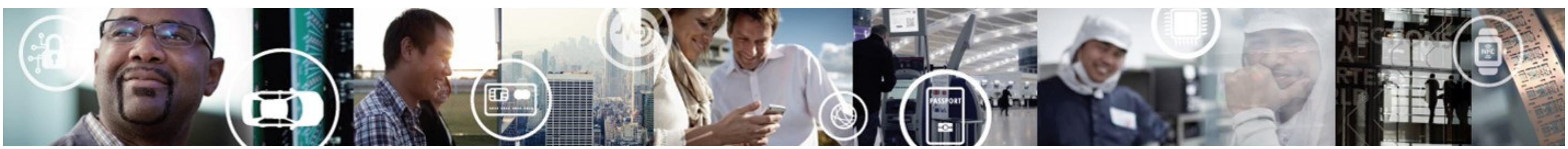
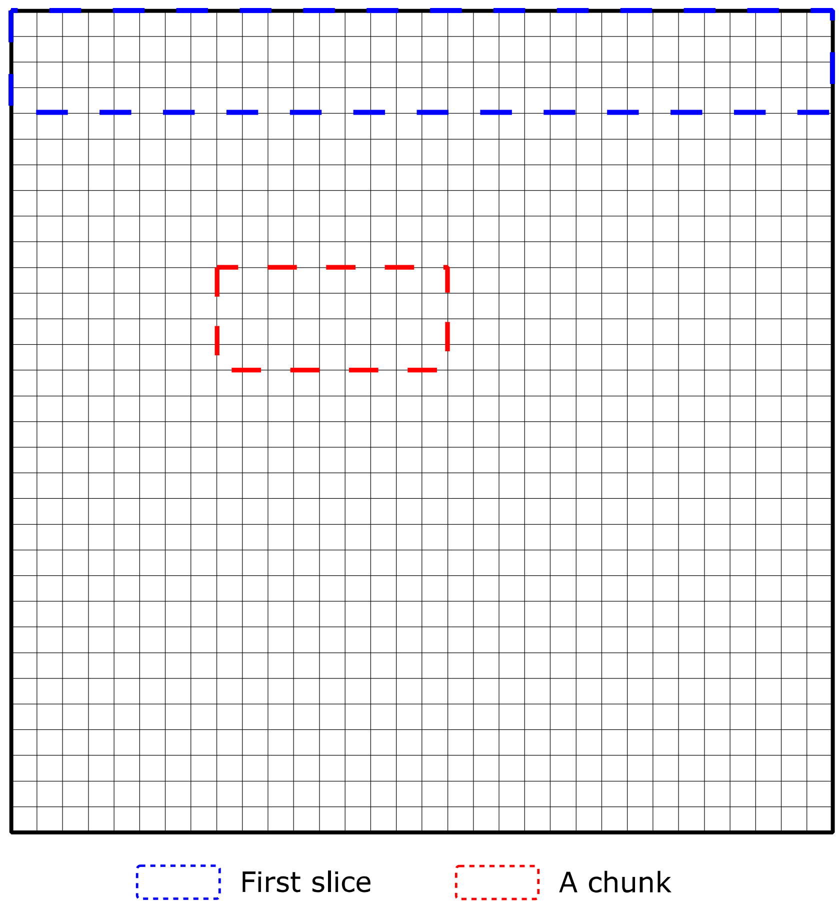
APEX DATA ORGANIZATION AND PROCESSING



SECURE CONNECTIONS
FOR A SMARTER WORLD

Let's start with an image. The first step is to split the image into slices. In this example, we'll use a slice of size 32x4. To process the slice, we must divide it into blocks. We will call these blocks "chunks" hereafter. For each block, represented as multidimensional array, stride measures the number of locations in memory from one row of pixels to the next row of pixels. If padding is presented, the stride will be wider than the width of the core image block.



When splitting the slice into chunks, we need to end up with one chunk for each CU. In this example we'll be using 4 CUs. The width of each chunk is the width of the slice divided by the number of CUs, so here the chunk width will be 32/4 = 8. The chunk height is the same as the slice height. Therefore our chunk size will be 8x4.

The numbers in each box are the data values. Each red box represents a chunk.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F

The data for each chunk will be transferred into the memory of its corresponding CU.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F

$\downarrow$ CU_0 $\quad$ $\downarrow$ CU_1 $\quad$ $\downarrow$ CU_2 $\quad$ $\downarrow$ CU_3

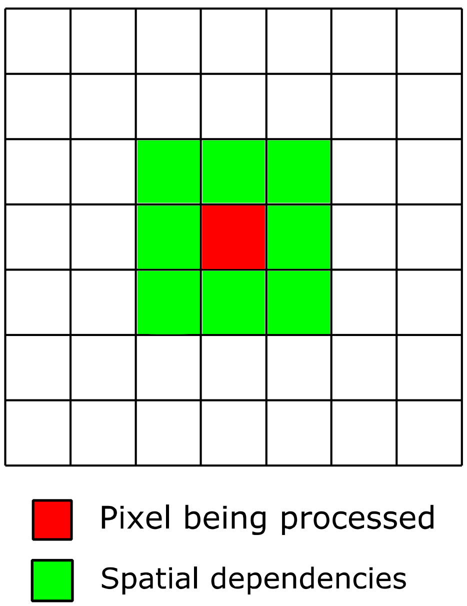
The numbers in blue represent the CMEM addresses. In this example, block stride is 8.

