PURDUE UNIVERSITY

SENG: An Enhanced Policy Language for SELinux

Paul Kuliniewicz
<kuliniew@purdue.edu>

CERIAS, Purdue University

CERIAS

Overview

- **What's wrong with macros?**
- Can we do better?
- The future...

What a Language Should Be

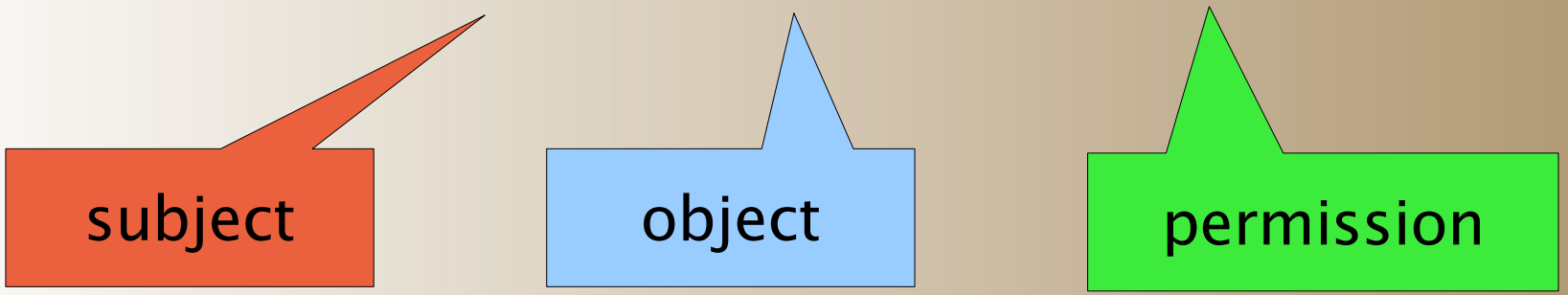
- Expressive
  - Can say what we want
- Succinct
  - Can say it briefly
- Analyzable
  - Well-defined semantics
- Natural
  - Reflects how we think

The Current Language

- **Expressive** and **analyzable**
  - We can write the desired policy
  - Statements have clear semantics
    - *(ignoring macros…)*
- Neither **succinct** nor **natural**
  - Each AV rule makes small changes
  - Need many rules to accomplish goals
  - Lower-level than we usually think

Anatomy of an AV Rule

```
allow foo_t bar_t:file getattr;
```



Access Matrix

	foo_t: file	foo_t: dir	bar_t: file	bar_t: dir
foo_t			read	
bar_t	create	create		
baz_t	create	create		

```
allow foo_t bar_t:file read;
allow {bar_t baz_t} foo_t:{file dir} create;
```

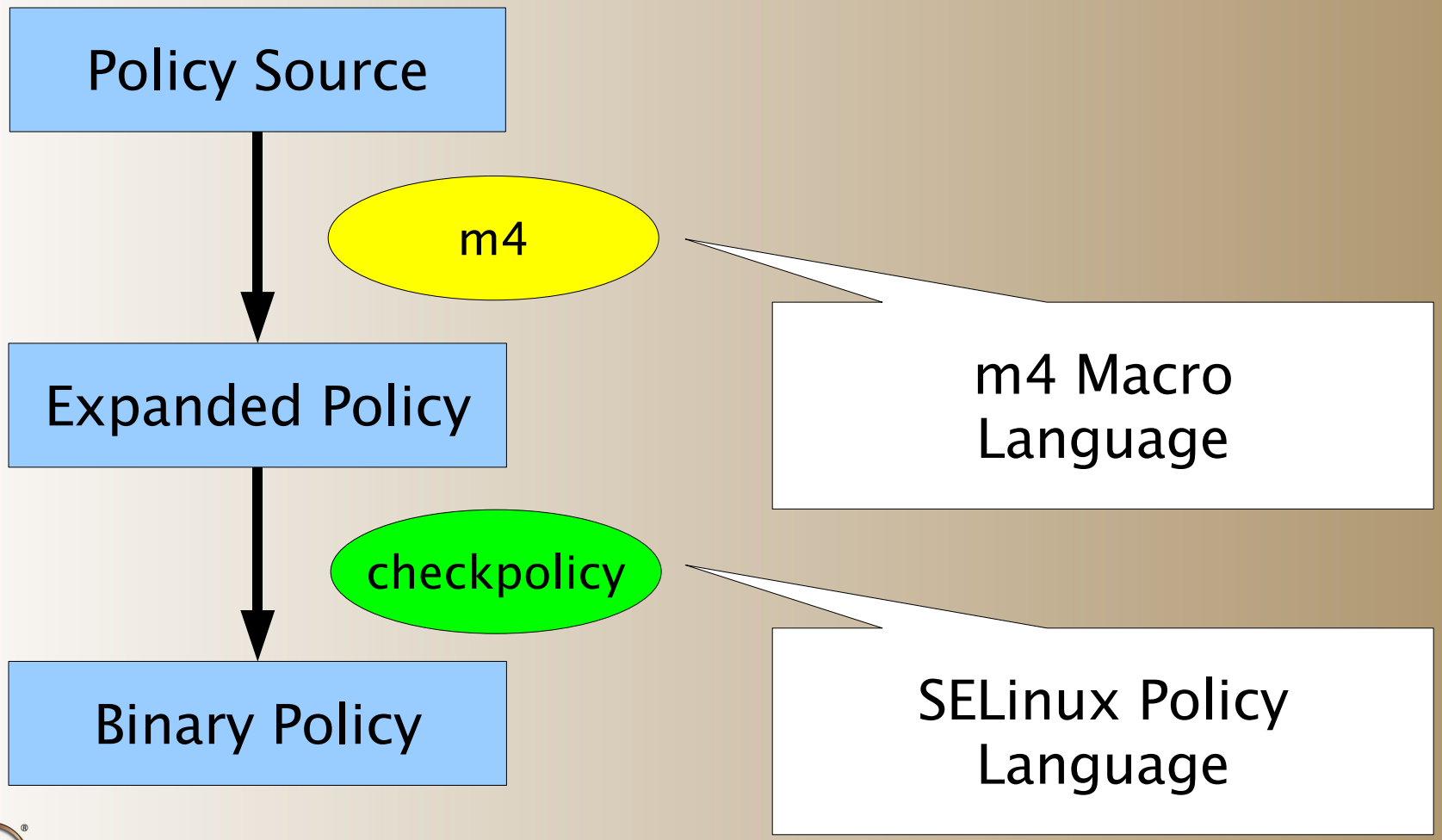
Quantifying Verbosity

- Monolithic example policy 1.26:
  - 2,024 types
  - 66,676 AV rules
  - 2,095 type transition rules

Macros

- **Succinct**
  - One macro can replace many rules
- Neither **analyzable** nor **natural**
  - Macro behavior is *unconstrained* by base language
  - Macros shoehorned into *all* abstractions needed by policy writer

Unconstrained



Simple Macro

```
define(`rw_dir_file', `
allow $1 $2:dir rw_dir_perms;
allow $1 $2:file rw_file_perms;
allow $1 $2:lnk_file { getattr read };
')
```

`rw_dir_file(foo_t, bar_t)` generates ...?

