

# CSSV: Towards a Realistic Tool for Statically Detecting **All** Buffer Overflows in C

**Nurit Dor**, Michael Rodeh, Mooly Sagiv

PLDI'2003

*DAEDALUS project*

# Vulnerabilities of C programs

```
/* from web2c [strupascal.c] */
```

```
void foo(char *s)
```

```
{
```

```
while ( *s != ' ' )
```

*Null dereference*

*Dereference to unallocated storage*

```
    s++
```

*Out of bound pointer arithmetic*

```
    *s = 0;
```

*Out of bound update*

```
}
```

# Is it common?

- General belief – yes!
- FUZZ study
  - Test reliability by random input
  - Tens of applications on 9 different UNIX systems
  - 18% – 23% hang or crash
- CERT advisory
  - Up to 50% of attacks are due to buffer overflow

***COMMON AND DANGEROUS***

# CSSV's Goals

- Efficient **conservative** static checking algorithm
  - **Verify the absence of buffer overflow --- not just finding bugs**
  - All C constructs
    - Pointer arithmetic, casting, dynamic memory, ...
  - Real programs
  - Minimum false alarms

# Complicated Example

---

```
/* from web2c [fixwrites.c] */
```

```
#define BUFSIZ 1024
```

```
char buf[BUFSIZ];
```

```
char insert_long(char *cp)  
{
```

```
    char temp[BUFSIZ];
```

```
    ...
```

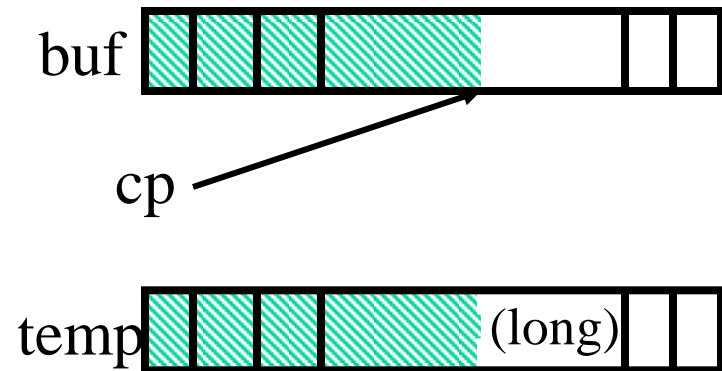
```
    for (i = 0; &buf[i] < cp ; ++i)
```

```
        temp[i] = buf[i];
```

```
    strcpy(&temp[i], "(long)");
```

```
    strcpy(&temp[i+6], cp);
```

```
    ...
```



# Complicated Example

---

```
/* from web2c [fixwrites.c] */
```

```
#define BUFSIZ 1024
```

```
char buf[BUFSIZ];
```

```
char insert_long(char *cp)  
{
```

```
    char temp[BUFSIZ];
```

```
    ...
```

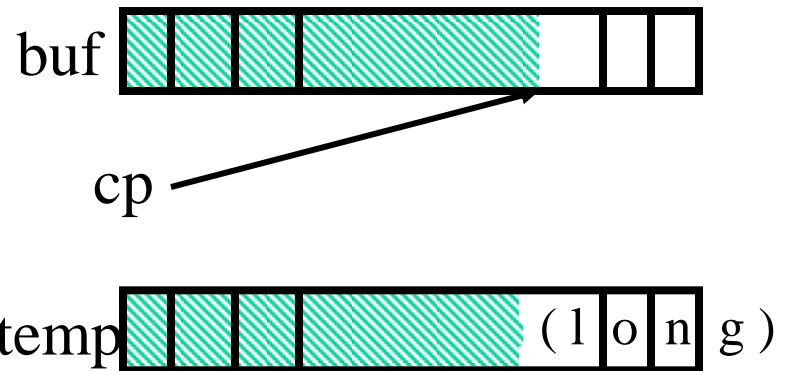
```
    for (i = 0; &buf[i] < cp ; ++i)
```

```
        temp[i] = buf[i];
```

```
    strcpy(&temp[i], "(long)");
```

```
    strcpy(&temp[i+6], cp);
```

```
    ...
```



Cleanness is potentially violated:

