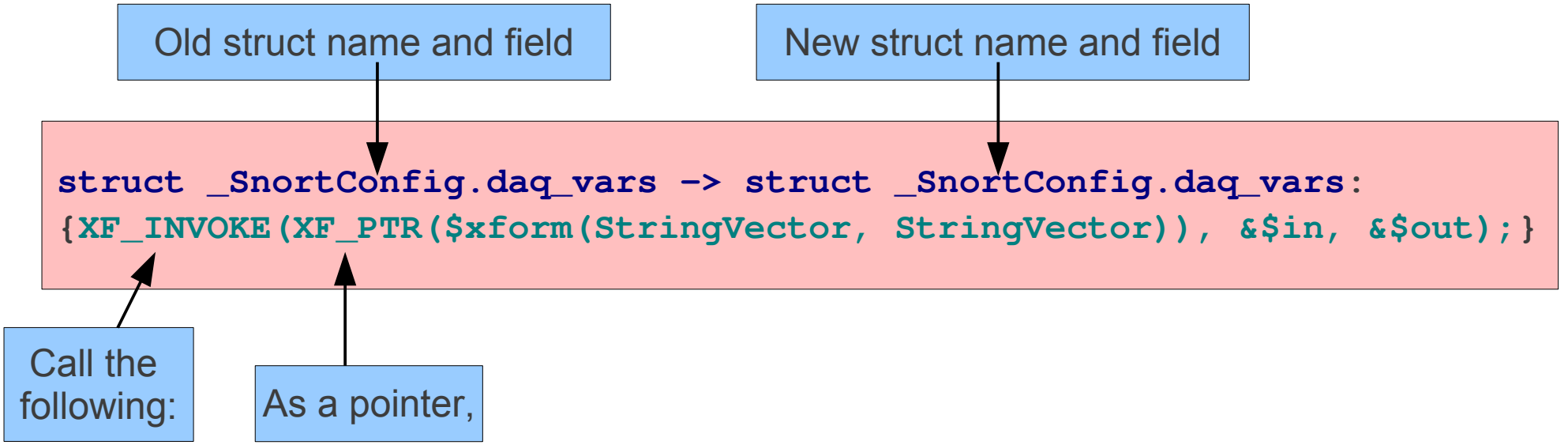
xfgen rules:
Transforming Values

Transform *old* variable or values to new:



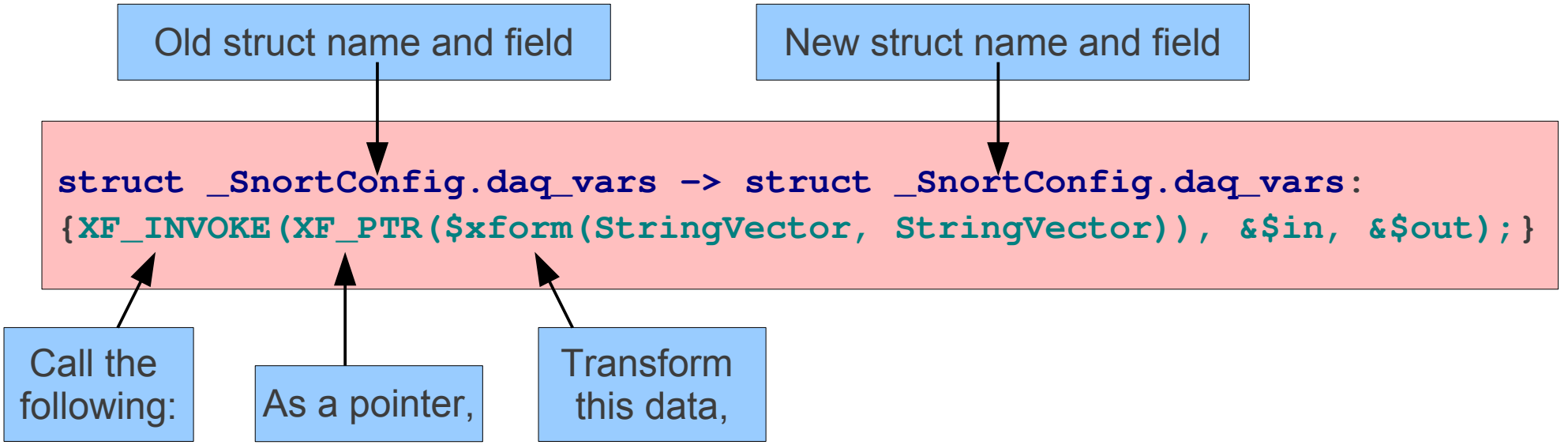
xfgen rules:
Transforming Values

Transform *old* variable or values to new:



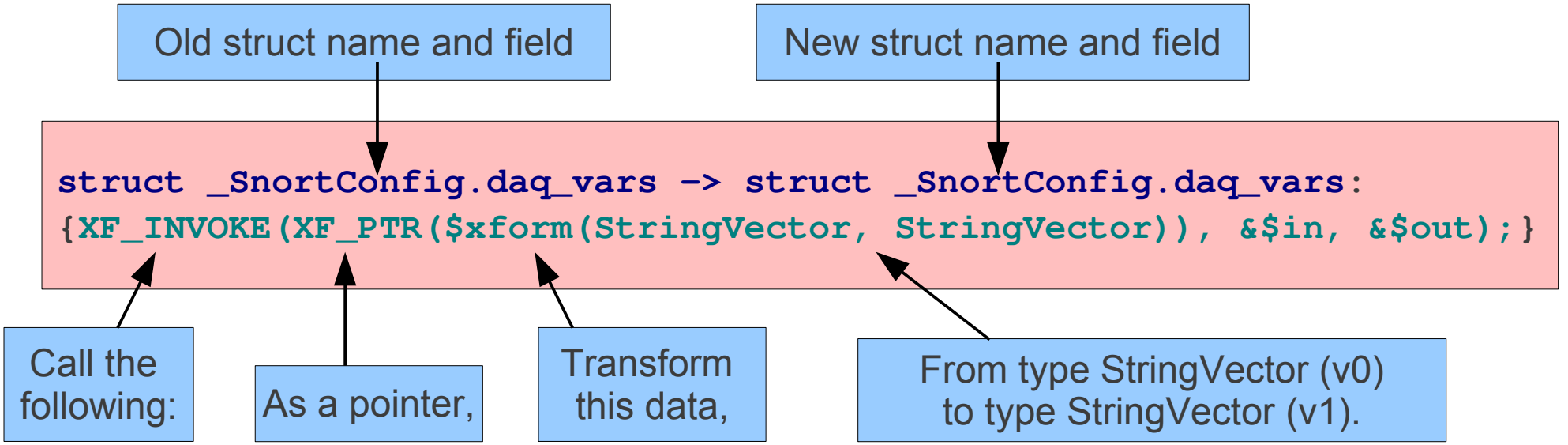
xfgen rules:
Transforming Values

Transform *old* variable or values to new:



