2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

The Untapped Power of UVM Resources and Why Engineers Should Use the uvm_resource_db API

Clifford E. Cummings — Paradigm Works, Inc.

Heath Chambers — HMC Design Verification

Mark Glasser — Elastics.cloud

Agenda

In The Paper

With more examples & detailed explanations

- Introduction
- `get_full_name()` -vs- `this`
- Resources stored w/ `uvm_resource_db`
- Name Table, Type Table & UVM Resources
- Resources retrieved w/ `uvm_resource_db`
- Resources stored w/ `uvm_config_db`
- Resources retrieved w/ `uvm_config_db`
- Examples converting `uvm_config_db` to `uvm_resource_db`
- Avoiding `p_sequencer` & macro
- GLOBS & POSIX regular expressions
- Debugging resources
- OVM `set_config_*` / `get_config_*`
- Resource efficiency
- Summary of Capabilities
- Conclusions

In This Presentation

Abbreviated

- Introduction
- Resources stored w/ `uvm_resource_db`
- Name Table, Type Table & UVM Resources
- Resources retrieved w/ `uvm_resource_db`
- Resources stored w/ `uvm_config_db`
- Resources retrieved w/ `uvm_config_db`
- Examples converting `uvm_config_db` to `uvm_resource_db`
- OVM `set_config_*` / `get_config_*`
- Summary of Capabilities
- Conclusions

Please read the paper for more information and details

Old OVM set_config_* Commands

set_config_int / set_config_string / set_config_object

```
set_config_int    ("*",        "cnt", 2);
```

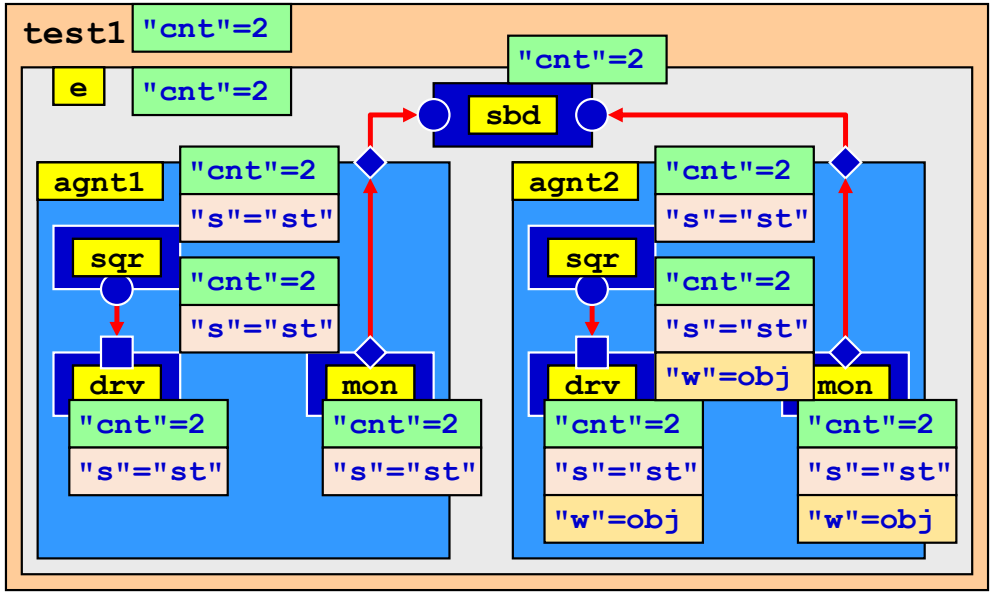
11 "cnt" int values stored

```
set_config_string("*agnt*",  "s",  "st");
```

8 "s" string values stored

```
set_config_object("*agnt2.*","w",  obj,0);
```

3 "w" obj handles stored

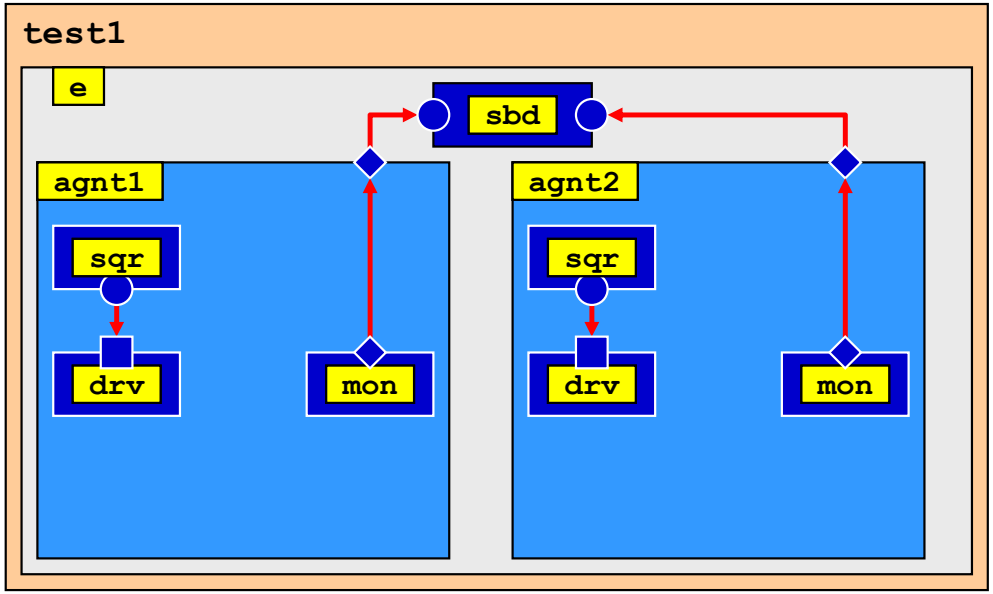


Very inefficient to store common configuration variables inside of multiple components

… and can only store int, string and ovm_object handle types

Central Resource Pool Added to UVM

To Store & Retrieve Usable Resources



uvm_resource_pool

	Type	Scope / Regex	Value
R1	*vir dut_if*	"*agnt*"	dif
R2	*env_cfg*	"*.e*"	cfg
R3	*agnt_cfg*	"*agnt1"	cfg1
R4	*agnt_cfg*	"*agnt2"	cfg2
R5	*int*	"*"	4
R6	*int*	"*.e*"	1
R7	*string*	"*agnt1"	"Warn1"
R8	*string*	"*agnt2"	"Err2"
R9	*int*	"LCNT::*"	10

Primary **access uses the `uvm_resource_db` API**

Secondary **access uses the `uvm_config_db` API**

Any type can be stored

Not directly stored in components

Resource Database

Recommended for UVM Use

- **uvm_resource_db** & **uvm_config_db** commands:
  - APIs to access general purpose resource database
  - Place to store & retrieve ***type***-specific information
  - Information can be written / read at anytime during simulation
  - All **uvm_resource_db** / **uvm_config_db** methods are **static**
  - **uvm_config_db** is layered on top of **uvm_resource_db**

Methods must be accessed using :: operator

uvm_resource_db methods:

```
get_by_type
get_by_name
set_default
set
set_anonymous
read_by_name
read_by_type
write_by_name
write_by_type
dump
```

Most commonly used methods

uvm_config_db methods:

```
get
set
exists
wait_modified
```

Most commonly used methods

See UVM Class Reference Manual for more details on all of the methods

uvm_resource_db / uvm_config_db APIs

The Bad News!