$$7 + \text{offset}(cp) \geq \text{BUFSIZ}$$

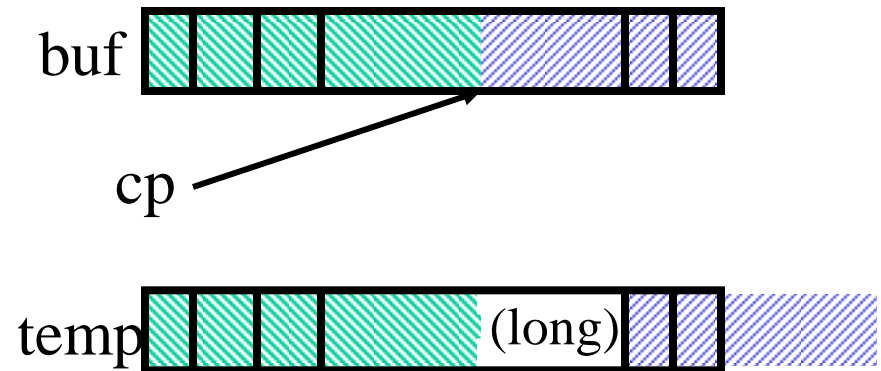
# Complicated Example

---

```
/* from web2c [fixwrites.c] */
#define BUFSIZ 1024
char  buf[BUFSIZ];

char insert_long(char *cp)
{
    char  temp[BUFSIZ];
    ...
    for (i = 0; &buf[i] < cp ; ++i)
        temp[i] = buf[i];

    strcpy(&temp[i], "(long)");
    strcpy(&temp[i+6], cp);
    ...
}
```



Cleanness is potentially violated:  
 $\text{offset}(cp) + 7 + \text{len}(cp) \geq \text{BUFSIZ}$   
 $7 + \text{offset}(cp) < \text{BUFSIZ}$

# Verifying Absence of Buffer Overflow is non-trivial

```
void safe_cat(char *dst, int size, char *src )  
{  
    {  
        string(src)  $\wedge$  string(dst)  $\wedge$   
        (size > len(src)+len(dst))  $\Rightarrow$  alloc(dst+len(dst)) > len(src)  
    }  
  
    if ( size > strlen(src) + strlen(dst) )  
    {  
        {string(src)  $\wedge$  string(dst)  $\wedge$  alloc(dst+len(dst)) > len(src)}  
        dst = dst + strlen(dst);  
  
        {string(src)  $\wedge$  alloc(dst) > len(src)}  
  
        strcpy(dst, src);  
    }  
}
```



# Can this be done for real programs?

- Complex linear relationships
- Pointer arithmetic
- Loops
- Procedures
- Use Polyhedra[CH78]
- Pointer analysis
- Widening
- Procedure contracts

