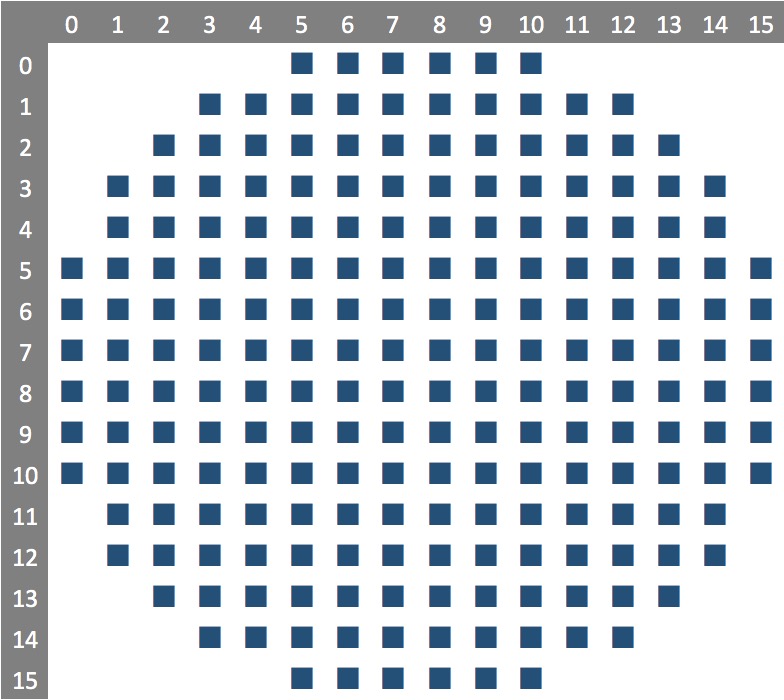
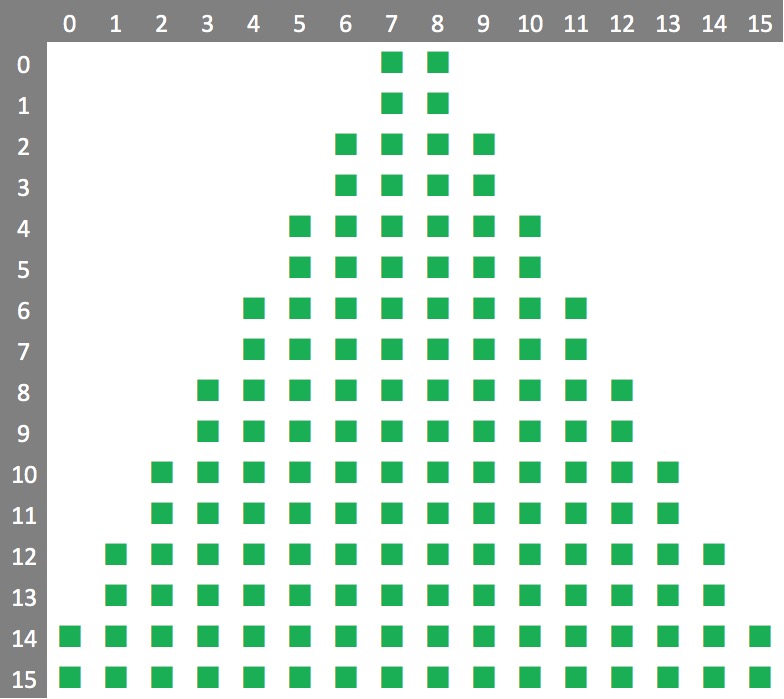
