Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Fixed Size Arrays

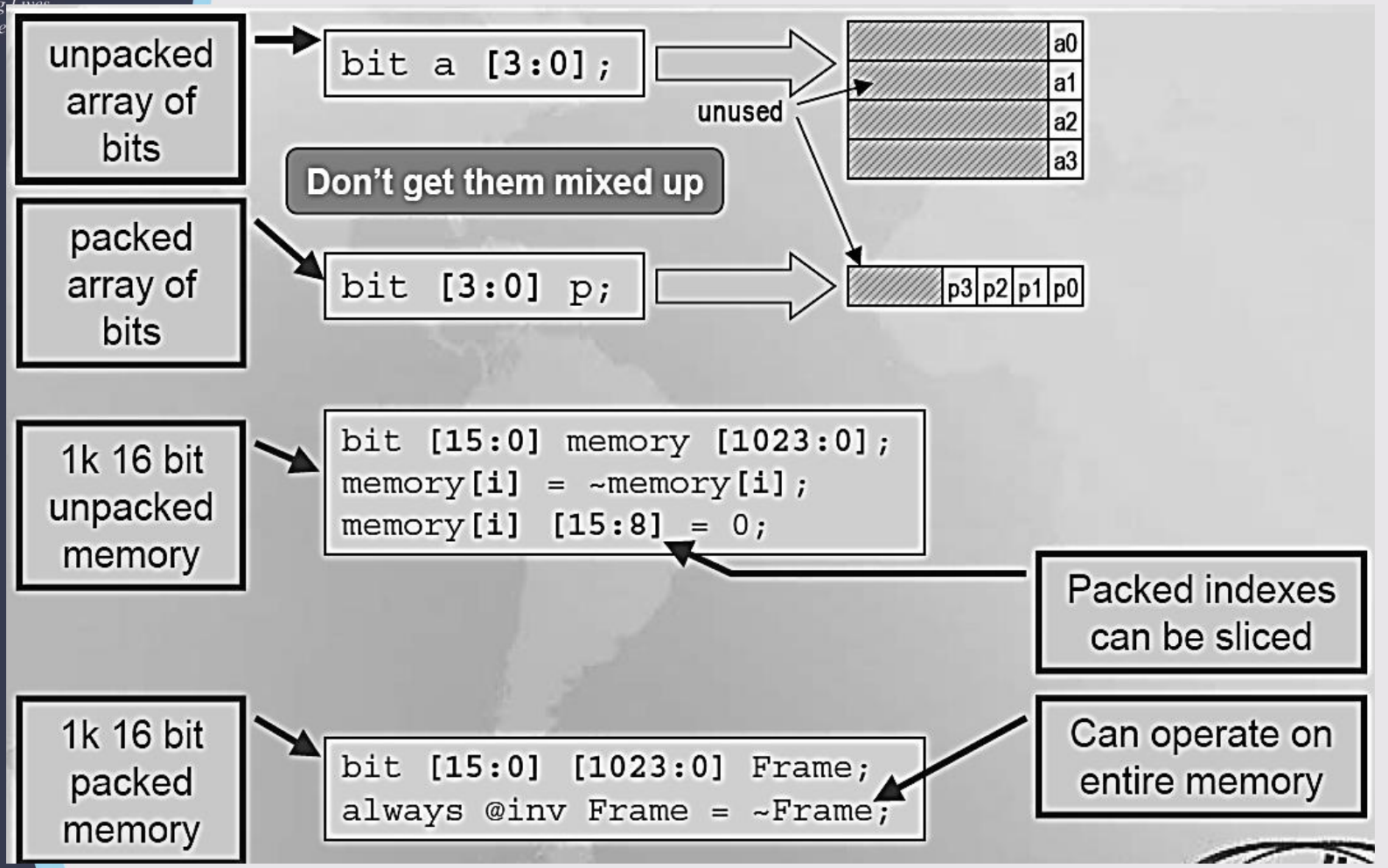
Declaring fixed-size arrays

```
int lo_hi[0:15]; // 16 ints [0]..[15]
int c_style[16]; // 16 ints [0]..[15]
```