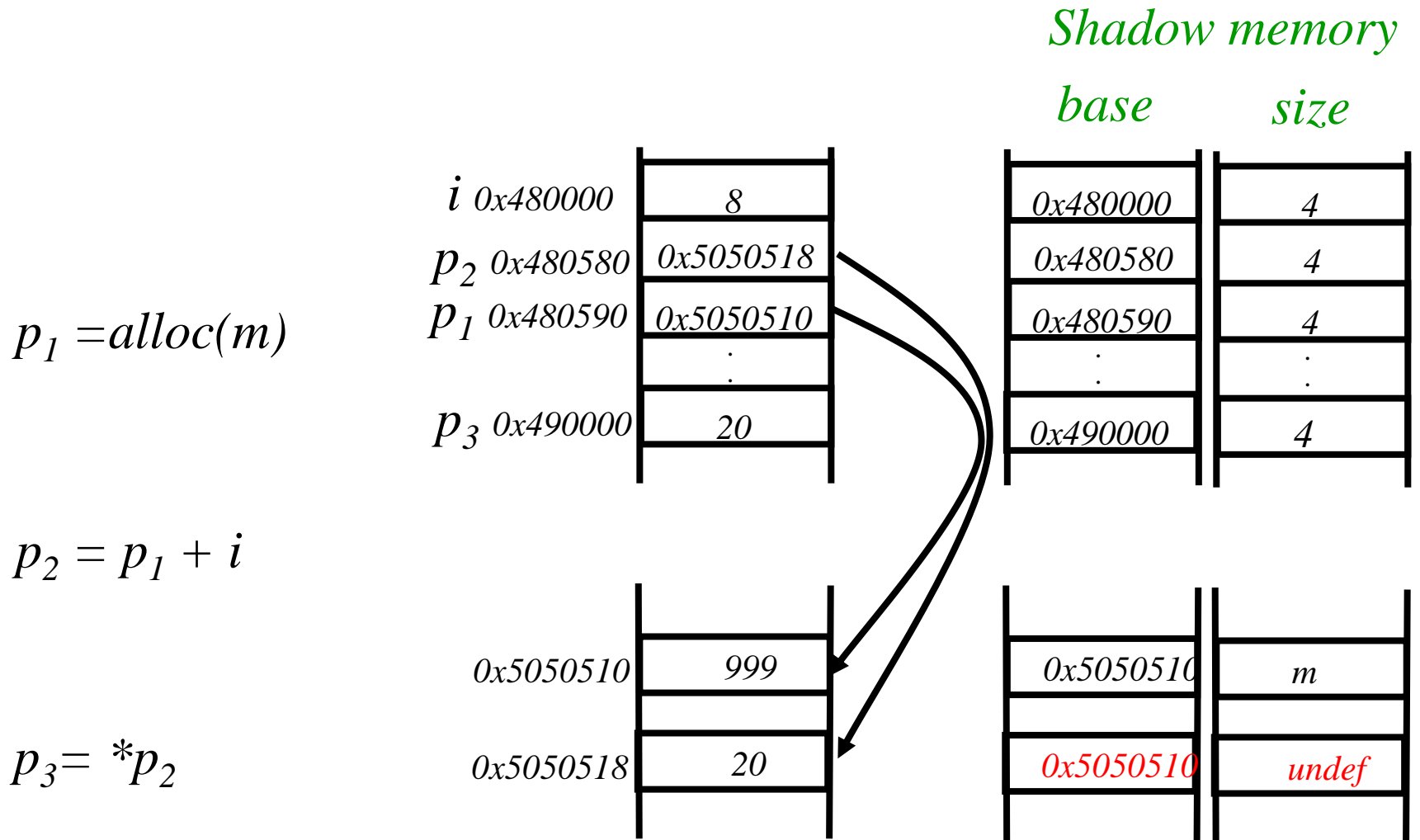
*Very few false alarms!*

# C String Static Verifier

- Detects string violations
  - Buffer overflow (update beyond bounds)
  - Unsafe pointer arithmetic
  - References beyond null termination
  - Unsafe library calls
- Handles full C
  - Multi-level pointers, pointer arithmetic, structures, casting, ...
- Applied to real programs
  - Public domain software
  - C code from Airbus