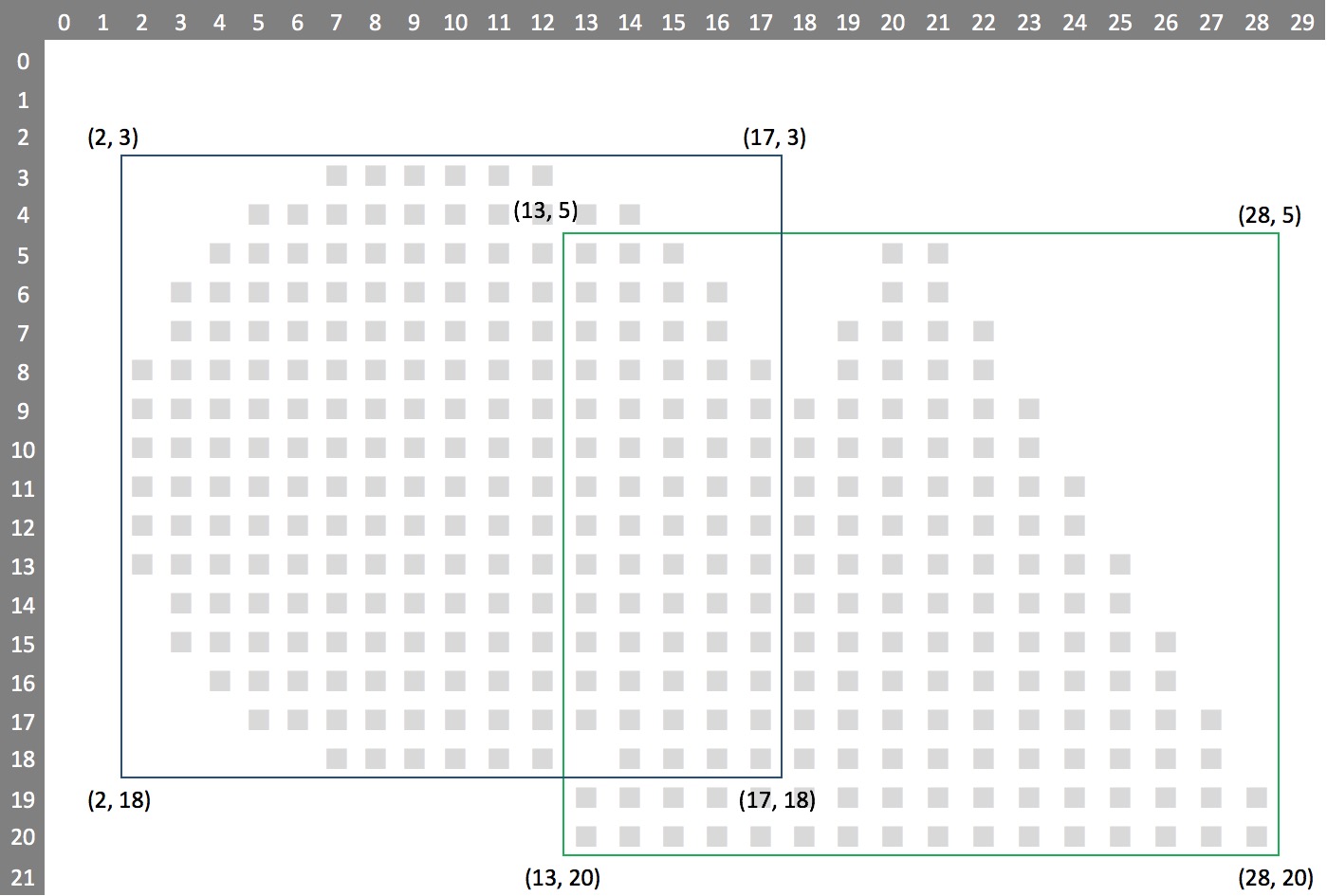
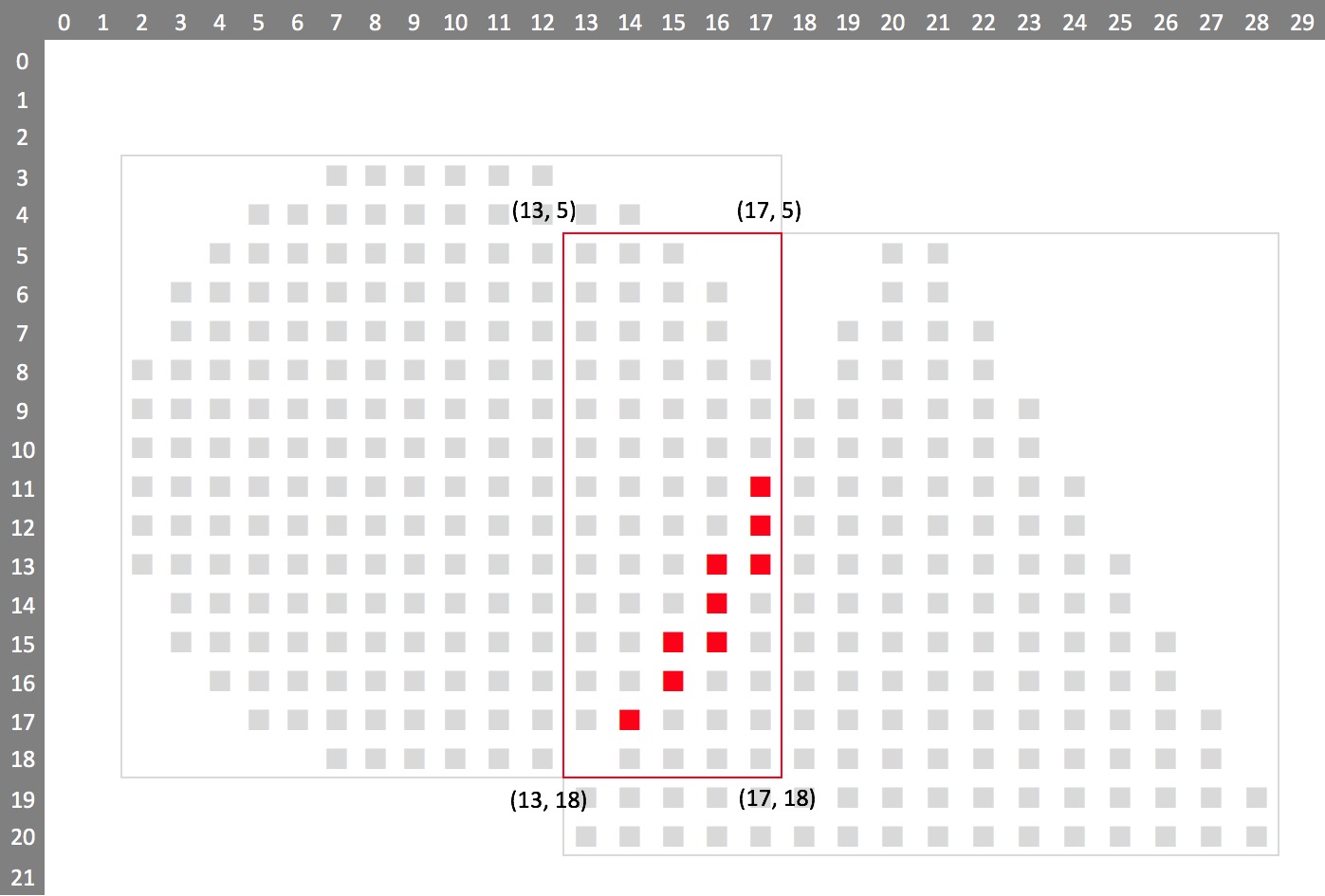
Consider the two images below:



The first image starts at position (2, 3) and extends 16 bits either way to (17, 18) inclusive.  The second image starts at position (13, 5) and extends 16 pixels either way to (28, 20) inclusive.



The overlapping rectangle area can be determined by applying the following formulas:

overlap\_left = max( img1.x, img2.x );

       = max( 2, 13 );

       = 13;

overlap\_top = max(img1.y, img2.y );

        = max( 3, 5 );

        = 5;

overlap\_right = min(img1.x + img1.width, img2.x + img2.width );

          = min( 2 + 16, 13 + 16 );

          = min( 18,  29 );

          = 18;

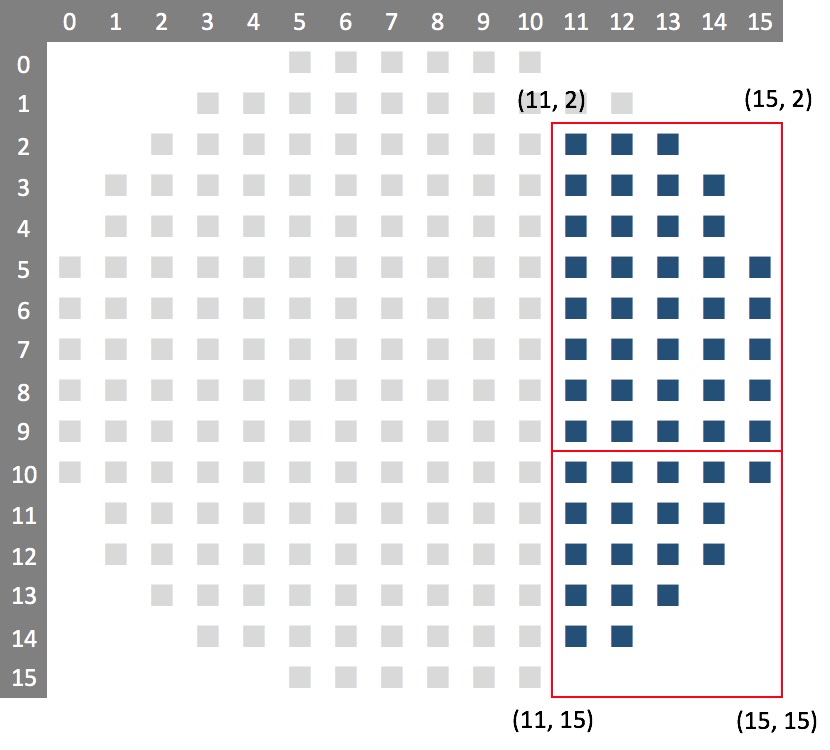
overlap\_bottom = min(img1.y + img1.width, img2. y + img2.width );

               = min( 3 + 16, 5 + 16 );

               = min( 19, 21 );

               = 19;

Overlapping rectangle starts at (13, 5) in the top left-hand corner and extends to (17, 18) in the bottom right-hand corner, inclusive. Note that our calculation returned a right and bottom value that were exclusive.



img1\_left = (overlap\_left – img1.x);

= 13 – 2;

= 11;

img1\_right = (overlap\_right - img1.x);

