Deep Neural Networks
Convolutional Networks III

Bhiksha Raj

Spring 2025

Outline

- Quick recap
- Back propagation through a CNN
- Modifications: Transposition, scaling, rotation and deformation invariance
- Segmentation and localization
- Some success stories
- Some advanced architectures
  - Resnet
  - Densenet

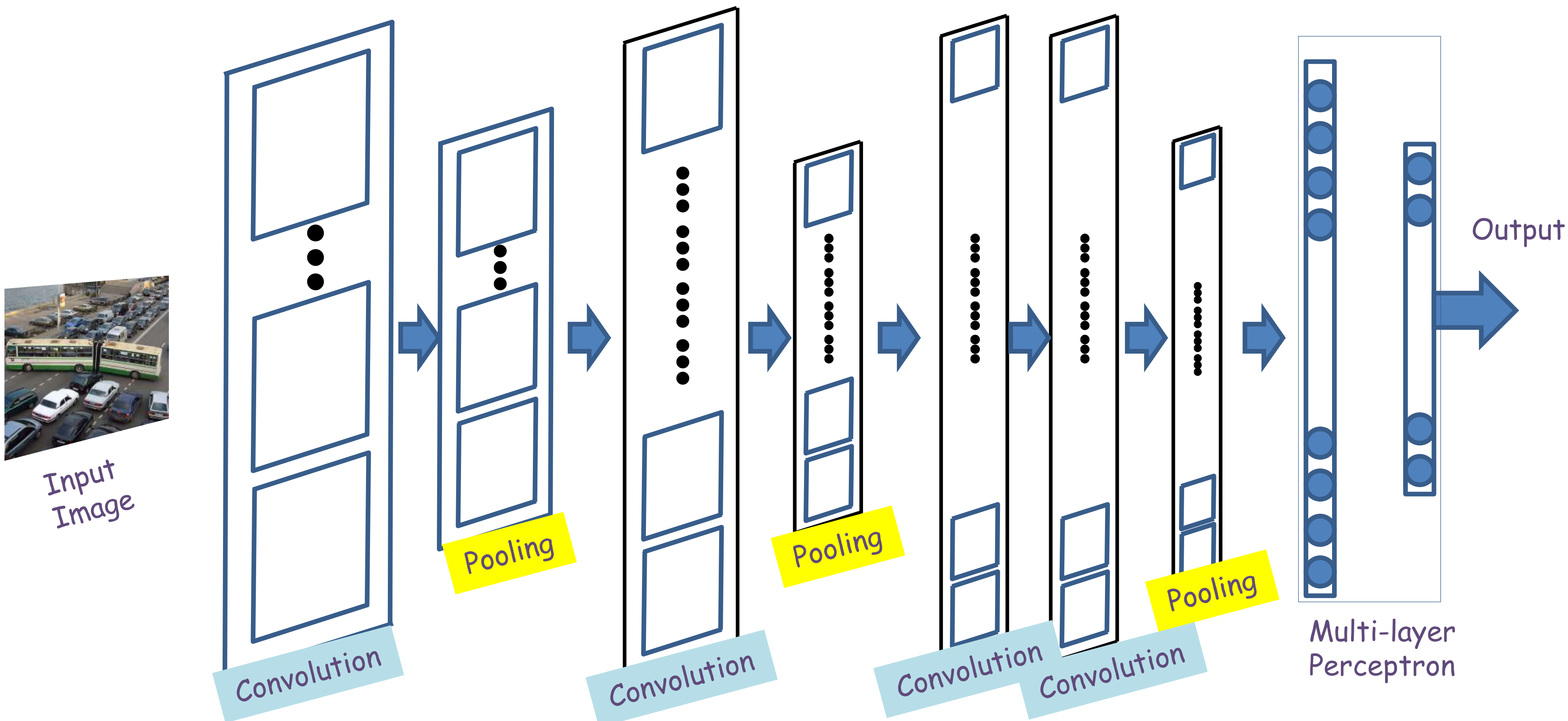
Story so far

- Pattern classification tasks such as “does this picture contain a cat”, or “does this recording include HELLO” are best performed by scanning for the target pattern

- Scanning an input with a network and combining the outcomes is equivalent to scanning with individual neurons hierarchically
  - First level neurons scan the input
  - Higher-level neurons scan the “maps” formed by lower-level neurons
  - A final “decision” unit or layer makes the final decision
  - Deformations in the input can be handled by “pooling”

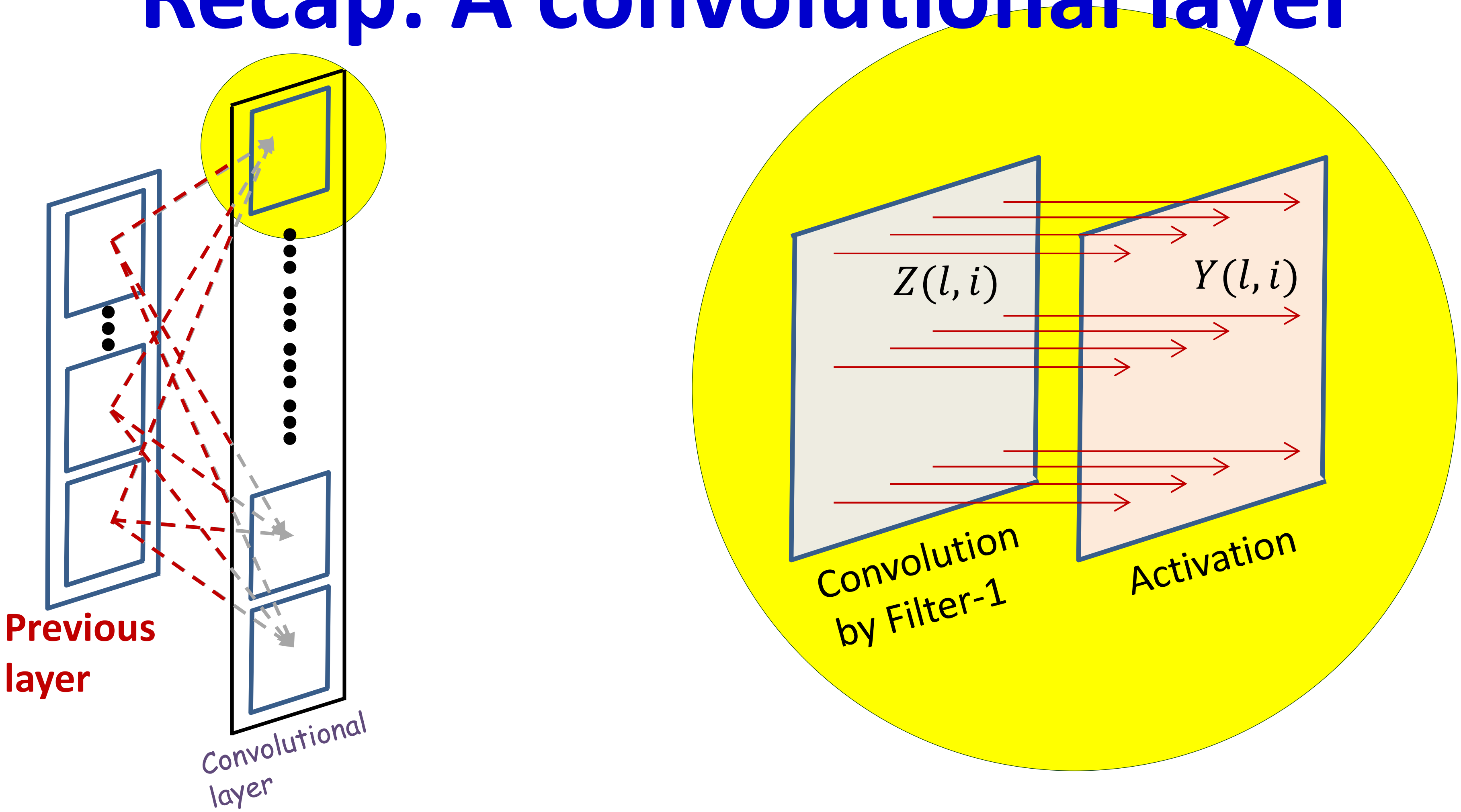
- For 2-D (or higher-dimensional) scans, the structure is called a Convolutional Neural Network
- For 1-D scan along time, it is called a Time-delay neural network

Recap: The general architecture of a convolutional neural network



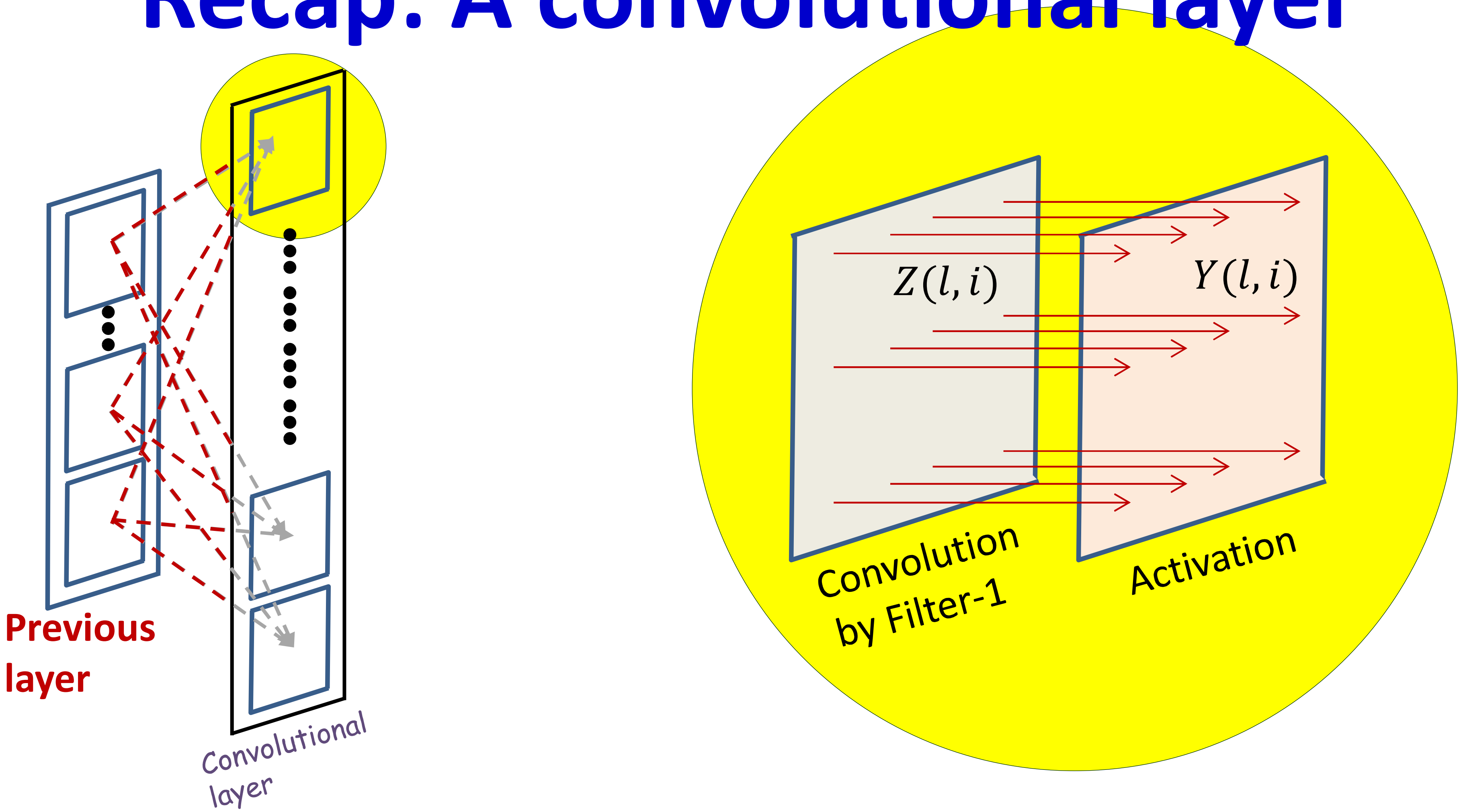
- A convolutional neural network comprises of "convolutional" and optional "pooling" layers
- Followed by an MLP with one or more layers

Recap: A convolutional layer



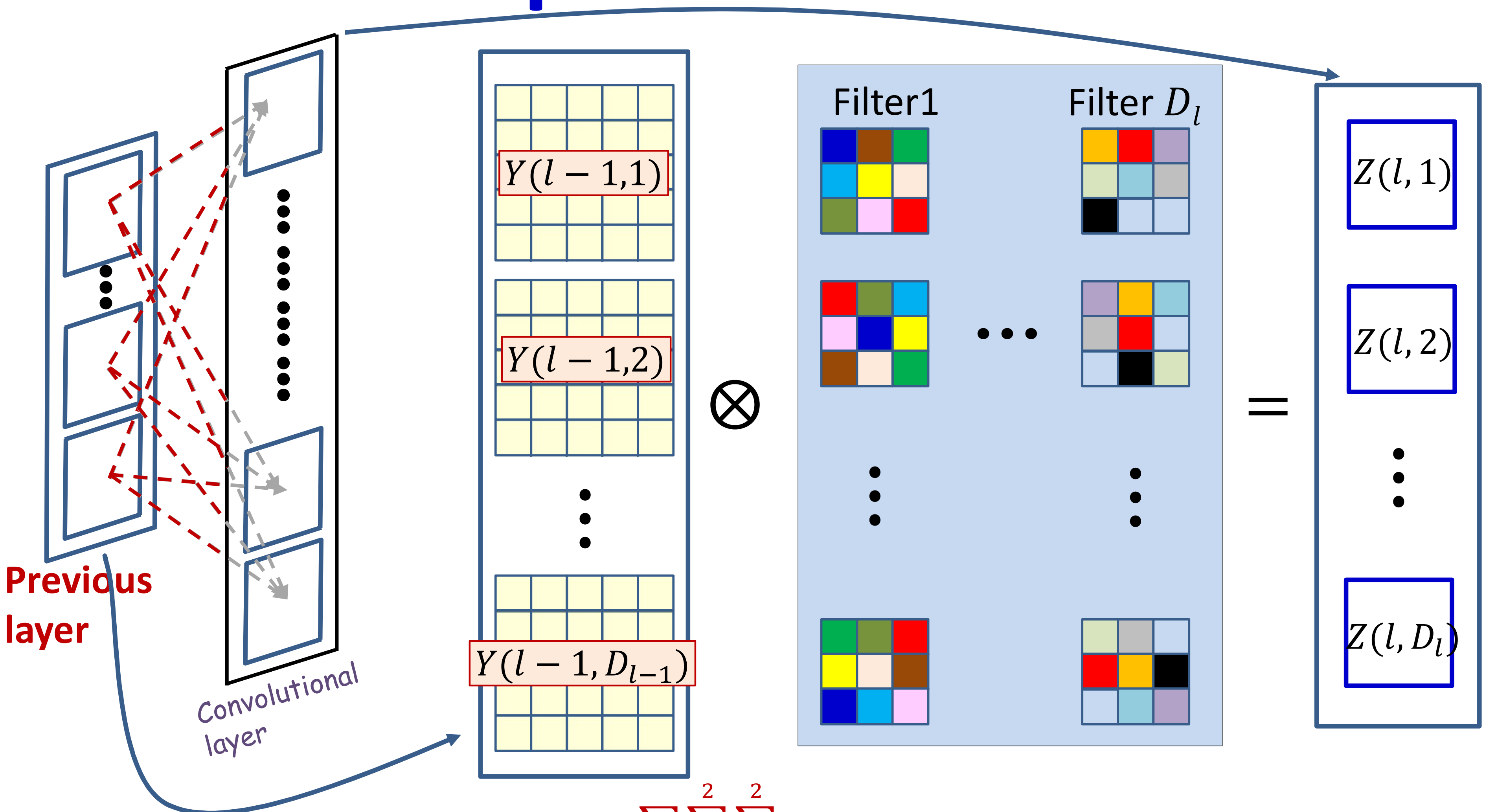
- The computation of each output map has two stages
  - Computing an *affine* map, by *convolving* a *filter* (representing a pattern of weights) over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates *point-wise* on the output of the convolution

Recap: A convolutional layer



- The computation of each output map has two stages
  - Computing an *affine* map, by *convolving* a *filter* (representing a pattern of weights) over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates *point-wise* on the output of the convolution

Recap: Convolution

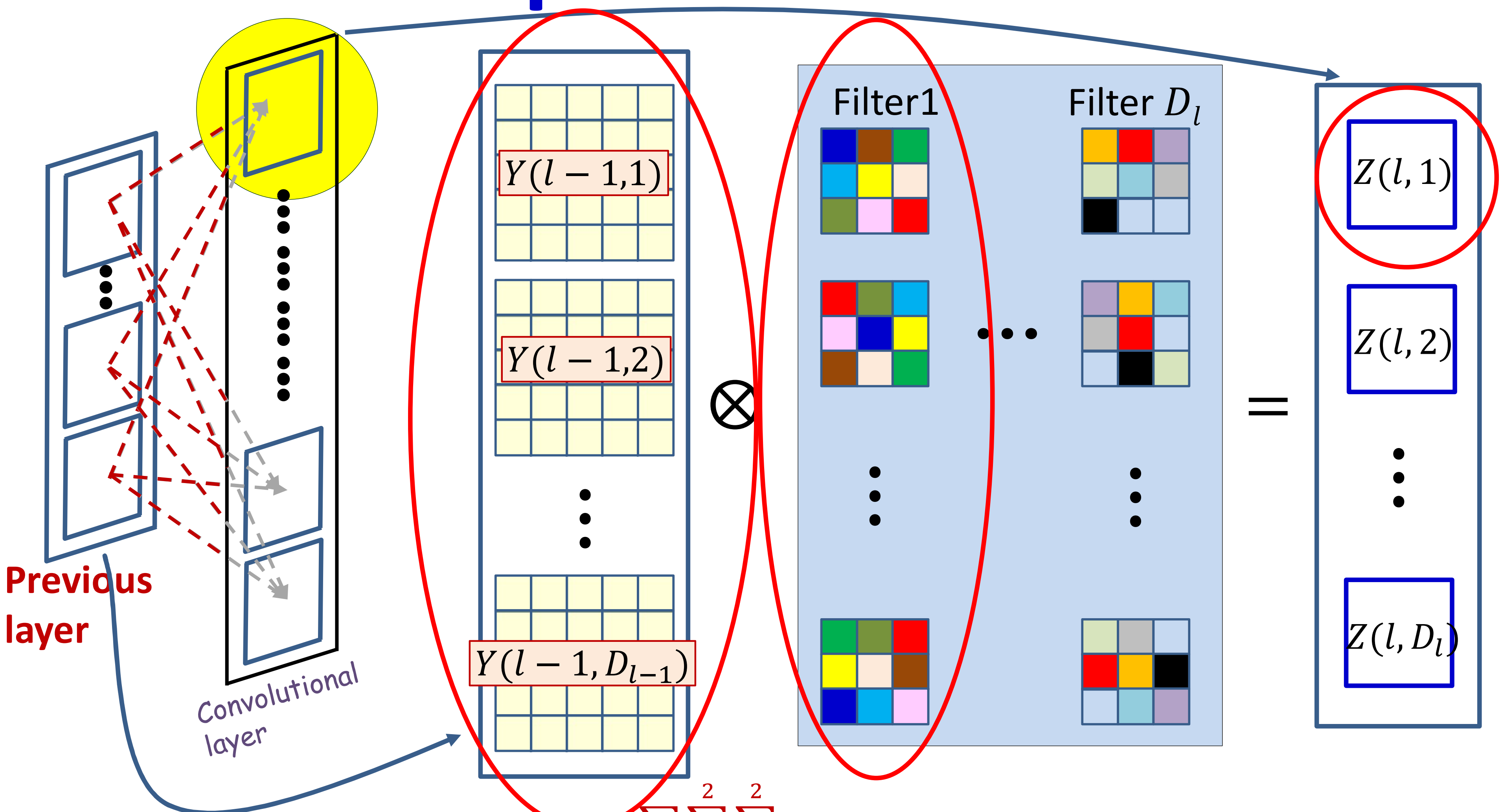


Caveat : 0-based indexing

$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output map is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

Recap: Convolution



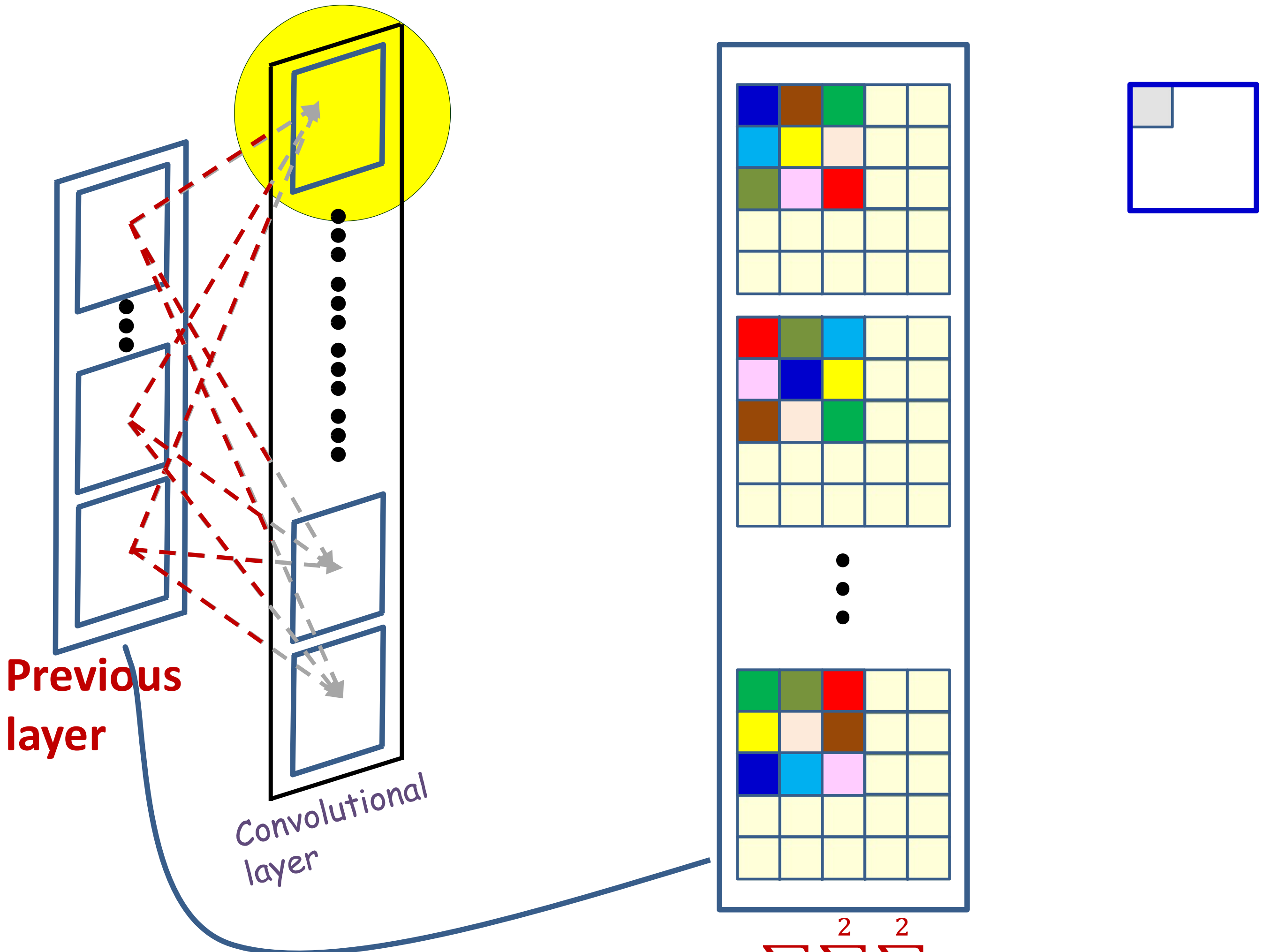
Caveat : 0-based indexing

$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

Recap: Convolution

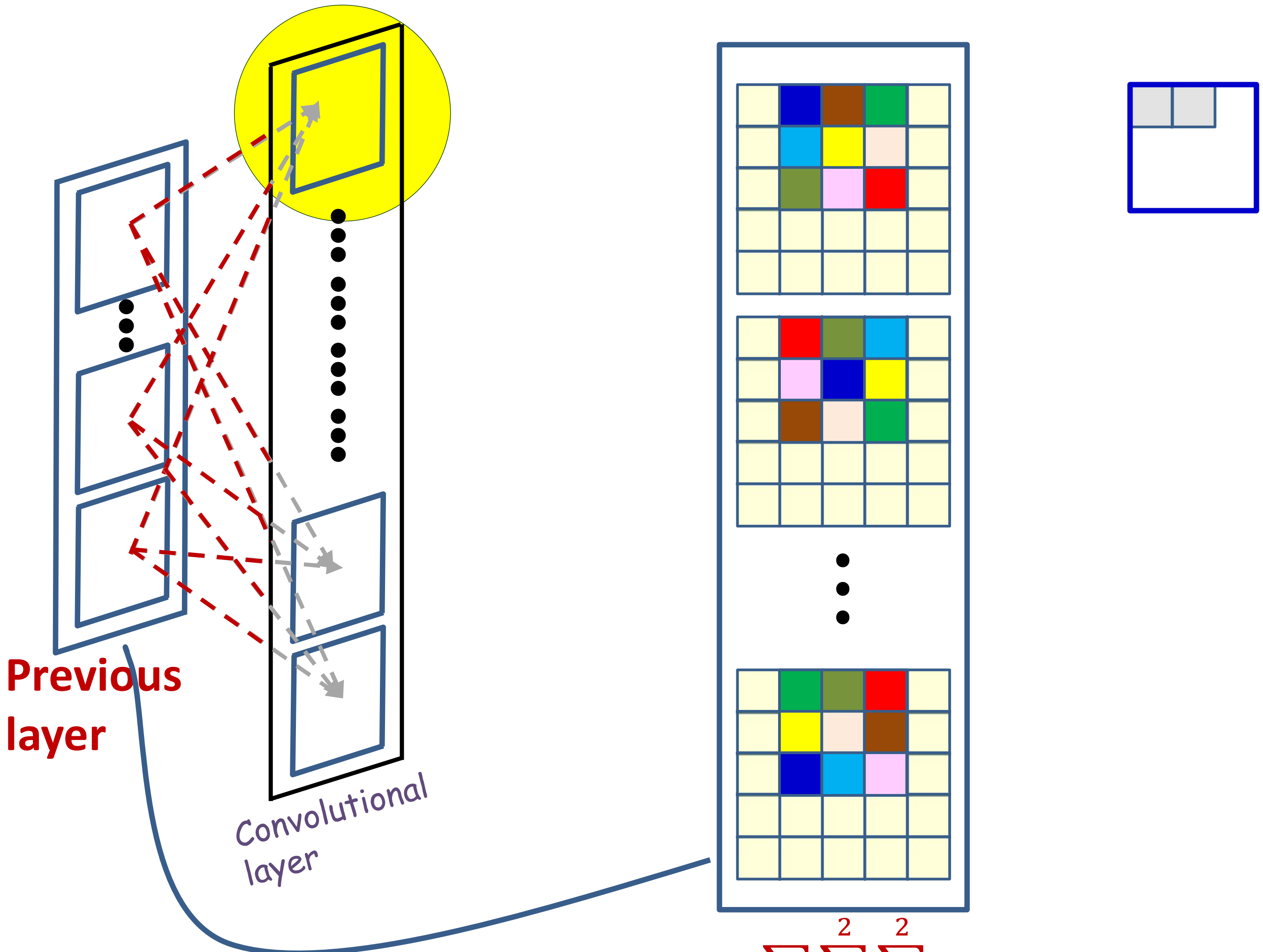


$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

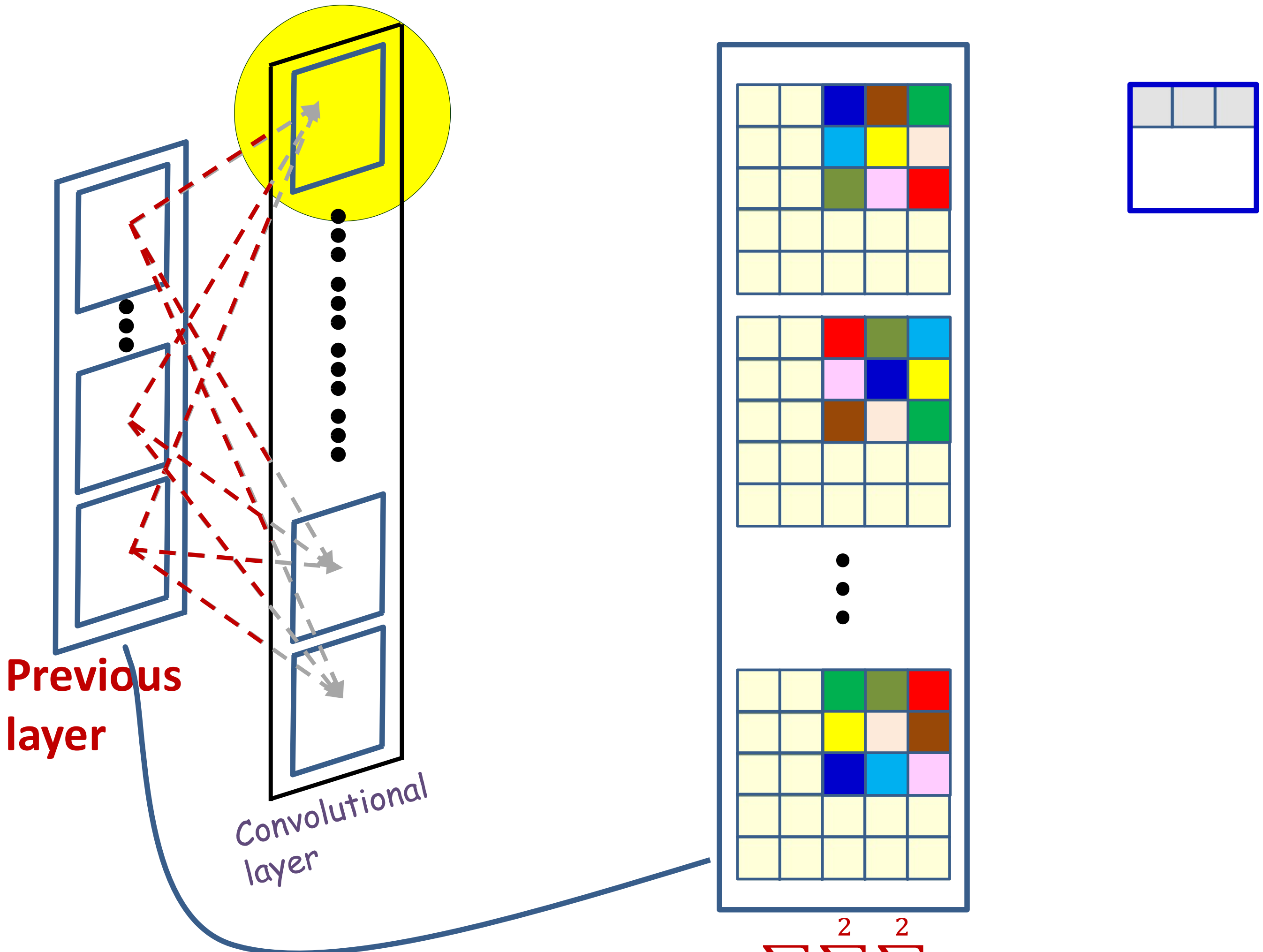
Recap: Convolution



$$z(l,n,x,y) = \sum_{m} \sum_{i=0}^{2} \sum_{j=0}^{2} w_l(m,n,i,j) y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

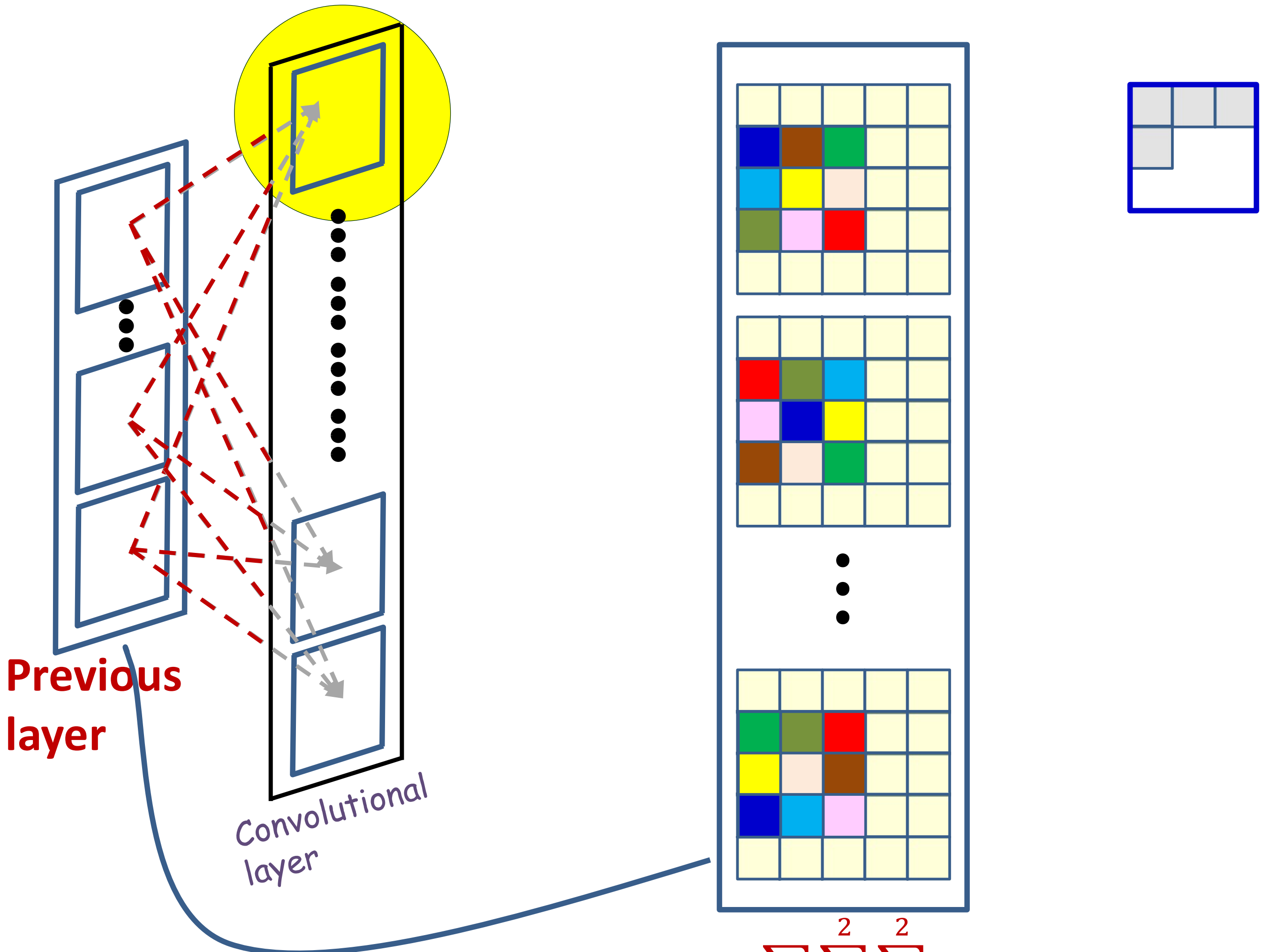
Recap: Convolution



$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

Recap: Convolution

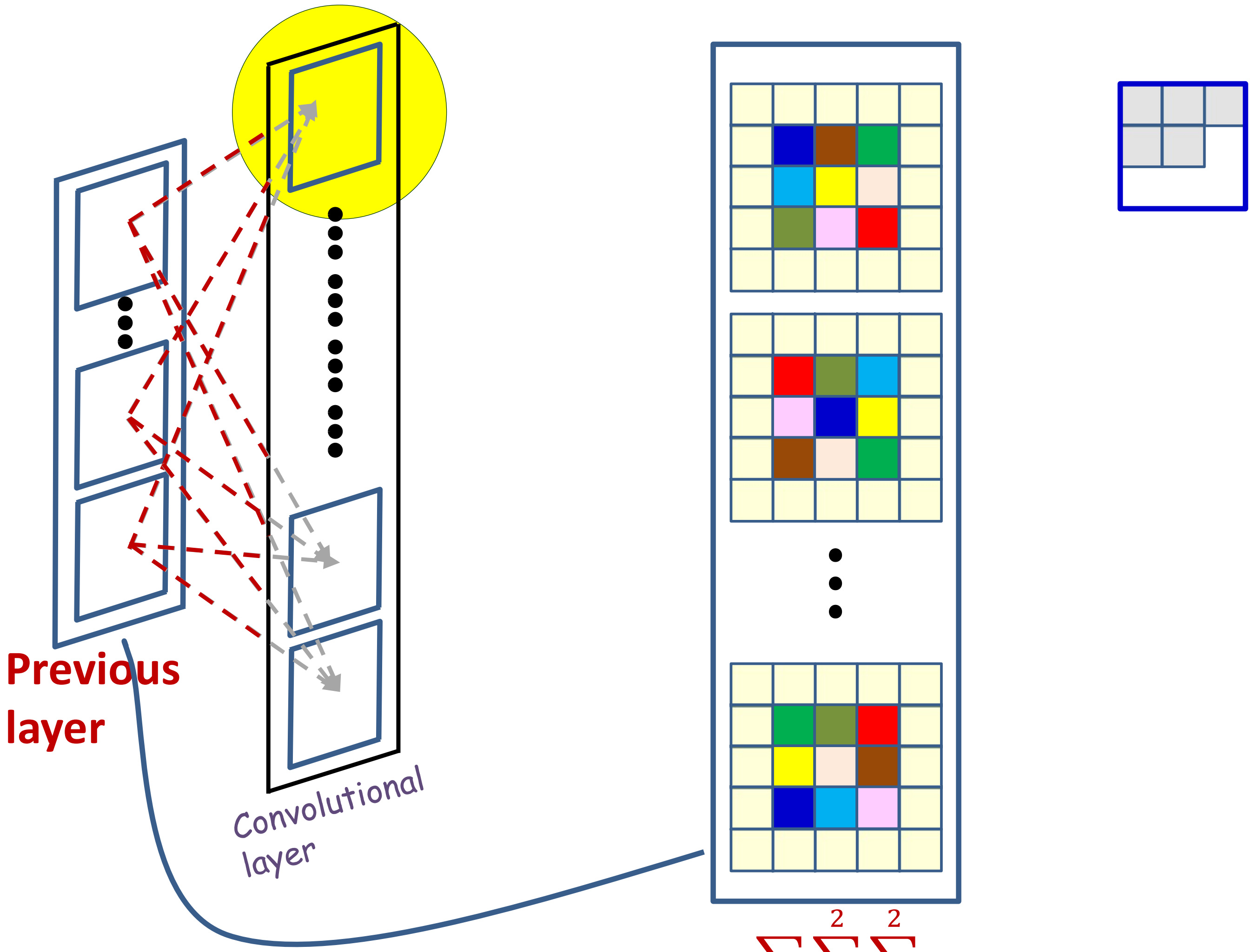


$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

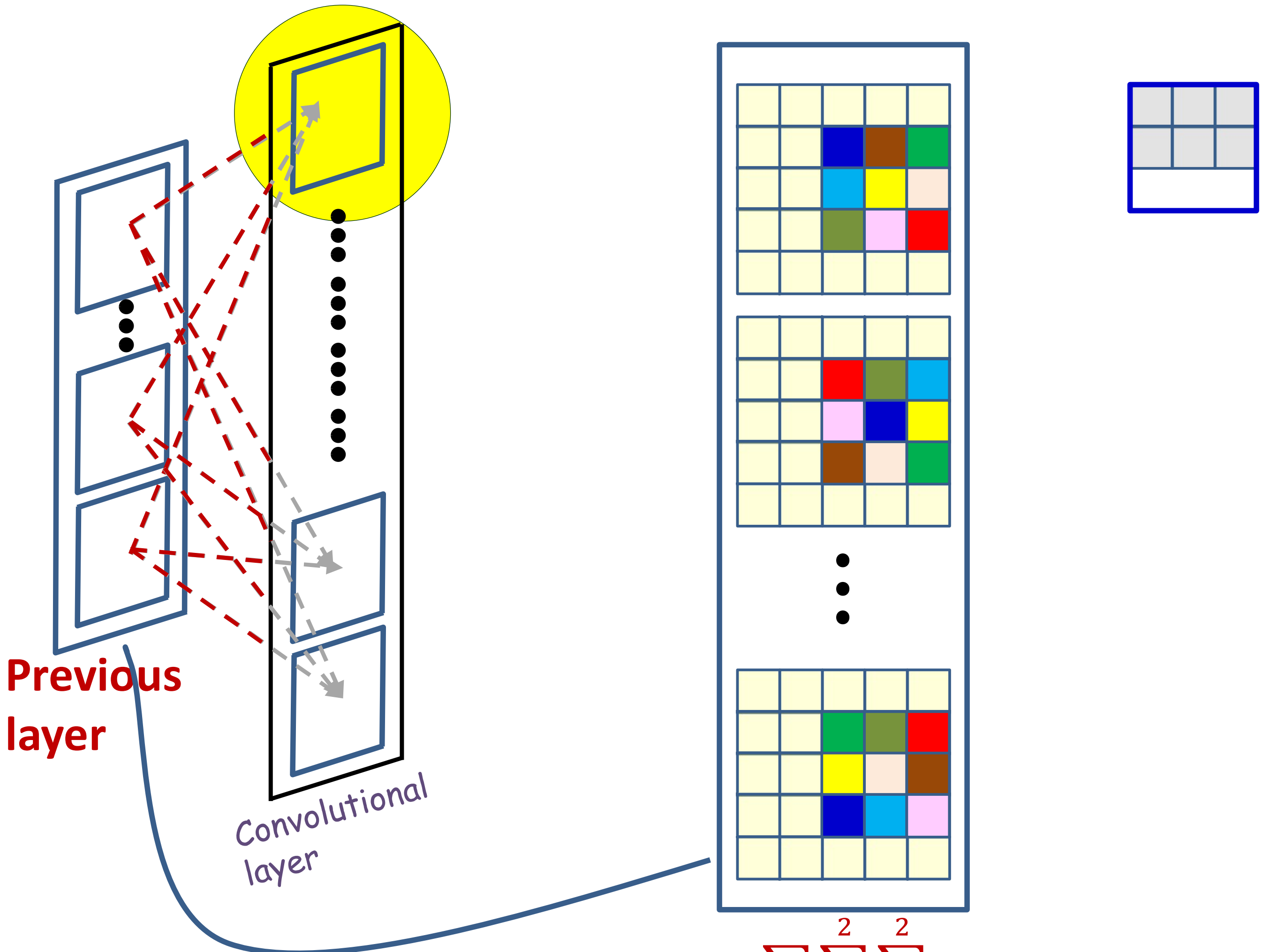
Recap: Convolution



$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

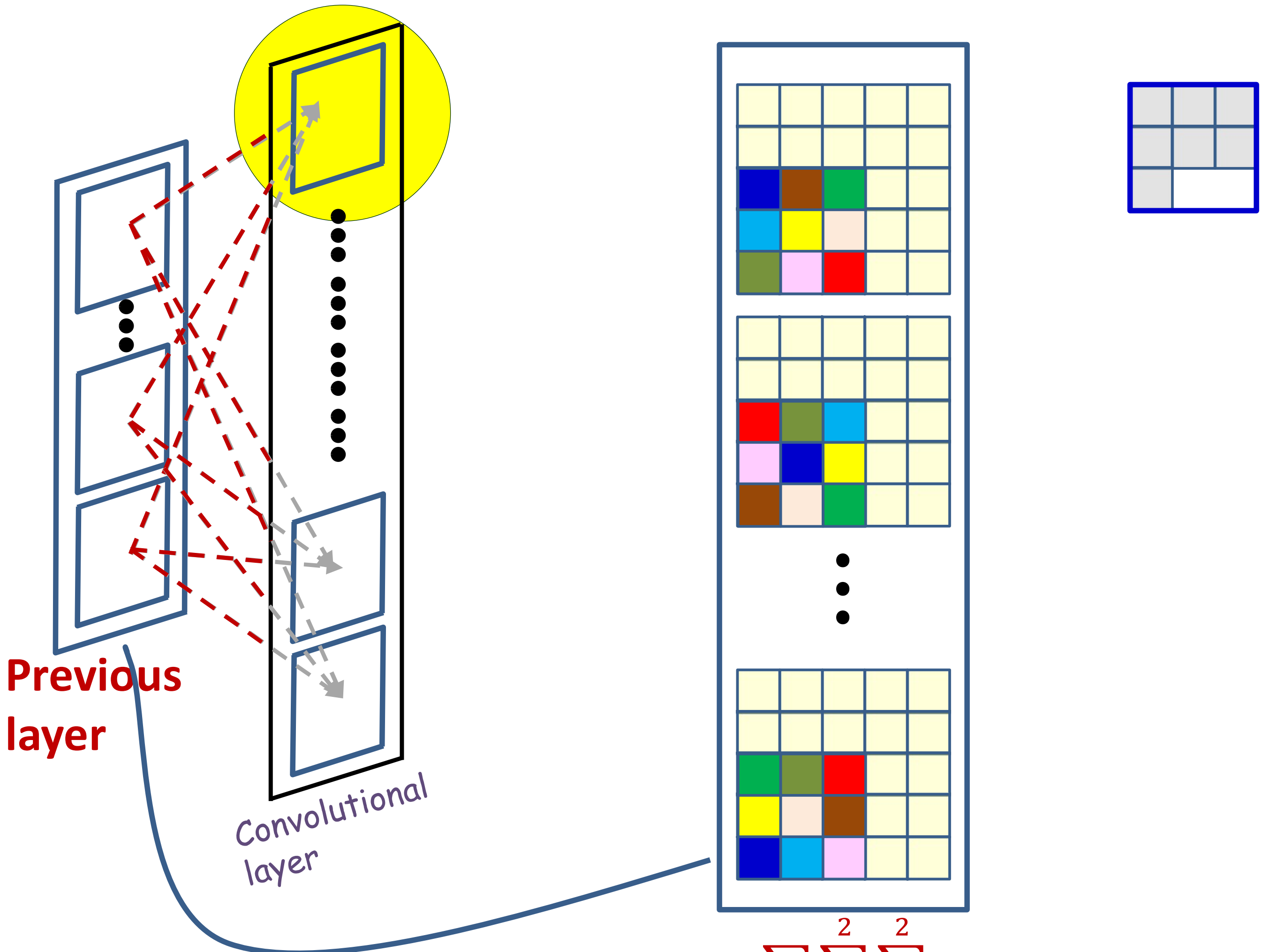
Recap: Convolution



$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

Recap: Convolution

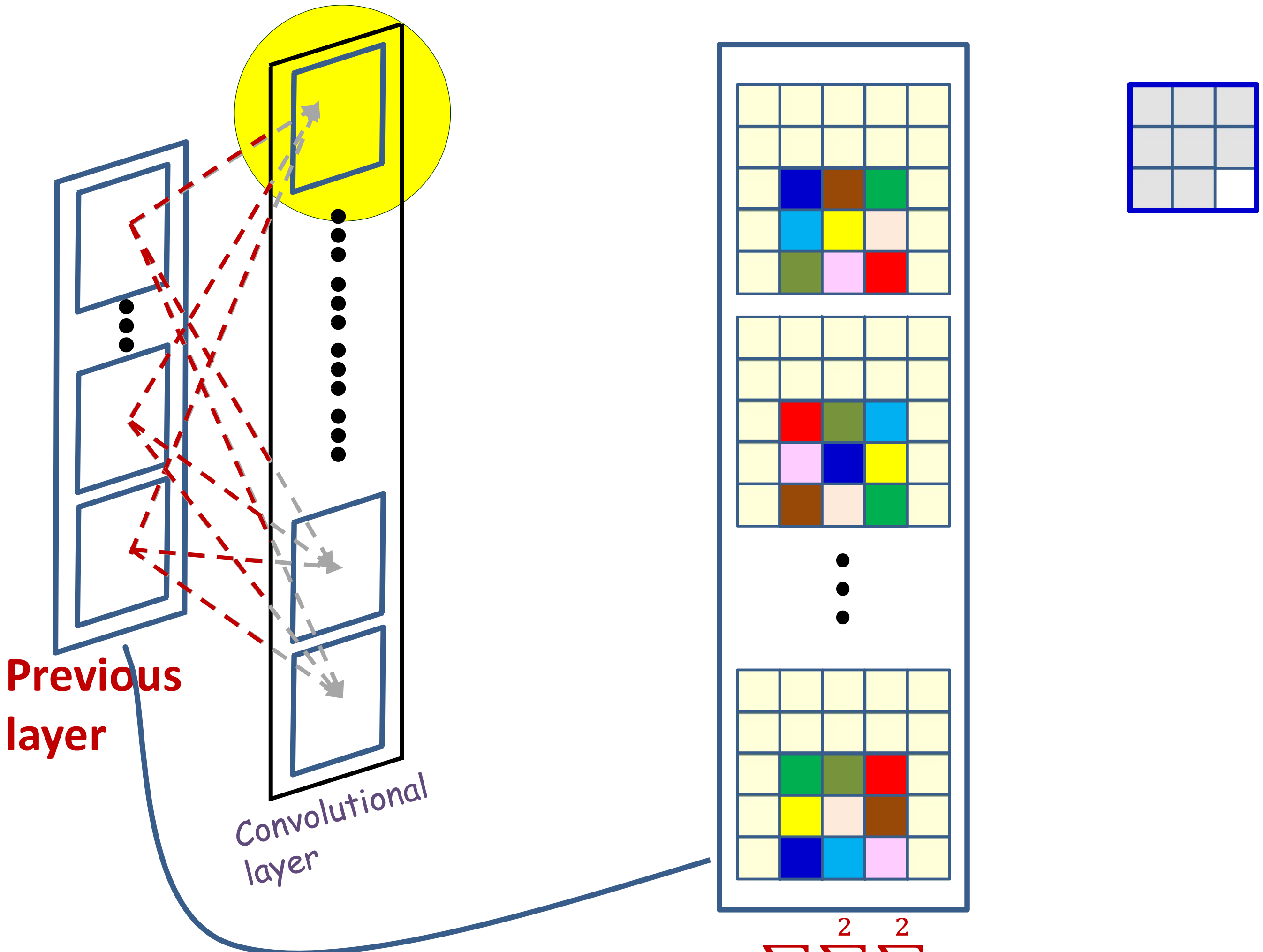


$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

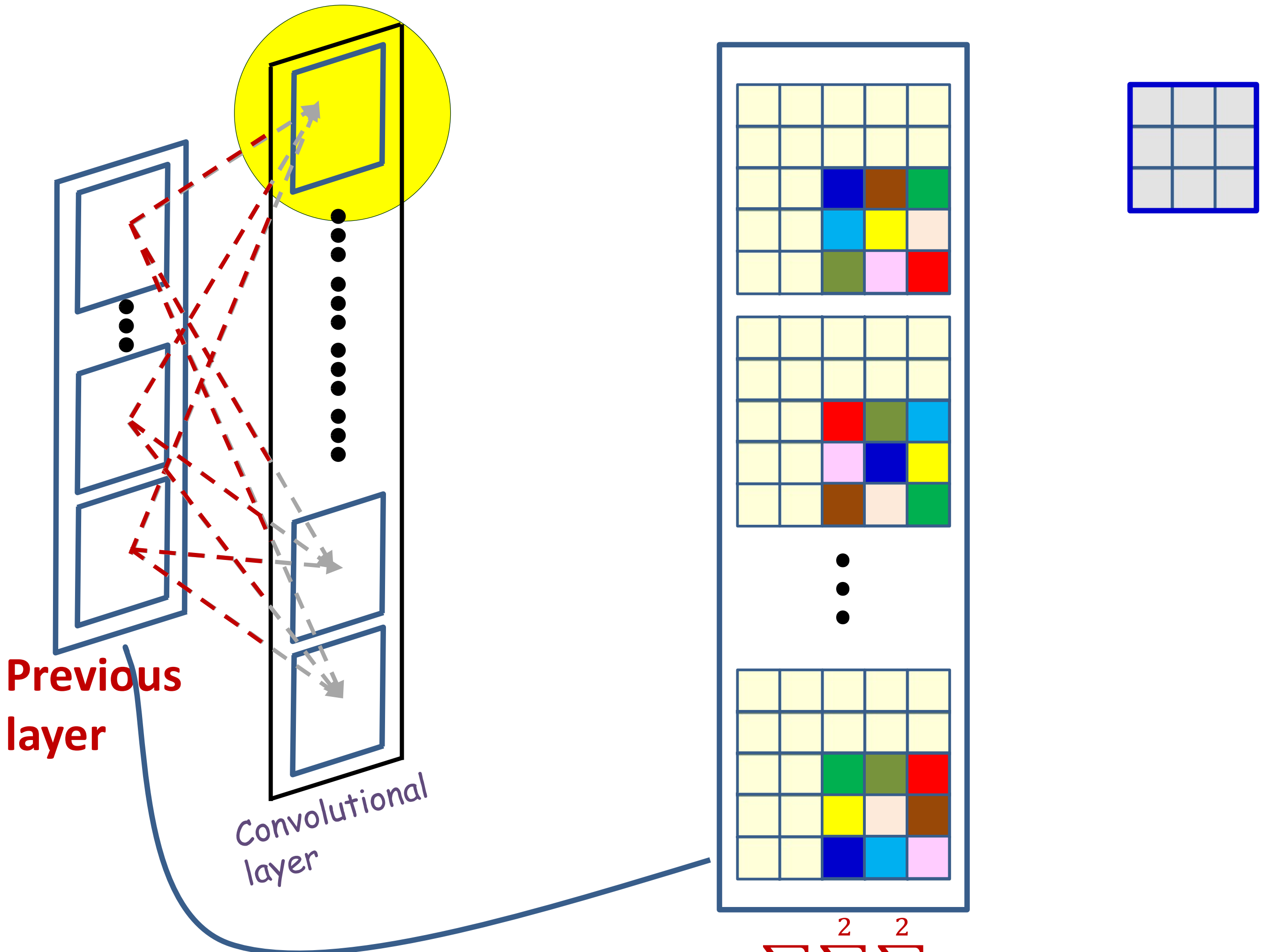
Recap: Convolution



$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

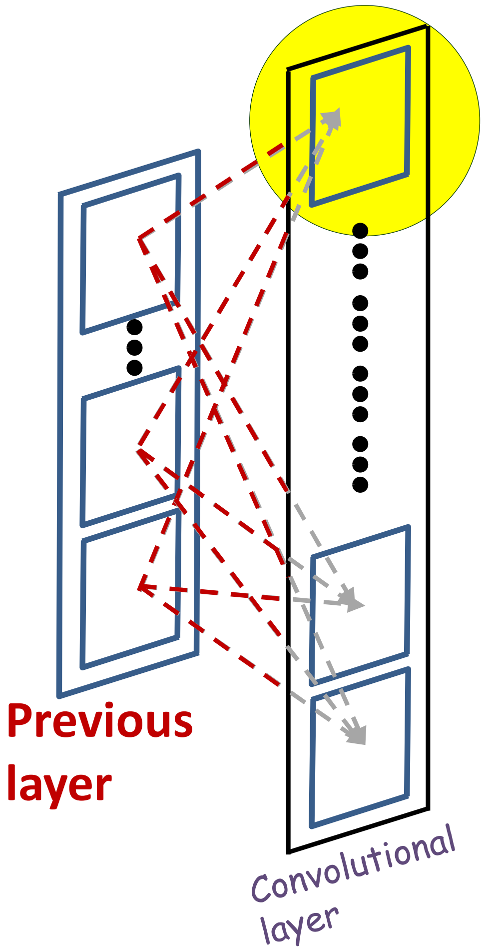
Recap: Convolution



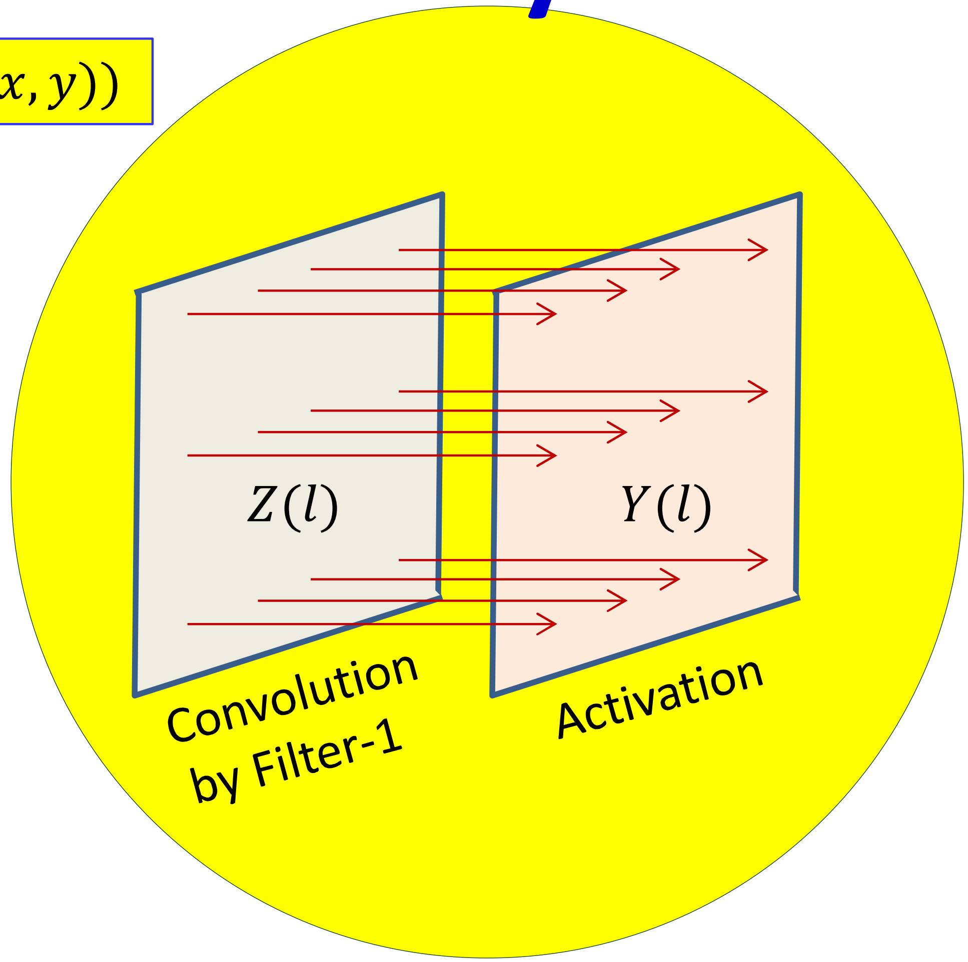
$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)\,y(l-1,m,x+i,y+j) + b_l(n)$$

- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

Recap: A convolutional layer

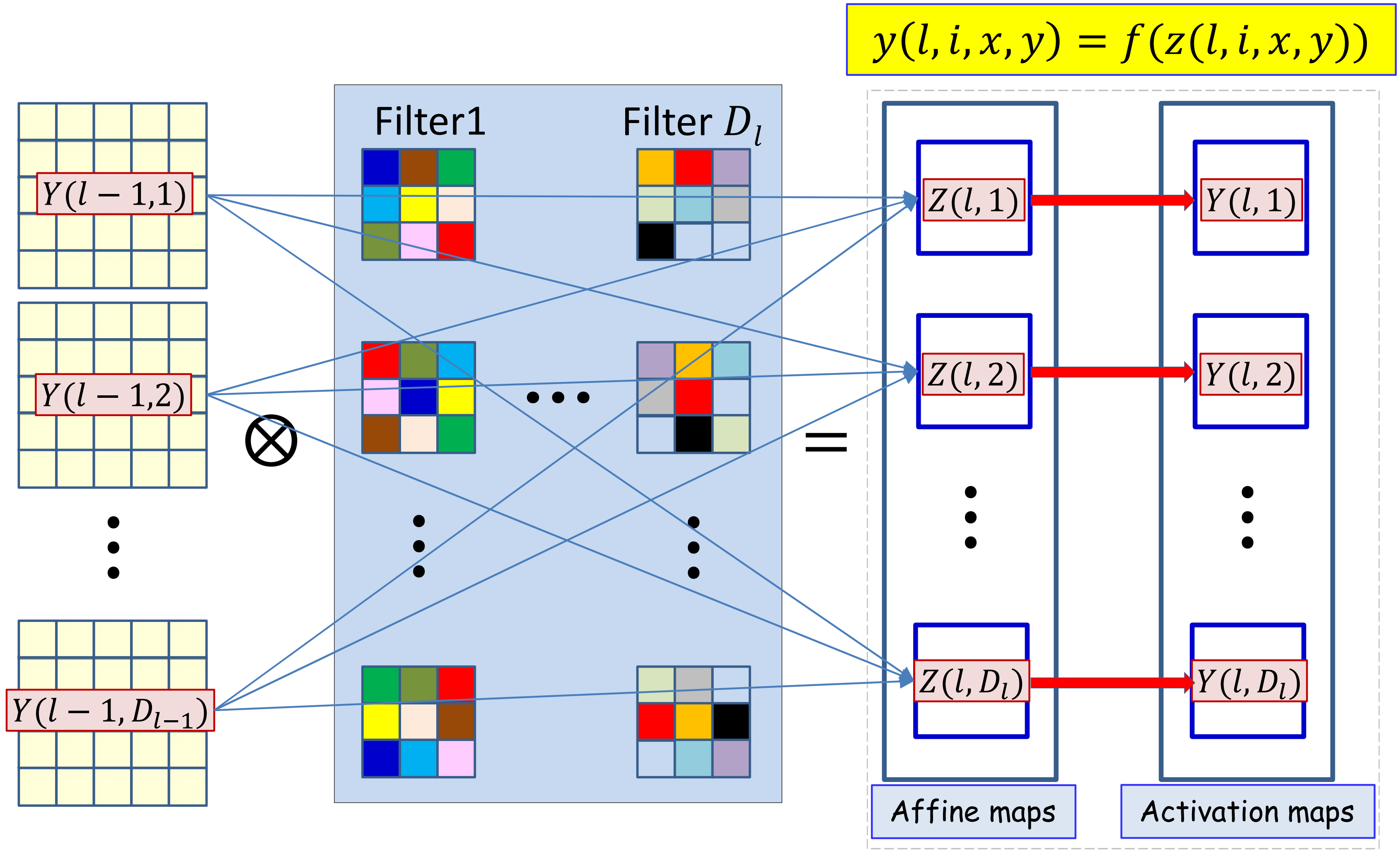


$$y(l,i,x,y) = f(z(l,i,x,y))$$



- The computation of each output map has two stages
  - Computing an *affine* map, by *convolution* of a *filter* (representing a pattern of weights) over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates on the output of the convolution

Convolution layer: A more explicit illustration



- Input maps $Y(l-1,*)$ are convolved with several filters to generate the affine maps $Z(l,*)$
  - Each filter consists of a set of square patterns of weights, with one set for each map in $Y(l-1,*)$
  - We get one affine map per filter
- A *point-wise* activation function $f(z)$ is applied to each map in $Z(l,*)$ to produce the activation maps $Y(l,*)$

Pseudocode: Vector notation

The weight W(l,j)**is a 3D $D_{l-1} \times K_l \times K_l$ tensor**

```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for x = 1:W_{l-1}-K_l+1
    for y = 1:H_{l-1}-K_l+1
      for j = 1:D_l
        segment = Y(l-1,:,x:x+K_l-1,y:y+K_l-1) #3D tensor
        z(l,j,x,y) = W(l,j).segment + b(l,j)#tensor prod.
        Y(l,j,x,y) = activation(z(l,j,x,y))
Y = softmax( {Y(L,:,:,:)} )
```

Pseudocode has 1-based indexing

Poll 1

Select all true statements about a convolution layer.

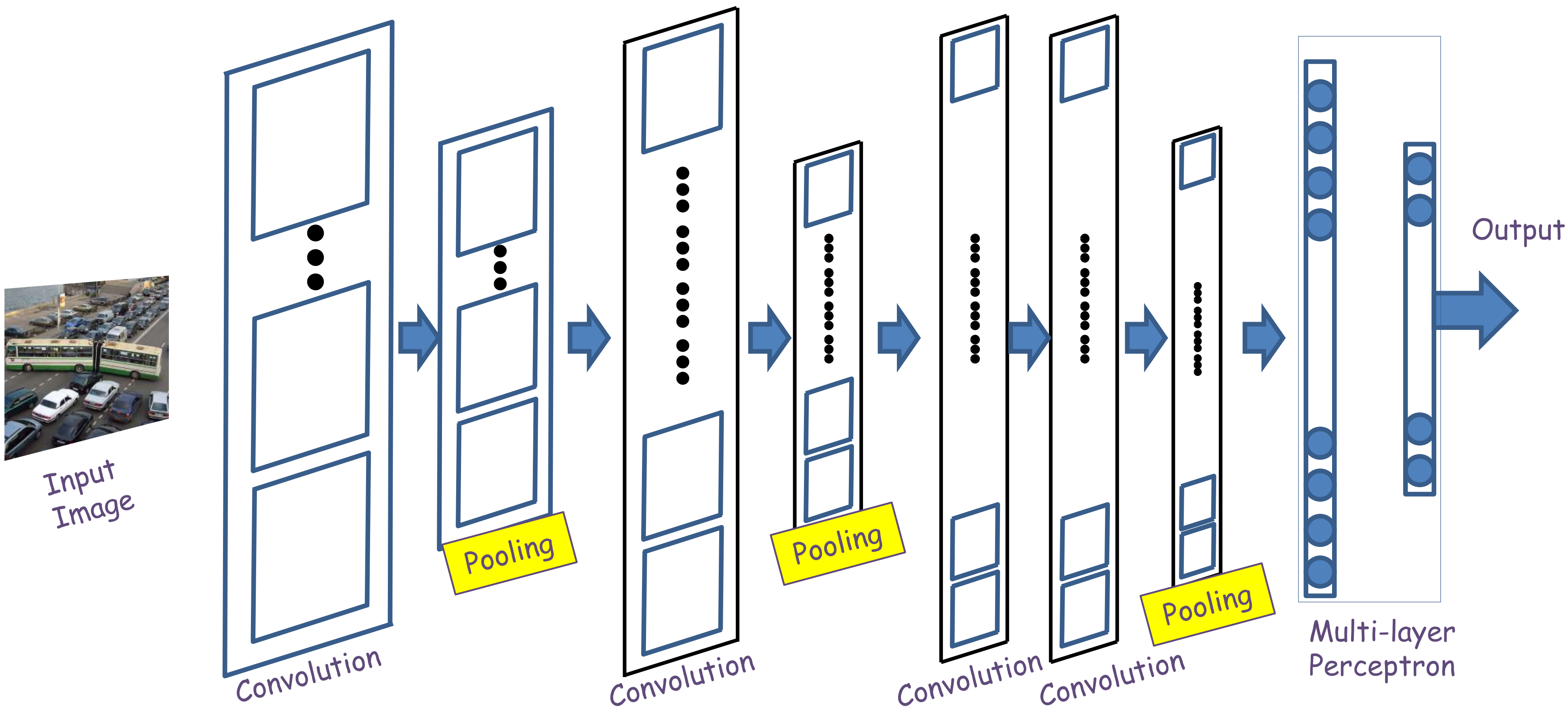
- The number of “channels” in any filter equals the number of input maps (output maps from the previous layer)
- The number of “channels” in any filter equals the number of output maps (affine maps output by the layer)
- The number of filters equals the number of input maps
- The number of filters equals the number of output maps

Poll 1

Select all true statements about a convolution layer.