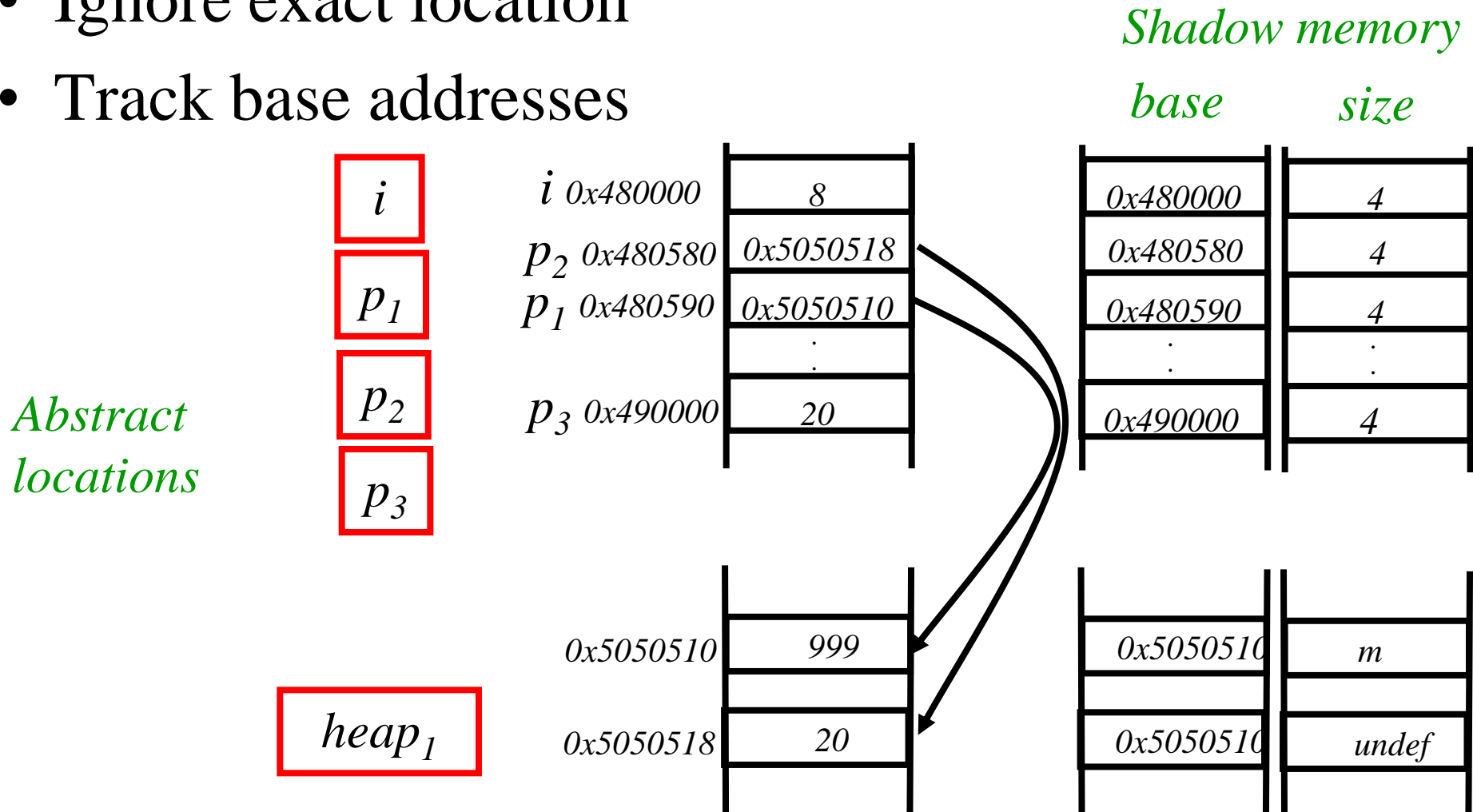


# Operational Semantics



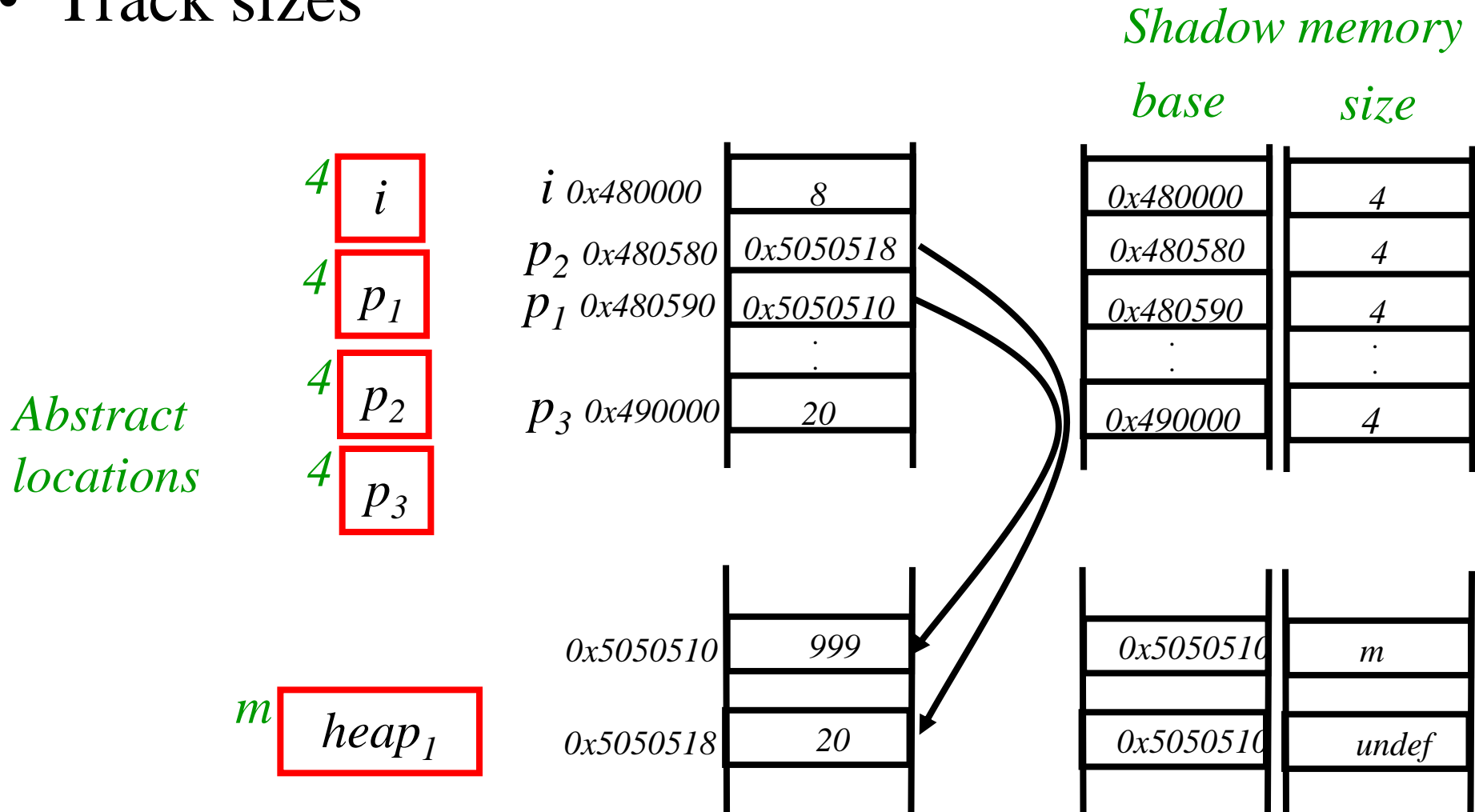
# CSSV's Abstraction

- Ignore exact location
- Track base addresses



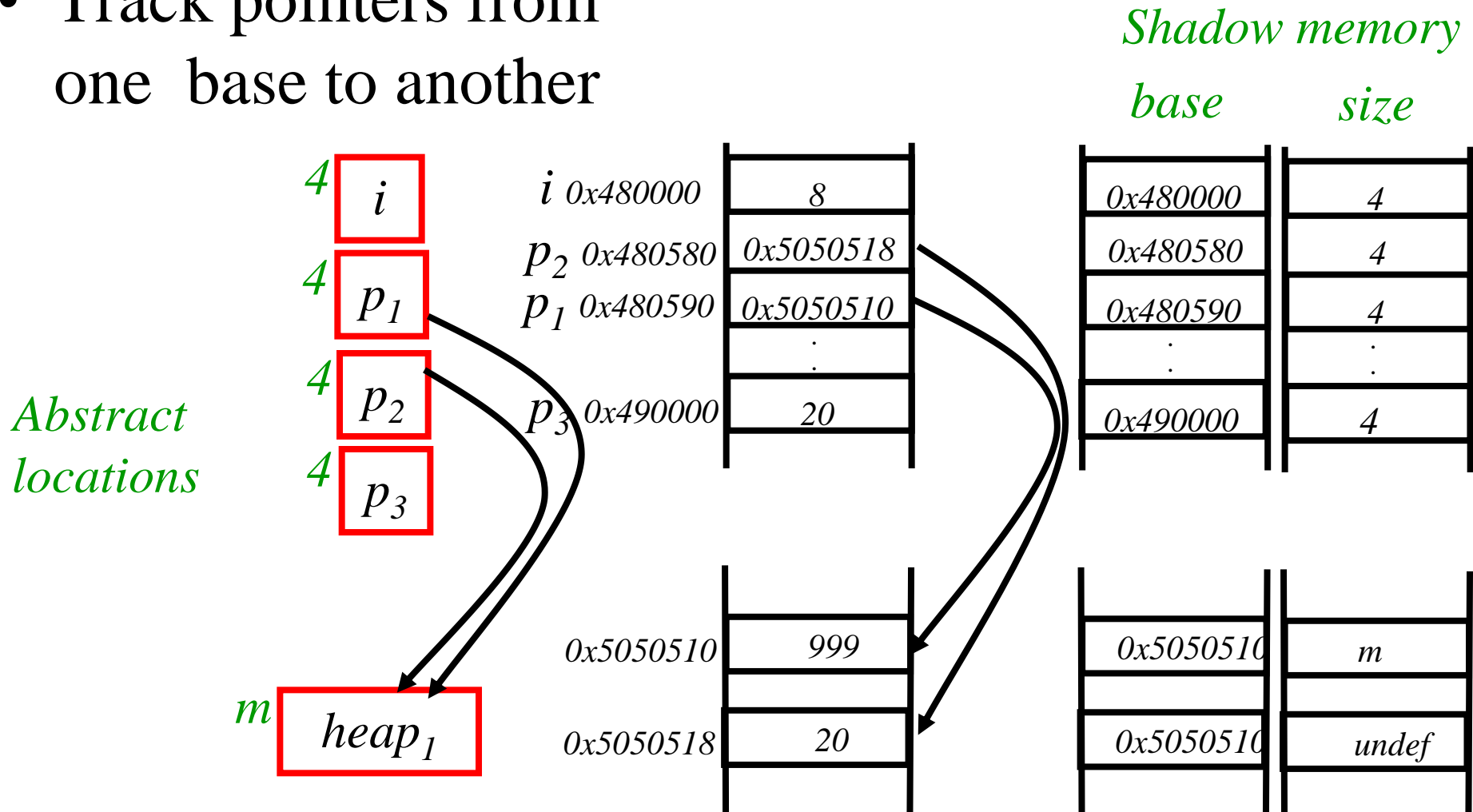