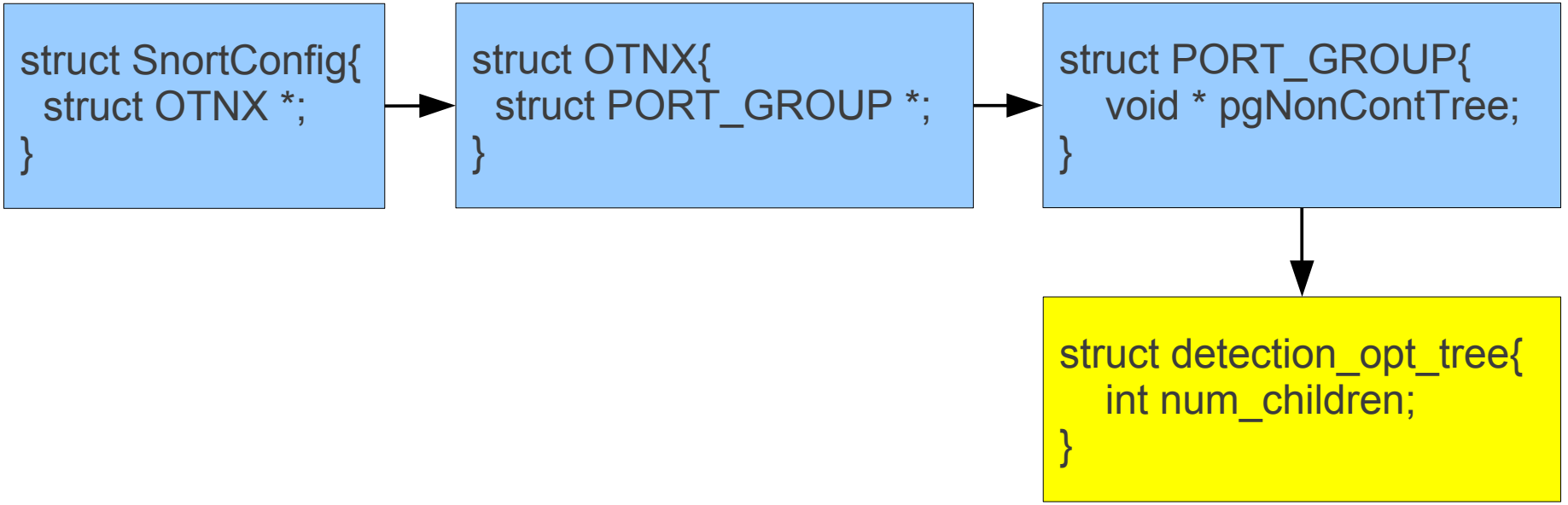
xfgen rules:
Transforming Values

Transform *old* variable or values to new:



Challenges – void *'s

- Identifying and annotating 144 void *'s
  - void *'s allow flexibility with plugins, but make code difficult to update dynamically
  - Structures are often nested many layers deep - cumbersome debugging.