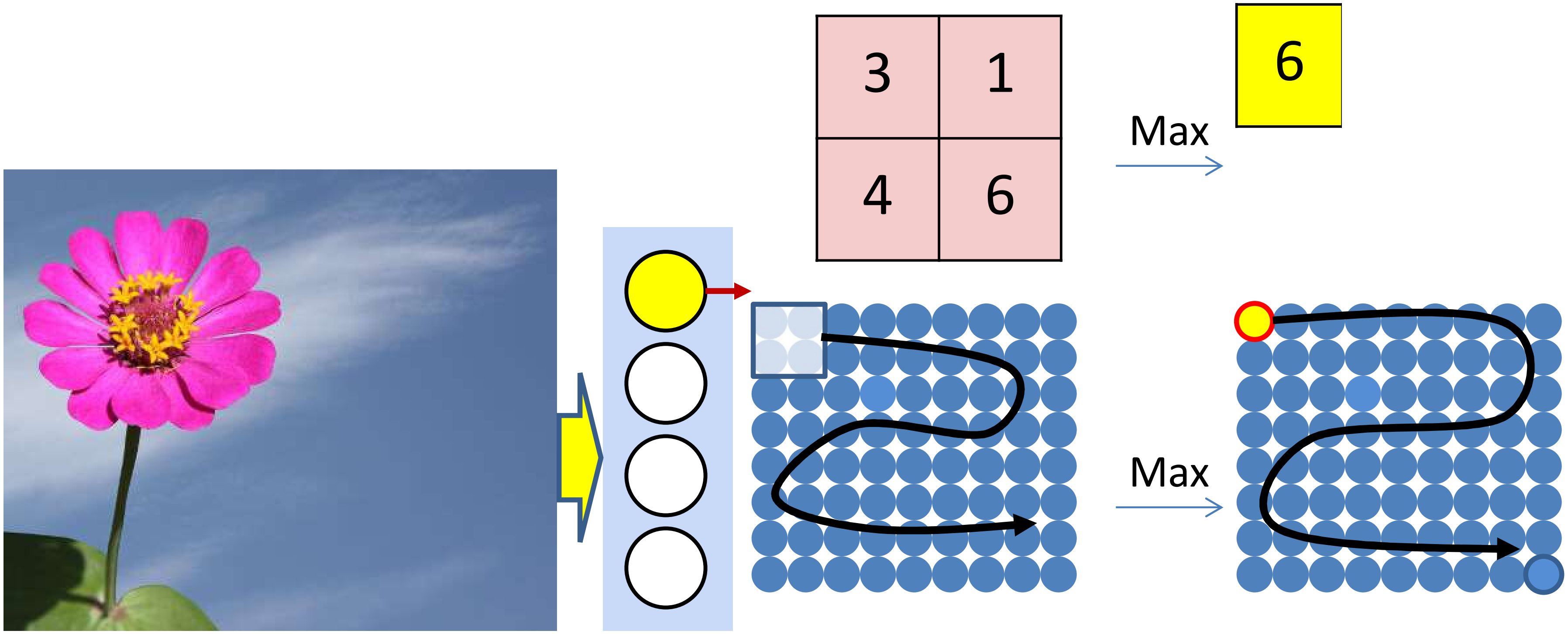
- **The number of "channels" in any filter equals the number of input maps (output maps from the previous layer)**
- The number of "channels" in any filter equals the number of output maps (affine maps output by the layer)
- The number of filters equals the number of input maps
- **The number of filters equals the number of output maps**

Pooling



- Convolutional (and activation) layers are followed intermittently by "pooling" layers
  - Often, they alternate with convolution, though this is not necessary

Recall: Max pooling



- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input with a "max-pooling filter"

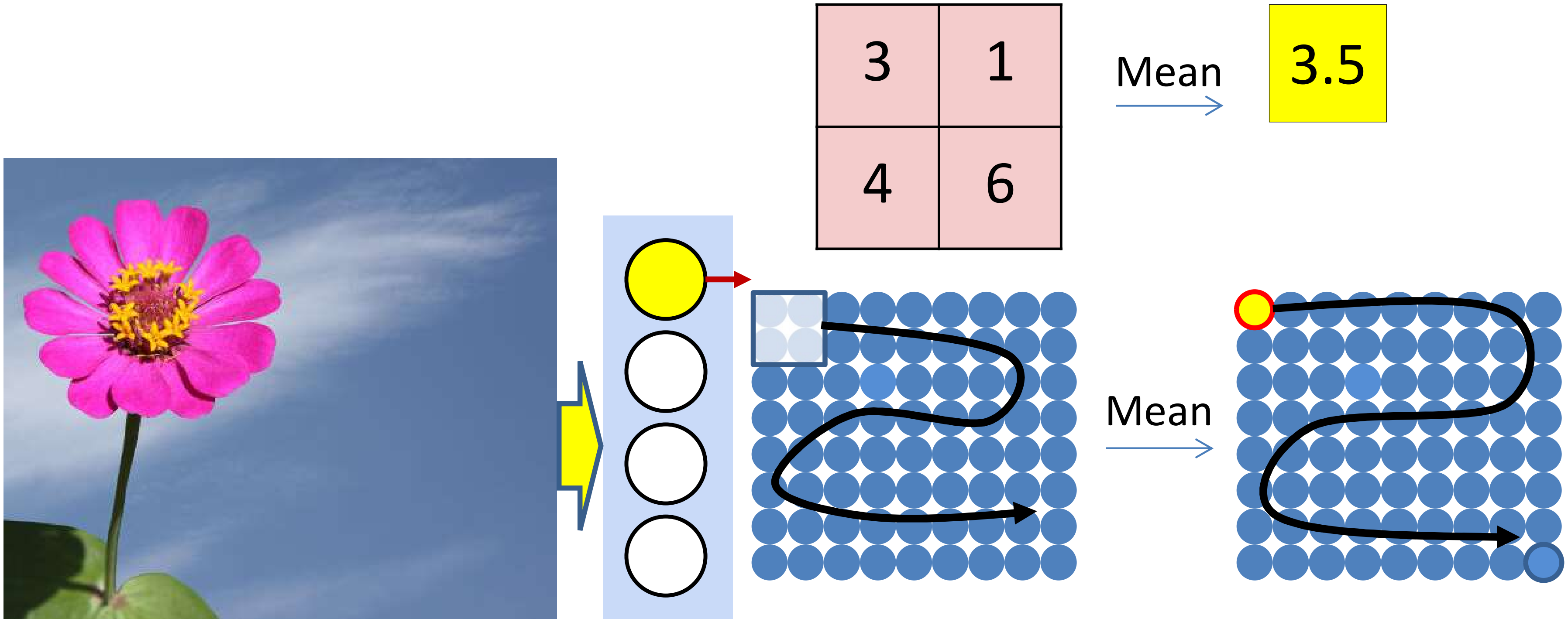
Recap: Max Pooling layer at layer l

a) Performed separately for every map (j).
   *) *Not* combining multiple maps within a single max operation.
b) Keeping track of location of max

Max pooling

```
for j = 1:D_l
  for x = 1:W_{l-1}-K_l+1
    for y = 1:H_{l-1}-K_l+1
      pidx(l,j,x,y) = maxidx(Y(l-1,j,x:x+K_l-1,y:y+K_l-1))
      u(l,j,x,y) = Y(l-1,j,pidx(l,j,m,n))
```

Recall: Mean pooling



- Mean pooling computes the *mean* of the window of values
  - As opposed to the max of max pooling

Recap: Mean Pooling layer at layer *l*

a) Performed separately for every map (j)

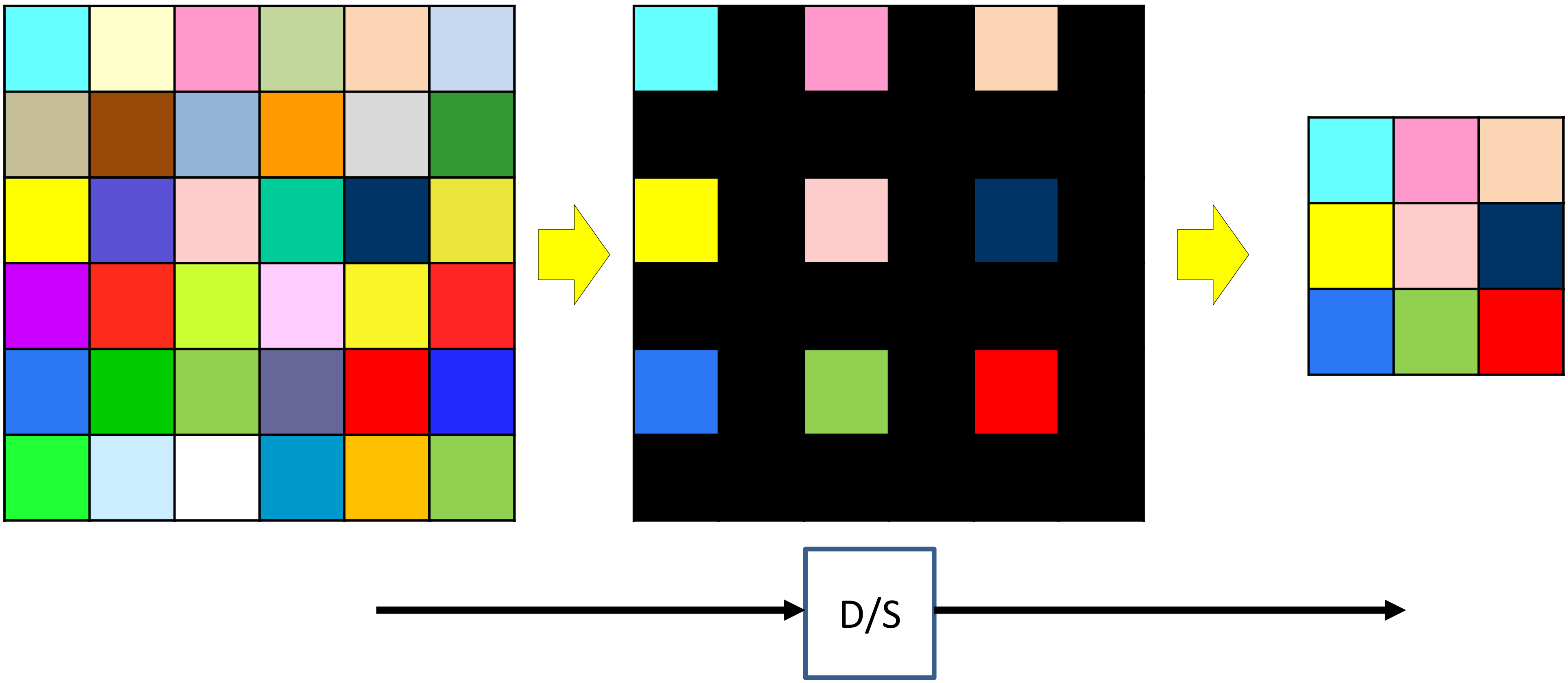
Mean pooling

```
for j = 1:D_l
  for x = 1:W_l-1-K_l+1
    for y = 1:H_l-1-K_l+1
      u(l,j,x,y) = mean(Y(l-1,j,x:x+K_l-1,y:y+K_l-1))
```

Recap: Resampling

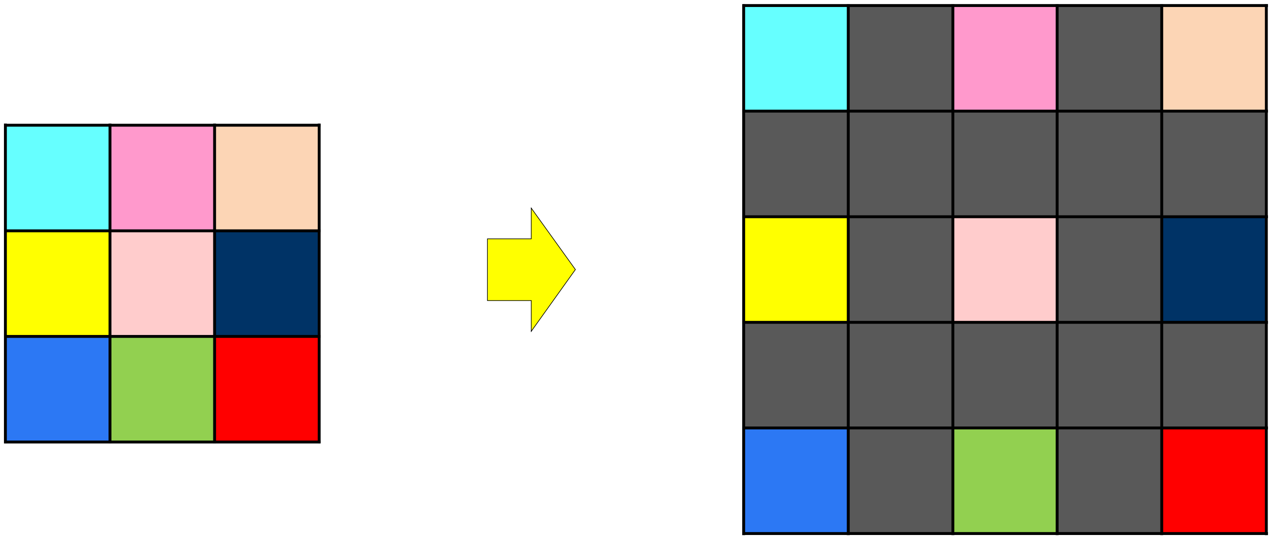
- We can also proportionately decrease or increase the size of the maps by dropping or inserting zeros
  - Downsampling: Drop S-1 rows/columns between rows/columns
    - Reduces the size of the maps by S on each side
  - Upsampling: Insert S-1 rows/columns of zeros between adjacent entries
    - Increases the size of the map by S on each side

The Downsampling Layer



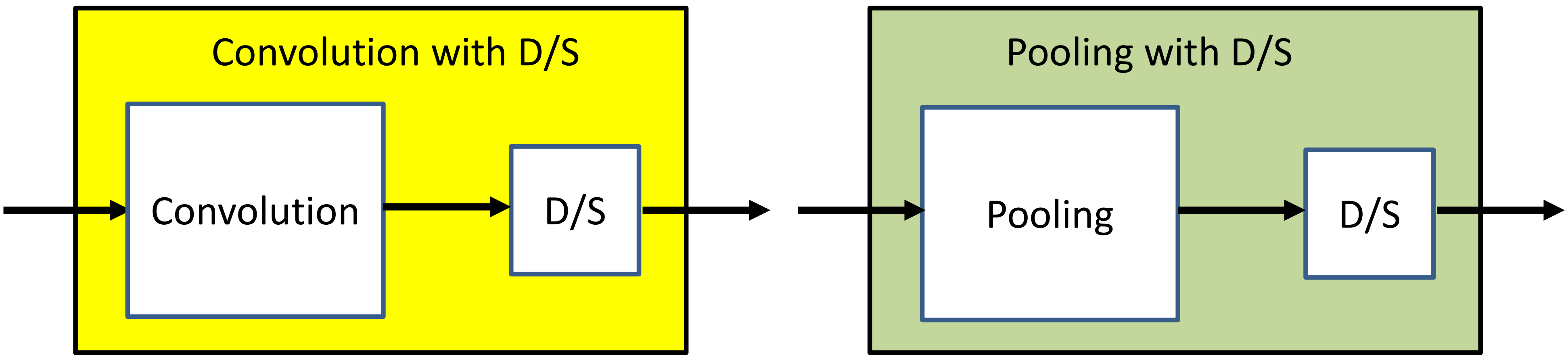
- A *downsampling* layer simply "drops" $S-1$ of S rows and columns for every map in the layer
  - Effectively reducing the size of the map by factor S in every direction

The Upsampling Layer



- A *upsampling* (or dilation) layer simply introduces $S - 1$ rows and columns for every map in the layer
  - Effectively *increasing* the size of the map by factor S in every direction
- Used explicitly to increase the map size by a uniform factor

Downsampling in practice



- In practice, the downsampling is combined with the layers just before it by performing the operations with a stride > 1
  - Could be convolutional or pooling layers

Convolution with downsampling

The weight W(l,j) is now a 4D $D_l \times D_{l-1} \times K_l \times K_l$ tensor

The product in blue is a tensor inner product with a scalar output

```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
   m = 1
   for x = 1:S:W_{l-1}-K_l+1
      n = 1
      for y = 1:S:H_{l-1}-K_l+1
         segment = Y(l-1,:,x:x+K_l-1,y:y+K_l-1) #3D tensor
         z(l,:,m,n) = W(l).segment #tensor inner prod.
         Y(l,:,m,n) = activation(z(l,:,m,n))
         n++
     m++
Y = softmax( {Y(L,:,:,:)} )
```

STRIDE

Downsampled indices

Max Pooling with Downsampling

Max pooling

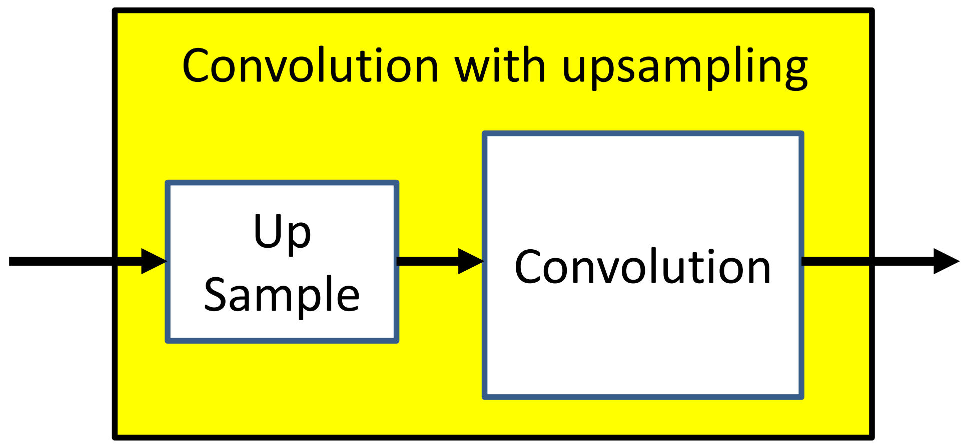
```
for j = 1:D_l
    m = 1
    for x = 1:stride(l):W_{l-1}-K_l+1
      n = 1
      for y = 1:stride(l):H_{l-1}-K_l+1
        pidx(l,j,m,n) = maxidx(Y(l-1,j,x:x+K_l-1,y:y+K_l-1))
        Y(l,j,m,n) = Y(l-1,j,pidx(l,j,m,n))
        n = n+1
      m = m+1
```

Mean Pooling with Downsampling

Mean pooling

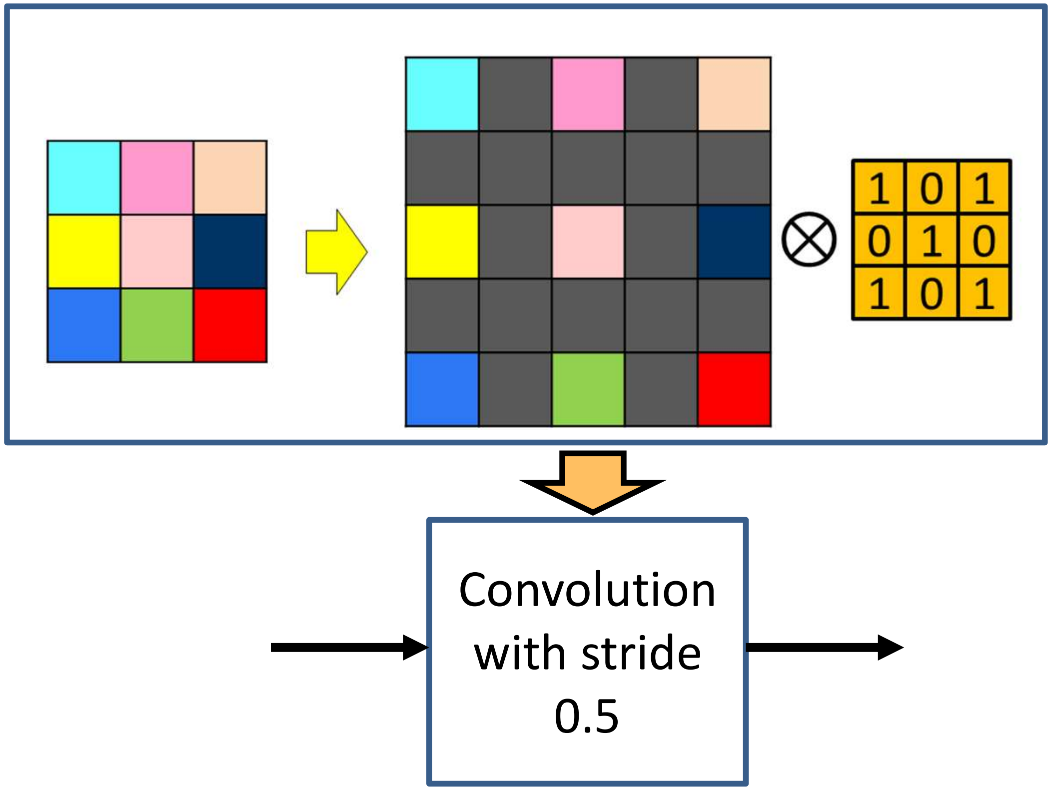
```
for j = 1:D_l
   m = 1
   for x = 1:stride(l):W_{l-1}-K_l+1
     n = 1
     for y = 1:stride(l):H_{l-1}-K_l+1
       Y(l,j,m,n) = mean(Y(l-1,j,x:x+K_l-1,y:y+K_l-1))
       n = n+1
     m = m+1
```

The Upsampling Layer



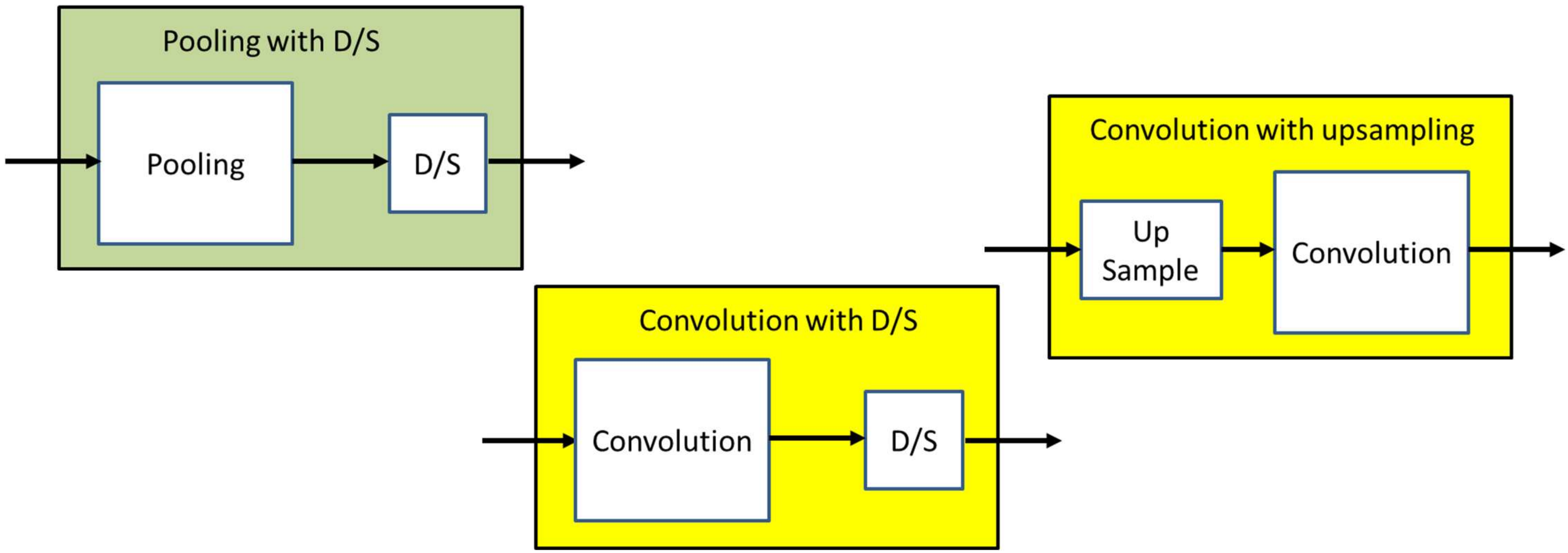
- A *upsampling* layer is generally followed by a CNN layer
  - It is not useful to follow it by a pooling layer
  - It is also not useful as the *final* layer of a CNN

The Upsampling Layer



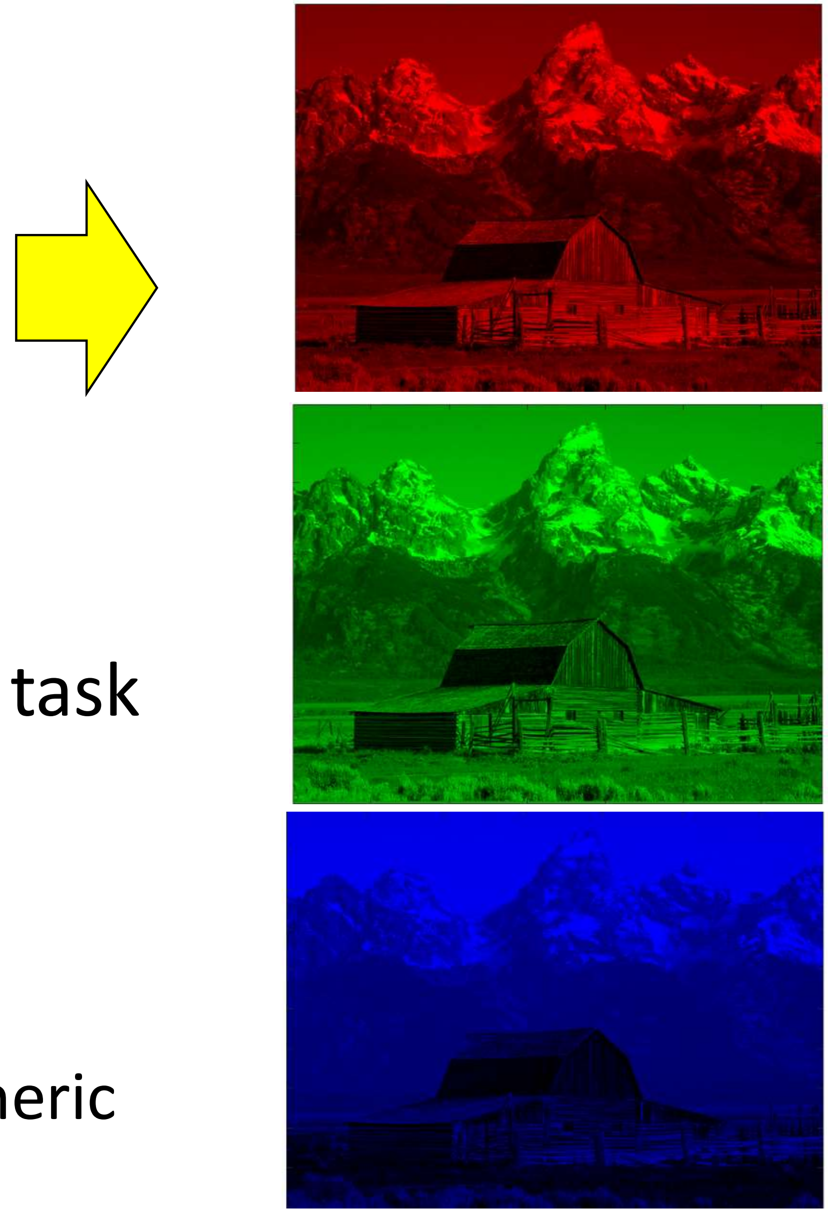
- Upsampling layers followed by a convolutional layer are also often viewed as convolving with a fractional stride
  - Upsampling by factor S is the same as striding by factor $1/S$
- Also called "transpose convolutions" for reasons we won't get into here

* with resampling



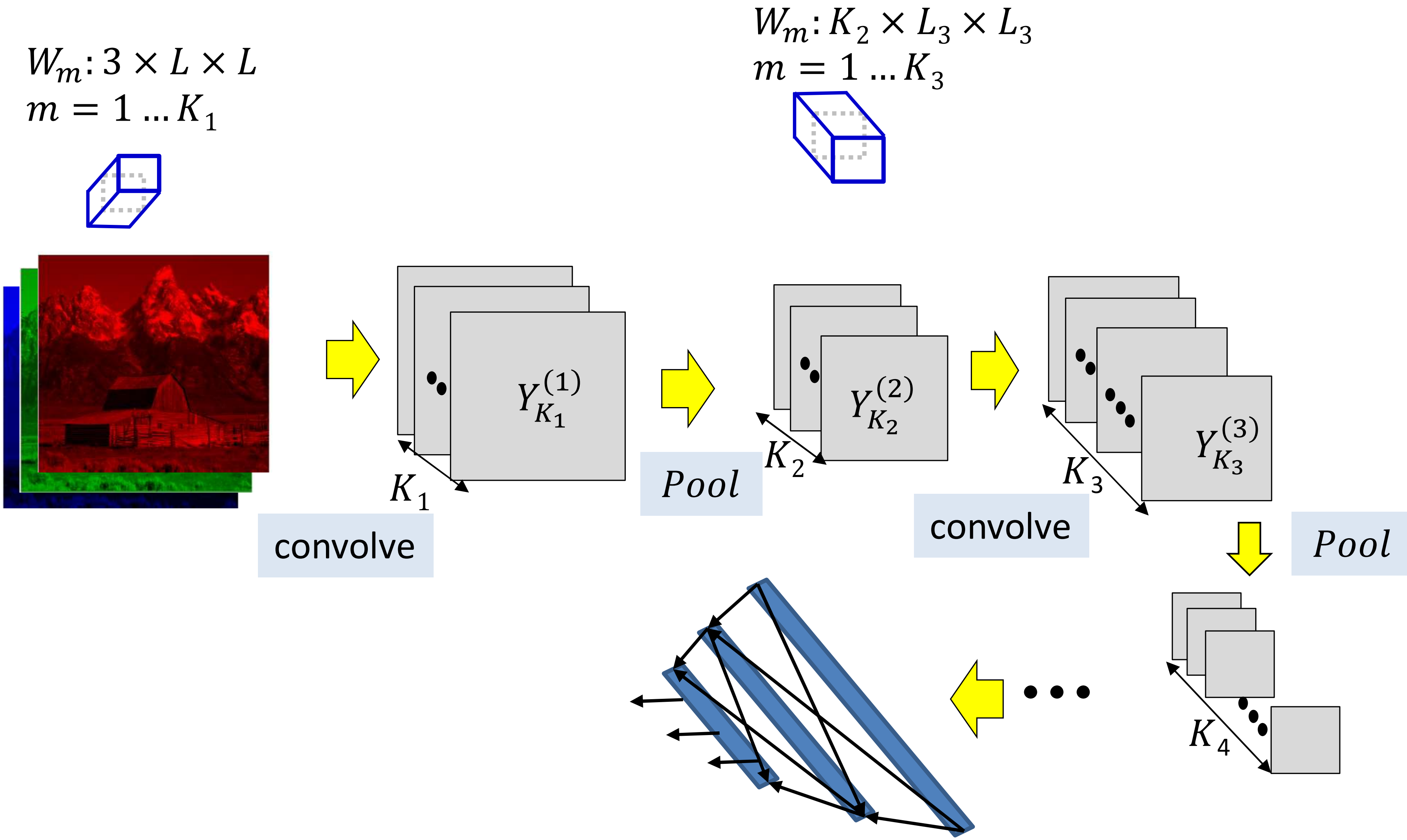
- Although the resampling operation is generally merged with convolutions or pooling (by changing their stride) in the forward pass in practical implementations…
- …It is more convenient to think of the two as separate operations in the backward pass
  – More on this later…

Recap: A CNN, end-to-end



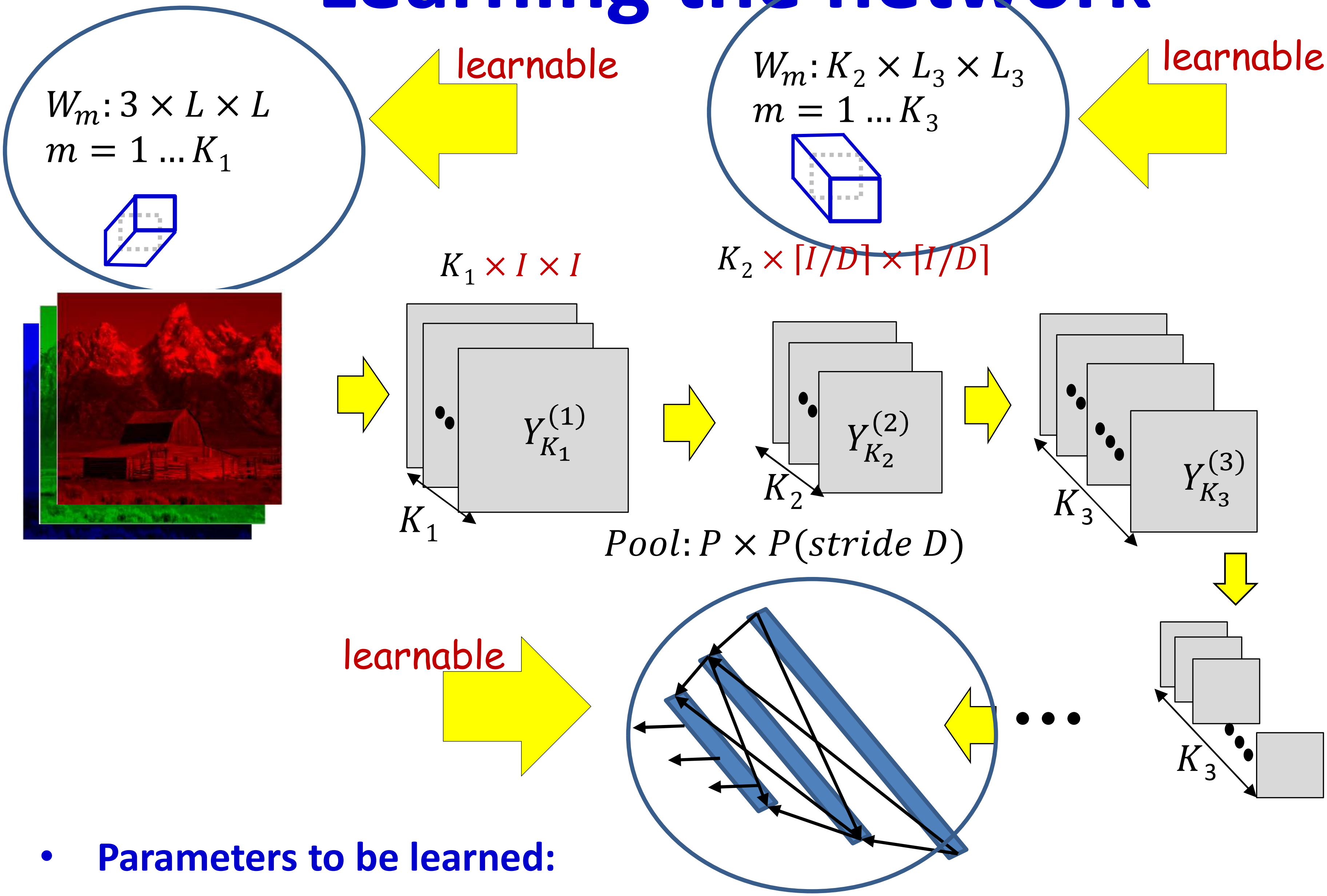
- Typical image classification task
  - Assuming maxpooling..
- Input: RBG images
  - Will assume color to be generic

Recap: A CNN, end-to-end



- Several convolutional and pooling layers.
- The output of the last layer is "flattened" and passed through an MLP

Learning the network



- **Parameters to be learned:**
  - **The weights of the neurons in the final MLP**
  - **The (weights and biases of the) filters for every *convolutional* layer**

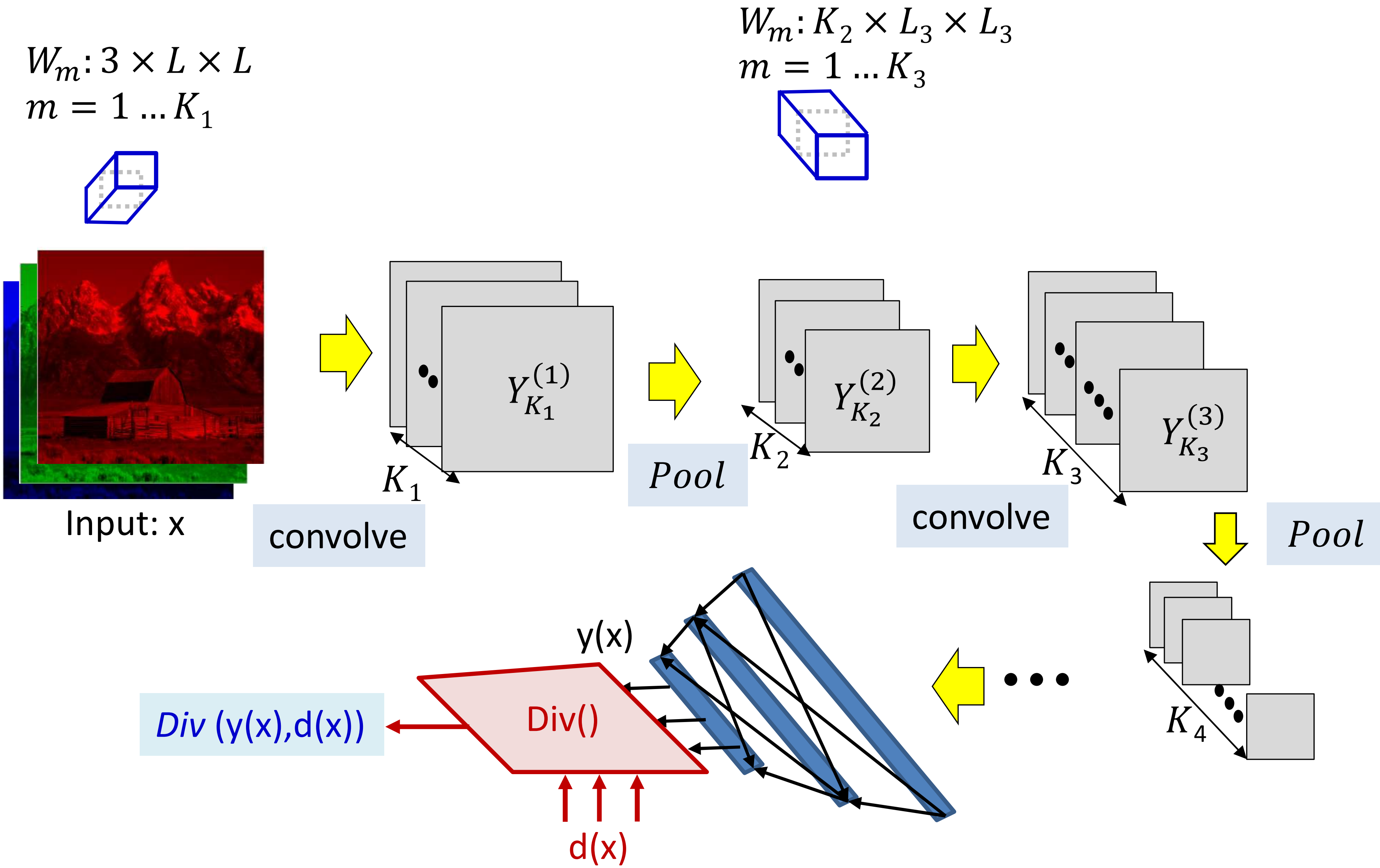
Recap: Learning the CNN

- Training is as in the case of the regular MLP
  - The *only* difference is in the *structure* of the network
- **Training examples of (Image, class) are provided**

- **Define a loss:**
  - Define a divergence between the desired output and true output of the network in response to any input
  - The loss aggregates the divergences of the training set

- **Network parameters are trained to minimize the loss**
  - Through variants of gradient descent
  - Gradients are computed through backpropagation

Defining the loss



- The loss for a single instance

Problem Setup

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- The divergence on the i^th instance is $div(Y_i, d_i)$
- The aggregate Loss

$$Loss = \frac{1}{T}\sum_{i=1}^{T} div(Y_i, d_i)$$

- Minimize $Loss$ w.r.t $\{W_m, b_m\}$
  – Using gradient descent

The derivative

Total training loss:

$$Loss = \frac{1}{T}\sum_i Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

$$\frac{dLoss}{dw} = \frac{1}{T}\sum_i \frac{dDiv(Y_i, d_i)}{dw}$$

The derivative

Total training loss:

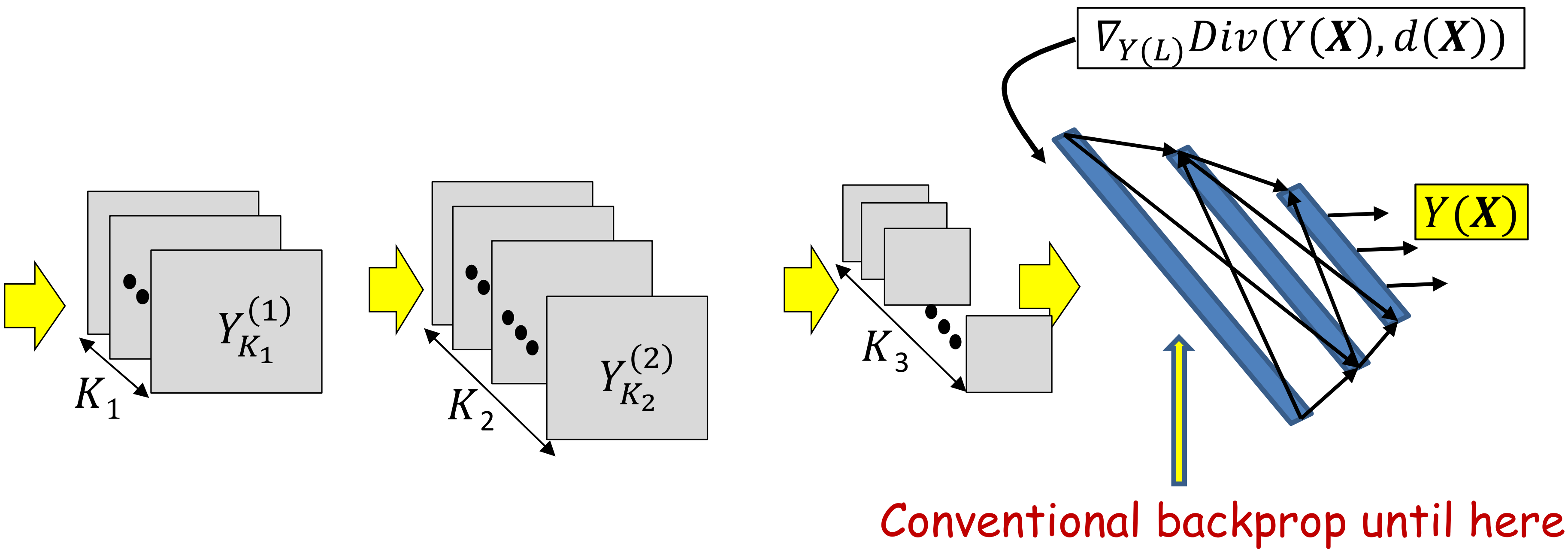
$$Loss = \frac{1}{T}\sum_i Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

$$\frac{dLoss}{dw} = \frac{1}{T}\sum_i \frac{dDiv(Y_i, d_i)}{dw}$$

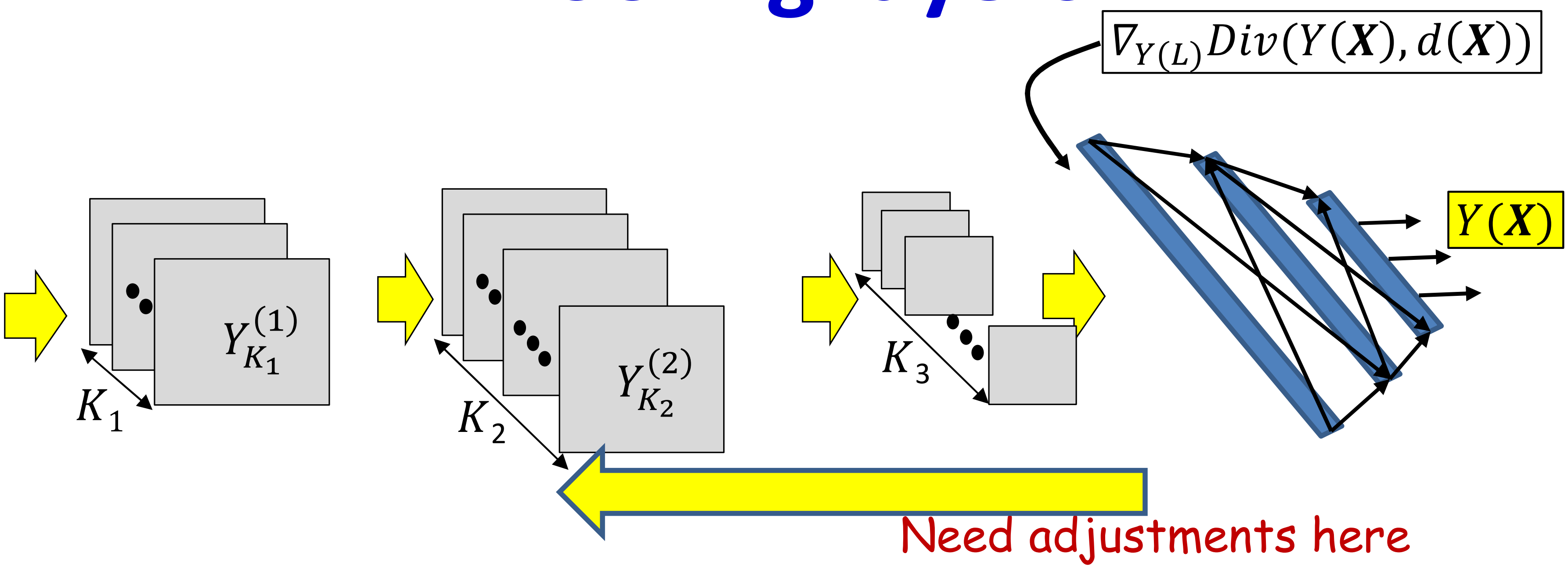
Backpropagation: Final flat layers



- For each training instance:
  - First, perform a forward pass through the net
  - Then perform the backpropagation of the derivative of the divergence

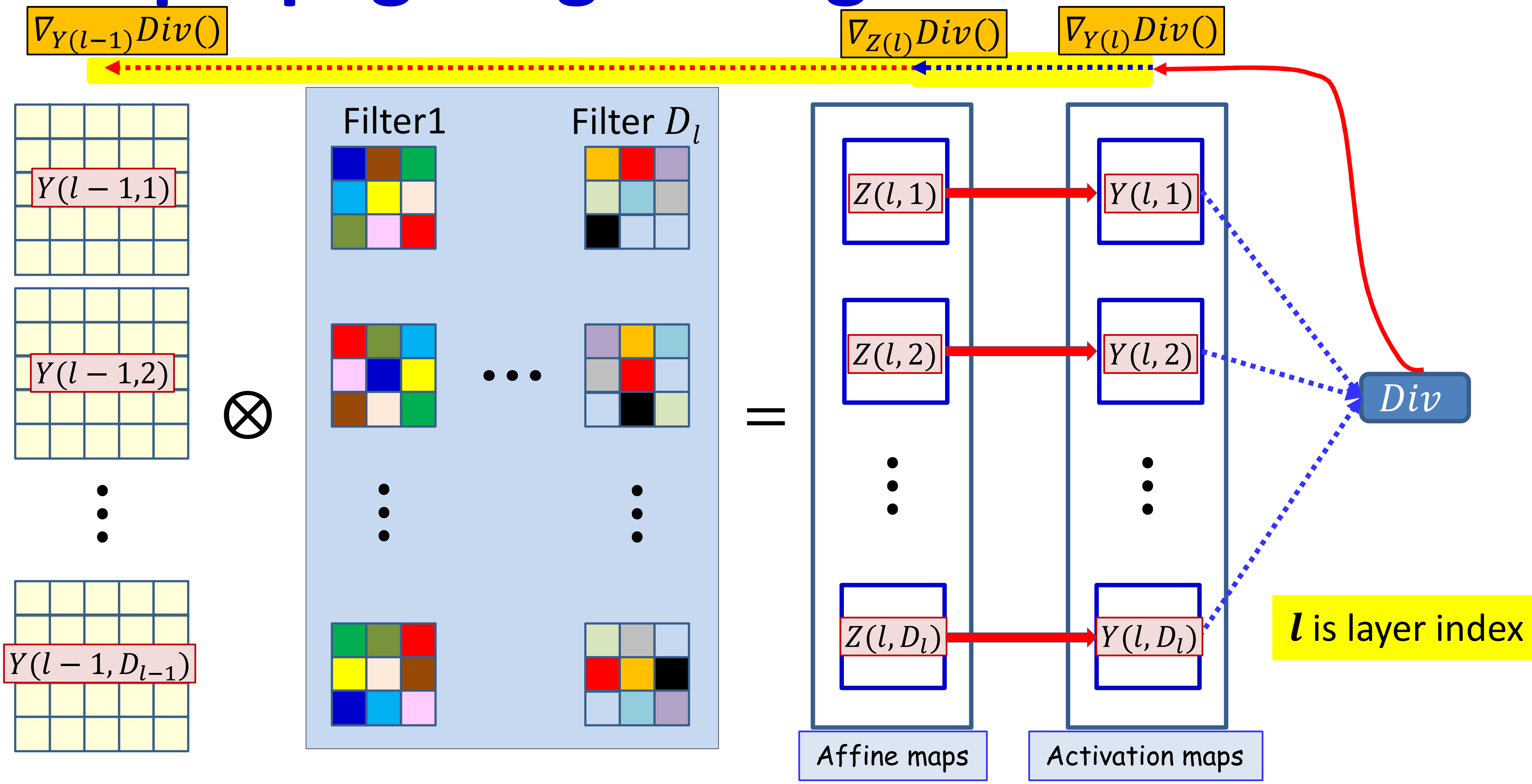
- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first "flat" layer
  - Important to recall: the first flat layer is only the "unrolling" of the maps from the final convolutional layer

Backpropagation: Convolutional and Pooling layers



- Backpropagation from the flat MLP requires special consideration of
  - The shared computation in the convolution layers
  - The pooling layers

Backpropagating through the convolution



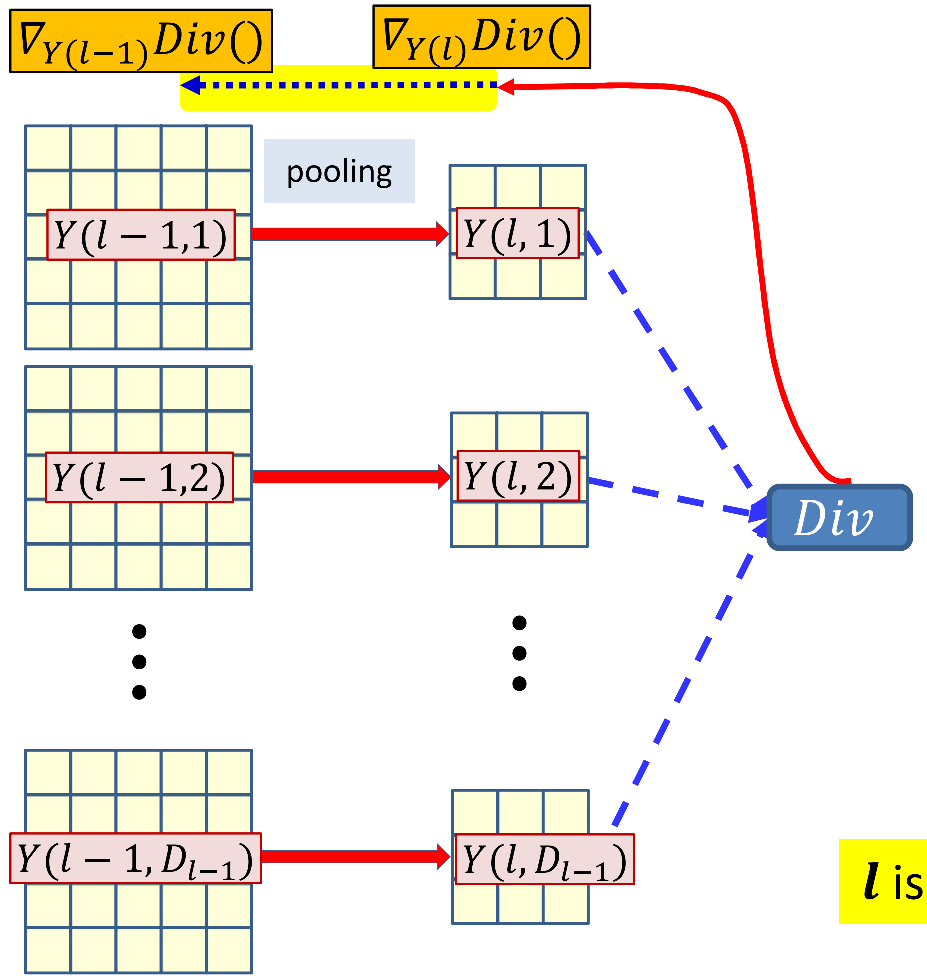
- **Convolution layers:**
- We already have the derivative w.r.t (all the elements of) activation map $Y(l,*)$
  - Having backpropagated it from the divergence
- We must backpropagate it through the activation to compute the derivative w.r.t. $Z(l,*)$ and further back to compute the derivative w.r.t the filters and $Y(l-1,*)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

- **Required:**
  - **For convolutional layers**:
    - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$

Backprop: Pooling layer



l is layer index

- **Pooling layers:**
- We already have the derivative w.r.t $Y(l,*)$
  - Having backpropagated it from the divergence
- We must compute the derivative w.r.t $Y(l-1,*)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

- **Required:**
  - **For convolutional layers**:
    - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
  - **For pooling layers**:
    - How to compute the derivative w.r.t. $Y(l-1)$ given derivatives w.r.t. $Y(l)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

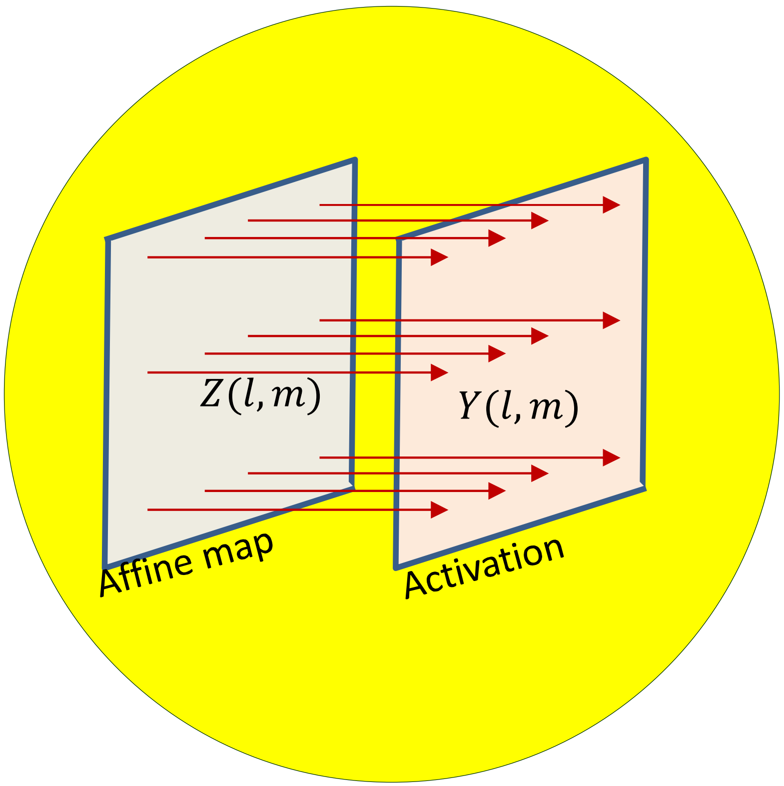
- **Required:**
  - **For convolutional layers**:
    - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
  - **For pooling layers**:
    - How to compute the derivative w.r.t. $Y(l-1)$ given derivatives w.r.t. $Y(l)$

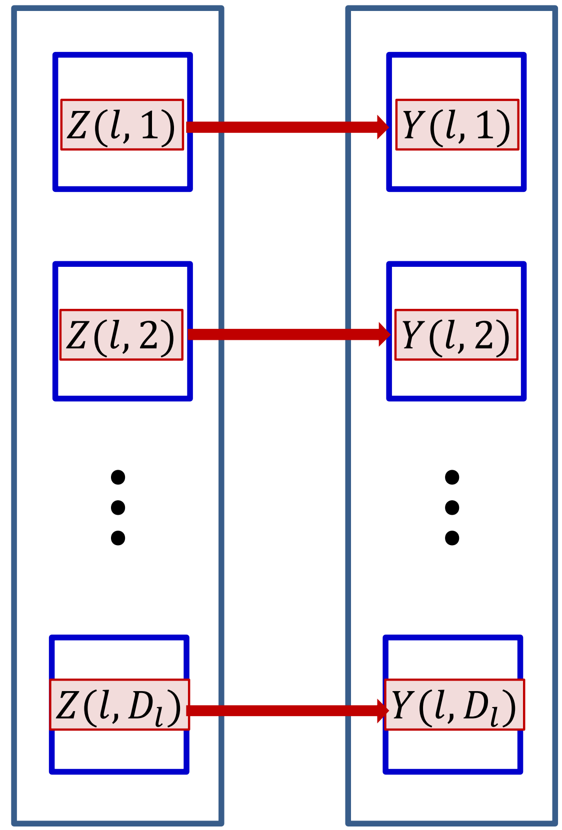
Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

- **Required:**
  - **For convolutional layers**:
    - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
  - **For pooling layers**:
    - How to compute the derivative w.r.t. $Y(l-1)$ given derivatives w.r.t. $Y(l)$

Backpropagating through the activation



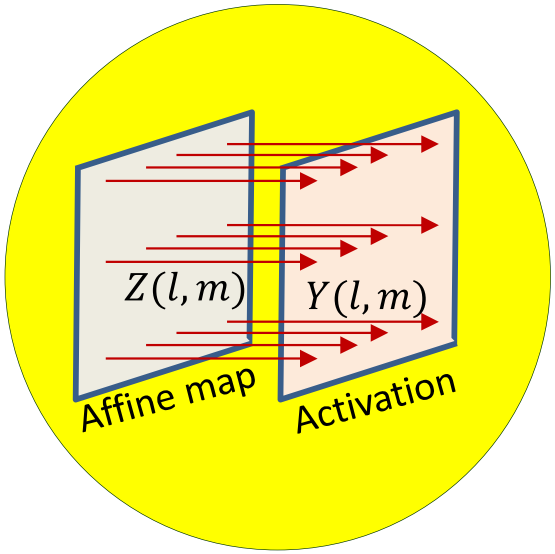


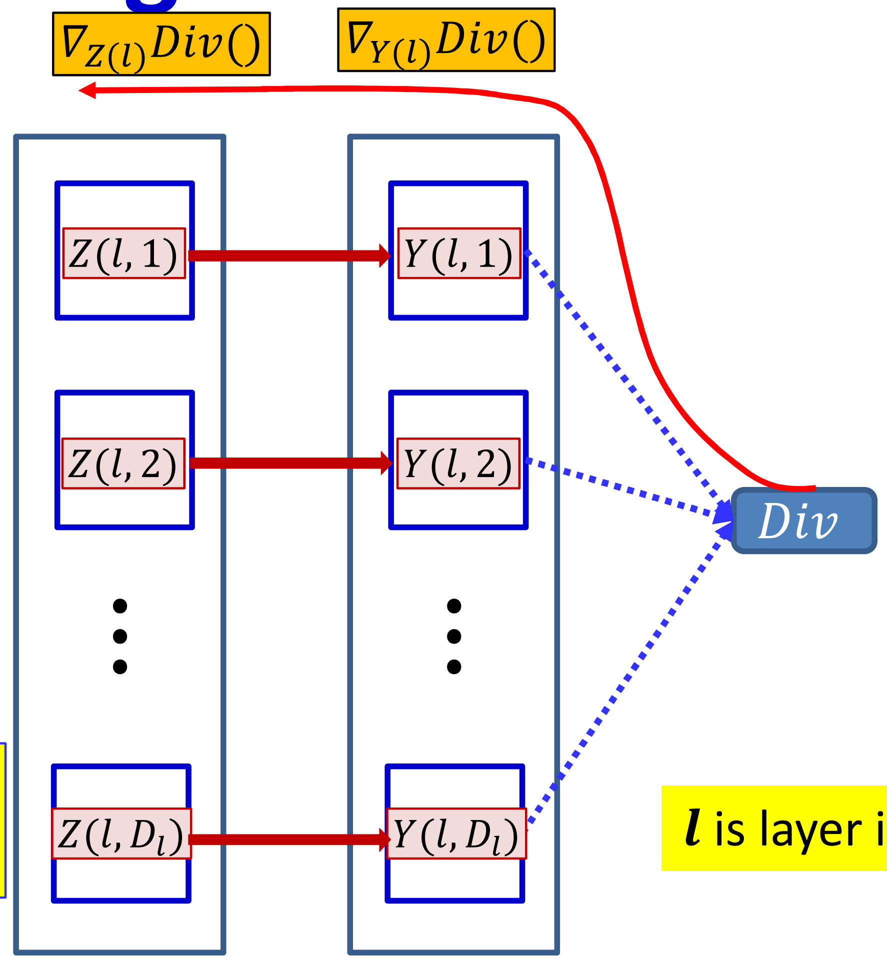
- **Forward computation:** The activation maps are obtained by point-wise application of the activation function to the affine maps

$$y(l,m,x,y) = f(z(l,m,x,y))$$

  – The affine map entries $z(l,m,x,y)$ have already been computed via convolutions over the previous layer

Backpropagating through the activation





$$y(l,m,x,y) = f(z(l,m,x,y))$$

$$\frac{dDiv}{dz(l,m,x,y)} = \frac{dDiv}{dy(l,m,x,y)} f'(z(l,m,x,y))$$

l is layer index

- **Backward computation:** For every map $Y(l,m)$ for every position (x,y), we already have the derivative of the divergence w.r.t. $y(l,m,x,y)$
  - Obtained via backpropagation
- We obtain the derivatives of the divergence w.r.t. $z(l,m,x,y)$ using the chain rule:

$$\frac{dDiv}{dz(l,m,x,y)} = \frac{dDiv}{d\,y(l,m,x,y)} f'(z(l,m,x,y))$$

  - Simple component-wise computation

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  – Obtained as a result of backpropagating through the flat MLP

- **Required:**
  – **For convolutional layers**:
    ✔ How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
  – **For pooling layers**:
    - How to compute the derivative w.r.t. $Y(l-1)$ given derivatives w.r.t. $Y(l)$

Backpropagating through affine map

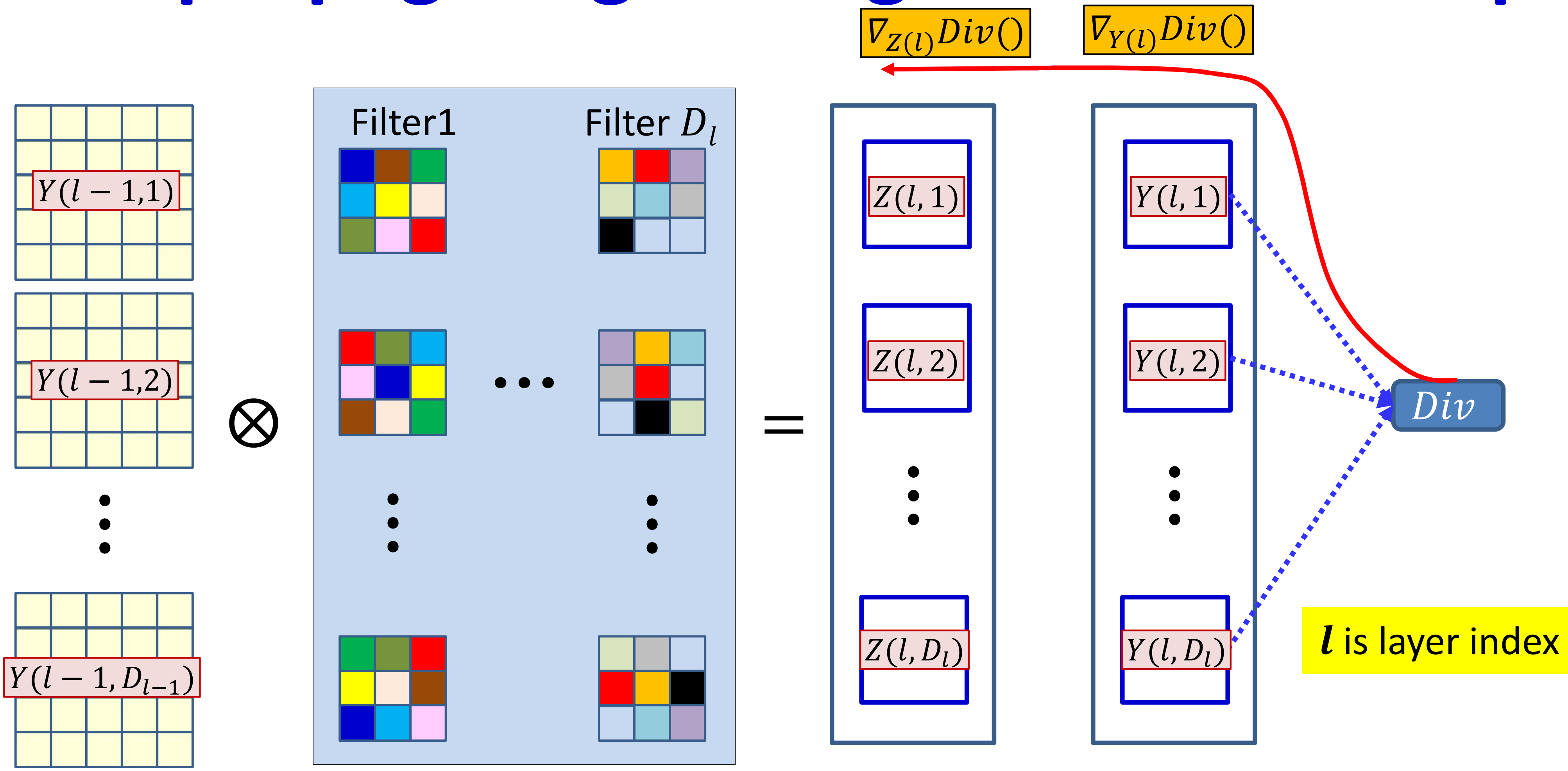
- Forward affine computation:
  - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l-1, m, x, y)$ and filters $w_l(m, n, x, y)$

- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
  - Compute derivative w.r.t. $y(l-1, m, x, y)$
  - Compute derivative w.r.t. $w_l(m, n, x, y)$

Backpropagating through affine map

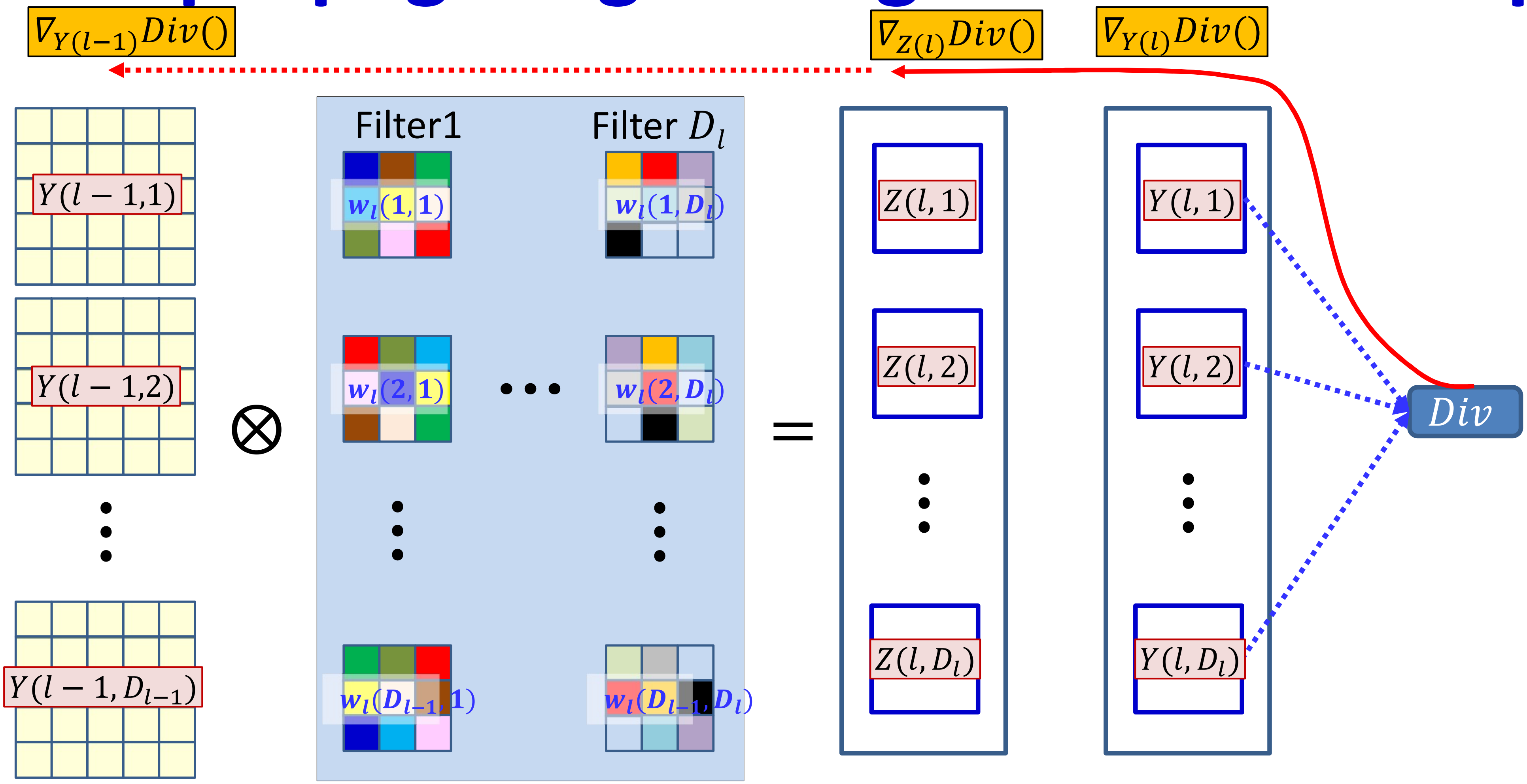
- Forward affine computation:
  - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l-1, m, x, y)$ and filters $w_l(m, n, x, y)$
- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
  - Compute derivative w.r.t. $y(l-1, m, x, y)$
  - Compute derivative w.r.t. $w_l(m, n, x, y)$

Backpropagating through the affine map



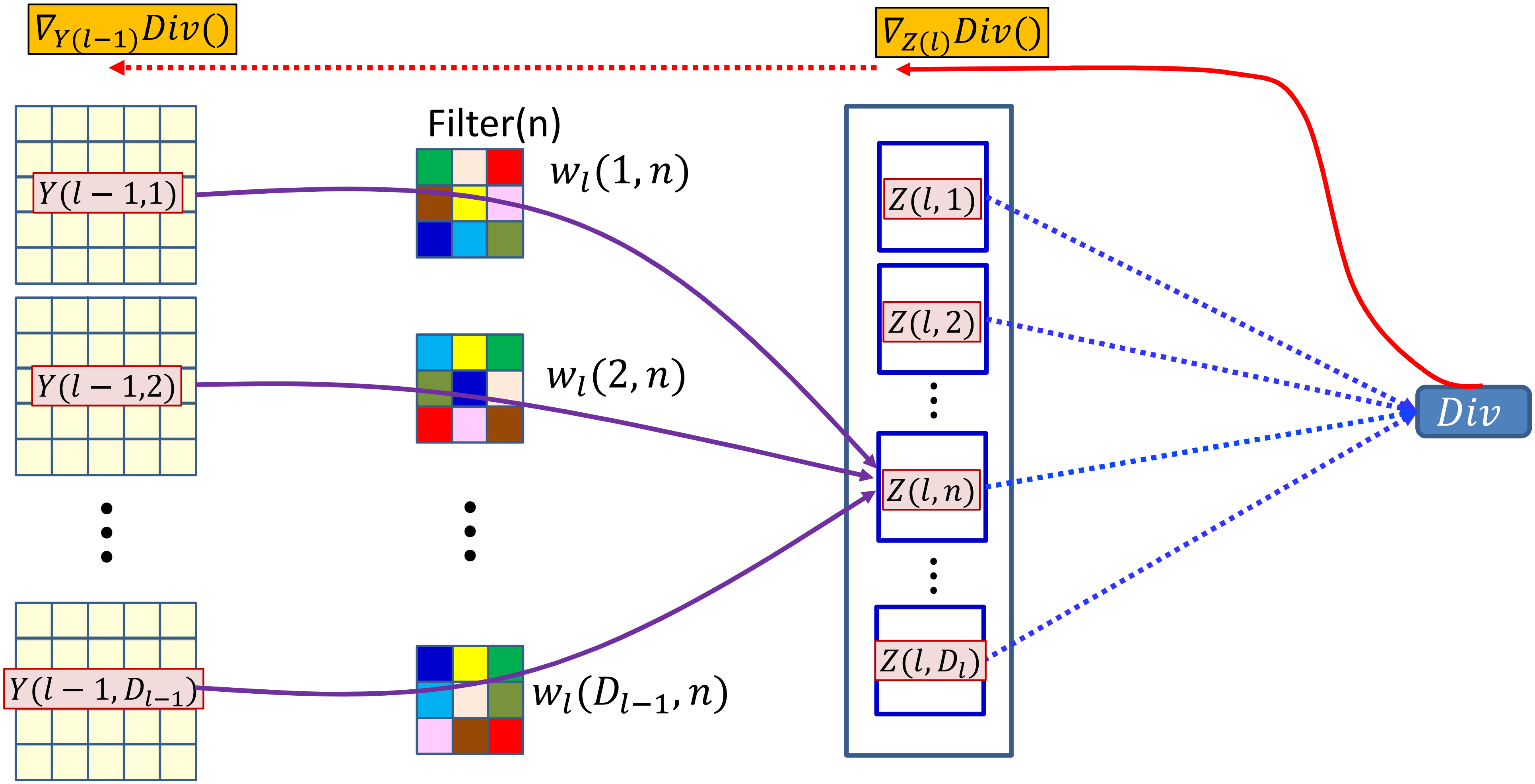
- We already have the derivative w.r.t $Z(l,*)$
  - Having backpropagated it past $Y(l,*)$

Backpropagating through the affine map



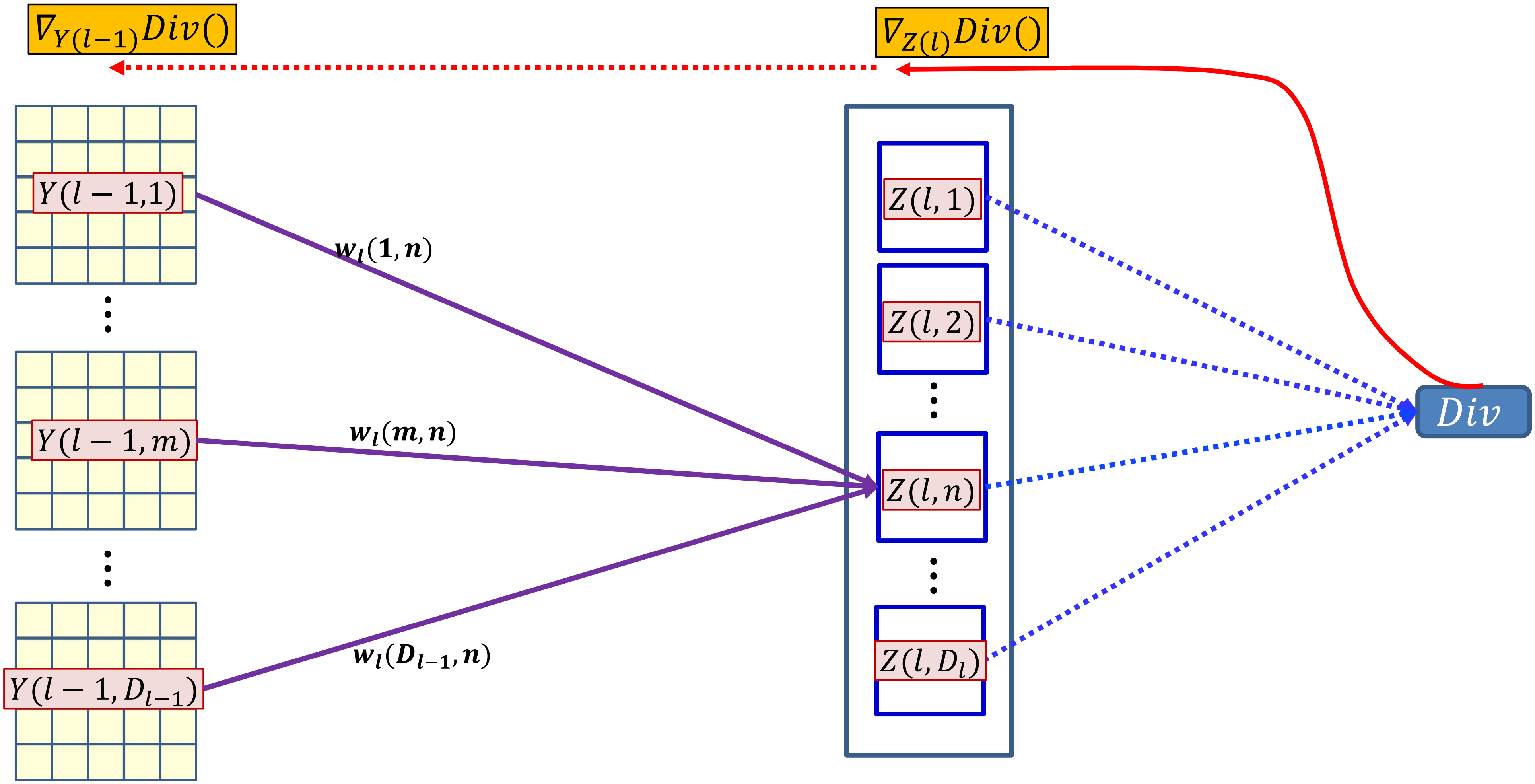
- We already have the derivative w.r.t $Z(l,*)$
  - Having backpropagated it past $Y(l,*)$
- We must compute the derivative w.r.t $Y(l-1,*)$

Dependency between Z(l,n) and Y(l-1,*)



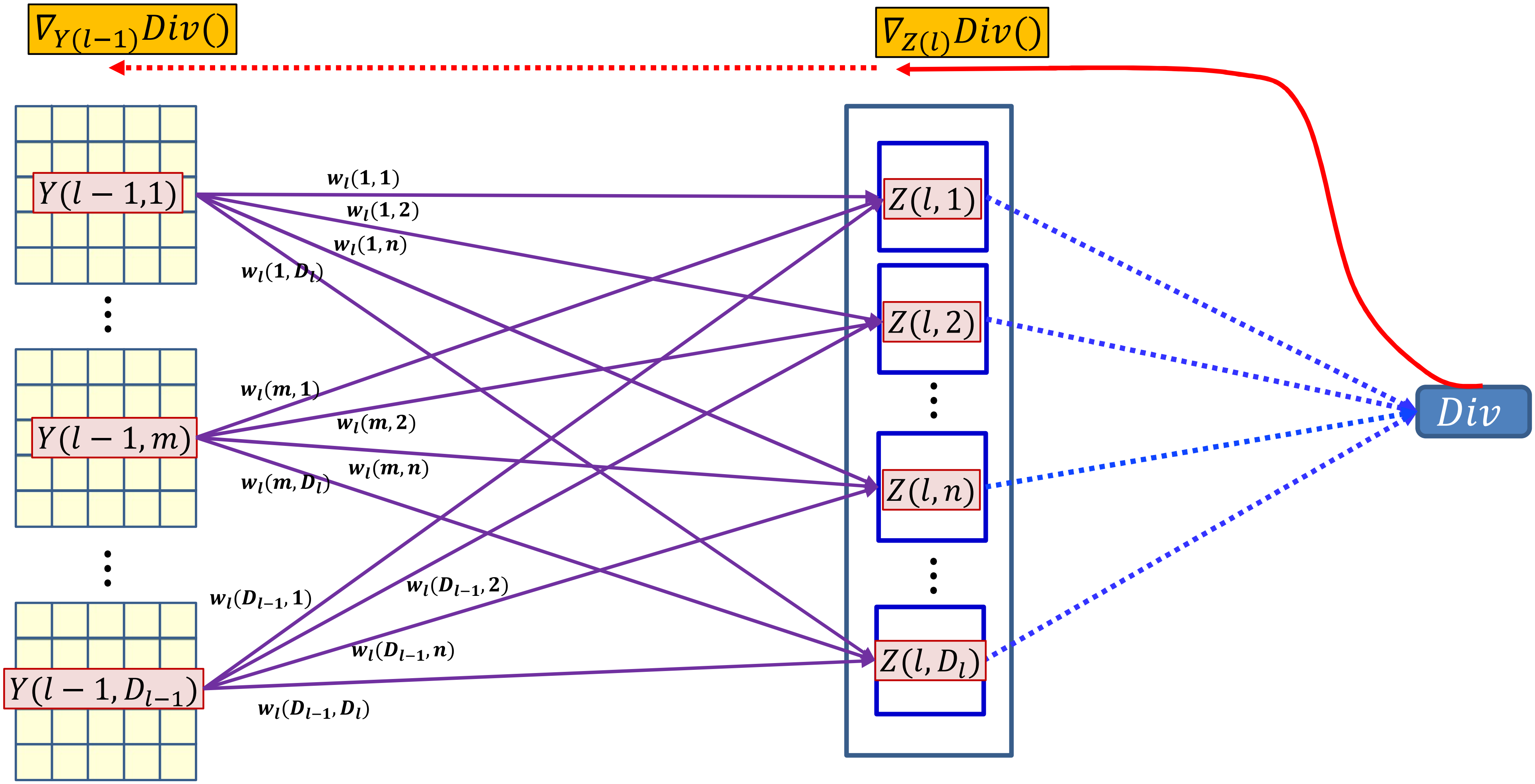
- Each $Y(l-1,m)$ map/channel influences $Z(l,n)$ through the mth channel of the nth filter $w_l(m,n)$

Dependency between Z(l,n) and Y(l-1,*)



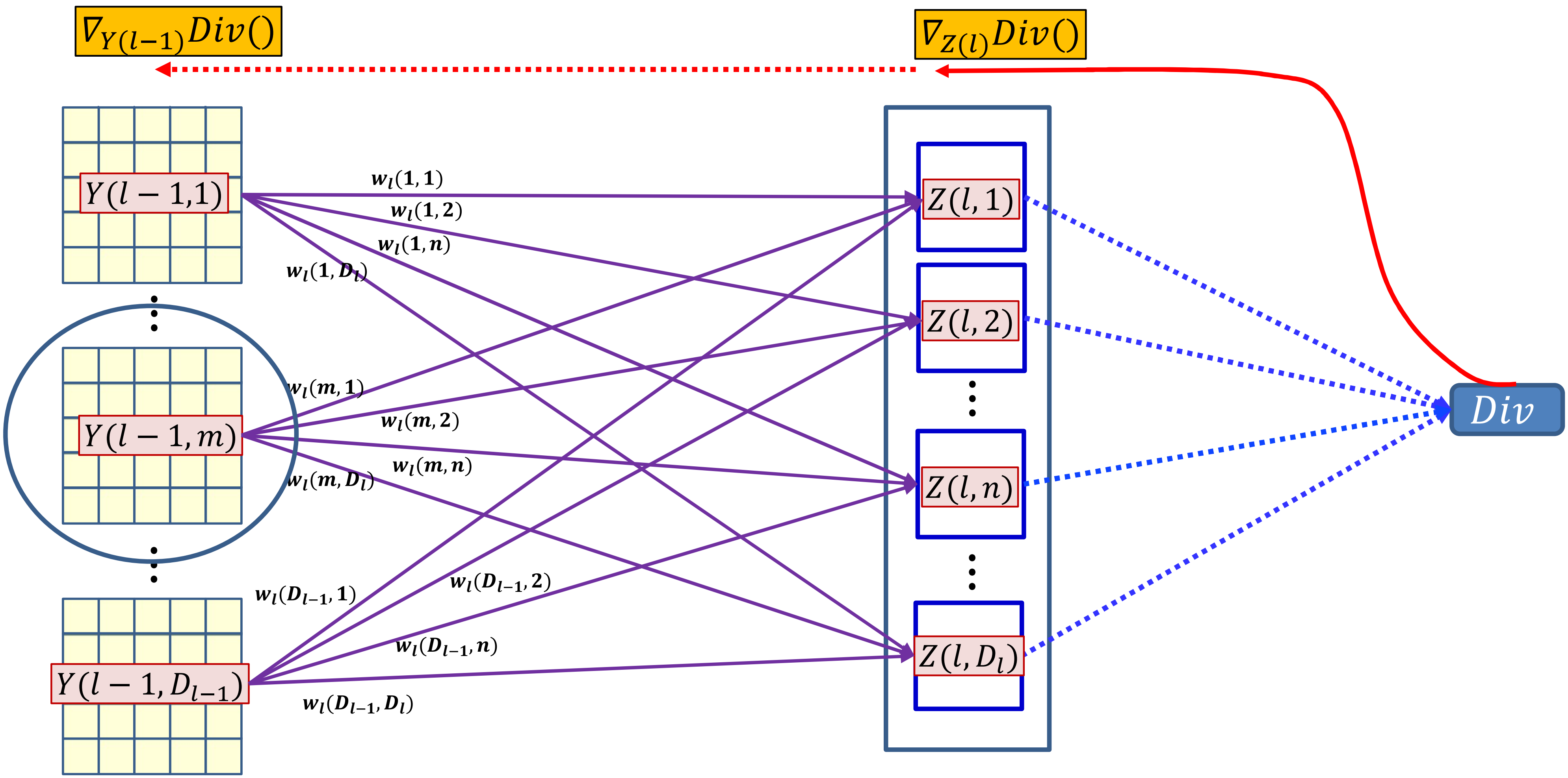
- Each $Y(l-1,m)$ map/channel influences $Z(l,n)$ through the mth channel of the nth filter $w_l(m,n)$

Dependency between Z(l,*) and Y(l-1,*)



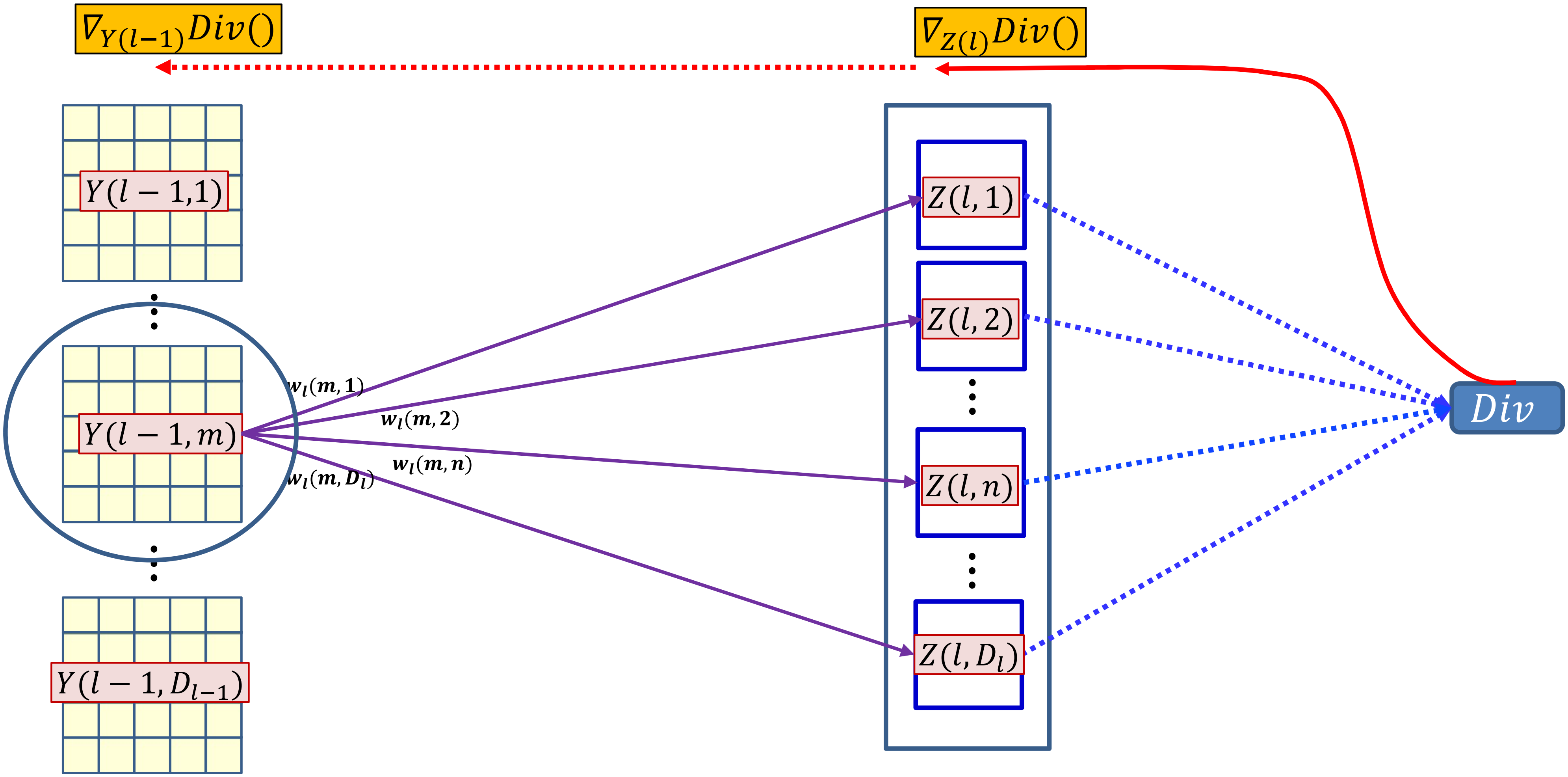
- Each $Y(l-1,m)$ map/channel influences $Z(l,n)$ through the mth channel of the nth filter $w_l(m,n)$