# CSSV's Abstraction

- Track sizes



# CSSV's Abstraction

- Track pointers from one base to another

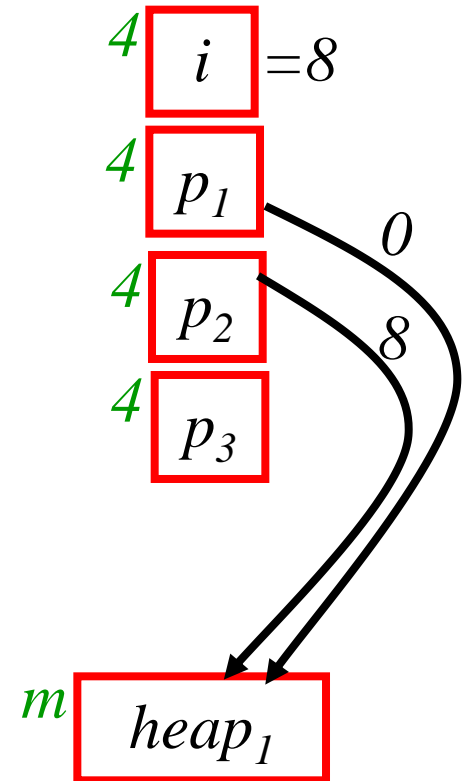


# Pointer Validation

- How can we validate pointer arithmetic?