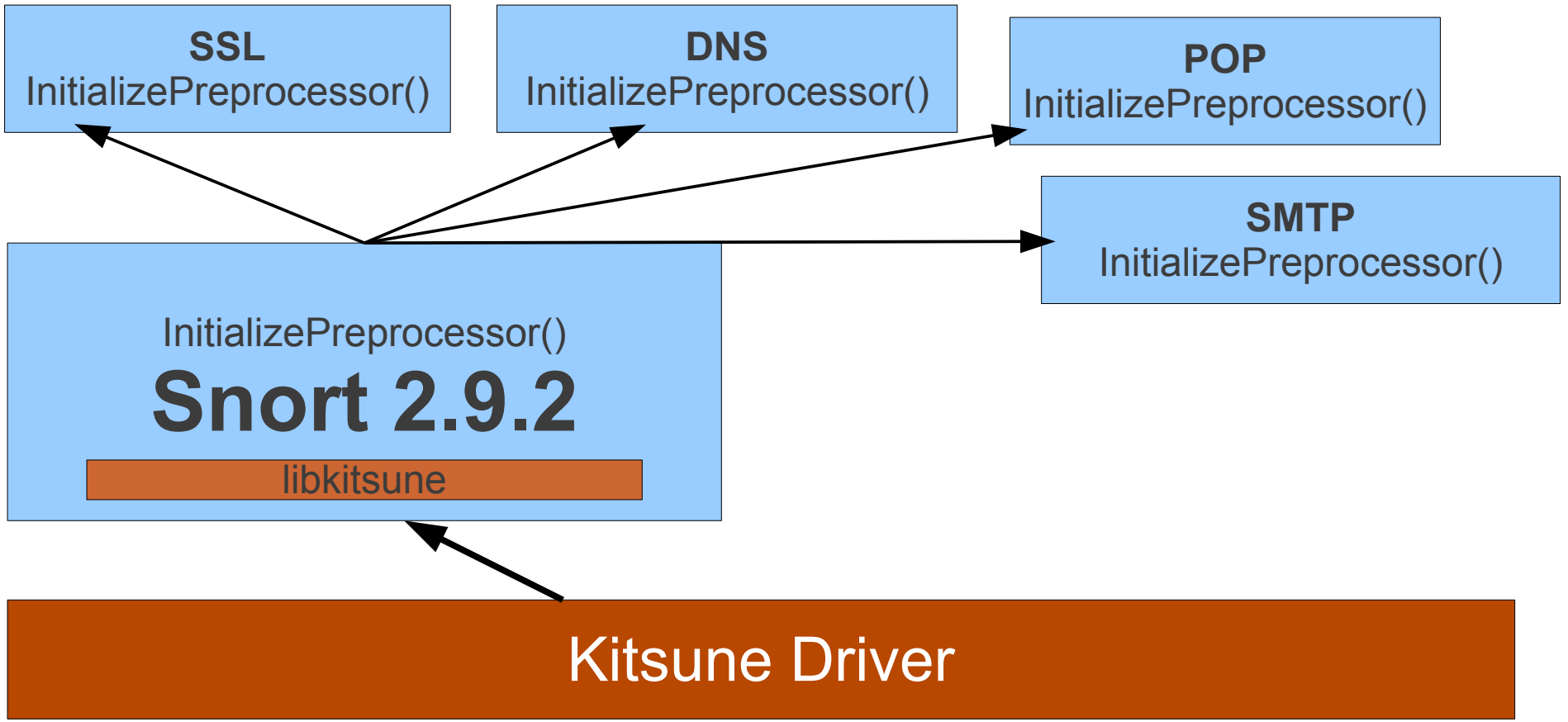
Challenges – void *'s

Some types change based on the snort.conf file

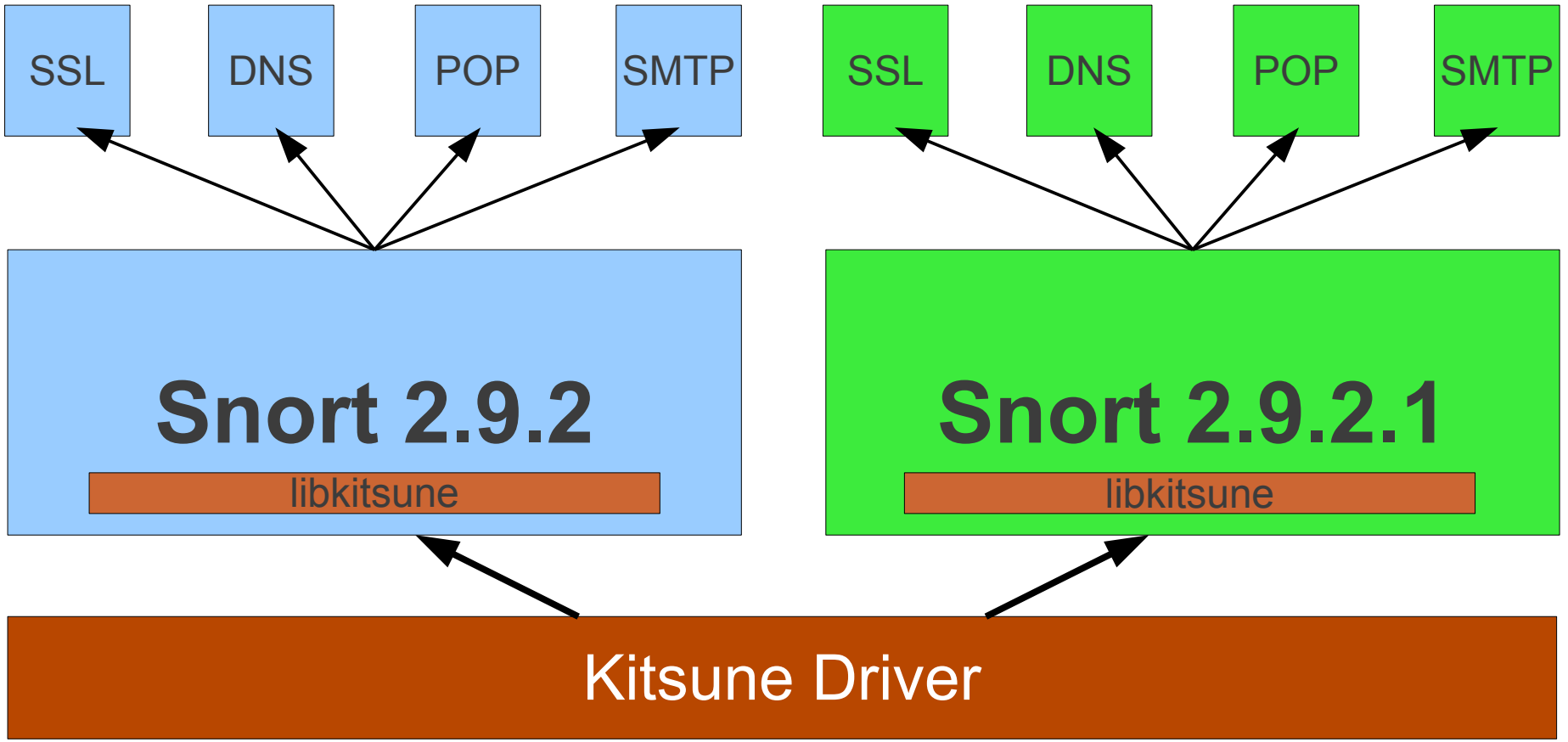
```
### snort.conf:

# unified2  # Recommended for most
output unified2

# syslog
####output alert_syslog
```

```
1 typedef struct _OutputFuncNode {
2     void *arg;
3     union {
4         OutputFunc fptr;
5         void *vfptr;
6     } fptr;
7     struct _OutputFuncNode *next;
8 } OutputFuncNode;
```