Dependency between Z(l,*) and Y(l-1,*)



- Each $Y(l-1,m)$ map/channel influences $Z(l,n)$ through the mth channel of the nth filter $w_l(m,n)$

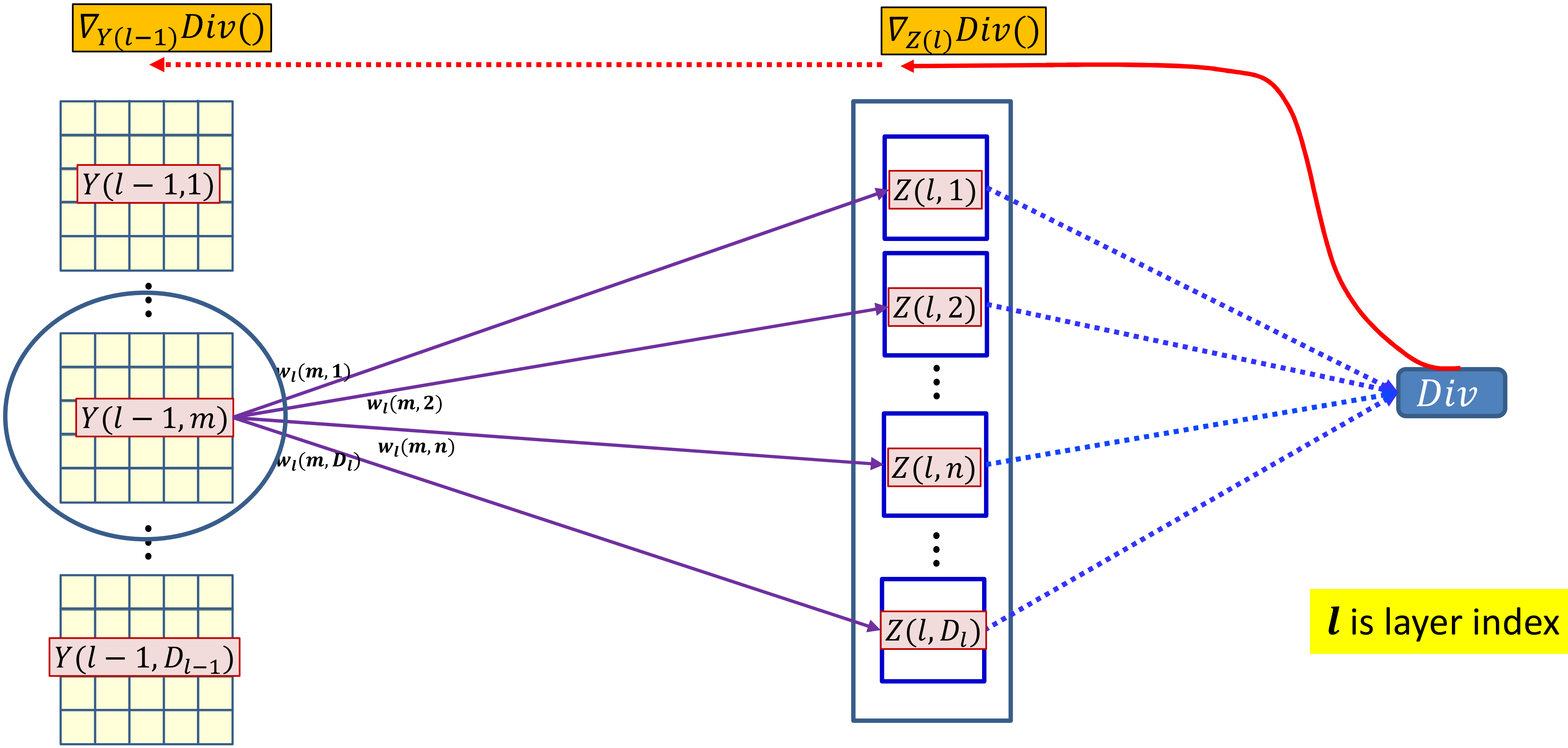
Dependency diagram for a single map



- Each $Y(l-1,m)$ map/channel influences $Z(l,n)$ through the mth channel of the nth filter $w_l(m,n)$

- $Y(l-1,m,*,*)$ influences the divergence through all $Z(l,n,*,*)$ maps

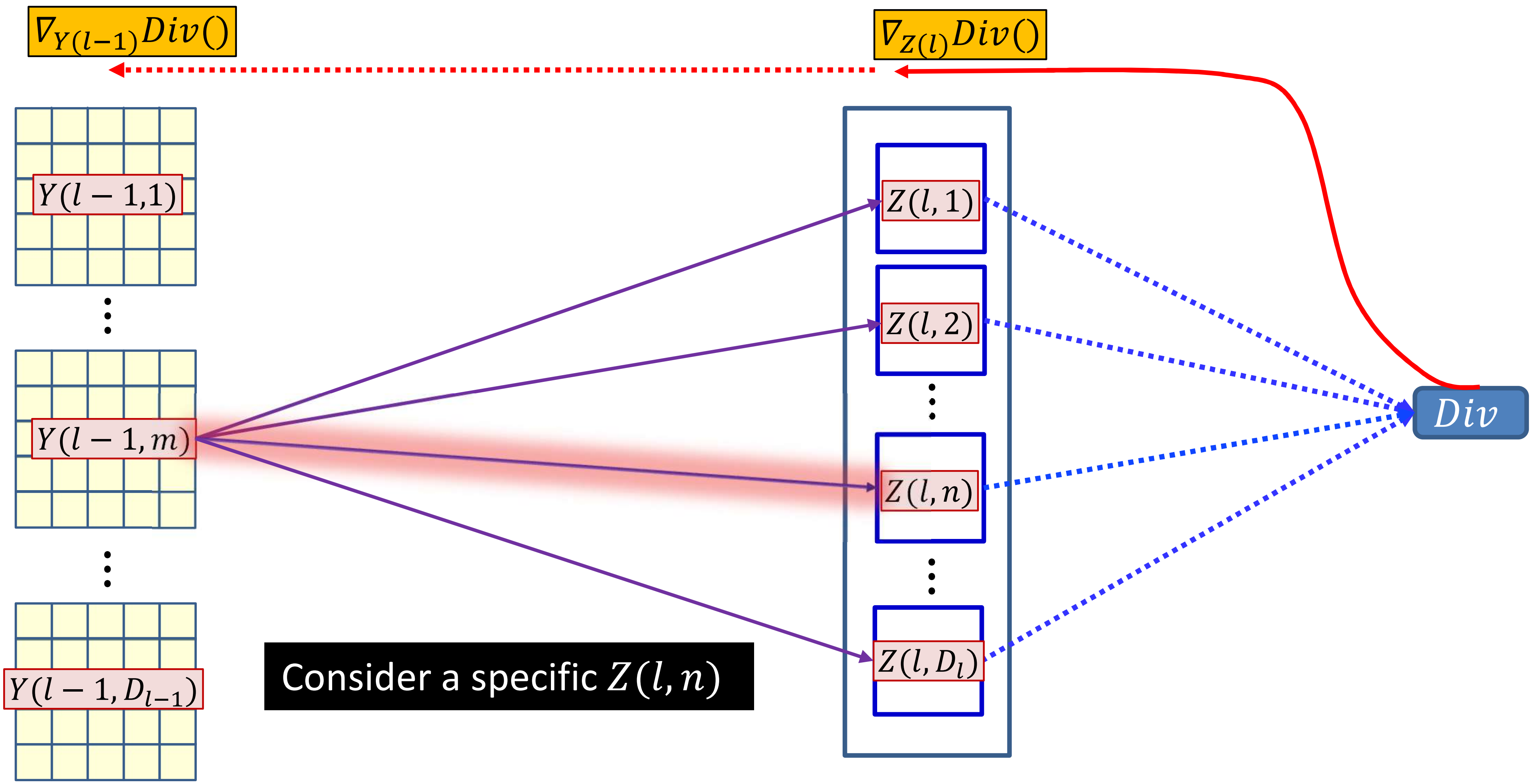
Dependency diagram for a single map



$$\nabla_{Y(l-1,m)} Div(.) = \sum_n \nabla_{Z(l,n)} Div(.)\, \underbrace{\nabla_{Y(l-1,m)} Z(l,n)}$$

- Need to compute $\nabla_{Y(l-1,m)} Z(l,n)$, the derivative of $Z(l,n)$ w.r.t. $Y(l-1,m)$ to complete the computation of the formula

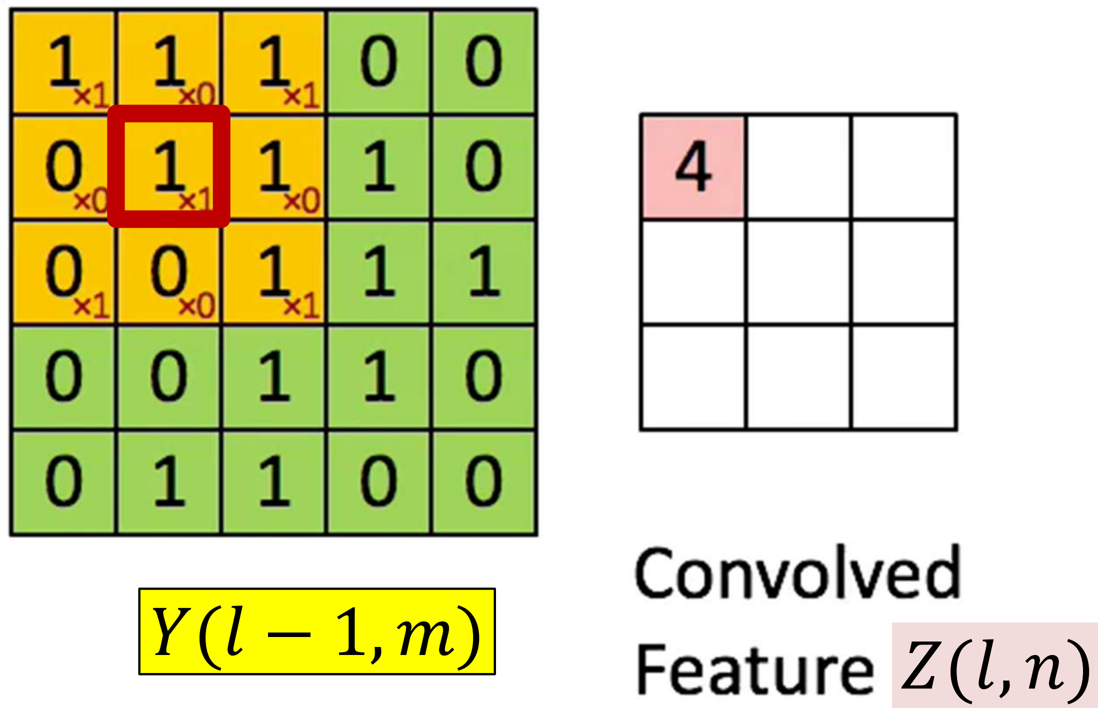
Dependency diagram for a single map



$$\nabla_{Y(l-1,m)}Div(.) = \sum_n \nabla_{Z(l,n)}Div(.)\ \underbrace{\nabla_{Y(l-1,m)}Z(l,n)}$$

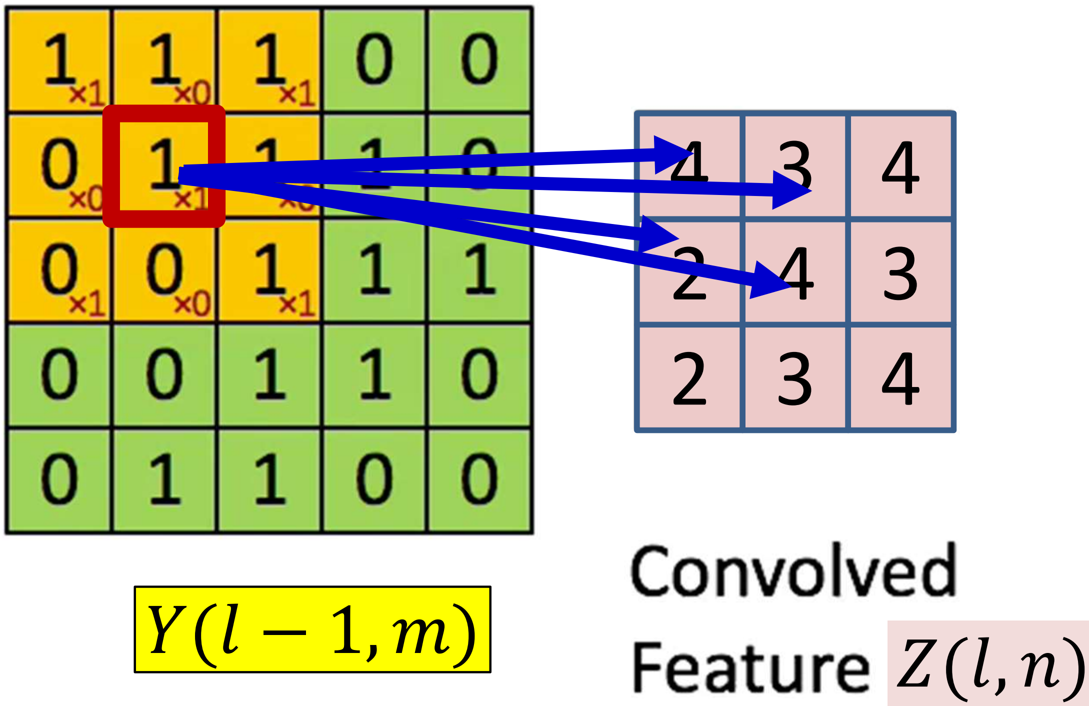
- Need to compute $\nabla_{Y(l-1,m)}Z(l,n)$, the derivative of $Z(l,n)$ w.r.t. $Y(l-1,m)$ to complete the computation of the formula

BP: Convolutional layer



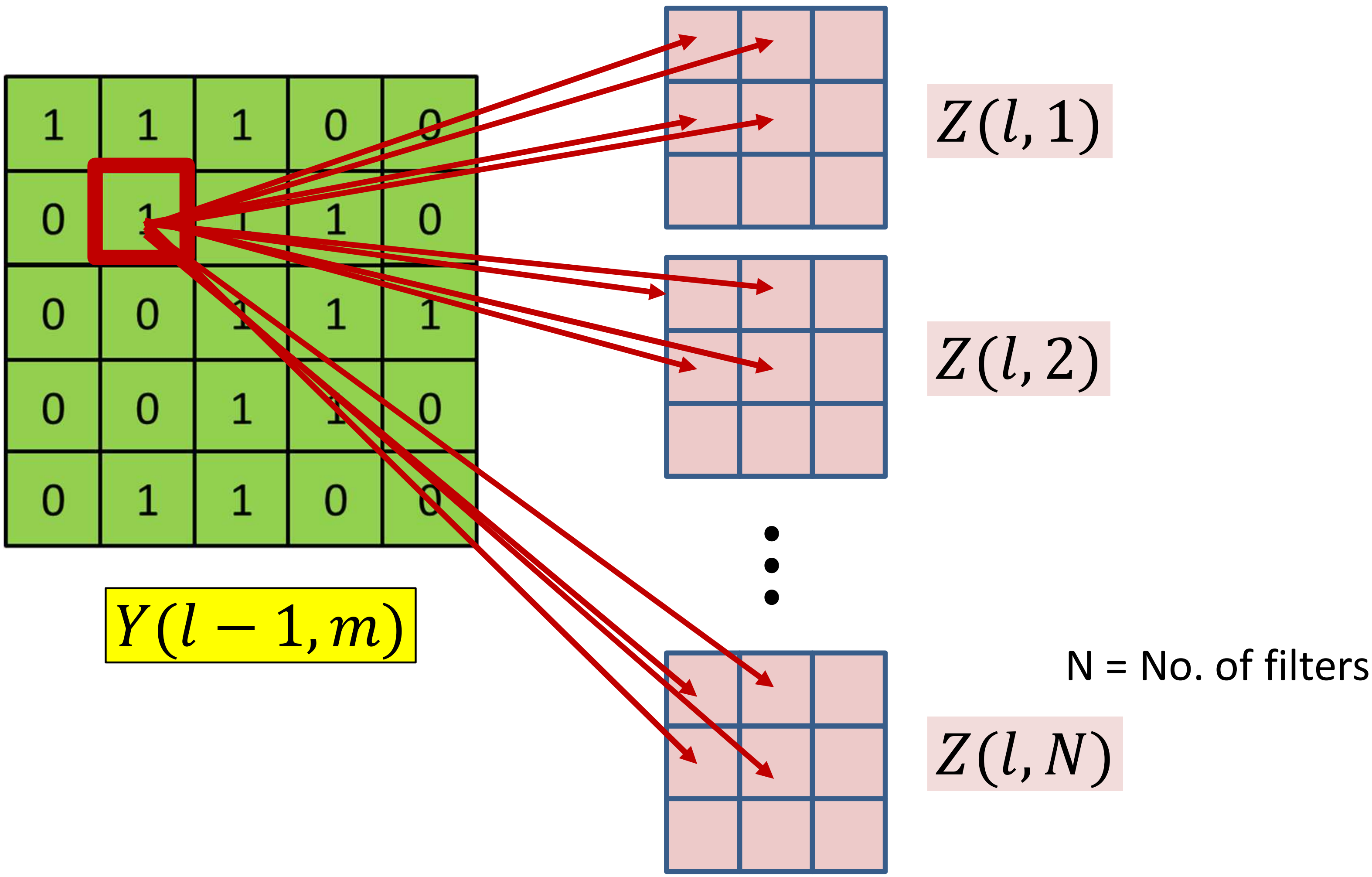
- Each $Y(l-1, m, x, y)$ affects several $z(l, n, x', y')$ terms

BP: Convolutional layer



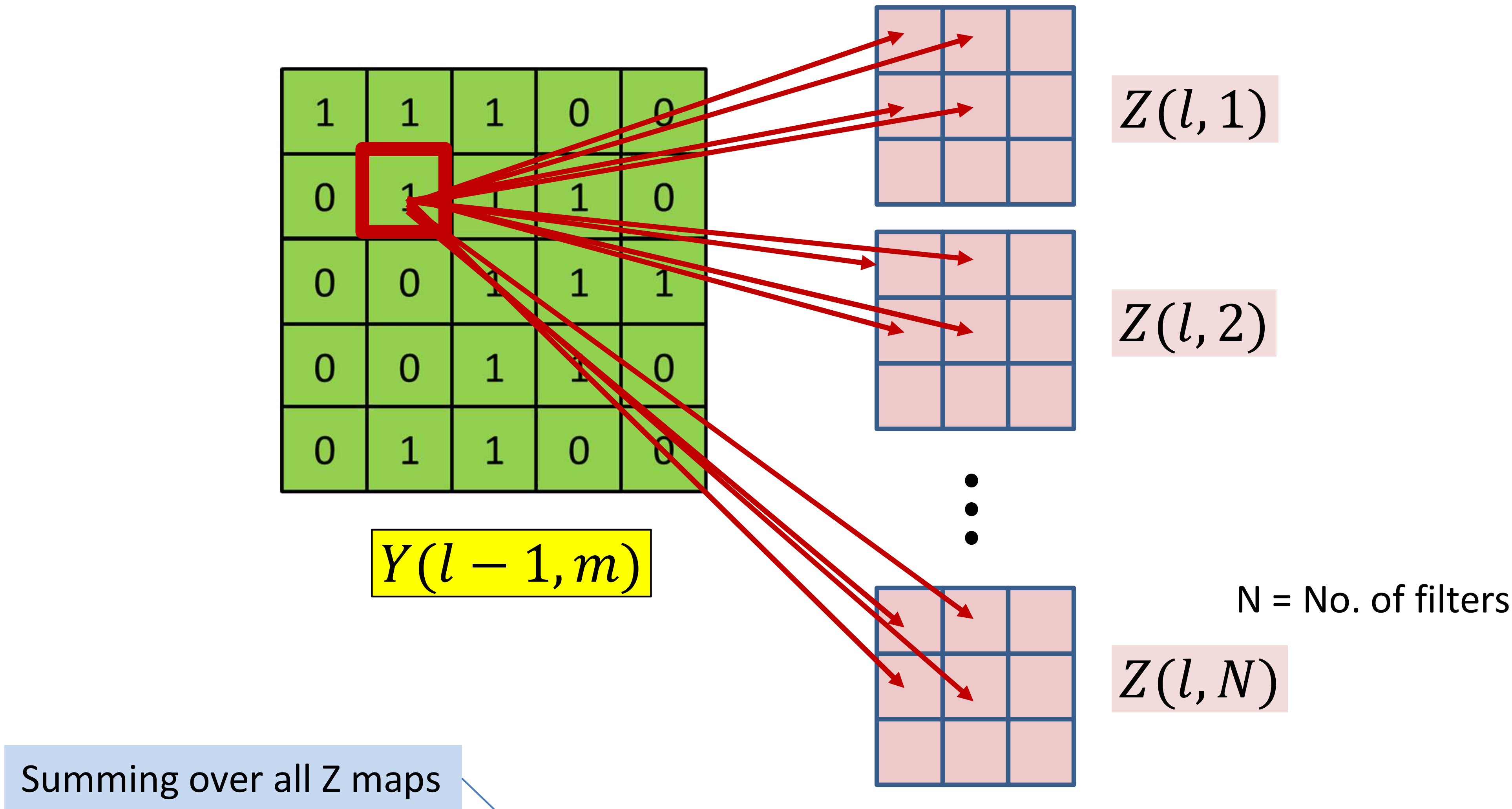
- Each $Y(l-1,m,x,y)$ affects several $z(l,n,x',y')$ terms

BP: Convolutional layer



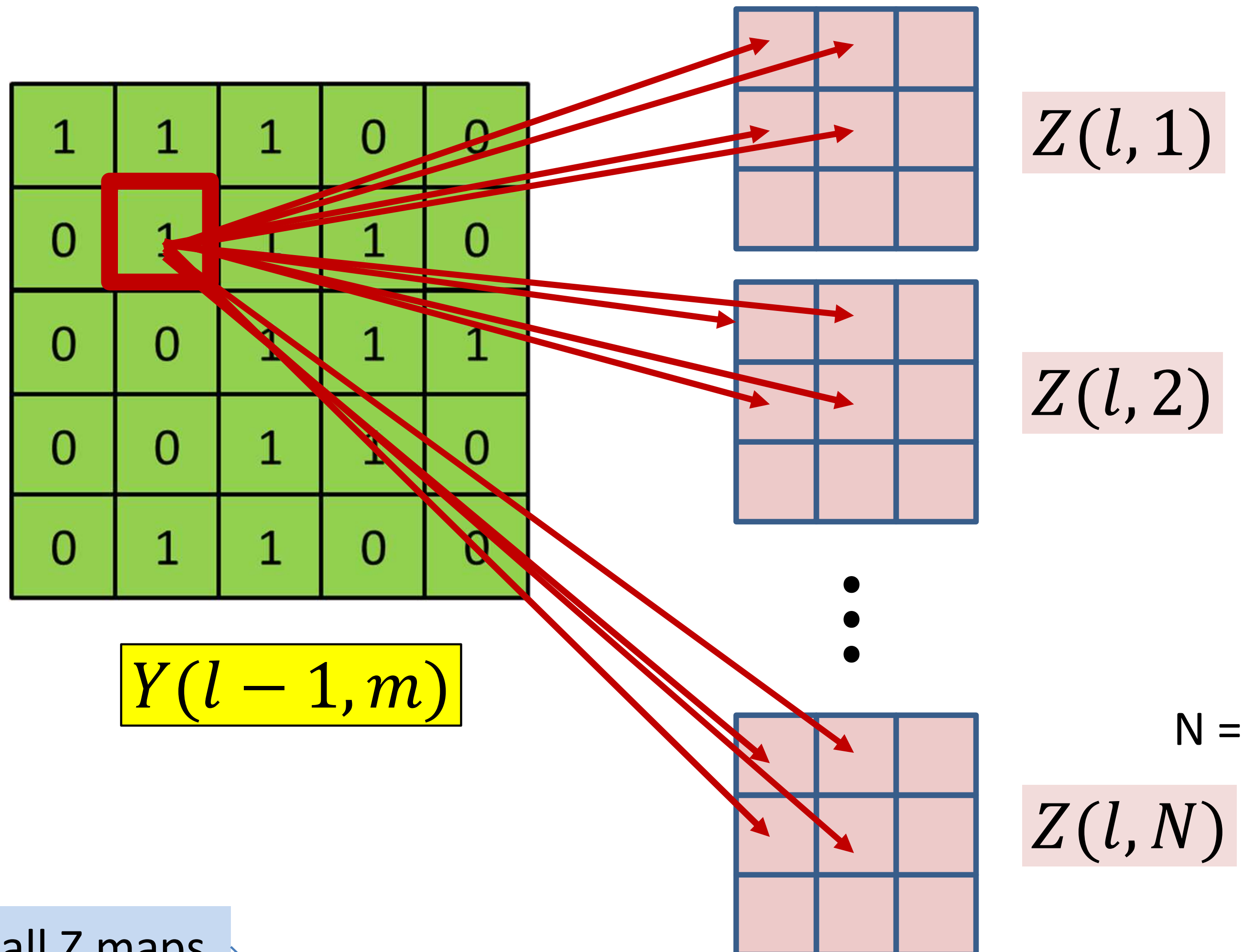
- Each $Y(l-1,m,x,y)$ affects several $z(l,n,x',y')$ terms
  - Affects terms in *all* l th layer Z maps

BP: Convolutional layer



$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} \frac{dz(l,n,x',y')}{dY(l-1,m,x,y)}$$

BP: Convolutional layer



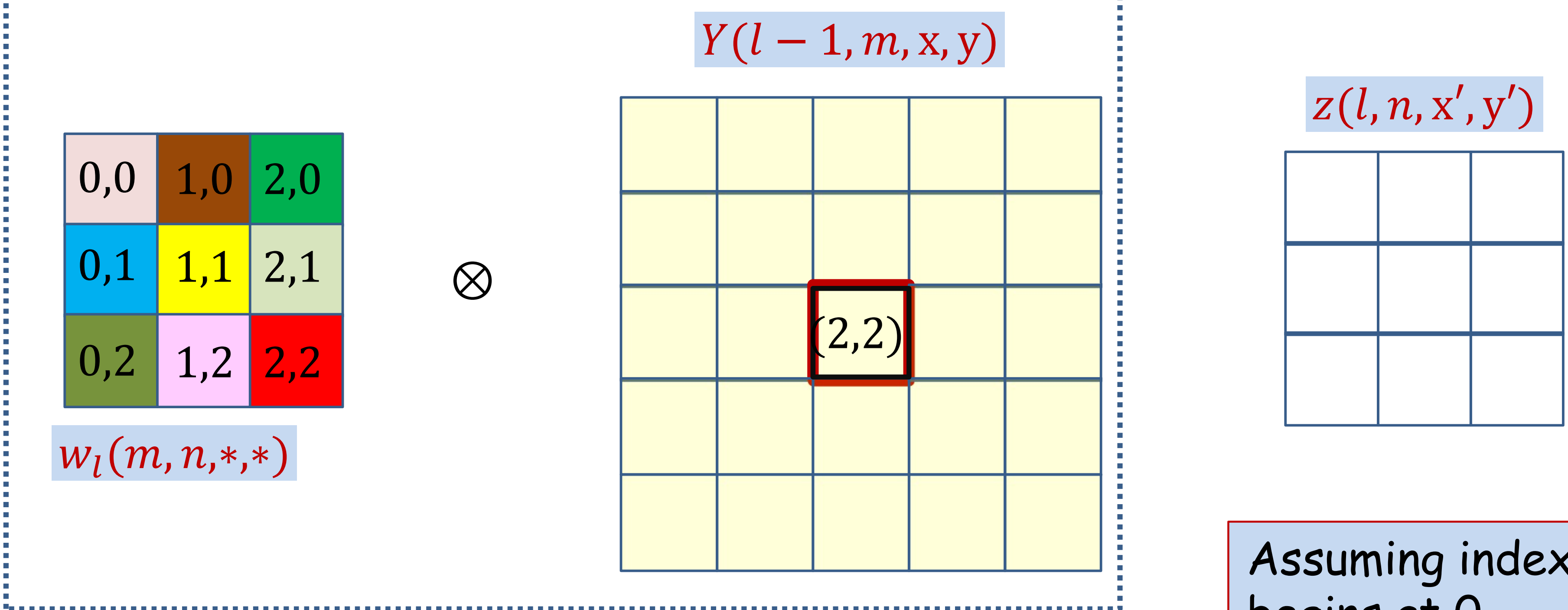
N = No. of filters

Summing over all Z maps

What is this?

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n}\sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')}\frac{dz(l,n,x',y')}{dY(l-1,m,x,y)}$$

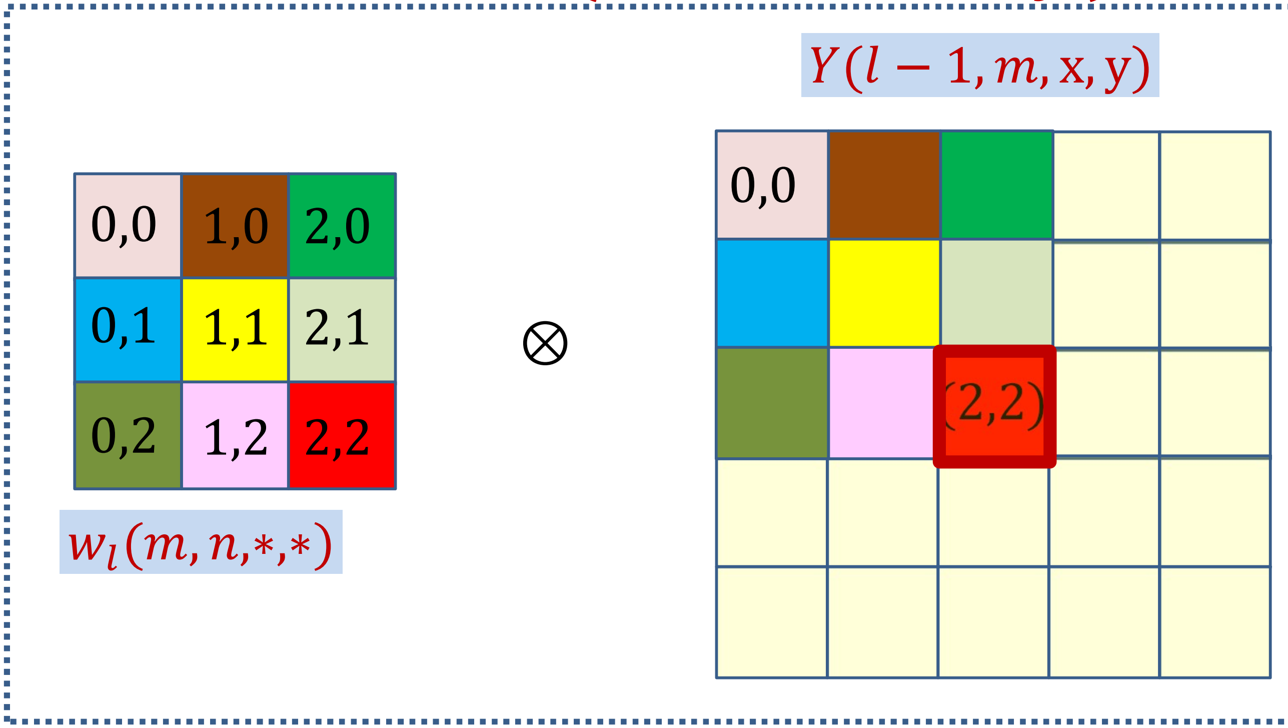
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$

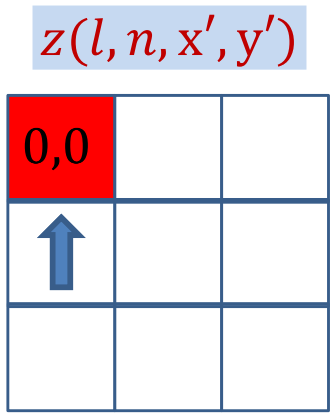


Assuming indexing begins at 0

- Compute how *each* x, y in Y influences various locations of z

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



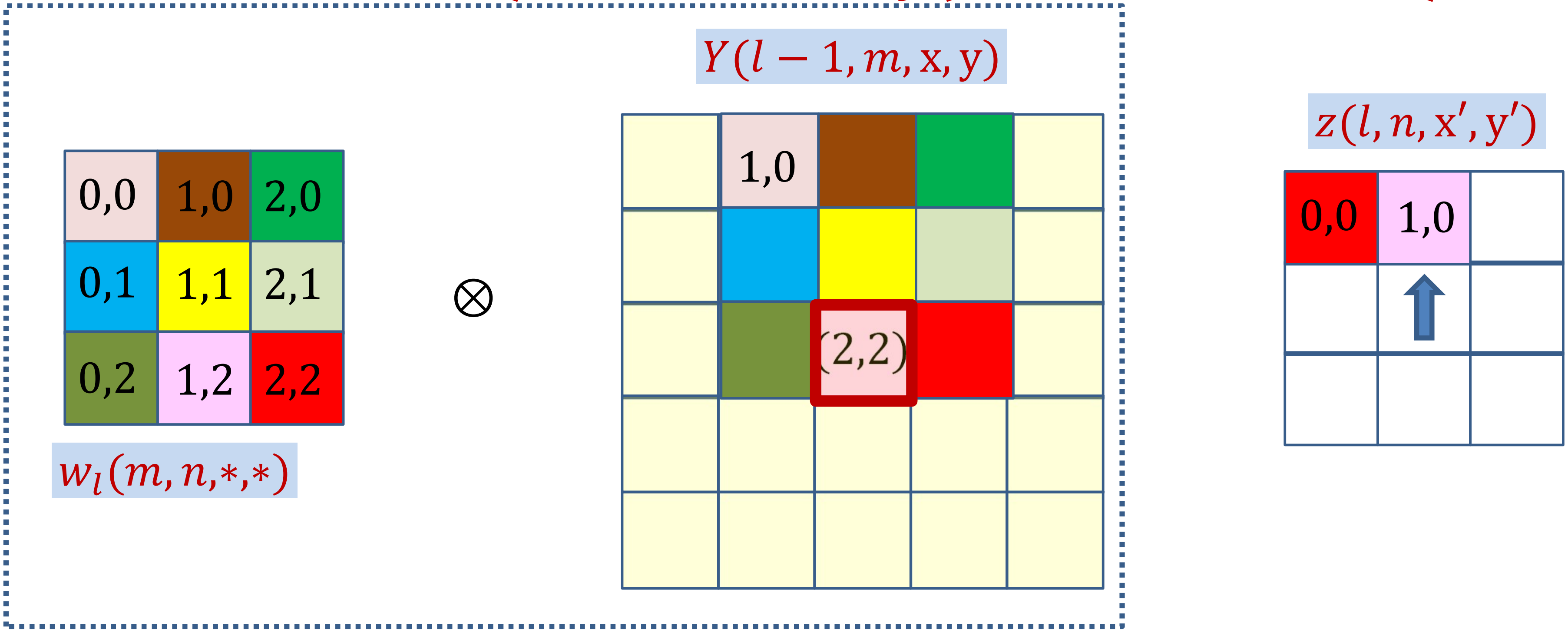


$$z(l,n,0{,}0) += Y(l-1,m,2{,}2)w_l(m,n,2{,}2)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') += Y(l-1,m,2{,}2)w_l(m,n,2-x',2-y')$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

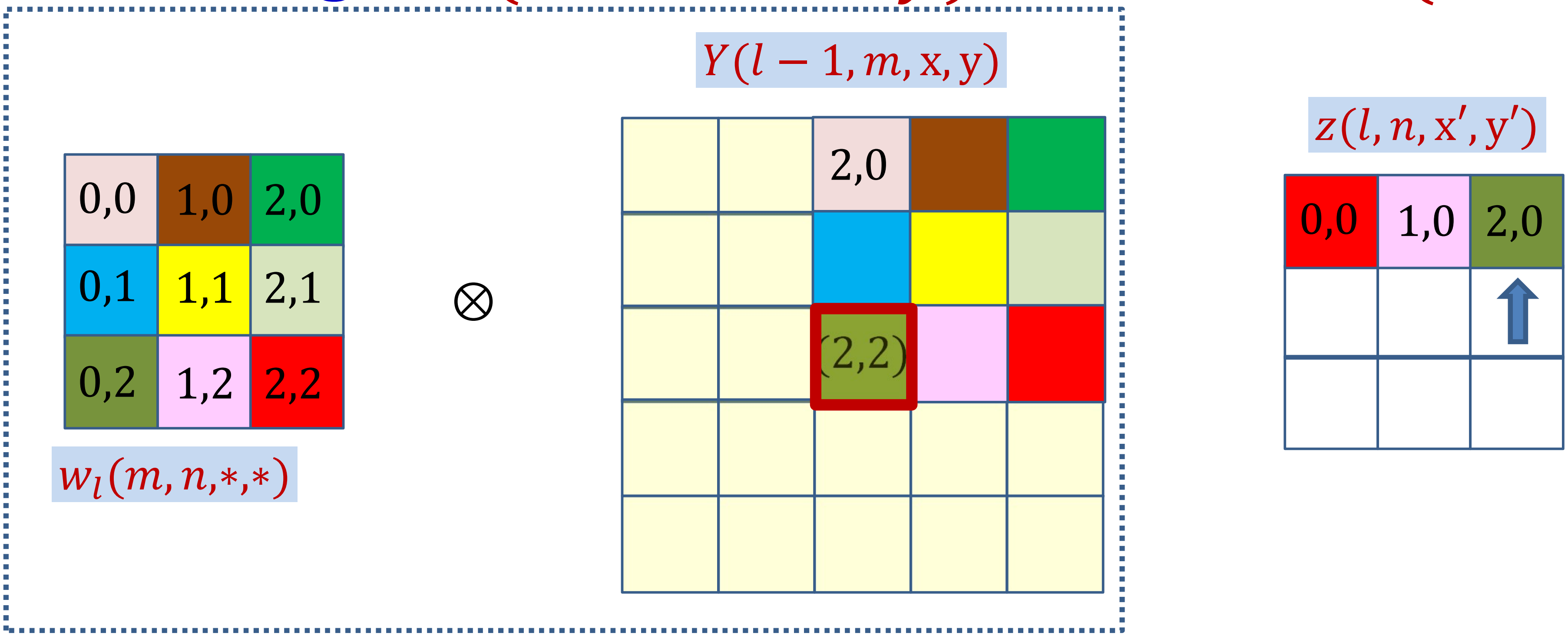


$$z(l,n,1{,}0) \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,1{,}2)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,2-x',2-y')$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

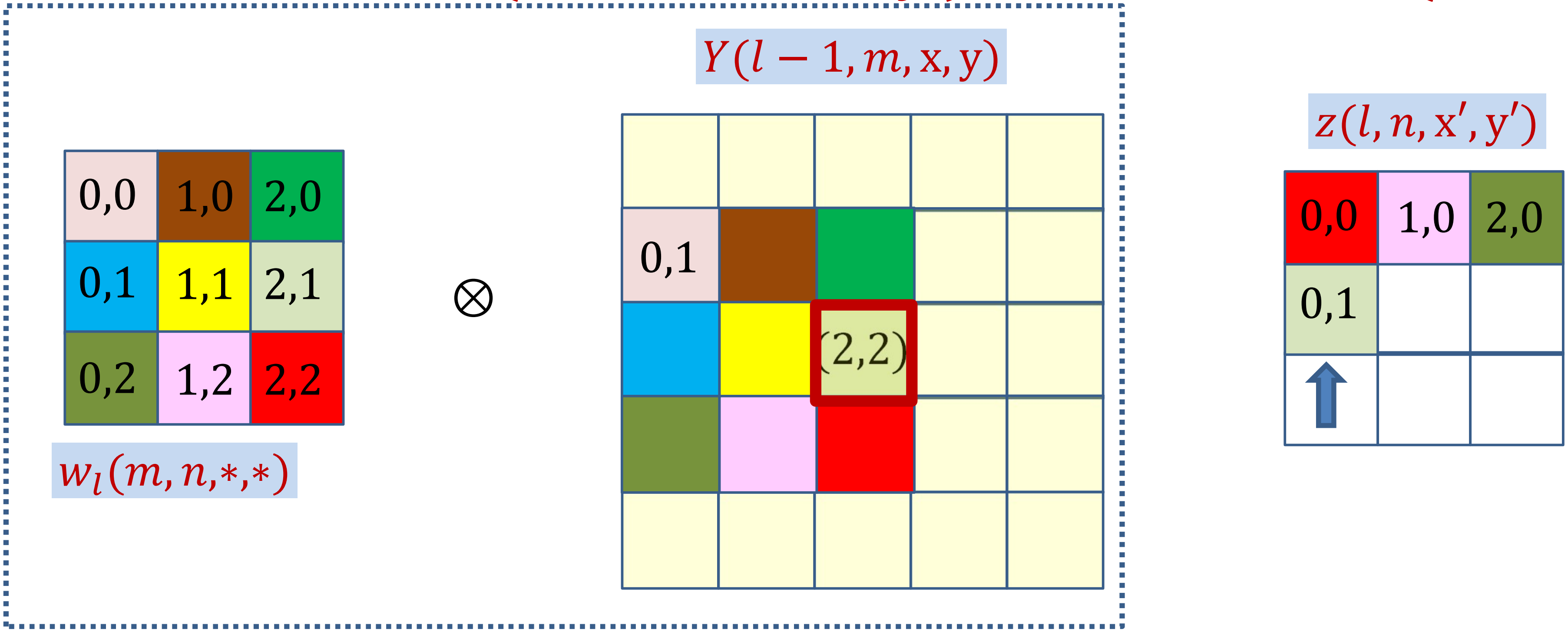


$$z(l,n,2{,}0) \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,0{,}2)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,2-x',2-y')$$

How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$

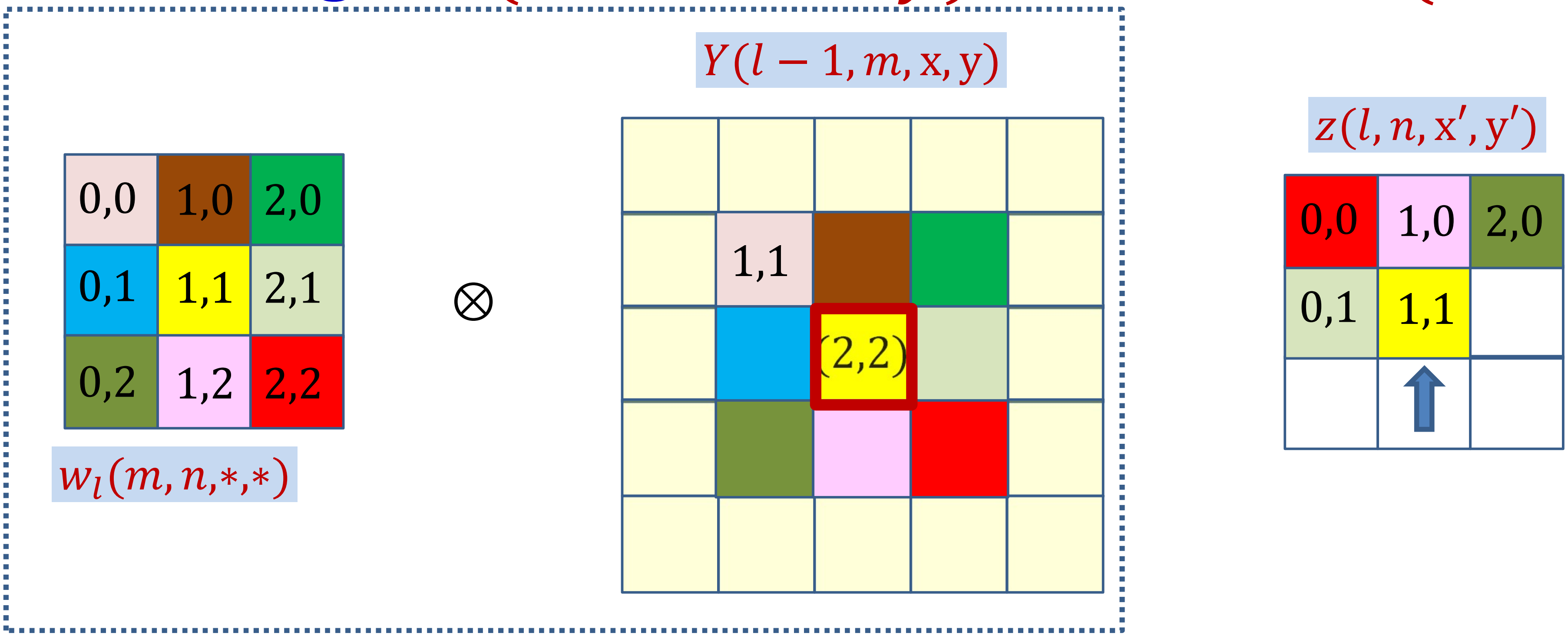


$$z(l, n, 0,1) += Y(l-1, m, 2,2)w_l(m, n, 2,1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l-1, m)$

$$z(l, n, x', y') += Y(l-1, m, 2,2)w_l(m, n, 2-x', 2-y')$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

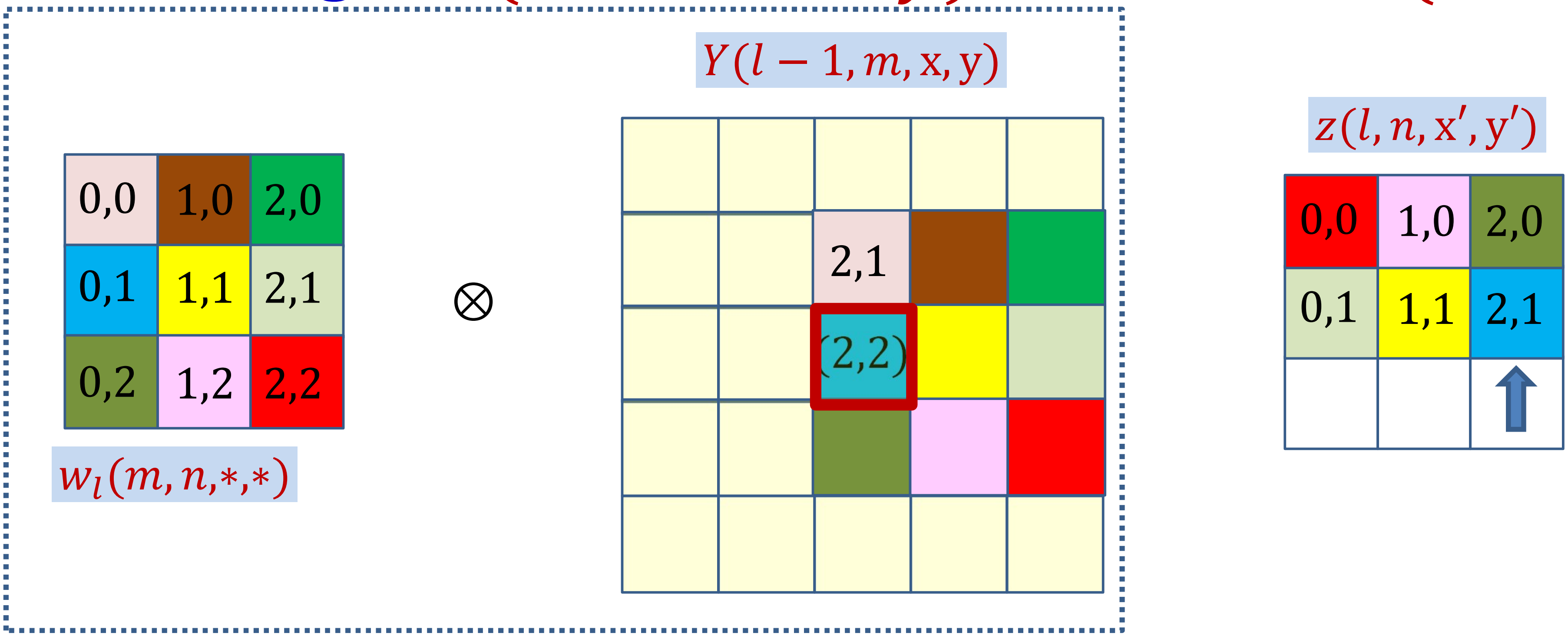


$$z(l,n,1,1) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,1,1)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2-x',2-y')$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

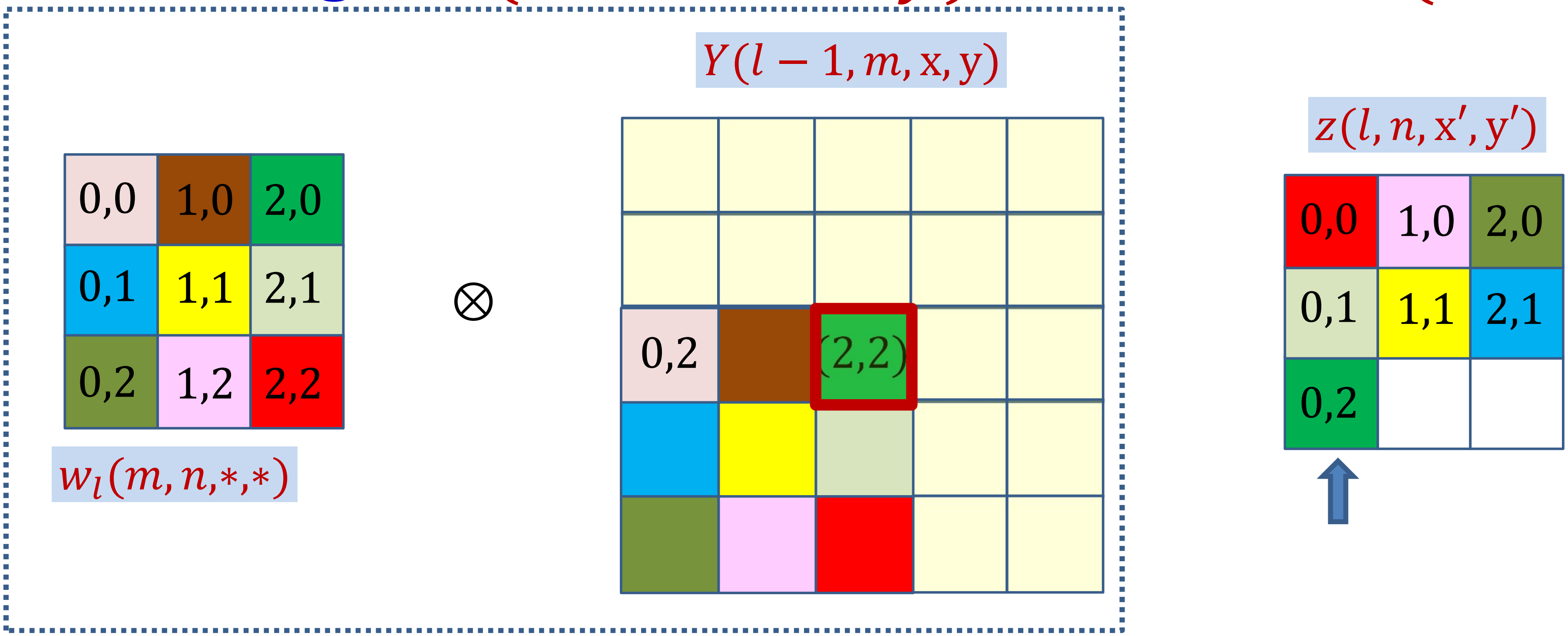


$$z(l,n,2,1) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,1)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2-x',2-y')$$

How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$

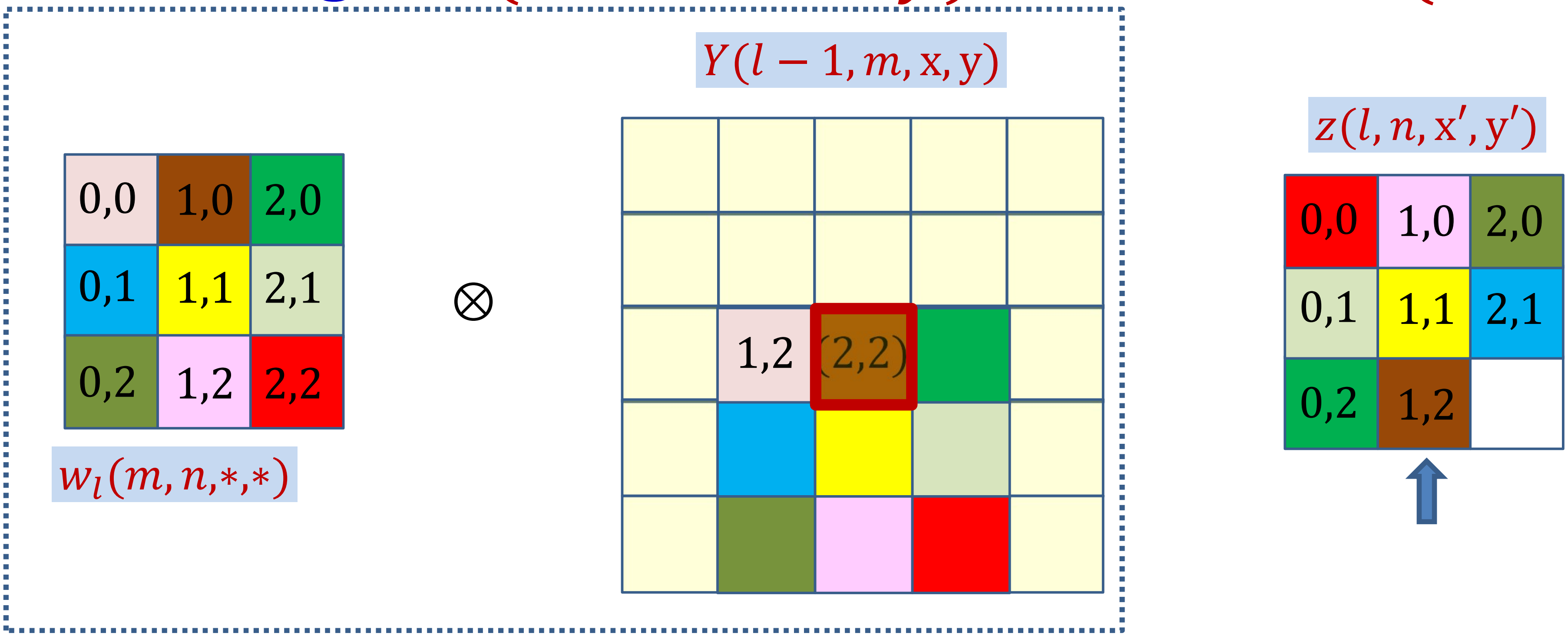


$$z(l, n, 0{,}2) \mathrel{+}= Y(l-1, m, 2{,}2)\, w_l(m, n, 2{,}0)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l-1, m)$

$$z(l, n, x', y') \mathrel{+}= Y(l-1, m, 2{,}2)\, w_l(m, n, 2-x', 2-y')$$

How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$

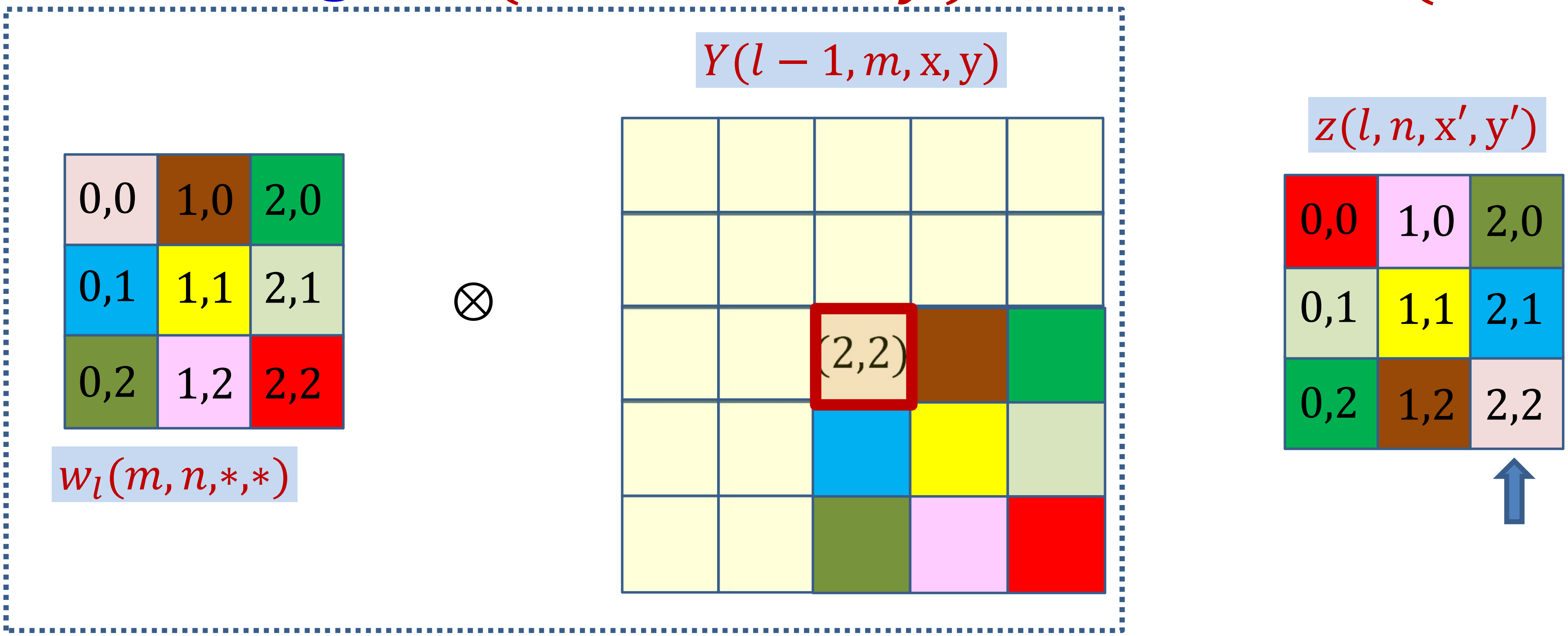


$$z(l, n, 1{,}2) \mathrel{+}= Y(l-1, m, 2{,}2)\, w_l(m, n, 2{,}1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l-1, m)$

$$z(l, n, x', y') \mathrel{+}= Y(l-1, m, 2{,}2)\, w_l(m, n, 2-x', 2-y')$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

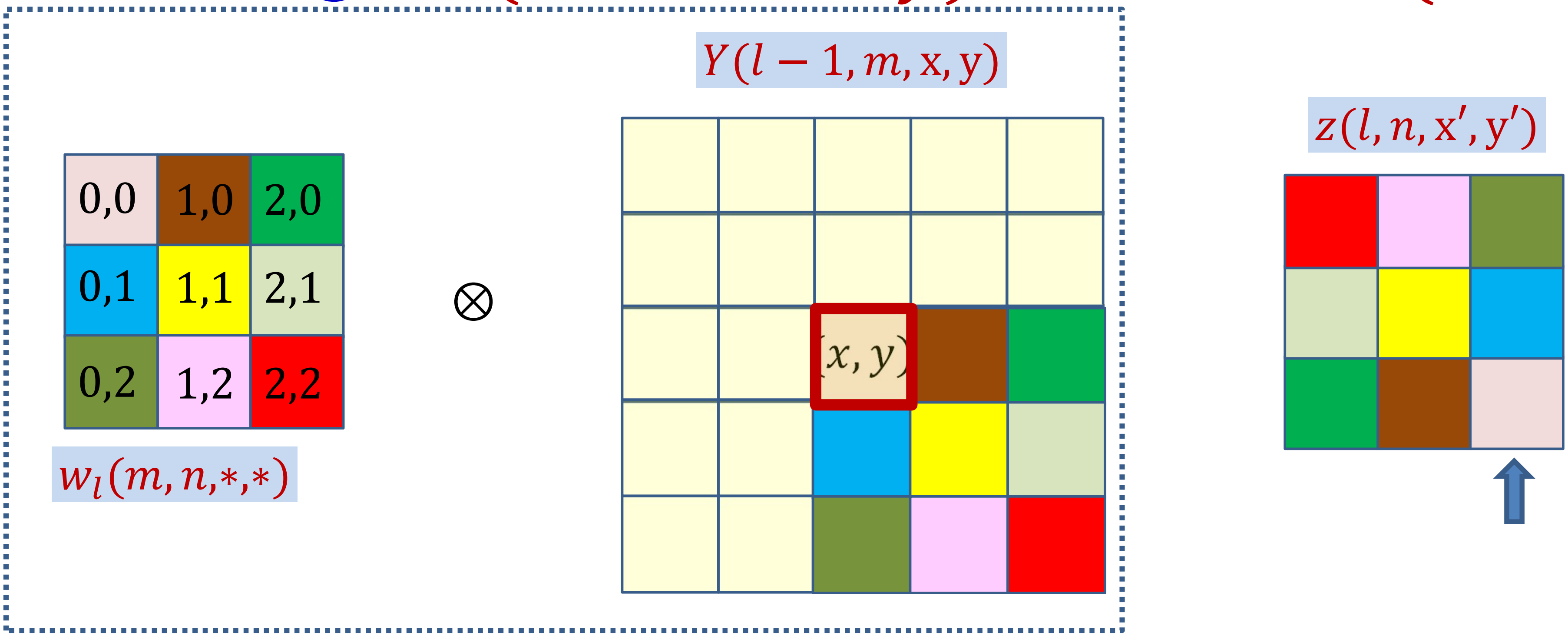


$$z(l,n,2{,}2) \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,0{,}0)$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,2{,}2)w_l(m,n,2-x',2-y')$$

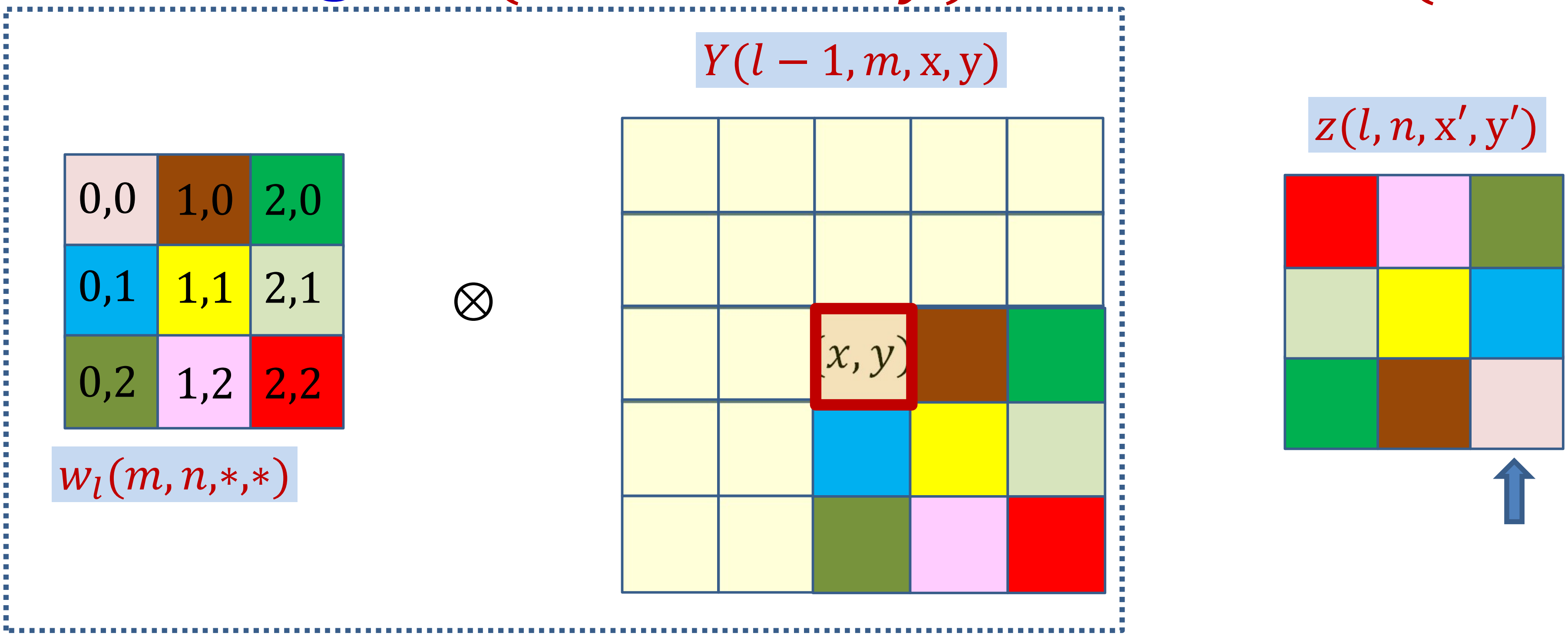
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,x,y)w_l(m,n,x-x',y-y')$$

- **Note:** The coordinates of $z(l,n)$ and $w_l(m,n)$ sum to the coordinates of $Y(l-1,m)$

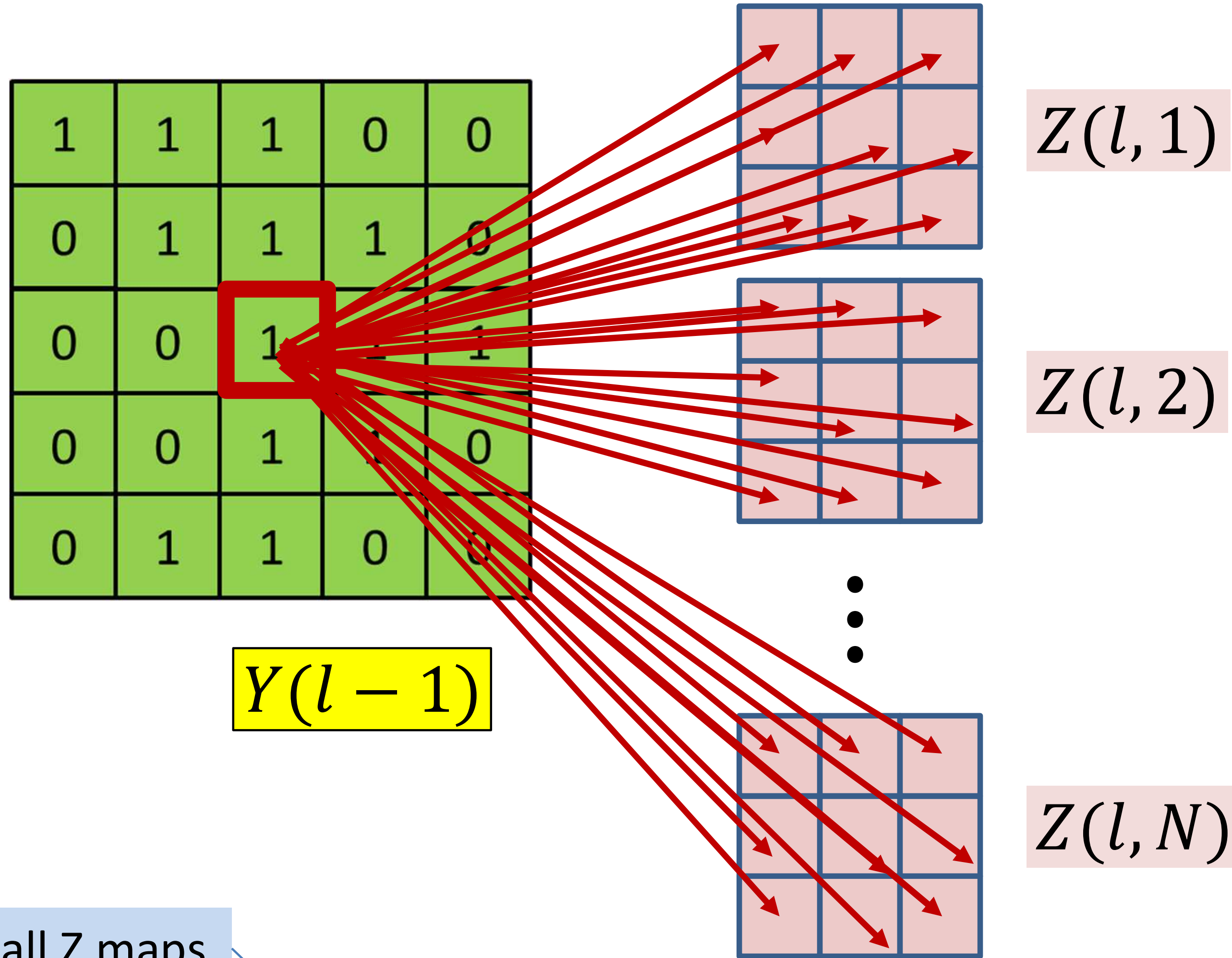
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$z(l,n,x',y') \mathrel{+}= Y(l-1,m,x,y)w_l(m,n,x-x',y-y')$$

$$\frac{dz(l,n,x',y')}{dY(l-1,m,x,y)} = w_l(m,n,x-x',y-y')$$

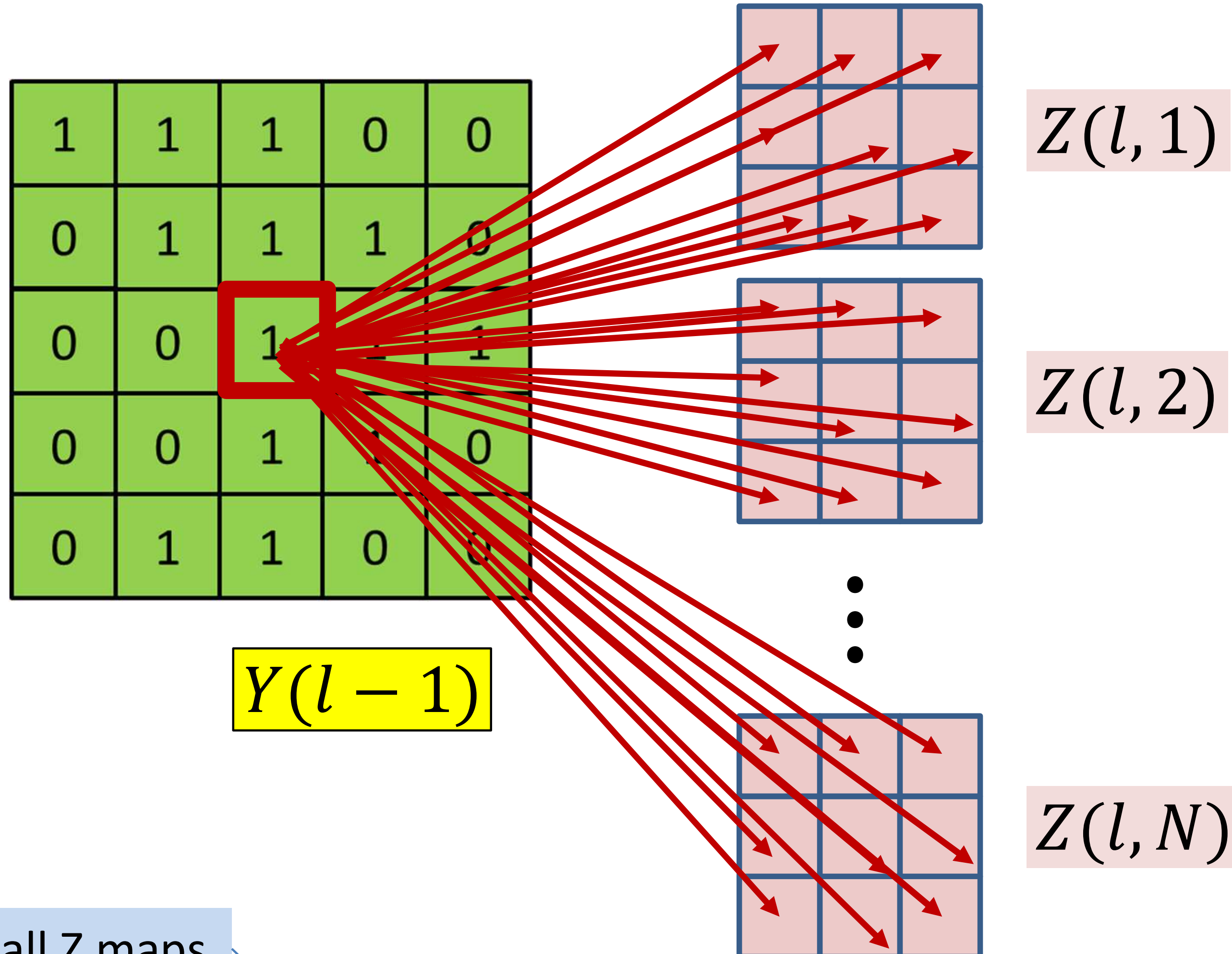
BP: Convolutional layer



Summing over all Z maps

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} \frac{dz(l,n,x',y')}{dY(l-1,m,x,y)}$$

BP: Convolutional layer



Summing over all Z maps

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

Poll 2

In order to compute the derivative at a single affine element Y(l,m,x,y), we must consider the contributions of *every* position of *every* affine map at the next layer: True or false

- True
- False

The derivative for a single affine element Y(l,m,x,y) will require summing over every position of every Z map in the next layer: True of false

- True
- False

Poll 2

In order to compute the derivative at a single affine element Y(l,m,x,y), we must consider the contributions of *every* position of *every* affine map at the next layer: True or false

- **True**
- False

The derivative for an single affine element Y(l,m,x,y) will require summing over every position of every Z map in the next layer: True of false

- **True**
- False

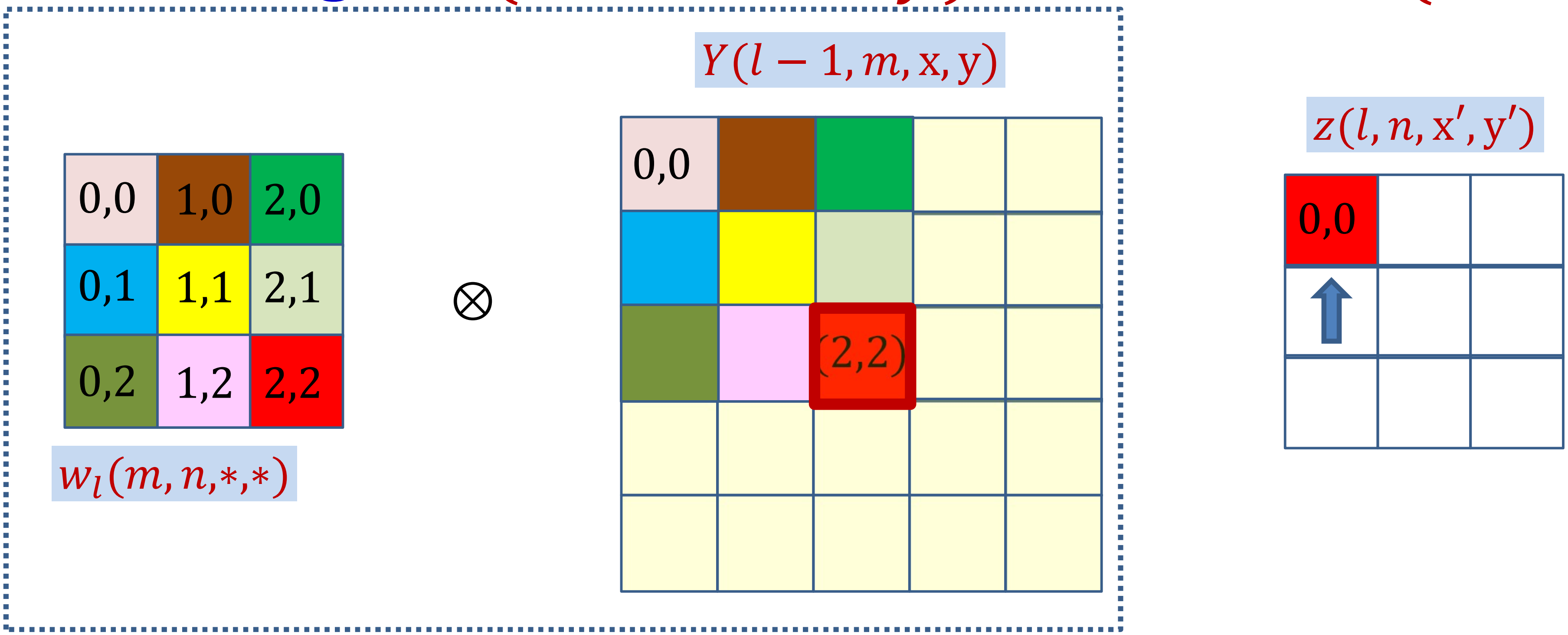
Computing derivative for $Y(l-1, m, *, *)$

- The derivatives for every element of every map in $Y(l-1)$ by direct implementation of the formula:

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

- But this is actually a convolution!
  - Let's see how

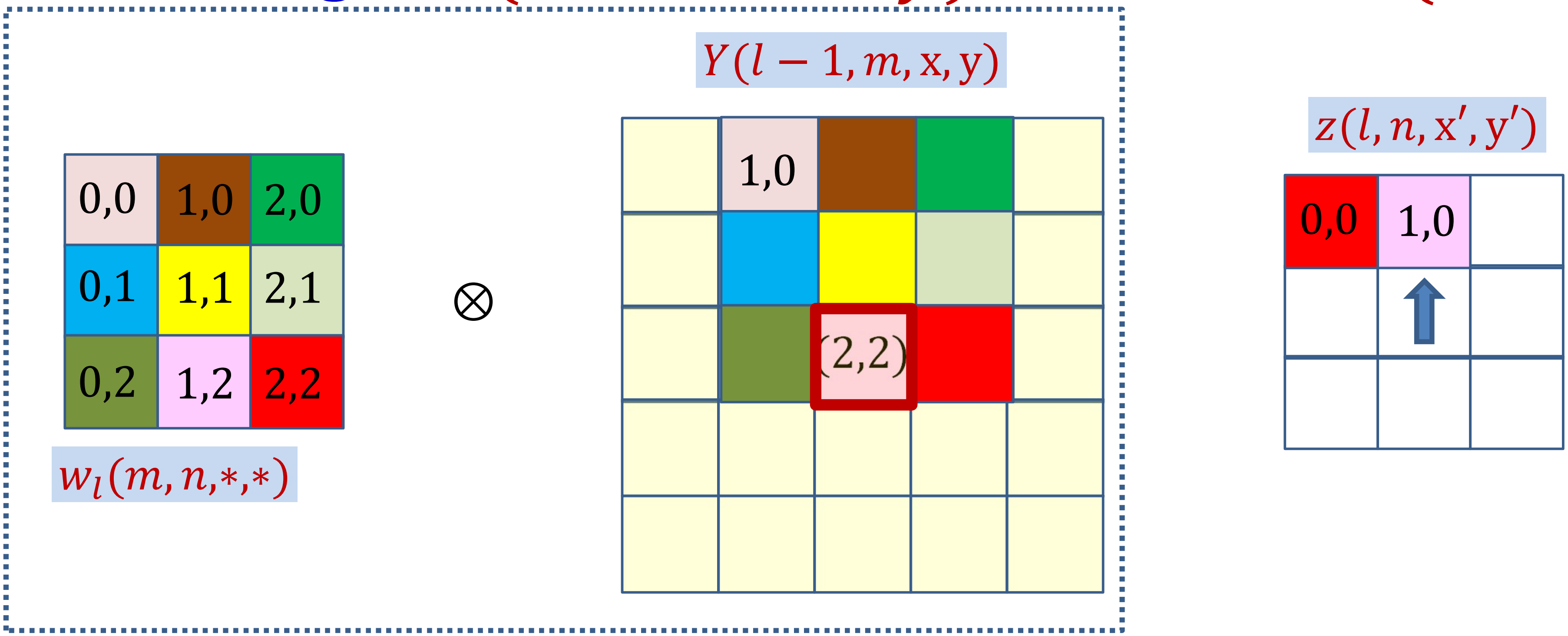
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$z(l,n,0,0) \mathrel{+}= Y(l-1,m,2{,}2)\,w_l(m,n,2{,}2)$$

$$\frac{dDiv}{dY(l-1,m,2{,}2)} \mathrel{+}= \frac{dDiv}{dz(l,n,0{,}0)}\,w_l(m,n,2{,}2)$$