- More than 90% of UVM verification engineers are using the wrong API !!

Early UVM books & papers are to blame

- `uvm_resource_db` API is powerful

Can be used with *components*, *sequences* and *transactions*

… and is simple to use

- `uvm_config_db` API has limitations

Can only be used with *components*

… and requires a confusing `cntxt-inst_name` matching mechanism

Requires `p_sequencer` gymnastics to pass test information to sequences

UVM Resource APIs - ::set Overview

Accessor guidelines:
For classes: `this`
For modules: `null`

Not required to be a component scope

Type that is stored

Simple `"match-string"` (with optional wildcards)

Where it is stored

What value is stored

```
uvm_resource_db #(type)::set( "scope", "field_name", value, accessor);
```

Used for optional debug tracing

`resource_db` API

`config_db` API

This is just a `"match-string"` used for matching during `get` / `read` operations

```
uvm_config_db #(type)::set( cntxt, "inst_name", "field_name", value );
```

Must be a component *class handle*

String (with optional wildcards)

UVM must convert this to a `"match-string"`

… but `uvm_config_db` string must match a real component path

UVM Resource APIs - ::get/read Overview

Declare *var1* to hold retrieved *value*

This is mostly what we care about

Accessor guidelines:
For classes: `this`
For modules: `null`

```
type var1;
```

Type that is stored

Simple *"match-string"* (with optional wildcards)

Where it is stored

Copy to *var1*

```
if(!uvm_resource_db #(type)::read_by_name( "scope", "field_name", var1, accessor))...
```

Used for optional debug tracing

`resource_db` API

`config_db` API

This is just a "match-string" used for matching during `get` / `read` commands

```
if(!uvm_config_db #(type)::get( cntxt, "inst_name", "field_name", var1 )...
```

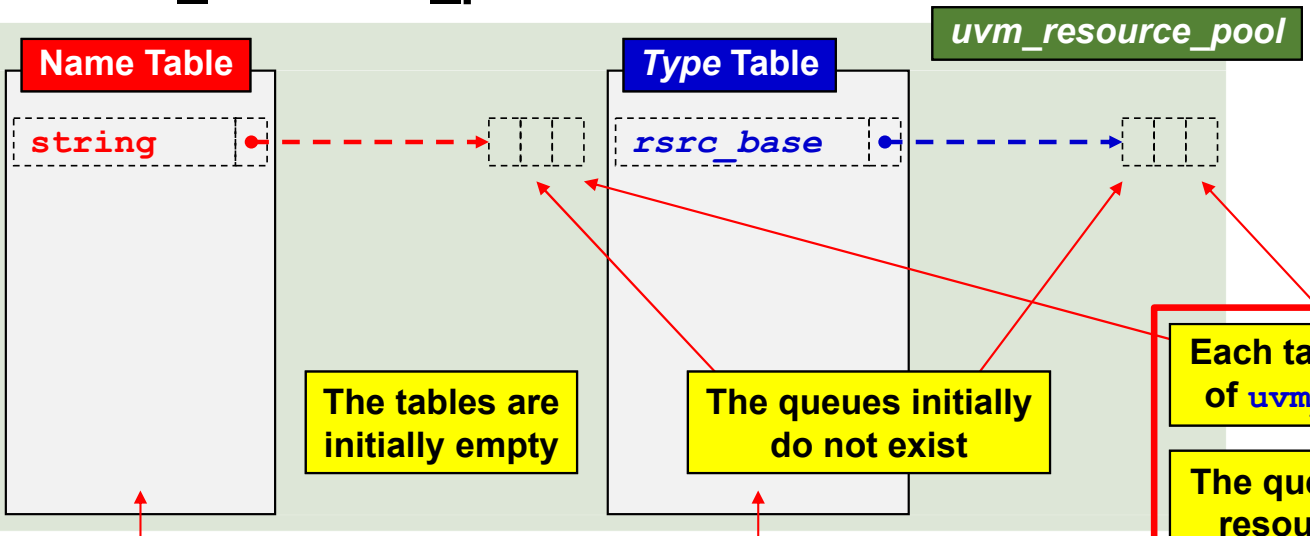
Must be a component *class handle*

String (with optional wildcards)

UVM must convert this to a *"match-string"*

... but `uvm_config_db` string must match a real component path

uvm_resource_pool - Tables



uvm_resource_db commands add a new entry into both tables …

… *or* pushes a new handle onto an existing queue

The tables are initially empty

The queues initially do not exist

Each table entry points to a uvm_queue of uvm_resource_base-type handles

The queues allow testbenches to have multiple resources of the same name and same *type*

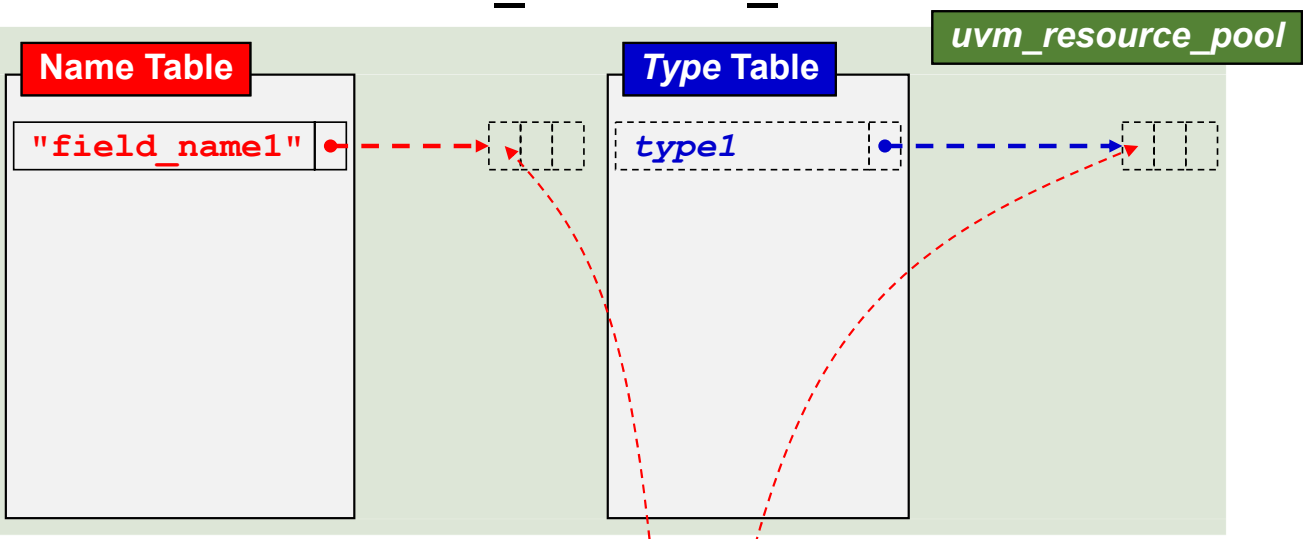
uvm_resource_pool declares two associative arrays: Name Table (ntab) & *Type Table* (*ttab*)

Name Table of handles that point to uvm_queues of uvm_resource_base-type handles (Indexed by *strings*)

```
class uvm_resource_pool;
  ...
  uvm_queue#(uvm_resource_base) ntab [string];
  uvm_queue#(uvm_resource_base) ttab [uvm_resource_base];
  ...
```