Multi Dimensional

```
int array2 [0:7][0:3]; // Verbose declaration
int array3 [8][4]; // Compact declaration
array2[7][3] = 1; // Set last array element
```

Packed and Unpacked Arrays



Dynamic Arrays

Dynamic declaration of one index of an unpacked array

Syntax:

```
data_type array_name[] ;
```

Declares a dynamic array *array_name* of type *data_type*

```
data_type array_name[] = new[ array_size ] [(array)] ;
```

Allocates a new array *array_name* of type *data_type* and size *array_size*
Optionally assigns values of *array* to *array_name*
If no value is assigned then element has default value of *data_type*

Examples:

```
bit [3:0] nibble[ ];          // Dynamic array of 4-bit vectors
integer mem[ ];               // Dynamic array of integers

int data[ ];                  // Declare a dynamic array
data = new[256];              // Create a 256-element array
int addr = new[100];          // Create a 100-element array
addr = new[200](addr);        // Create a 200-element array
                              // preserving previous values in lower 100 addresses
```

Dynamic Array - Methods

Centurion University
Shaping Lives...
Empowering Communities...

- **Resizing**

<array> = new[<size>](<src_array>);

dyn= new[j * 2](fix);

function int size() Returns the current size of the array

```
int addr[ ] = new[256];
int j = addr.size();          // j =  256
```

function void delete() Empties array contents and zero-sizes it

```
int addr[ ] = new[256];
addr.delete();
```

Associative Arrays

- Associative arrays are used when the size of the array is not known or the data is sparse.

- ***Syntax:***

  data_type array_name [index_type];

  In other words

  value_type array_name [key_type];

Associative Array Methods

Function	Use
num()	Returns number of entries
delete(<index>)	Index for delete optional. When specified used to delete given index else whole array.
exists (<index>)	Returns 1 if element exists at index else 0
first (<index>), last (<index>)	assigns to the given index variable the value of the first/last (smallest/largest) index in the associative array. It returns 0 if the array is empty, and 1 otherwise.
next (<index>), prev (<index>)	finds the entry whose index is greater/smaller than the given index.

Queues and Lists

Centurion University
Shaping Lives... Empowering Communities...

SV has a built-in list mechanism which is ideal for queues, stacks, etc.

A list is basically a variable size array of any SV data type.

```
int q1[$];          // $ represents the 'upper' array boundary
int n, item;

q1 = '{ n, q1 };  // uses concatenate syntax to write n to the left end of q1
q1 = '{ q1, n };  // uses concatenate syntax to write n to the right end of q1
```

Queues and Lists

```
item = q1[0];          // read leftmost ( first ) item from list

item = q1[$];          // read rightmost ( last ) item from list

n = q1.size;           // determine number of items on q1

q1 = q1[1:$];          // delete leftmost ( first ) item of q1

q1 = q1[0:$-1];        // delete rightmost ( last ) item of q1

for (int i=0; i < q1.size; i++)     // step through a list using integers
begin … end

q1 = { };              // clear the q1 list
```

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Queue Methods

size()	Returns the number of items in the queue. If the queue is empty, it returns 0.
insert()	Inserts the given item at the specified index position. Q.insert (i, e) => Q = '{Q[0:i-1], e, Q[i,$]}
delete()	Deletes the item at the specified index position. Q.delete (i) => Q = '{Q[0:i-1], Q[i+1,$]}

Centurion UNIVERSITY
Shaping Lives...
Empow

Queue Methods

pop_front()	Removes and returns the first element of the queue. e = Q.pop_front() => e = Q[0]; Q = Q[1,$]
pop_back()	Removes and returns the last element of the queue. e = Q.pop_back() => e = Q[$]; Q = Q[0,$-1]
push_front()	Inserts the given element at the front of the queue. Q.push_front(e) => Q = '{e, Q}
push_back()	Inserts the given element at the end of the queue. Q.push_back(e) => Q = '{Q, e}