= 18 – 2;

= 16;

img1\_top\_row = (overlap\_top - img1.y) / 8;

= (5 – 3) / 8;

= 0;

img1\_top\_bit = (overlap\_top - img1.y) % 8;

= (5 – 3) % 8;

= 2;

img1\_bottom\_row = (overlap\_bottom - img1.y - 1) / 8;

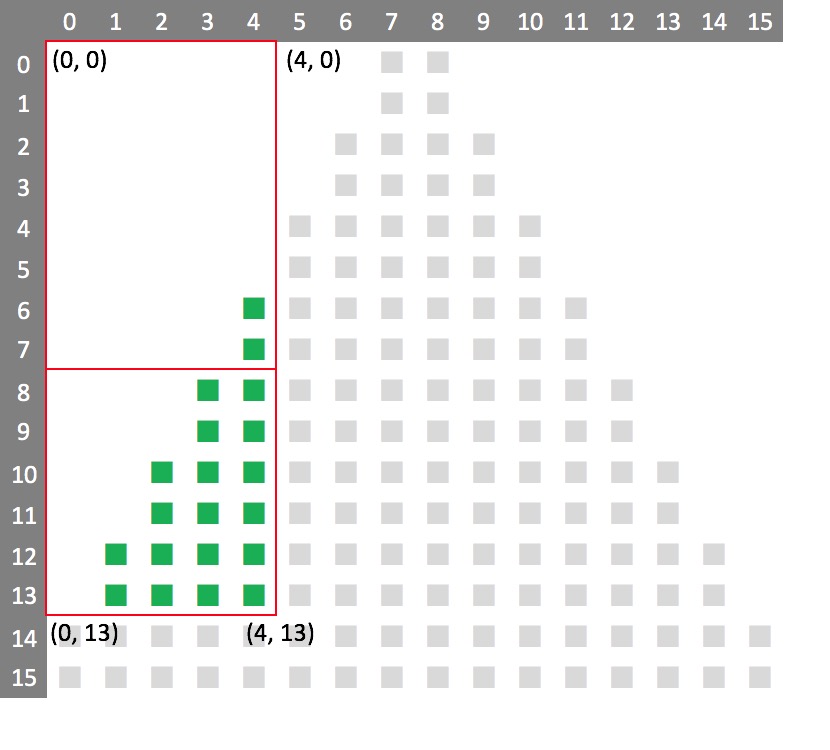
= (19 – 3 – 1) / 8;

= 1;

img1\_bottom\_bit = (overlap\_bottom - img1.y) % 8;

= (19 – 3) % 8;

= 0;



img2\_left = (overlap\_left – img2.x);

= 13 – 13;

= 0;

img2\_right = (overlap\_right – img2.x);

= 18 – 13;

= 5;

img2\_top\_row = (overlap\_top – img2.y) / 8;

= (5 – 5) / 5;

= 0;

img2\_top\_bit = (overlap\_top – img2.y) % 8;

= (5 – 5) % 8;

= 0;

img2\_bottom\_row = (overlap\_bottom – img2.y - 1) / 8;

= (19 – 5 – 1) / 8;

= 1;

img2\_bottom\_bit = (overlap\_bottom – img2.y) % 8;

= (19 – 5) % 8;

= 6;

const uint8\_t PROGMEM lookup[] { 0xFF >> 8, 0xFF >> 7, 0xFF >> 6, 0xFF >> 5, 0xFF >> 4, 0xFF >> 3, 0xFF >> 2, 0xFF >> 1 };

int16\_t i1 = (img1\_top\_row \* w1) + img1\_left + IMG\_DATA\_OFFSET;

uint16\_t d1 = pgm\_read\_byte(&img1[i1]) & (img1\_top\_row == img1\_bottom\_row && img1\_bottom\_bit != 0 ? pgm\_read\_byte(&lookup[img1\_bottom\_bit]) : 0xFF);

uint16\_t d1 = pgm\_read\_byte(&img1[i1])

= 0xFE;

this is ignored..

& (img1\_top\_row == img1\_bottom\_row && img1\_bottom\_bit != 0 ? pgm\_read\_byte(&lookup[img1\_bottom\_bit]) : 0xFF);

If the 8 bits of data we are retrieving spans over two bytes in the image, we retrieve the next byte down

0X7F

Then because this is the lowest row, we mask of any bits not required ..

if (img1\_top\_bit > 0 && img1\_top\_row < img1\_bottom\_row) {

d1 = d1 | ((pgm\_read\_byte(&img1[i1 + w1]) & (img1\_top\_row + 1 == img1\_bottom\_row ? pgm\_read\_byte(&lookup[img1\_bottom\_bit]) : 0xFF )) << 8);

}

Finally,

d1 = (d1 >> img1\_top\_bit) & 0xFF;