Complex Macro

```
define(`can_create_internal', `
ifelse(`$3', `dir', `
allow $1 $2:$3 create_dir_perms;
', `$3', `lnk_file', `
allow $1 $2:$3 create_lnk_perms;
', `
allow $1 $2:$3 create_file_perms;
')')

define(`can_create', `
ifelse(regexp($3, `\w'), -1, `', `
can_create_internal($1, $2, regexp($3, `\(\w+\)', `\1'))
can_create($1, $2, regexp($3, `\w+\(.*\)', `\1'))
')')
```

`can_create(foo_t, bar_t, `{dir file}')` generates...?

Unnatural

- `uses_shlib(`**`foo_t`**`)`
  - assigns permissions to `foo_t`
- `tmp_domain(`**`foo`**`)`
  - also assigns permissions to `foo_t`
- Both operate on `foo_t`
  - Leaky abstraction
- Neither looks like an AV rule

Overview

- What's wrong with macros?
- **Can we do better?**
- The future...

Introducing SENG

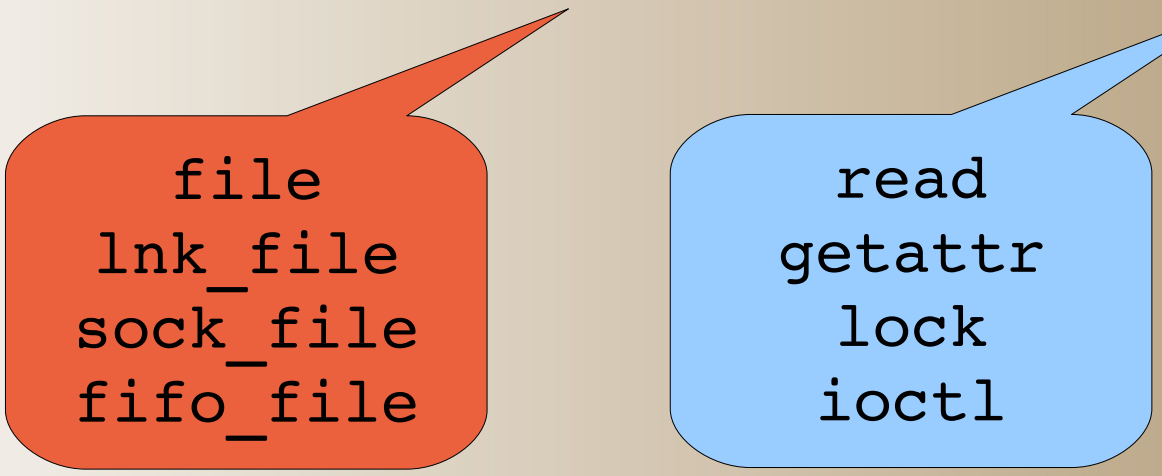
- Experimental alternative policy language
- Replaces macros with **well-defined abstractions**
  - Easier to read
  - Easier to write
  - Easier to analyze

Features

- **Class and permission sets**
- Abstract resources
- Abstract permissions
- Templates
- Abstract type transitions
  - All of these currently implemented ad-hoc using m4

Class and Permission Sets

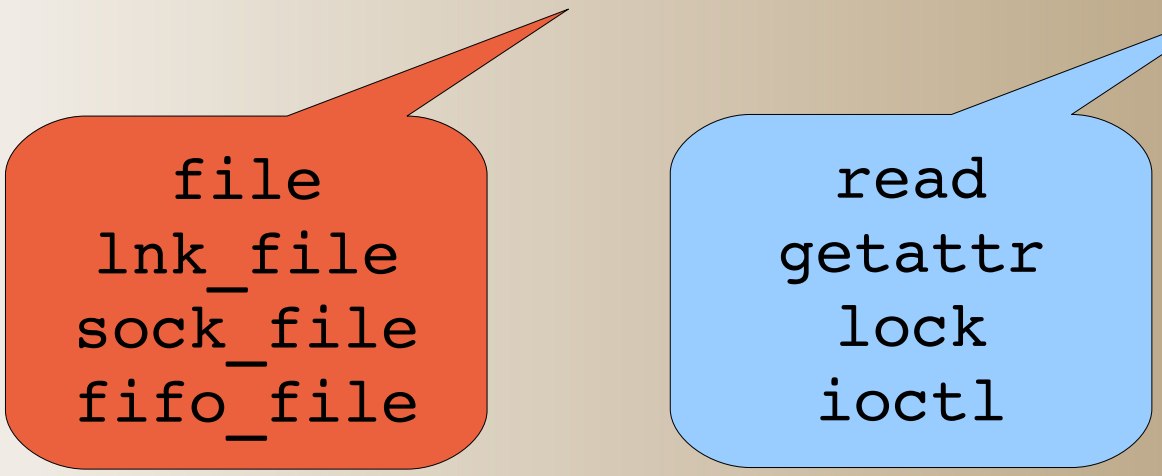
```
allow foo_t bar_t:notdevfile_class_set r_file_perms;
```




```
define(`notdevfile_class_set', `{ file lnk_file ... }')
define(`r_file_perms', `{ read getattr ... }')
```

Class and Permission Sets

```
allow foo_t bar_t:notdevfile_class_set r_file_perms;
```



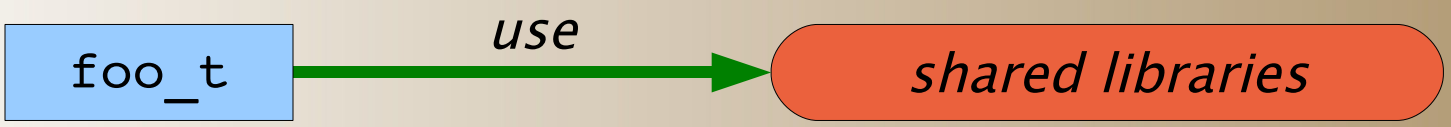

```
classset notdevfile_class_set { file lnk_file ... };
permset r_file_perms { read getattr ... };
```

Features

- Class and permission sets
- **Abstract resources**
- Abstract permissions
- Templates
- Abstract type transitions