Type Table of handles that point to *uvm_queues* of *uvm_resource_base*-type handles (Indexed by *uvm_resource_base* class handles)

Resources - uvm_resource_db::set



Each `uvm_resource_db::set` command creates a *typed* resource

Each `uvm_resource_db::set` command creates two table entries

Store the *typed resource* handle into Name Table & *Type Table* queues

Each new resource handle is stored in two locations (queues)

Each queue is of type `uvm_resource_base` (*typed* `uvm_resources` are upcast to the queues)

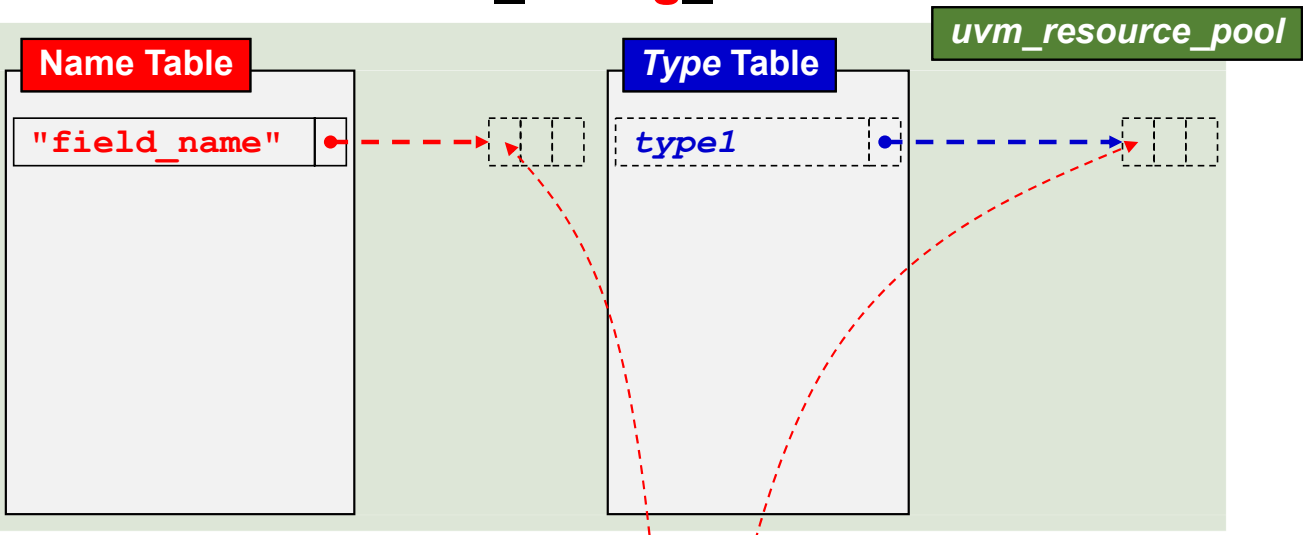
```
uvm_resource_db #(type1)::set( "*scope*", "field_name1", value1, accessor);
```

Create a *typed* `uvm_resource`

`uvm_resource#(type1)`	*type1*	`"*scope*"`	`value1`
	Type	Scope / Regex	Value

Each `uvm_resource` has 3 fields

Resources - uvm_config_db::set



Each `uvm_config_db::set` *also* creates a *typed* resource

Each `uvm_config_db::set` *also* creates two table entries

Store the *typed resource* handle into Name Table & *Type Table* queues

UVM combines `cntxt` and `"inst_name"` into a new `"scope"`-string

```
uvm_config_db #(type1)::set( cntxt, "inst_name", "field_name", value1 );
```