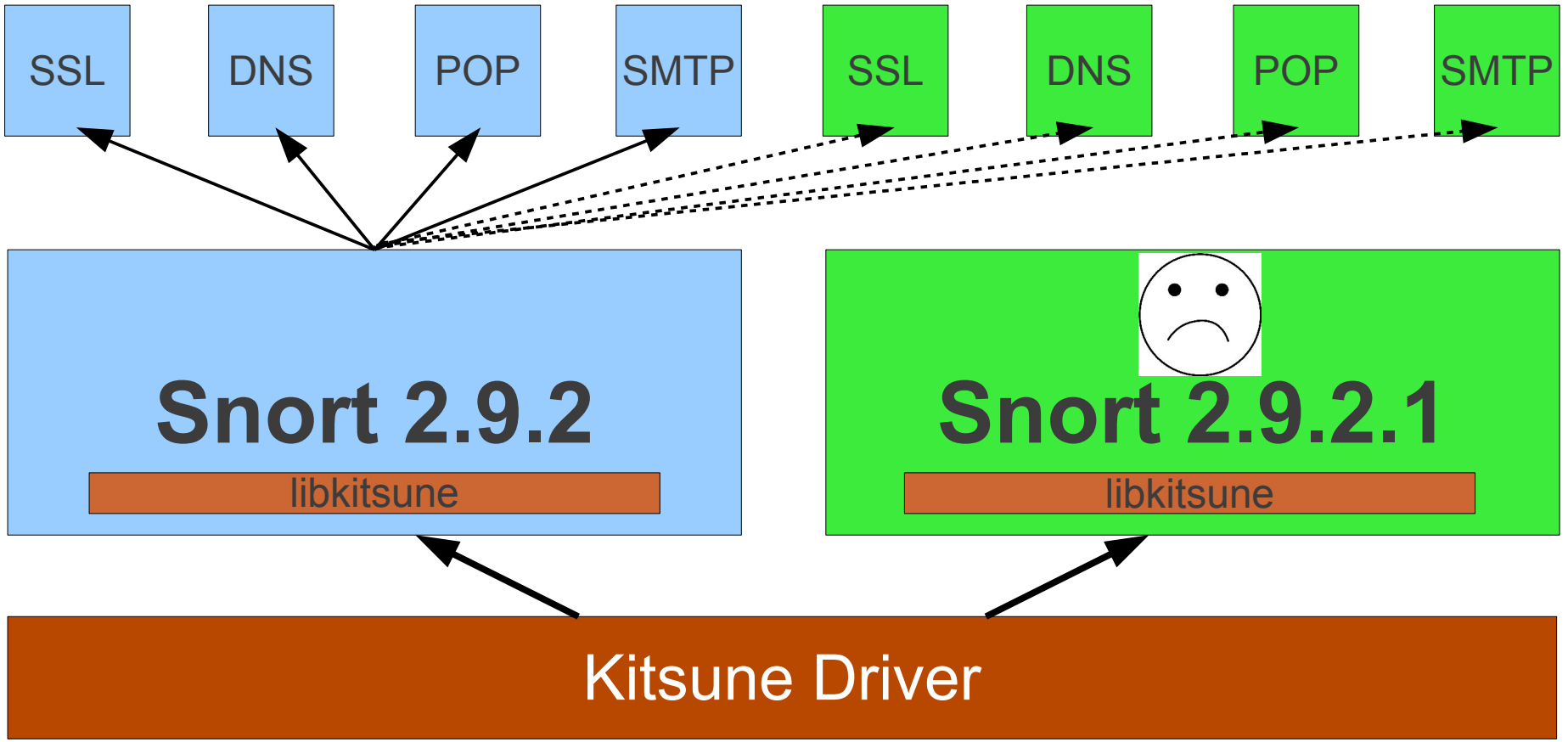
Challenges - Plugins



Challenges – Dynamic Linking



Challenges – Dynamic Linking



Challenges – Dynamic Linking

mainv1.so:
gcc -shared **-Wl,-soname,mainv1.so.1,--version-script=mainv1.ld** ...
mainv2.so:
gcc -shared **-Wl,-soname,mainv2.so.1,--version-script=mainv2.ld** ...

pluginv1.so:
gcc -shared **-Wl,-soname,pluginv1.so.1** pluginv1.o ...
pluginv2.so:
gcc -shared **-Wl,-soname,pluginv2.so.1** pluginv2.o **-L./ -lmainv2** ...

mainv1.ld

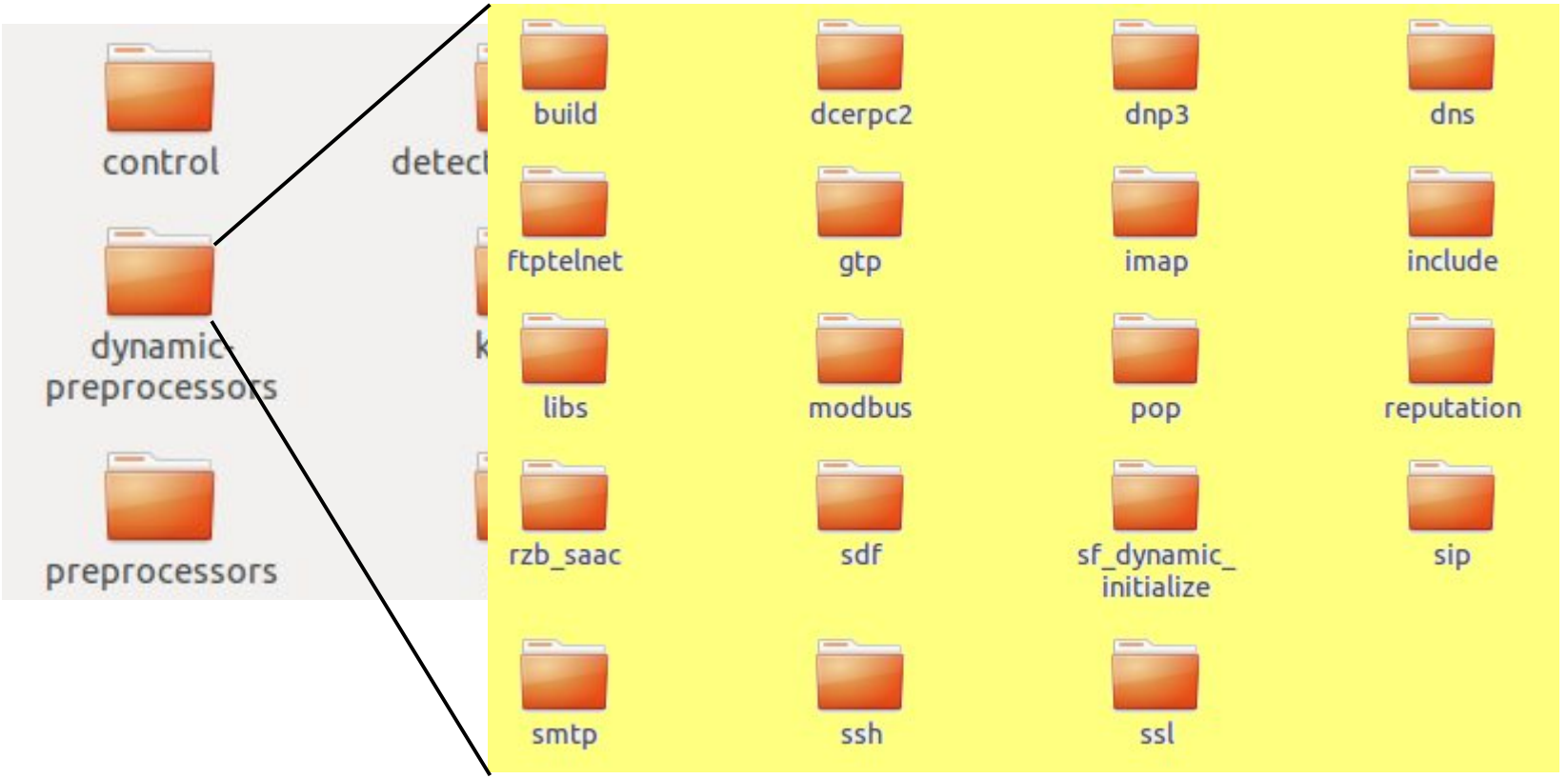
```
VERS_2.9.2 {
   global: *;
};
```

mainv2.ld

```
VERS_2.9.2.1 {
   global: *;
};
```

Challenges - Build

Snort 2.9.2 has 44 Makefiles in /src, all which required build changes



Snort Changes

Version	LoC*	LoC Changed	xf
2.9.2	214,078	493	221
2.9.2.1	214,329	0	15
2.9.2.2	214,625	36	13
2.9.2.3	214,703	0	48

*Lines of ANSI-C only as reported by sloccount

- Large majority of the changes were setting up the *initial* version to work with Kitsune
- Subsequent changes are often very simple if not automatic

Update Time

Median of 11 trials on a Core2Duo 1.8Ghz 2GB Update Time in ms			
	plugins & 30 pkts per sec	no plugins & 30 pkts per sec	plugins & no traffic
2.9.2 → 2.9.2.1	759.5	692.2	221.4
2.9.2.1 → 2.9.2.2	230.6	149.5	221.5
2.9.2.2 → 2.9.2.3	248.8	160.8	236.6

- No packets are dropped at any point
  - Also tested at full-speed with tcpreplay
- Opening 14 plugins adds ~80ms
- 2.9.2 → 2.9.2.1 update is slowest due to number of transformations