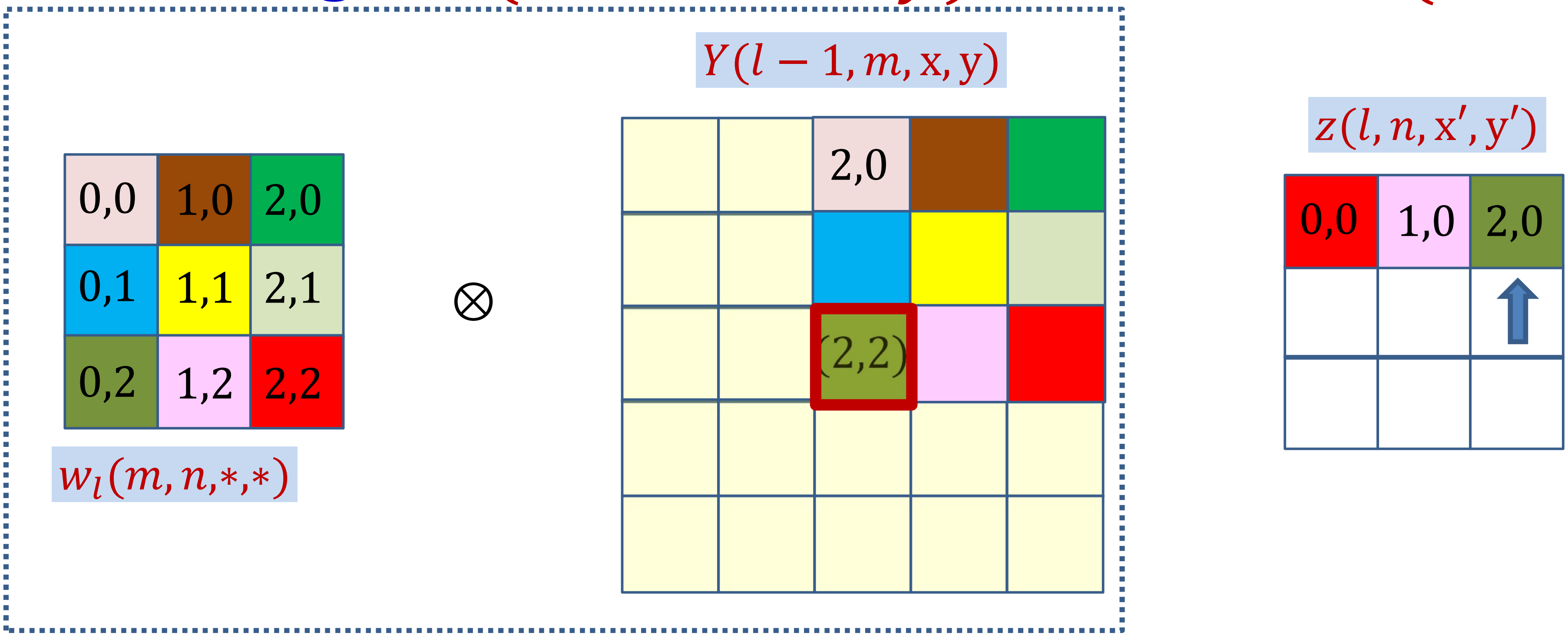
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 1{,}0) \mathrel{+}= Y(l-1, m, 2{,}2)\, w_l(m, n, 1{,}2)$$

$$\frac{dDiv}{dY(l-1, m, 2{,}2)} \mathrel{+}= \frac{dDiv}{dz(l, n, 1{,}0)} w_l(m, n, 1{,}2)$$

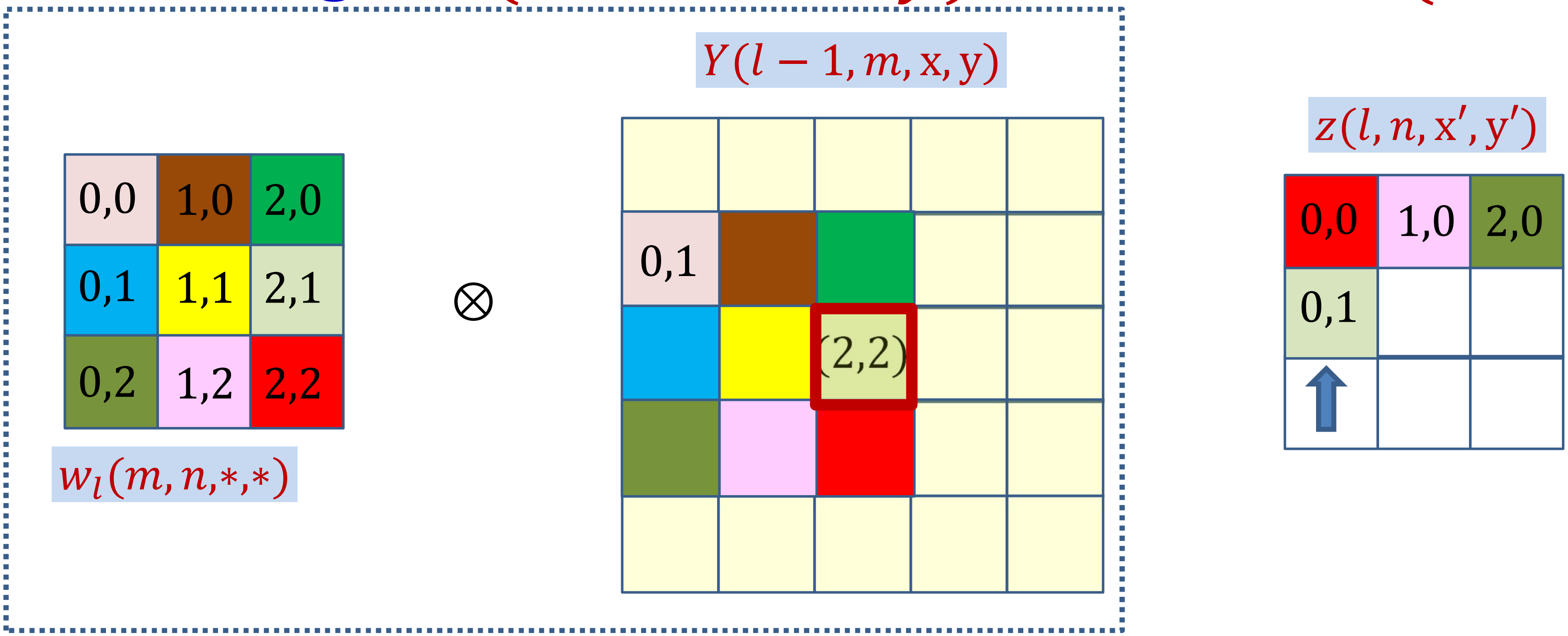
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$z(l,n,2,0) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,2)$$

$$\frac{dDiv}{dY(l-1,m,2,2)} \mathrel{+}= \frac{dDiv}{dz(l,n,2,0)} w_l(m,n,0,2)$$

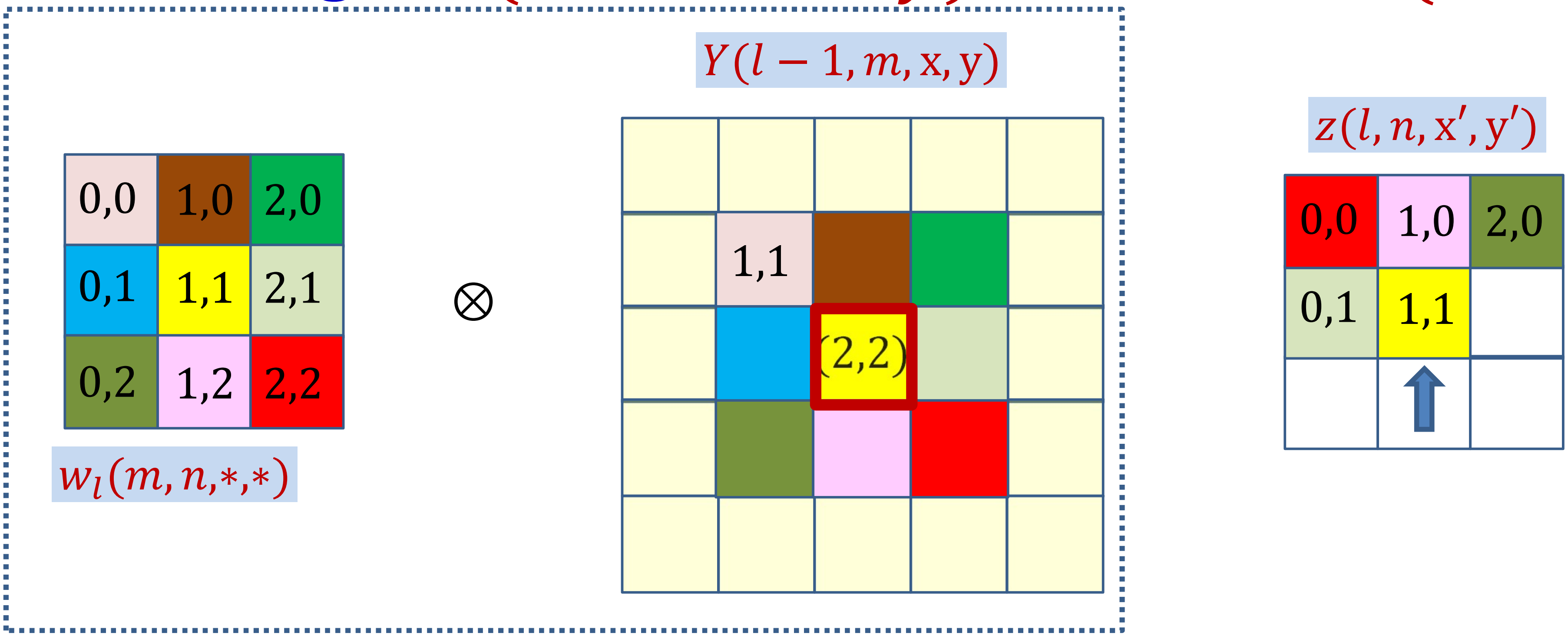
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 0,1) += Y(l-1, m, 2,2) w_l(m, n, 2,1)$$

$$\frac{dDiv}{dY(l-1, m, 2,2)} += \frac{dDiv}{dz(l, n, 0,1)} w_l(m, n, 2,1)$$

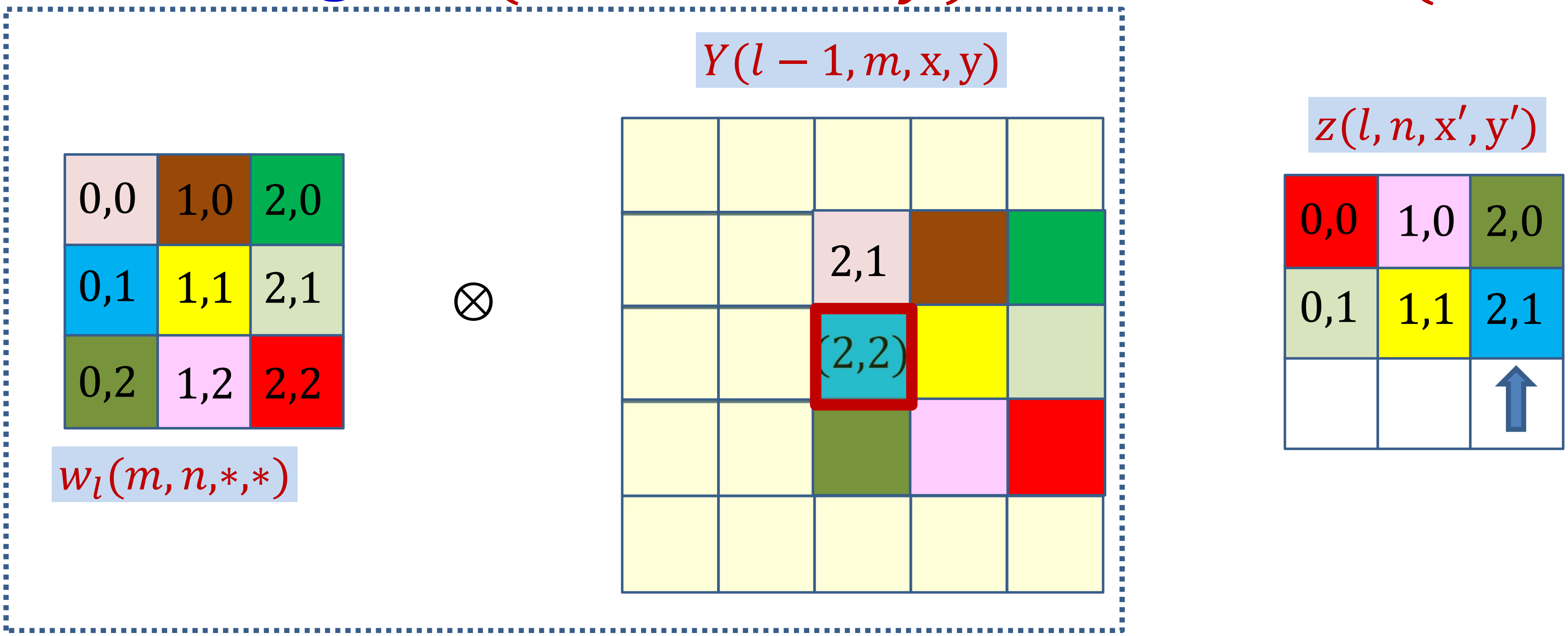
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 1,1) += Y(l-1, m, 2,2) w_l(m, n, 1,1)$$

$$\frac{dDiv}{dY(l-1, m, 2,2)} += \frac{dDiv}{dz(l, n, 1, 1)} w_l(m, n, 1,1)$$

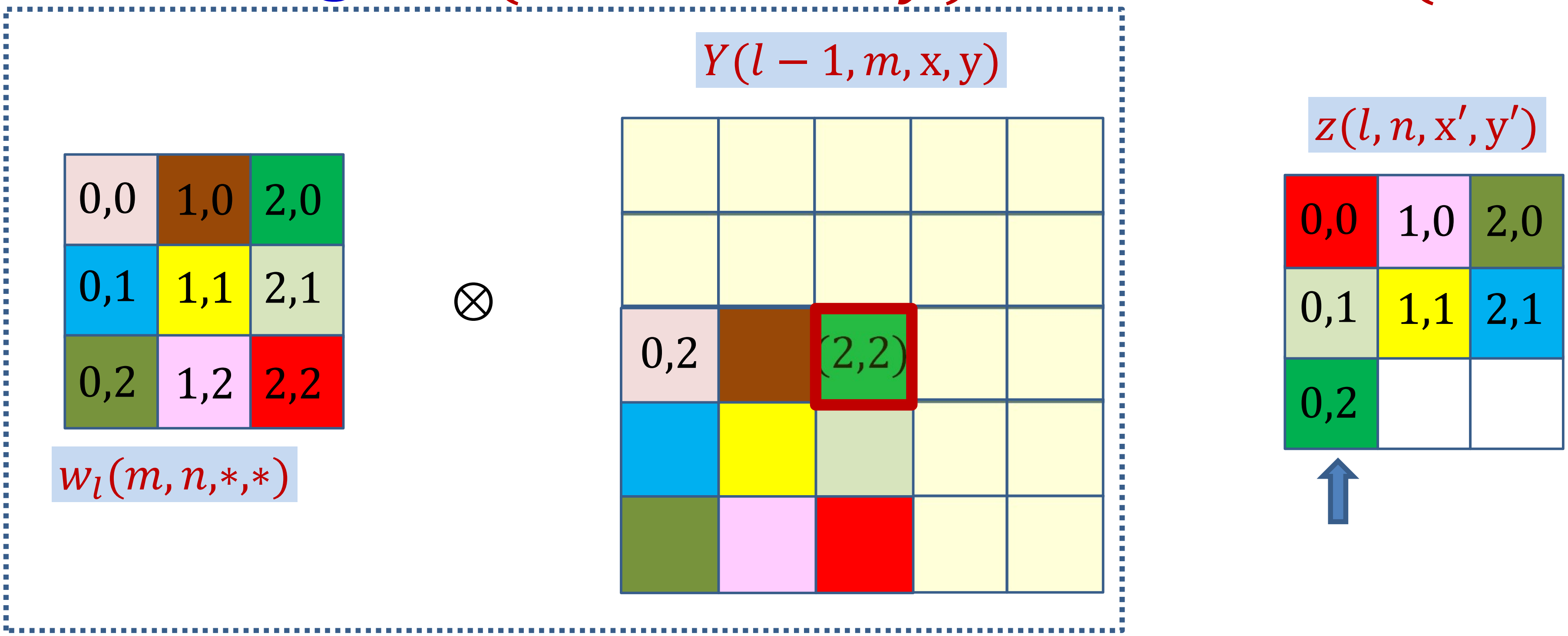
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 2,1) += Y(l-1, m, 2,2) w_l(m, n, 0,1)$$

$$\frac{dDiv}{dY(l-1, m, 2,2)} += \frac{dDiv}{dz(l, n, 2, 1)} w_l(m, n, 0,1)$$

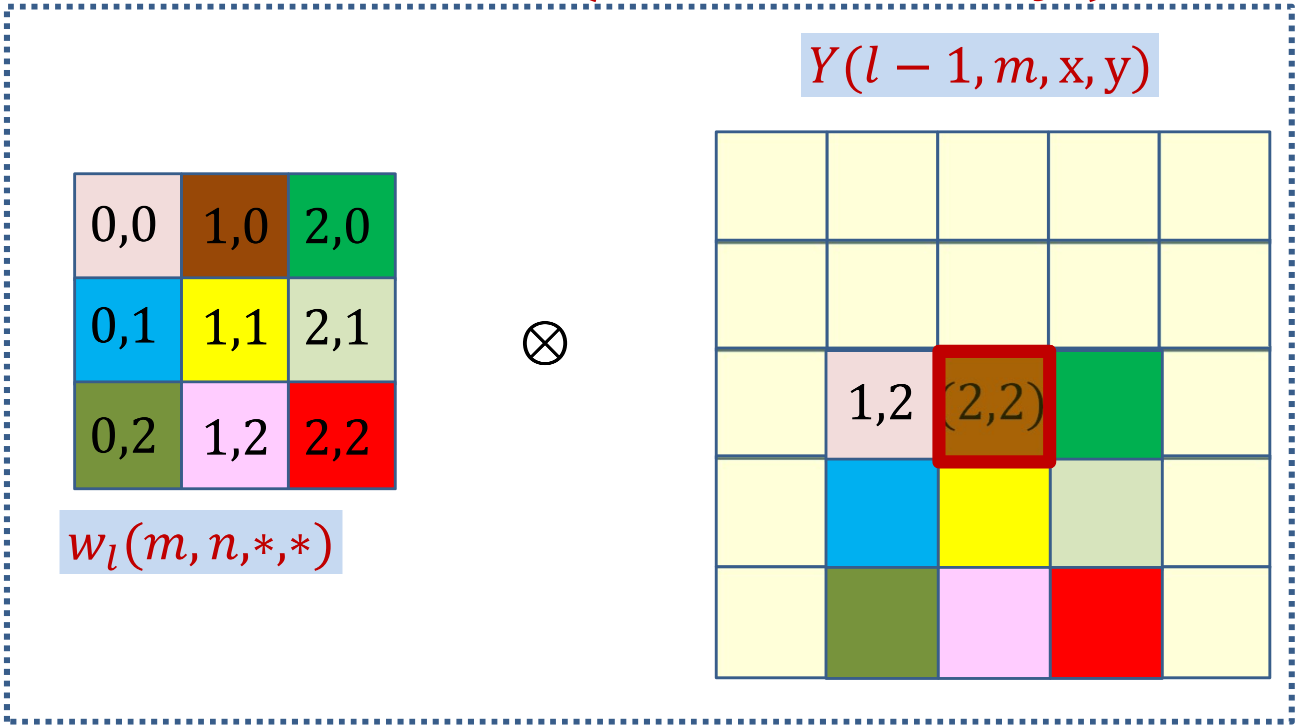
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

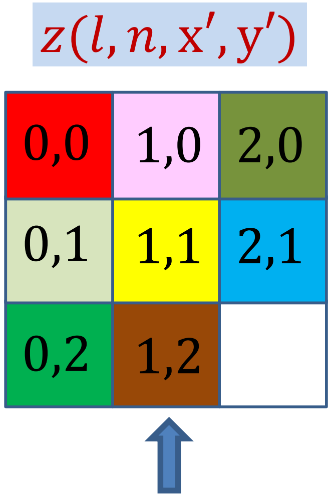


$$z(l,n,0,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,0)$$

$$\frac{dDiv}{dY(l-1,m,2,2)} \mathrel{+}= \frac{dDiv}{dz(l,n,0,2)} w_l(m,n,2,0)$$

How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$

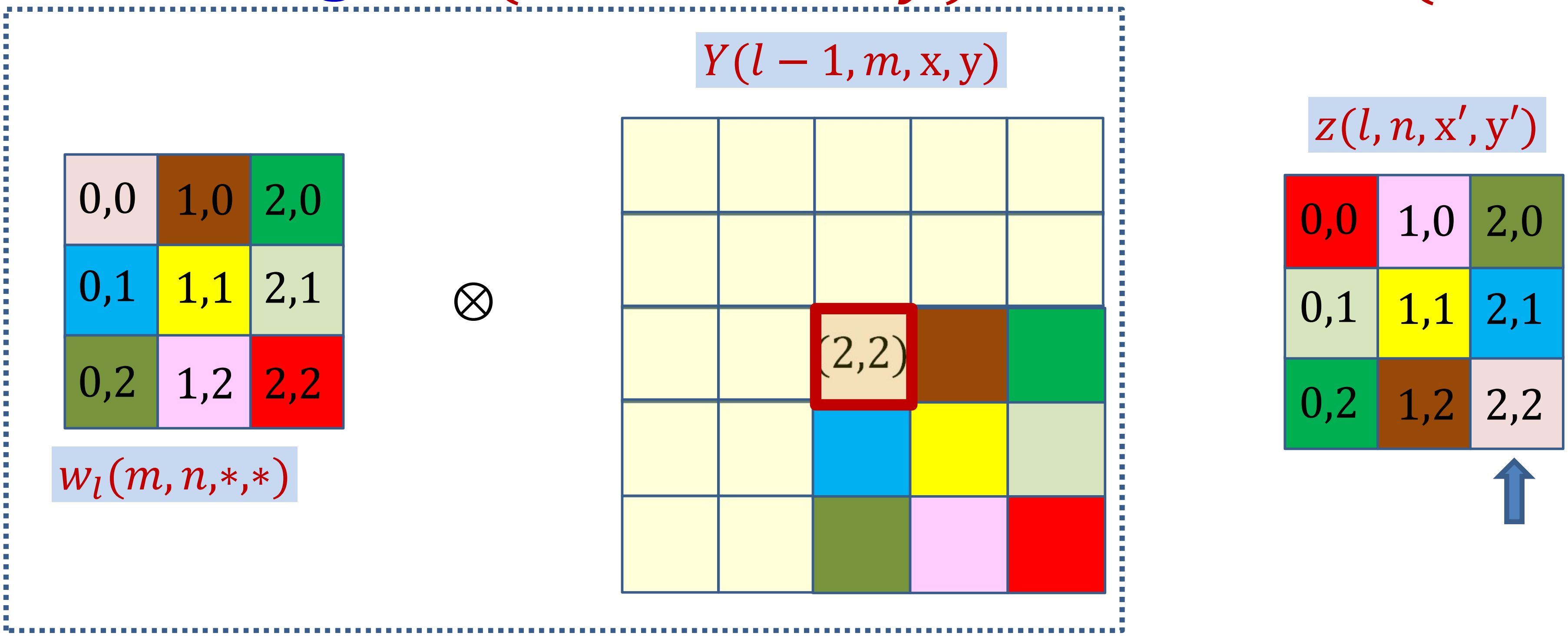




$$z(l,n,1,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,1)$$

$$\frac{dDiv}{dY(l-1,m,2,2)} \mathrel{+}= \frac{dDiv}{dz(l,n,1,2)}w_l(m,n,1,0)$$

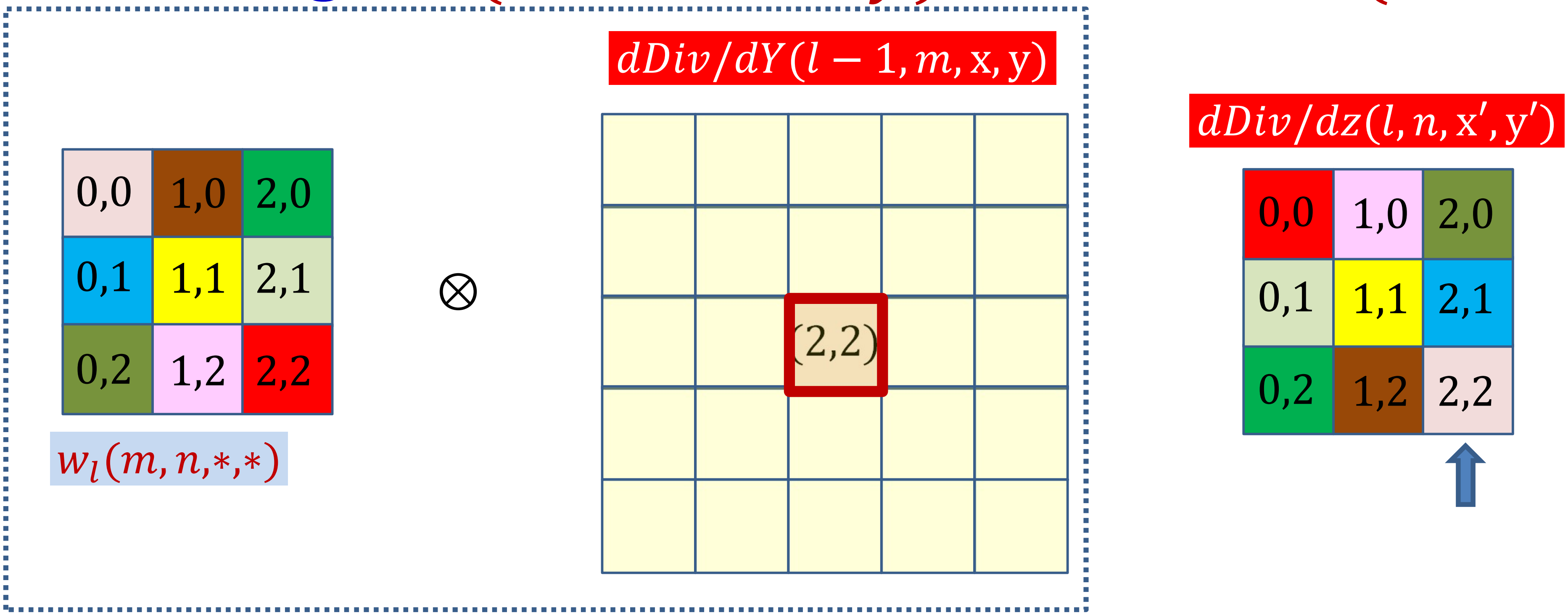
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$z(l,n,2,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,0)$$

$$\frac{dDiv}{dY(l-1,m,2,2)} \mathrel{+}= \frac{dDiv}{dz(l,n,2,2)} w_l(m,n,0,0)$$

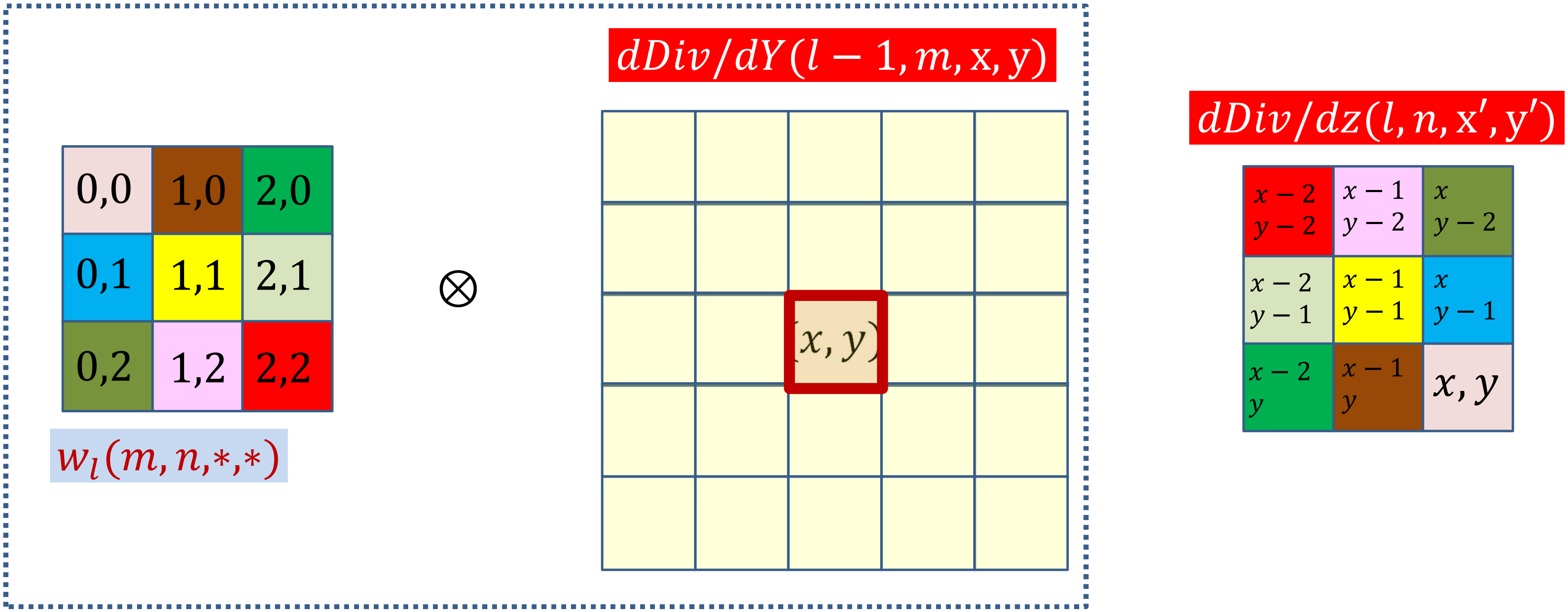
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



$$\frac{dDiv}{dY(l-1,m,2,2)} = \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,2-x',2-y')$$

- The derivative at $Y(l-1,m,2,2)$ is the sum of component-wise product of the filter elements (shown by color) and the elements of the derivative at $z(l,m,.,.)$

Derivative at $Y(l-1, m, x, y)$ from a single $Z(l, n)$ map

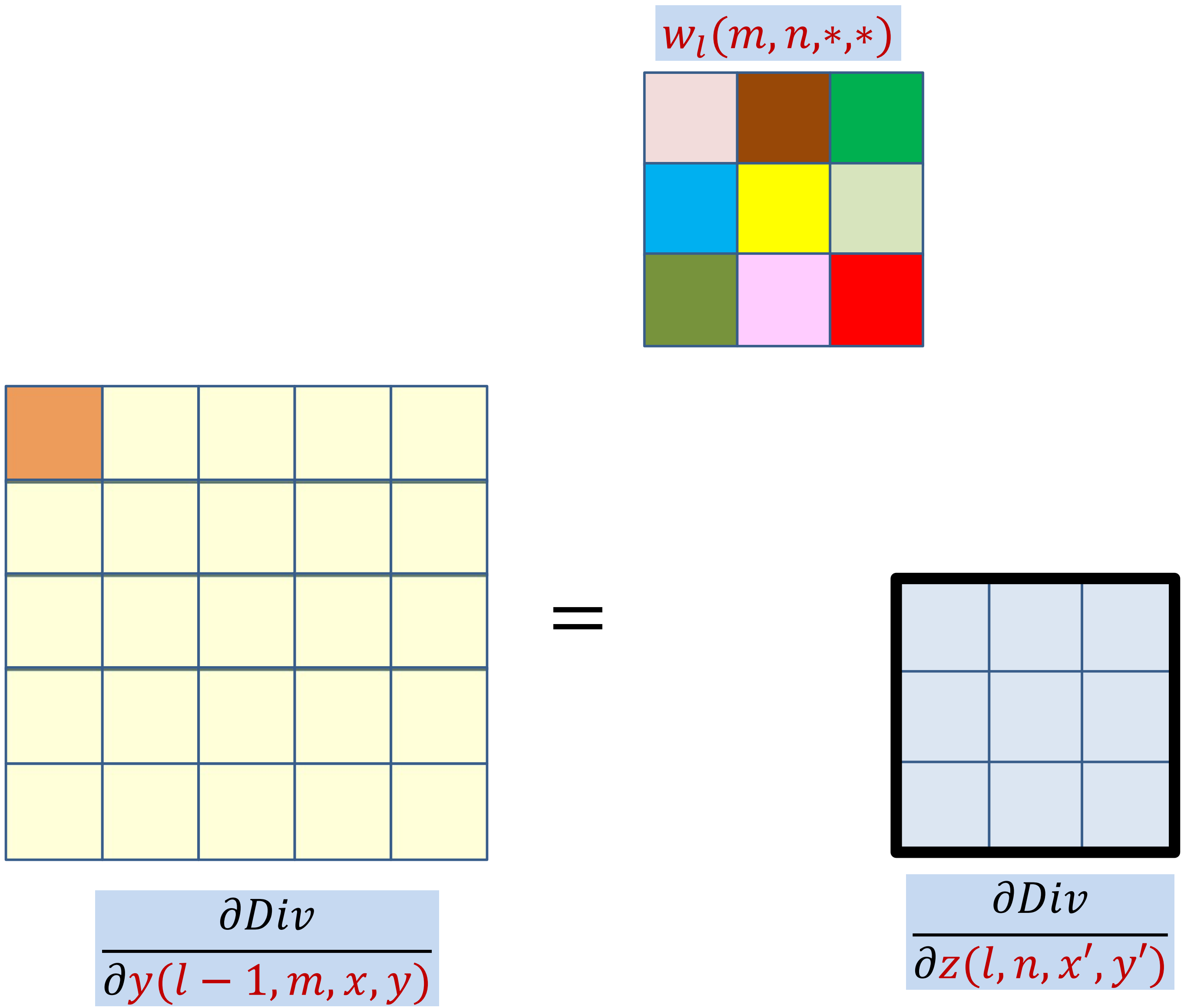


$$z(l, n, x', y') \mathrel{+}= Y(l-1, m, x, y)\, w_l(m, n, x-x', y-y')$$

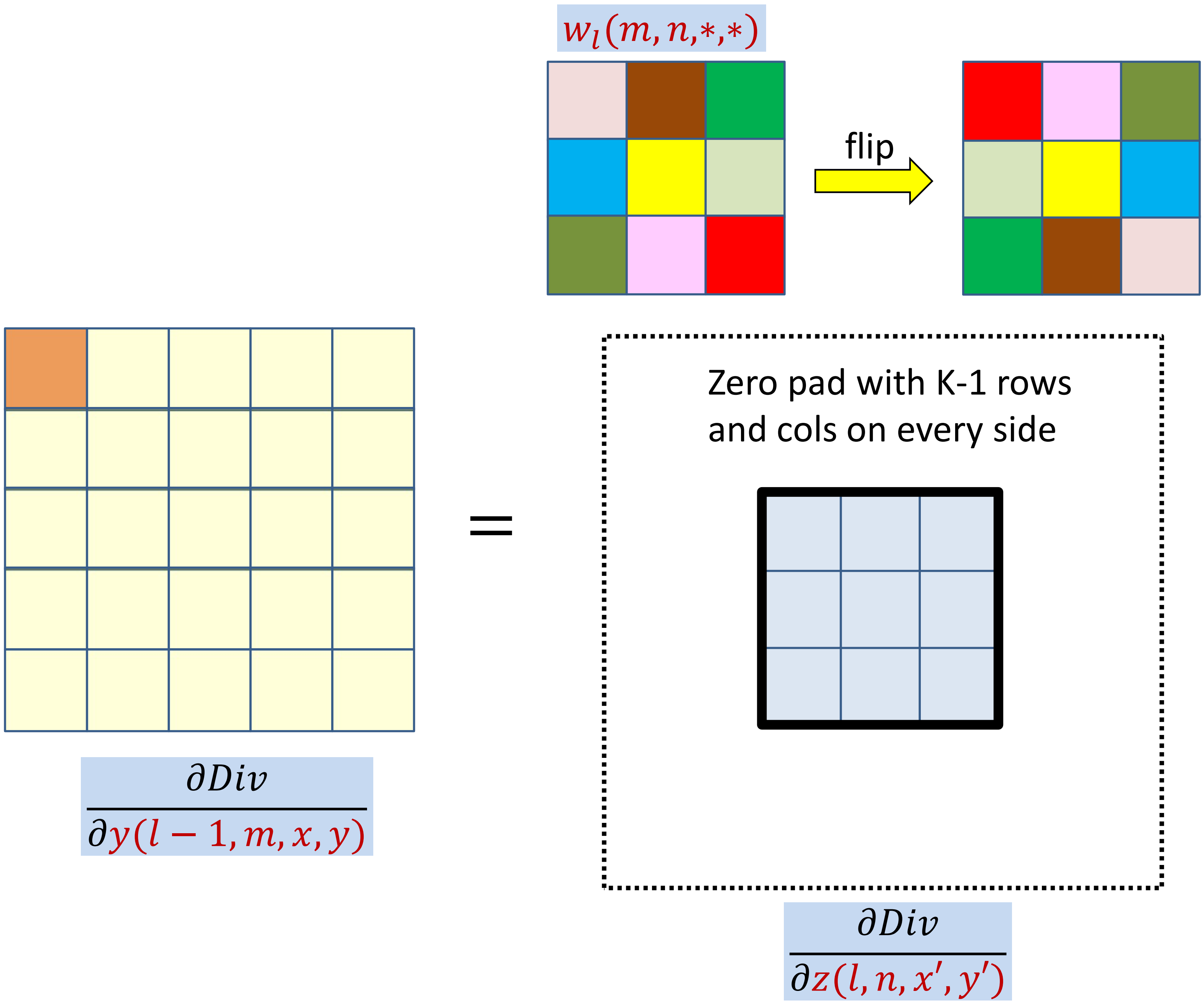
$$\frac{dDiv}{dY(l-1, m, x, y)} \mathrel{+}= \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x-x', x-y')$$

Contribution of the entire nth affine map $z(l, n, *, *)$

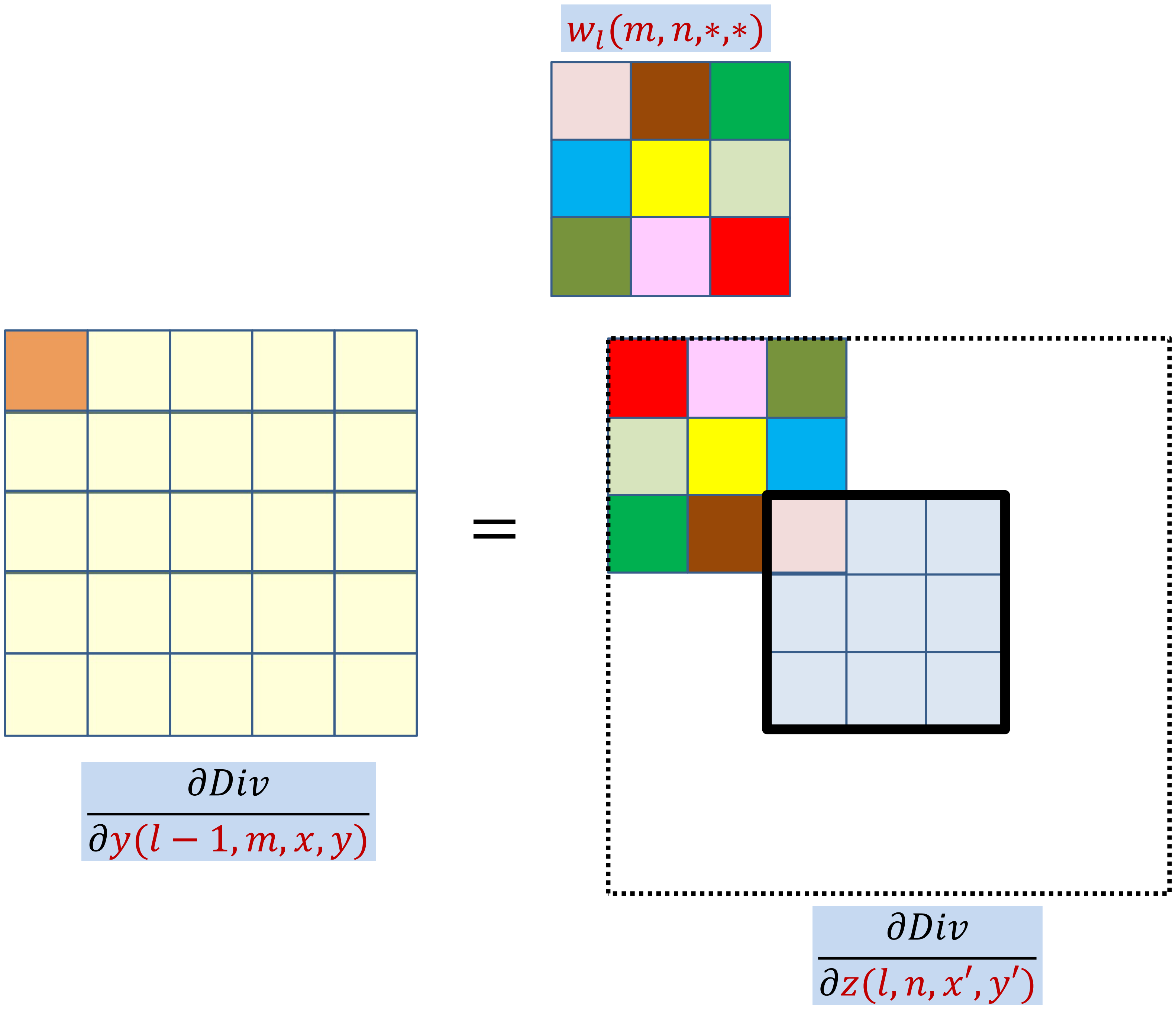
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



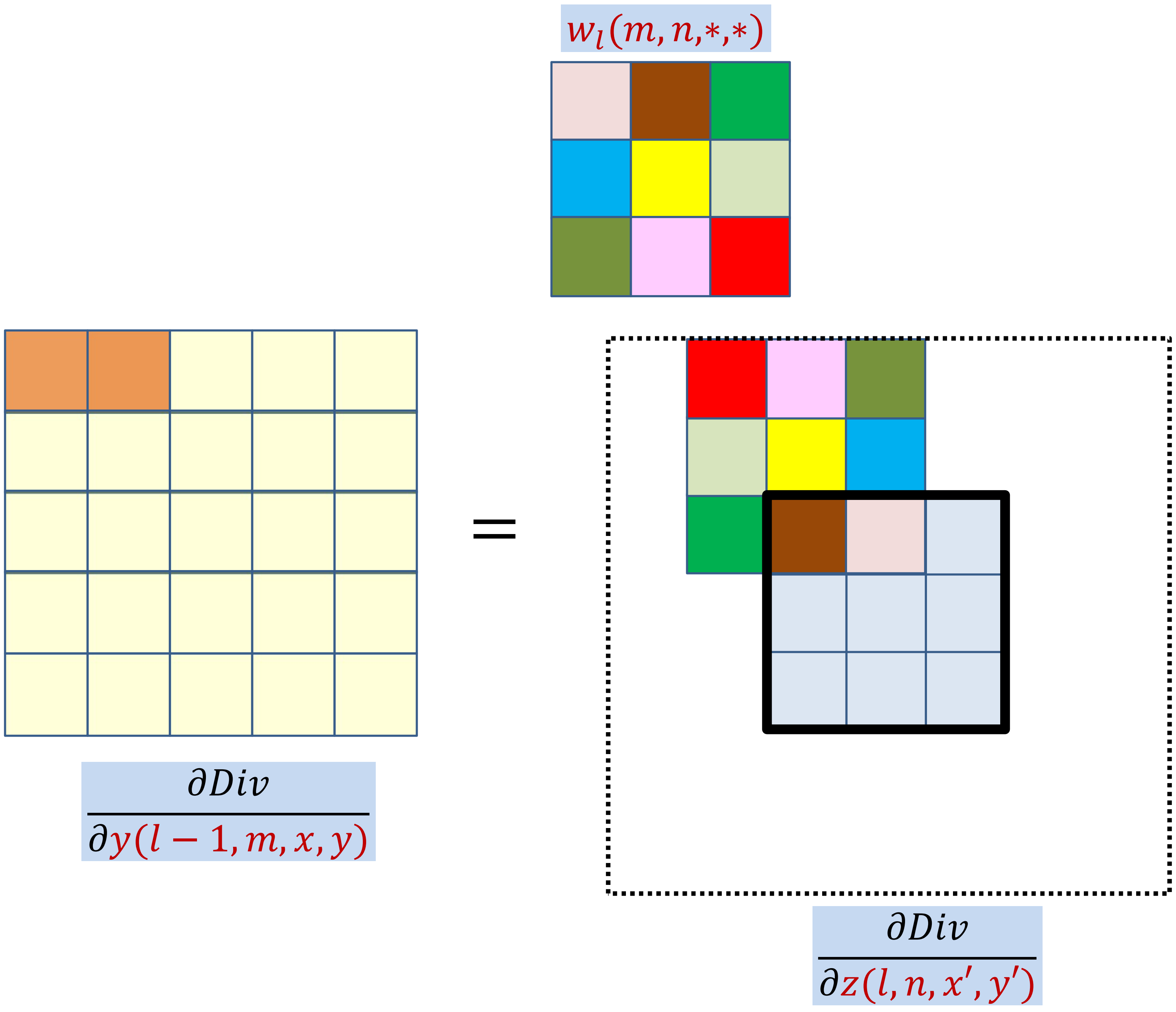
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



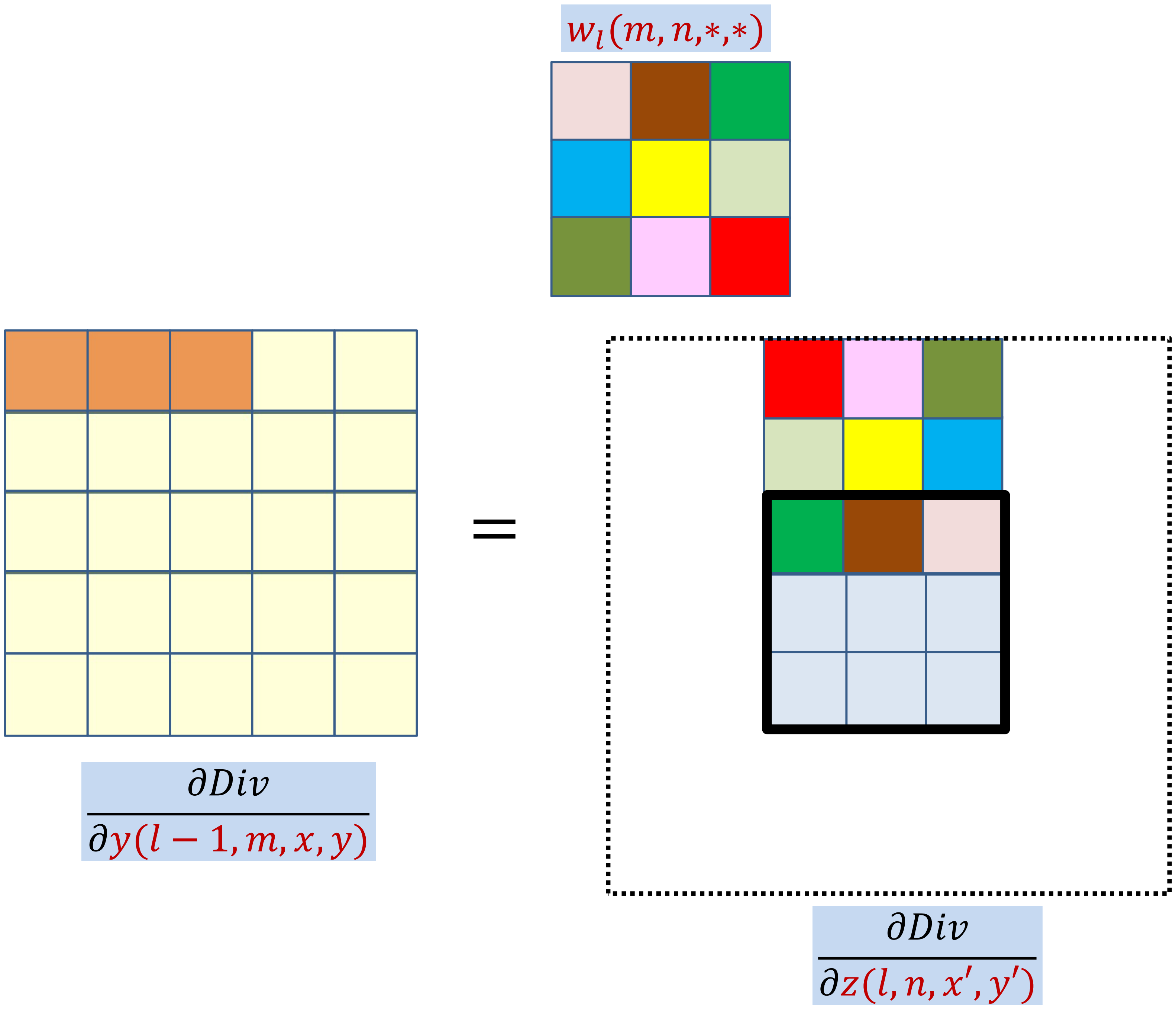
Derivative at $Y(l-1, m)$ from a single $Z(l, n)$ map



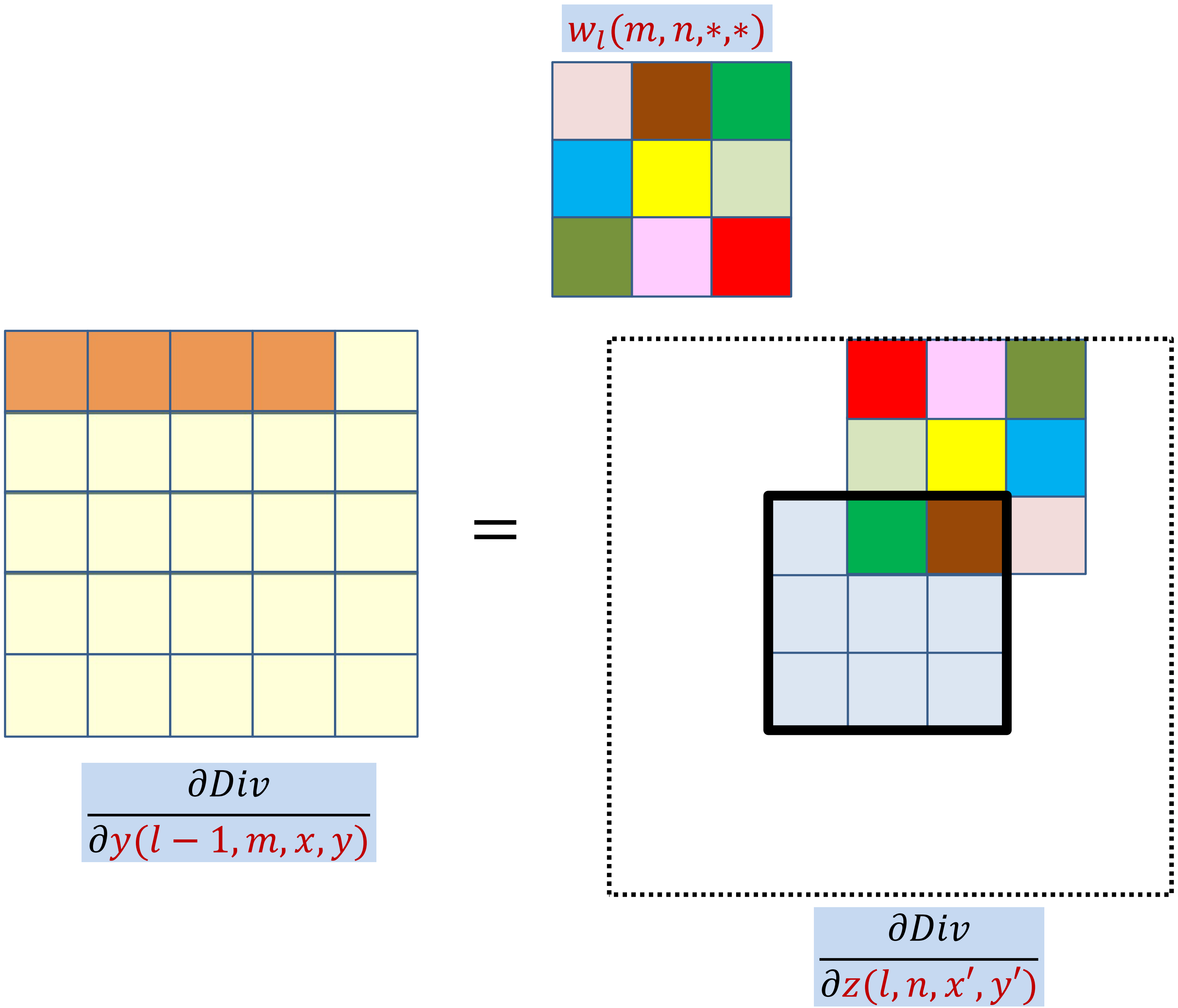
Derivative at $Y(l-1, m)$ from a single $Z(l, n)$ map



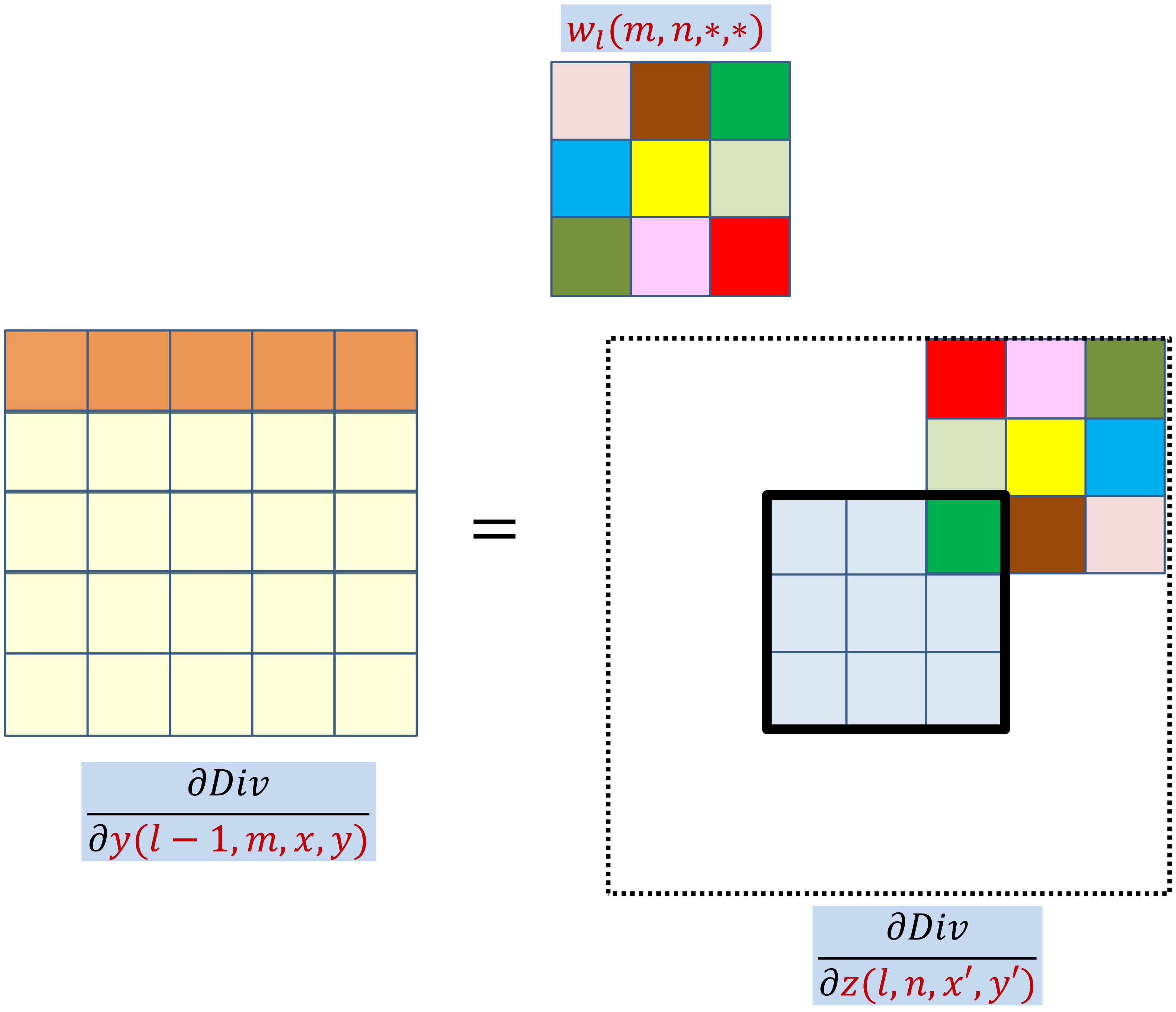
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



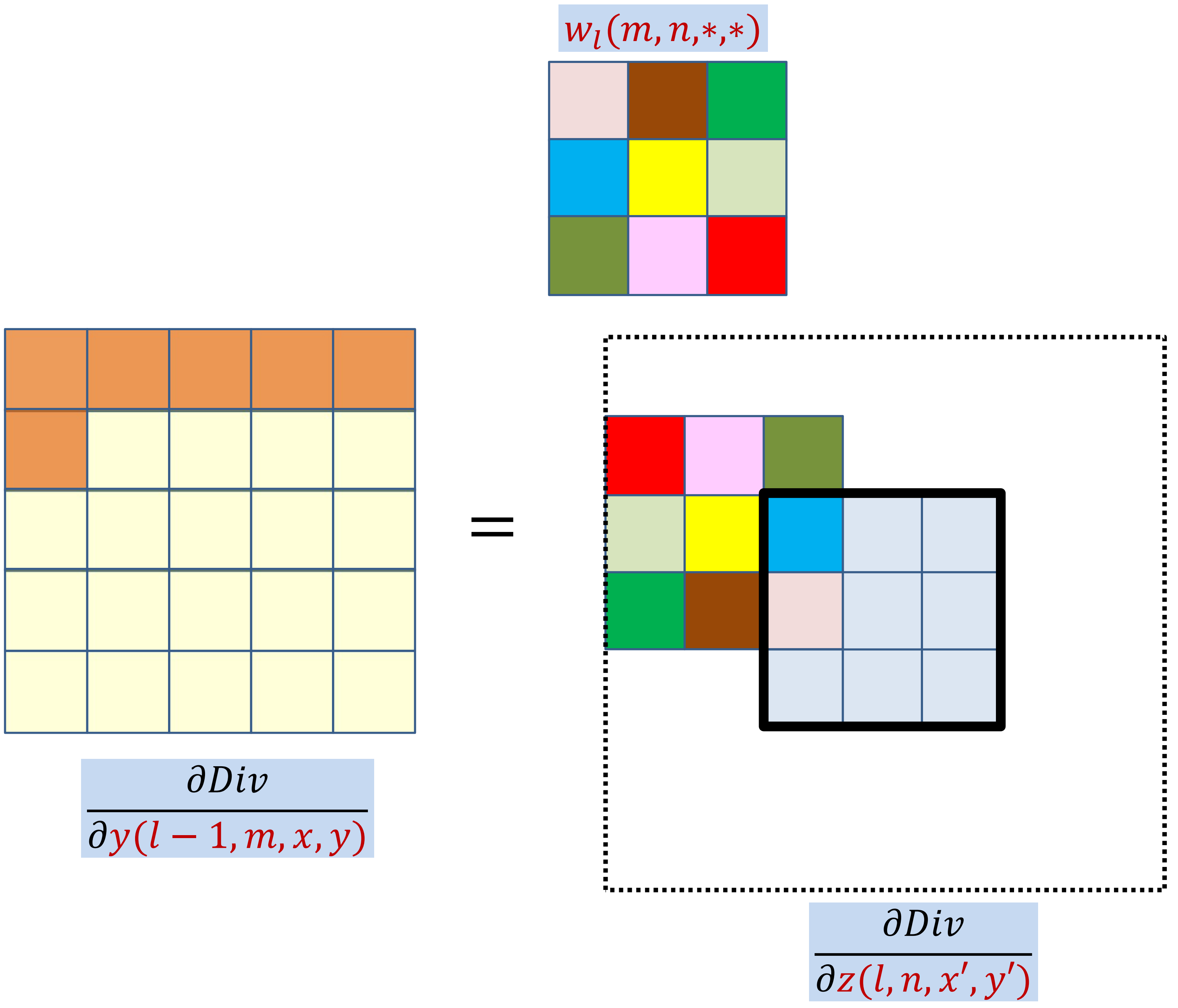
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



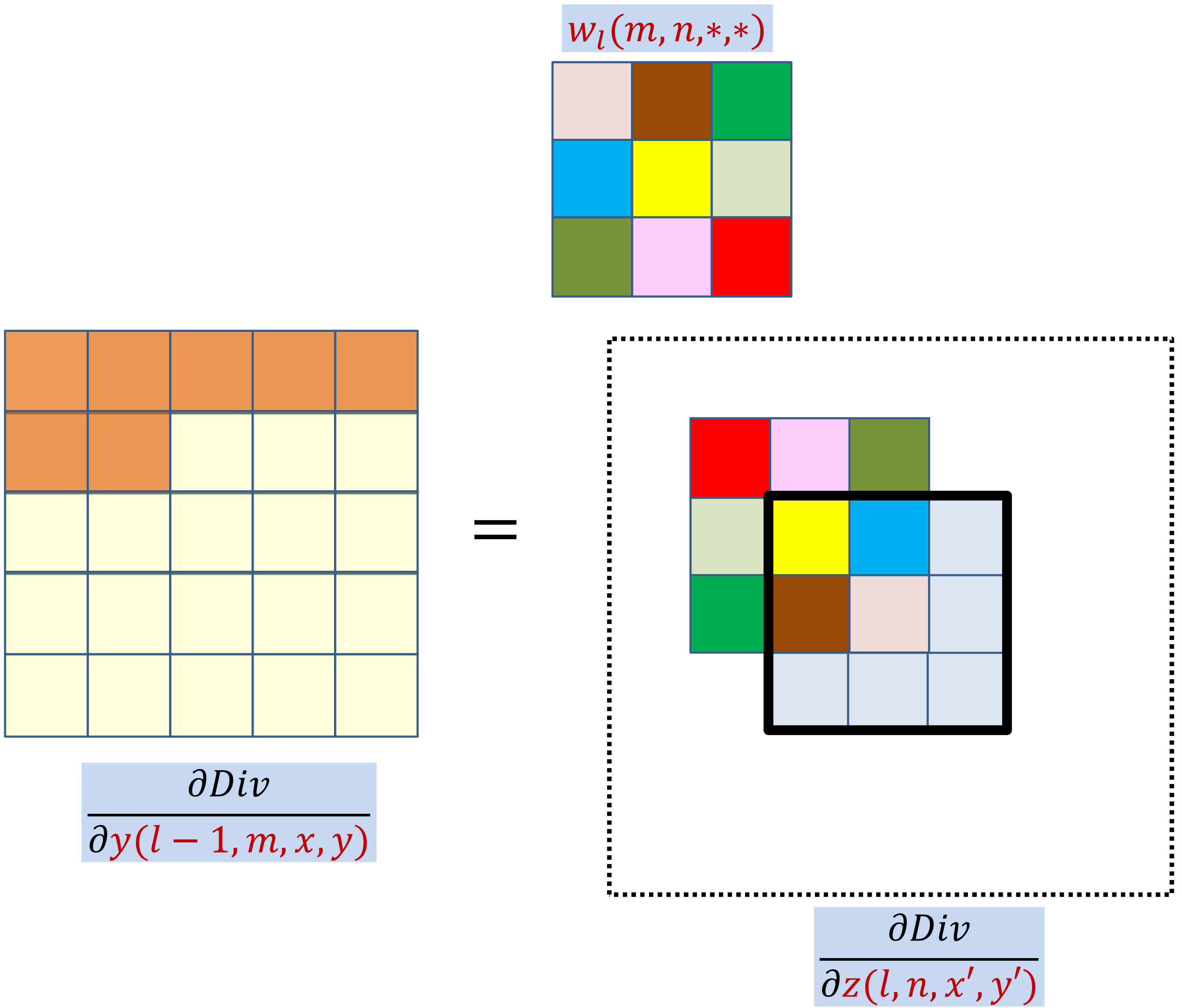
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



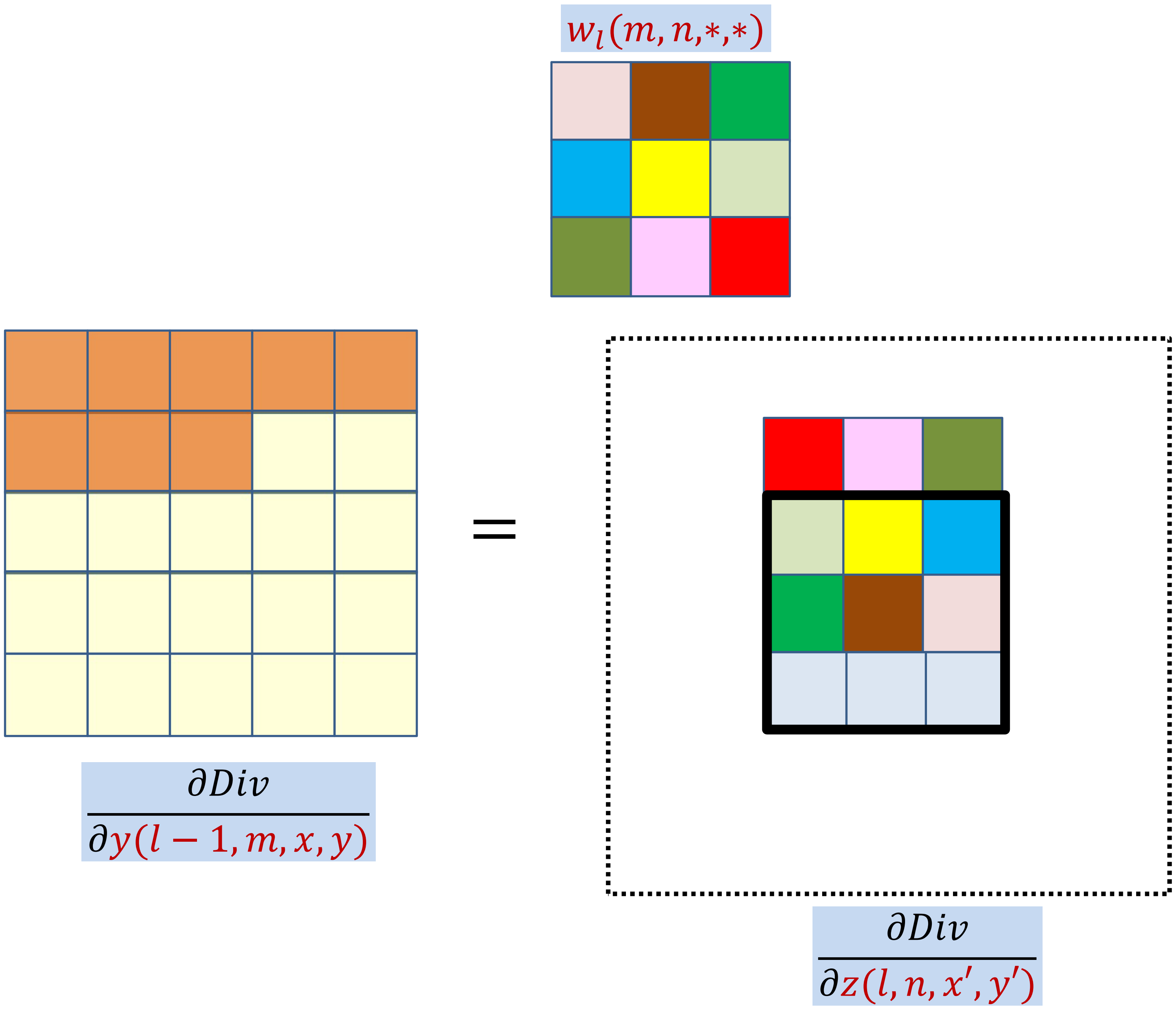
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



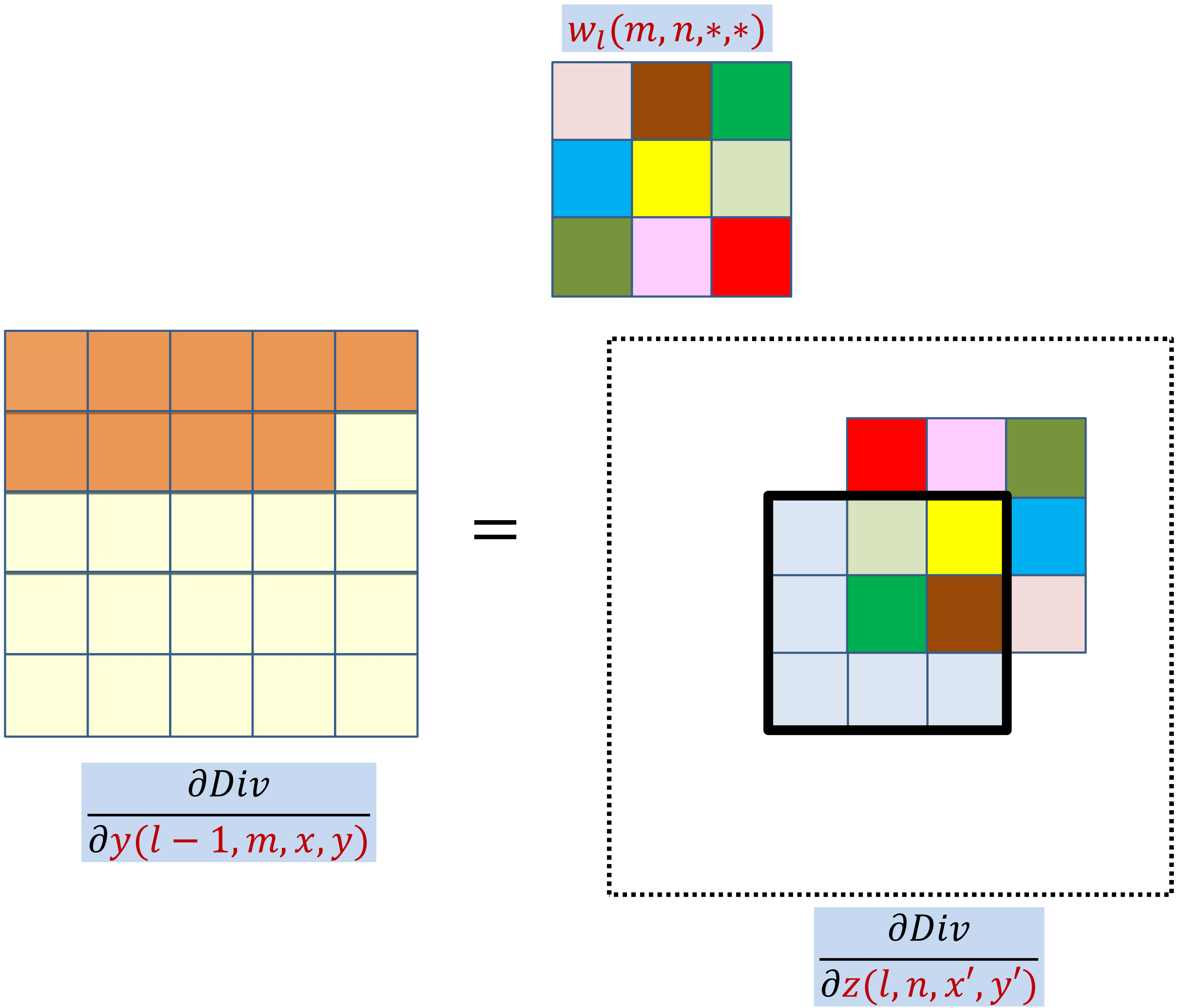
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



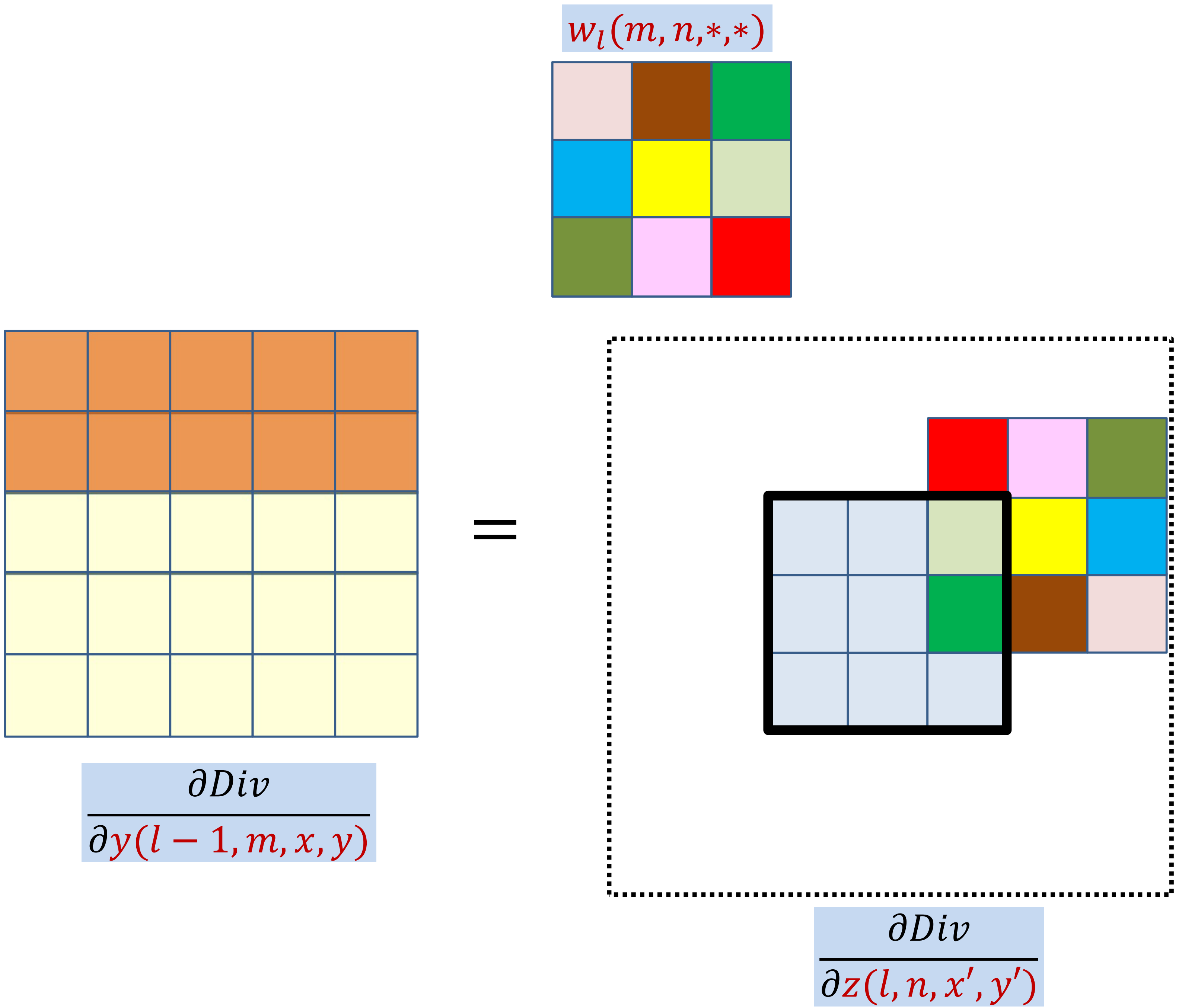
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