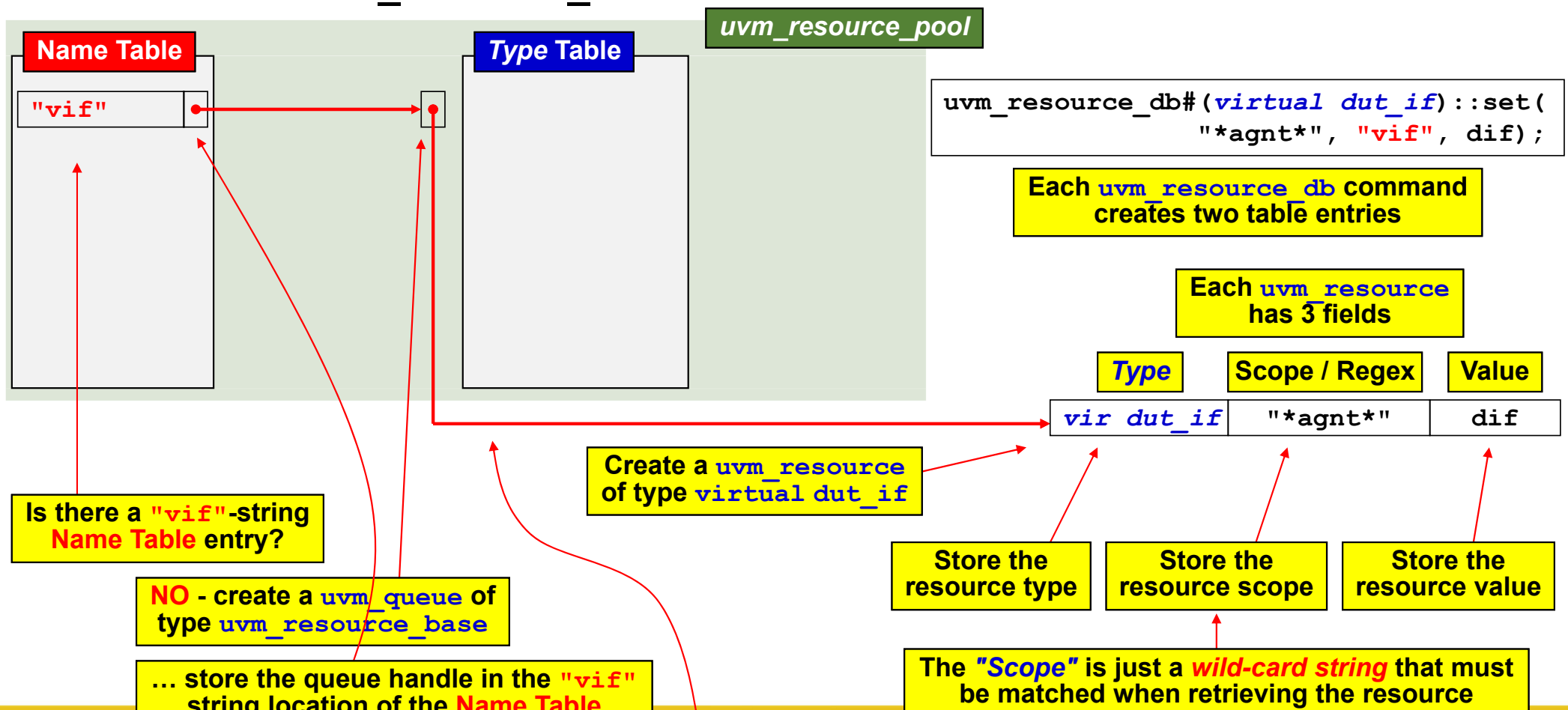
Create a *typed* `uvm_resource`

The new `"scope"`-string must be a valid *"component-path" string*

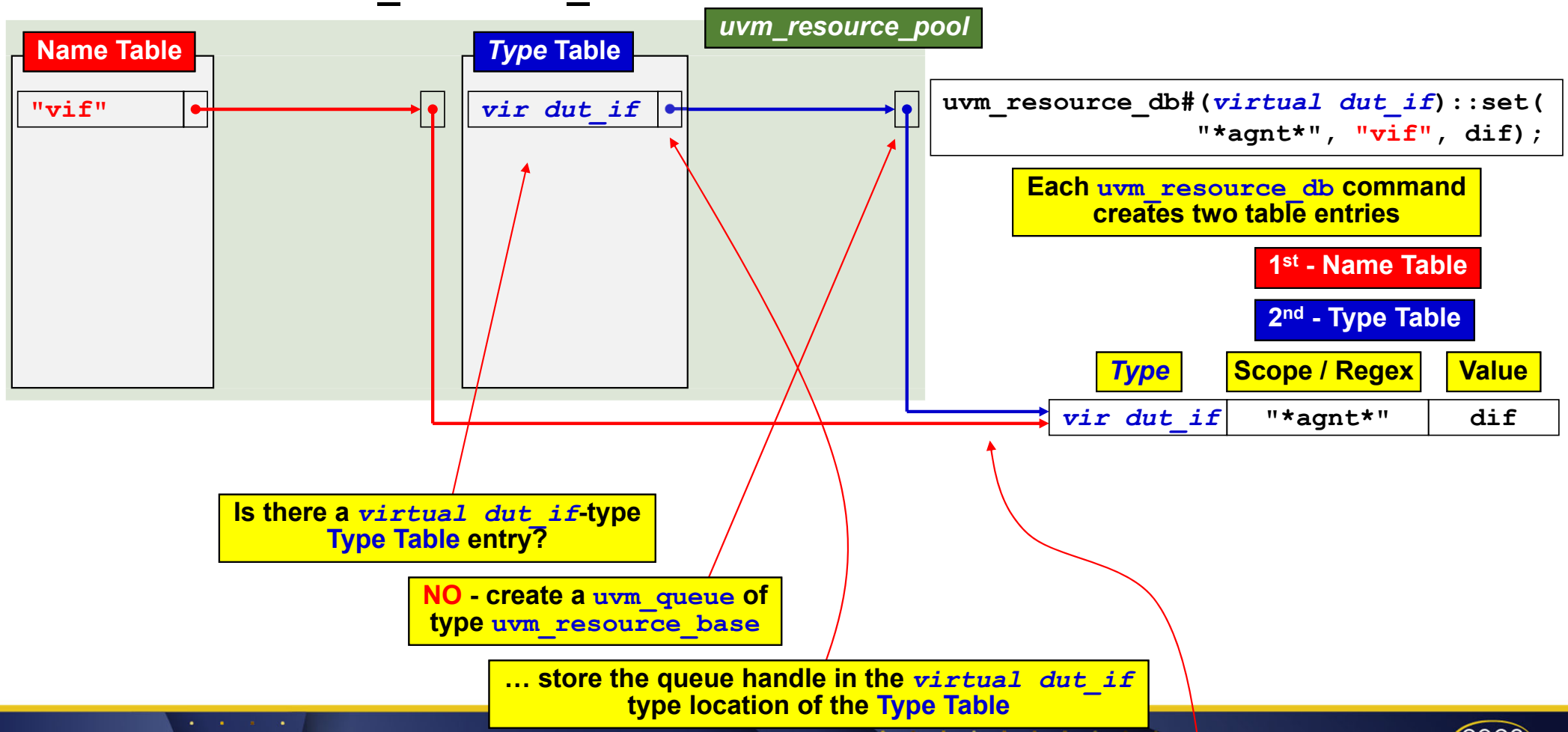
`uvm_resource#(type1)`	*type1*	`"scope"`	`value1`
	Type	Scope / Regex	Value