Summary

- We updated Snort, a ~215K line program using Kitsune
  - Relatively few changes
  - Low update pause time – no packets dropped
  - Maintained full state across update
- Lessons learned for large systems:
  - void *'s critical
  - Building/linking becomes a consideration
  - Consider external factors (conf file, etc)

Current Work

- Parallel Updating
  - Use multiple threads to perform heap traversal and transformation

Updating to	Tasks	1 Thd (siqr)	4 Thds (siqr)
snort 2.9.2.1	991831	396.56ms (2.74ms)	153.92ms (7.50ms)

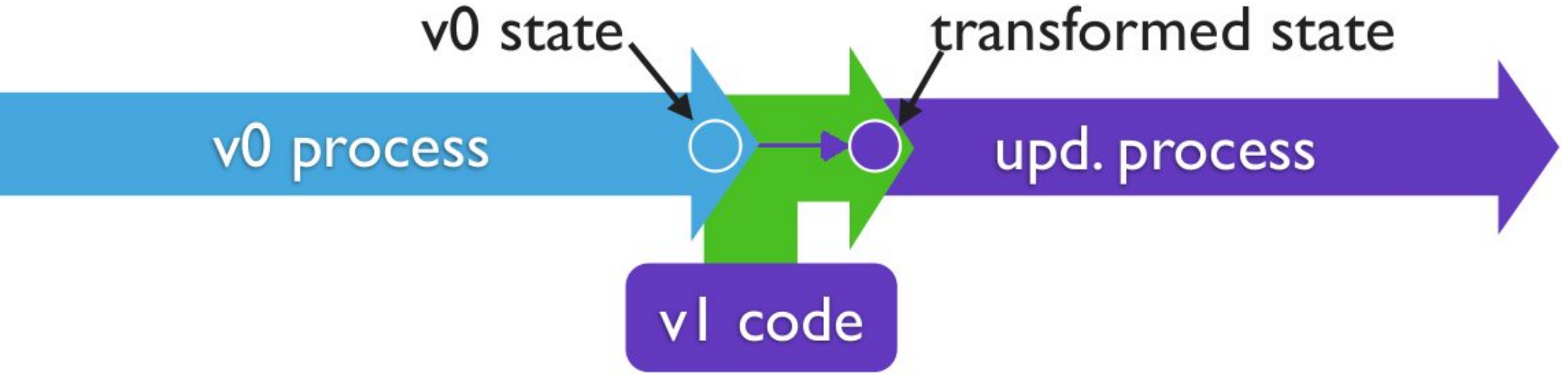
For traffic at 30 pkts/s

- C-Strider
  - General framework for writing heap traversals based on xfgen
  - Parallel updating, serialization, heap checking, etc

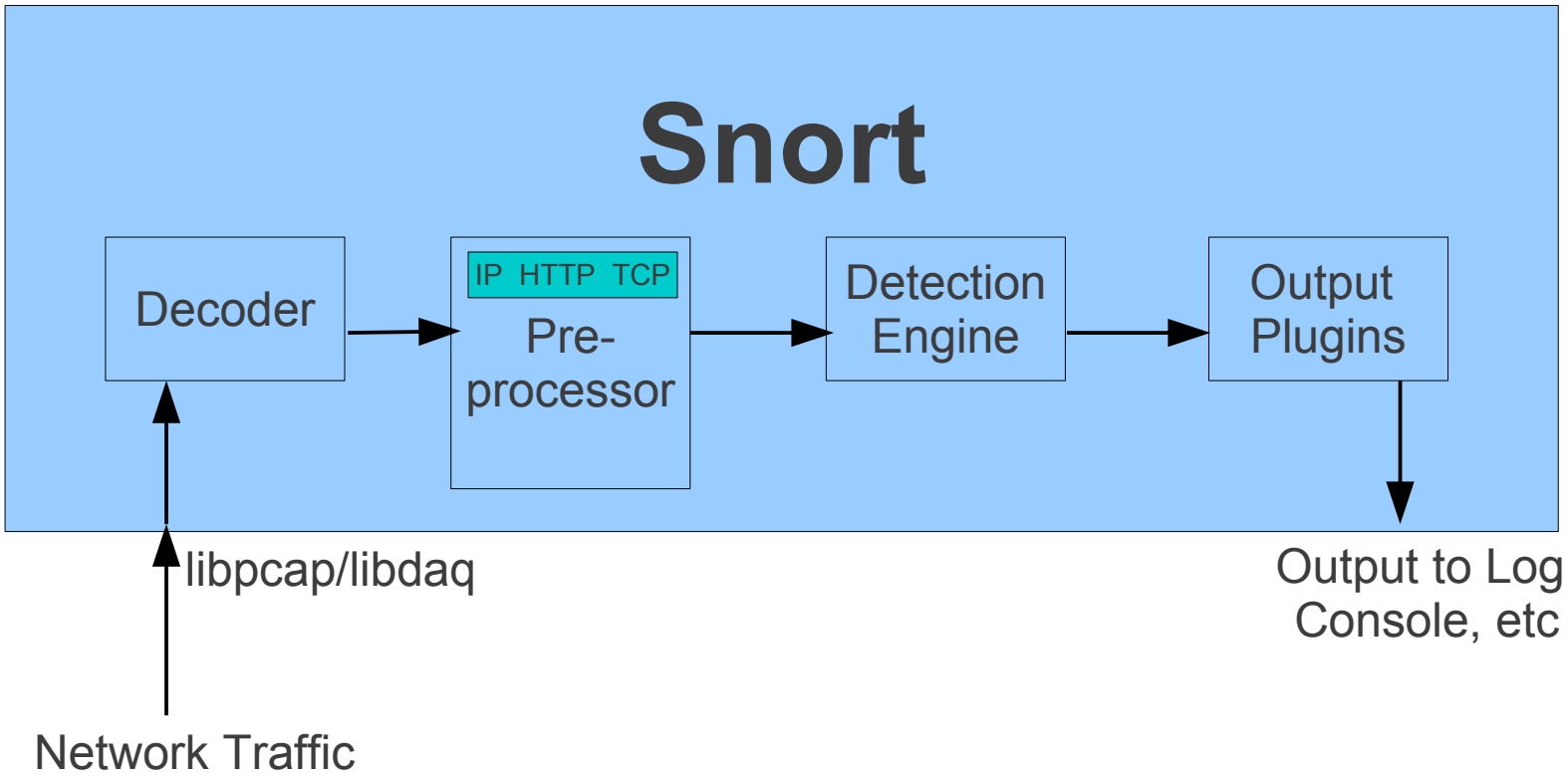
(Backup Slides)