	CU_0	CU_1	CU_2	CU_3
A+00:	00	08	10	18
A+01:	01	09	11	19
A+02:	02	0A	12	1A
A+03:	03	0B	13	1B
A+04:	04	0C	14	1C
A+05:	05	0D	15	1D
A+06:	06	0E	16	1E
A+07:	07	0F	17	1F
A+08:	20	28	30	38
A+09:	21	29	31	39
A+22:	46	4E	56	5E
A+23:	47	4F	57	5F
A+24:	60	68	70	78
A+25:	61	69	71	79
A+26:	62	6A	72	7A
A+27:	63	6B	73	7B
A+28:	64	6C	74	7C
A+29:	65	6D	75	7D
A+30:	66	6E	76	7E
A+31:	67	6F	77	7F

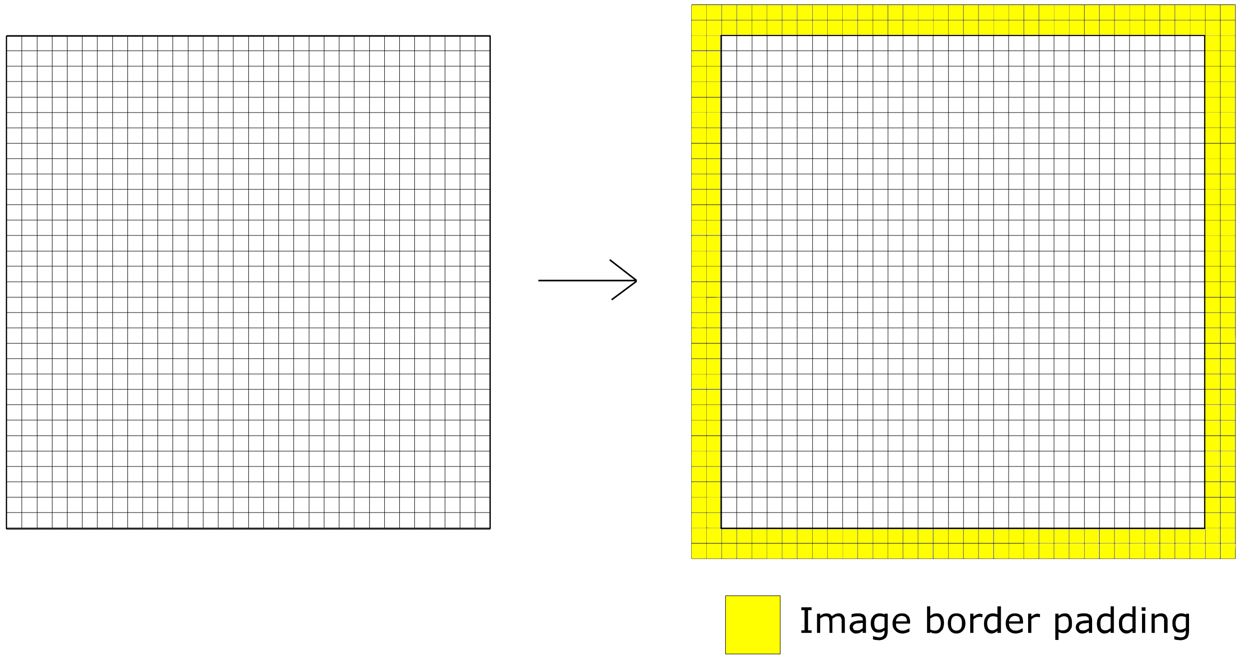
In image processing, some algorithms have spatial dependencies. This means that when they're processing a section of the image, they need to have access to the data that surrounds it.

Ordinarily, a CU will only contain the data from the chunk that it's currently processing. However if we're dealing with an algorithm that has spatial dependencies, we use padding to ensure that the chunk has access to its surrounding data.

In the figure below, we can see how a spatial dependency of 1 pixel on each side maps out.



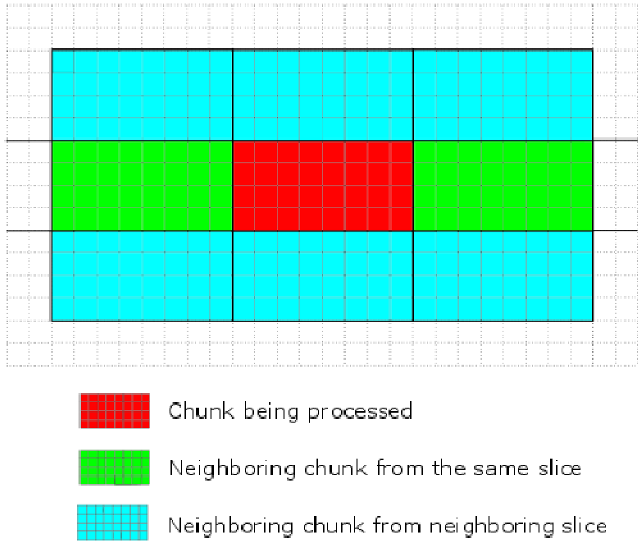
Spatial dependencies add some requirements to properly process the image. The pixels that are on the edge of the image can't be processed because there is no data beyond the border. To solve this problem, we use image border padding that adds padding to the edge of the original image. The data beyond the edge might not exist, so it is often replicated from the edge values while the inner rectangle consists of valid region pixels.



