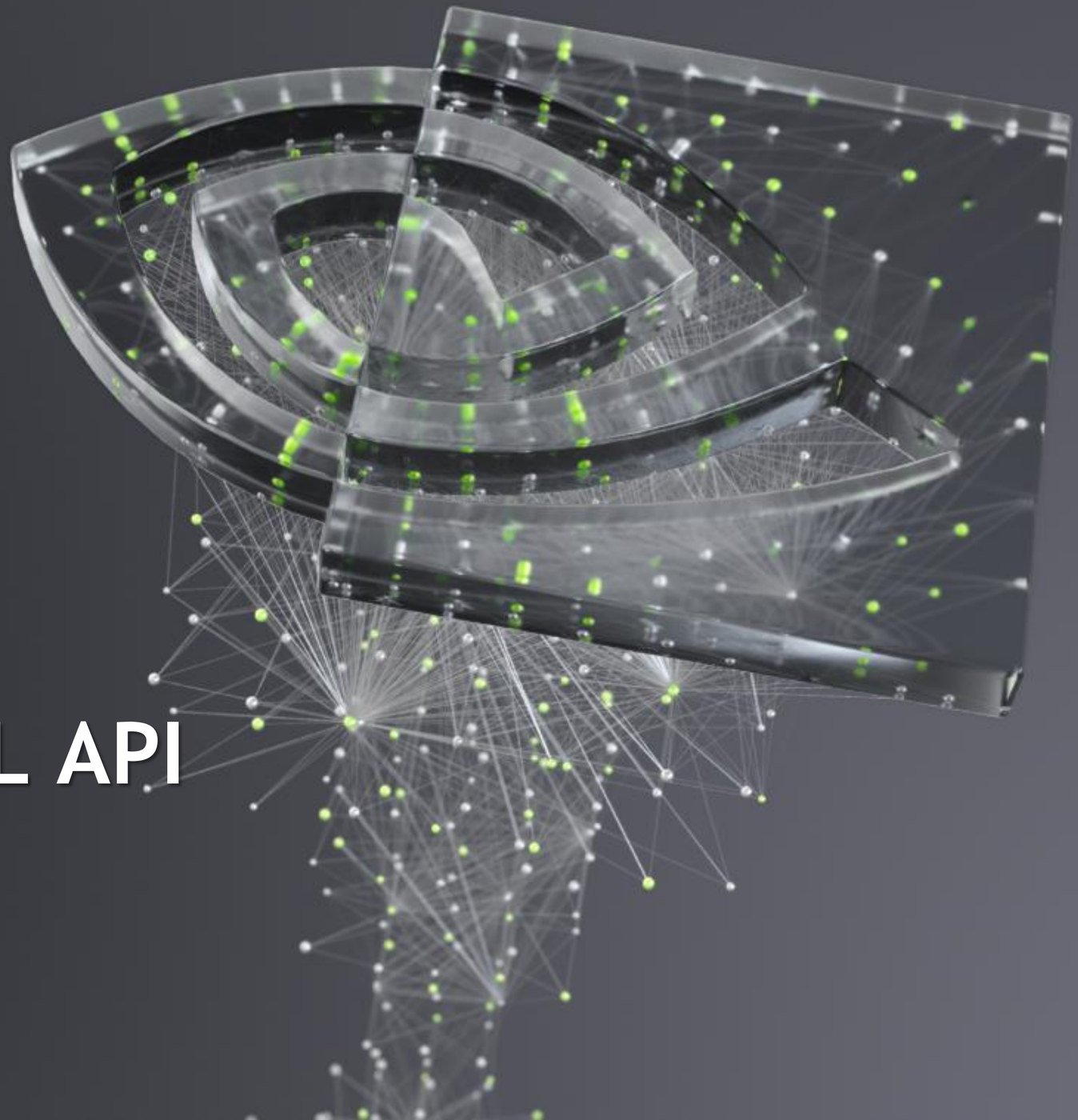


UCD – UCP INTERNAL API PROPOSAL

Artem Y. Polyakov, 12/14/2020

DATA TYPE CREATION

```
/**
* @brief Create a structured datatype.
 *
 * This routine create a structured datatype object. The structured datatype is described by a set of
field
 * descriptors provided through @a desc_ptr @ref ucp_struct_dt_desc_t and @a desc_count.
 * @a rep-count parameter indicates whether or not an array of structures is created The application is
 * responsible for releasing the @a datatype_p  object using
 * @ref ucp_dt_destroy "ucp_dt_destroy()" routine.
 *
 * @param [in]  desc_ptr     And array of descriptiors specifying structure
 *                           elements, in particular:
 *                            - a displacement from the "buffer pointer" provided to communication
operations
 *                            - the desired extent (aka stride)
 *                            - previously created UCP datatype describing this field content
 *                           @ref ucp_struct_dt_desc_t.
 * @param [in]  desc_count   Numebr of field descriptors
 * @param [in]  rep_count    How many time structure has to be repeated
 * @param [out] datatype_p   A pointer to datatype object.
 *
 * @return Error code as defined by @ref ucs_status_t
 */
ucs_status_t ucp_dt_create_struct(ucp_struct_dt_desc_t *desc_ptr,
                                  size_t desc_count, size_t rep_count,
                                  ucp_datatype_t *datatype_p);
```