$$p_2 = p_1 + i$$

- Track offsets from origin
- Track numeric values

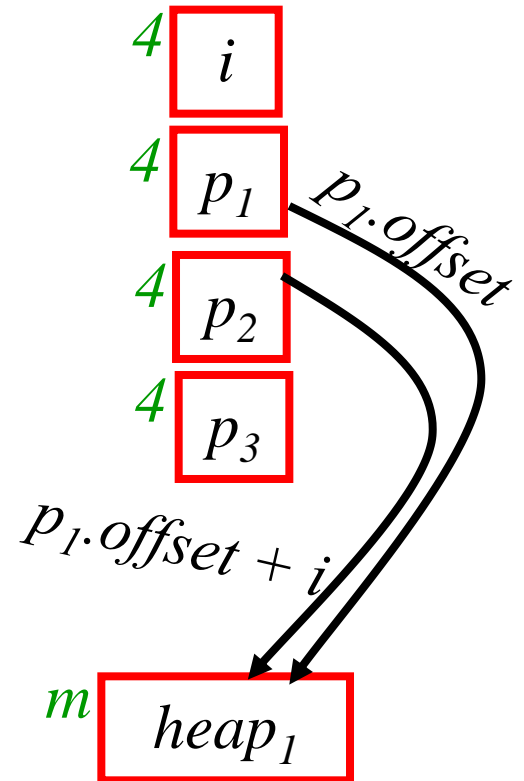


# Numeric values are unknown

- Track integer relationships

$$p_2 = p_1 + i$$

$$p_2.\text{offset} = p_1.\text{offset} + i$$





# Validation

- Pointer arithmetic

$$p_2 = p_1 + i$$

$$*p_1.size \geq p_1.offset + i$$

- Pointer dereference

$$p_3 = *p_2$$

$$*p_2.size \geq p_2.offset$$

# The null-termination byte

- Many expressions involve the '\0' byte

*strcpy(dst, src)*

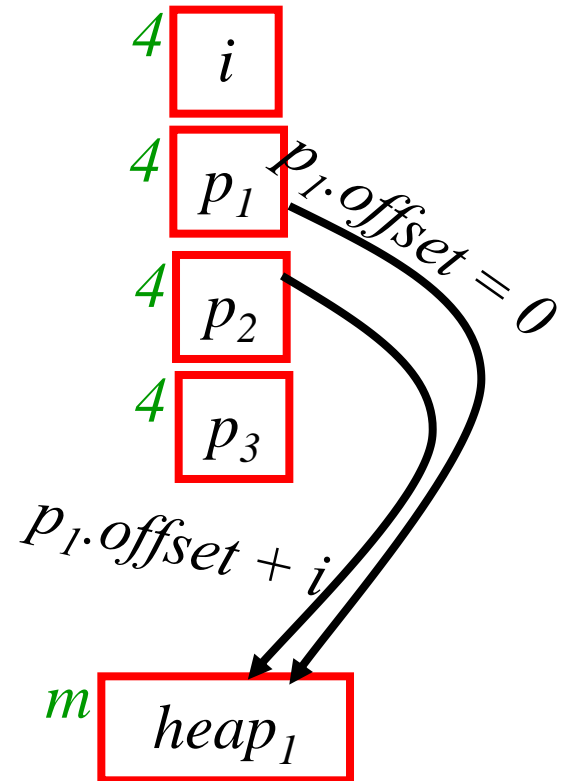
- Track the existence of null-termination
- Track the index of the first one

# Abstract Transformers

- Defines the effect of statements on the abstract representation

$p_1 = \text{alloc}(m)$

$p_2 = p_1 + i$



# Abstract Transformers

- Unknown value

$$p_3 = *p_2$$

$$p_3 = 0$$

$$p_3 = \text{unknown}$$

$$\left. \begin{array}{l} *p_2.is\_nullt \wedge *p_2.len == p_2.offset \\ otherwise \end{array} \right\}$$

# Overly Conservative

- Representing infeasible concrete states
- Infeasible pointer aliases
- Infeasible integer variables

# Procedure Calls – Contracts

`char* strcpy(char* dst, char *src)`

**requires** ( `string(src) ^`  
`alloc(dst) > len(src)`  
)

**mod** `len(dst), is_nullt(dst)`

**ensures** ( `len(dst) == pre@len(src) ^`  
`return == pre@dst`  
)

# Advantages of Procedure Contracts

- Modular analysis
  - Not all the code is available
  - Enables more expensive analyses
- User control of the verification
  - Detect errors at point of logical error
  - Improve the precision of the analysis
  - Check additional properties
    - Beyond ANSI-C

# Specification and Soundness

- All errors are detected
- Violation of procedure's precondition
  - Call
- Violation of procedure's postcondition
  - Return
- Violation of statement's precondition
  - ...a[i]...