Abstract Resources

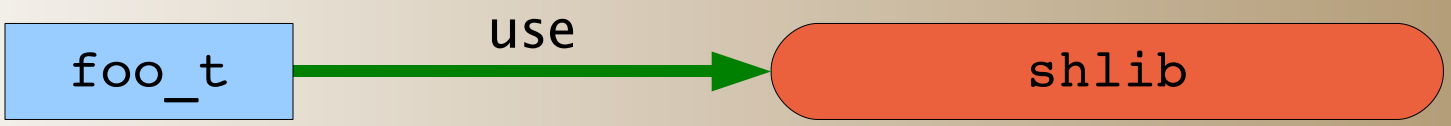
```
uses_shlib(foo_t)
```




```
define(`uses_shlib', `
allow $1 { root_t usr_t lib_t etc_t }:dir r_dir_perms;
allow $1 lib_t:lnk_file r_file_perms;
allow $1 ld_so_t:file rx_file_perms;
...
')
```

Abstract Resources

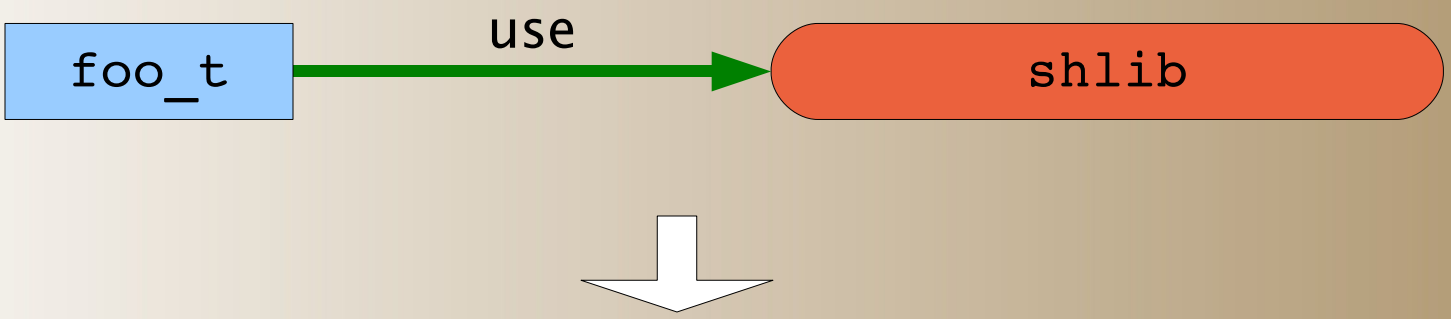
```
allow foo_t shlib use;
```




```
resource shlib { use };
permission shlib use ($dom)
{
  allow $dom { root_t usr_t lib_t etc_t }:dir r_dir_perms;
  allow $dom lib_t:lnk_file r_file_perms;
  allow $dom ld_so_t:file rx_file_perms;
  ...
};
```

Abstract Resources

```
allow foo_t shlib use;
```



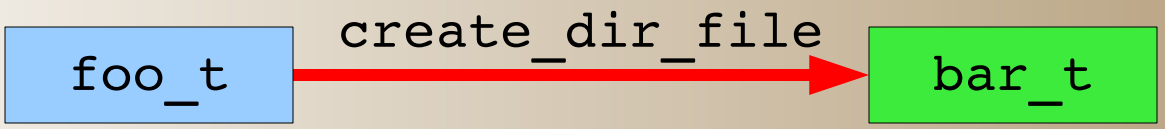

```
allow foo_t { root_t usr_t lib_t etc_t }:dir r_dir_perms;
allow foo_t lib_t:lnk_file r_file_perms;
allow foo_t ld_so_t:file rx_file_perms;
...
```

Features

- Class and permission sets
- Abstract resources
- **Abstract permissions**
- Templates
- Abstract type transitions

Abstract Permissions

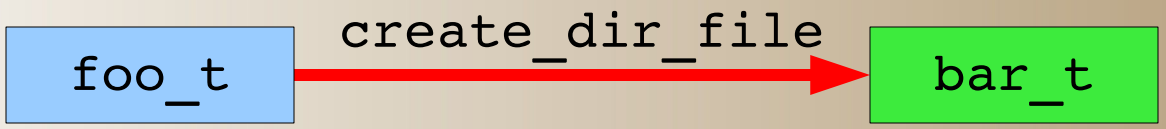
```
create_dir_file(foo_t, bar_t)
```




```
define(`create_dir_file', `
allow $1 $2:dir create_dir_perms;
allow $1 $2:file create_file_perms;
allow $1 $2:lnk_file create_lnk_perms;
')
```

Abstract Permissions

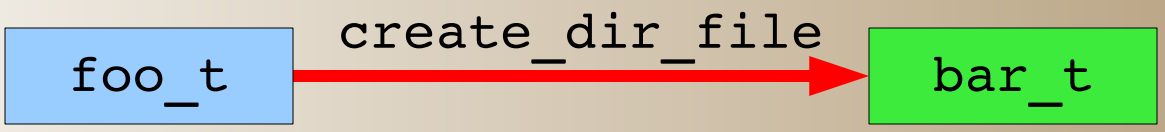
```
allow foo_t bar_t create_dir_file;
```

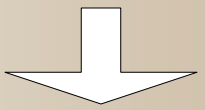



```
permission create_dir_file ($dom, $typ)
{
     allow $dom $typ:dir create_dir_perms;
     allow $dom $typ:file create_file_perms;
     allow $dom $typ:lnk_file create_lnk_perms;
};
```