Each `uvm_resource` has 3 fields

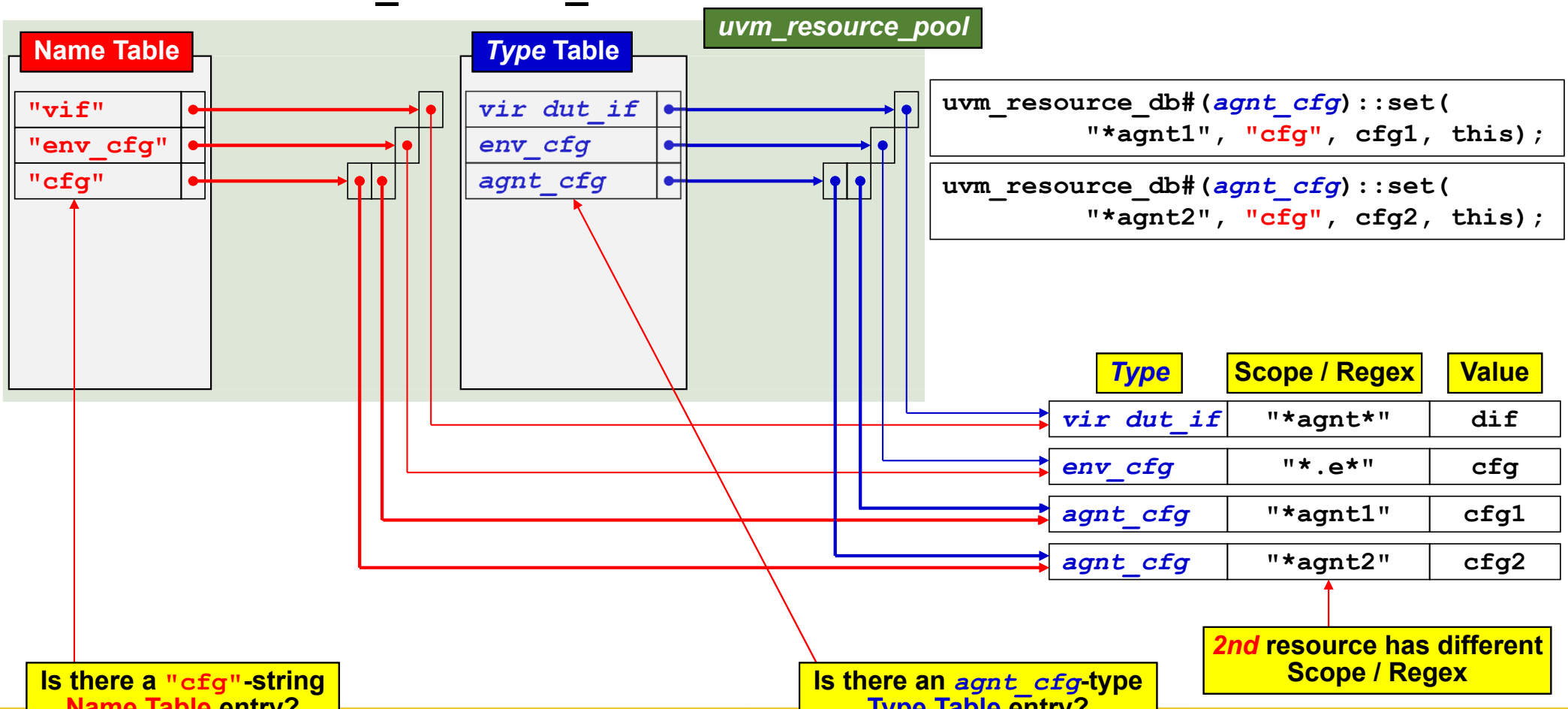
Resources - uvm_resource_db::set



Resources - uvm_resource_db::set



Resources - uvm_resource_db::set



2nd resource has different Scope / Regex

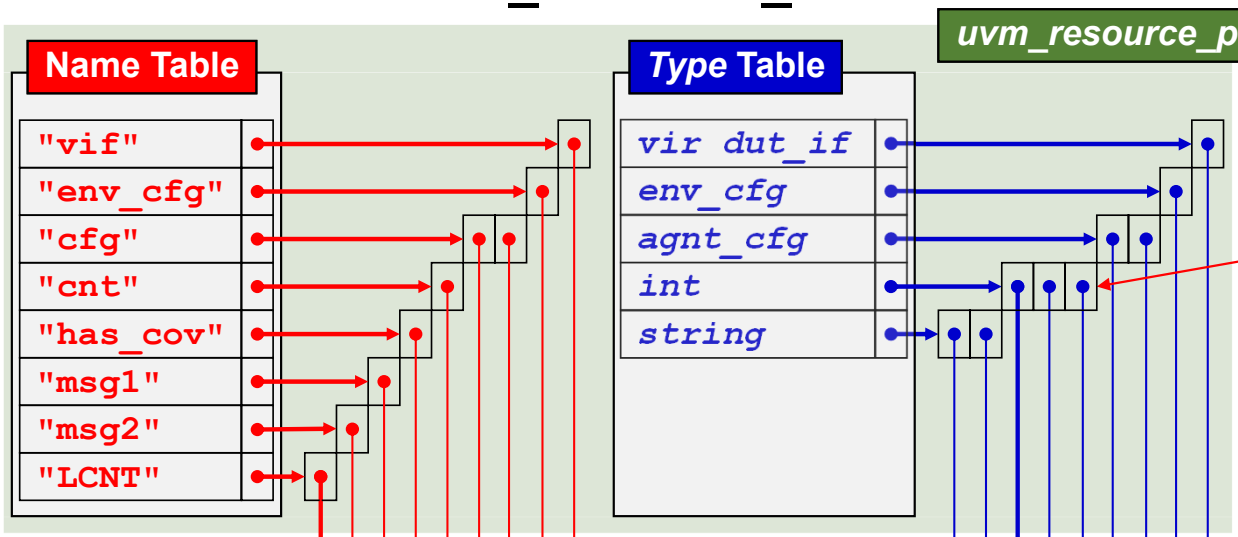
Is there a "cfg"-string Name Table entry?

Is there an agnt_cfg-type Type Table entry?

YES - push the *2nd* uvm_resource#(agnt_cfg) handle onto the queue (*upcast*)

YES - push the *2nd* uvm_resource#(agnt_cfg) handle onto the queue (*upcast*)

Resources - uvm_resource_db::set



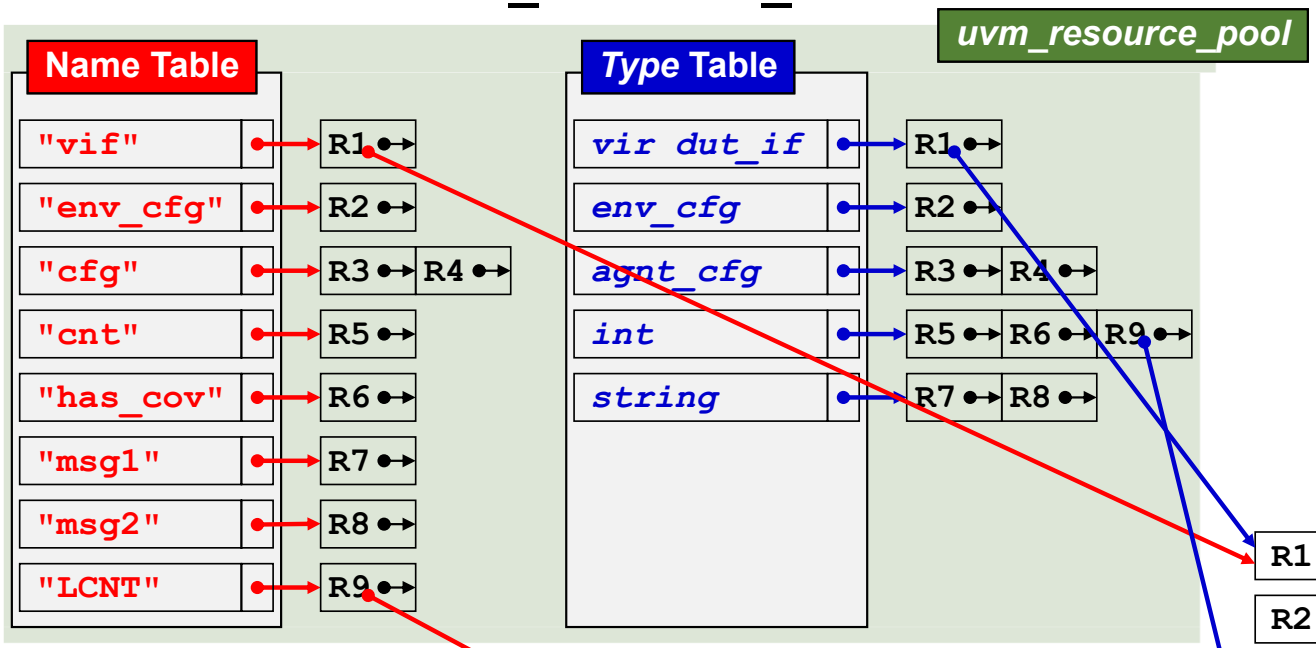

```
uvm_resource_db#(int)::set(
                    "LCNT::*", "LCNT", 10);
```