# Procedure Calls – Contracts

`char* strcpy(char* dst, char *src)`

**requires** ( `string(src) ^`  
`alloc(dst) > len(src)`  
)

**mod** `len(dst), is_nullt(dst)`

**ensures** ( `len(dst) == pre@len(src) ^`  
`return == pre@dst`  
)

# safe\_cat's contract

```
void safe_cat(char* dst, int size, char* src)
```

```
  requires  ( string(src) ∧ string(dst)  
              alloc(dst) == size  
            )
```

```
  mod      dst
```

```
  ensures  ( len(dst) ≤ [len(src)]pre +  
              [len(dst)]pre ∧  
              len(dst) ≥ [len(dst)]pre  
            )
```

# Specification – insert\_long()

```
/* insert_long.c */
#include "insert_long.h"
char buf[BUFSIZ];
char * insert_long (char *cp) {
    char temp[BUFSIZ];
    int i;
    for (i=0; &buf[i] < cp; ++i){
        temp[i] = buf[i];
    }
    strcpy (&temp[i], "(long)");
    strcpy (&temp[i + 6], cp);
    strcpy (buf, temp);
    return cp + 6;
}
```

```
char * insert_long(char *cp)
    requires( string(cp) ^
              buf ≤ cp < buf + BUFSIZ
            )
    mod cp.strlen
    ensures (
        len(cp) == pre[len(cp) + 6]
            ^
            return_value == cp + 6 ;
    )
```

# Complicated Example

---

```
/* from web2c [fixwrites.c] */
```

```
#define BUFSIZ 1024
```

```
char buf[BUFSIZ];
```

```
char insert_long(char *cp)  
{
```

```
    char temp[BUFSIZ];
```

```
    ...
```

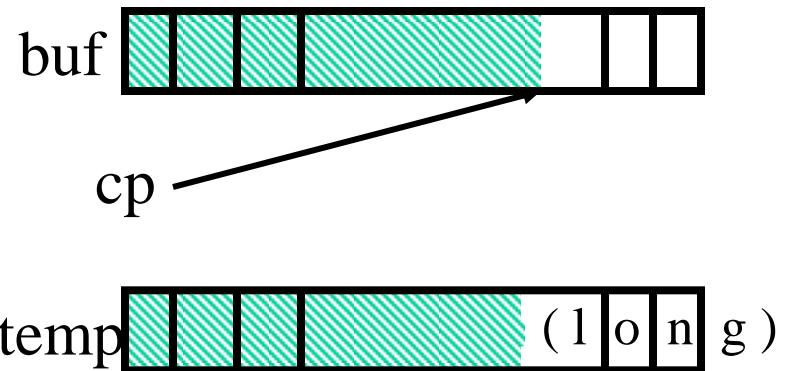
```
    for (i = 0; &buf[i] < cp ; ++i)
```

```
        temp[i] = buf[i];
```

```
    strcpy(&temp[i], "(long)");
```

```
    strcpy(&temp[i+6], cp);
```

```
    ...
```



Cleanness is potentially violated:

$$7 + \text{offset}(cp) \geq \text{BUFSIZ}$$

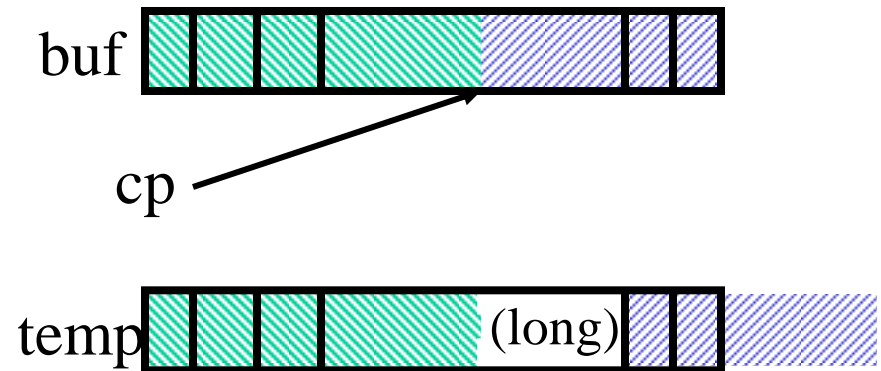
# Complicated Example

---

```
/* from web2c [fixwrites.c] */
#define BUFSIZ 1024
char  buf[BUFSIZ];

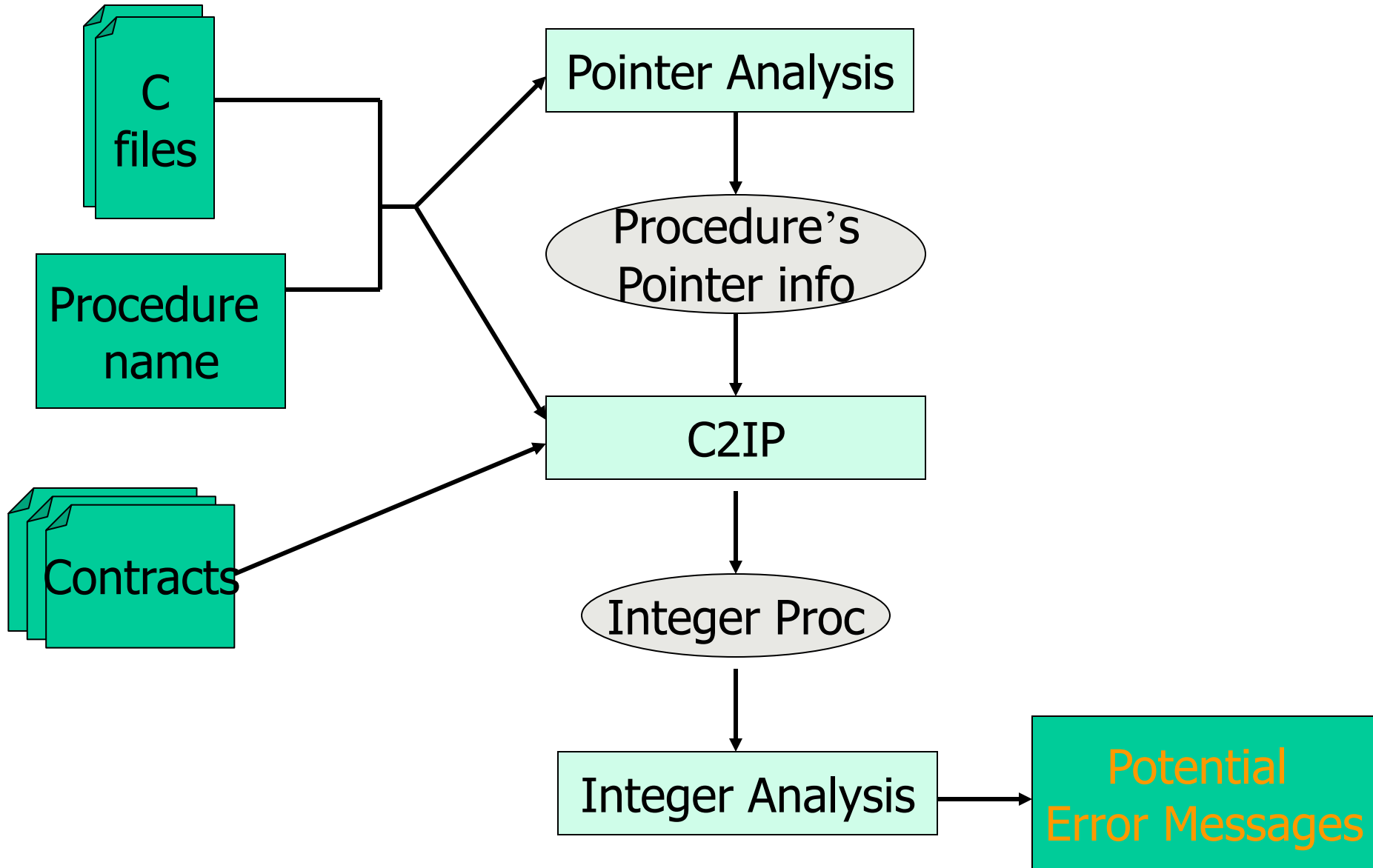
char insert_long(char *cp)
{
    char  temp[BUFSIZ];
    ...
    for (i = 0; &buf[i] < cp ; ++i)
        temp[i] = buf[i];

    strcpy(&temp[i], "(long)");
    strcpy(&temp[i+6], cp);
    ...
}
```



Cleanness is potentially violated:  
 $\text{offset}(cp) + 7 + \text{len}(cp) \geq \text{BUFSIZ}$   
 $7 + \text{offset}(cp) < \text{BUFSIZ}$

# CSSV – Technical overview



# Used Software

- ASToolKit [Microsoft]
- Core C [TAU - Greta Yorsh]
- GOLF [Microsoft - Manuvir Das]
- New Polka [Inria - Bertrand Jeannet]

# CSSV Static Analysis

1. Inline contracts
  - Expose behavior of called procedures
2. Pointer analysis (global)
  - Find relationship between base addresses
  - **Project into procedures**
3. Integer analysis
  - Compute offset information



# Preliminary results (web2C)

Proc	line	coreC line	time (sec)	space (Mb)	errors	FA
insert_long	14	64	2.0	13	2	0
fprintf_pascal_string	10	25	0.1	0.3	2	0
space_terminate	9	23	0.1	0.2	0	0
external_file_name	14	28	0.2	1.7	2	0
join	15	53	0.6	5.2	2	1
remove_newline	25	105	0.6	4.6	0	0
null_terminate	9	23	0.1	0.2	2	0

# Preliminary results (EADS/RTC\_Si)

Proc	line	coreC line	time (sec)	space (Mb)	errors	FA
FiltrerCarNonImp	19	34	1.6	0.5	0	0
SkipLine	12	42	0.8	1.9	0	0
StoreIntInBuffer	37	134	7.9	21	0	0

# CSSV: Summary

- Semantics
  - Safety checking
  - Full C
  - Enables abstractions
- Contract language
  - String behavior
  - Omit pointer aliasing
- Procedural points-to
  - Scalable
  - Improve precision
- Static analysis
  - Tracks important string properties
  - Utilizes integer analysis

# Related Projects

- SAL Microsoft
- Brian Hacket static analysis

# Conclusion

---

- 👍 Ambitious sound analyses
- 👍 Very few false alarms
- ❖ Scaling is an issue
  - Use staged analyses
  - Use modular analysis
  - Use encapsulation