DSU: Dynamic Software Updating

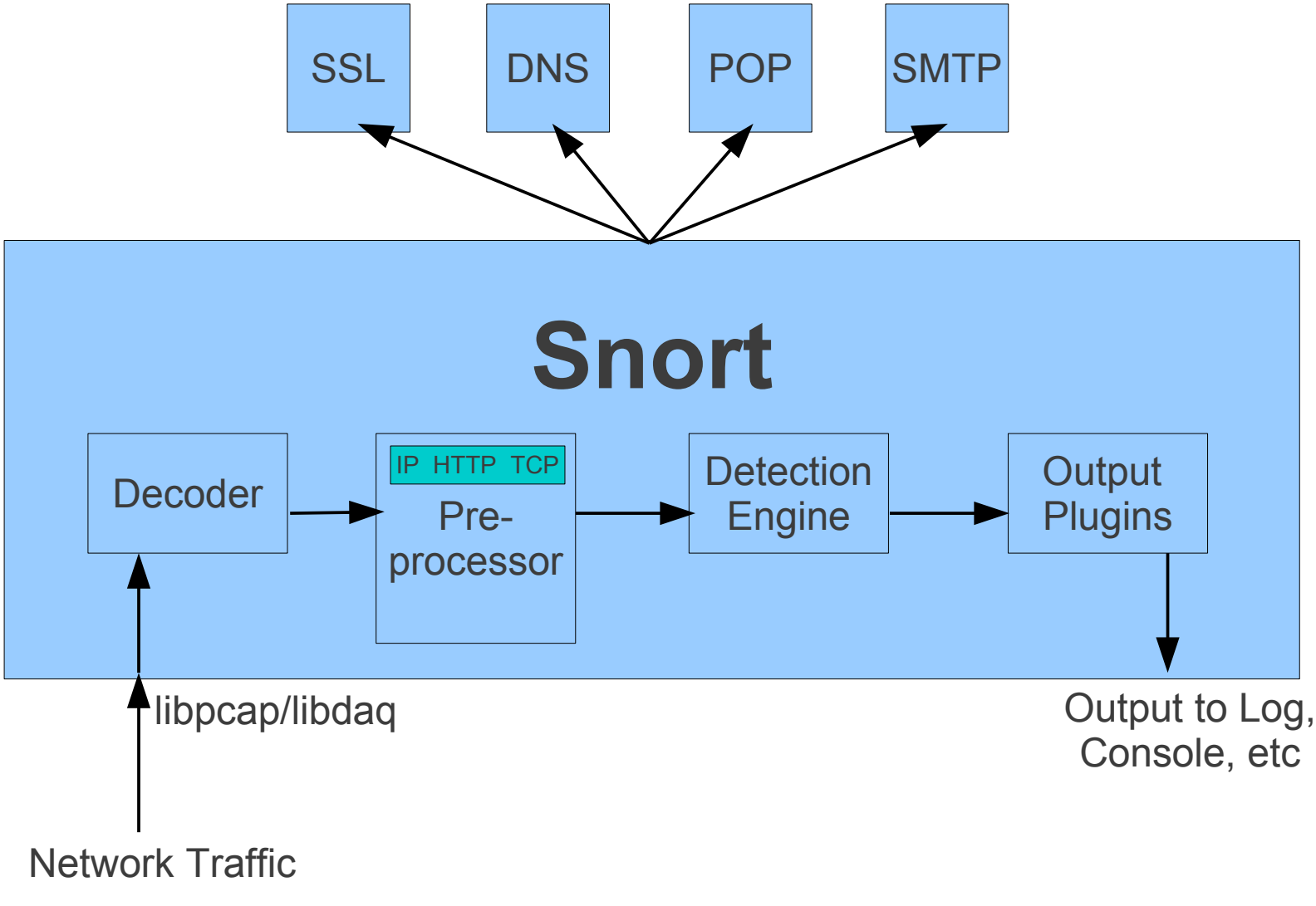
- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...



Snort: Main Components



Snort: Main Components

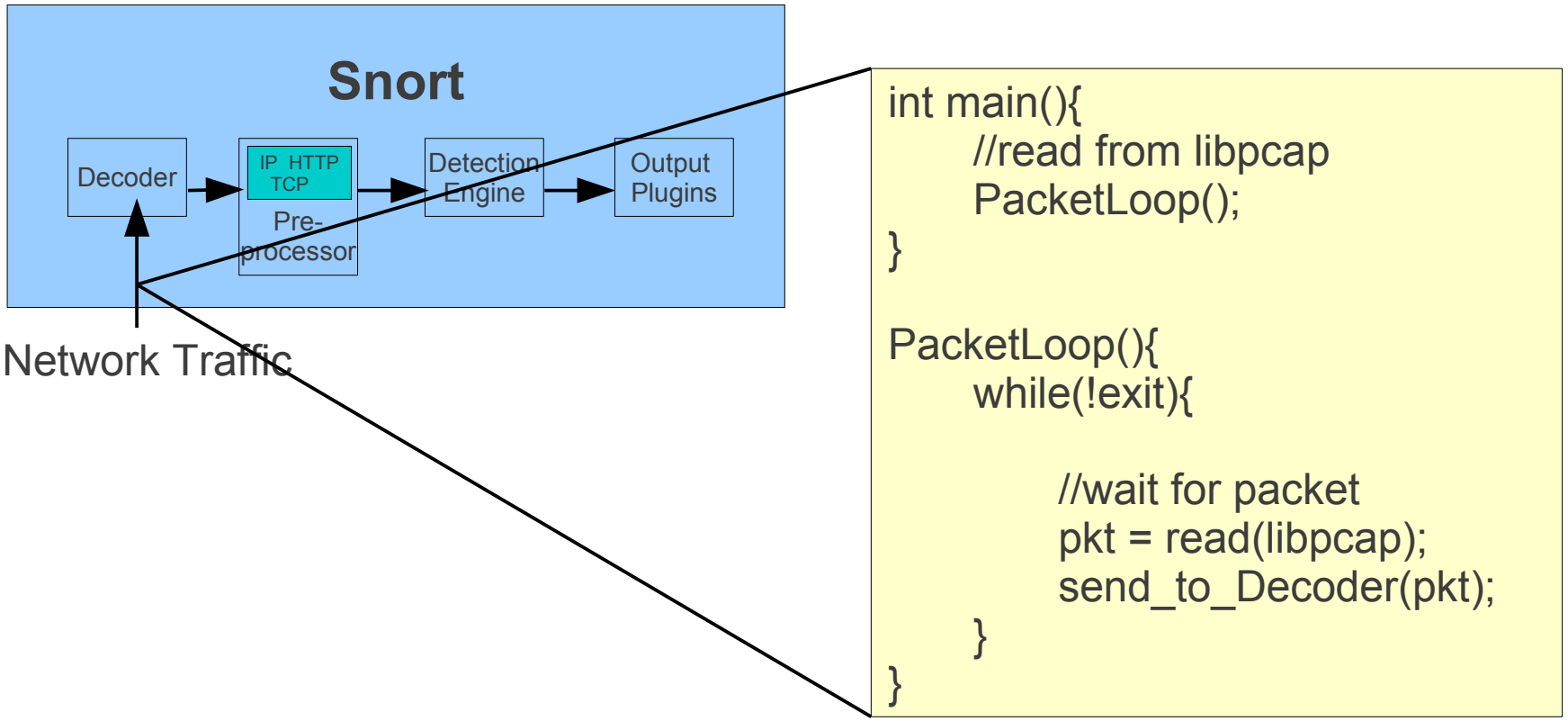


Programmer Obligations

- Kitsune DSU requires the programmer to
  - Choose update points: identify where updates may take place