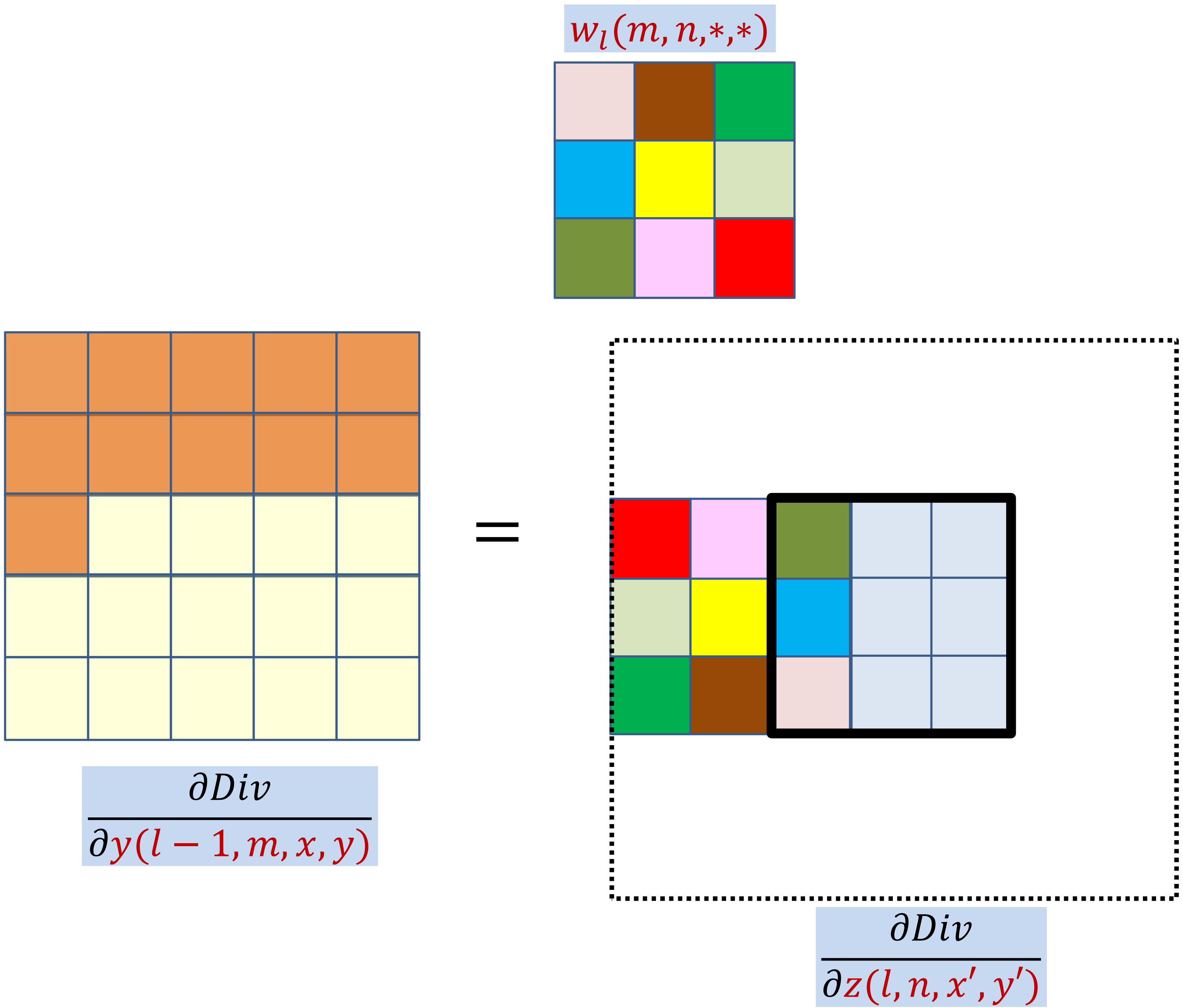
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



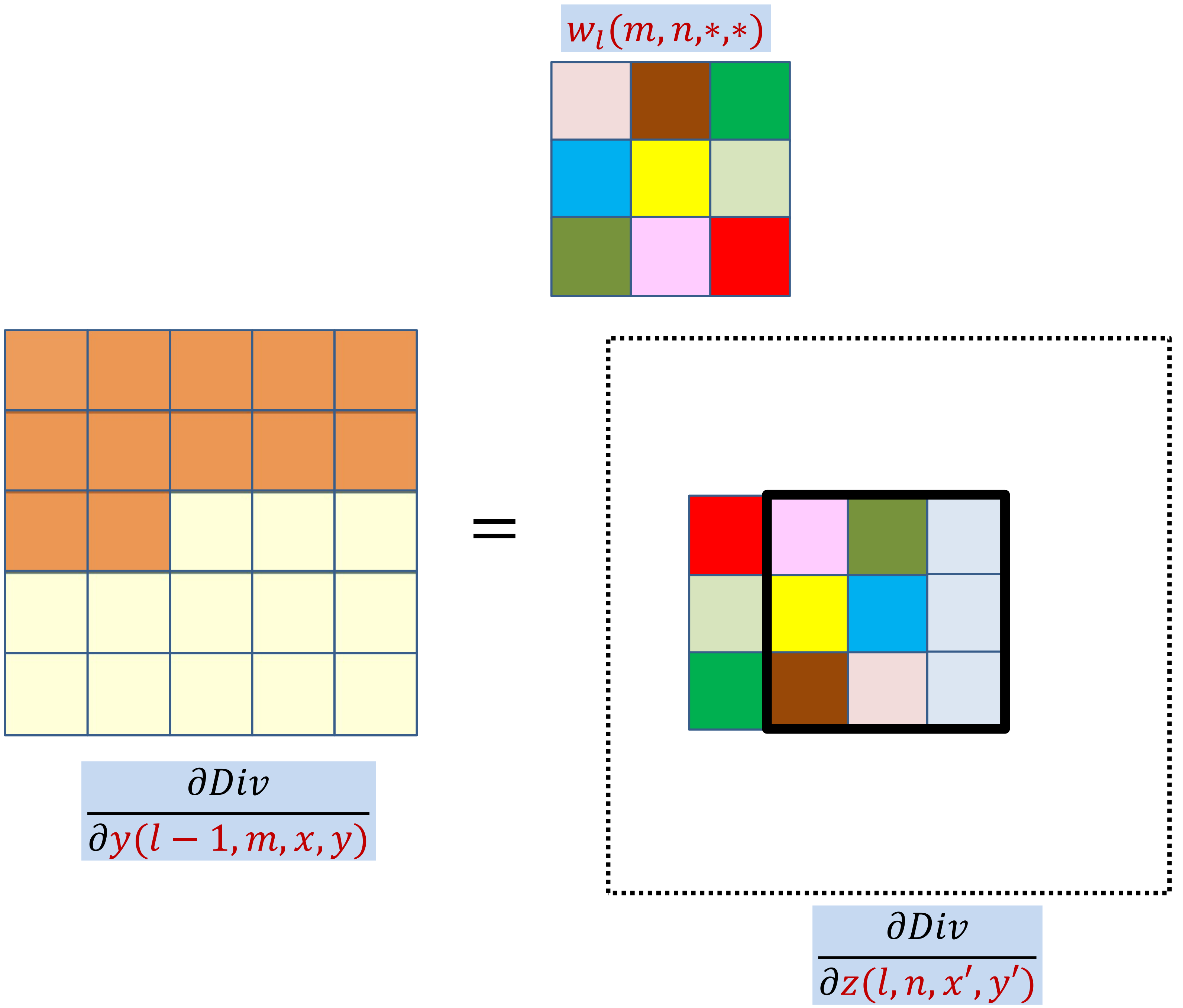
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



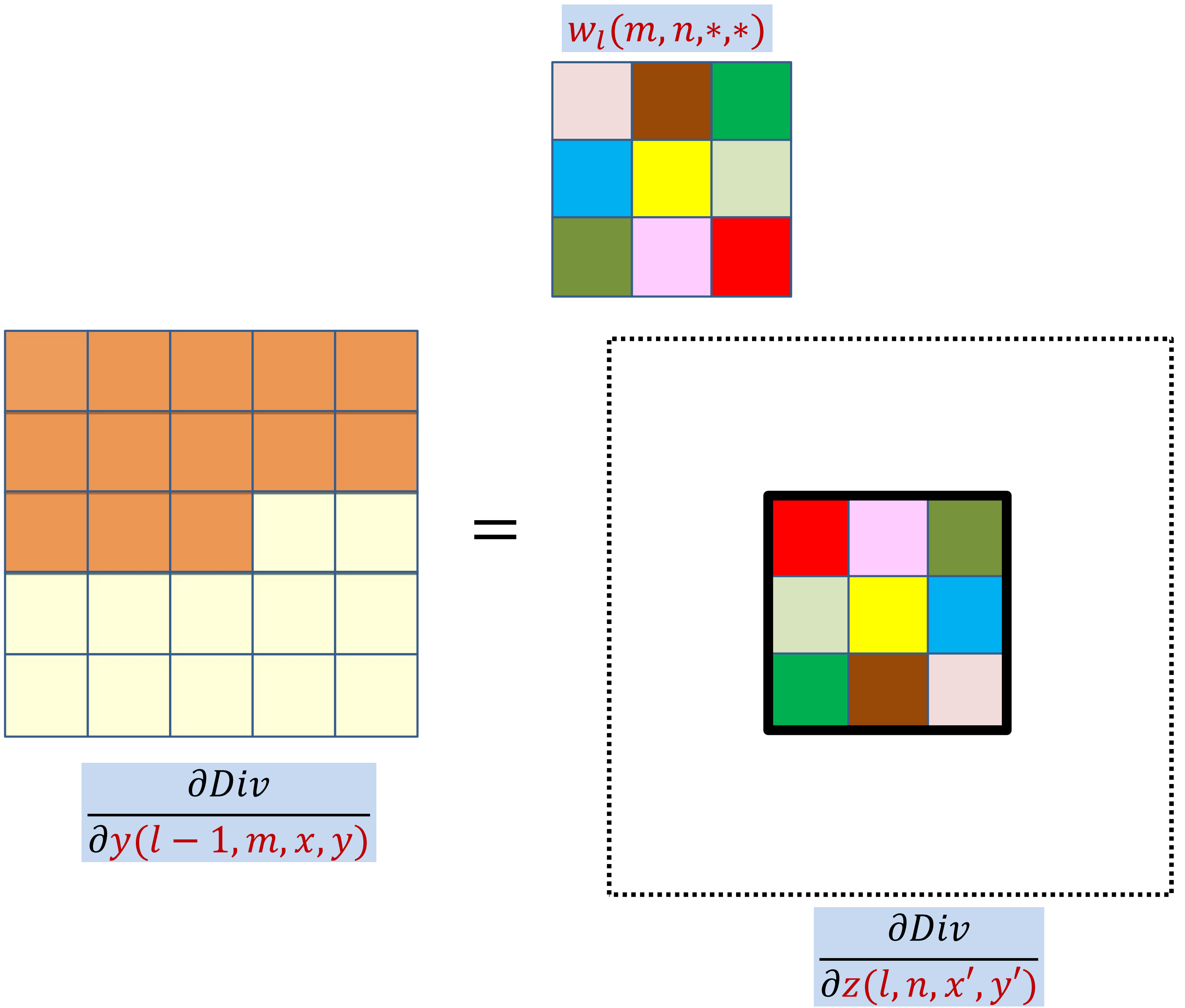
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



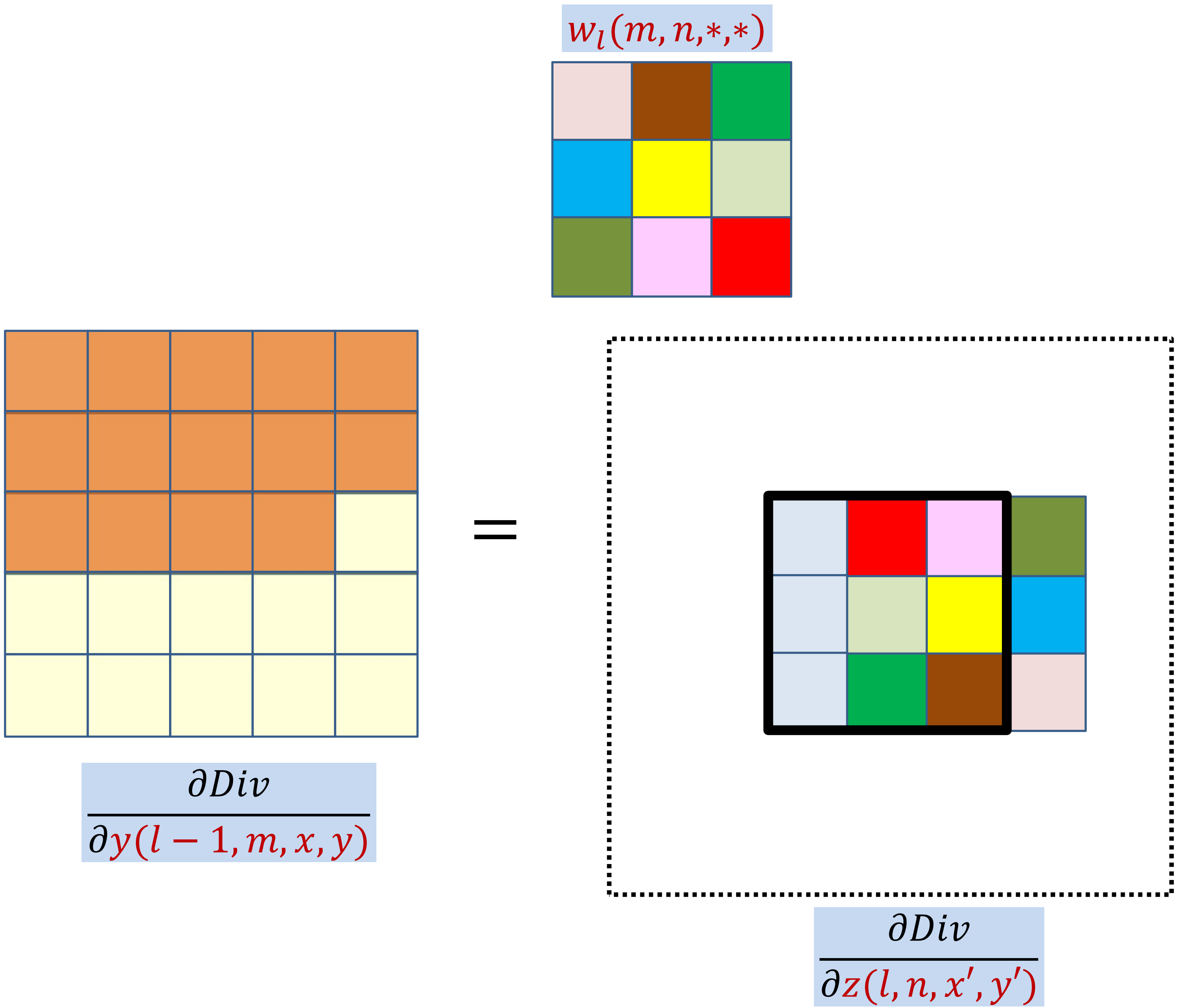
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



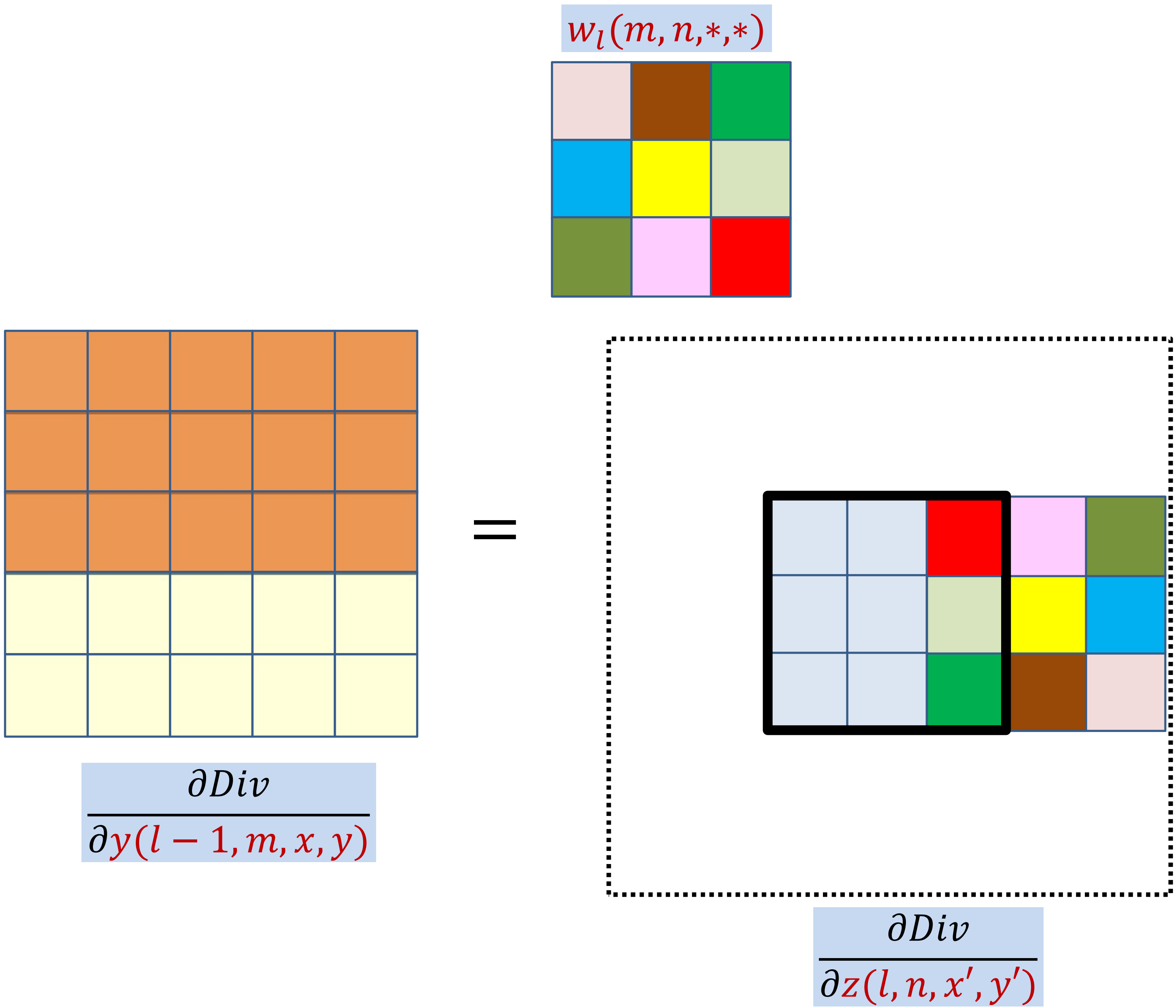
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



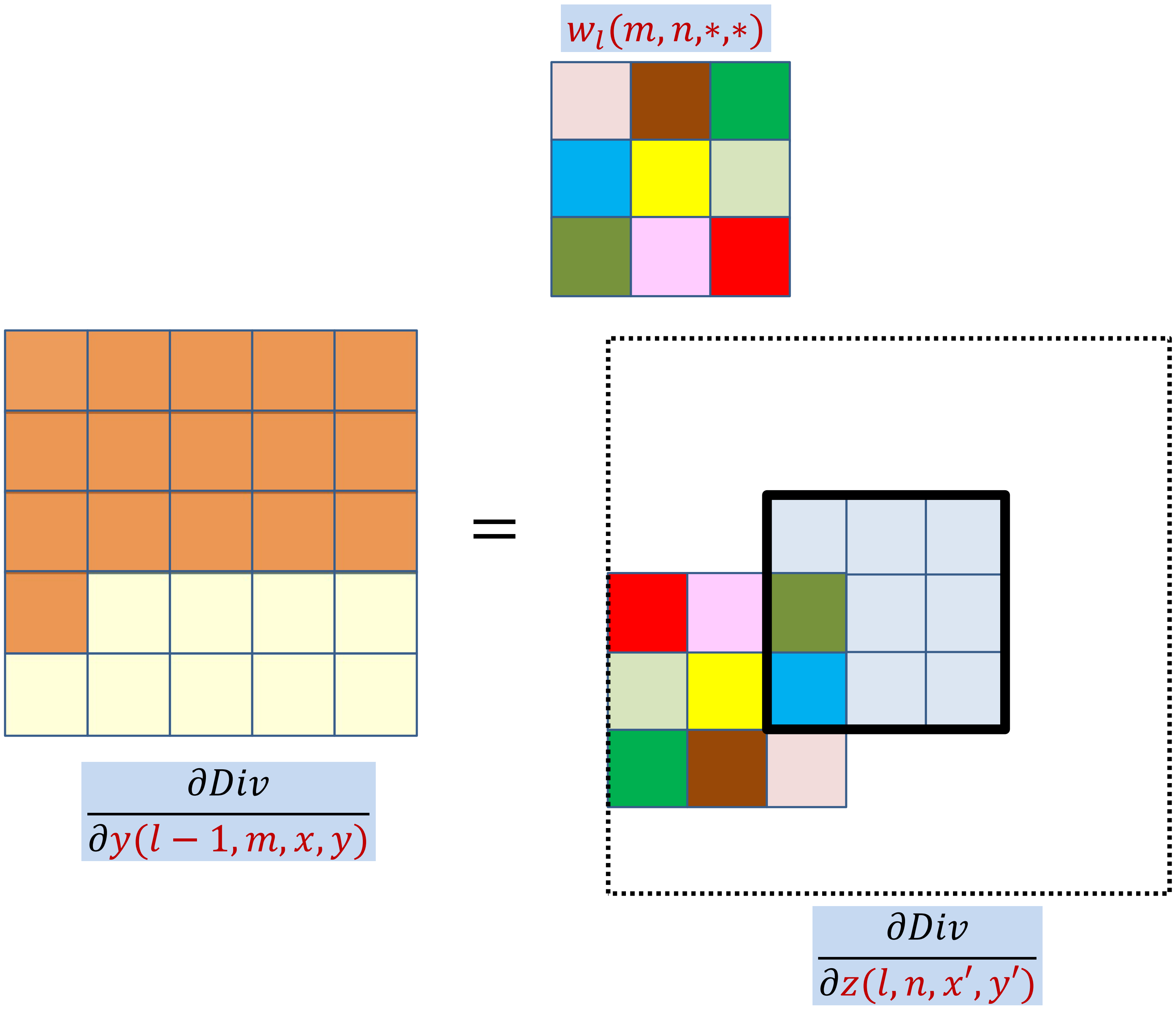
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



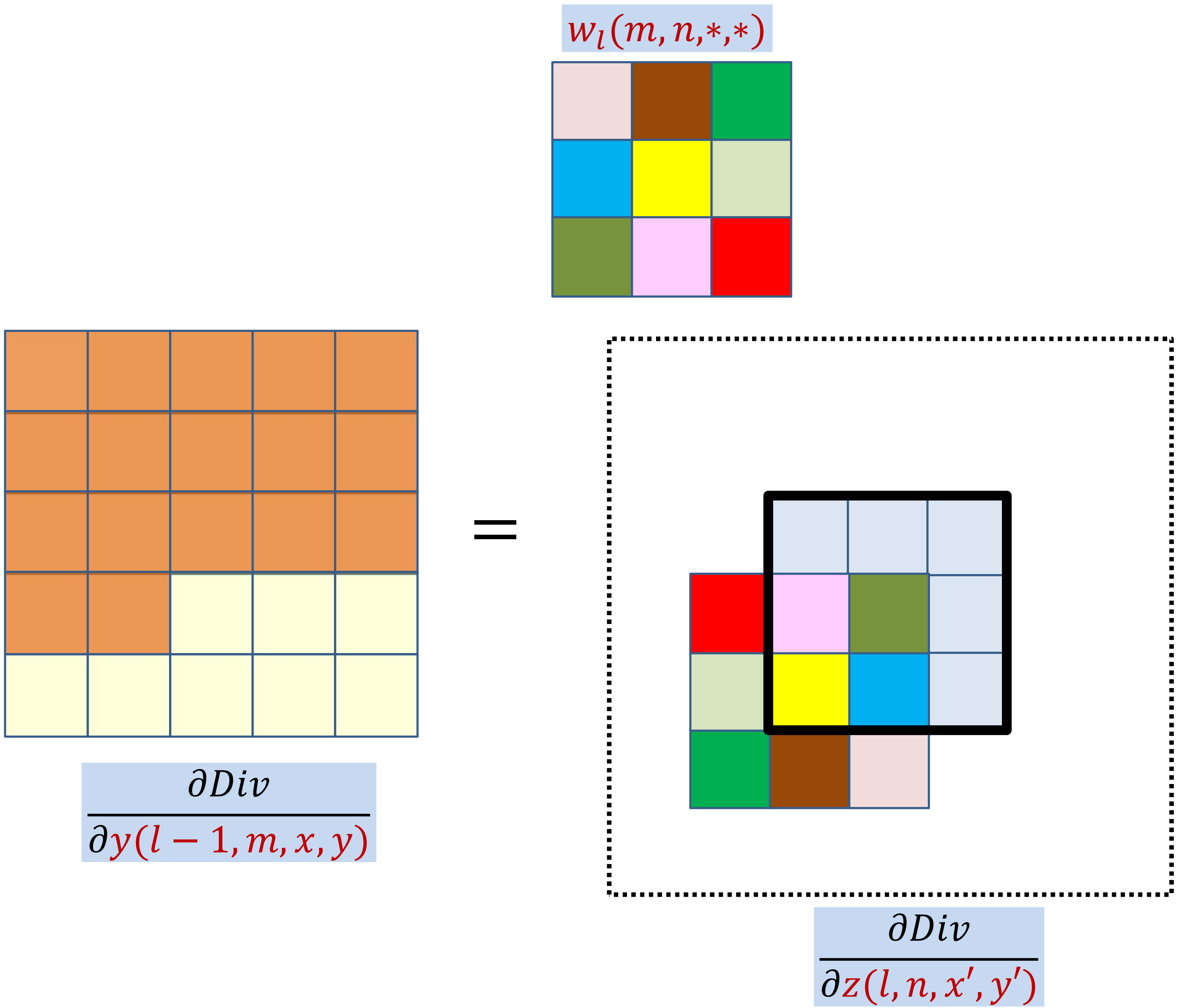
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



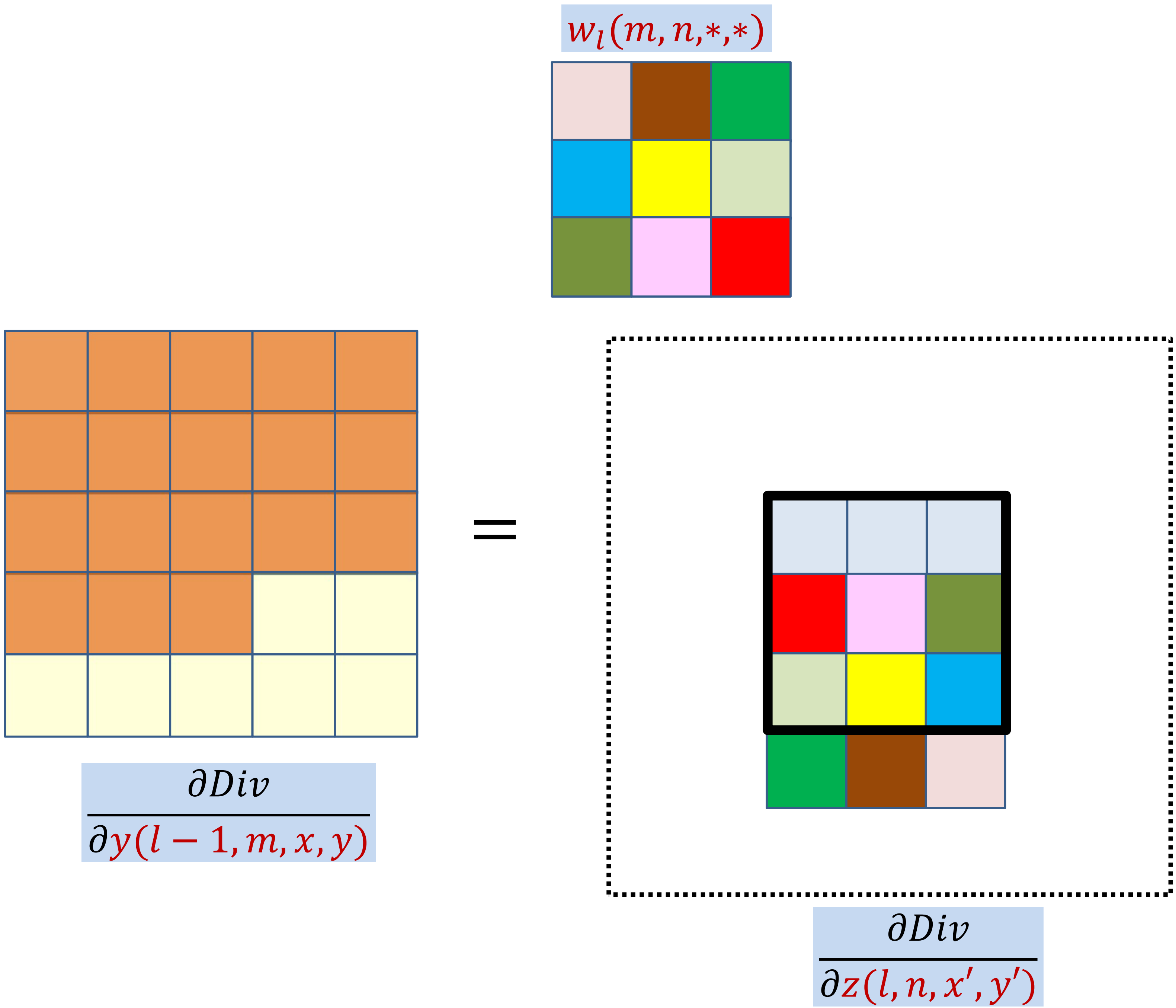
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



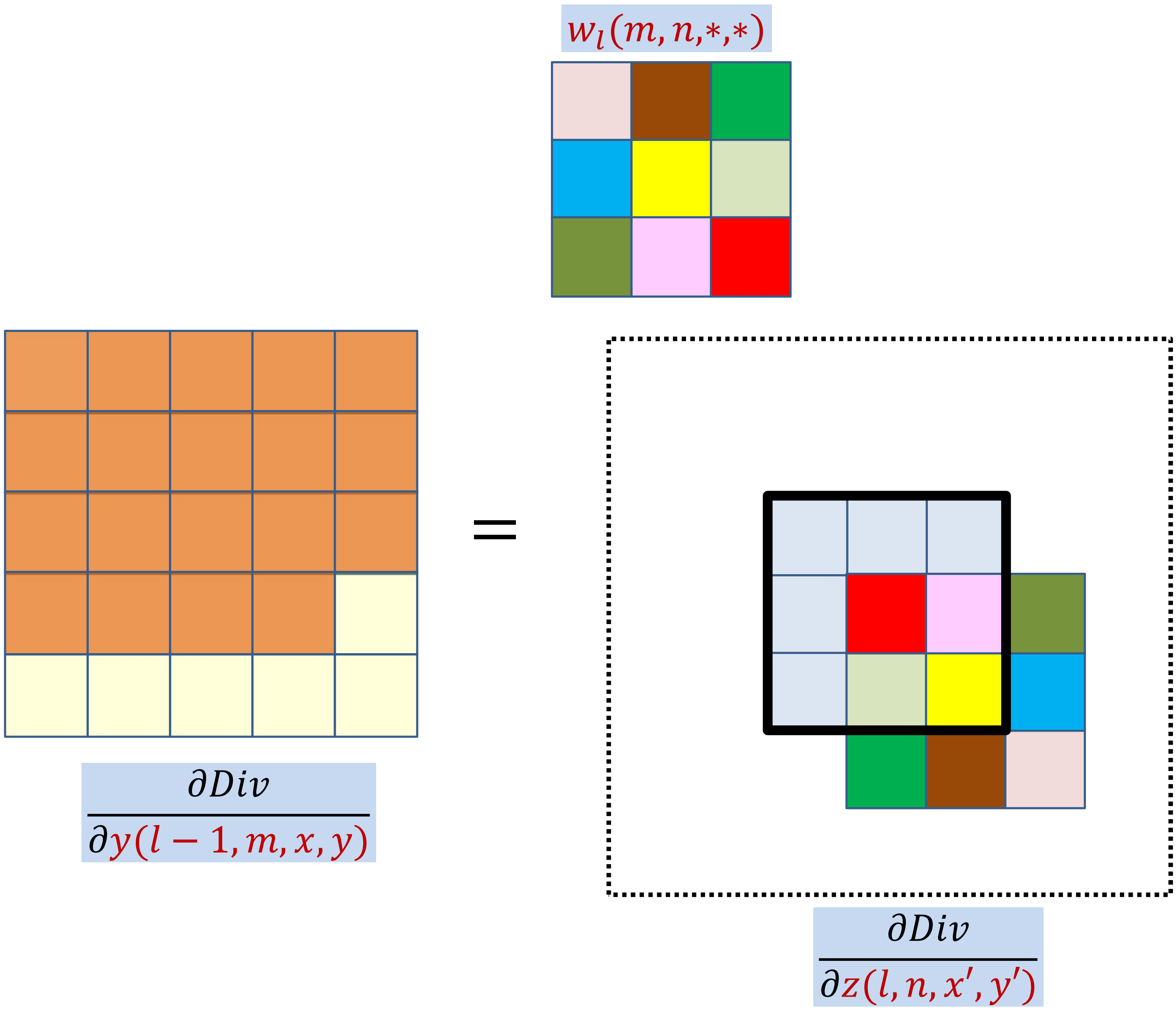
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



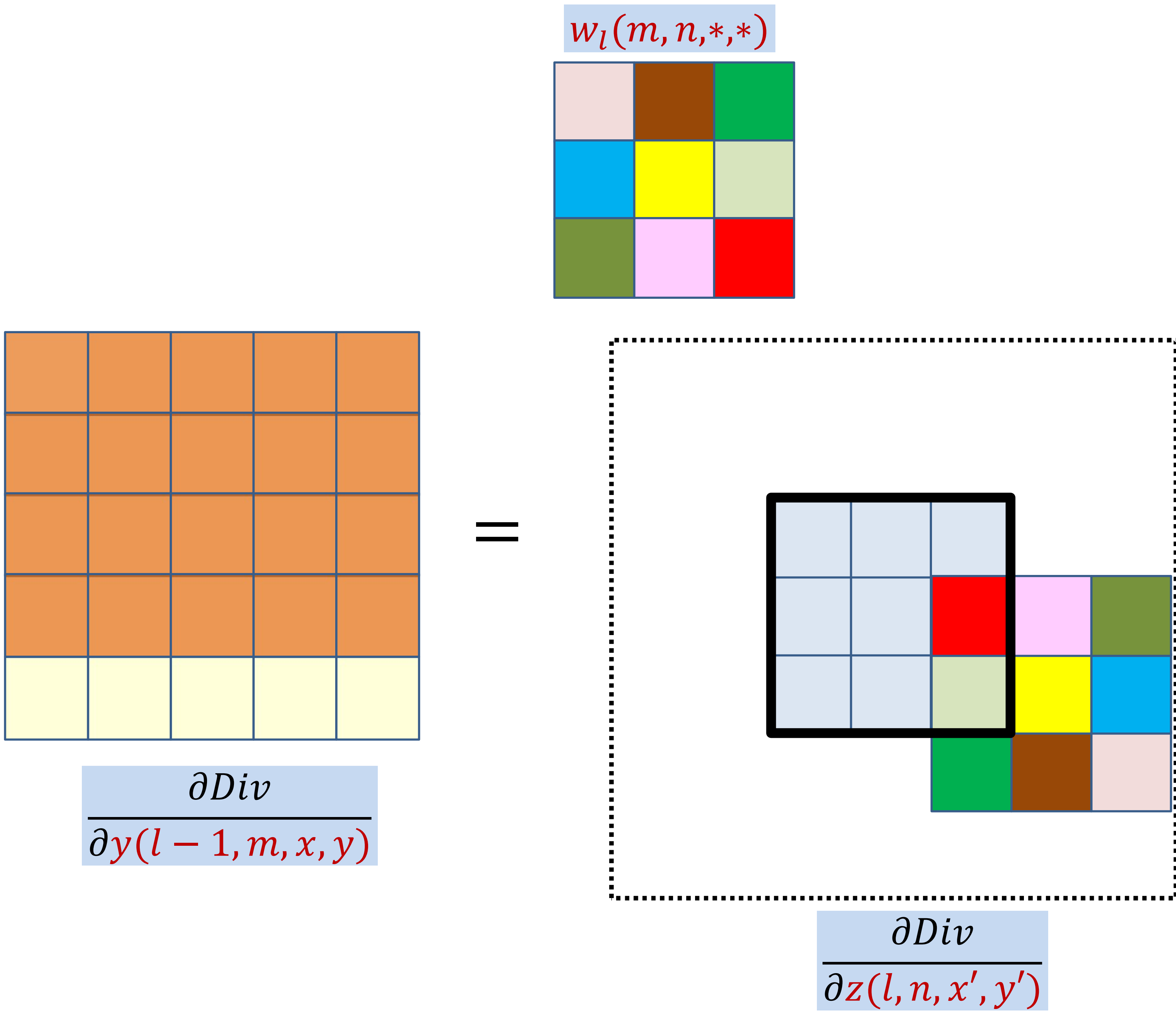
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



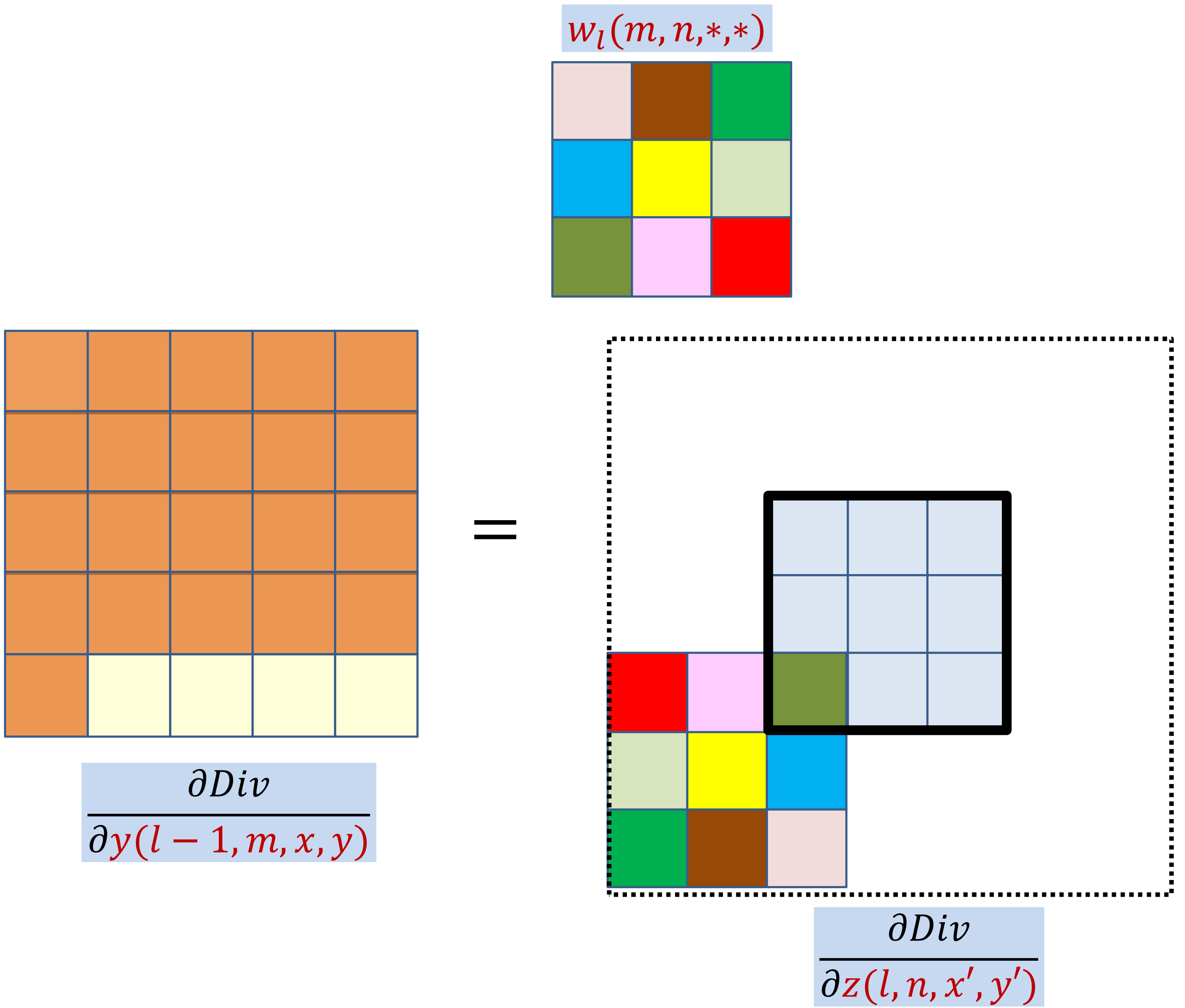
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



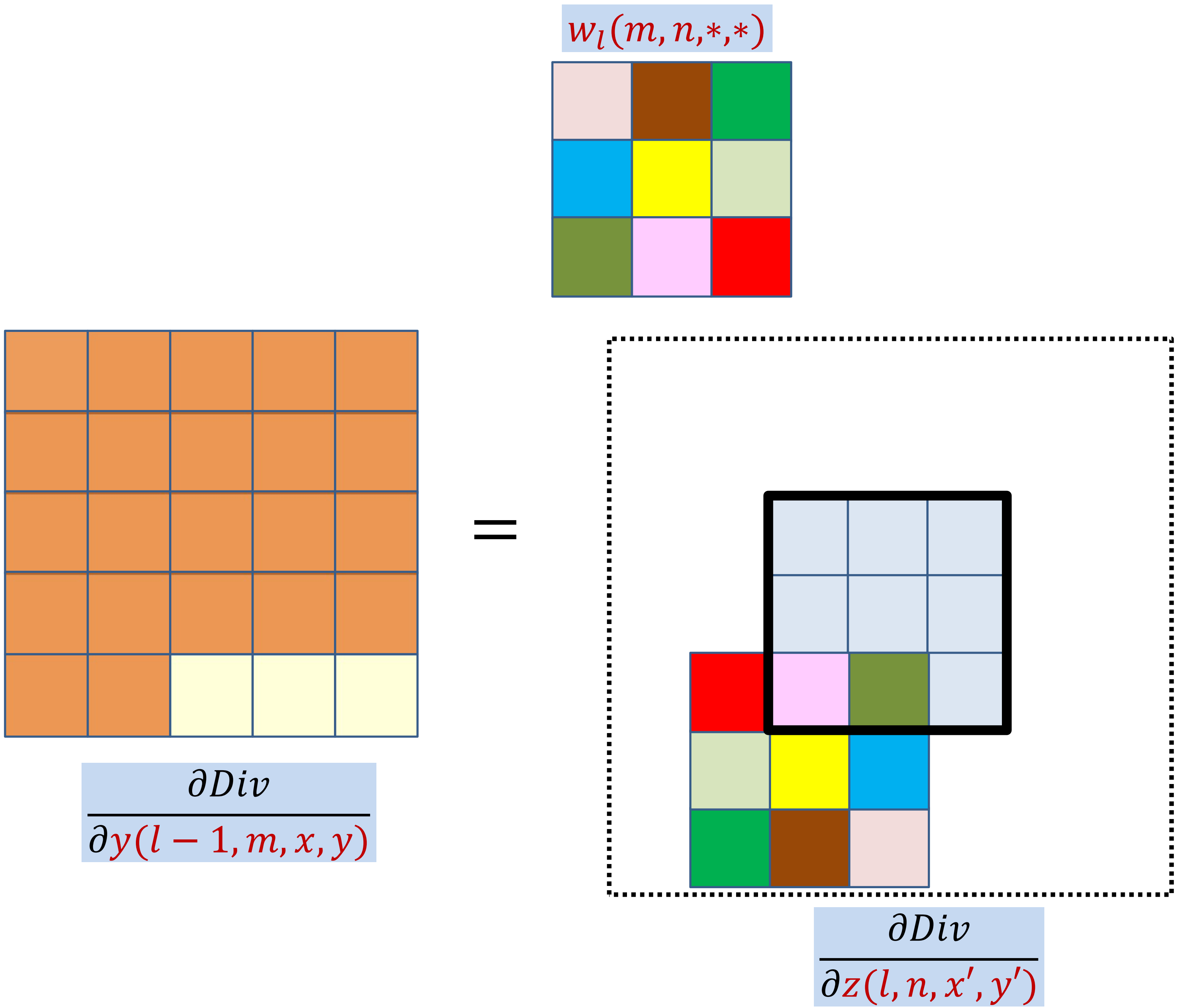
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



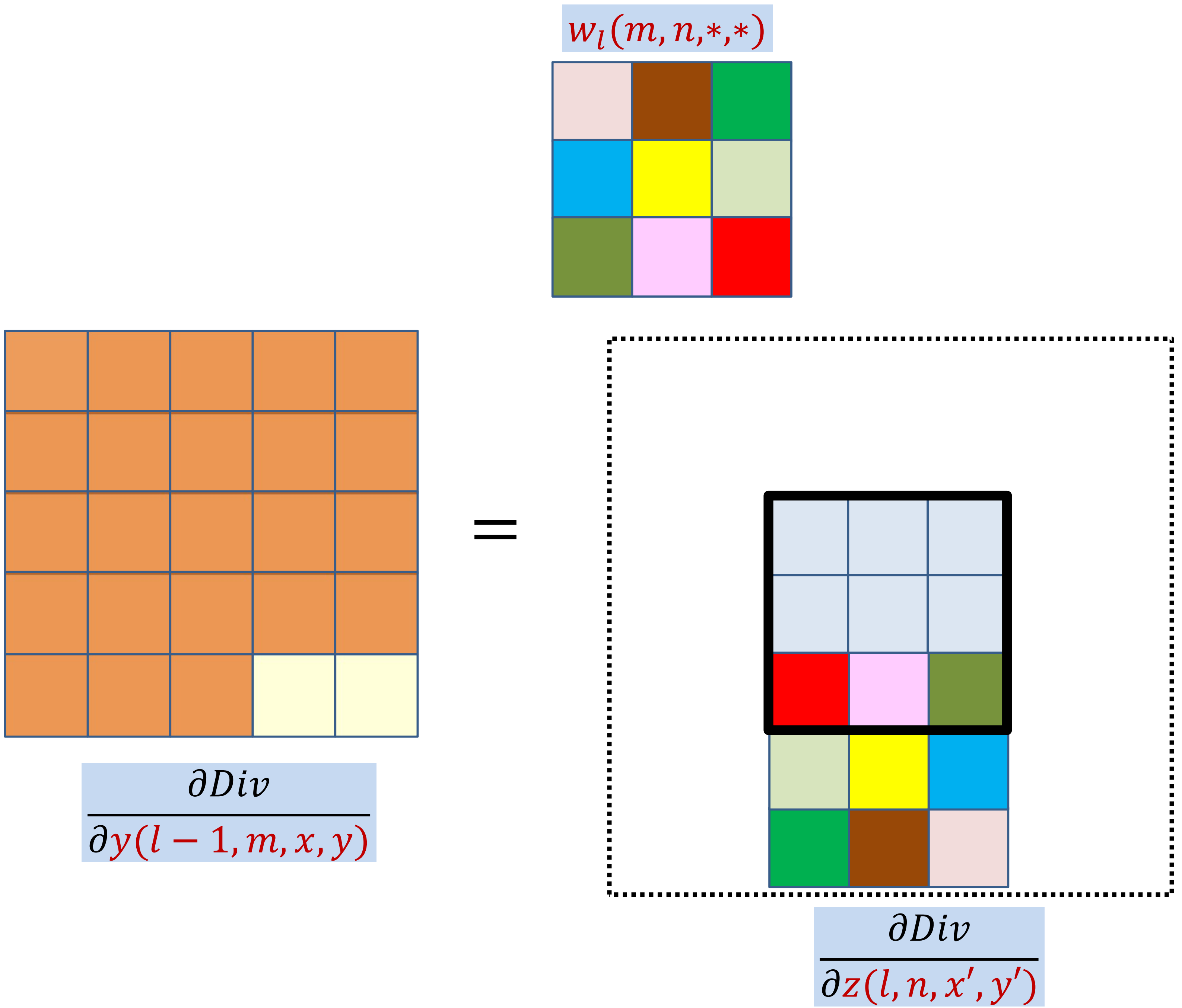
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



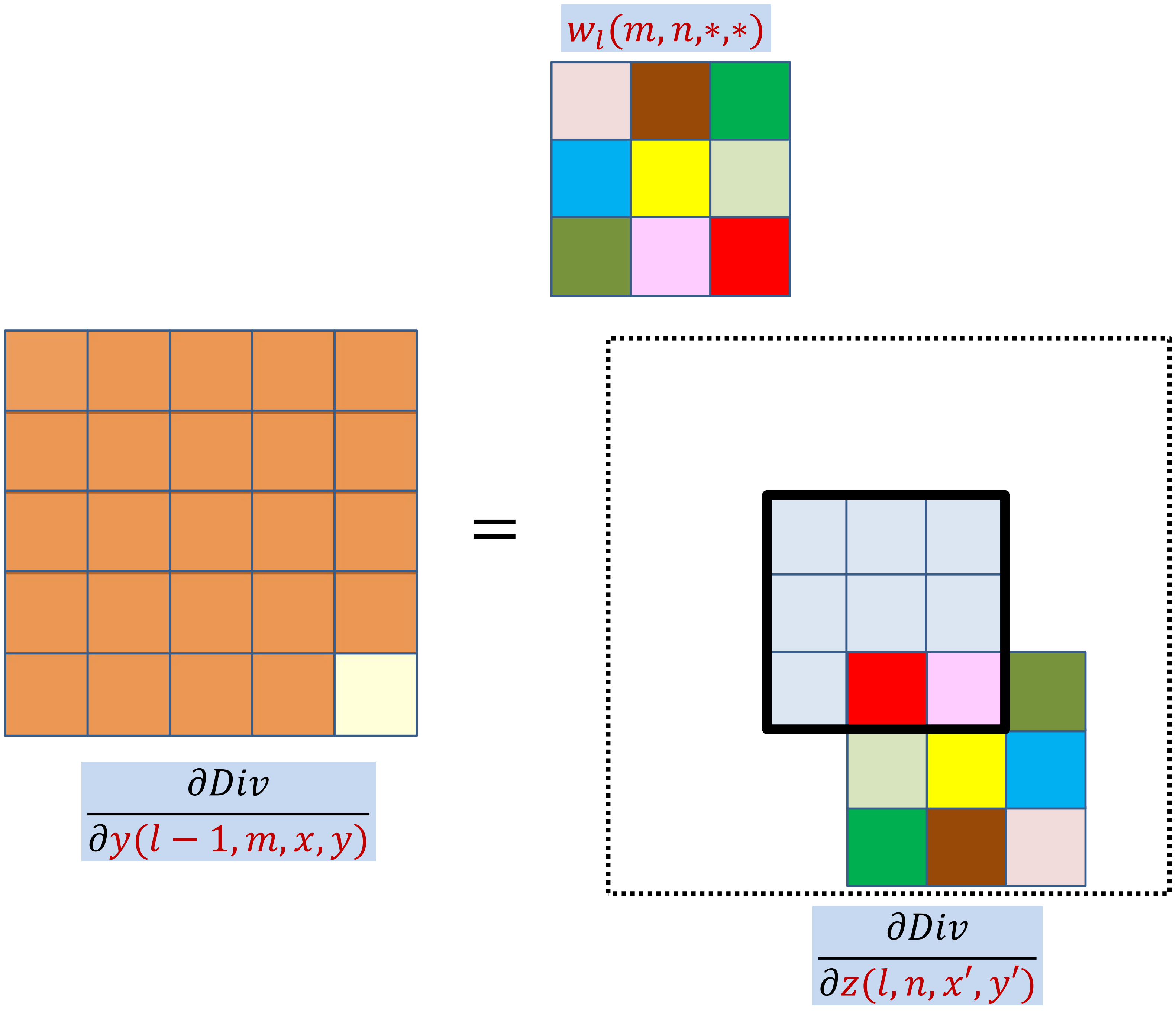
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



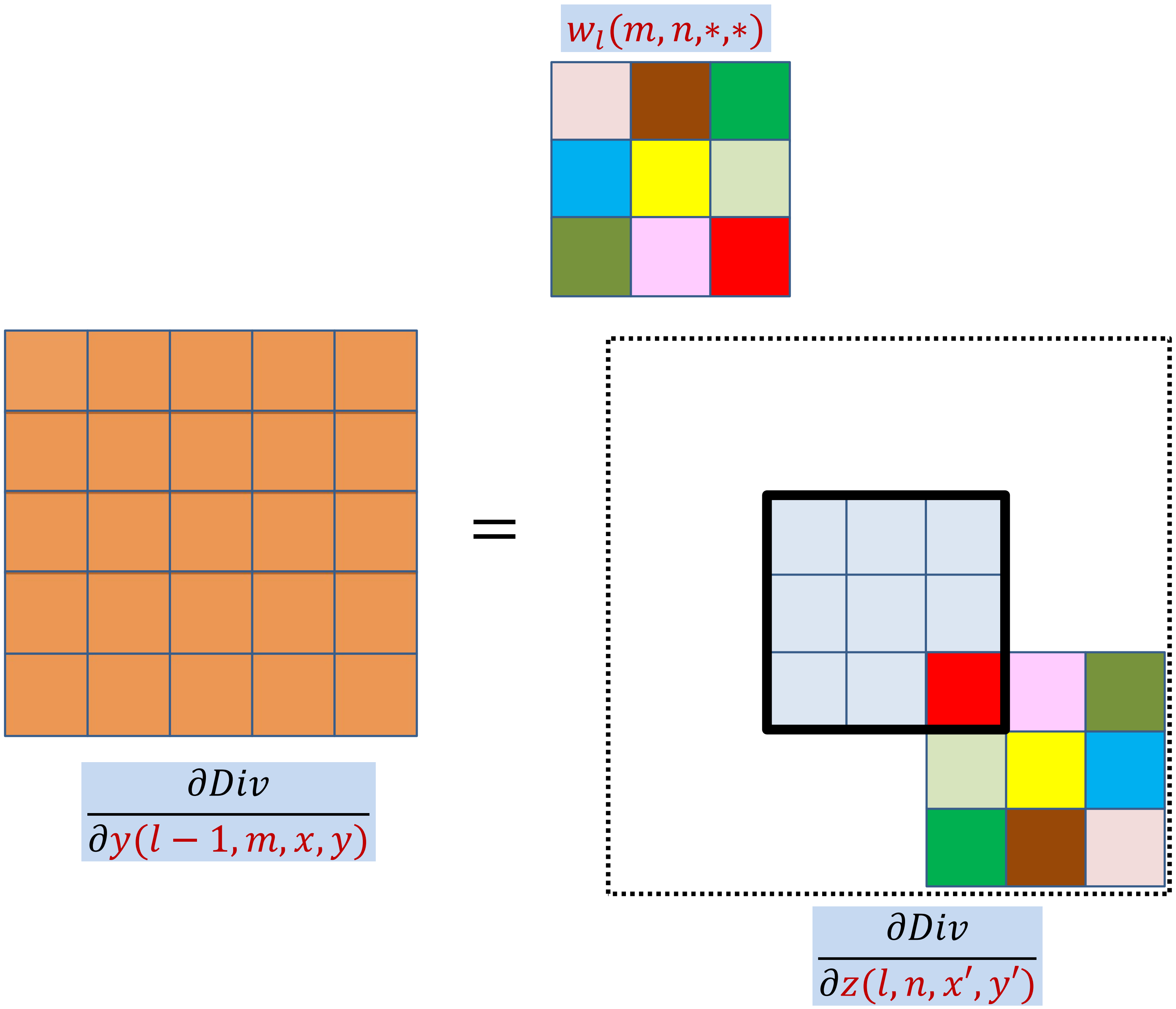
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



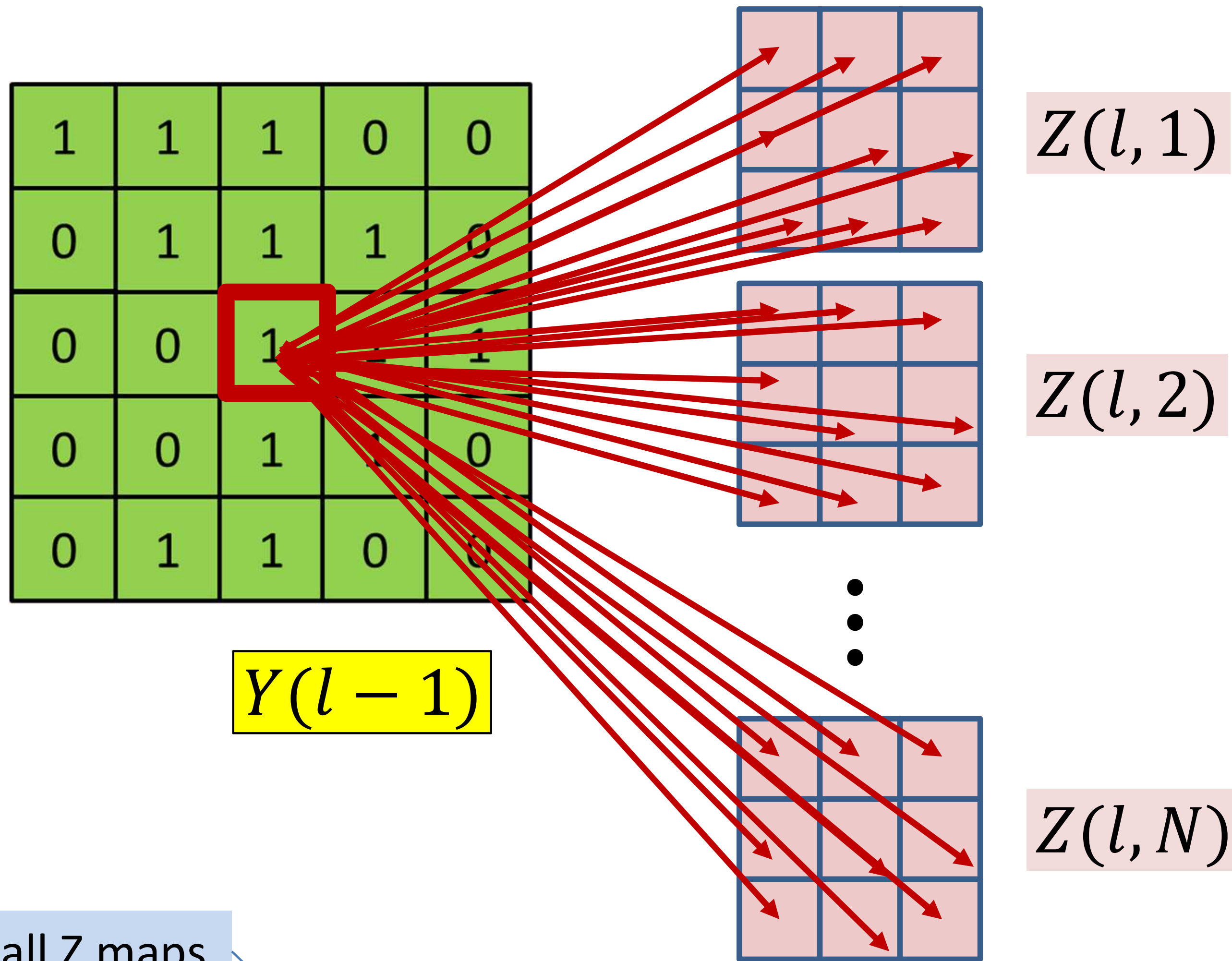
Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



Derivative at $Y(l-1,m)$ from a single $Z(l,n)$ map



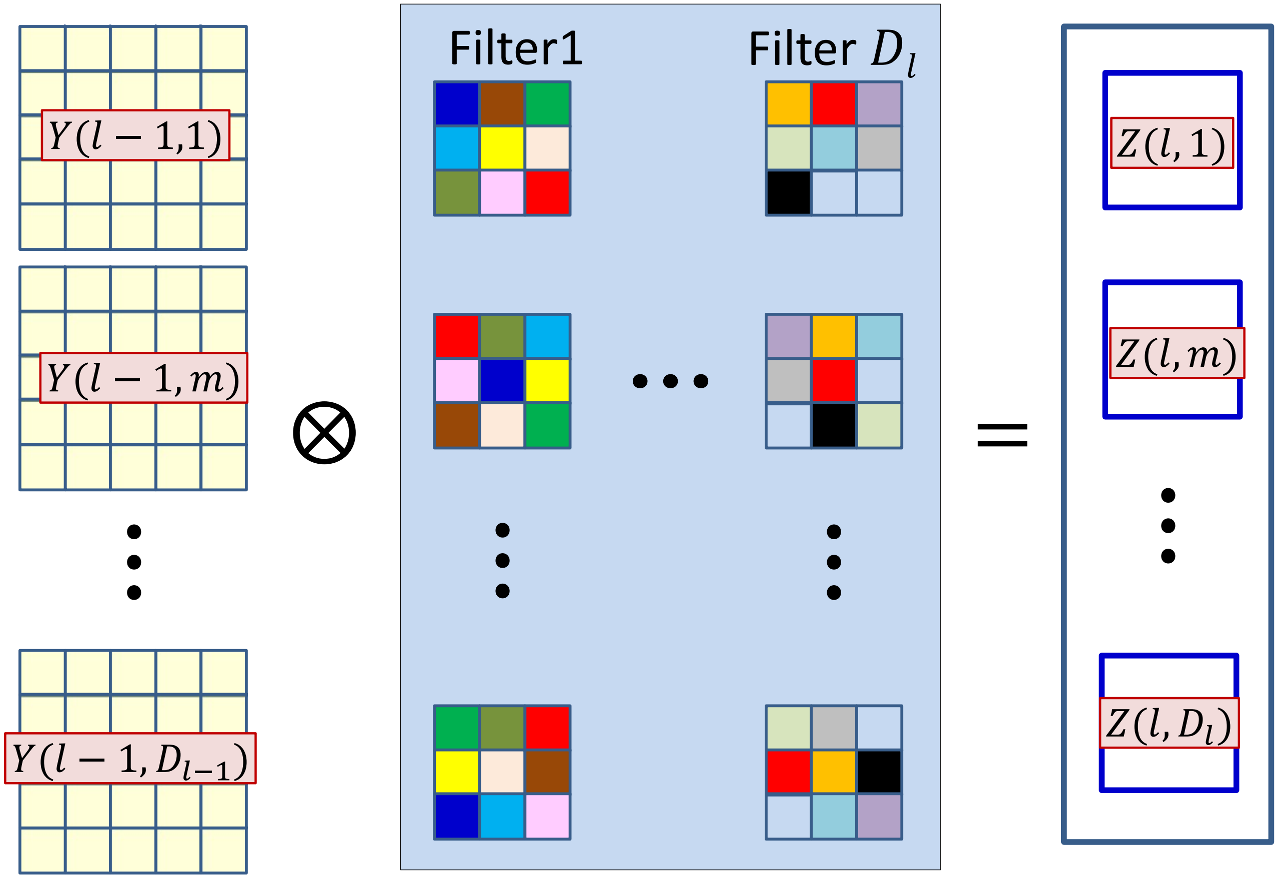
BP: Convolutional layer



Summing over all Z maps

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

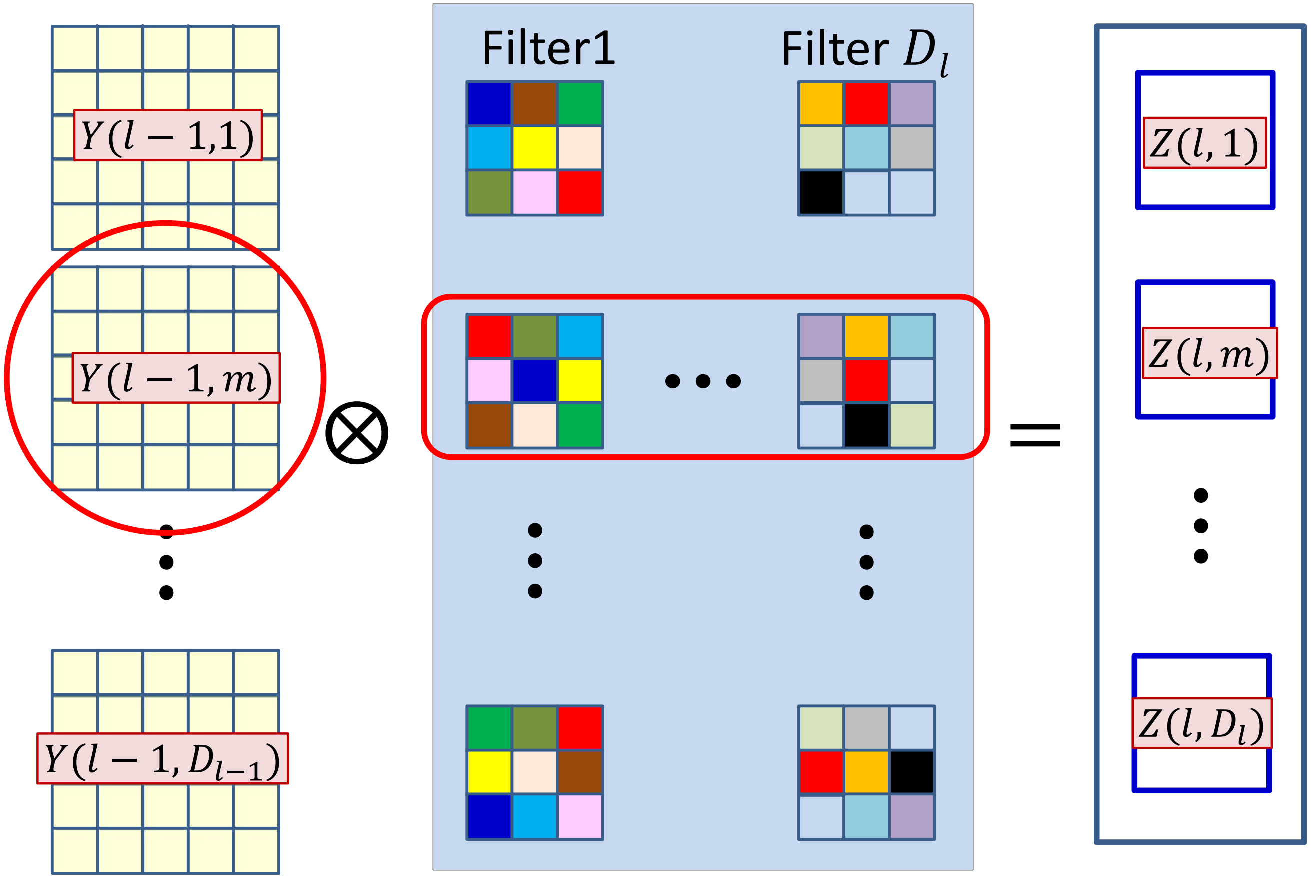
The actual convolutions



l is layer index

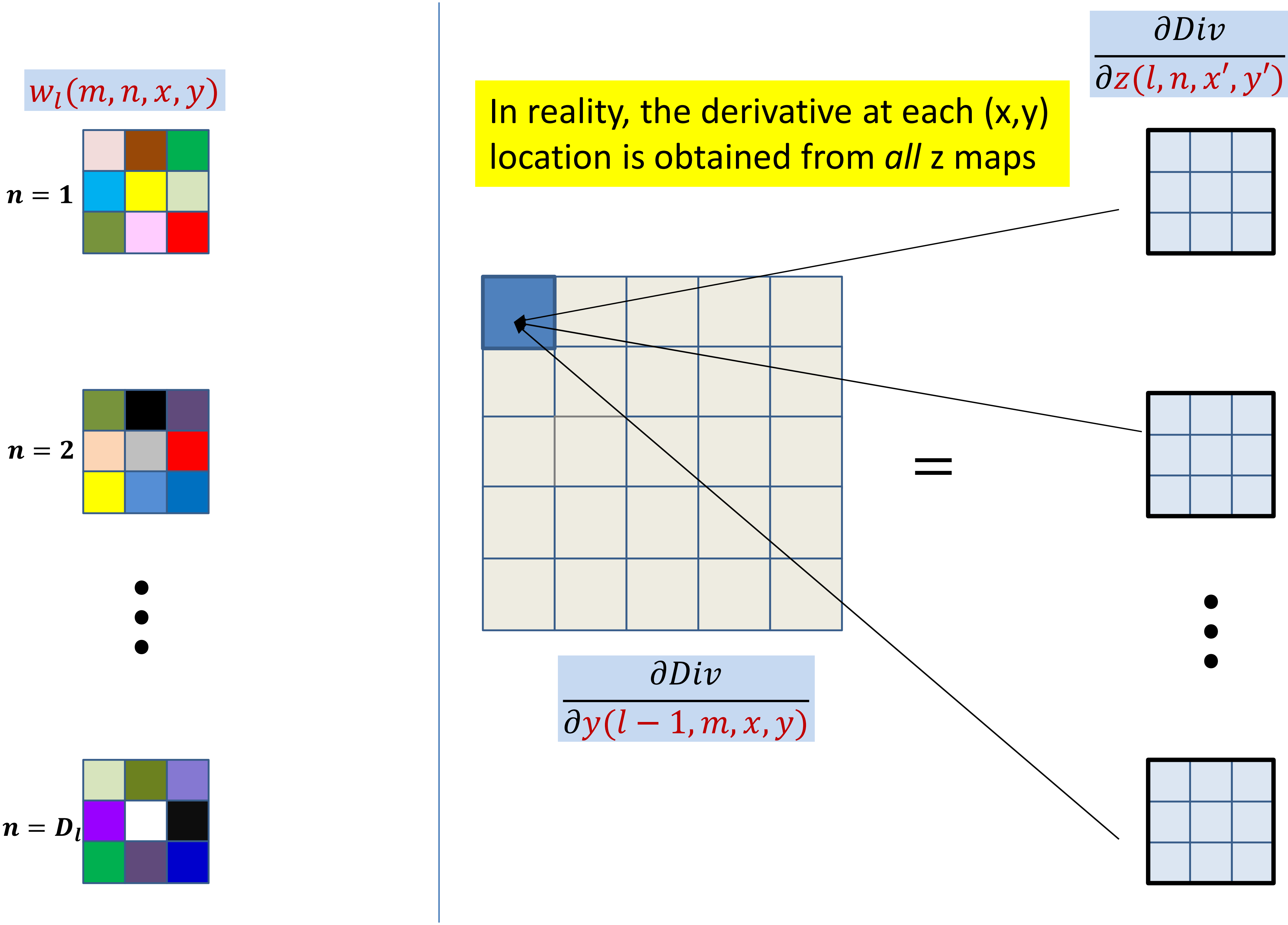
- The D_l affine maps are produced by convolving with D_l filters

The actual convolutions

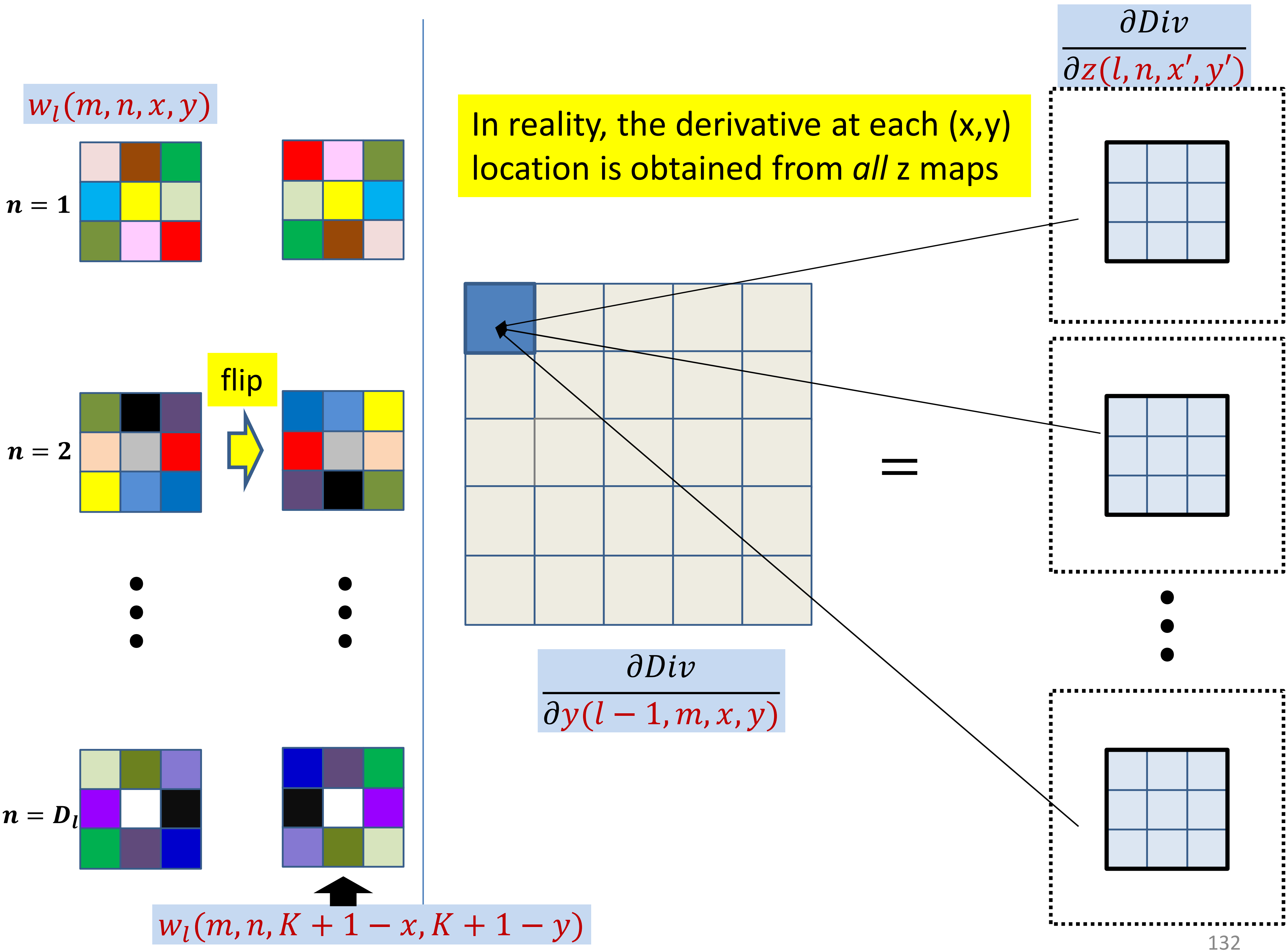


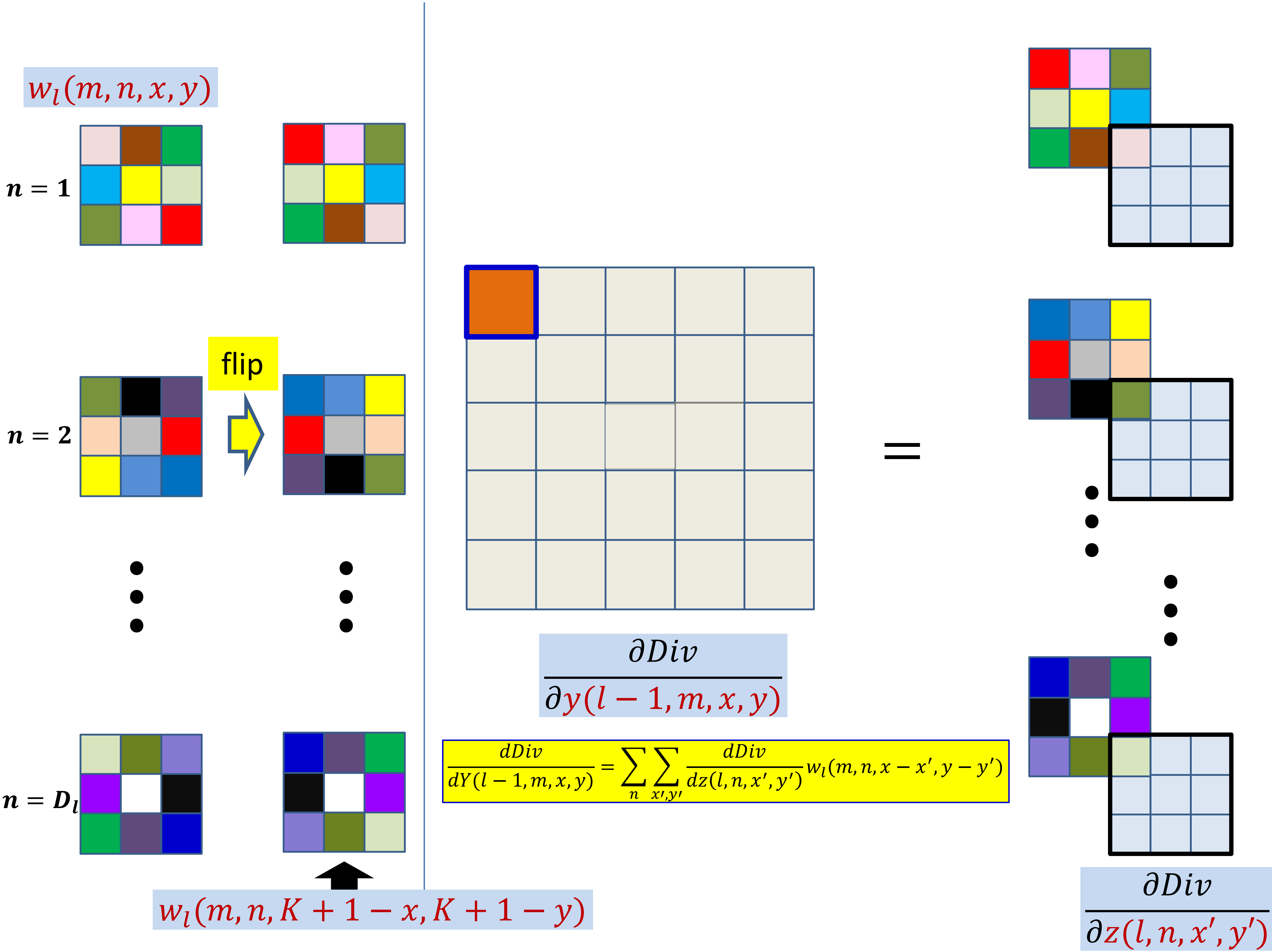
$\boldsymbol{l}$ is layer index

- The D_l affine maps are produced by convolving with D_l filters
- The m^{th} Y map always convolves the m^{th} plane of the filters
- The derivative for the m^{th} Y map will invoke the m^{th} plane of *all* the filters

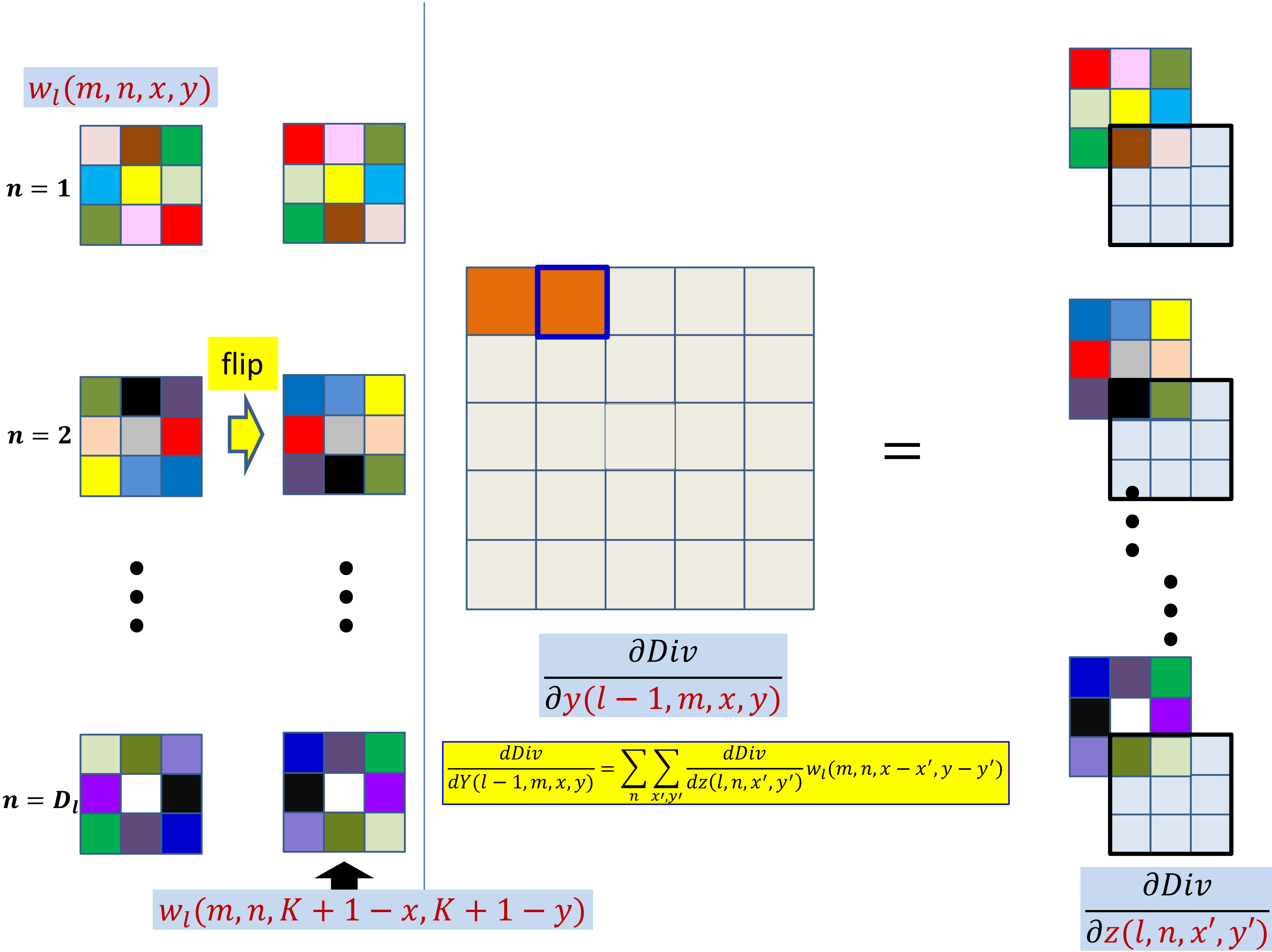


$w_l(m,n,x,y)$
$n = 1$
$n = 2$
$n = D_l$
In reality, the derivative at each (x,y) location is obtained from *all* z maps
$\frac{\partial Div}{\partial z(l,n,x',y')}$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=