Abstract Permissions

```
allow foo_t bar_t create_dir_file;
```



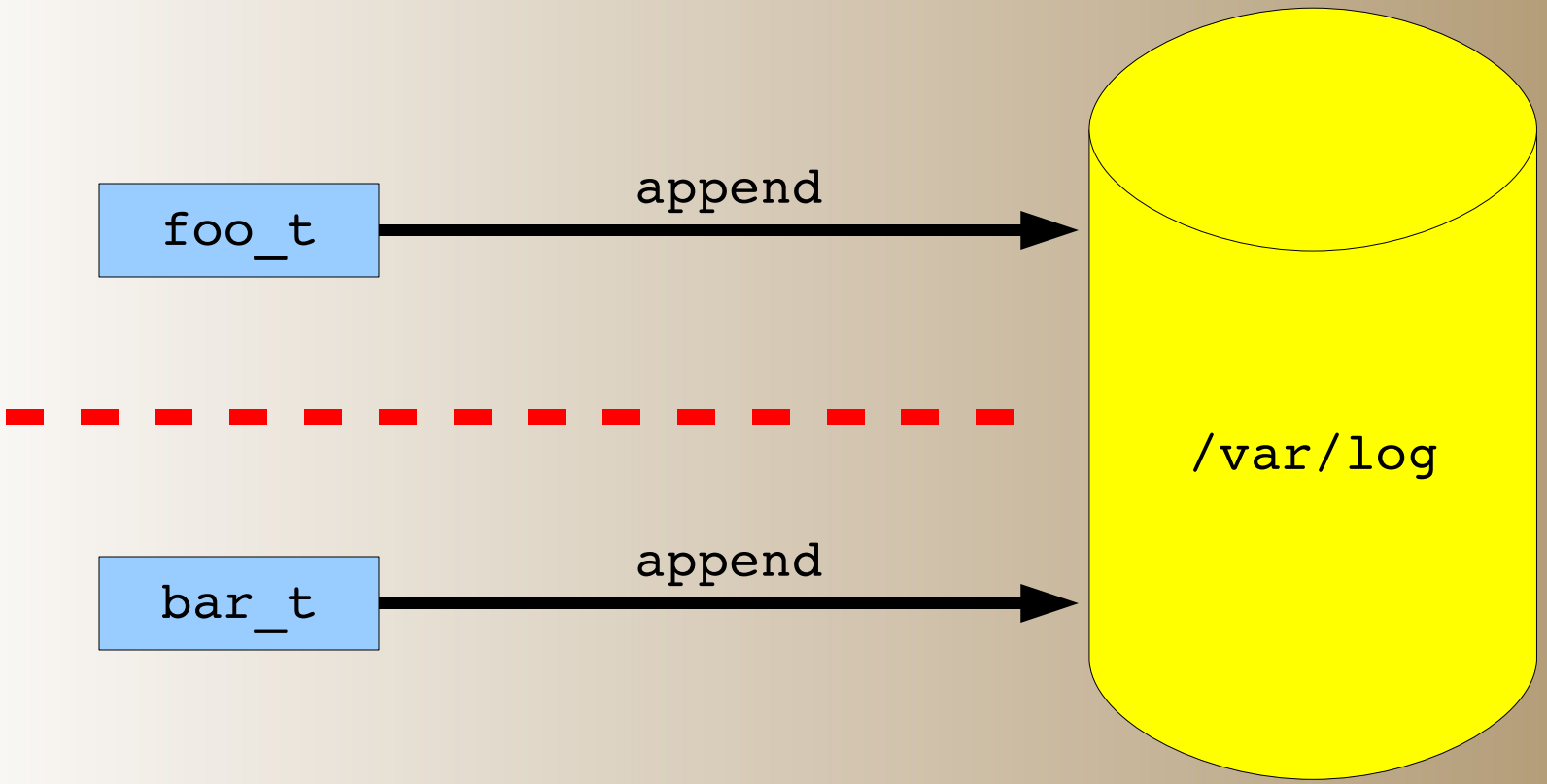


```
allow foo_t bar_t:dir create_dir_perms;
allow foo_t bar_t:file create_file_perms;
allow foo_t bar_t:lnk_file create_lnk_perms;
```

Features

- Class and permission sets
- Abstract resources
- Abstract permissions
- **Templates**
- Abstract type transitions

Motivating Templates



Template Declaration

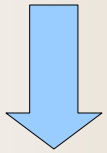
```
type ANYROLE.suffix_t;
type ANYTYPE.suffix_t;
```

Replaced with the name of an existing **role** or **type** at instantiation.

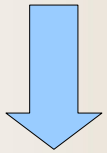
The "." character divides a name into a series of tokens.

Template Instantiation

```
role foo_r { foo_t };
```



```
type ANYROLE.suffix_t;
```

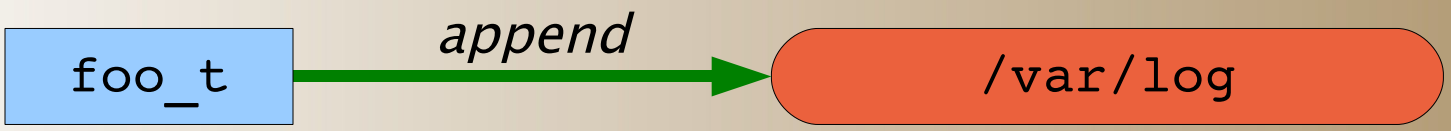


Compiler instantiates template automatically.

```
allow foo_r.suffix_t bar_t:file read;
```

Using Templates

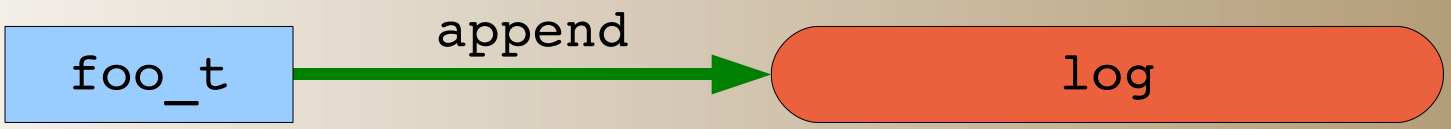
```
append_log_domain(foo)
```




```
define(`append_log_domain', `
type $1_log_t, file_type, sysadmfile, logfile;
allow $1_t var_log_t:dir ra_dir_perms;
allow $1_t $1_log_t:file { create ra_file_perms };
type_transition $1_t var_log_t:file $1_log_t;
')
```

Using Templates

```
allow foo_t log append;
```

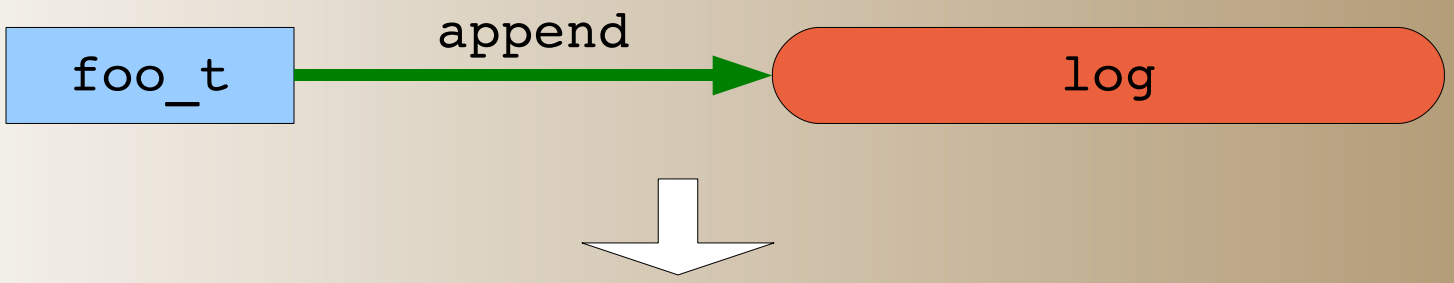



```
resource log { append ... }
type ANYTYPE.log_t { file_type sysadmfile logfile };

permission log append ($dom) {
  allow $dom var_log_t:dir ra_dir_perms;
  allow $dom $dom.log_t:file { create ra_file_perms };
  type_transition $dom var_log_t $dom.log_t:file;
};
```