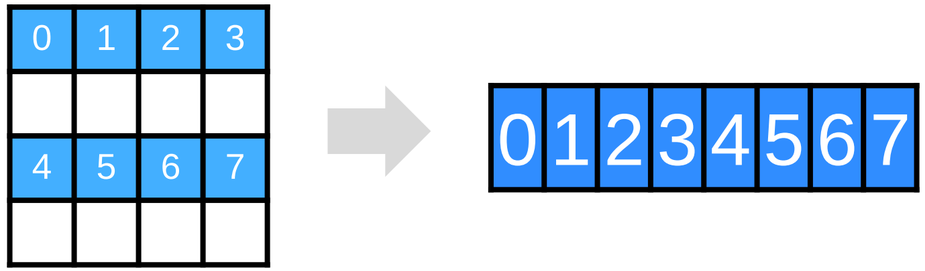
STRUCTURE DESCRIPTOR

```
/**
 * @ingroup UCP_DATATYPE
 * @brief UCP struct data type descriptor
 *
 * This structure provides a structured datatype descriptor that
 * is used for definition of application defined datatypes.
 */

typedef struct ucp_struct_dt_desc {
    ptrdiff_t displ;
    size_t extent;
    ucp_datatype_t dt;
} ucp_struct_dt_desc_t;
```

USAGE EXAMPLE #1

```
payload = 4;
stride = 8;
count = 2;
```




```
typedef struct ucp_struct_dt_desc {
    ptrdiff_t displ;
    size_t extent;
    size_t count;
    ucd_dt_t dt;
} ucp_struct_dt_desc_t;
```

```
typedef struct ucp_struct_dt_desc {
    ucp_struct_dt_desc_t desc[];
    int desc_count;
    int rep_count;
} ucd_int_dt_t;
```

```
ucp_struct_dt_desc_t desc[] = { { 0, 8, ucp_make_contig(4 * sizeof(X)) } }

ucp_dt_create_struct(ucp_struct_dt_desc_t *desc_ptr,
                                    size_t desc_count, size_t rep_count,
                                    ucp_datatype_t *datatype_p)
ucp_dt_create_struct(desc, 1, 2, &new_dt);
```

USAGE EXAMPLE #2

```
payload = 4;
stride = 8;
count = 2;
```

0x100

0	1	2	3
8	9	10	11

```
payload = 4;
stride = 8;
count = 2;
```

0x400

4	5	6	7
12	13	14	15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

OPTION #1

```
ucp_struct_dt_desc_t desc[] = { { displ(0x000), ext(8), dt(ucp_make_contig(4 * X)) },
                                { displ(0x300), ext(8), dt(ucp_make_contig(4 * X)) }  }
ucp_dt_create_struct(desc, desc_cnt = 2, rep_cnt = 2, &new_dt);
```

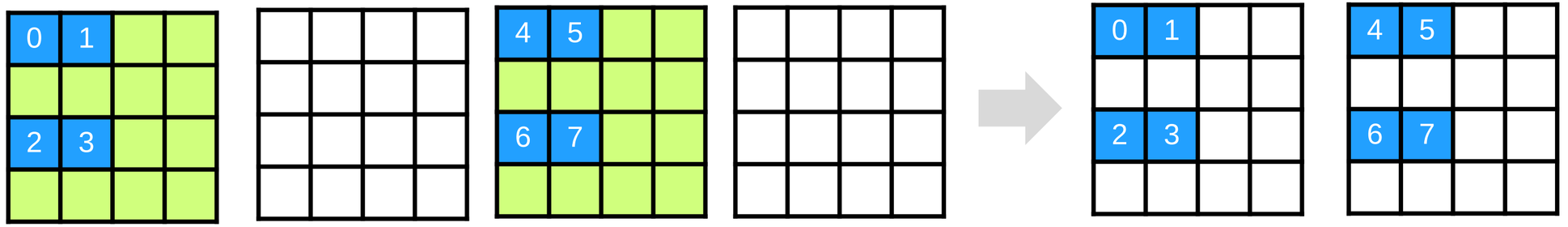
OPTION #2

```
ucp_struct_dt_desc_t desc[] = { { displ(-0x300), ext(8), dt(ucp_make_contig(4 * X)) },
                                { displ( 0x000), ext(8), dt(ucp_make_contig(4 * X)) } }
ucp_dt_create_struct(desc, desc_cnt = 2, rep_cnt = 2, &new_dt);
```