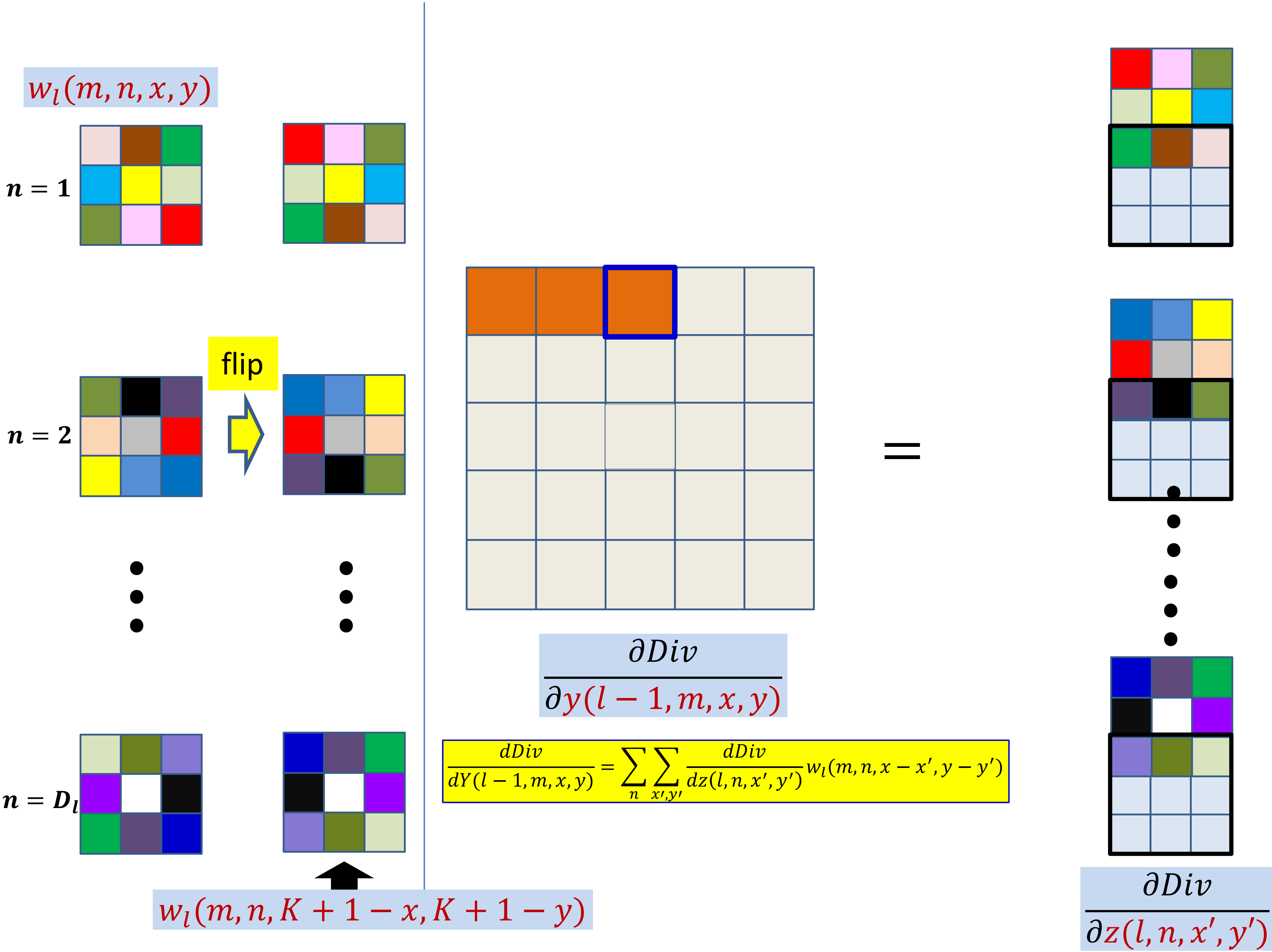

$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
In reality, the derivative at each (x,y) location is obtained from *all* z maps
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{\partial Div}{\partial z(l,n,x',y')}$



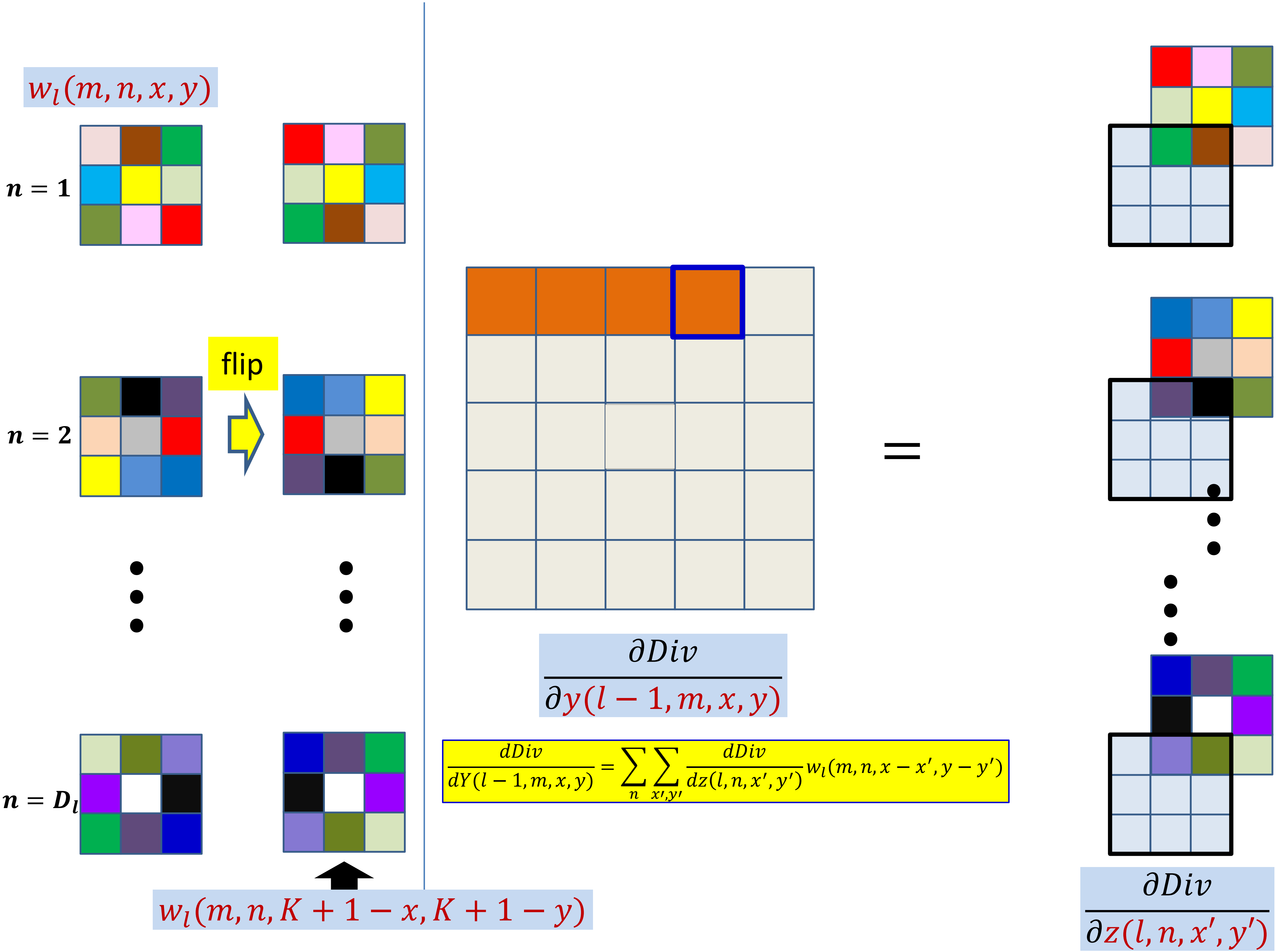
$w_l(m, n, x, y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m, n, K + 1 - x, K + 1 - y)$
$\frac{\partial Div}{\partial y(l-1, m, x, y)}$
=
$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$
$\frac{\partial Div}{\partial z(l, n, x', y')}$



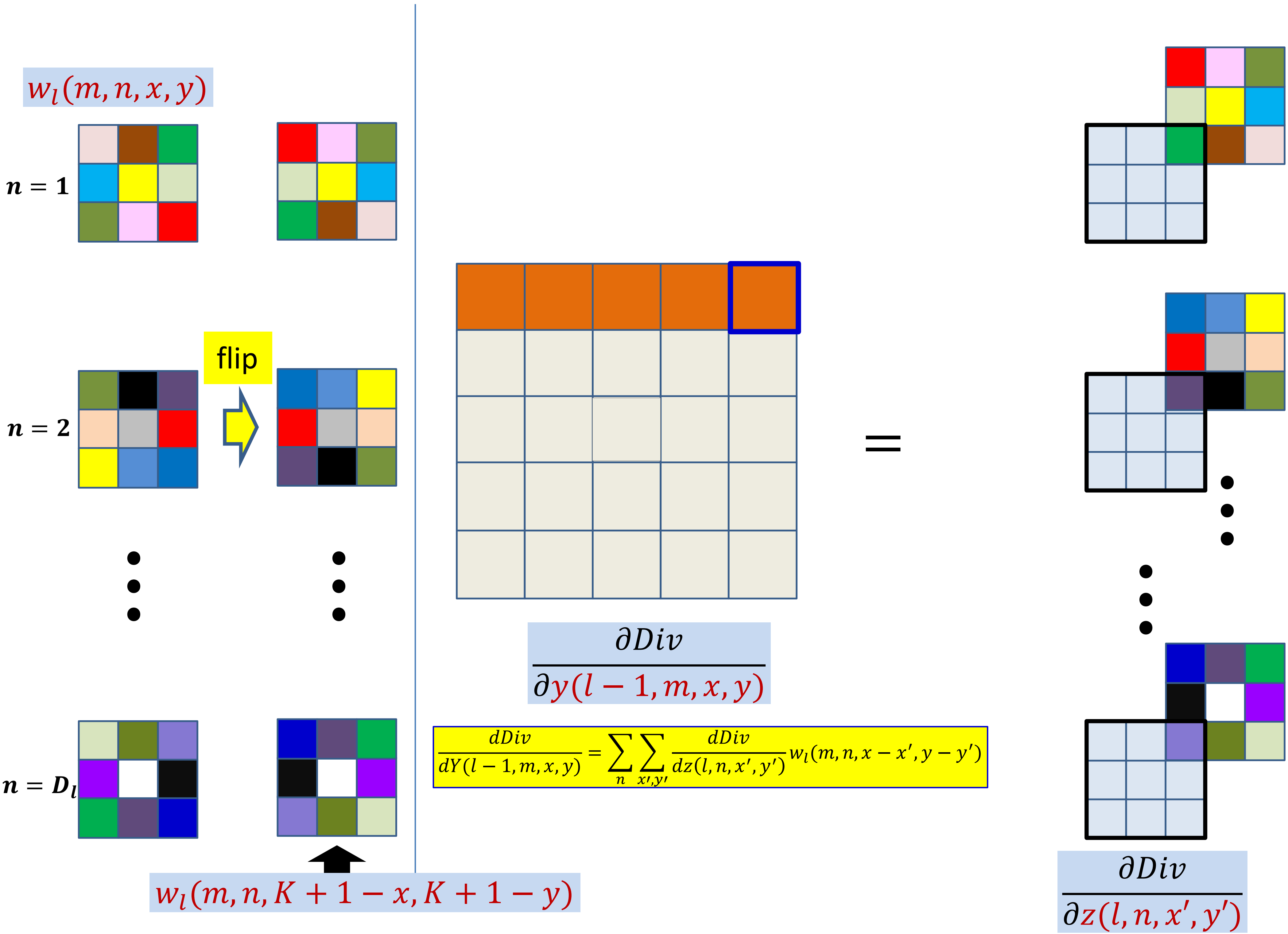
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
=
$\frac{\partial Div}{\partial z(l,n,x',y')}$



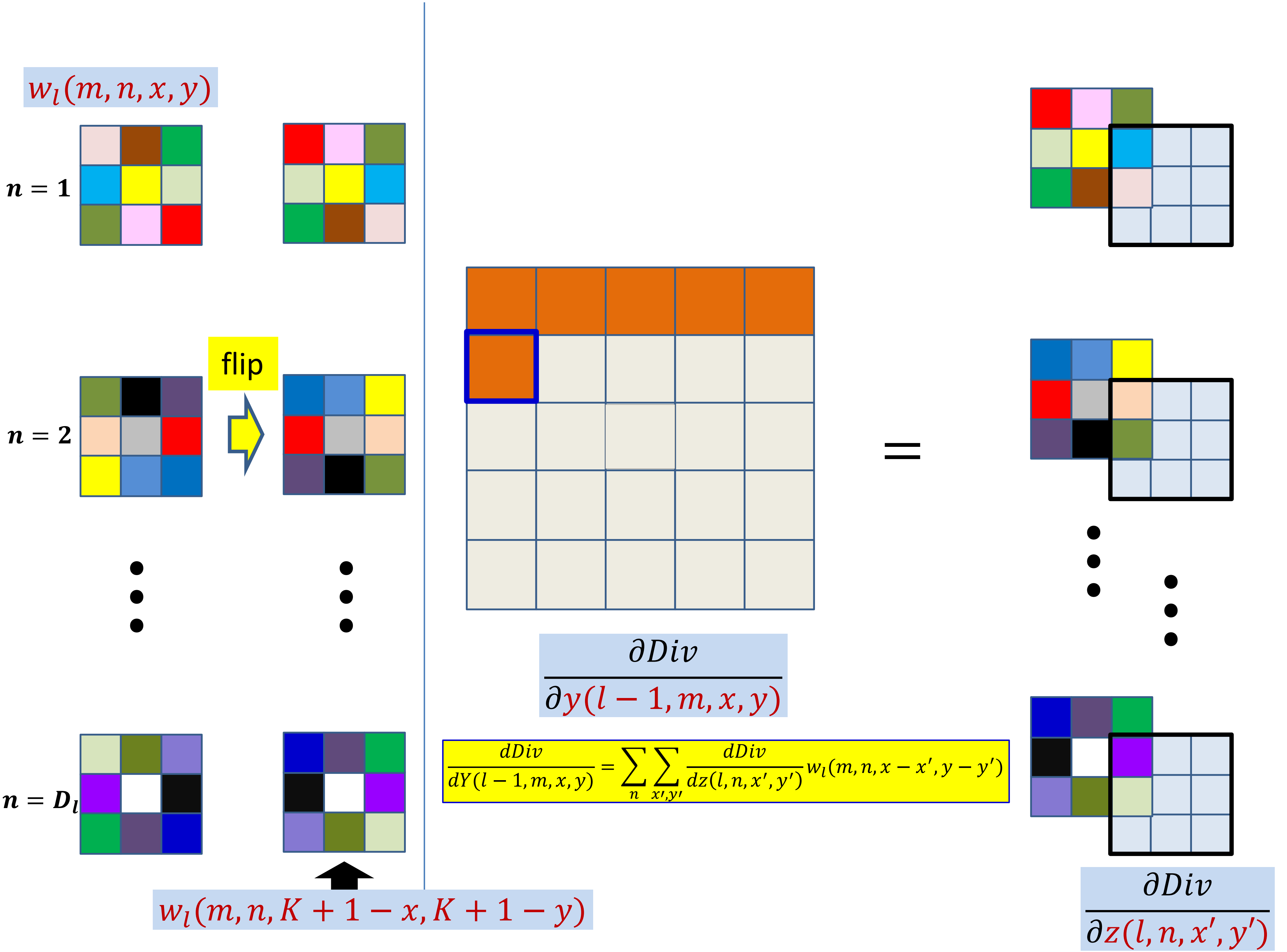
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_{n} \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



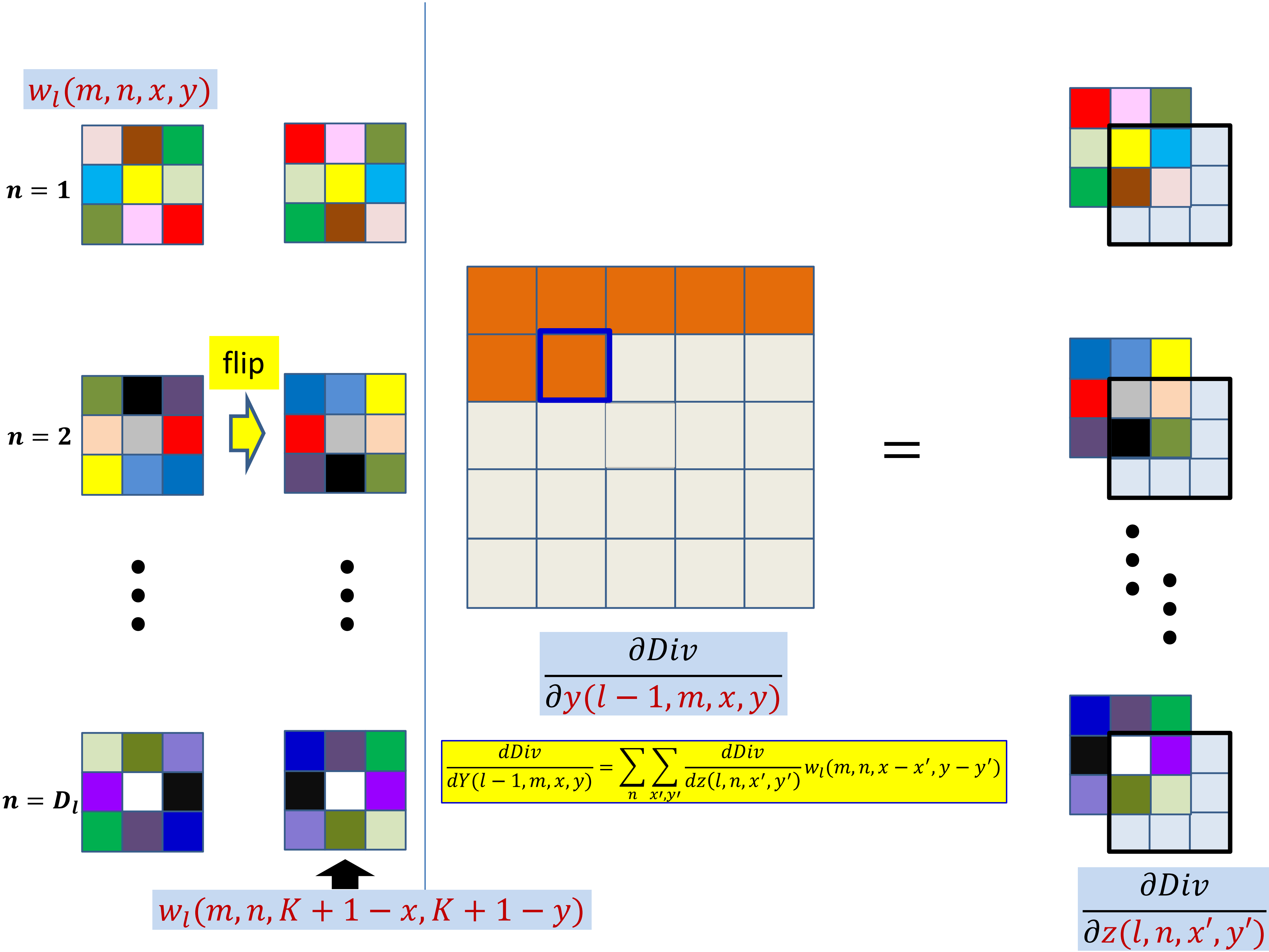
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$



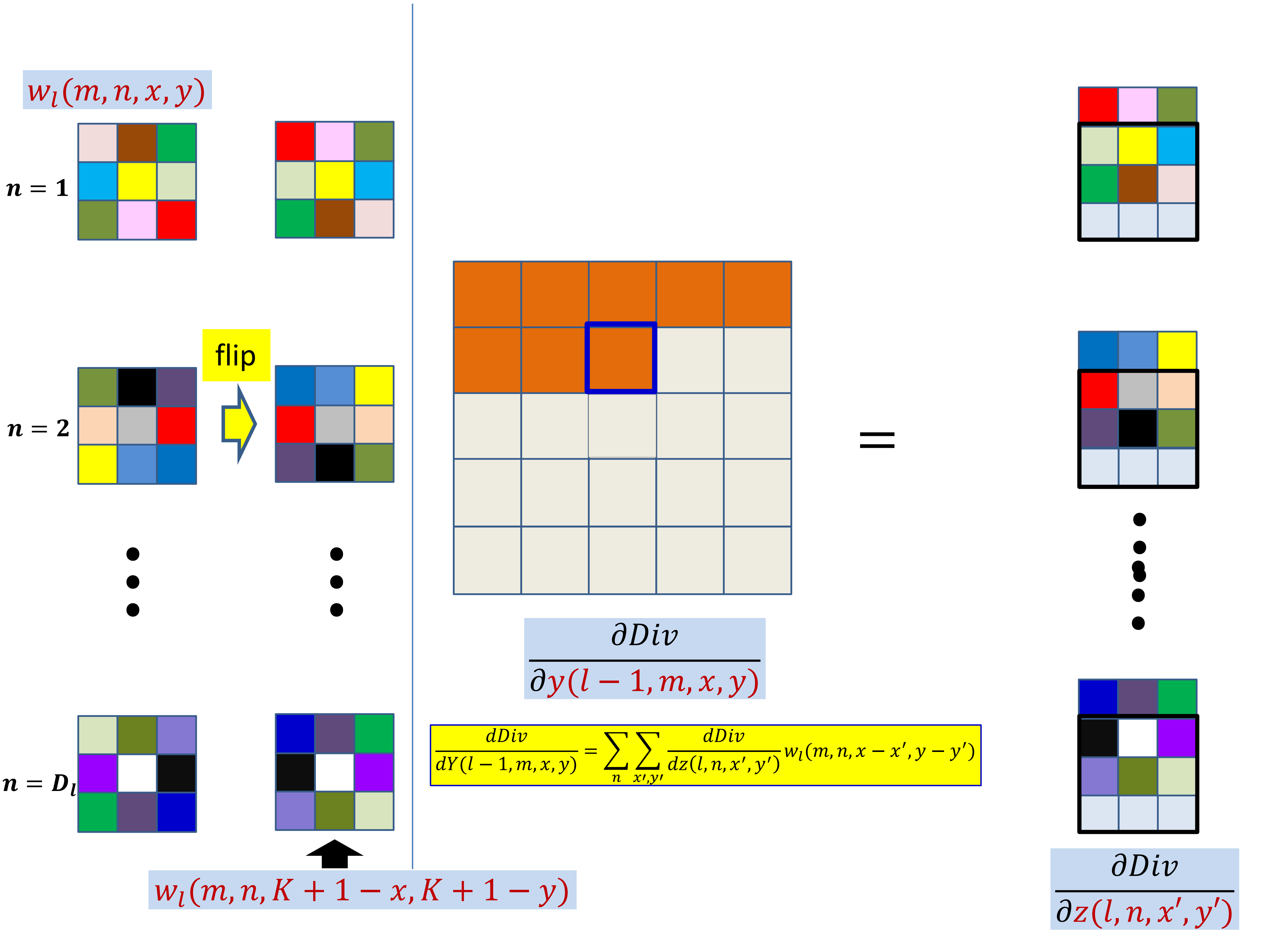
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
=
$\frac{\partial Div}{\partial z(l,n,x',y')}$



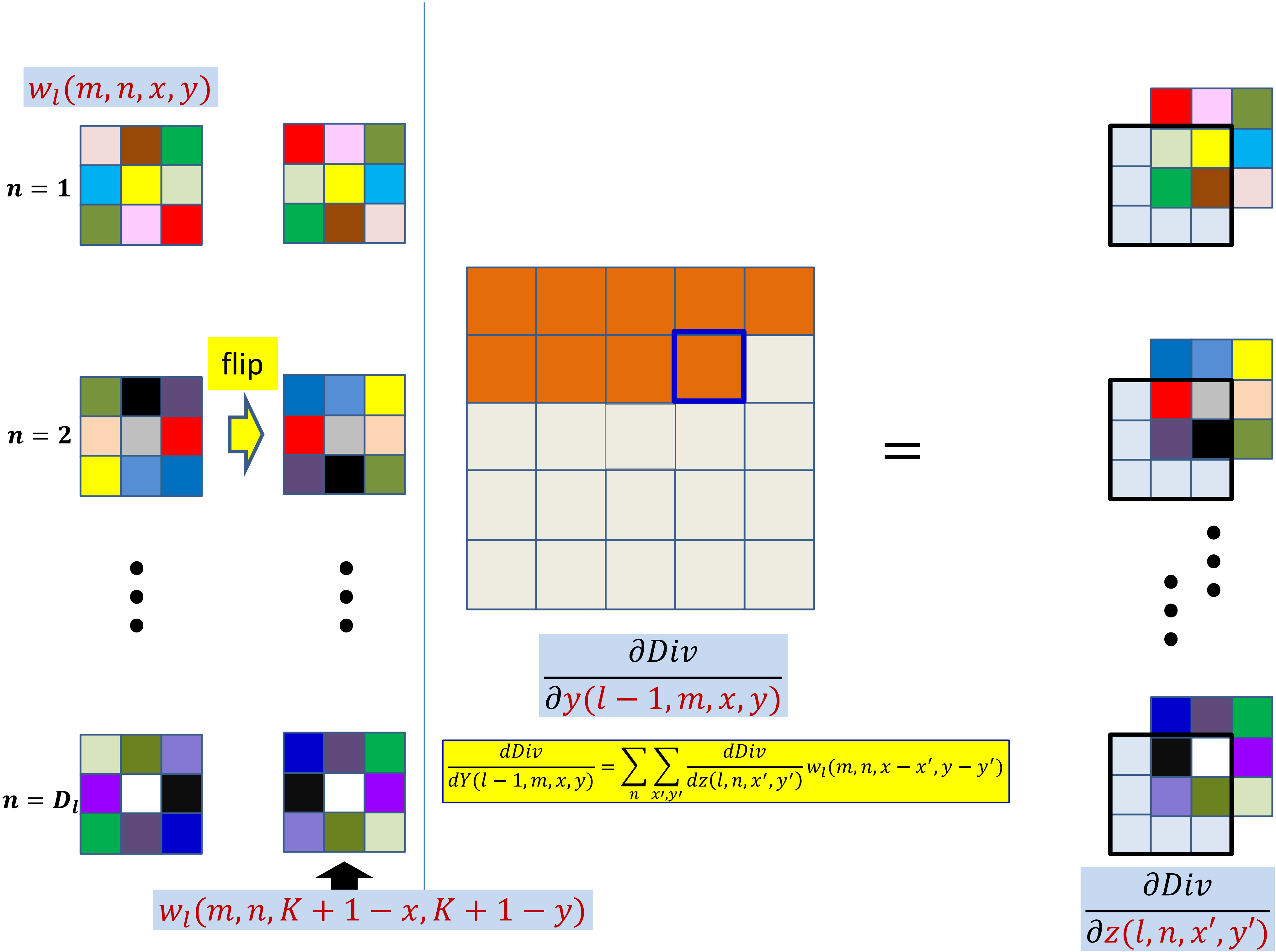
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
=
$\frac{\partial Div}{\partial z(l,n,x',y')}$



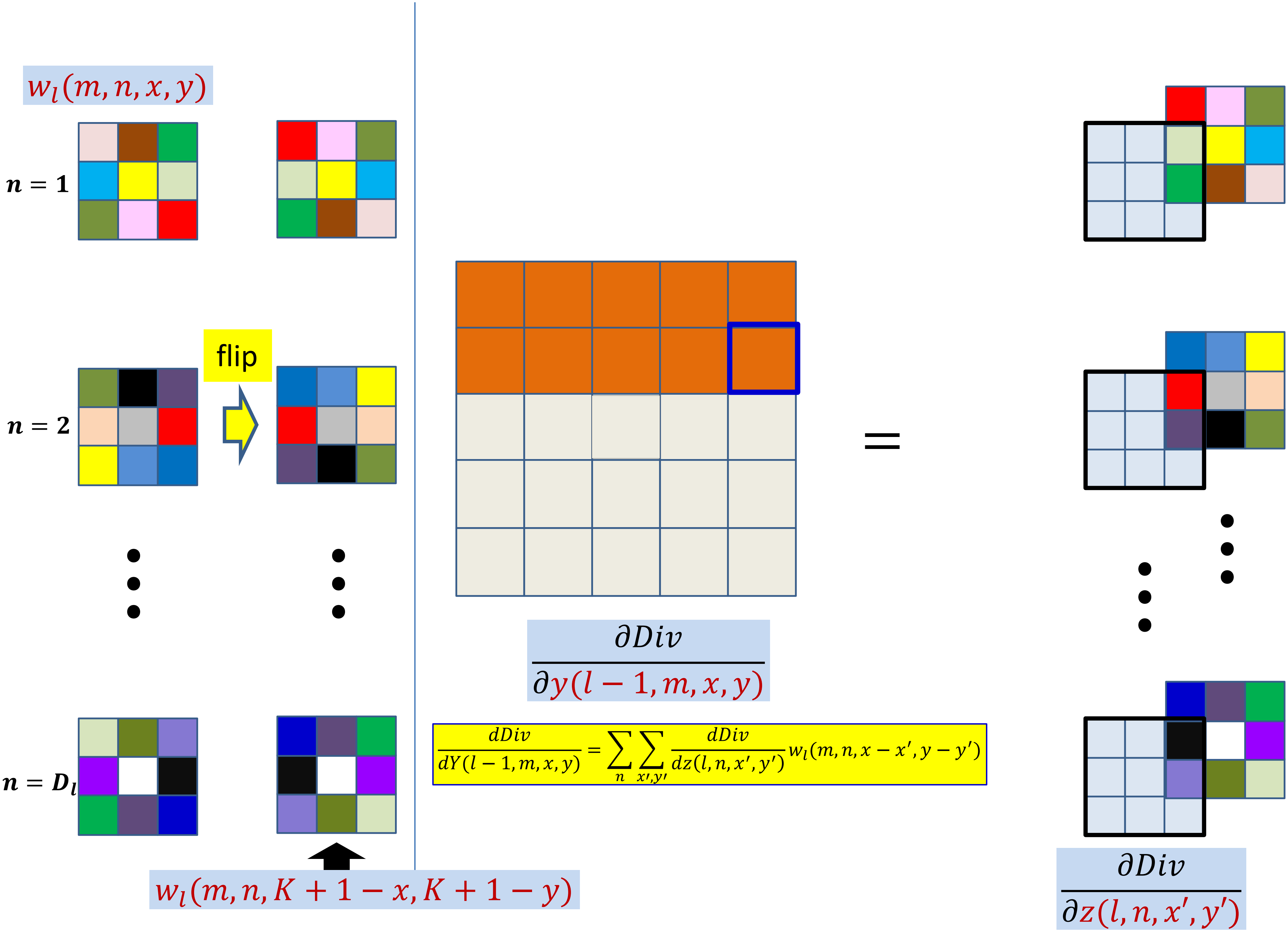
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$

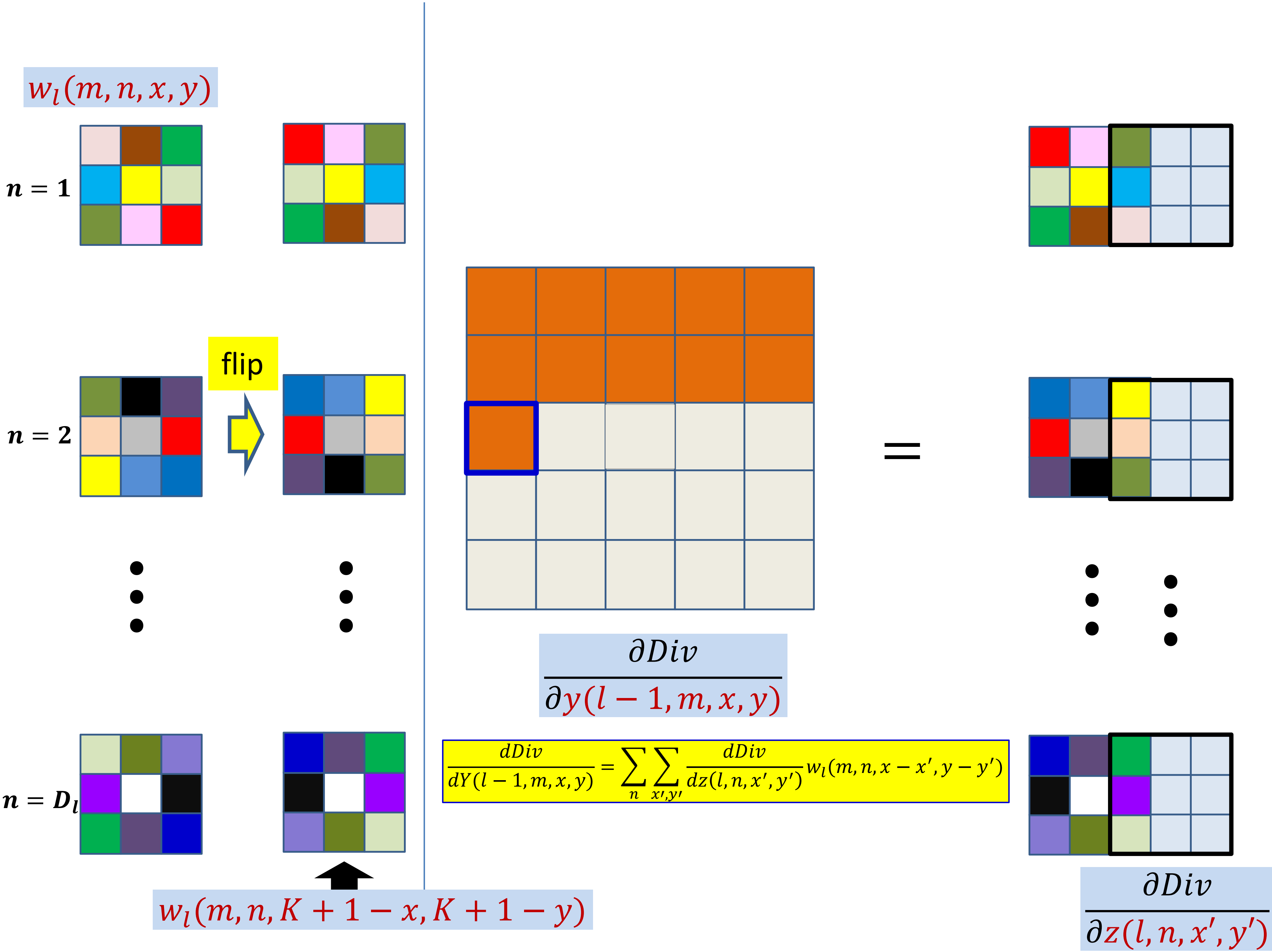


$w_l(m,n,x,y)$
$n = 1$
$n = 2$
$n = D_l$
flip
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$

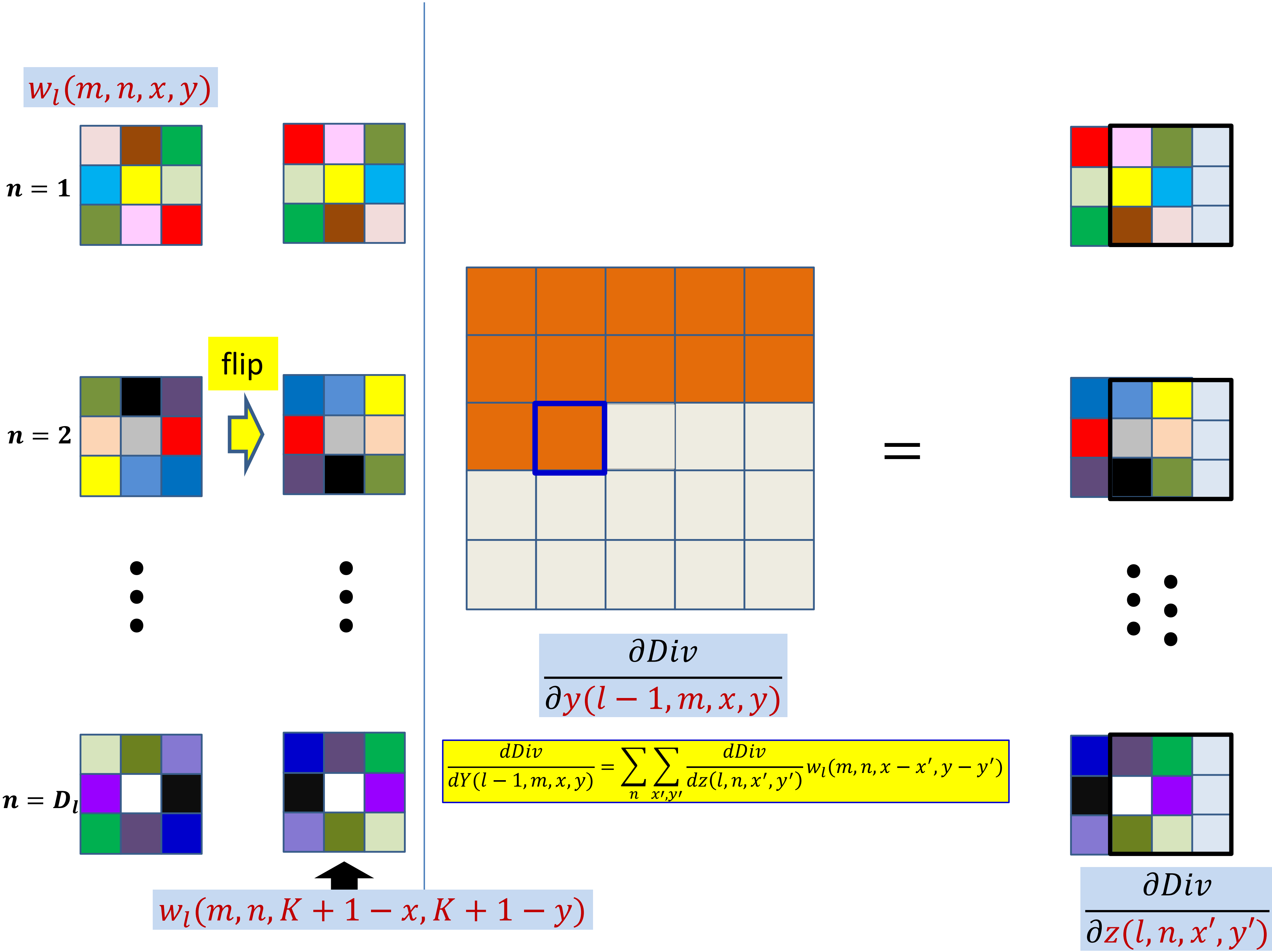


$w_l(m,n,x,y)$
$n = 1$
flip
$n = 2$
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$

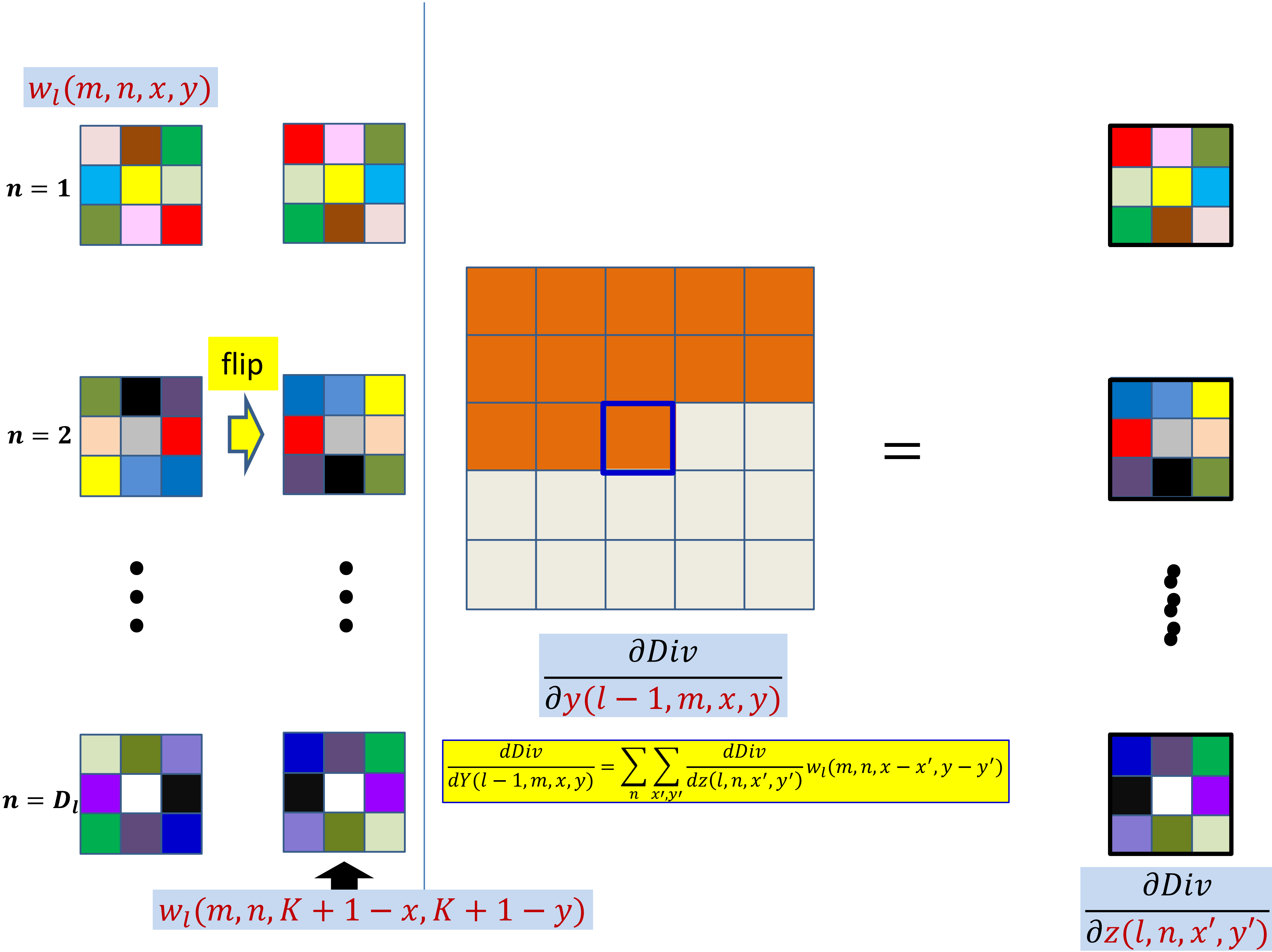

$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
=
$\frac{\partial Div}{\partial z(l,n,x',y')}$



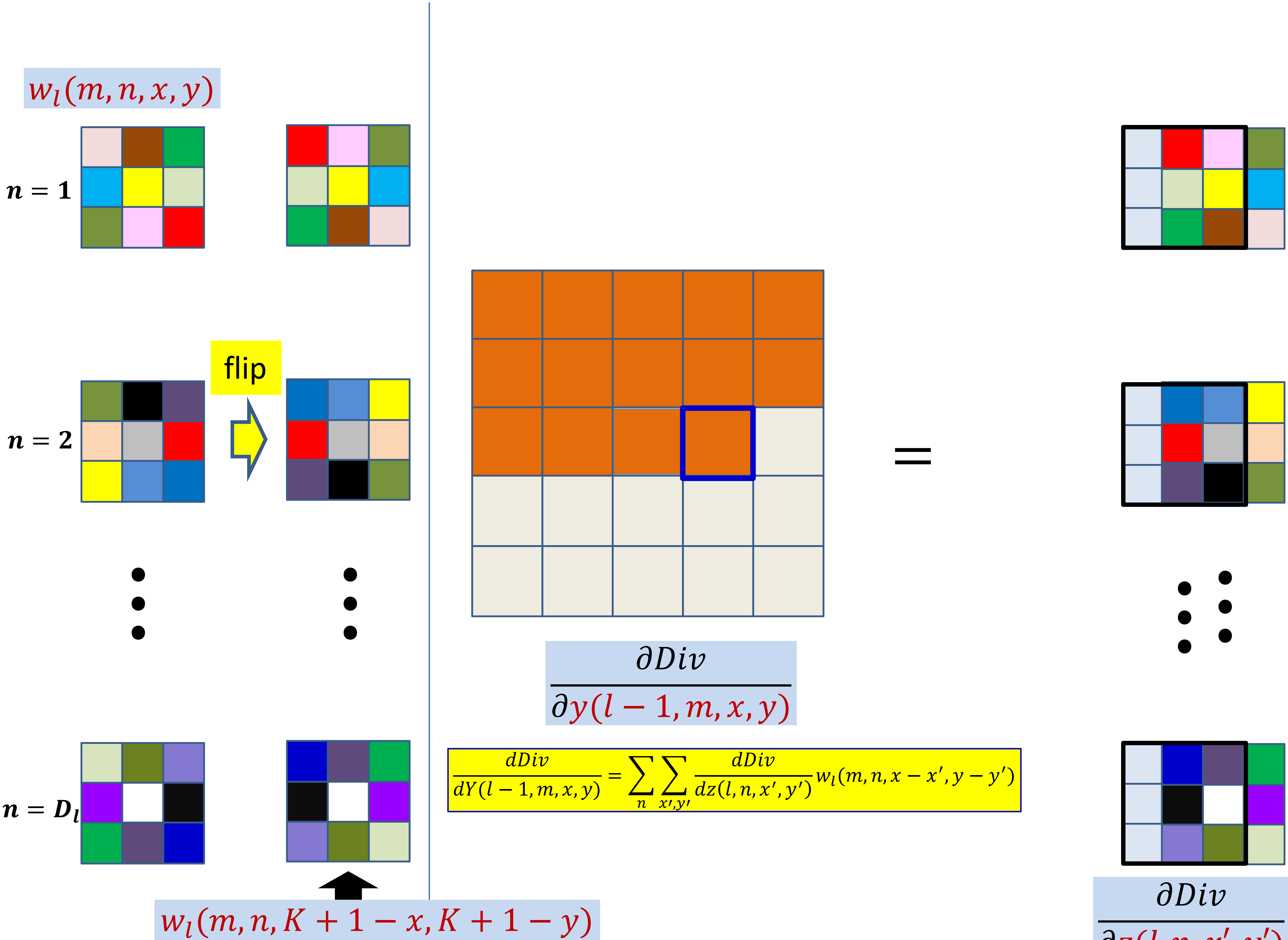
$w_l(m, n, x, y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m, n, K + 1 - x, K + 1 - y)$
=
$\frac{\partial Div}{\partial y(l-1, m, x, y)}$
$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$
$\frac{\partial Div}{\partial z(l, n, x', y')}$



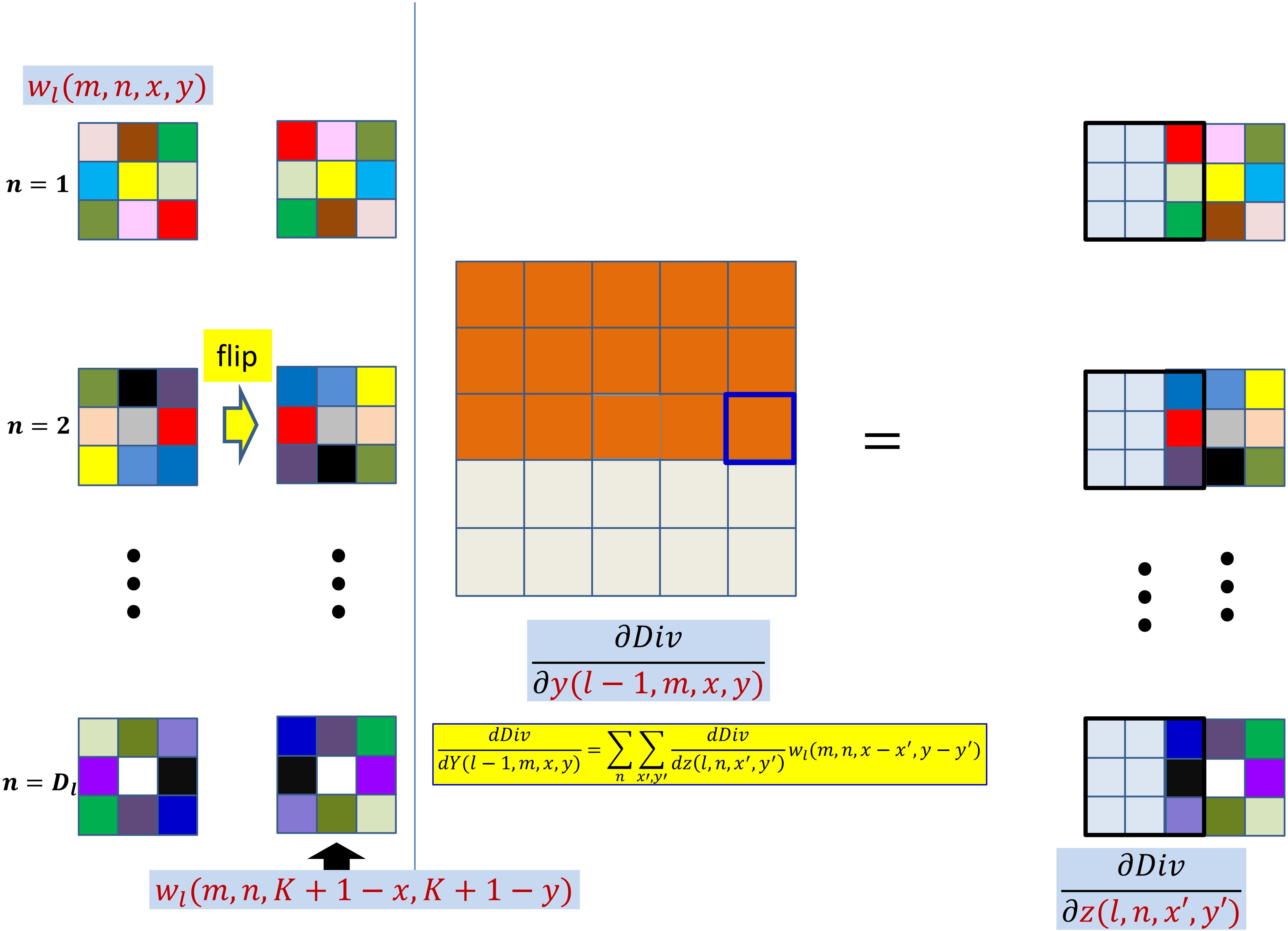
$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$



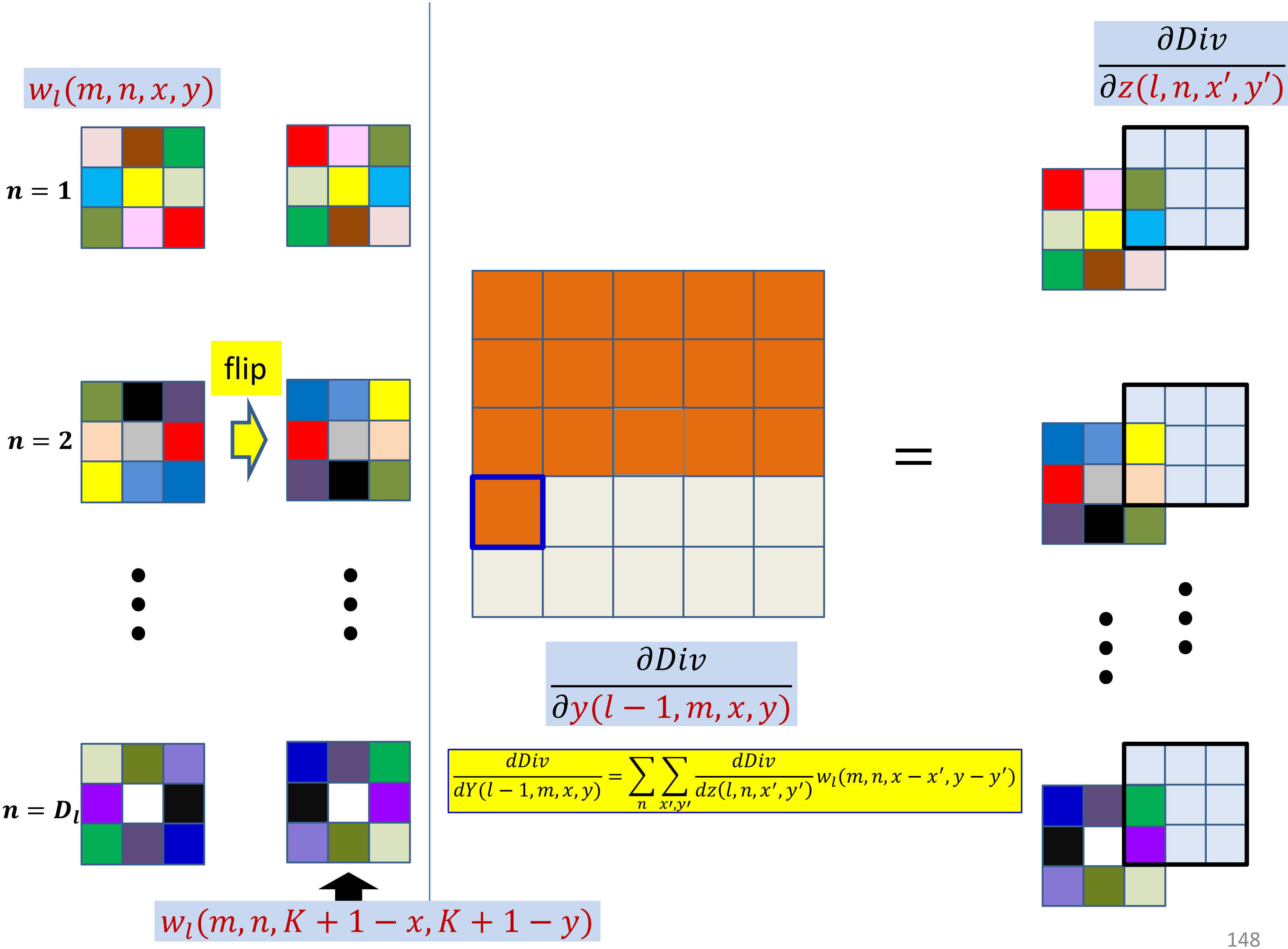
$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$



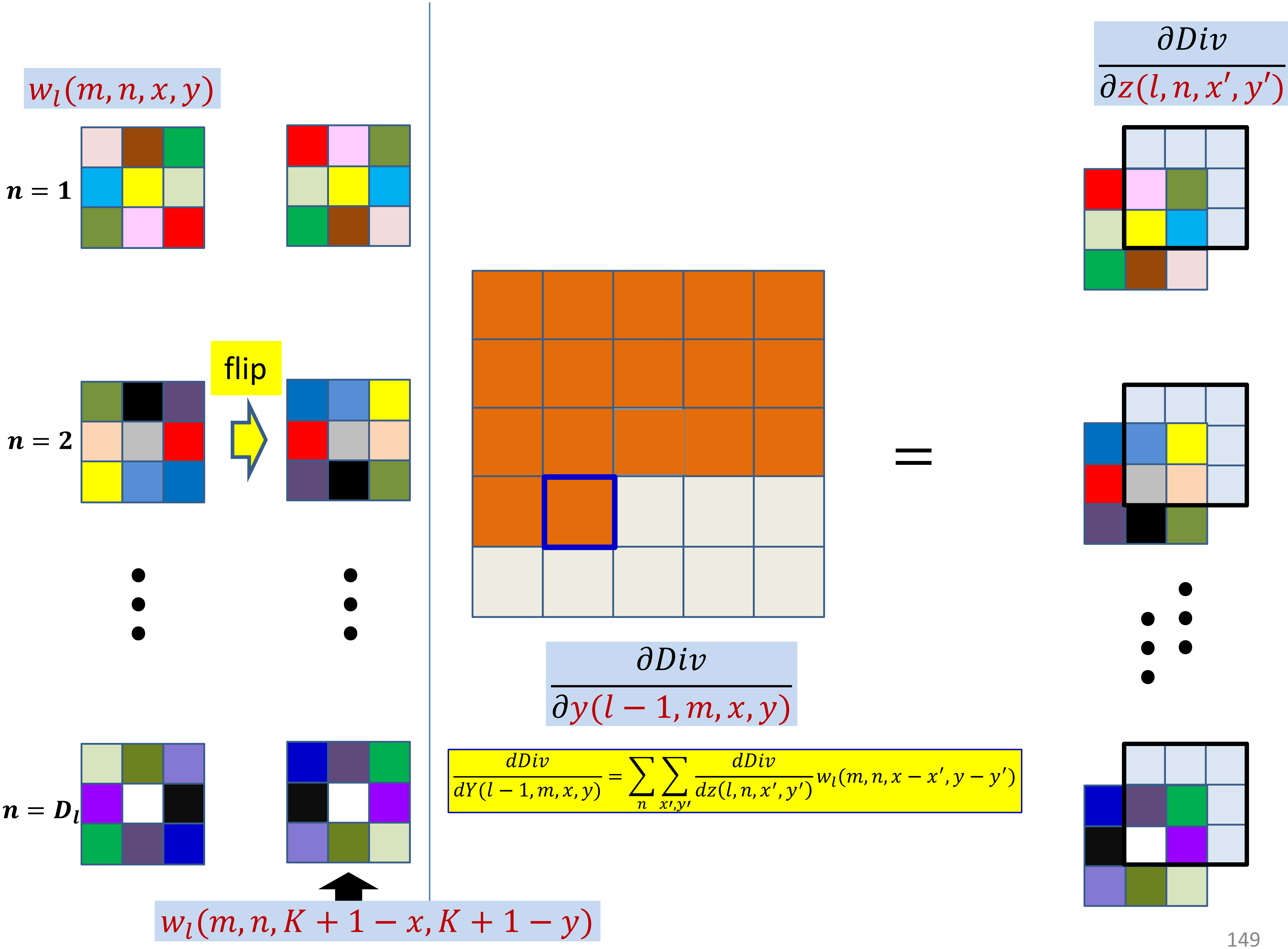
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$



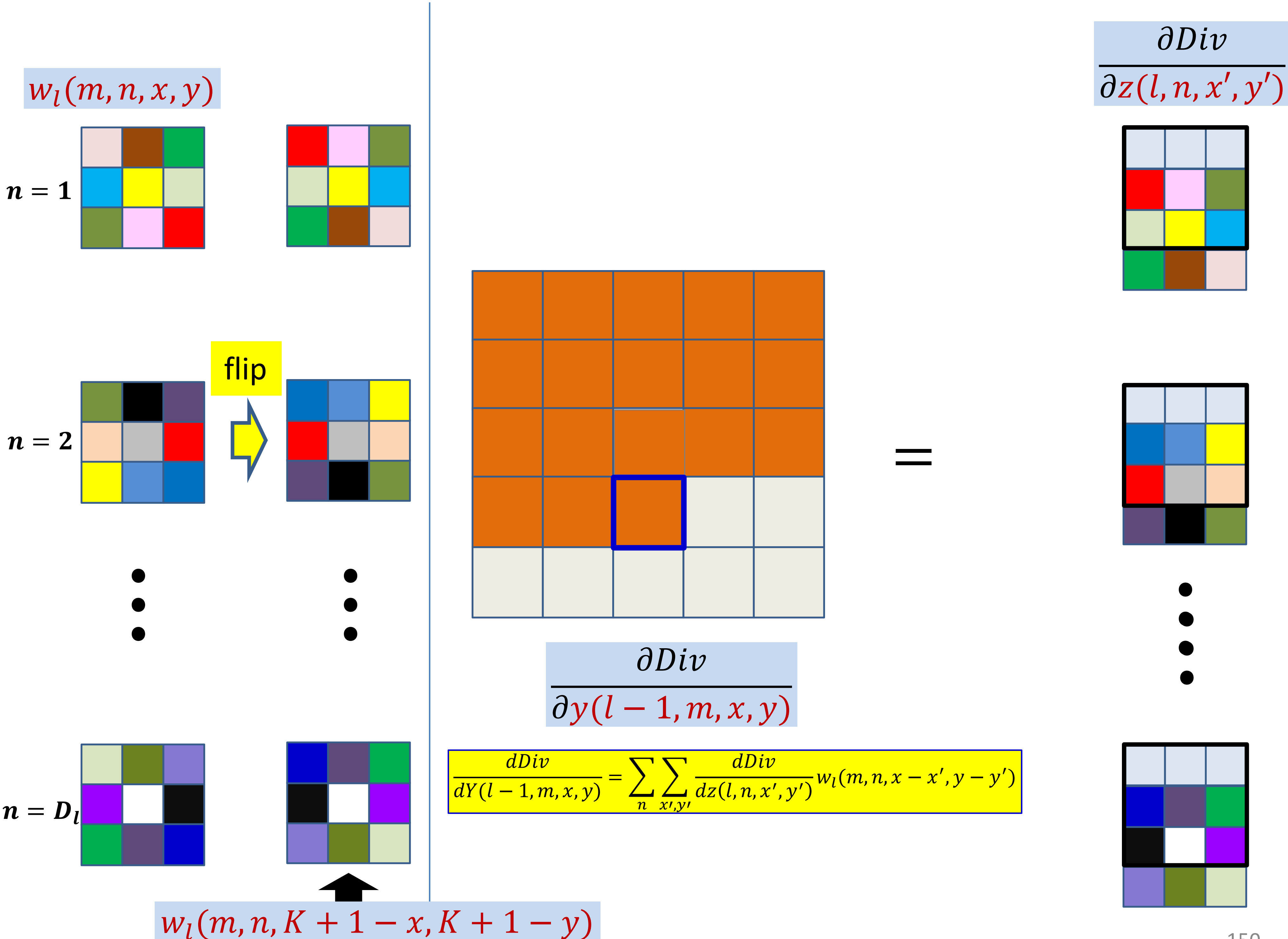
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
=
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$
$\frac{\partial Div}{\partial z(l,n,x',y')}$

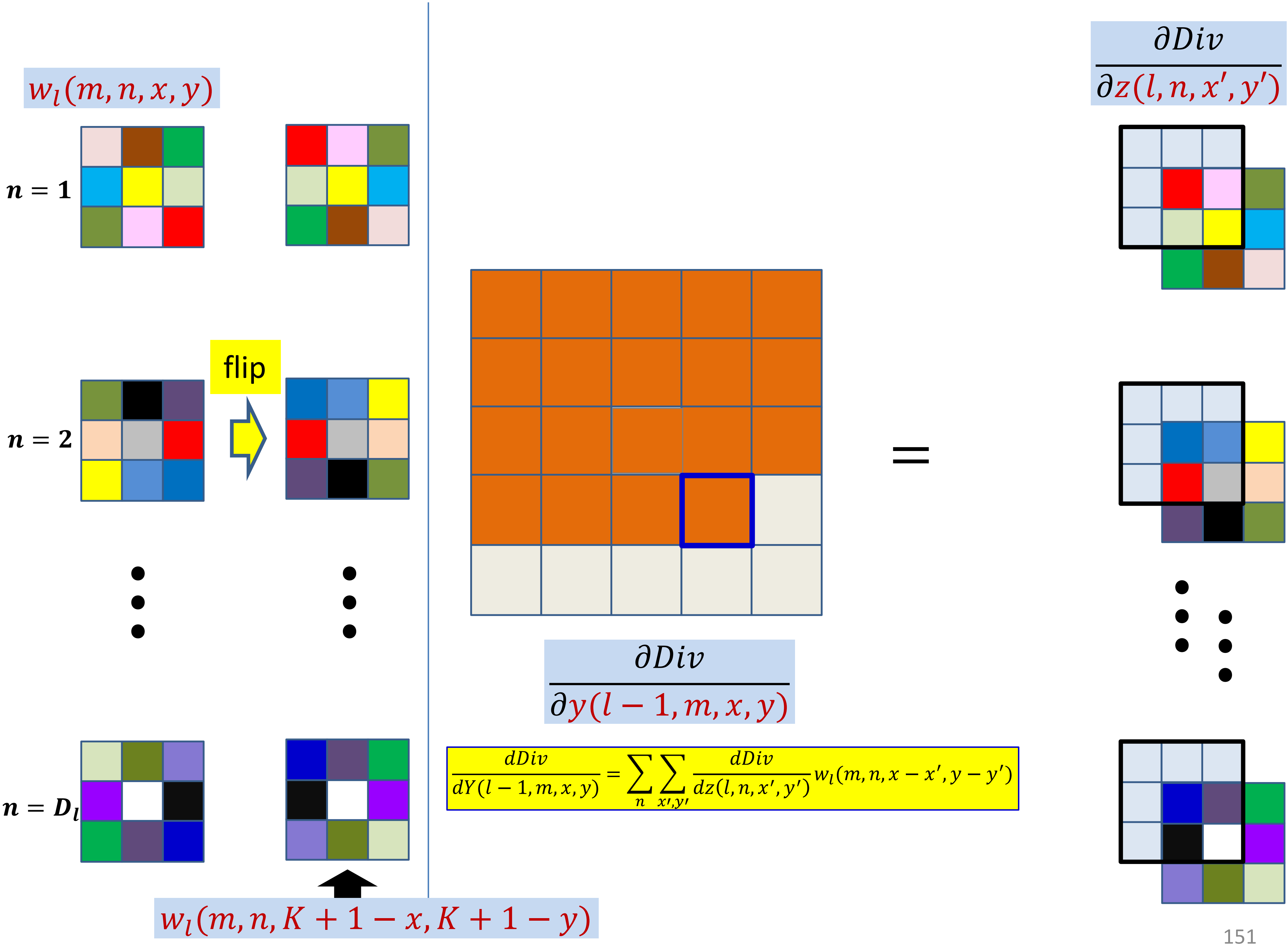


$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n}\sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

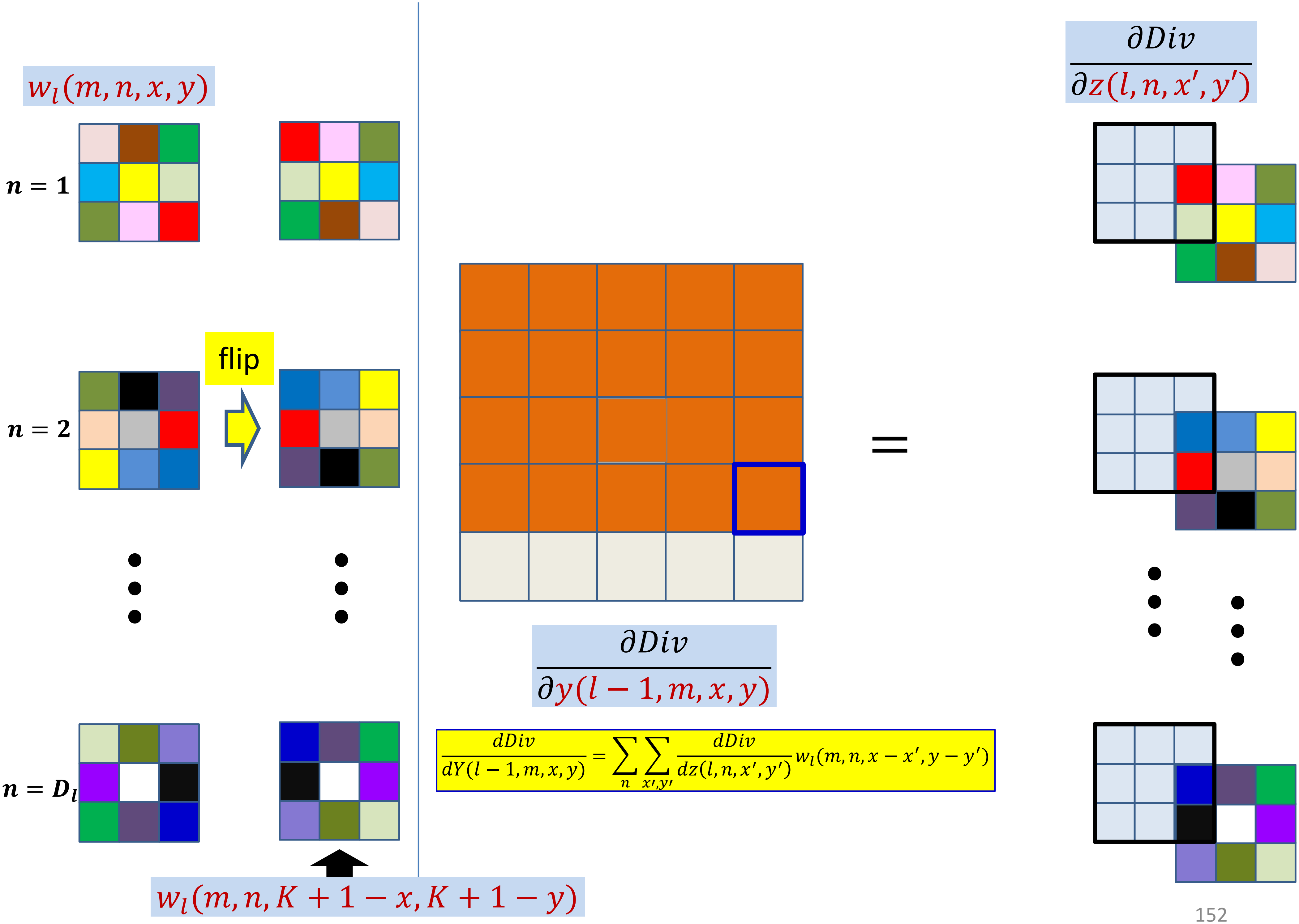


$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

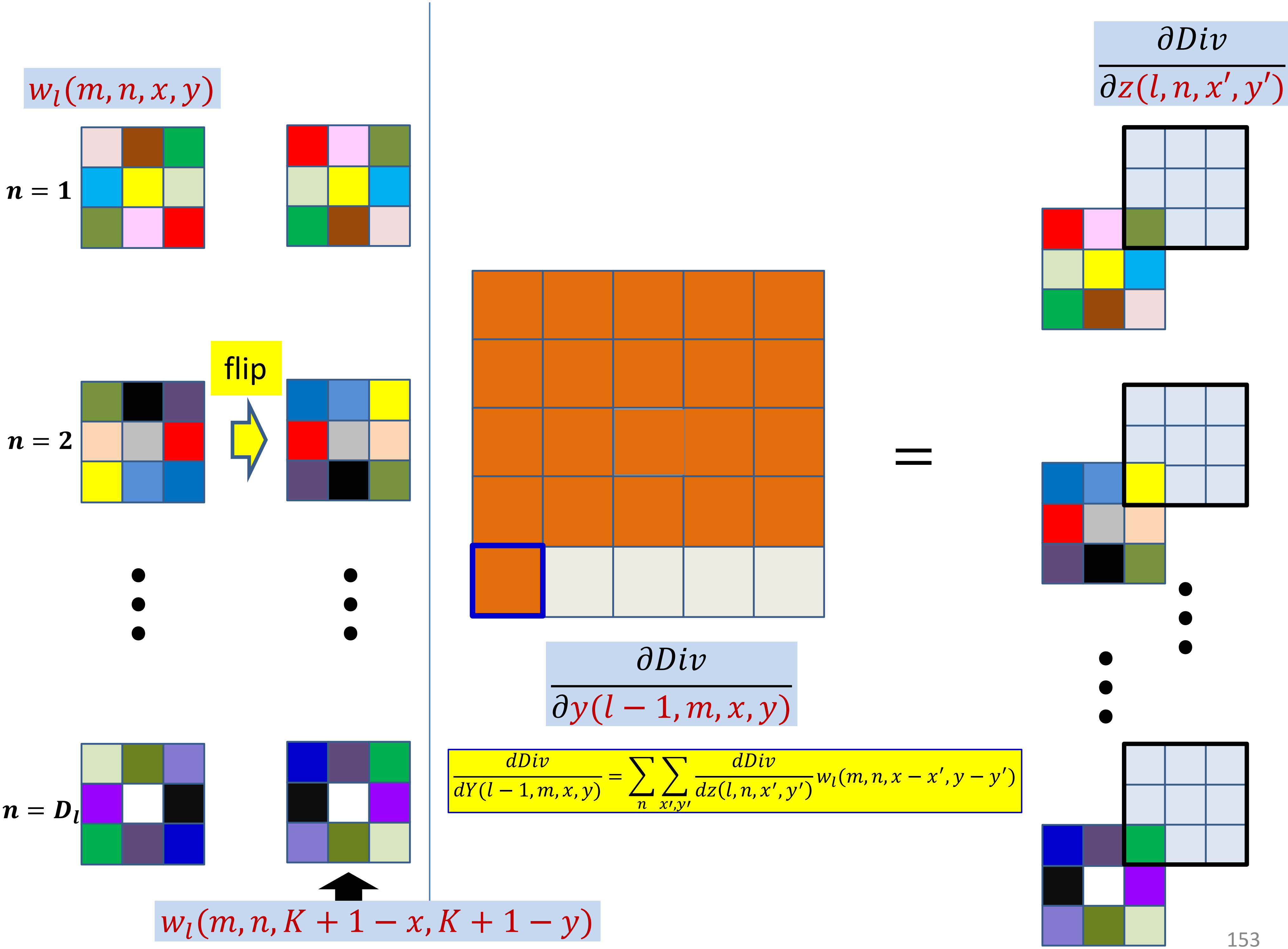

$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$=$
$\frac{\partial Div}{\partial z(l,n,x',y')}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$



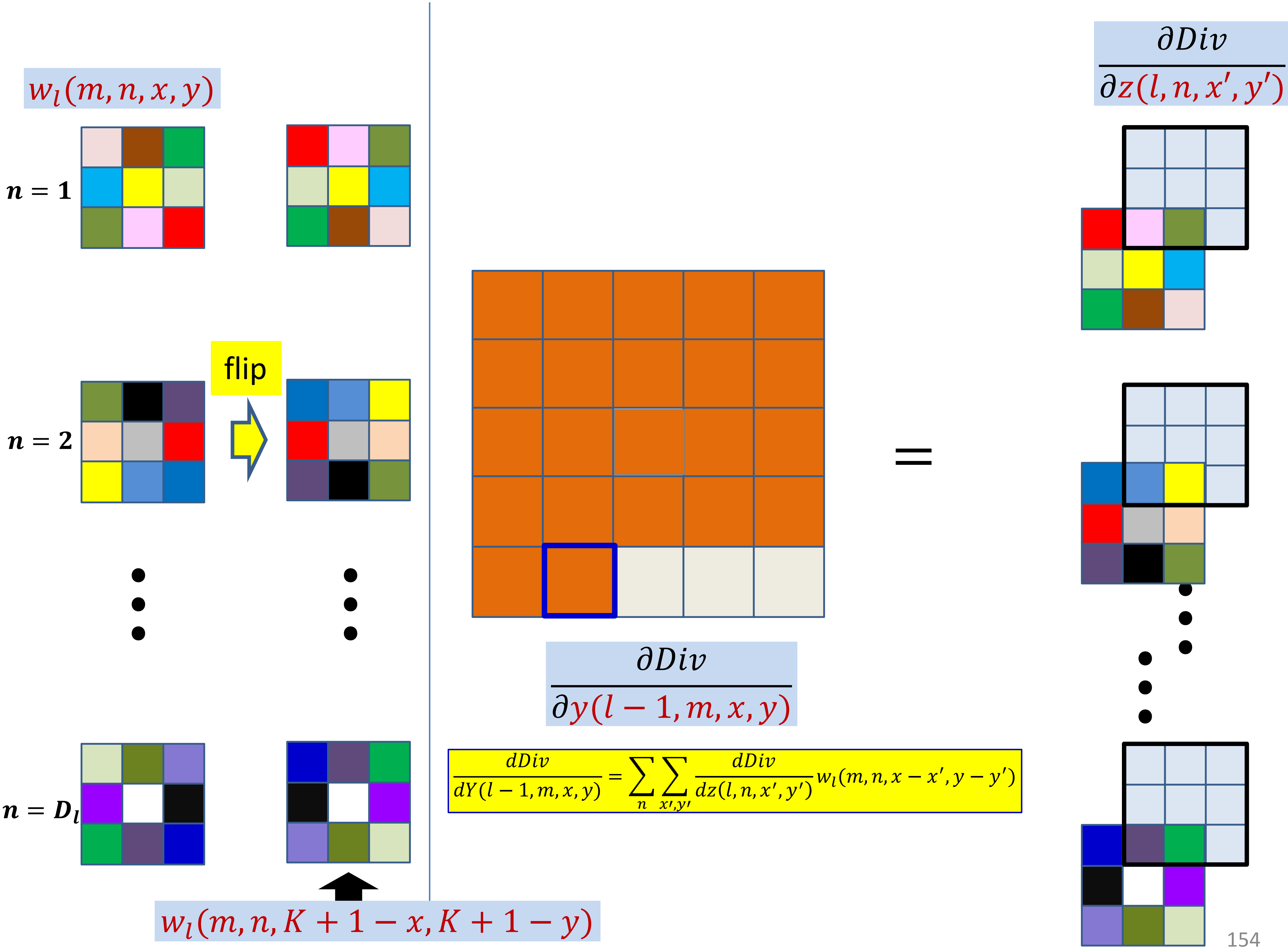
$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$