Using Templates

```
allow foo_t log append;
```




```
type ANYTYPE.log_t { file_type sysadmfile logfile };

allow foo_t var_log_t:dir ra_dir_perms;
allow foo_t foo_t.log_t:file { create ra_file_perms };
type_transition foo_t var_log_t foo_t.log_t:file;
```

Prefix Resolution

```
type foo_t;
type ANYTYPE.suffix_t;
```

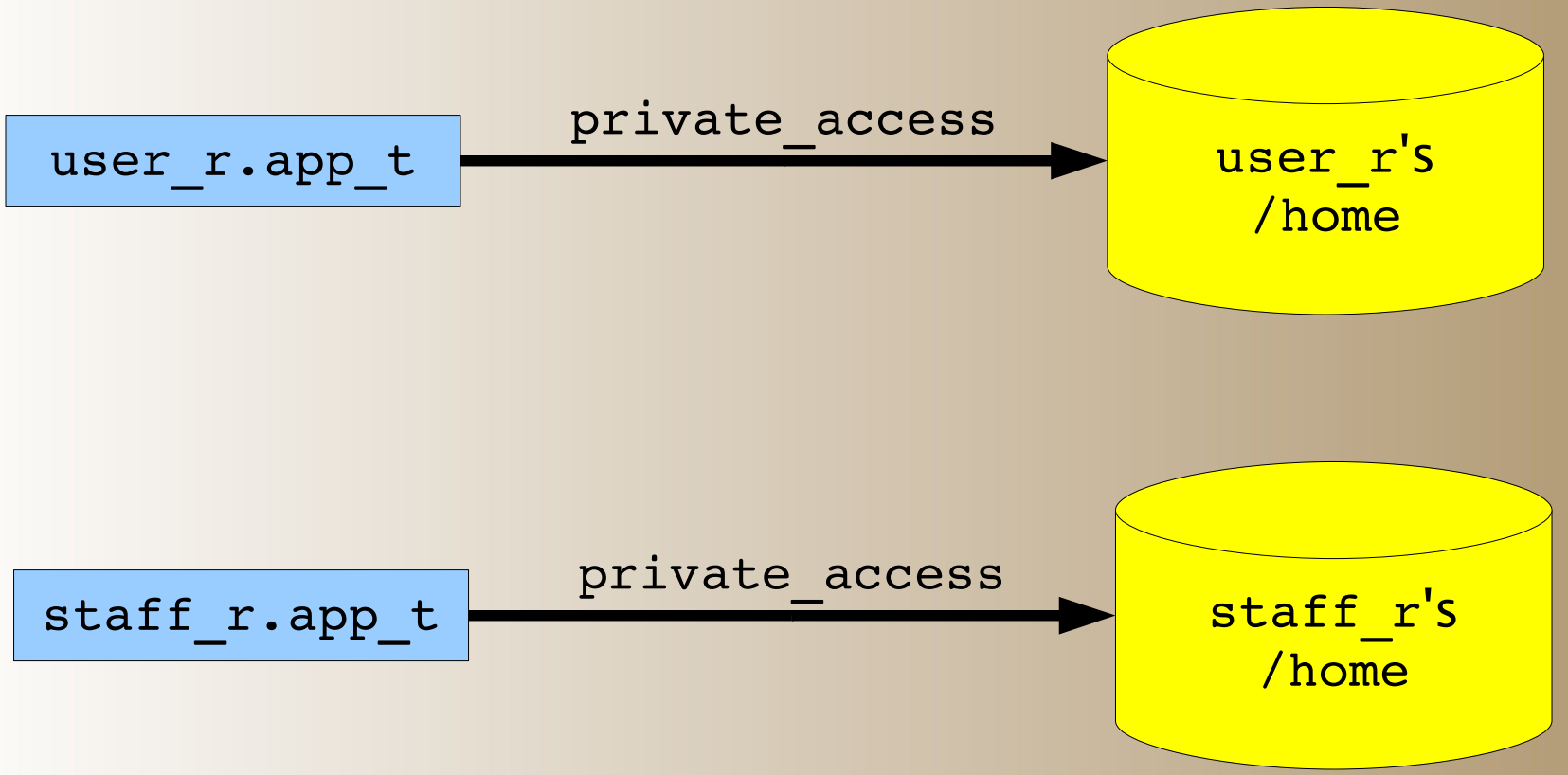
```
foo_t.suffix_t
```

Extracts the name of the type or role used as the **prefix** of the template instantiation.

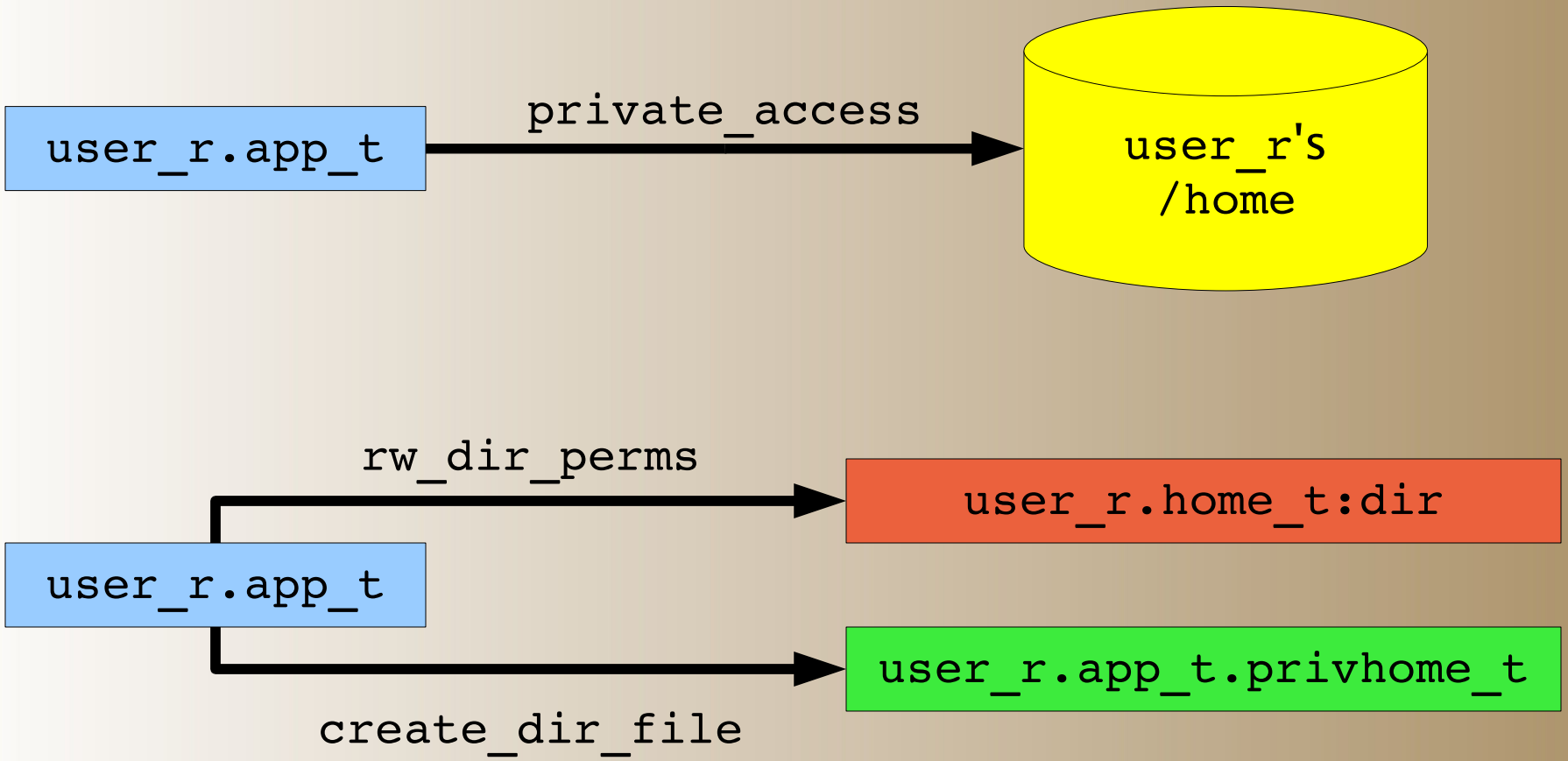
```
prefix(foo_t.suffix_t)
```

```
foo_t
```