Programmer Obligations:

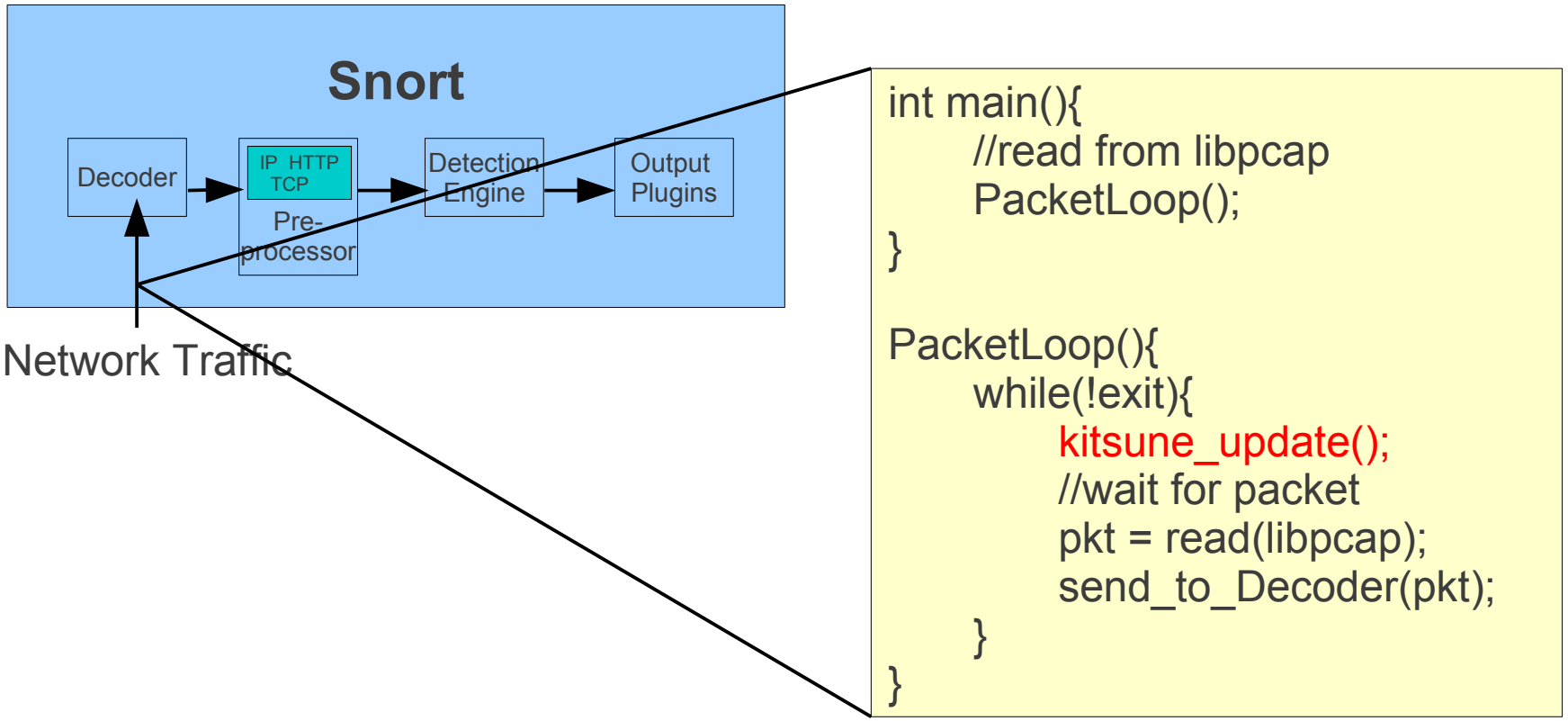
Update Point



Update: between packet reads – has least amount of in-flight state
Resume: Read next packet from libpcap

Programmer Obligations:

Update Point



Update: between packet reads – has least amount of in-flight state
Resume: Read next packet from libpcap

Programmer Obligations

- Kitsune DSU requires the programmer to
  - Choose update points: identify where updates may take place
  - Control Migration: Transfer control to the right event loop when the new version restarts

Programmer Obligations

- Kitsune DSU requires the programmer to
  - Choose update points: identify where updates may take place
  - Control Migration: Transfer control to the right event loop when the new version restarts
  - Data Migration: Identify the state to be transformed, and where it should be received in the new code

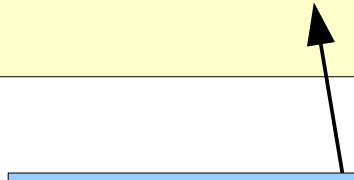
Programmer Obligations:

Data Migration

```
char *snort_conf_dir = NULL;
SnortConfig *snort_conf = NULL;

int main(){

        if(!kitsune_is_updating()){  // only perform initialization if not updating
            SnortInit();
        }
        //read from libpcap
        PacketLoop();
}

void SnortInit(){
        //initialize main SnortConfig structure
}

void PacketLoop(){
        while(!exit){
                kitsune_update();
                //wait for packet
                pkt = read(libpcap);
                send_to_Decoder(pkt);
        }
}
```

xfgen: Annotations

```
int proto_node_num;
. . .
struct _RuleTreeNode ** E_PTRARRAY(proto_node_num)
proto_nodes;
```



This is a pointer to an array of size proto_node_num

xfgen rules:
Initializing New Fields

Snort 2.9.2.1:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Snort 2.9.2.2:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint32_t maxNumDialogsInSession;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Initialize *new* variable or values of a new type:

xfgen rules:
Initializing New Fields

Snort 2.9.2.1:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Snort 2.9.2.2:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint32_t maxNumDialogsInSession;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Initialize *new* variable or values of a new type:

```
INIT struct _sipConfig.maxNumDialogsInSession:{ $out = 4;}
```

xfgen rules:
Initializing New Fields

Snort 2.9.2.1:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Snort 2.9.2.2:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint32_t maxNumDialogsInSession;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Initialize *new* variable or values of a new type:

```
INIT struct _sipConfig.maxNumDialogsInSession:{ $out = 4;}
```

Initialize

xfgen rules:
Initializing New Fields

Snort 2.9.2.1:

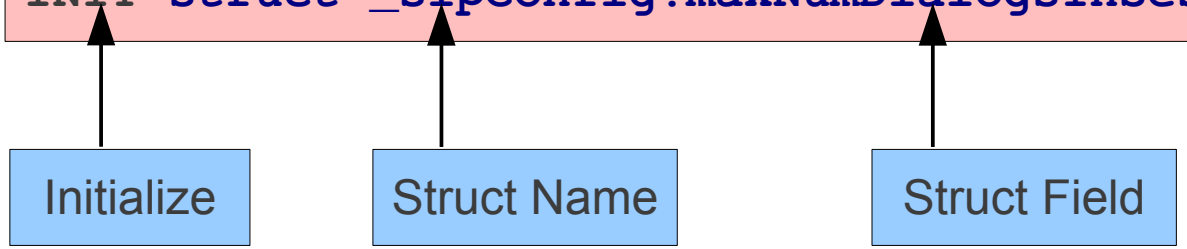
```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Snort 2.9.2.2:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint32_t maxNumDialogsInSession;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Initialize *new* variable or values of a new type:

```
INIT struct _sipConfig.maxNumDialogsInSession:{ $out = 4;}
```



xfgen rules:
Initializing New Fields

Snort 2.9.2.1:

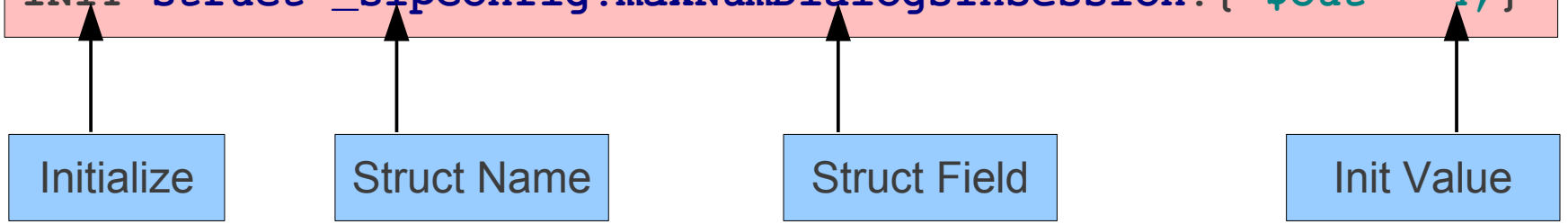
```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Snort 2.9.2.2:

```
typedef struct _sipConfig
{
    uint8_t  disabled;
    uint32_t maxNumSessions;
    uint32_t maxNumDialogsInSession;
    uint8_t  ports[MAXPORTS/8];
    ....
} SIPConfig;
```

Initialize *new* variable or values of a new type:

```
INIT struct _sipConfig.maxNumDialogsInSession:{ $out = 4;}
```



xfgen rules:
Transforming Function Pointers

```
typedef struct OutputFuncNode{
   union {
      OutputFunc fptr;
      void *vfptr ;
   } fptr ;
   struct OutputFuncNode *next;
} OutputFuncNode;
```

Transform *old* function pointers to point to the new version function:

xfgen rules:
Transforming Function Pointers

```
typedef struct OutputFuncNode{
   union {
      OutputFunc fptr;
      void *vfptr ;
   } fptr ;
   struct OutputFuncNode *next;
} OutputFuncNode;
```

Transform *old* function pointers to point to the new version function:

```
struct _OutputFuncNode.fptr -> struct _OutputFuncNode.fptr:
{XF_INVOKE(XF_FPTR(), &$in.vfptr, &$out.vfptr);}
```

xfgen rules:
Transforming Function Pointers

```
typedef struct OutputFuncNode{
   union {
      OutputFunc fptr;
      void *vfptr ;
   } fptr ;
   struct OutputFuncNode *next;
} OutputFuncNode;
```

Transform *old* function pointers to point to the new version function:

```
struct _OutputFuncNode.fptr -> struct _OutputFuncNode.fptr:
{XF_INVOKE(XF_FPTR(), &$in.vfptr, &$out.vfptr);}
```

Call the following:

xfgen rules:
Transforming Function Pointers

```
typedef struct OutputFuncNode{
   union {
      OutputFunc fptr;
      void *vfptr ;
   } fptr ;
   struct OutputFuncNode *next;
} OutputFuncNode;
```

Transform *old* function pointers to point to the new version function:

```
struct _OutputFuncNode.fptr -> struct _OutputFuncNode.fptr:
{XF_INVOKE(XF_FPTR(), &$in.vfptr, &$out.vfptr);}
```

Call the following:

Change this function pointer,

xfgen rules:
Transforming Function Pointers

```
typedef struct OutputFuncNode{
   union {
      OutputFunc fptr;
      void *vfptr ;
   } fptr ;
   struct OutputFuncNode *next;
} OutputFuncNode;
```

Transform *old* function pointers to point to the new version function:

```
struct _OutputFuncNode.fptr -> struct _OutputFuncNode.fptr:
{XF_INVOKE(XF_FPTR(), &$in.vfptr, &$out.vfptr);}
```

Call the following:

Change this function pointer,

from v0 function address

to v1 function address

Normal Execution Overhead

- Ran Snort 2.9.2 over a folder of various pre-recorded packet capture files
  - Found that processing speed varied ±11% depending on packet capture file size, so tested over a folder of files ranging from 343 - 304,298 packets
  - Essentially no overhead

Program	**Orig** (siqr)	**Kitsune**
64-bit, 4×2.4Ghz E7450 (6 core), 24GB mem		
snort 2.9.2	23.09s (0.50s)	-1.74%
32-bit, 1×3.6Ghz Pentium D (2 core), 2GB mem		
snort 2.9.2	10.42s (0.50s)	+1.73%

Snort Versions - Update Highlights

- **Started with Snort 2.9.2 (released December 14, 2011)**
  - First step is to migrate 2.9.2 → 2.9.2 to cover the majority of the state transfer

- **Snort 2.9.2.1 - Jan 19, 2012 (changes in ~40 src files)**
  - Added a preprocessor alert to alert when a HTTP method being parsed is not a GET or a POST or not defined by the user.
  - server/hi_server.c: Added checking bounds before unfolding.
  - Fixed a bug where the DNP3 preprocessor would generate alerts for "reserved function" on valid DNP3 functions.

Snort Versions - Update Highlights

- **Snort 2.9.2.2 - Mar 27, 2012 (changes in ~80 src files)**
  - Fix stream5 to not purge too early when normalizing streams.
  - Fix overhead calculation to ensure sufficient buffer space for defragging a maximum length IP datagram regardless of encapsulations.
  - Eliminate false positives (no content-length or transfer-encoding) when chunk size spans across multiple packets
- **Snort 2.9.2.3 - May 15, 2012 (changes in ~20 src files)**
  - Add stricter checking on packets before processing by dnp3 and check reassembly buffer size to avoid overrun
  - Pcre Ovector size needs to be dynamic to compensate for pcre-8.30 segfault.