$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

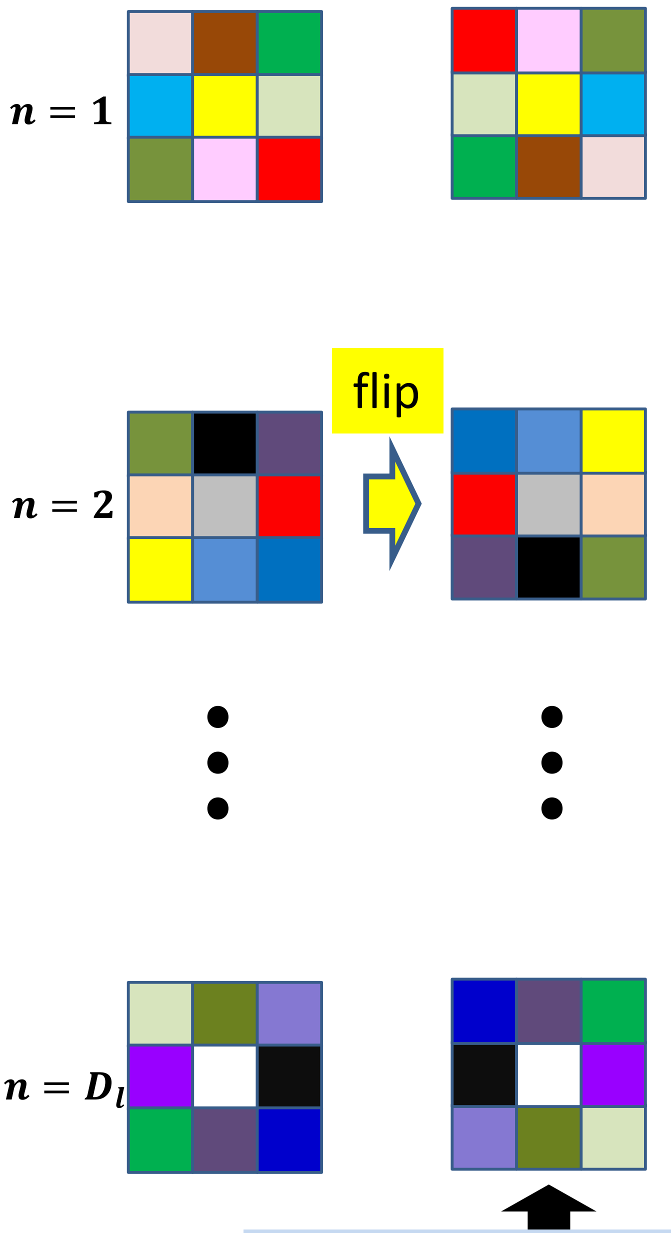


$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$=$
$\frac{\partial Div}{\partial z(l,n,x',y')}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$

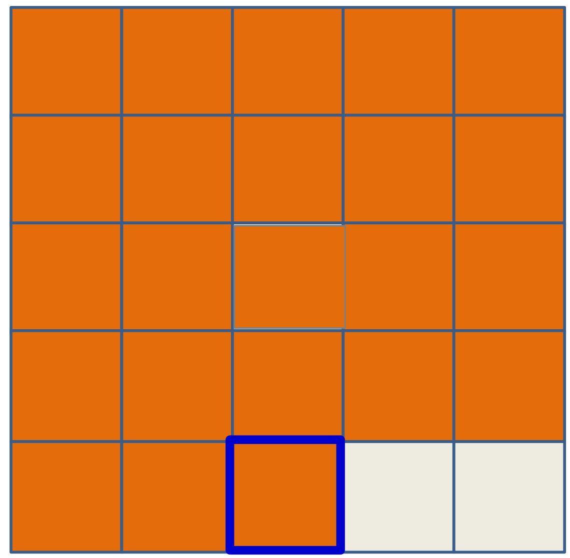


$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n} \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

$w_l(m,n,x,y)$



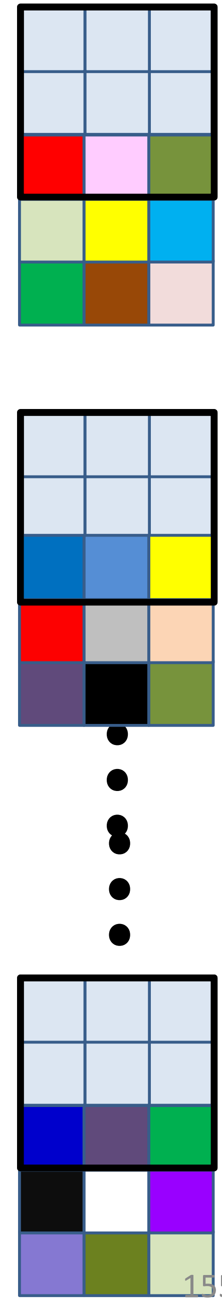
$w_l(m,n,K+1-x,K+1-y)$



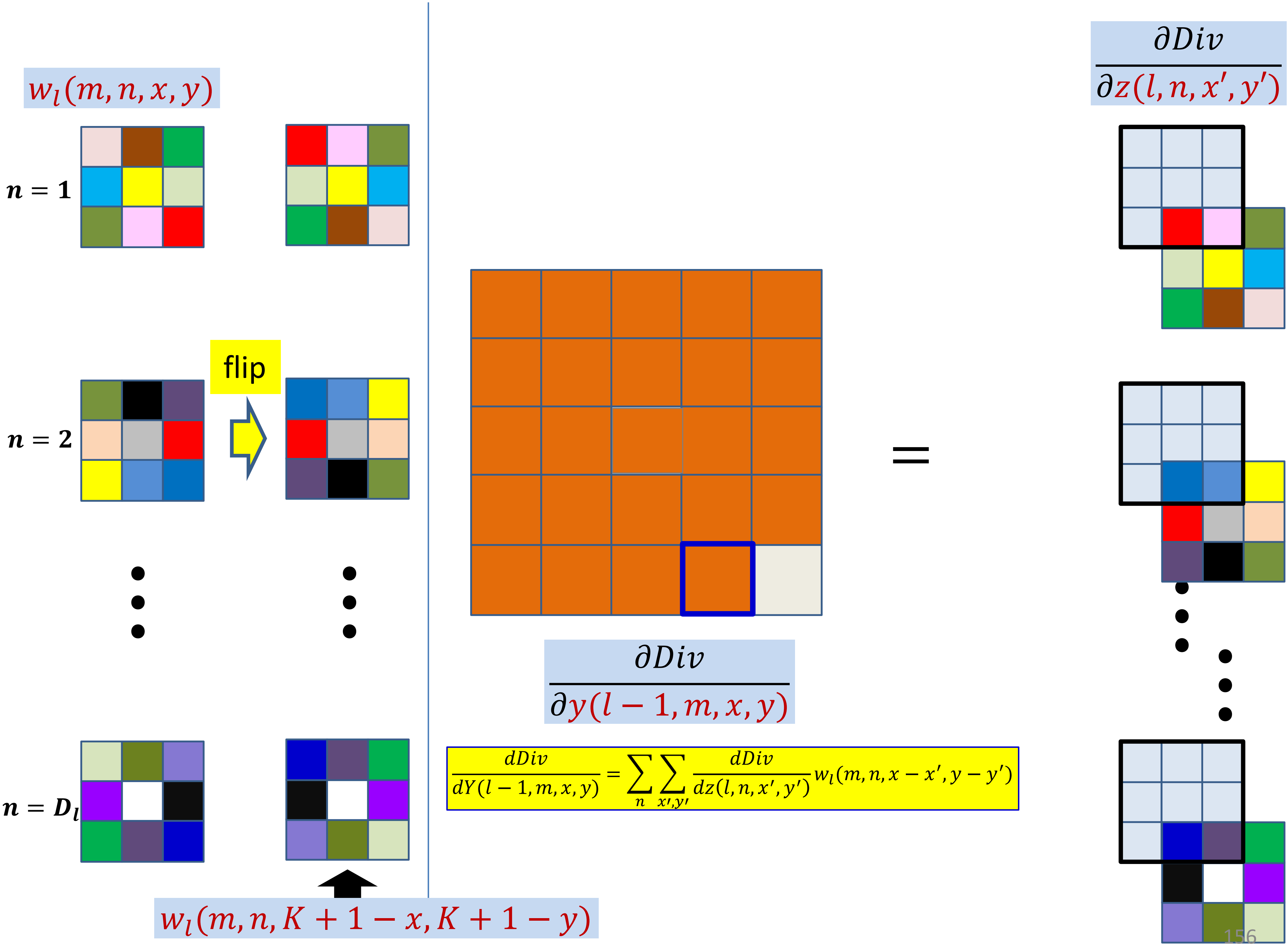
$$\frac{\partial Div}{\partial y(l-1,m,x,y)}$$

$=$

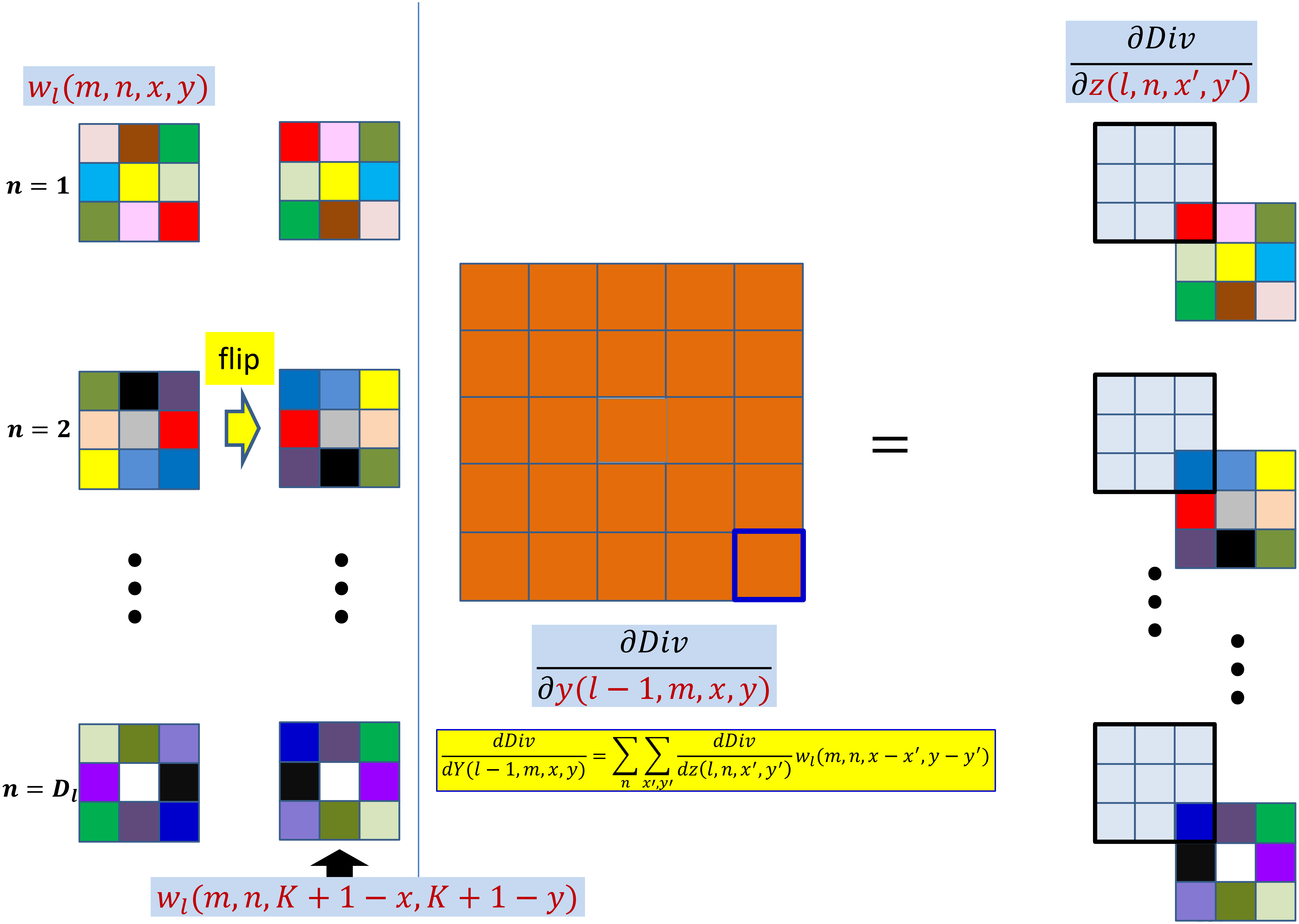
$$\frac{\partial Div}{\partial z(l,n,x',y')}$$



$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

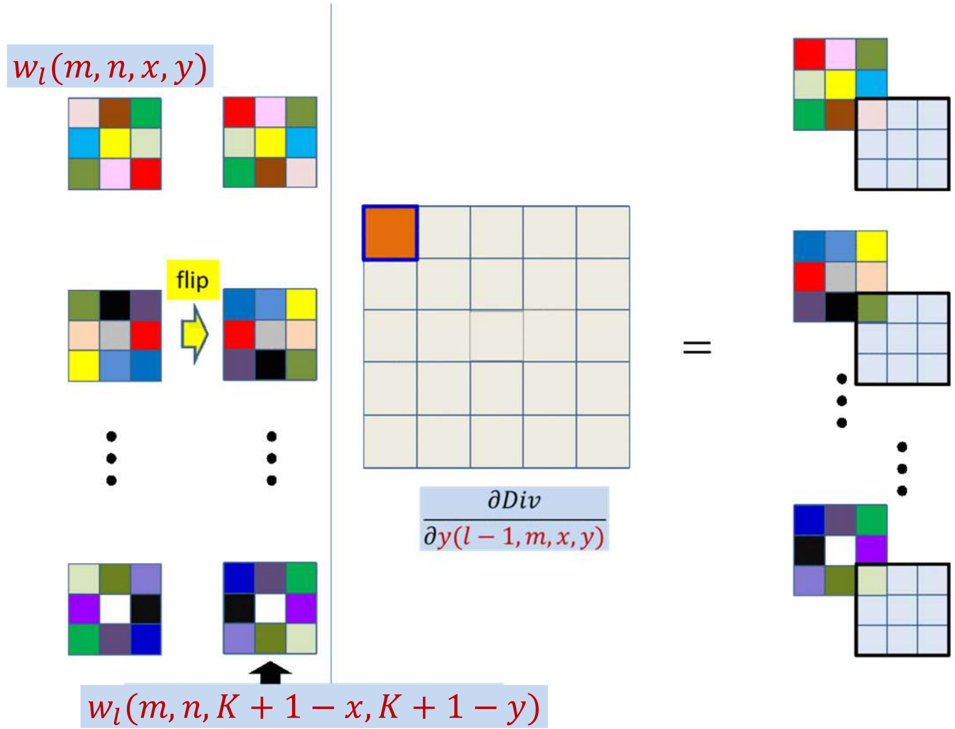


$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n}\sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$



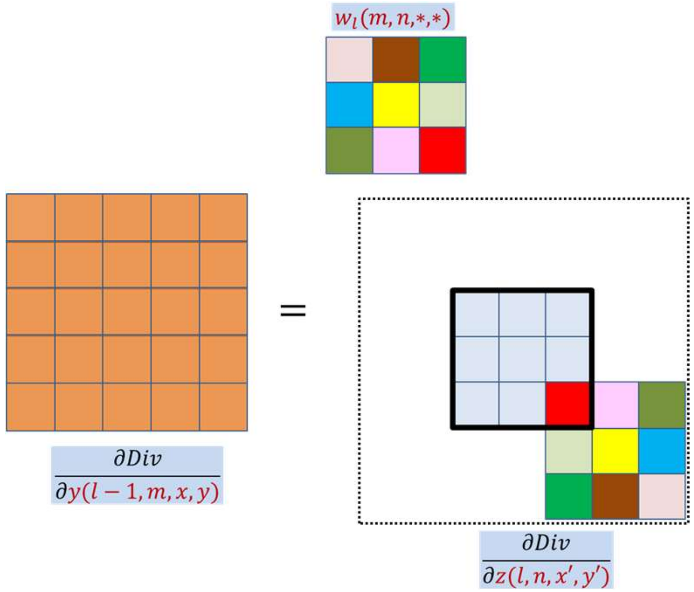
$w_l(m,n,x,y)$
$n = 1$
$n = 2$
flip
$n = D_l$
$w_l(m,n,K+1-x,K+1-y)$
$\frac{\partial Div}{\partial z(l,n,x',y')}$
=
$\frac{\partial Div}{\partial y(l-1,m,x,y)}$
$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$

Computing the derivative for $Y(l-1, m)$



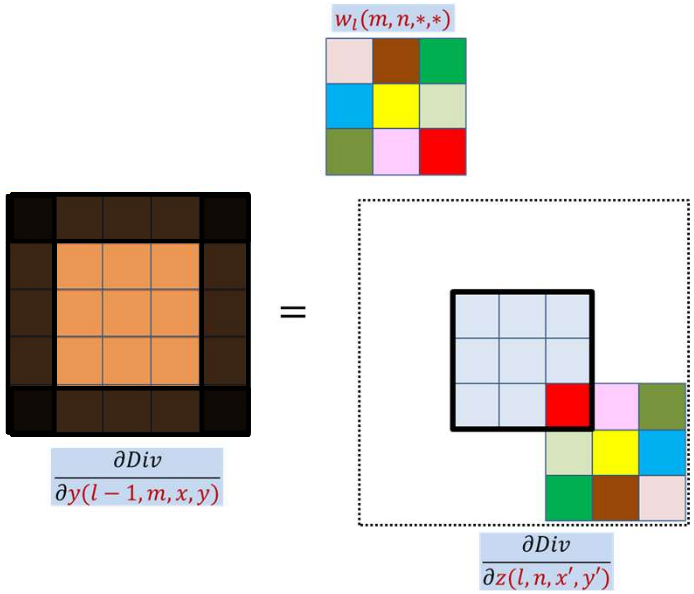
- This is just a convolution of the zero-padded $\frac{\partial Div}{\partial z(l,n,x,y)}$ maps by the transposed and flipped filter
  - After zero padding it first with $K-1$ zeros on every side

The size of the Y-derivative map



- We continue to compute elements for the derivative Y map as long as the (flipped) filter has at least one element in the (unpadded) derivative Zmap
  - I.e. so long as the Y derivative is non-zero
- The size of the Y derivative map will be $(H + K - 1) \times (W + K - 1)$
  - H and W are the height and width of the Zmap
- This will be the size of the actual Y map that was originally convolved

The size of the Y-derivative map



- If the Y map was zero-padded in the forward pass, the derivative map will be the size of the *zero-padded* map
  - The zero padding regions must be deleted before further backprop

Poll 3

Select all statements that are true about how to compute the derivative of the divergence w.r.t lth layer activation maps by backpropagation

- To compute the derivative w.r.t. the mth activation map of the lth convolutional layer, we must select the mth "planes" of all the (l+1)th layer filters
- The selected filter planes must be flipped left-right and up-down
- They must convolve the derivative (maps) for the (l+1)th layer affine values
- The output of the convolution must be flipped back left-right and up-down

Poll 3

Select all statements that are true about how to compute the derivative of the divergence w.r.t lth layer activation maps by backpropagation

- **To compute the derivative w.r.t. the mth activation map of the lth convolutional layer, we must select the mth "planes" of all the (l+1)th layer filters**
- **The selected filter planes must be flipped left-right and up-down**
- **They must convolve the derivative (maps) for the (l+1)th layer affine values**
- The output of the convolution must be flipped back left-right and up-down

Overall algorithm for computing derivatives w.r.t. $Y(l-1)$

- Given the derivatives $\frac{dDiv}{dz(l,n,x,y)}$

- Compute derivatives using:

$$\frac{dDiv}{dY(l-1,m,x,y)} = \sum_{n}\sum_{x',y'} \frac{dDiv}{dz(l,n,x',y')} w_l(m,n,x-x',y-y')$$

Can be computed by convolution with flipped filter

$\boldsymbol{l}$ is layer index

Derivatives for a single layer l: Vector notation

```
# The weight W(l,m)is a 3D Dl-1xKlxKl
# Assuming dz has already been obtained via backprop

dzpad = zeros(Dlx(Hl+2(Kl-1))x(Wl+2(Kl-1))) # zeropad
for j = 1:Dl
    for i = 1:Dl-1   # Transpose and flip
        Wflip(i,j,:,:) = flipLeftRight(flipUpDown(W(l,i,j,:,:)))
    dzpad(j,Kl:Kl+Hl-1,Kl:Kl+Wl-1) = dz(l,j,:,:) #center map
end
```

```
for j = 1:Dl-1
  for x = 1:Wl-1
    for y = 1:Hl-1
       segment = dzpad(:, x:x+Kl-1, y:y+Kl-1) #3D tensor
       dy(l-1,j,x,y) = Wflip.segment #tensor inner prod.
```

Backpropagating through affine map

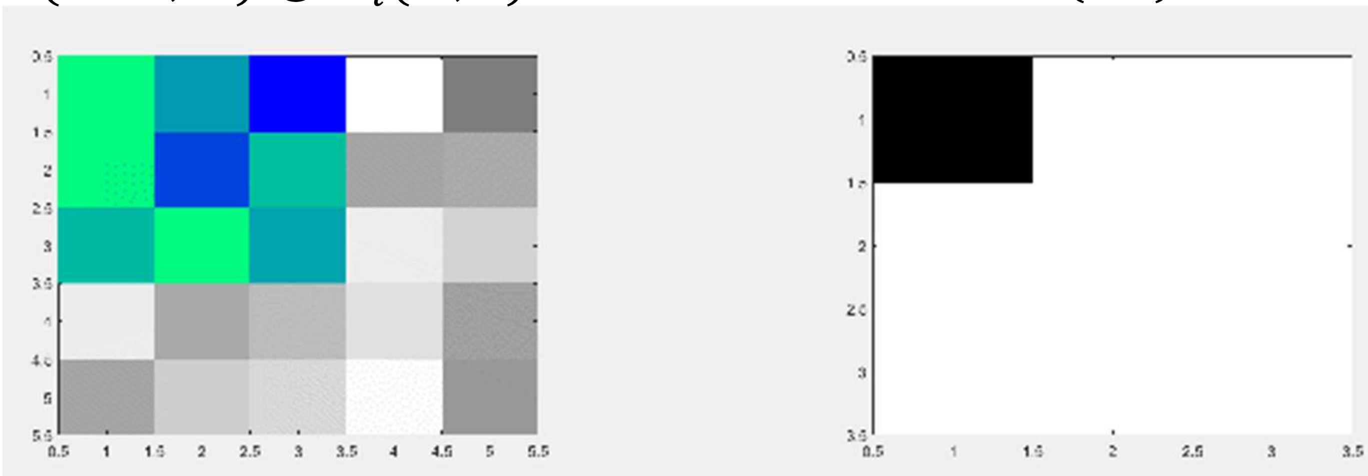
- Forward affine computation:
  - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l-1, m, x, y)$ and filters $w_l(m, n, x, y)$

- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
  - Compute derivative w.r.t. $y(l-1, m, x, y)$
  - Compute derivative w.r.t. $w_l(m, n, x, y)$

The derivatives for the filters

$Y(l-1,m) \otimes w_l(m,n)$ $\quad\quad$ $Z(l,n)$

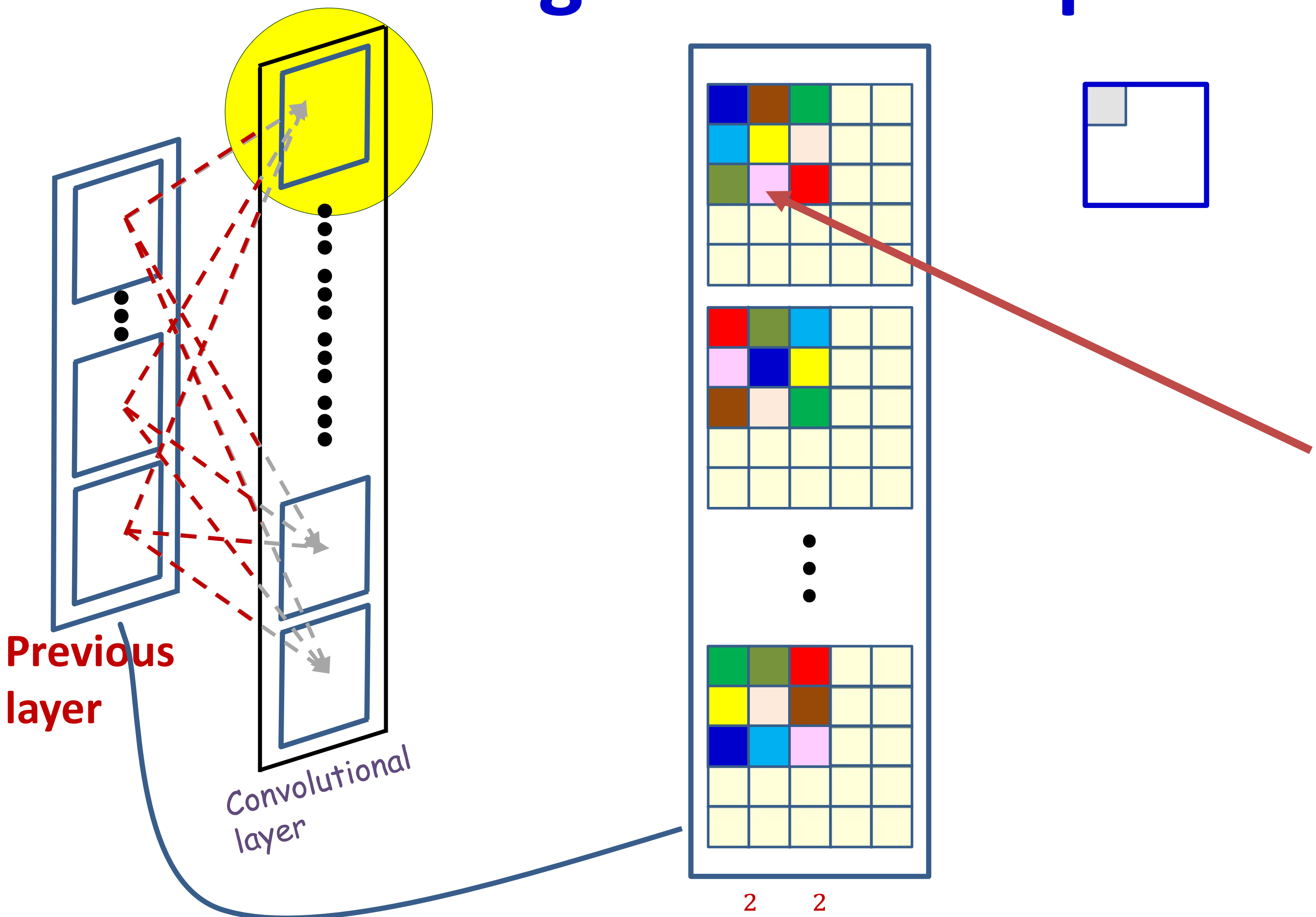
l **is layer index**



$$z(l,n,x,y) = \sum_m \sum_{x',y'} w_l(m,n,x',y')y(l-1,m,x+x',y+y') + b_l(n)$$

- Each ***filter component*** $w_l(m,n,x',y')$ affects several $z(l,n,x,y)$ but only within a *single* output affine ($z(l,n,*,*)$) map/channel
  - And is also linked to several $y(l-1,m,x,y)$ but only within a single input channel $y(l-1,m,*,*)$
    - A single filter channel $w_l(m,n,*,*)$ connects $y(l-1,m,*,*)$ to $z(l,n,*,*)$
  - Consider the contribution of one filter component: $w_l(m,n,i,j)$ (e.g. $w_l(m,n,1,2)$) in the above animation for illustration

Convolution: the contribution of a single filter component

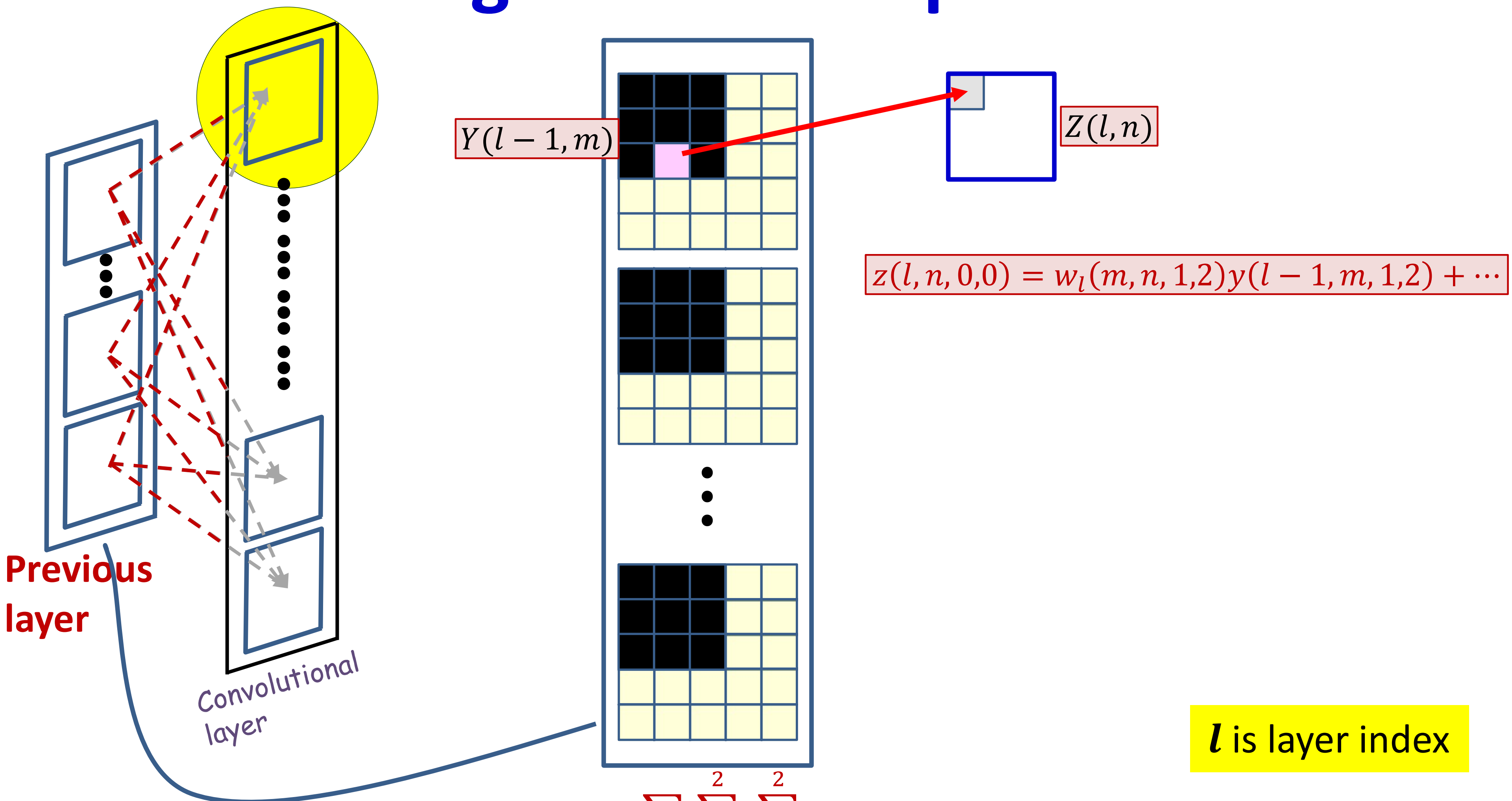


l **is layer index**

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each ***filter component*** $w_l(m, \mathbf{n}, i, j)$ affects several $z(l,n,x,y)$ within the nth output affine map

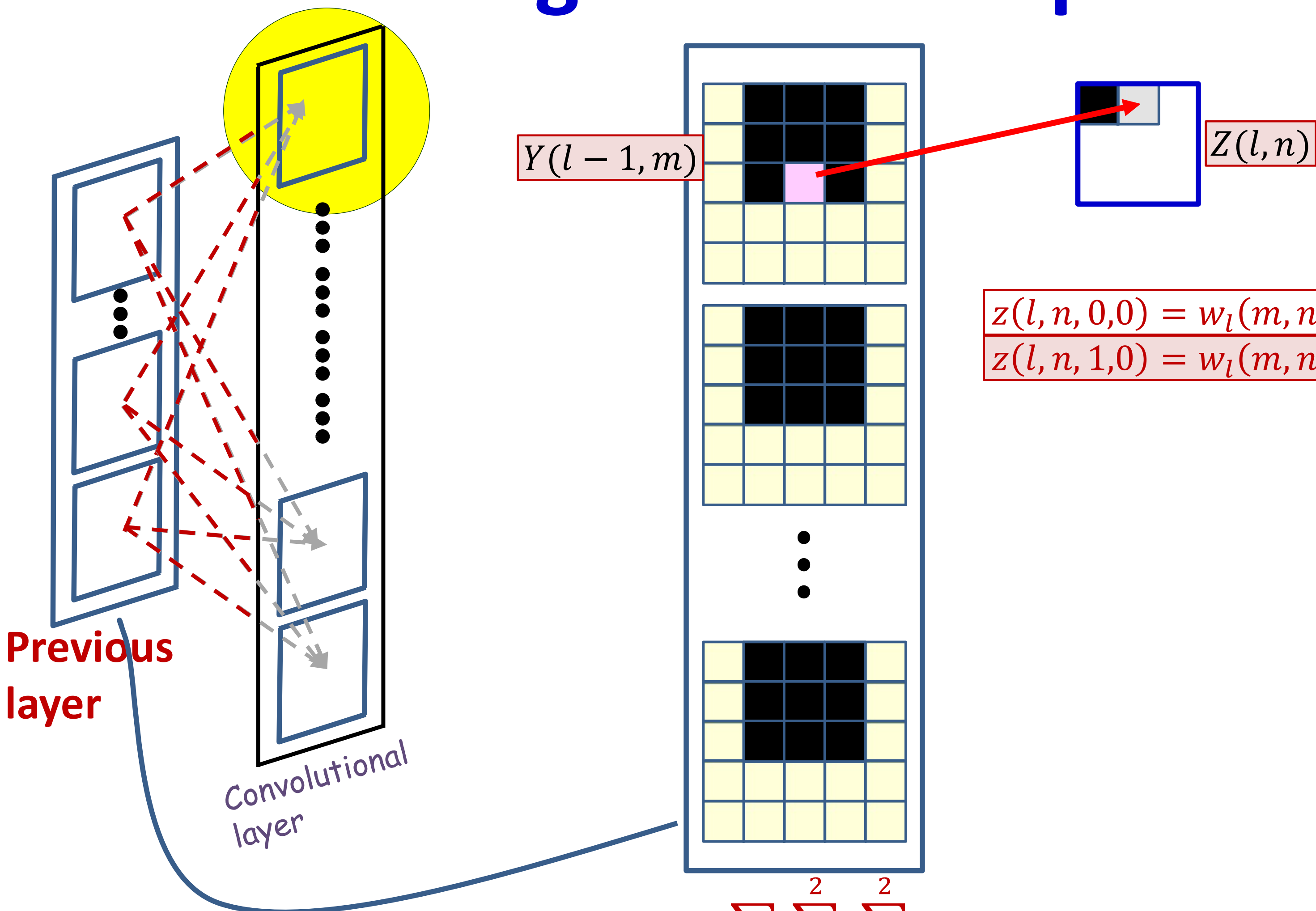
Convolution: the contribution of a single filter component



$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1{,}2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1{,}2)y(l-1,m,1{,}2) + \cdots$

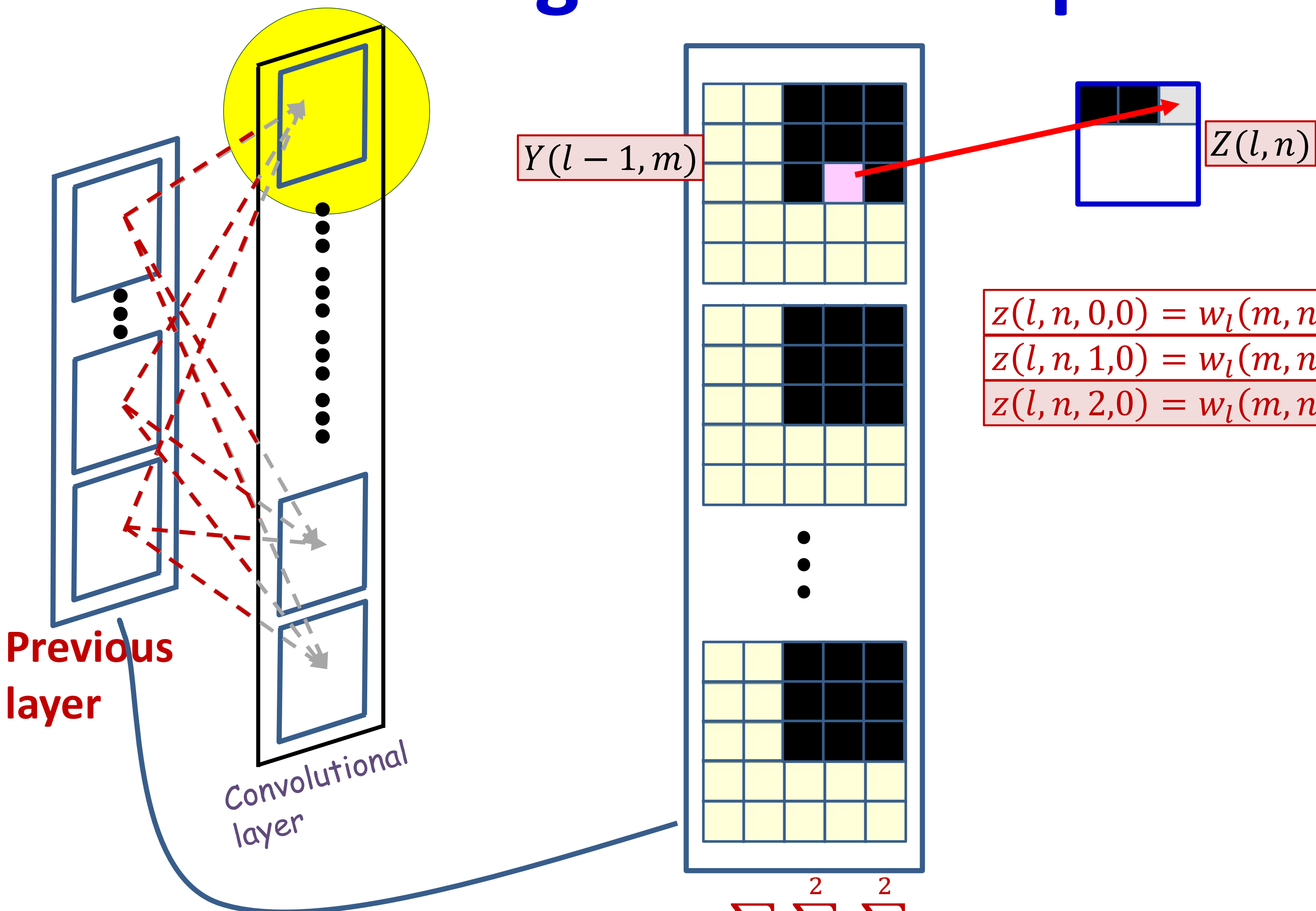
$z(l,n,1{,}0) = w_l(m,n,1{,}2)y(l-1,m,2{,}2) + \cdots$

l is layer index

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1{,}2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$

$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$

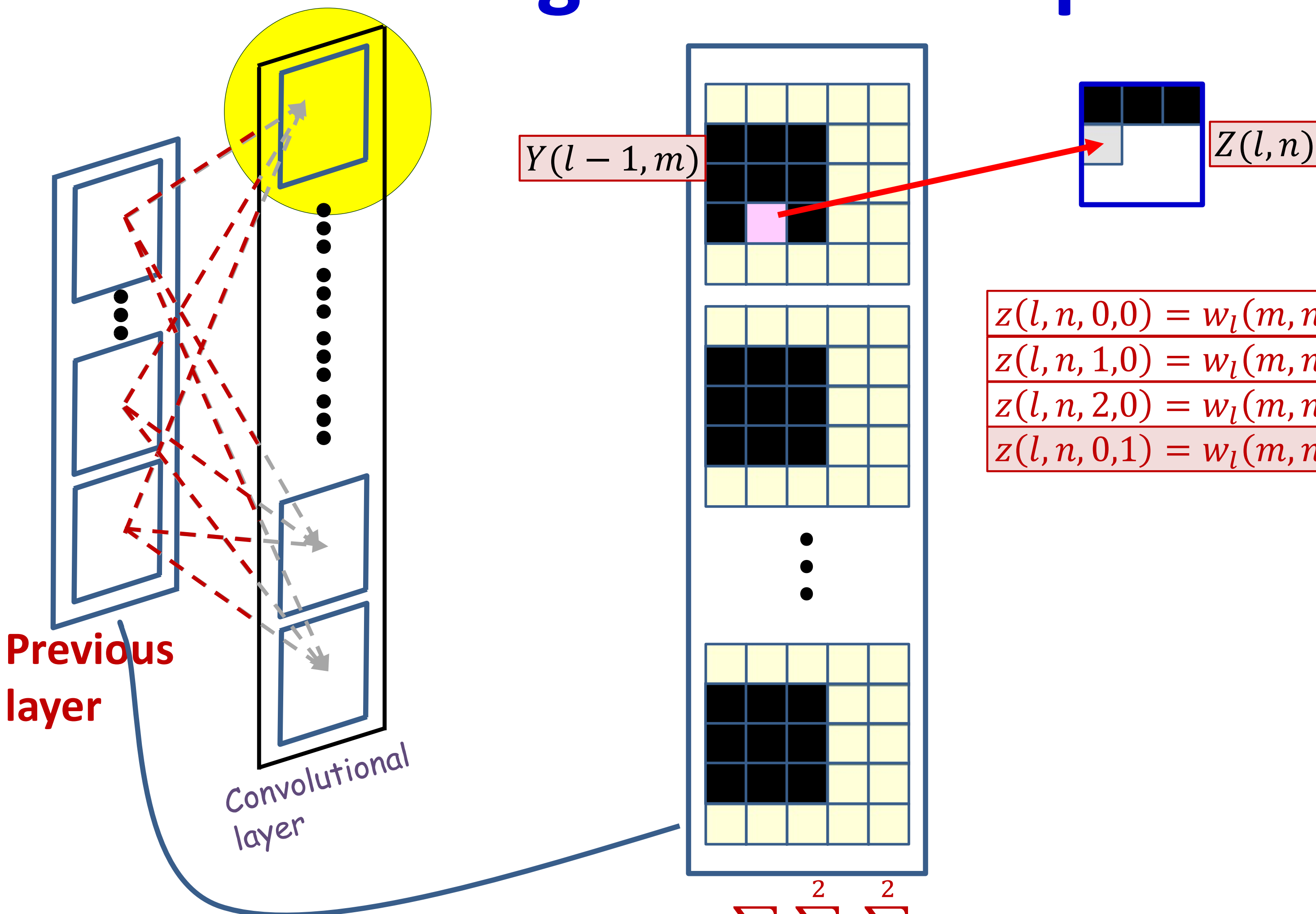
$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$

$\boldsymbol{l}$ is layer index

$$z(l,n,x,y) = \sum_m \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



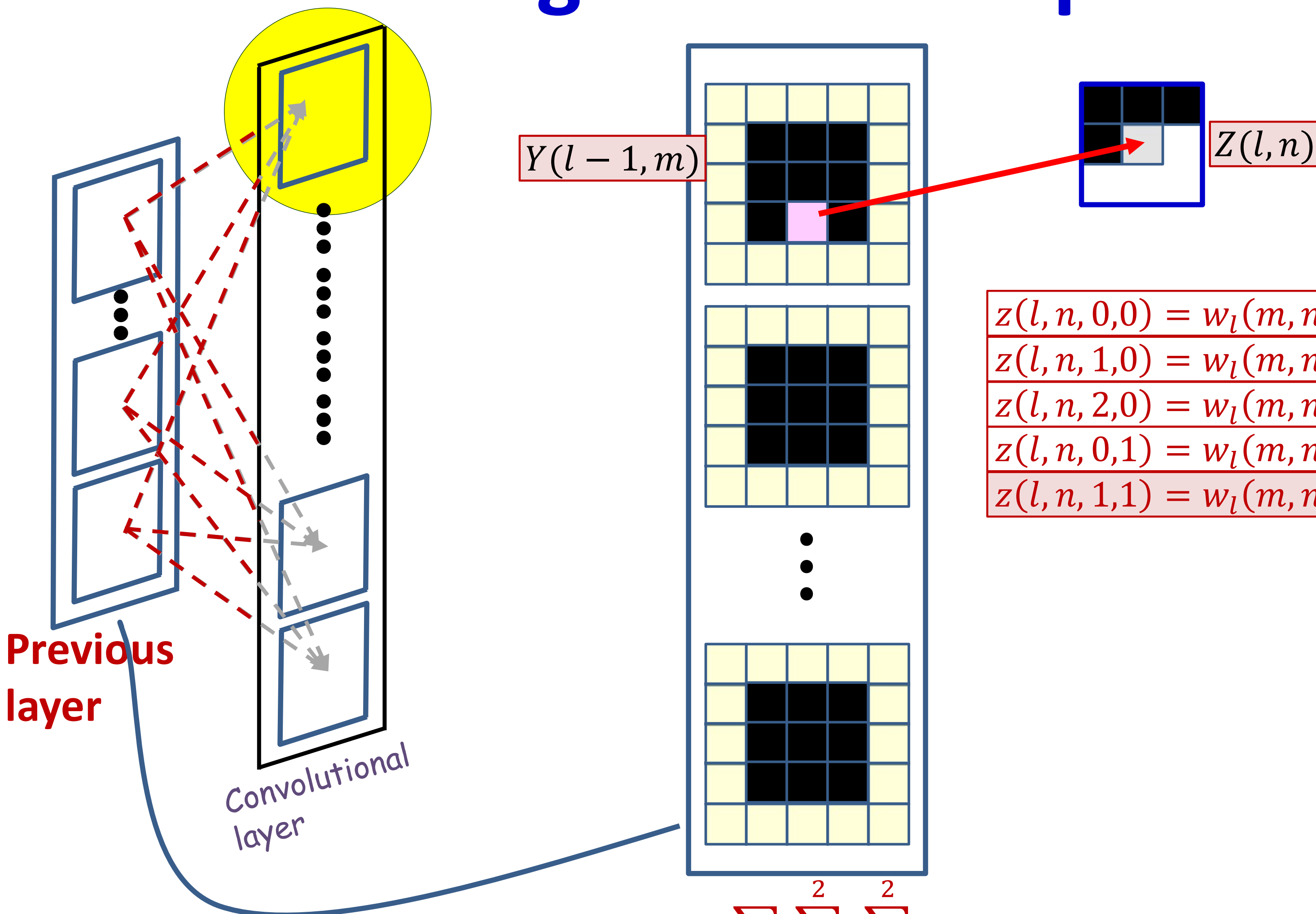
$$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$$
$$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$$
$$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$$
$$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$$

l is layer index

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$

$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$

$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$

$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$

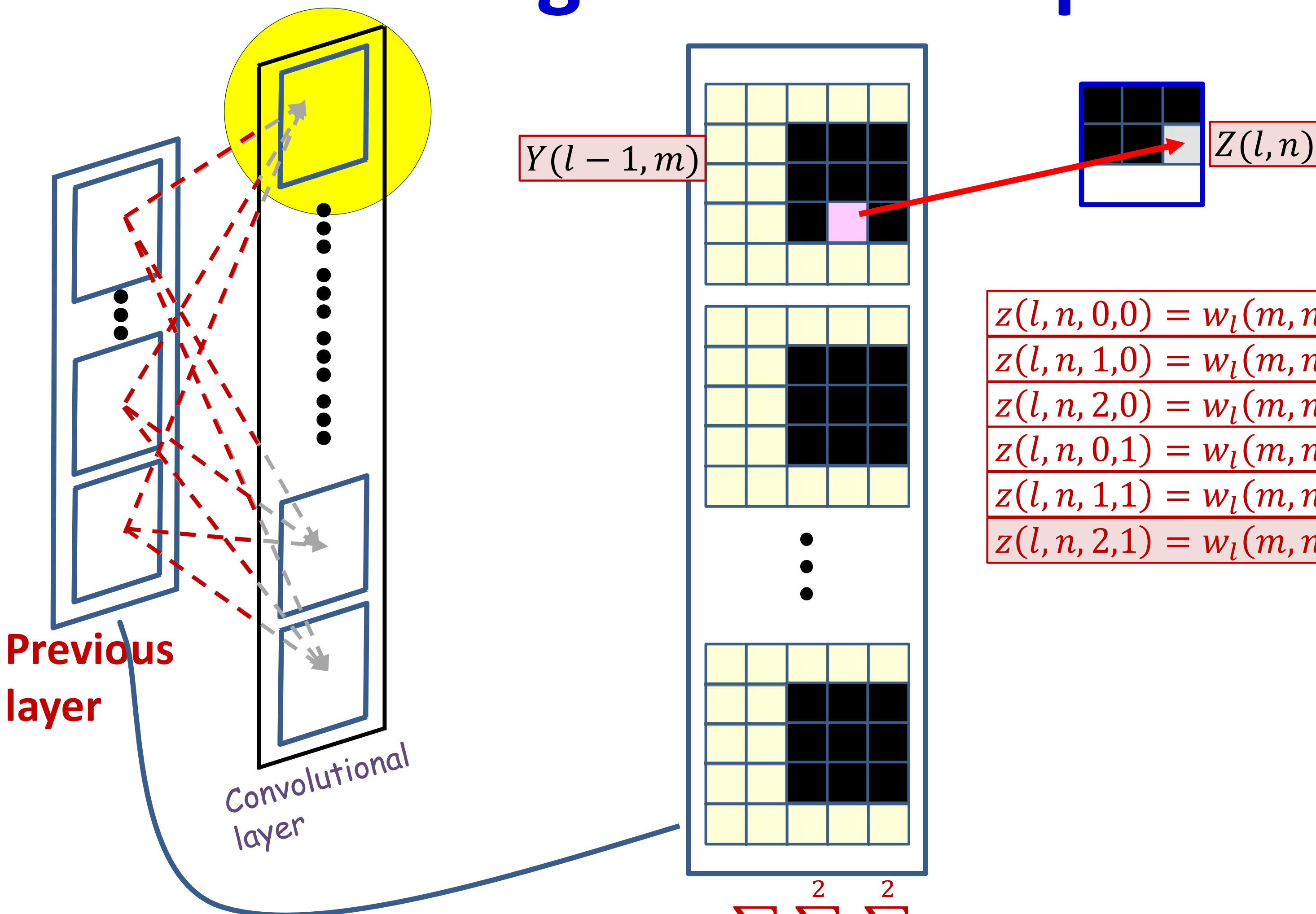
$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$

l is layer index

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



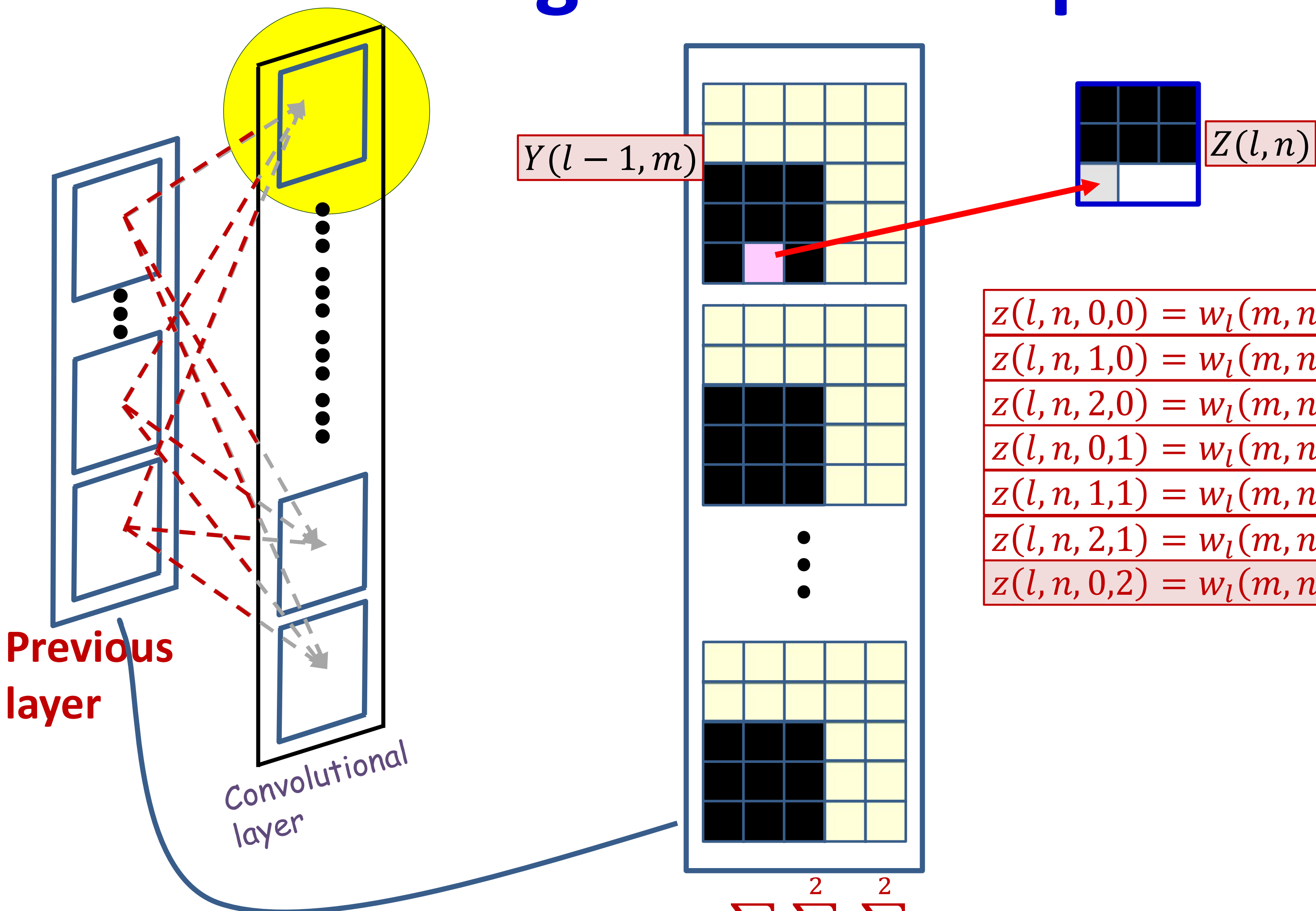
$$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$$
$$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$$
$$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$$
$$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$$
$$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$$
$$z(l,n,2,1) = w_l(m,n,1,2)y(l-1,m,3,3) + \cdots$$

$\boldsymbol{l}$ is layer index

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$

$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$

$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$

$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$

$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$

$z(l,n,2,1) = w_l(m,n,1,2)y(l-1,m,3,3) + \cdots$

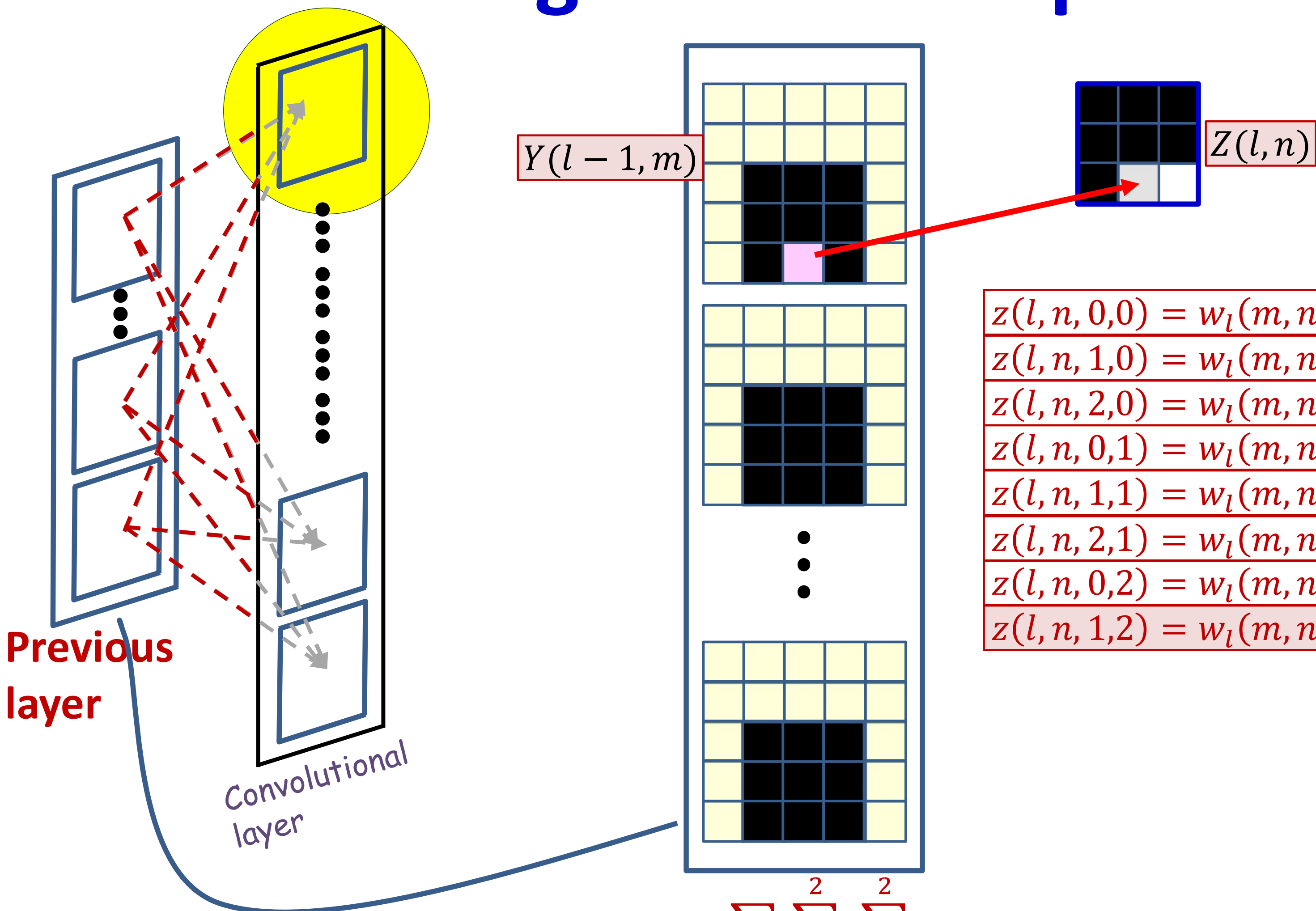
$z(l,n,0,2) = w_l(m,n,1,2)y(l-1,m,1,4) + \cdots$

l **is layer index**

$$z(l,n,x,y) = \sum_m \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



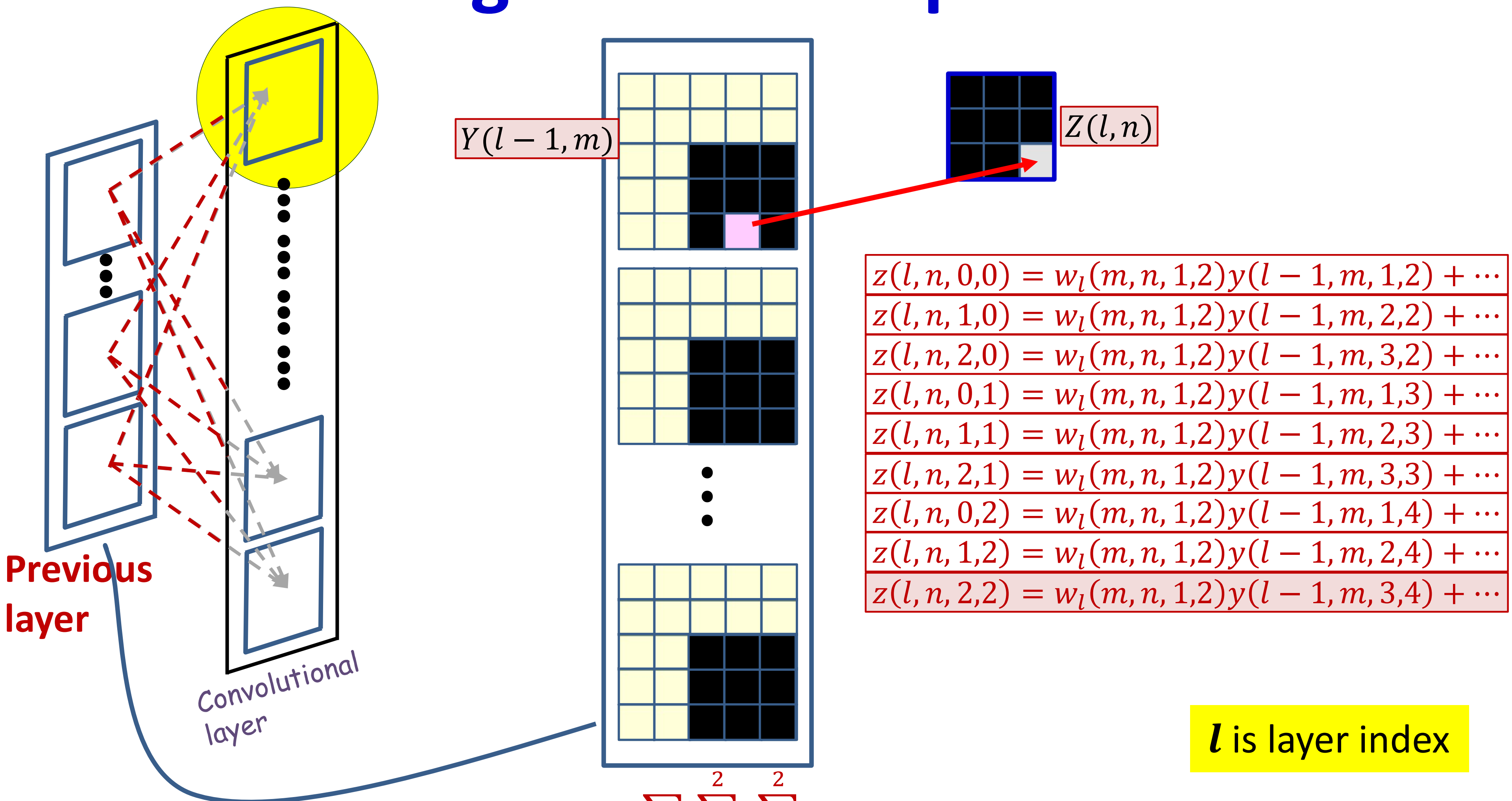
$$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$$
$$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$$
$$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$$
$$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$$
$$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$$
$$z(l,n,2,1) = w_l(m,n,1,2)y(l-1,m,3,3) + \cdots$$
$$z(l,n,0,2) = w_l(m,n,1,2)y(l-1,m,1,4) + \cdots$$
$$z(l,n,1,2) = w_l(m,n,1,2)y(l-1,m,2,4) + \cdots$$

$\boldsymbol{l}$ is layer index

$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

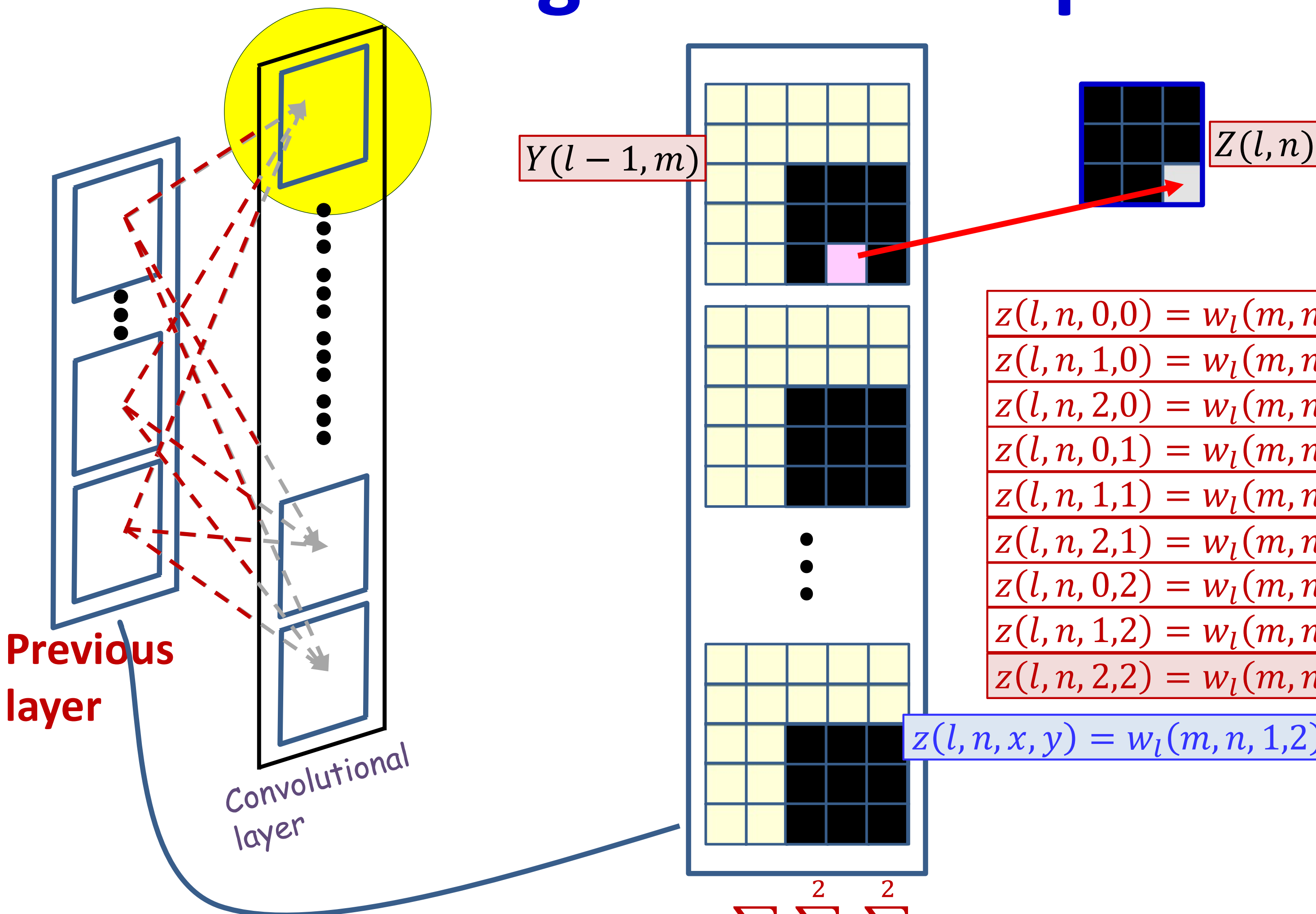
Convolution: the contribution of a single filter component



$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$

$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$

$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$

$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$

$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$

$z(l,n,2,1) = w_l(m,n,1,2)y(l-1,m,3,3) + \cdots$

$z(l,n,0,2) = w_l(m,n,1,2)y(l-1,m,1,4) + \cdots$

$z(l,n,1,2) = w_l(m,n,1,2)y(l-1,m,2,4) + \cdots$

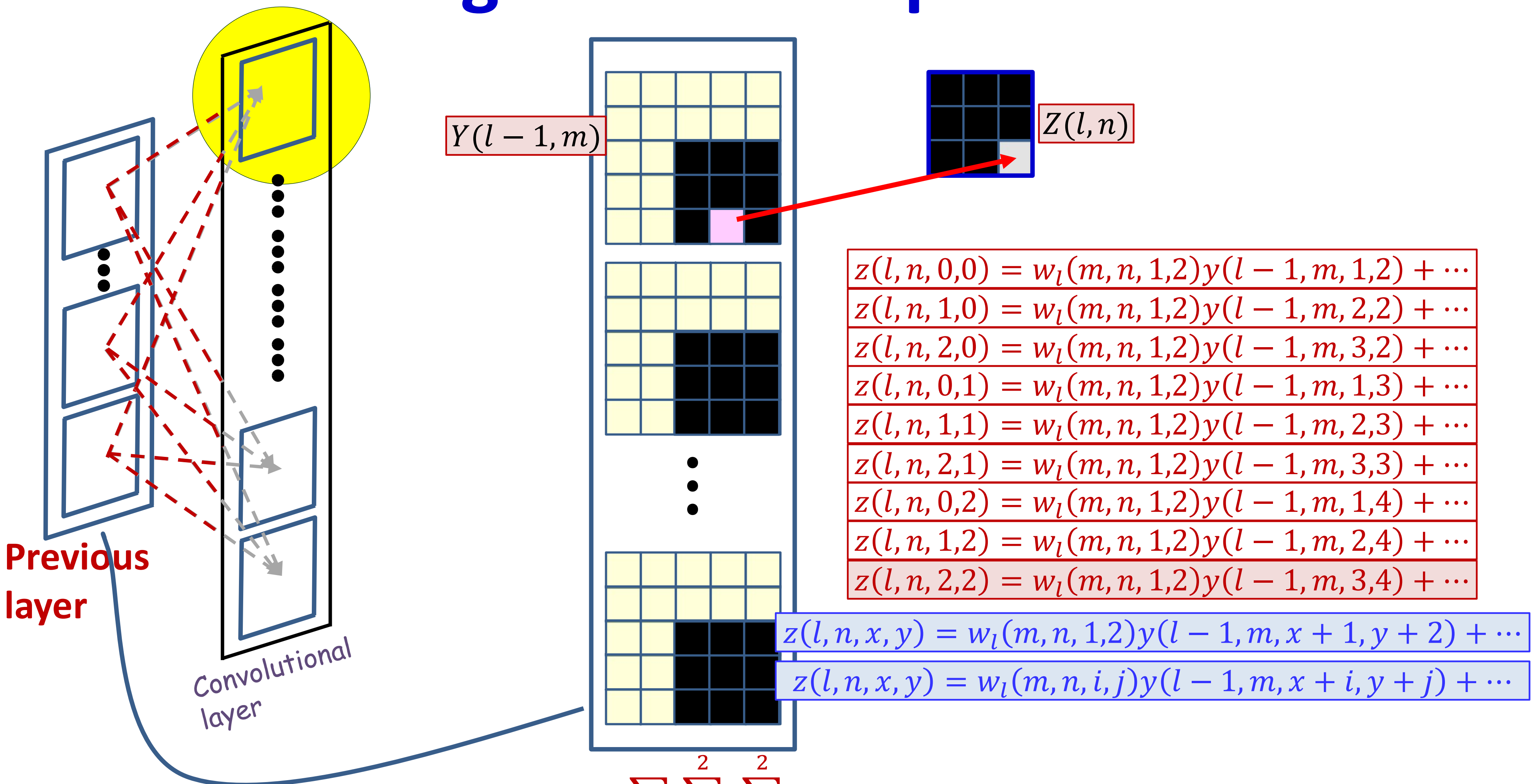
$z(l,n,2,2) = w_l(m,n,1,2)y(l-1,m,3,4) + \cdots$

$z(l,n,x,y) = w_l(m,n,1,2)y(l-1,m,x+1,y+2) + \cdots$

$$z(l,n,x,y) = \sum_m \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

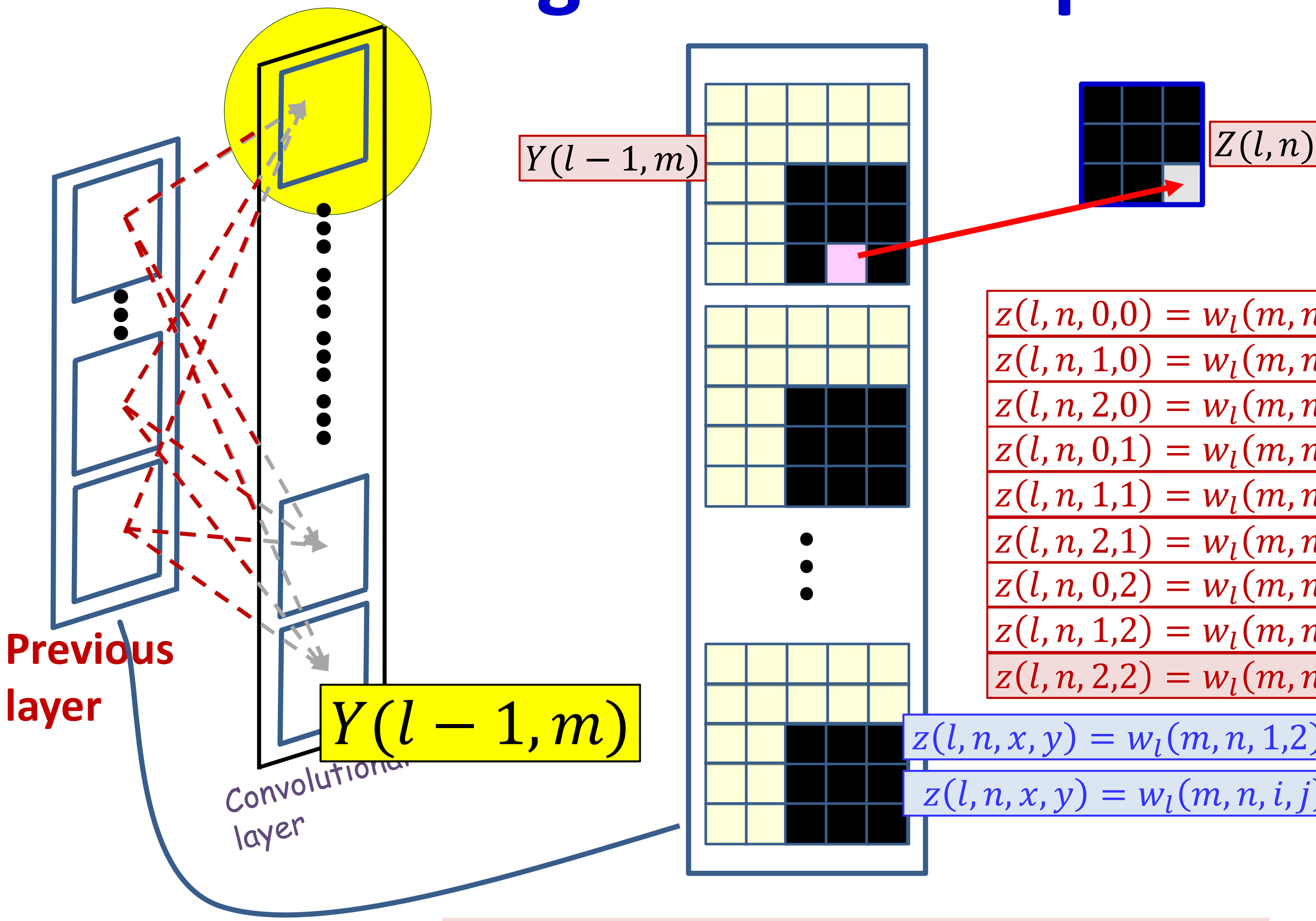
Convolution: the contribution of a single filter component



$$z(l,n,x,y) = \sum_{m} \sum_{x'=0}^{2} \sum_{y'=0}^{2} w_l(m,n,x',y')y(l-1,m,x+x',y+x') + b_l(n)$$

- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$ in the nth output map
  - Consider the contribution of one filter component: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single filter component



$z(l,n,0,0) = w_l(m,n,1,2)y(l-1,m,1,2) + \cdots$

$z(l,n,1,0) = w_l(m,n,1,2)y(l-1,m,2,2) + \cdots$

$z(l,n,2,0) = w_l(m,n,1,2)y(l-1,m,3,2) + \cdots$

$z(l,n,0,1) = w_l(m,n,1,2)y(l-1,m,1,3) + \cdots$

$z(l,n,1,1) = w_l(m,n,1,2)y(l-1,m,2,3) + \cdots$

$z(l,n,2,1) = w_l(m,n,1,2)y(l-1,m,3,3) + \cdots$

$z(l,n,0,2) = w_l(m,n,1,2)y(l-1,m,1,4) + \cdots$

$z(l,n,1,2) = w_l(m,n,1,2)y(l-1,m,2,4) + \cdots$

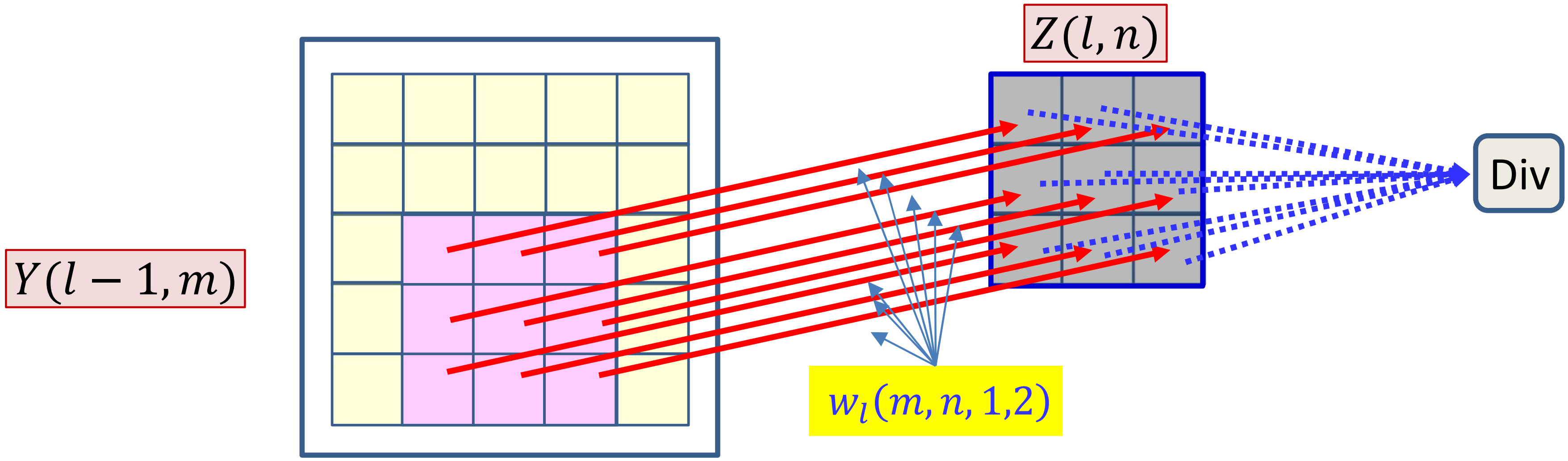
$z(l,n,2,2) = w_l(m,n,1,2)y(l-1,m,3,4) + \cdots$