int queue is now 3-deep

uvm_resource_db does not require component path names for matching

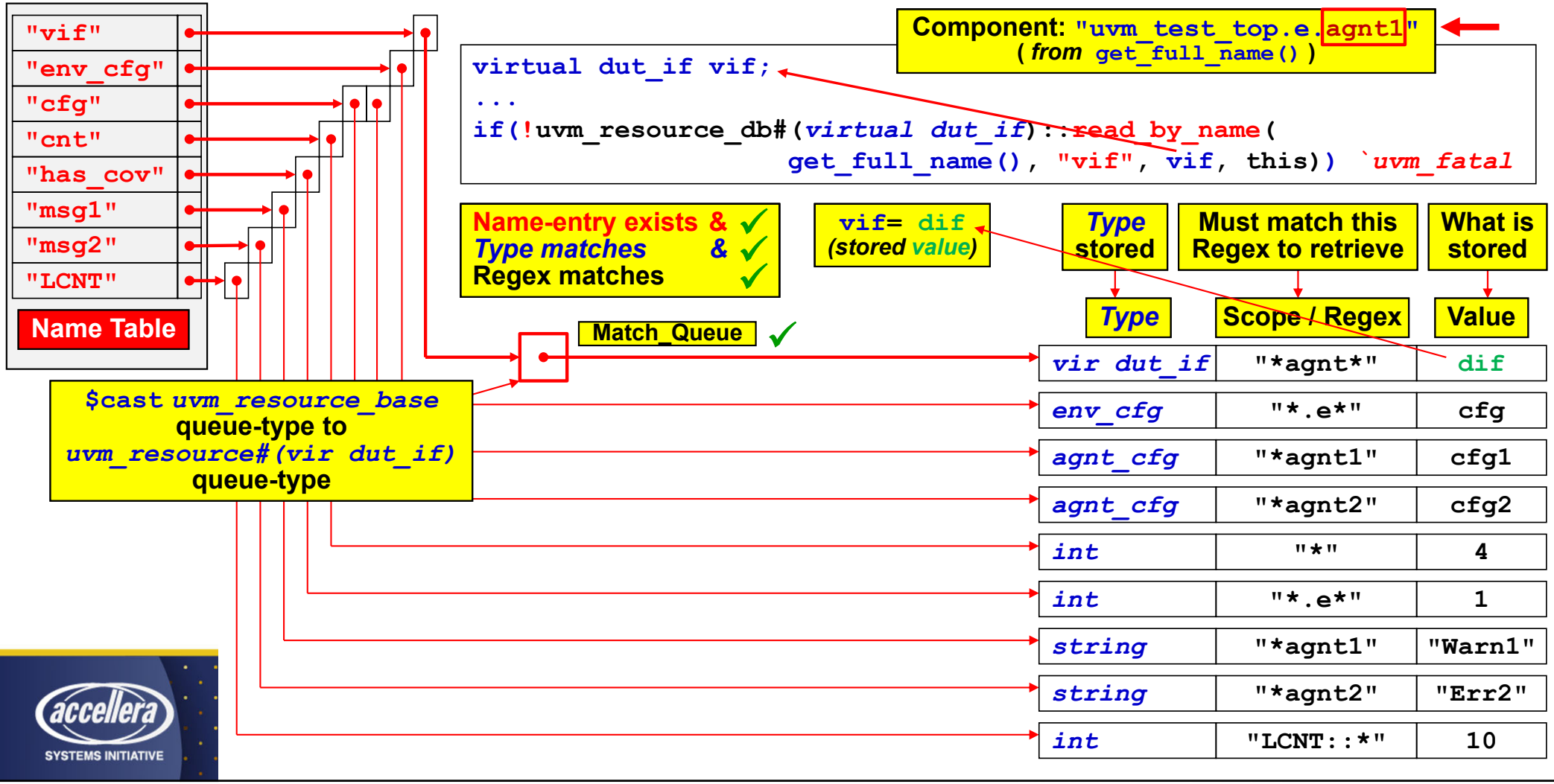
Type	Scope / Regex	Value
vir dut_if	"*agnt*"	dif
env_cfg	"*.e*"	cfg
agnt_cfg	"*agnt1"	cfg1
agnt_cfg	"*agnt2"	cfg2
int	"*"	4
int	"*.e*"	1
string	"*agnt1"	"Warn1"
string	"*agnt2"	"Err2"
int	"LCNT::*"	10

Resources - uvm_resource_db::set

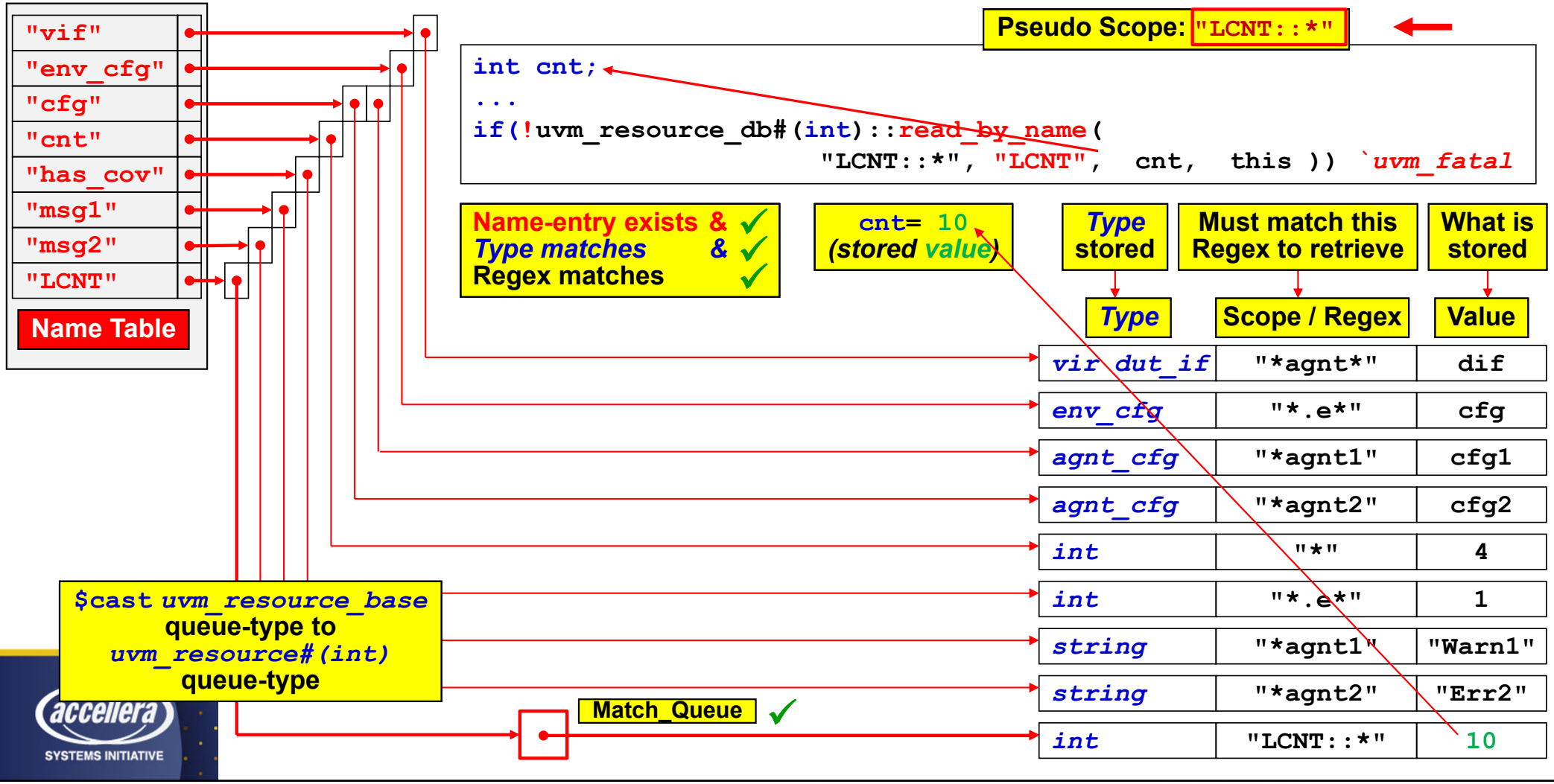


	Type	Scope / Regex	Value
R1	vir dut_if	"*agnt*"	dif
R2	env_cfg	"*.e*"	cfg
R3	agnt_cfg	"*agnt1"	cfg1
R4	agnt_cfg	"*agnt2"	cfg2
R5	int	"*"	4
R6	int	"*.e*"	1
R7	string	"*agnt1"	"Warn1"
R8	string	"*agnt2"	"Err2"
R9	int	"LCNT::*"	10