USAGE EXAMPLE #3

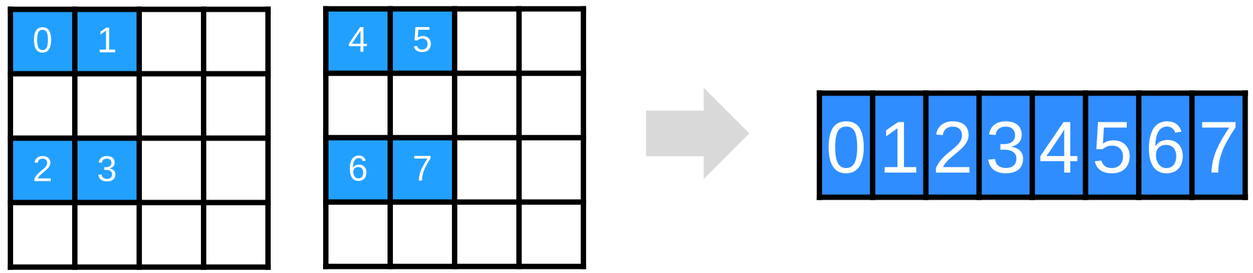
1. Register one **repeated** indirect keys that will correspond to 2nd level data type level, *N* = 2 in our case.

```
payload = 16;
stride = 32;
count = 2;
```



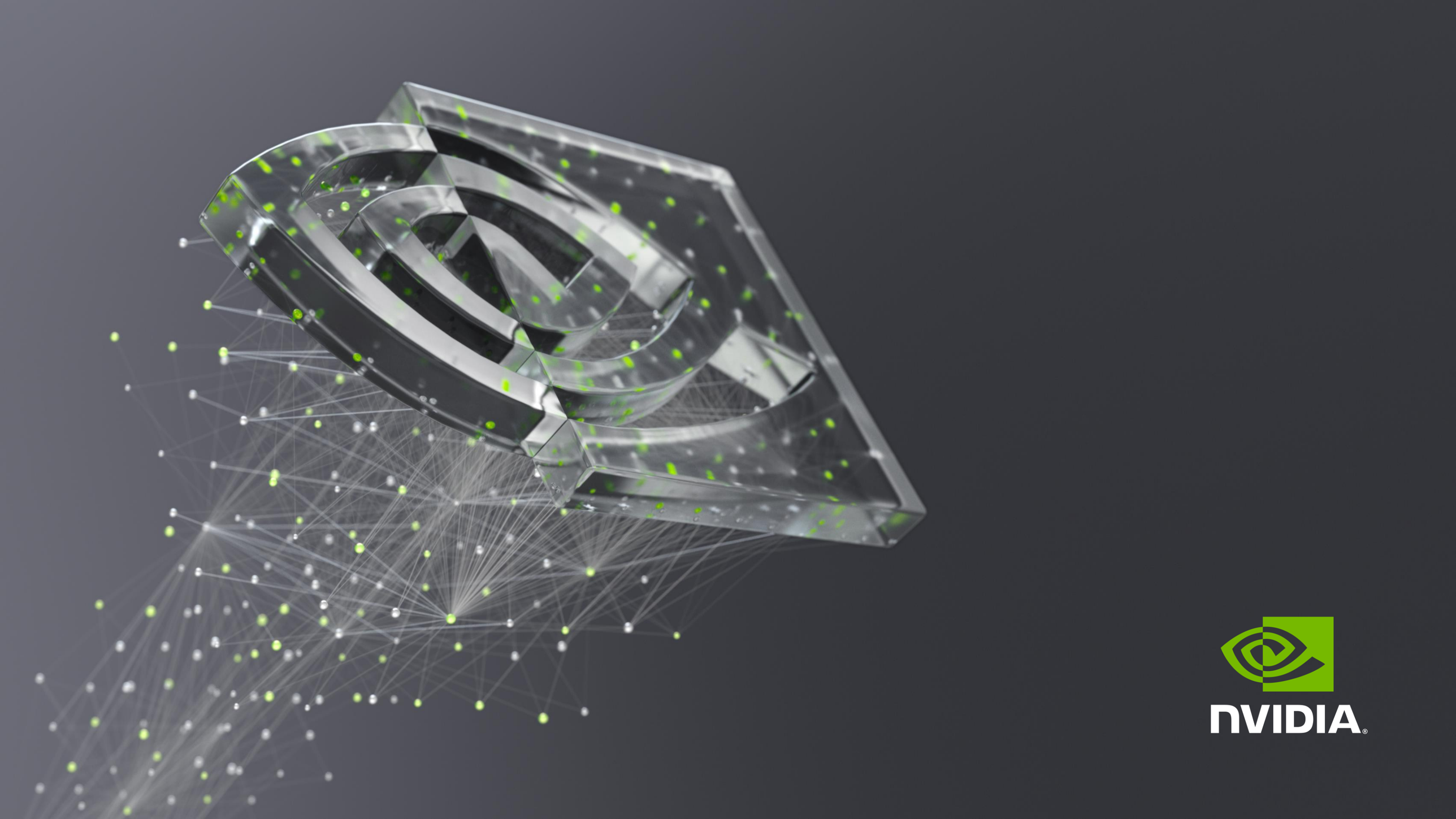
2. Register one top-level **repeated** indirect key that will fetch final elements corresponding to 1st level datatype keys.

```
payload = 2;
stride = 8;
count = 4;
```




```
desc1 = { displ(0), 8, contig(2)}
ucp_dt_create_struct(&desc1, 1, 2, &dt1);

desc2 = { displ(0), ext(32), dt1}
ucp_dt_create_struct(desc2, 1, 2, &dt2);
```


NVIDIA