Using Prefix Resolution



Using Prefix Resolution



Using Prefix Resolution

```
home_private_access(user, app)
```


```
define(`home_private_access', `
type $1_$2_privhome_t;
allow $1_$2_t $1_home_t:dir rw_dir_perms;
create_dir_file($1_$2_t, $1_$2_privhome_t)
')
```

Using Prefix Resolution

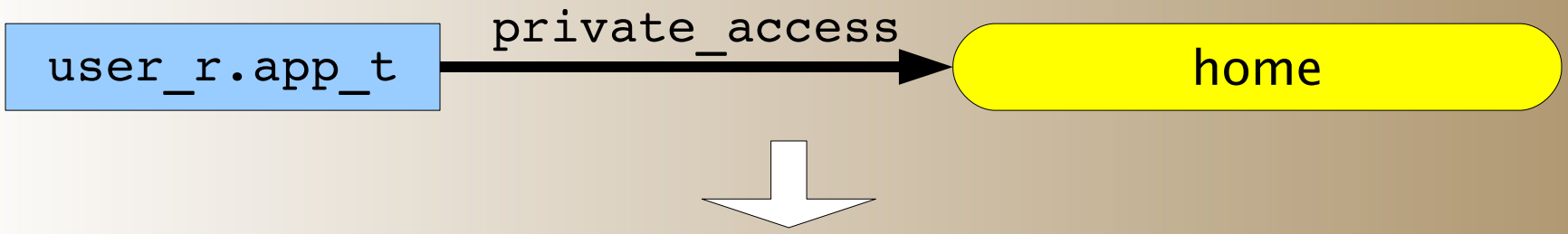
```
allow user_r.app_t home private_access;
```


```
type ANYROLE.app_t;
type ANYROLE.home_t;
type ANYTYPE.privhome_t;

permission home private_access ($dom) {
    allow $dom prefix($dom).home_t:dir rw_dir_perms;
    allow $dom $dom.privhome_t create_dir_perms;
};
```

Using Prefix Resolution

```
allow user_r.app_t home private_access;
```

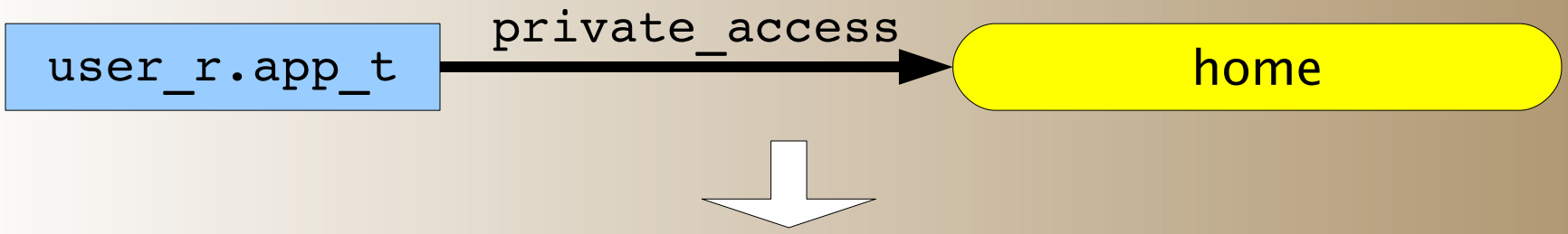



```
type ANYROLE.app_t;
type ANYROLE.home_t;
type ANYTYPE.privhome_t;

allow user_r.app_t prefix(user_r.app_t).home_t:dir
                                        rw_dir_perms;
allow user_r.app_t user_r.app_t.privhome_t
                                    create_dir_perms;
```

Using Prefix Resolution

```
allow user_r.app_t home private_access;
```



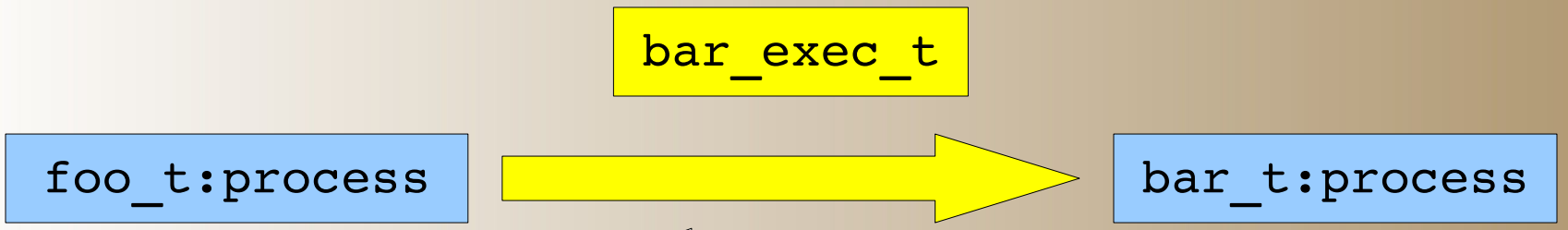

```
type ANYROLE.app_t;
type ANYROLE.home_t;
type ANYTYPE.privhome_t;

allow user_r.app_t user_r.home_t:dir rw_dir_perms;
allow user_r.app_t user_r.app_t.privhome_t
                                    create_dir_perms;
```