Resources - uvm_resource_db::read_by_name

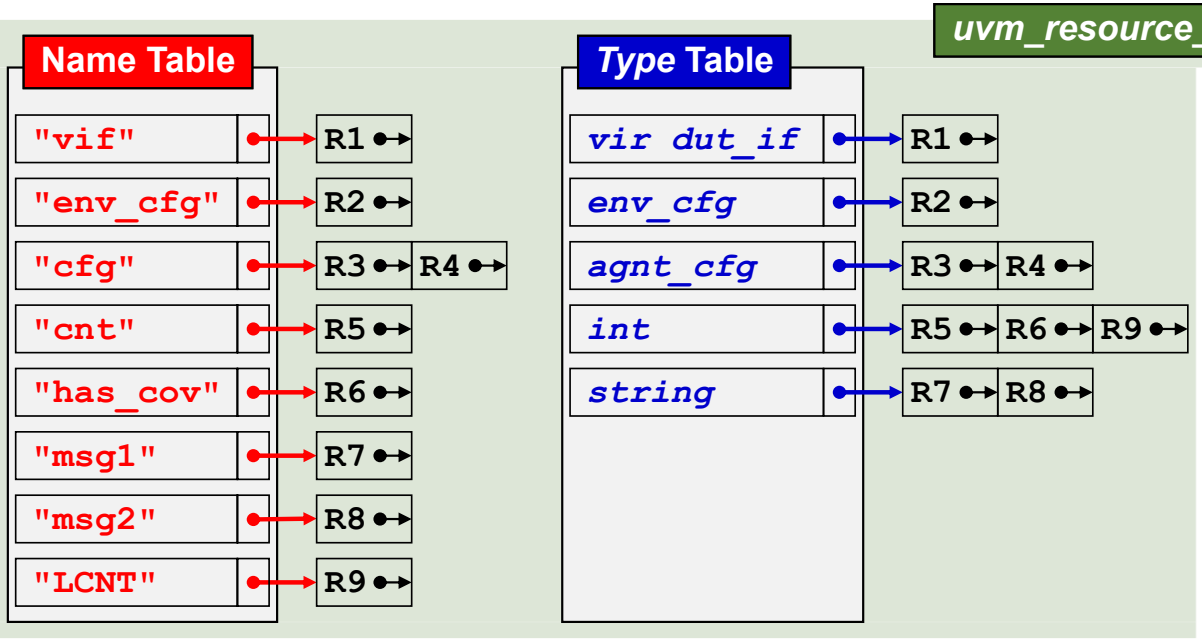


Resources - uvm_resource_db::read_by_name



uvm_*resource*_db::set Commands

Settable Resources



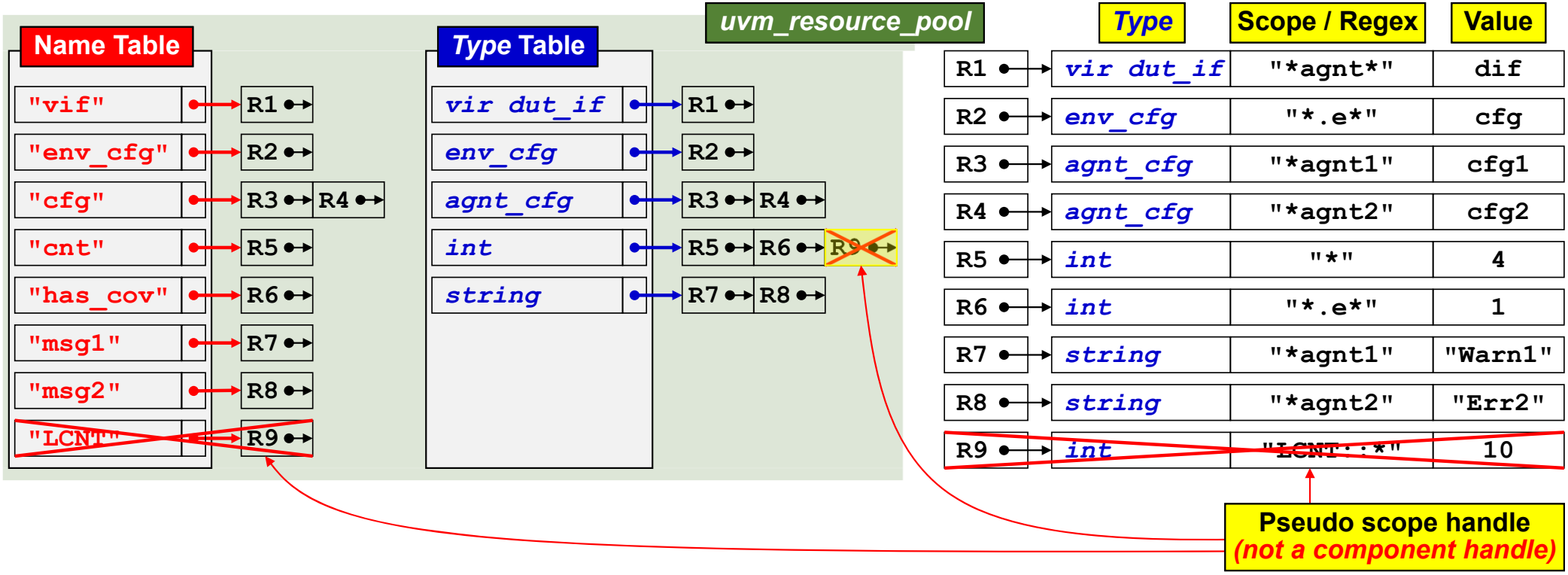
	Type	Scope / Regex	Value
R1	vir dut_if	"*agnt*"	dif
R2	env_cfg	"*.e*"	cfg
R3	agnt_cfg	"*agnt1"	cfg1
R4	agnt_cfg	"*agnt2"	cfg2
R5	int	"*"	4
R6	int	"*.e*"	1
R7	string	"*agnt1"	"Warn1"
R8	string	"*agnt2"	"Err2"
R9	int	"LCNT::*"	10

`uvm_resource_db` API stores resource handles in both Name Table and *Type Table* queues