With spatial dependencies, we encounter another problem when dealing with chunks: the kernel needs to have access to data from the neighboring chunks to process the pixels at the edge of the chunks. To do so, we perform chunk padding. Chunk padding reduces the complexity of kernel implementation and increases its reusability.

There are two types of chunk padding: vertical padding and horizontal padding.

Vertical padding is used to access data that is above or below the chunk being processed. Since the chunk has a set height, that data will come from the chunk in neighboring slice that is either above or below the current slice.



Horizontal padding is used to access data that is to the left or right of the chunk being processed.

Let’s go back to our slice example. This time we’re going to assume that we have a spatial dependency of 1 pixel on each side. Since this is the topmost slice, we need to pad perform image border padding by pixel replication. We also need to perform chunk padding. Here is what it would look like:

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71

00	00	01	02	03	04	05	06	07	08
00	00	01	02	03	04	05	06	07	08
20	20	21	22	23	24	25	26	27	28
40	40	41	42	43	44	45	46	47	48
60	60	61	62	63	64	65	66	67	68
80	80	81	82	83	84	85	86	87	88

07	08	09	0A	0B	0C	0D	0E	0F	10
07	08	09	0A	0B	0C	0D	0E	0F	10
27	28	29	2A	2B	2C	2D	2E	2F	30
47	48	49	4A	4B	4C	4D	4E	4F	50
67	68	69	6A	6B	6C	6D	6E	6F	70
87	88	89	8A	8B	8C	8D	8E	8F	90

Image border padding

Horizontal chunk padding

Vertical chunk padding

After the transfer, each CU contains the data from its corresponding chunk.
Note that chunk padding is automatically managed and provided
by the APEX Core Framework.

CU_0

00	00	01	02	03	04	05	06	07	08
00	00	01	02	03	04	05	06	07	08
20	20	21	22	23	24	25	26	27	28
40	40	41	42	43	44	45	46	47	48
60	60	61	62	63	64	65	66	67	68
80	80	81	82	83	84	85	86	87	88

CU_1

07	08	09	0A	0B	0C	0D	0E	0F	10
07	08	09	0A	0B	0C	0D	0E	0F	10
27	28	29	2A	2B	2C	2D	2E	2F	30
47	48	49	4A	4B	4C	4D	4E	4F	50
67	68	69	6A	6B	6C	6D	6E	6F	70
87	88	89	8A	8B	8C	8D	8E	8F	90

CU_2

0F	10	11	12	13	14	15	16	17	18
0F	10	11	12	13	14	15	16	17	18
2F	30	31	32	33	34	35	36	37	38
4F	50	51	52	53	54	55	56	57	58
6F	70	71	72	73	74	75	76	77	78
8F	90	91	92	93	94	95	96	97	98

CU_3

17	18	19	1A	1B	1C	1D	1E	1F	1F
17	18	19	1A	1B	1C	1D	1E	1F	1F
37	38	39	3A	3B	3C	3D	3E	3F	3F
57	58	59	5A	5B	5C	5D	5E	5F	5F
77	78	79	7A	7B	7C	7D	7E	7F	7F
97	98	99	9A	9B	9C	9D	9E	9F	9F

	CU_0	CU_1	CU_2	CU_3
A -11:	00	07	0F	17
A -10:	00	08	10	18
A -09:	01	09	11	19
A -08:	02	0A	12	1A
A -07:	03	0B	13	1B
A -06:	04	0C	14	1C
A -05:	05	0D	15	1D
A -04:	06	0E	16	1E
A -03:	07	0F	17	1F
A -02:	08	10	18	1F
A -01:	00	07	0F	17
A+00:	00	08	10	18
A+01:	01	09	11	19
A+02:	02	0A	12	1A
A+36:	65	6D	75	7D
A+37:	66	6E	76	7E
A+38:	67	6F	77	7F
A+39:	68	70	78	7F
A+40:	80	87	8F	97
A+41:	80	88	90	98
A+42:	81	89	91	99
A+43:	82	8A	92	9A
A+44:	83	8B	93	9B
A+45:	84	8C	94	9C
A+46:	85	8D	95	9D
A+47:	86	8E	96	9E
A+48:	87	8F	97	9F
A+49:	88	90	98	9F

The numbers in blue represent the CMEM addresses.
Note that the base address (A) that gets passed into the kernel will be the pointing to the beginning of the chunk, not the beginning of the padding. It is marked in green.

SECURE CONNECTIONS
FOR A SMARTER WORLD