Features

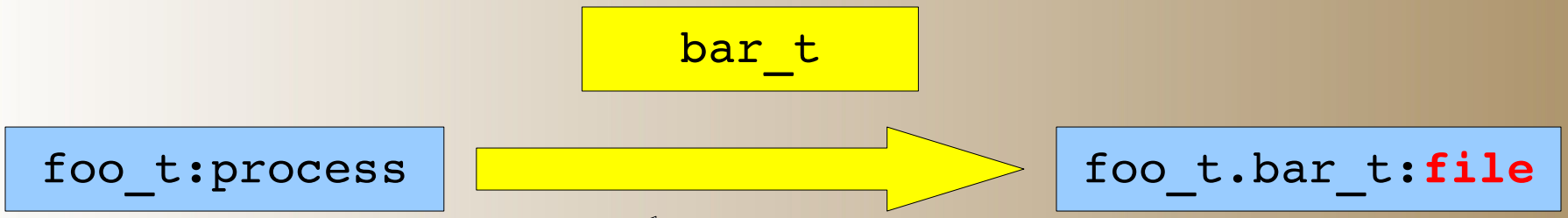
- Class and permission sets
- Abstract resources
- Abstract permissions
- Templates
- **Abstract type transitions**

Permissions and Transitions



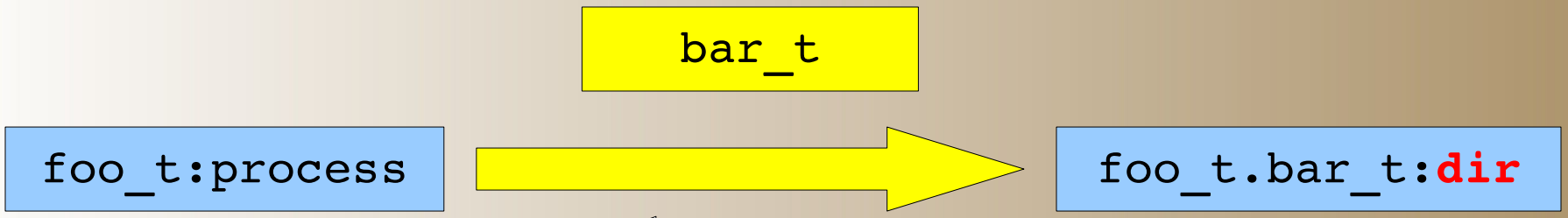

```
allow foo_t bar_t:process transition
allow foo_t bar_exec_t:file { read x_file_perms};
allow bar_t bar_exec_t:file rx_file_perms;
allow bar_t foo_t:process sigchld;
...
```

Permissions and Transitions




```
allow foo_t bar_t:dir rw_dir_perms;
allow foo_t foo_t.bar_t:file create_file_perms;
```

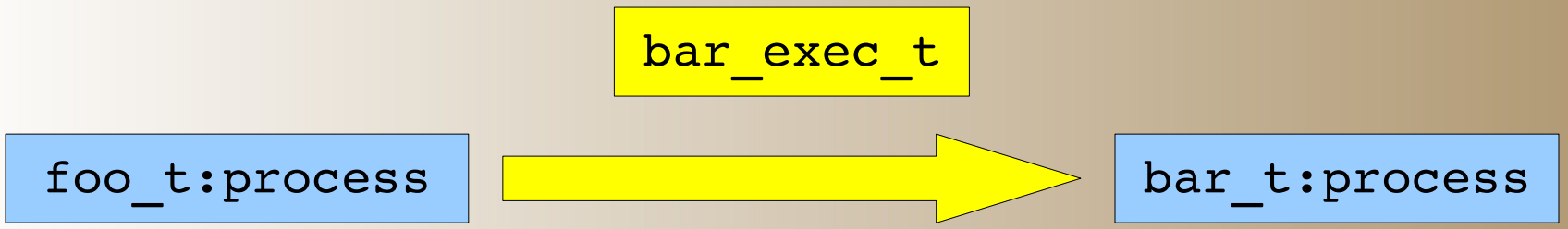
Permissions and Transitions




```
allow foo_t bar_t:dir rw_dir_perms;
allow foo_t foo_t.bar_t:dir create_dir_perms;
```

Abstract Transitions

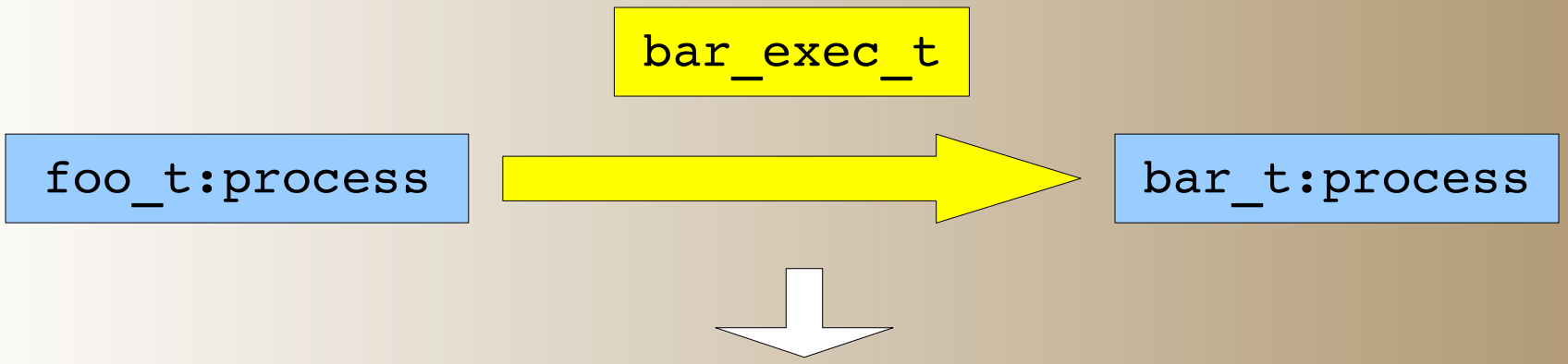
```
type_transition foo_t bar_exec_t bar_t:process domain;
```




```
trans domain ($from_dom, $via_typ, $to_dom:process)
{
  allow $from_dom $to_dom:process transition;
  allow $from_dom $via_typ:file { read x_file_perms };
  allow $to_dom $from_dom:process sigchld;
  allow $to_dom $via_typ:file rx_file_perms;
  ...
};
```