`uvm_resource_db` API can create all these resources

uvm_*config*_db::set Commands

Settable Resources

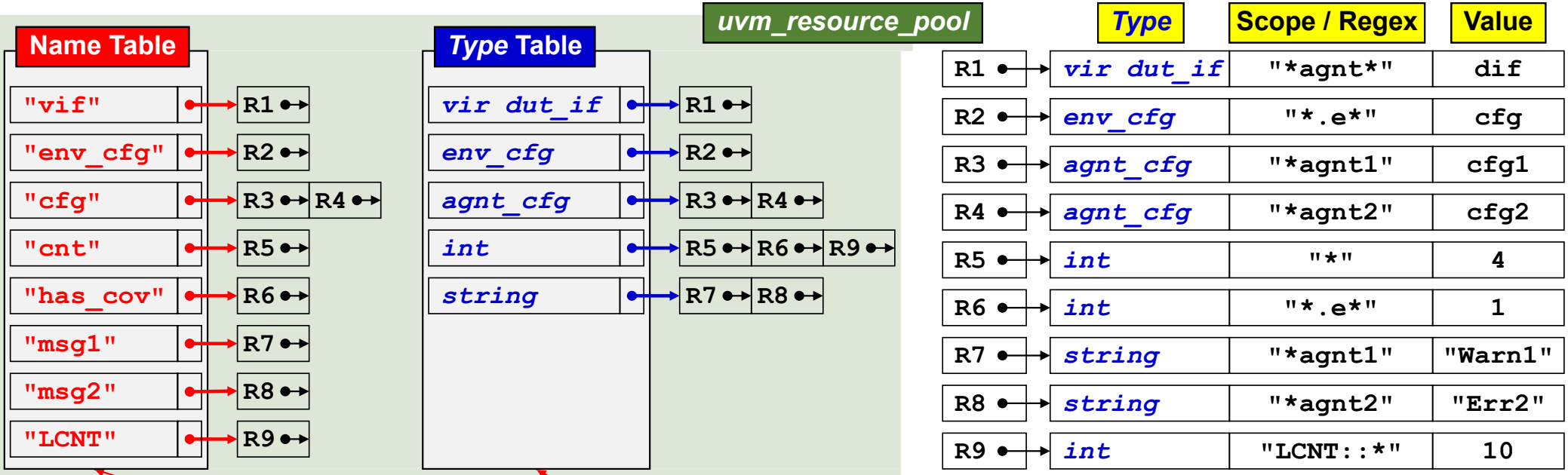


`uvm_config_db` API stores resource handles in both Name Table and *Type Table* queues

`uvm_config_db` API can only create resources for valid component handles

uvm_*resource*_db::read_by_name / *read_by_type*

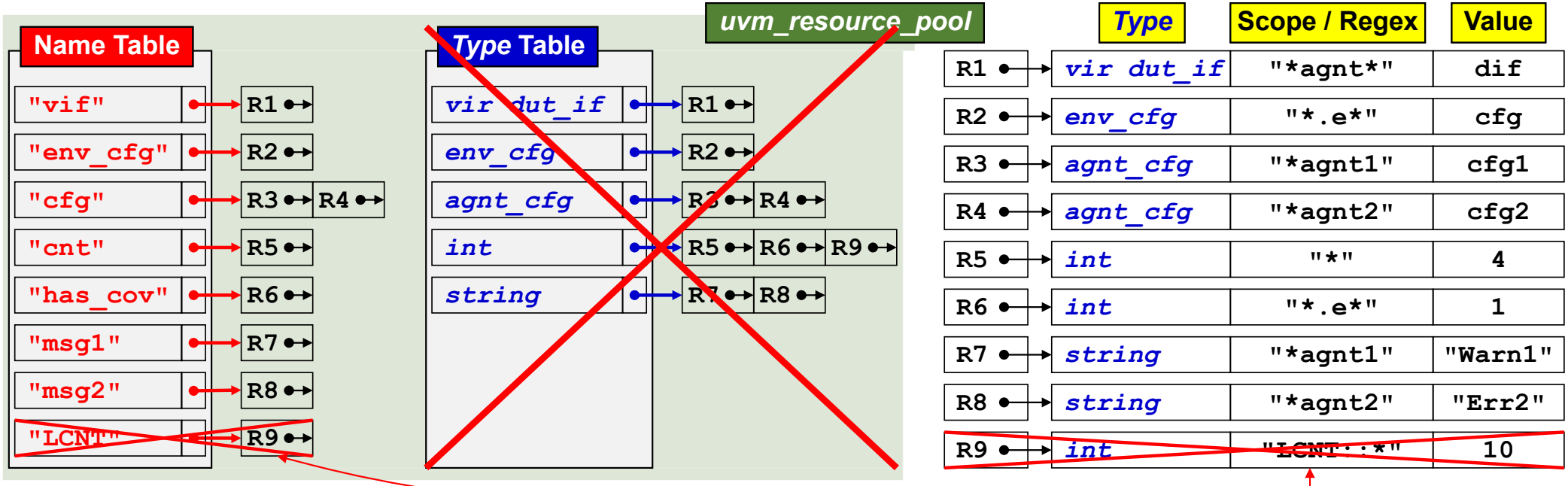
Readable Resources



`uvm_resource_db#()::read_by_name` & `read_type_type` APIs can retrieve resources using Name Table and *Type Table* accesses

uvm_*config*_db::get Commands

Readable Resources



`uvm_config_db#()::get` API can only retrieve legal resources using Name Table accesses

Pseudo scope handle *(not a component handle)*

Guideline: Use the simpler and more powerful `uvm_resource_db` API and stop using the `uvm_config_db` API

Why-Oh-Why uvm_config_db ??

How Did We Get This Messy uvm_config_db API?

- OVM had `set_config_*` commands
  - Placed storage into each referenced component
- UVM removed per-component storage
  - UVM goal: maintain a common storage database
  - UVM goal: do not support both old `set_config_*` storage and new UVM resources
  - UVM goal: older style `set_config_*` commands should work with new UVM resources
  - `uvm_config_db` was layered on UVM resources
  - `uvm_config_db` was shown in first UVM books & examples
  - More than 90% of UVM engineers have been using the WRONG API for 10+ years!

Only worked with components and only stored `int`, `string` and `ovm_object`

Wildcards propagated storage elements across multiple components

Added resources & database

More efficient storage

Forced backward compatible *component-only* storage required by `set_config_*`

Very unfortunate!

Including Cliff & Heath
(Until we worked with Mark Glasser)

set_config_* / uvm_config_db / uvm_resource_db

Summary of Capabilities

	set_config_*	uvm_config_db	uvm_resource_db
Used in OVM testbenches	✓	✗	✗
Used in UVM testbenches	✗	✓	✓
Stores `int` / `string` / object data types	✓	✓	✓
Stores any data type	✗	✓	✓
Allows use of glob regular expressions	✓	✓	✓
Allows use of POSIX regular expressions	✗	✗	✓
Distributes stored information across components	✓ BAD	✗ Good	✗ Good
Stores information in a common resource database	✗	✓ Good	✓ Good
Requires complex component handle & string scoping	✗	✓ BAD	✗ Good
Allows simple string scoping	✗	✗ BAD	✓ Good
Can store & retrieve information by name	✗	✓	✓
Can store & retrieve information by type	✗	✗	✓ Good
Can store & retrieve information into components	✓	✓	✓
Can store & retrieve information into sequences	✗	✗ BAD	✓ Great!
Can store & retrieve information into modules	✗	✗ BAD	✓ Great!

Conclusions

- *Quit* using `set_config_*` / `get_config_*` commands

  These commands deprecated in UVM *(and for a very good reason!)*

- *Quit* using the `uvm_config_db` API

  Needlessly complicated `cntxt`-*handle* `inst_name`-*string scoping* requirements

  `uvm_config_db` lacks important features that simplify UVM testbench development

  No need to remove existing `uvm_config_db` commands

- *USE* the `uvm_resource_db` API

  Much easier, simple-string *scoping* requirements

- `uvm_resource_db` simplifies advanced UVM testbench techniques

  Simplifies virtual sequences and more

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Questions?

accellera
SYSTEMS INITIATIVE