Abstract Transitions

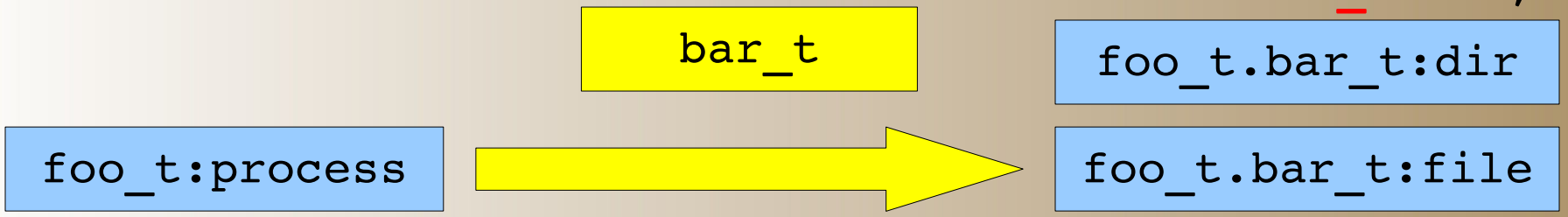
```
type_transition foo_t bar_exec_t bar_t:process domain;
```




```
allow foo_t bar_t:process transition;
allow foo_t bar_exec_t:file { read x_file_perms };
allow bar_t foo_t:process sigchld;
allow bar_t bar_exec_t:file rx_file_perms;
...
```

Abstract Transitions

```
type_transition foo_t bar_t foo_t.bar_t:{ dir file }
                                          file_trans;
```



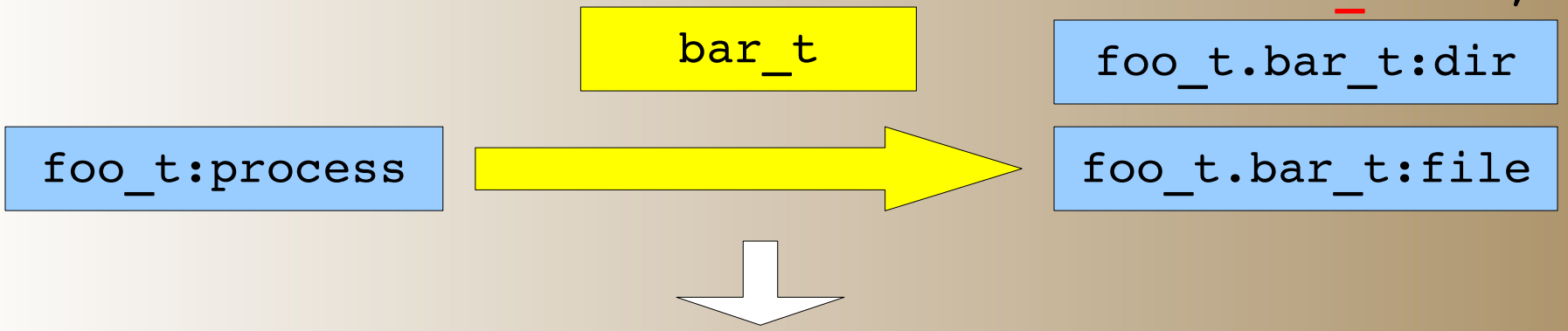

```
trans file_trans ($from_dom, $via_typ, $to_dom:$to_class)
    { allow $from_dom $via_typ:dir rw_dir_perms; };

trans file_trans ($from_dom, $via_typ, $to_dom:dir)
    { allow $from_dom $to_typ:dir create_dir_perms; };

trans file_trans ($from_dom, $via_typ, $to_dom:file)
    { allow $from_dom $to_typ:file create_file_perms; };
```

Abstract Transitions

```
type_transition foo_t bar_t foo_t.bar_t:{ dir file }
                                              file_trans;
```




```
allow foo_t bar_t:dir rw_dir_perms;
allow foo_t foo_t.bar_t:dir create_dir_perms;

allow foo_t bar_t:dir rw_dir_perms;
allow foo_t foo_t.bar_t:file create_file_perms;
```

Overview

- What's wrong with macros?
- Can we do better?
- **The future...**

SENG Status

- Proof-of-concept SENG compiler
  - Emits equivalent policy in existing monolithic policy language
- Small-scale smoke testing
  - Reimplementing small subset of monolithic policy using SENG

What's Left: Language

- Support for reference policy
  - Use SENG abstractions for module interfaces instead of m4?
- Support for MLS/MCS
  - Unknown what changes to SENG would be needed
- Formally defined semantics for SENG

What's Left: Toolset

- Policy analysis using SENG
  - SENG *should* be relatively easy to analyze
- Full-featured SENG compiler
  - Current proof-of-concept not ready for production use

More Information

- Website
  - http://web.ics.purdue.edu/~kuliniew/seng/
  - (coming soon...)
- Or Ask
  - kuliniew@purdue.edu
  - The person standing at the lectern

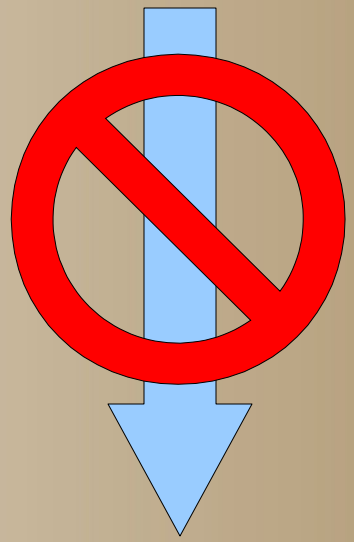
Bonus Slides

Recursive Instantiation

```
type foo_t;
```

```
type foo_t.suffix_t;
```

```
type ANYTYPE.suffix_t;
```

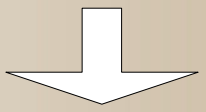


Could lead to **unbounded** number of types!

```
type foo_t.suffix_t.suffix_t;
```

Recursive Instantiation

```
type ANYTYPE.suffix_t;
resource recursive { use };
permission recursive use ($dom) {
    allow $dom $dom.suffix_t:file read;
    allow $dom.suffix_t recursive use;
};
allow foo_t recursive use;
```



```
allow foo_t foo_t.suffix:file read;
allow foo_t.suffix_t foo_t.suffix_t.suffix_t:file read;
allow foo_t.suffix_t.suffix_t
      foo_t.suffix_t.suffix_t.suffix_t:file read;
...
```

Class and Permission Sets

- Mere aggregation of multiple classes or permissions
- Similar to **attributes** for types in current language
  - In SENG, attributes are called **type sets**
- Nothing really new here

Abstract Resources

- Grant domain access to a resource
  - Resource composed of multiple things
- `allow` syntax makes intent clear
  - Rules associated with resource's permission do the necessary work

Abstract Permissions

- Grant a domain various permissions over a type
  - Permissions spread across multiple classes
- Again, use of `allow` makes intent clear

Motivating Templates

- Intuitive that `/var/log` should be an **abstract resource**
- `foo_t` mustn't access `bar_t`'s files
  - And vice versa
- Need to generate new types for each domain's log files
  - We need **type templates**

Permissions and Transitions

- Transitions need supporting permissions to be granted
  - Involve source type, target type, and related type
  - The permissions may depend on what class of object is being transitioned to
- Associate these permissions **with the transition itself**!