$z(l,n,x,y) = w_l(m,n,1,2)y(l-1,m,x+1,y+2) + \cdots$

$z(l,n,x,y) = w_l(m,n,i,j)y(l-1,m,x+i,y+j) + \cdots$

$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

The derivative for a single filter component



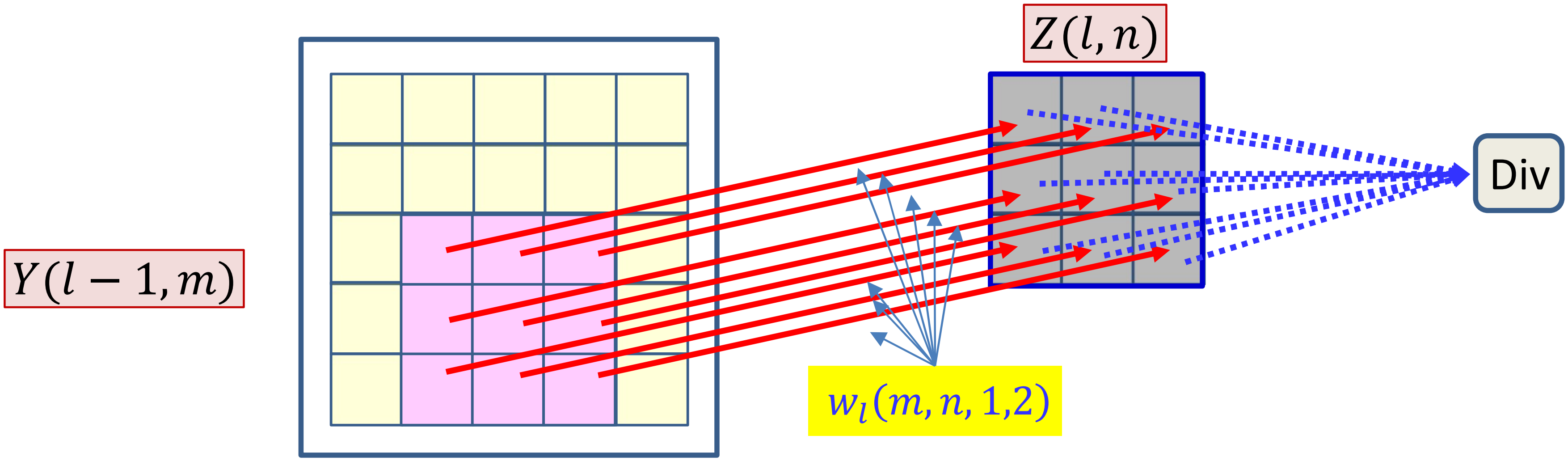
- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

- The final divergence is influenced by *every* $z(l,n,x,y)$
- The derivative of the divergence w.r.t $w_l(m,n,i,j)$ must sum over all $z(l,n,x,y)$ terms it influences

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} \frac{dz(l,n,x,y)}{dw_l\,(m,n,i,j)}$$

The derivative for a single filter component



- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

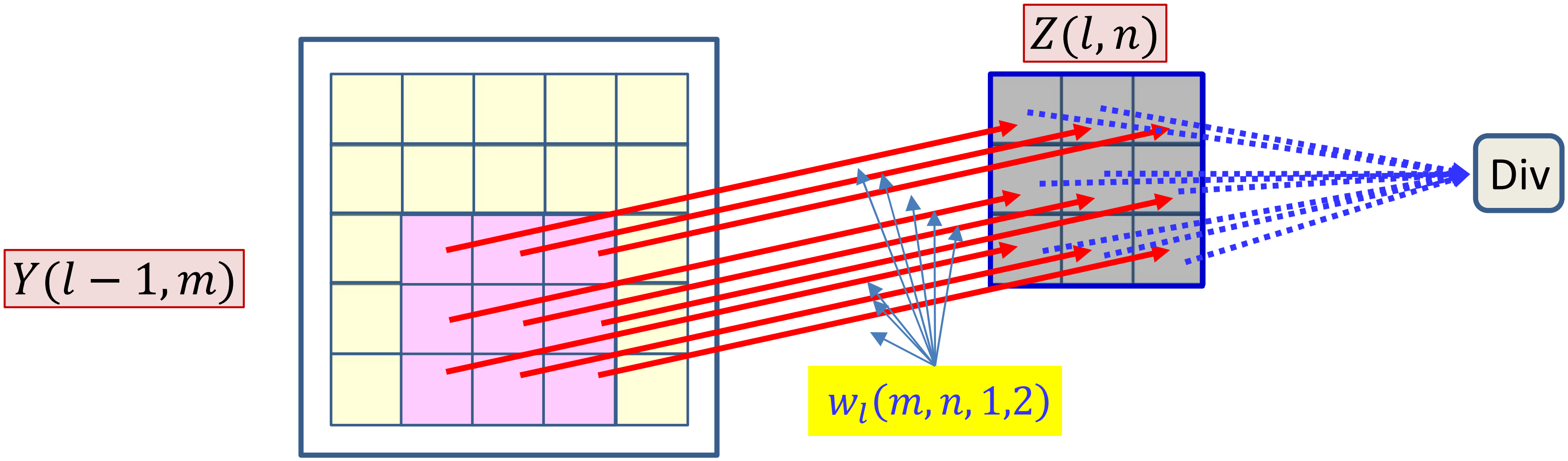
$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

- The final divergence is influenced by *every* $z(l,n,x,y)$
- The derivati … r.t $w_l(m,n,i,j)$ must sum over all $z(l,n,x,y)$ terms it influences

Already computed

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} \frac{dz(l,n,x,y)}{dw_l\,(m,n,i,j)}$$

The derivative for a single filter component



- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

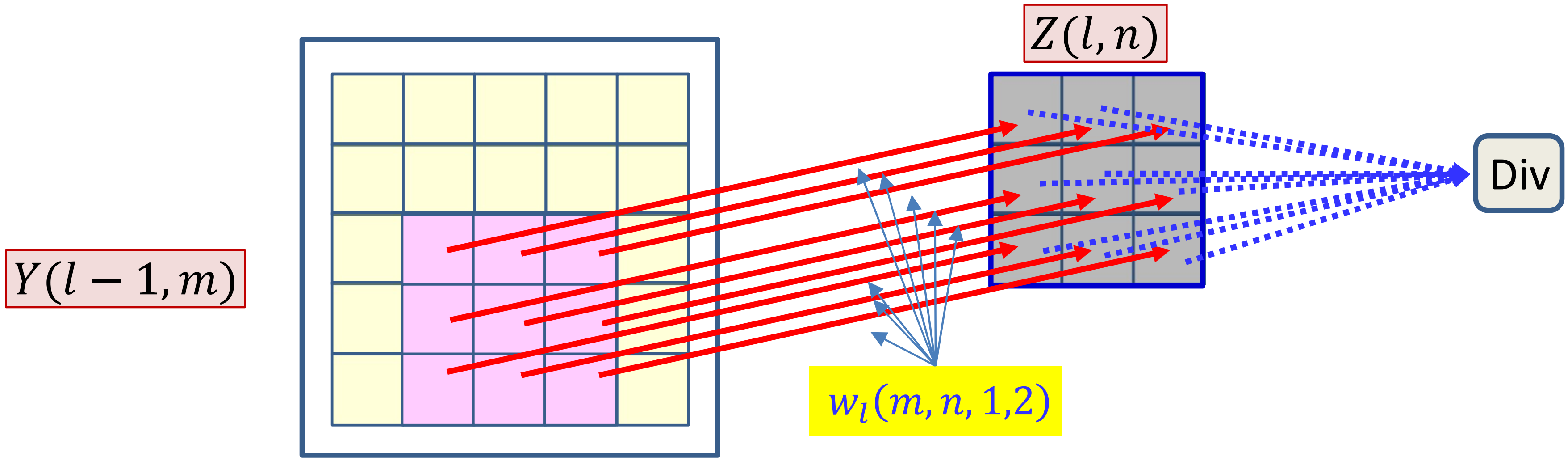
$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

- The final divergence is influenced by *every* $z(l,n,x,y)$
- The derivati[...] w.r.t $w_l(m,n,i,j)$ must sum over all $z(l,n,x,y)$ terms it influences

Already computed

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} \frac{dz(l,n,x,y)}{dw_l\,(m,n,i,j)}$$

The derivative for a single filter component



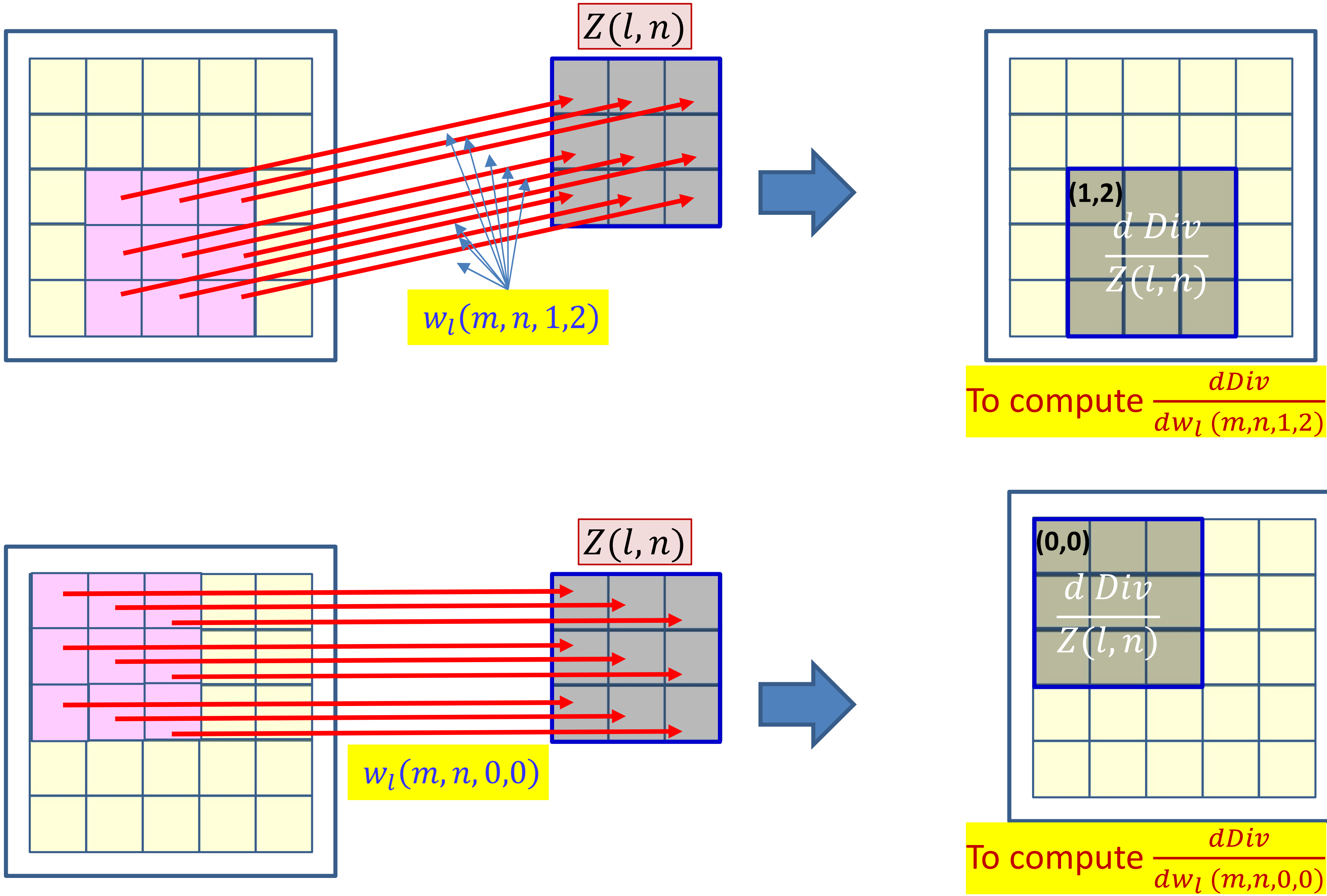
- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

- The final divergence is influenced by *every* $z(l,n,x,y)$
- The derivative of the divergence w.r.t $w_l(m,n,i,j)$ must sum over all $z(l,n,x,y)$ terms it influences

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

The derivative for a single filter component



But this too is a convolution

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1, m, x+i, y+j)$$

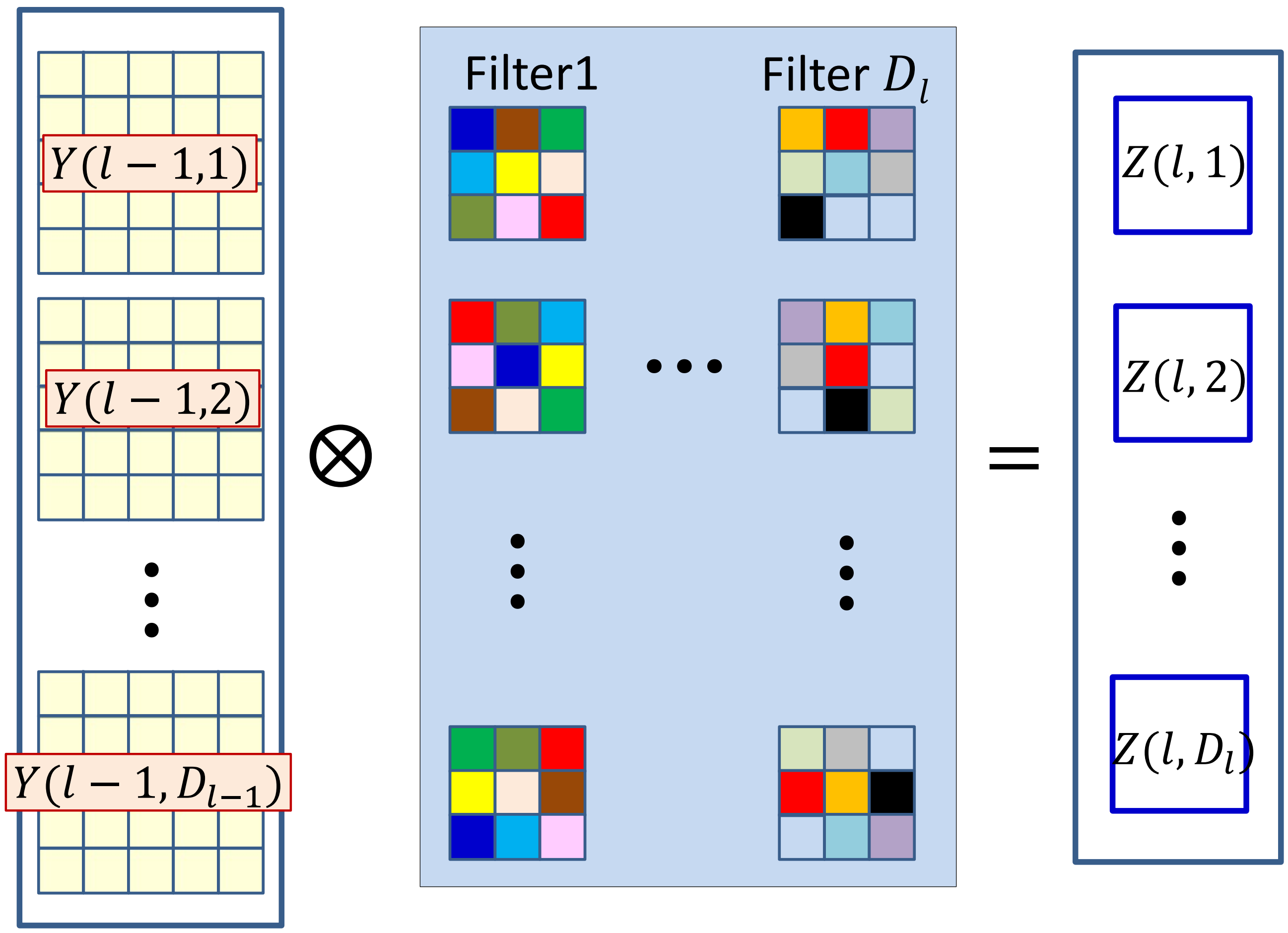
- The derivatives for all components of all filters can be computed directly from the above formula
  - To compute the derivative for $w_l\,(m,n,i,j)$, “place” the $dDiv/dz(l,n)$ map on $y(l-1,m)$ map positioned at (i,j) and compute the inner product

- In fact, it is just a convolution

$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \frac{dDiv}{dz(l,n)} \otimes y(l-1,m)$$

- How?

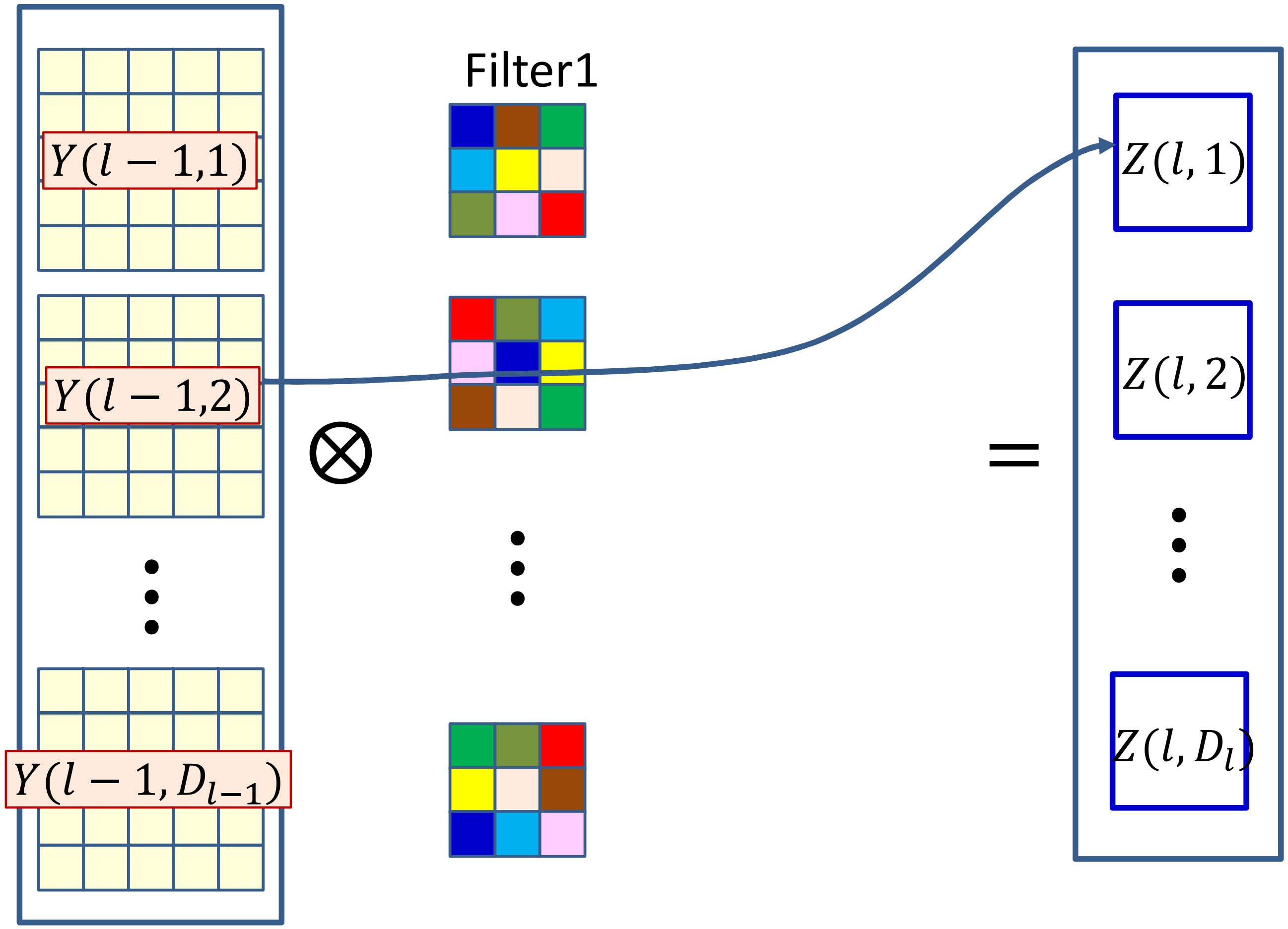
Recap: Convolution



$$z(l,n,x,y) = \sum_m \sum_{i=0}^{2} \sum_{j=0}^{2} w_l(m,n,i,j) y(l-1,m,x+i,y+j) + b_l(n)$$

- Forward computation: Each filter produces an affine map

Recap: Convolution

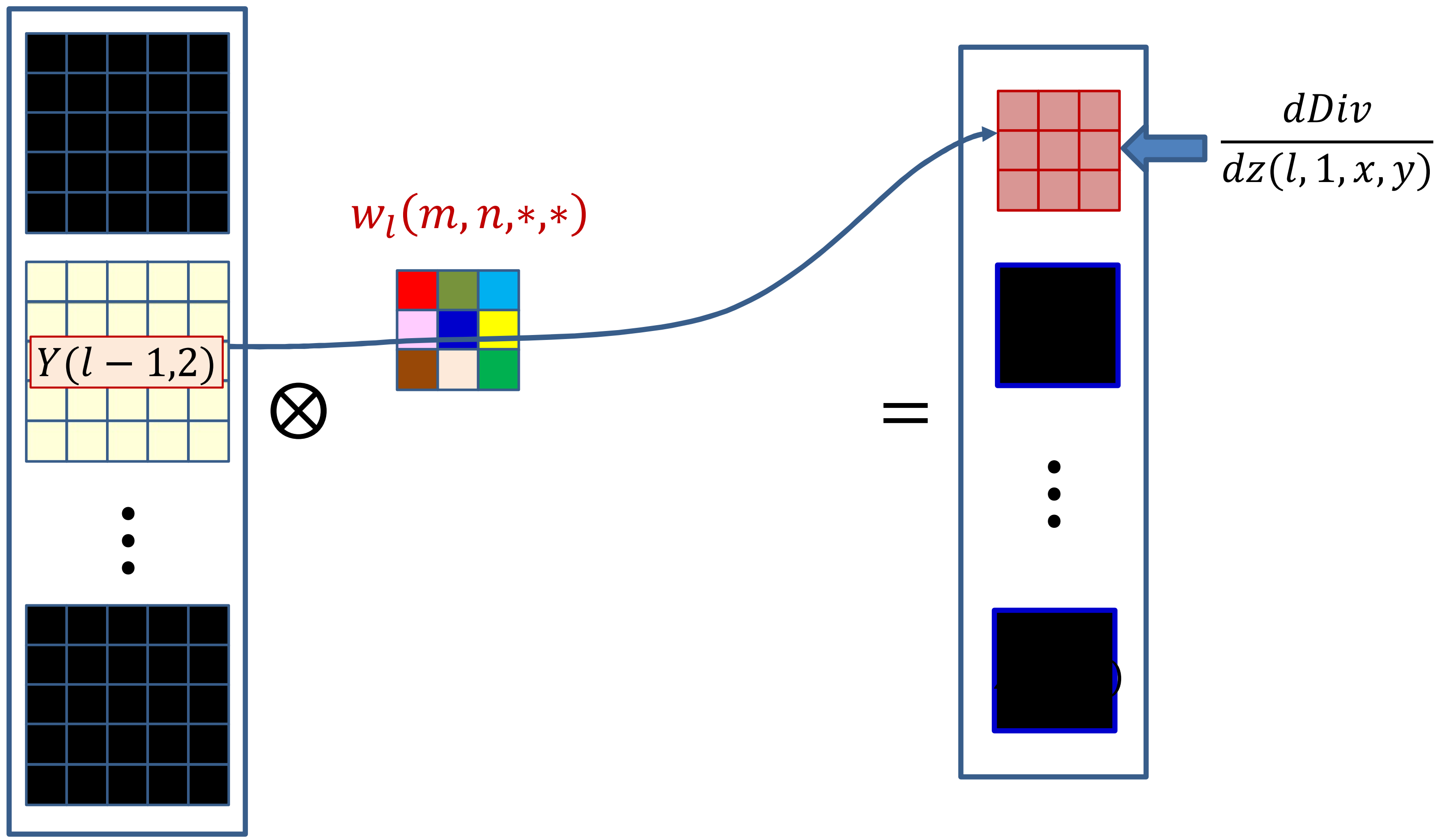


$\boldsymbol{l}$ is layer index

$$z(l,n,x,y) = \sum_{m}\sum_{i=0}^{2}\sum_{j=0}^{2} w_l(m,n,i,j)y(l-1,m,x+i,y+j) + b_l(n)$$

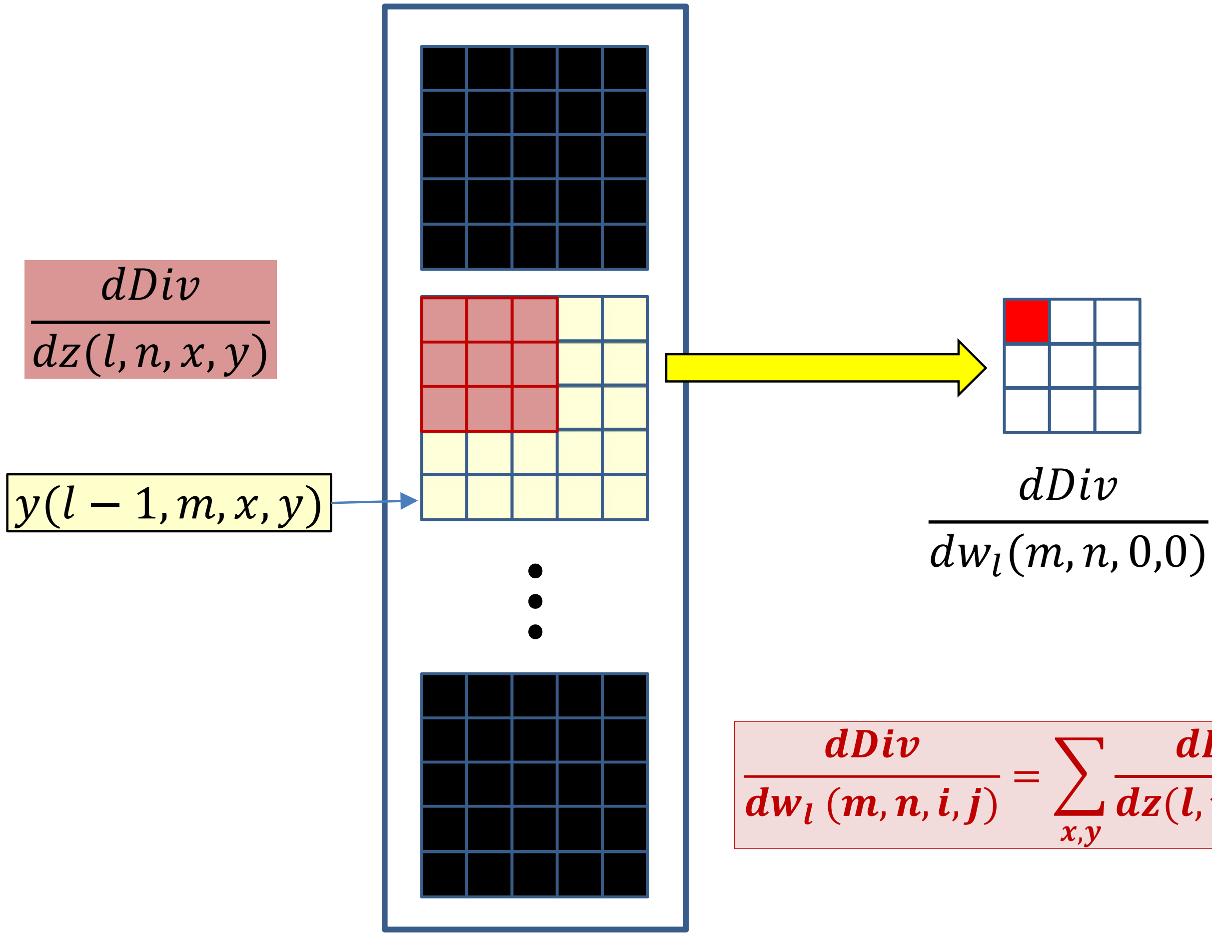
- $Y(l-1,m)$ influences $Z(l,n)$ through $w_l(m,n)$

The filter derivative



- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

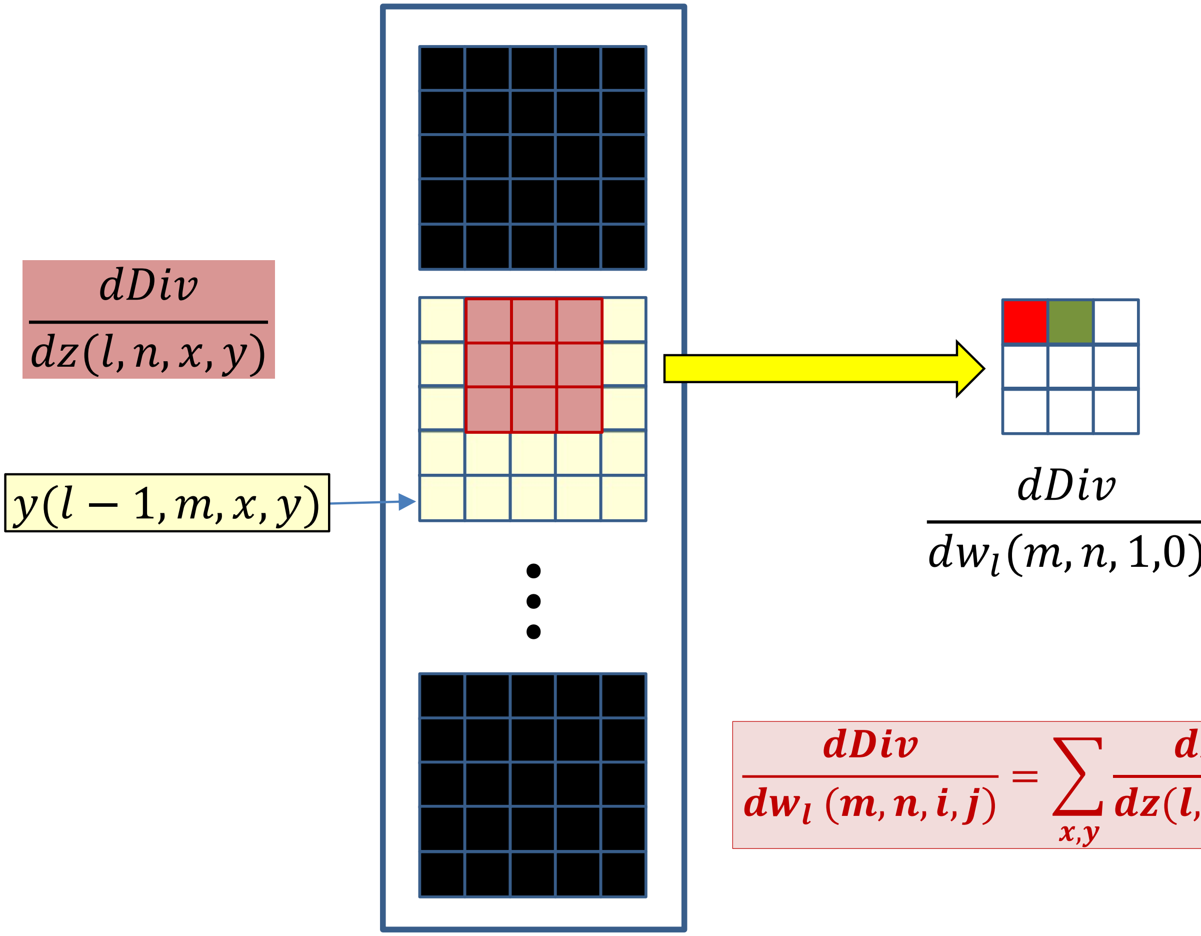
The filter derivative



$$\frac{dDiv}{dw_l(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

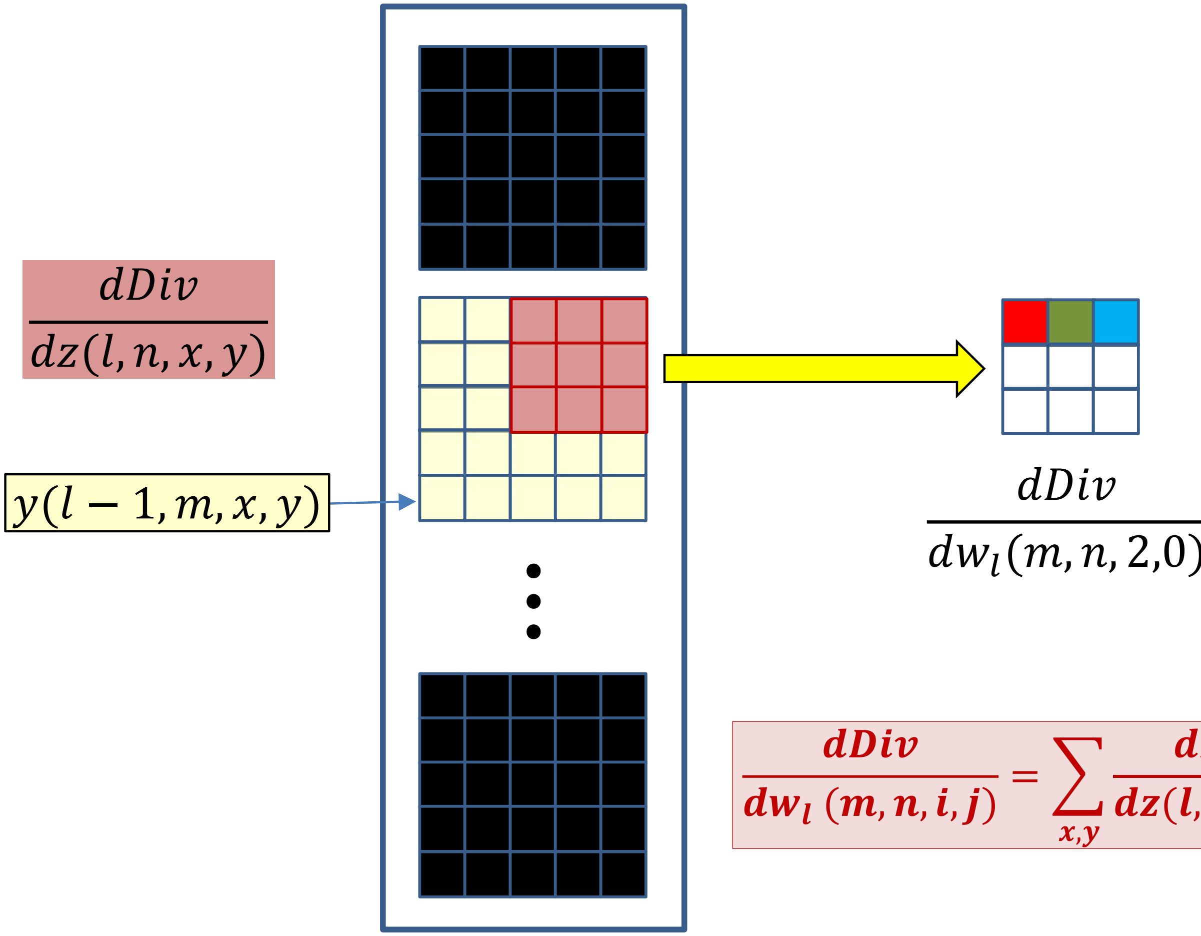
The filter derivative



$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  – Must use them to compute the derivative for $w_l(m,n,*,*)$

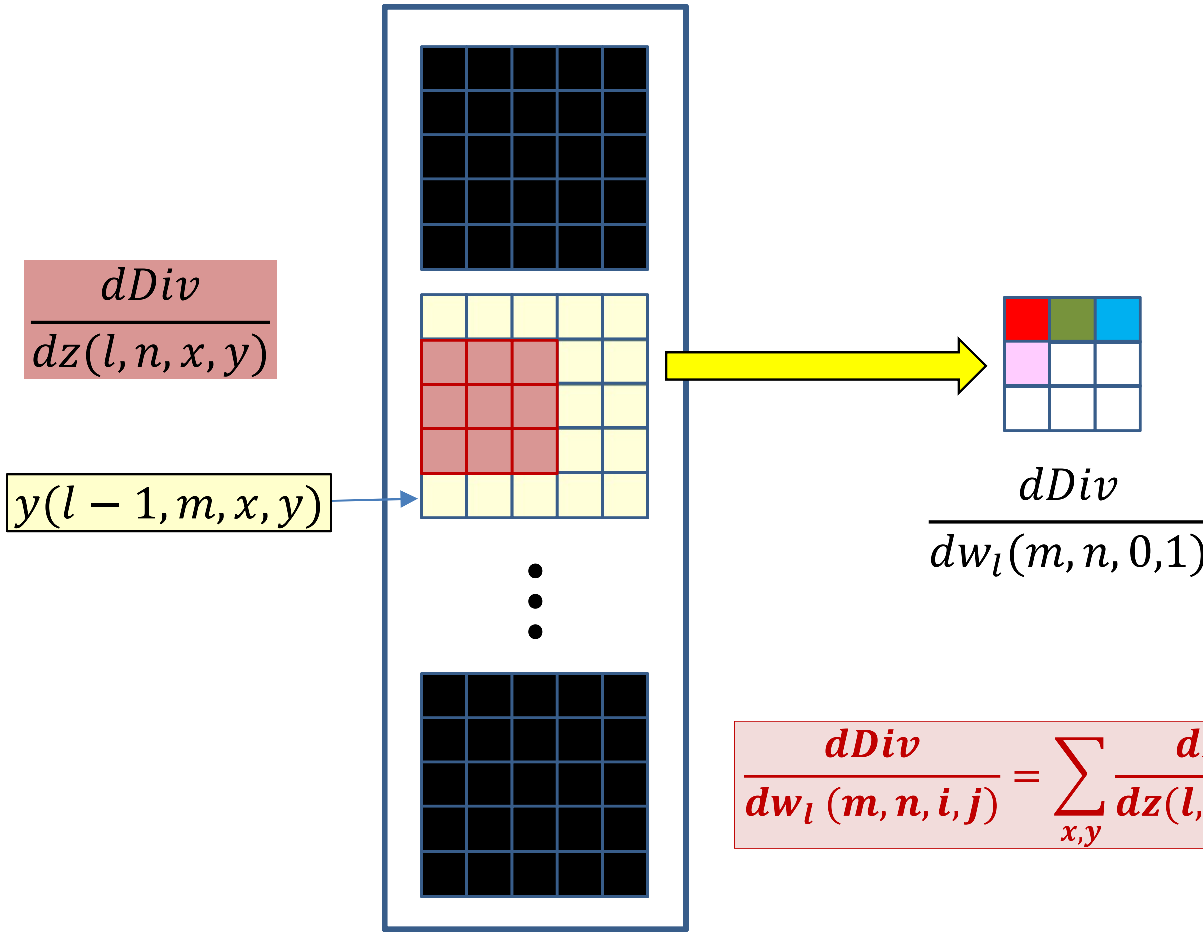
The filter derivative



$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

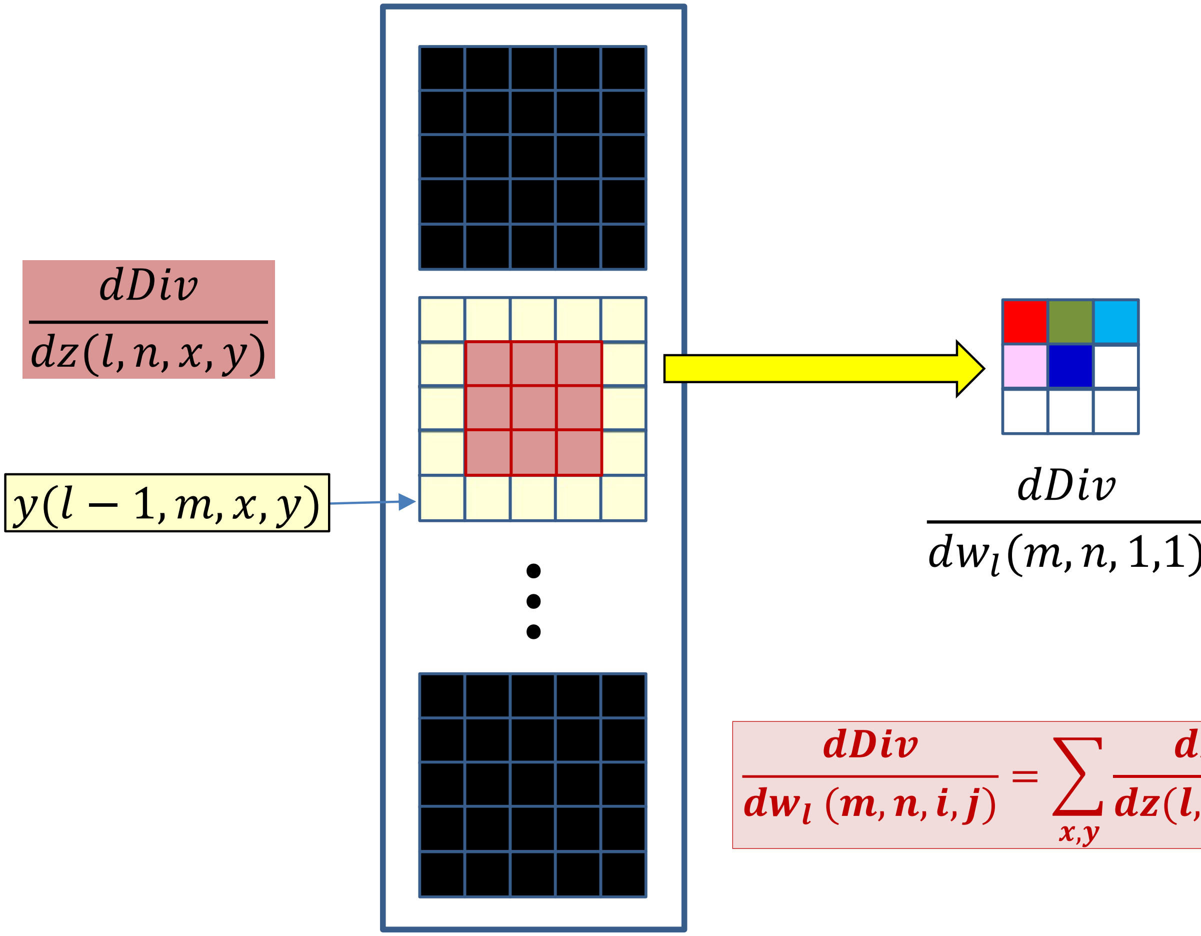
The filter derivative



$$\frac{dDiv}{dw_l(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

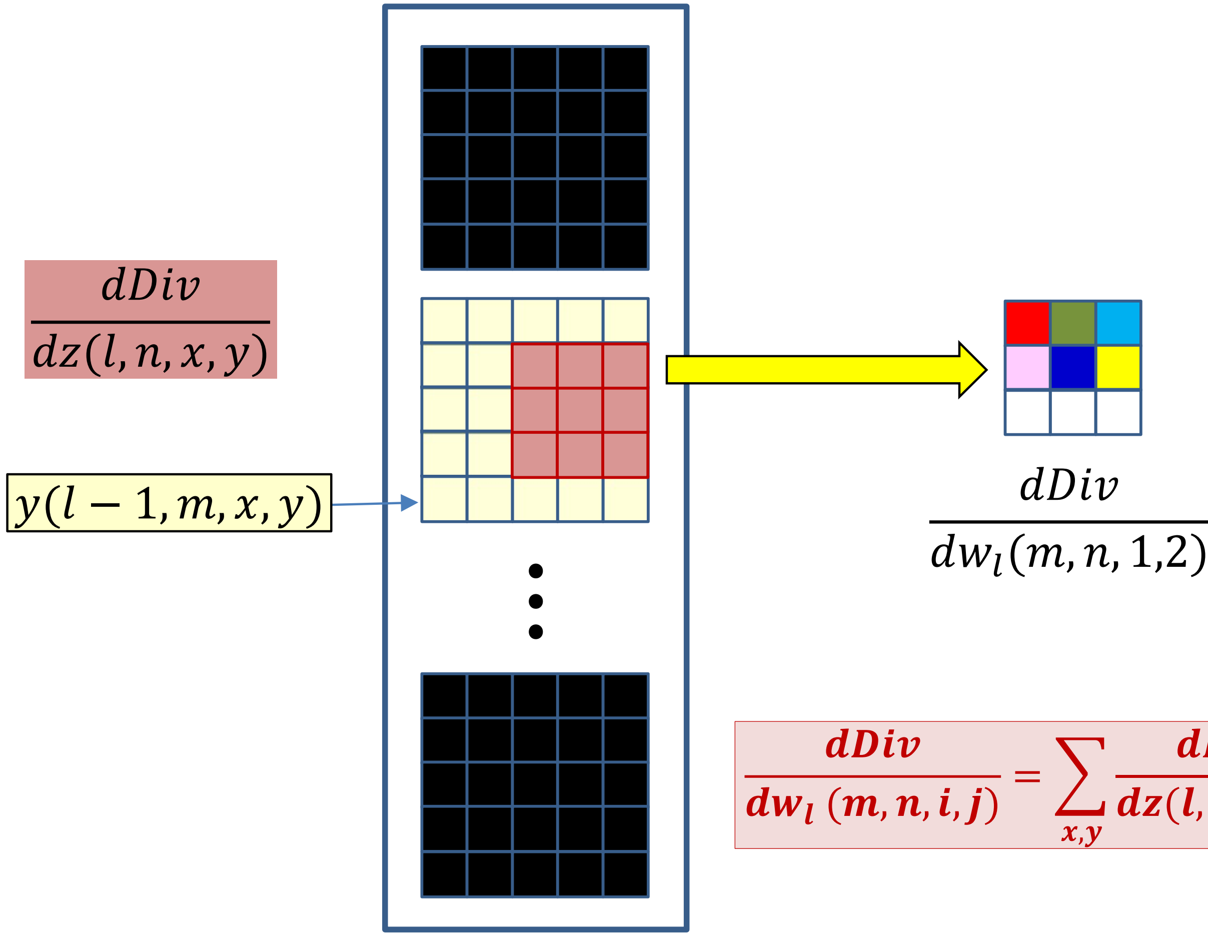
The filter derivative



$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  – Must use them to compute the derivative for $w_l(m,n,*,*)$

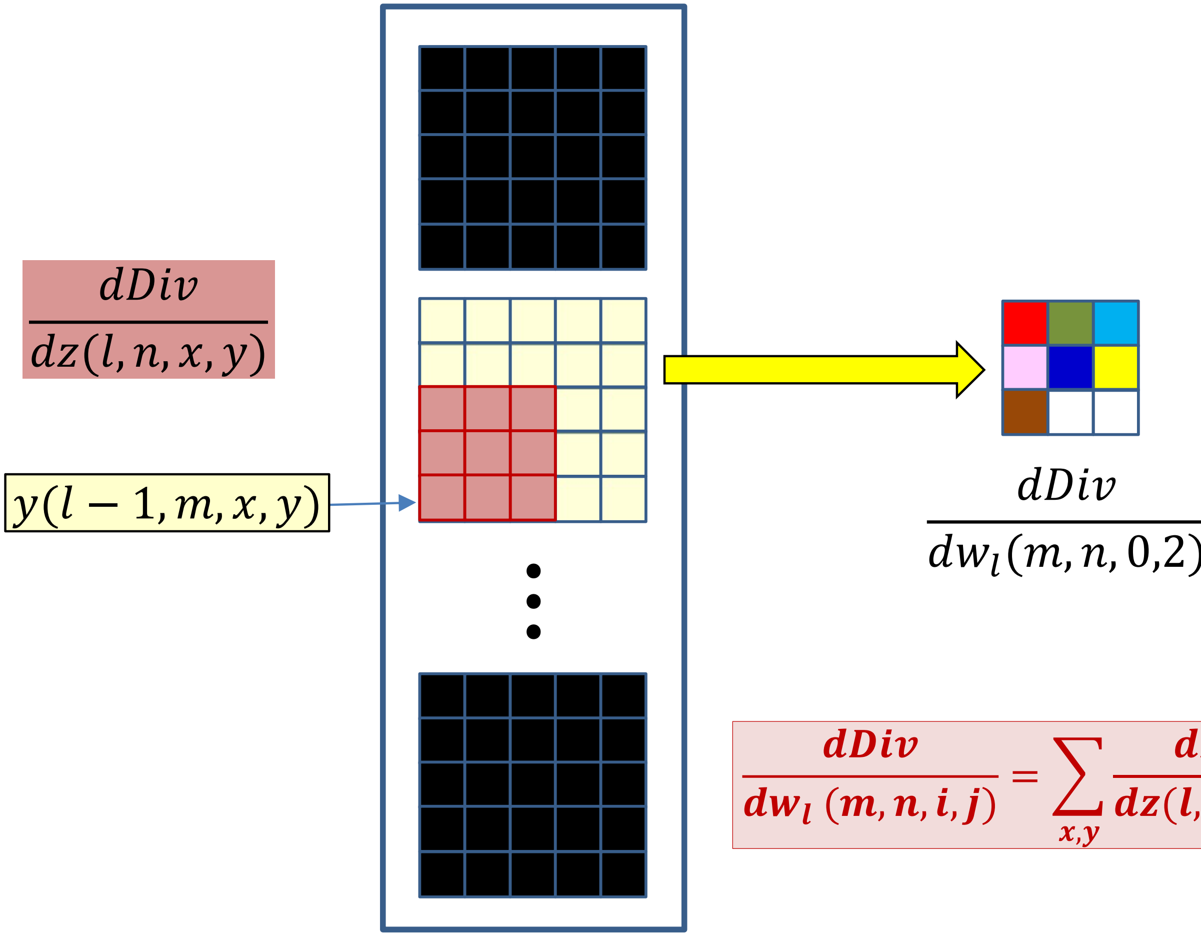
The filter derivative



$$\frac{dDiv}{dw_l(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

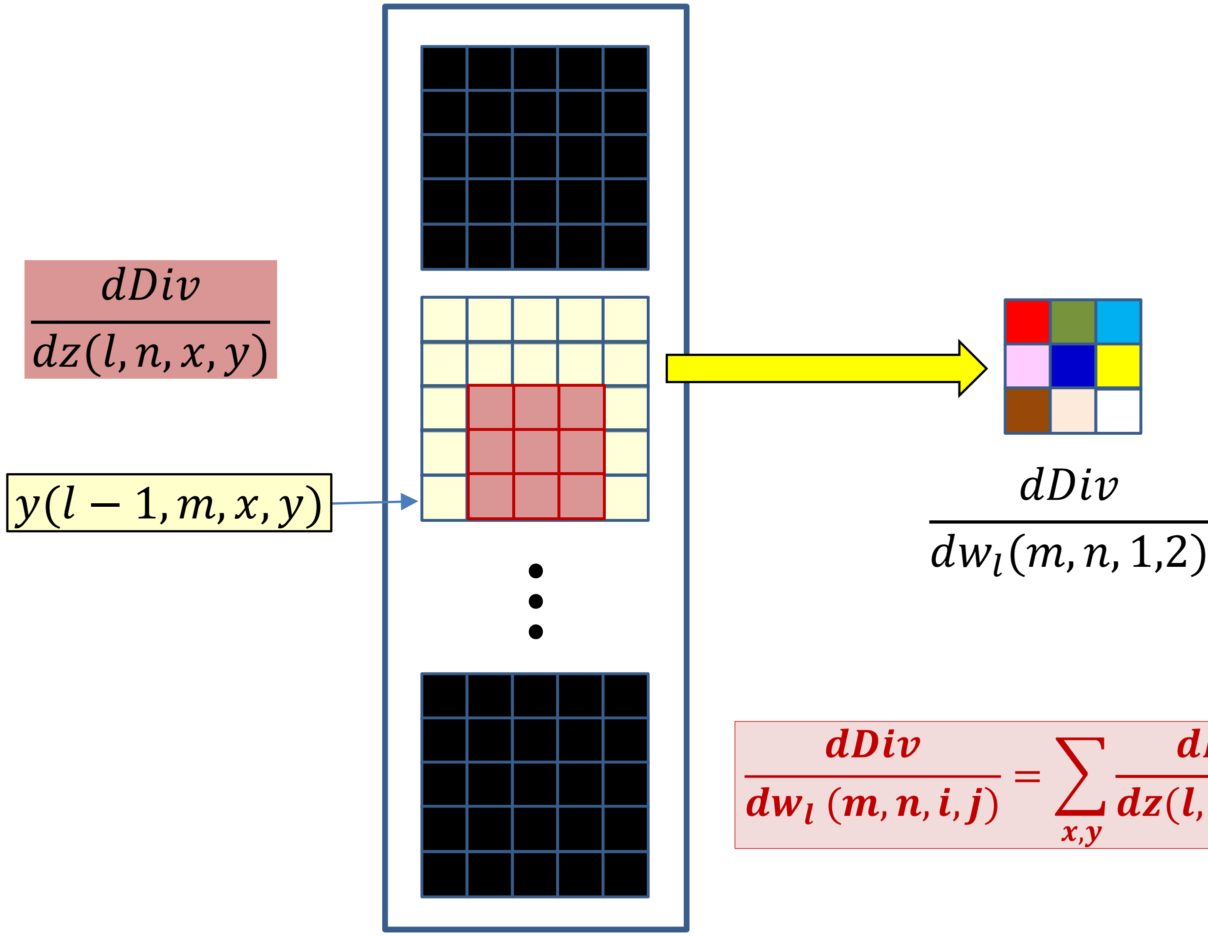
The filter derivative



$$\frac{dDiv}{dw_l(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

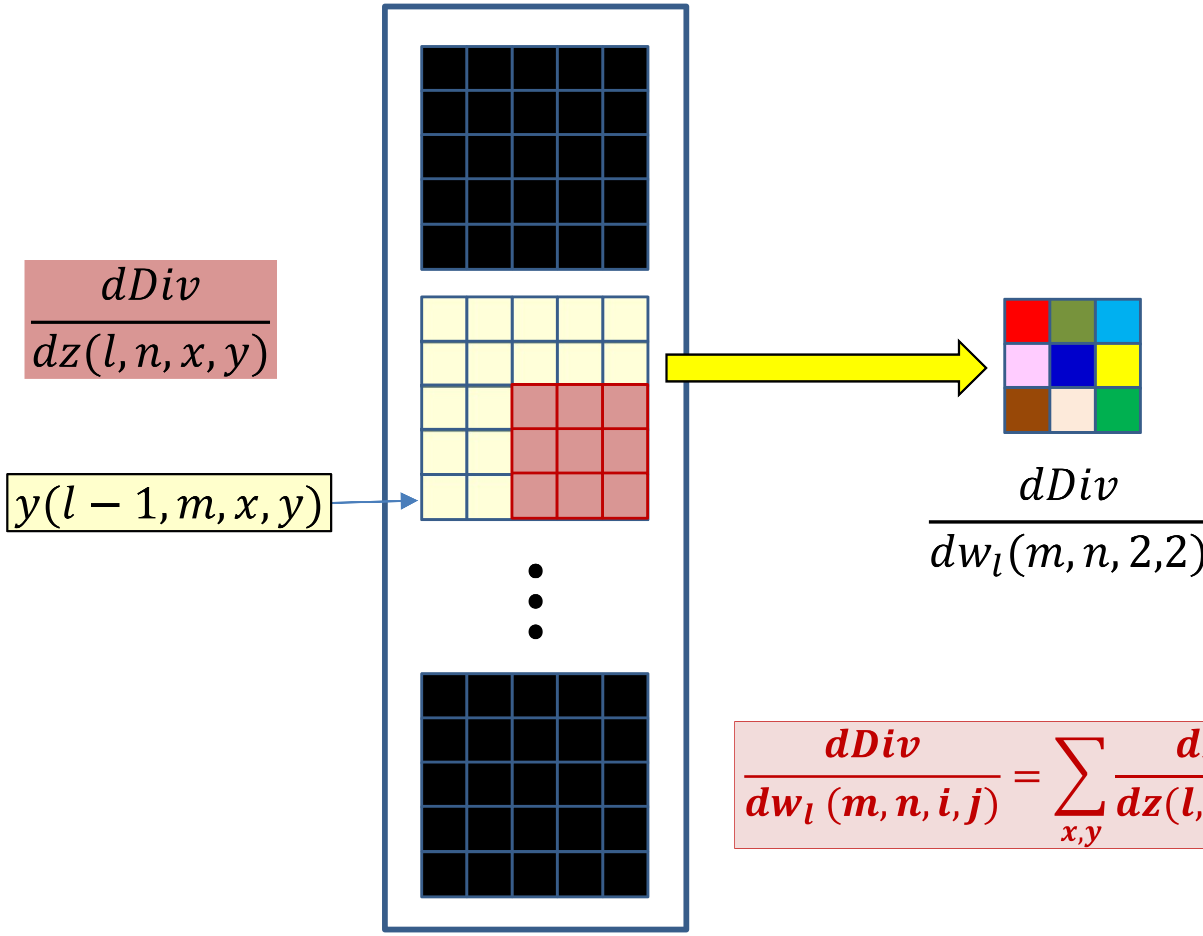
The filter derivative



$$\frac{dDiv}{dw_l(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

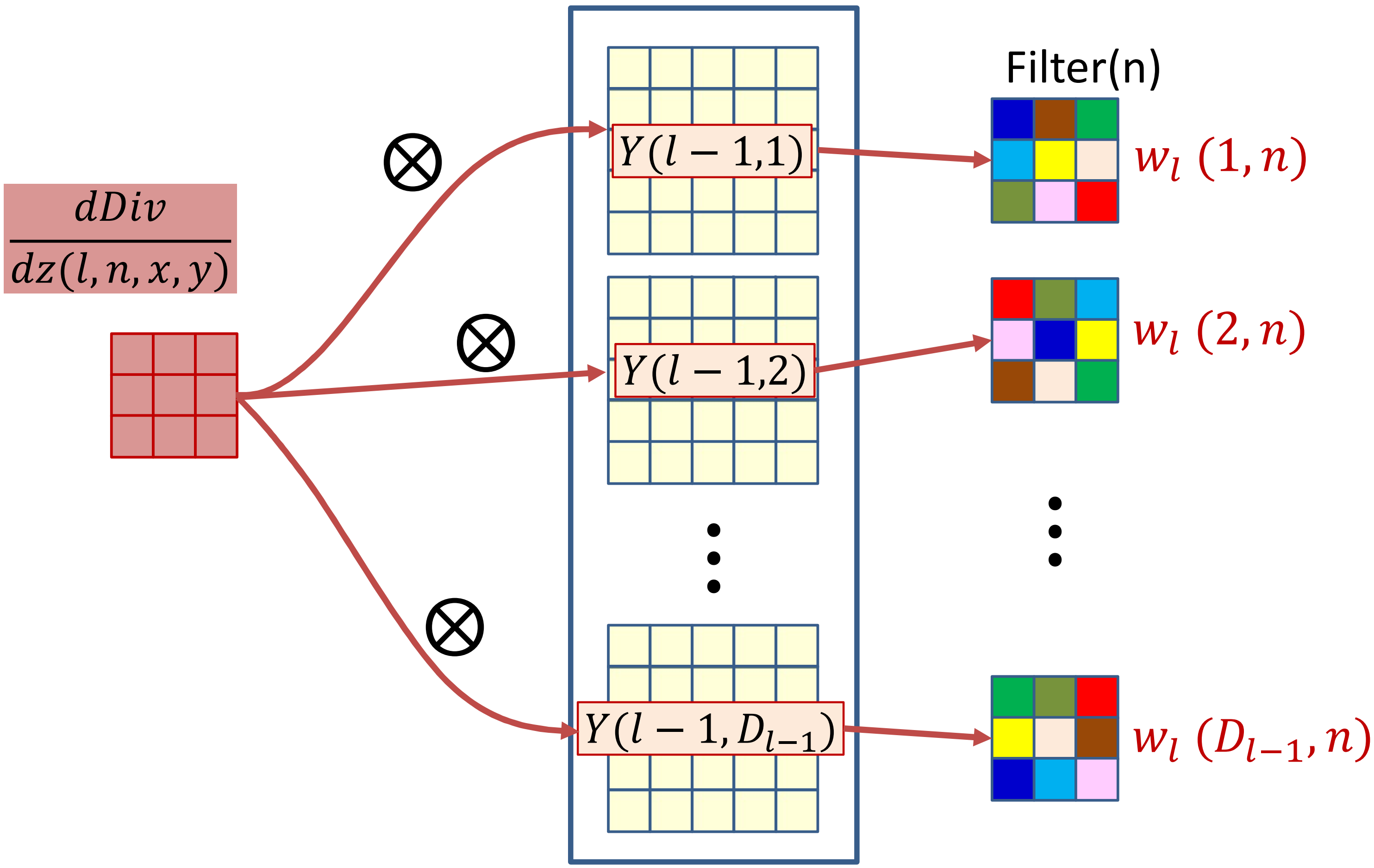
The filter derivative



$$\frac{dDiv}{dw_l\,(m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

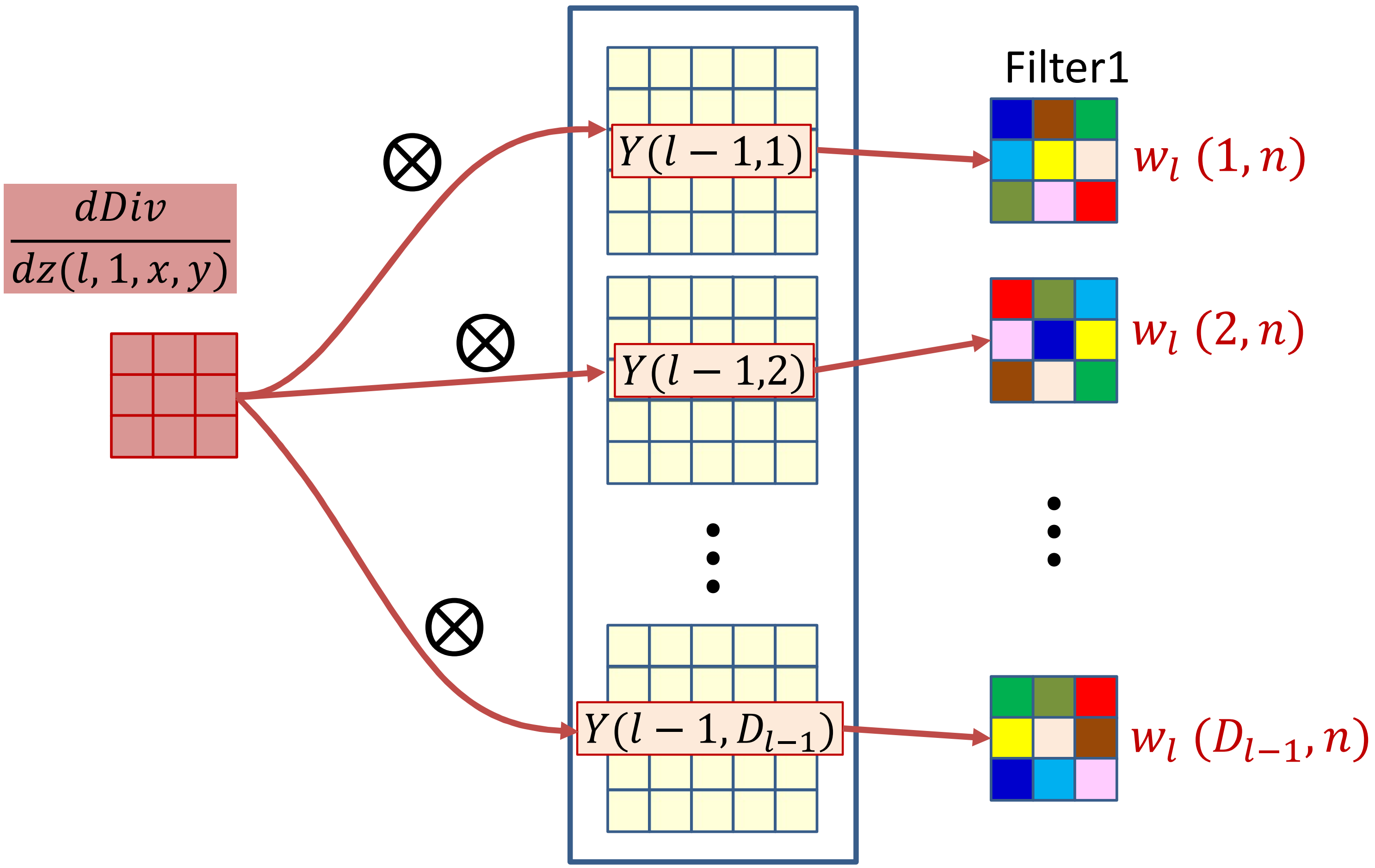
- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

The filter derivative



- The derivative of the n^{th} affine map $Z(l,n)$ convolves with every output map $Y(l-1,m)$ of the $(l-1)^{\text{th}}$ layer, to get the derivative for $w_l(m,n)$, the m^{th} “channel” of the n^{th} filter

The filter derivative



$$\frac{dDiv}{dw_l\ (m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

$$= \frac{dDiv}{dz(l,n)} \otimes y(l-1,m)$$

If $Y(l-1,m)$ was zero padded in the forward pass, it must be zero padded for backprop

Poll 4

Select all statements that are true about how to compute the derivative of the divergence w.r.t lth layer filters using backpropagation

- The derivative for the mth plane of the nth filter is computed by convolving the mth input (l-1th) layer map with the nth output (lth) layer affine derivative map
- The output map must be flipped left-right/up-down before convolution

Poll 4

Select all statements that are true about how to compute the derivative of the divergence w.r.t lth layer filters using backpropagation

- **The derivative for the mth plane of the nth filter is computed by convolving the mth input (l-1th) layer map with the nth output (lth) layer affine derivative map**
- The output map must be flipped left-right/up-down before convolution

Derivatives for the filters at layer l: Vector notation

```
# The weight W(l,j)is a 3D Dl-1xKlxKl
# Assuming that derivative maps have been upsampled
#     if stride > 1
# Also assuming y map has been zero-padded if this was
#     also done in the forward pass
# The width and height of the dz map are W and H
```

```
for n = 1:Dl
  for x = 1:Kl
    for y = 1:Kl
      for m = 1:Dl-1
        dw(l,m,n,x,y) = dz(l,n,:,:).           #dot product
                         y(l-1,m,x:x+H-1,y:y+W-1)
```

Derivatives through a convolutional layer

- The entire process is simpler if we simply look at it through code
  - Through the reapplication of two simple rules:

- For any computation of the form

$$y = \sigma(z)$$

  - The loss derivative for z given the loss derivative of y is

$$\frac{dL}{dz} = \frac{dL}{dy}\sigma^{'(z)}$$

- For any computation in the forward pass

$$z = wy$$

  - The backward computation to compute loss derivatives for the terms on the right, given loss derivatives to the left is

$$dL/dy \mathrel{+}= wdL/dz \; ; \; dL/dw \mathrel{+}= ydL/dz$$

  - Since this is "backpropgation", all computations are reversed

CNN: Forward

```
Y(0,:,:,:) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for x = 1:W_{l-1}-K_l+1
        for y = 1:H_{l-1}-K_l+1
            for j = 1:D_l
                z(l,j,x,y) = 0
                for i = 1:D_{l-1}
                    for x' = 1:K_l
                        for y' = 1:K_l
                            z(l,j,x,y) += w(l,j,i,x',y')
                                    Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Switching to 1-based indexing with appropriate adjustments

Backward layer *l*

```
dw(l) = zeros(D_l x D_{l-1} x K_l x K_l)
dY(l-1) = zeros(D_{l-1} x W_{l-1} x H_{l-1})
for x = W_{l-1}-K_l+1:downto:1
  for y = H_{l-1}-K_l+1:downto:1
     for j = D_l:downto:1
        dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
        for i = D_{l-1}:downto:1
          for x' = K_l:downto:1
            for y' = K_l:downto:1
              dY(l-1,i,x+x'-1,y+y'-1) +=
                          w(l,j,i,x',y')dz(l,j,x,y)
              dw(l,j,i,x',y') +=
                    dz(l,j,x,y)Y(l-1,i,x+x'-1,y+y'-1)
```

Complete Backward (no pooling)

```
dY(L) = dDiv/dY(L)
for l = L:downto:1   # Backward through layers
    dw(l) = zeros(D_l x D_l-1 x K_l x K_l)
    dY(l-1) = zeros(D_l-1 x W_l-1 x H_l-1)
    for x = W_l-1 - K_l + 1:downto:1
        for y = H_l-1 - K_l + 1:downto:1
            for j = D_l:downto:1
                dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
                for i = D_l-1:downto:1
                    for x' = K_l:downto:1
                        for y' = K_l:downto:1
                            dY(l-1,i,x+x'-1,y+y'-1) +=
                                w(l,j,i,x',y')dz(l,j,x,y)
                            dw(l,j,i,x',y') +=
                            dz(l,j,x,y)y(l-1,i,x+x'-1,y+y'-1)
```

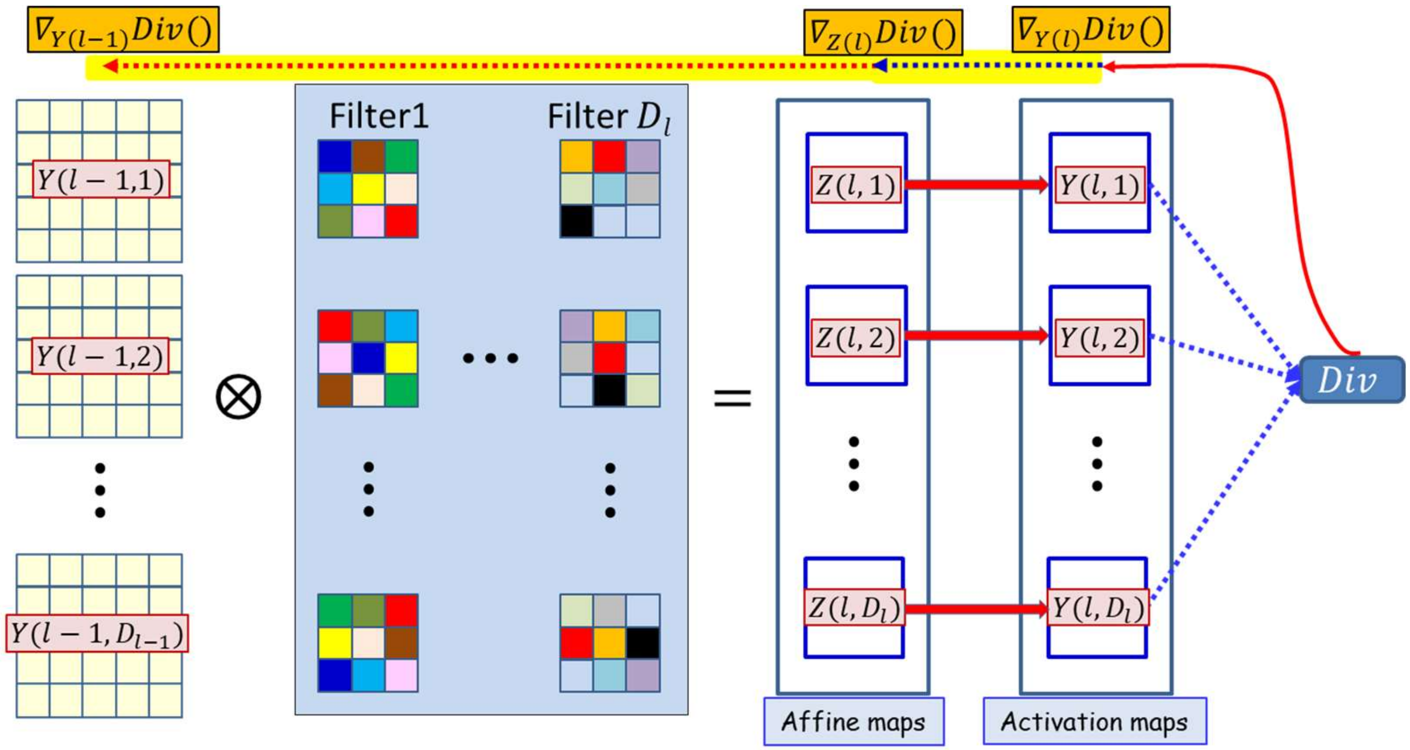
Complete Backward (no pooling)

```
dY(L) = dDiv/dY(L)
for l = L:downto:1  # Backward through layers
    dw(l) = zeros(DlxDl-1xKlxKl)
    dY(l-1) = zeros(Dl-1xWl-1xHl-1)
    for x = Wl-1-Kl+1:downto:1
        for y = Hl-1-Kl+1:downto:1
            for j = Dl:downto:1
                dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
                for i = Dl-1:downto:1
                    for x' = Kl:downto:1
                        for y' = Kl:downto:1
                            dY(l-1,i,x+x'-1,y+y'-1) +=
                                w(l,j,i,x',y')dz(l,j,x,y)
                            dw(l,j,i,x',y') +=
                            dz(l,j,x,y)y(l-1,i,x+x'-1,y+y'-1)
```

Multiple ways of recasting this as tensor/ vector operations.

Will not discuss here

Backpropagation: Convolutional layers



- **For convolutional layers:**

  ✓ How to compute the derivatives w.r.t. the affine $Z(l)$ maps from the activation output maps $Y(l)$

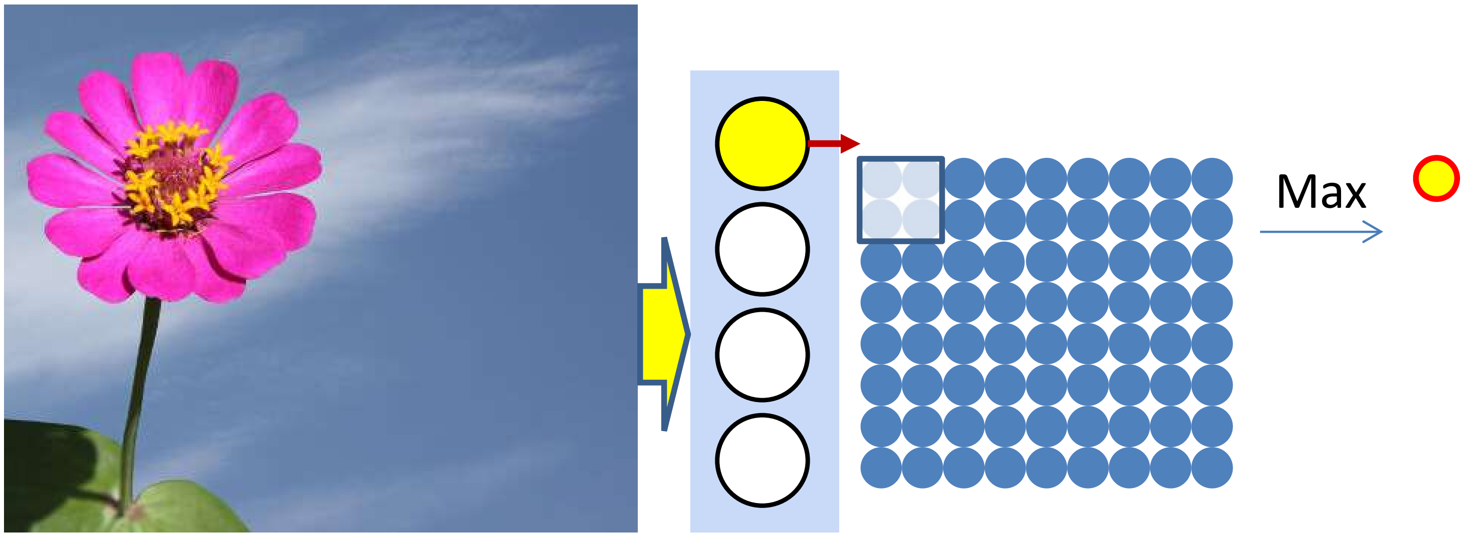
  ✓ How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

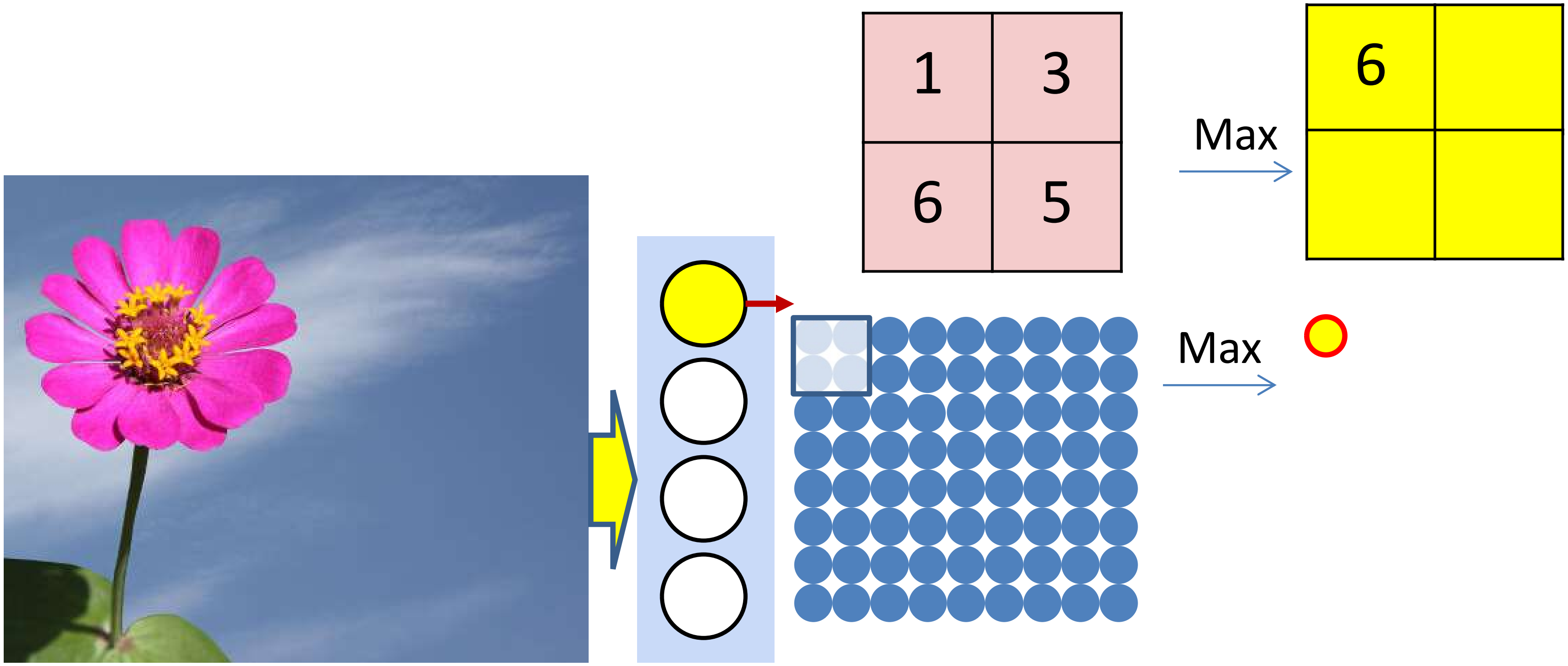
- **Required:**
  - **For convolutional layers**:
    - How to compute the derivatives w.r.t. the affine $Z(l)$ maps from the activation output maps $Y(l)$
    - How to compute the derivative w.r.t. $Y(l-1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
  - **For pooling layers**:
    - How to compute the derivative w.r.t. $Y(l-1)$ given derivatives w.r.t. $Y(l)$

Pooling



- Pooling "pools" groups of values to reduce jitter-sensitivity
  - Scanning with a "pooling" filter
- The most common pooling is "Max" pooling

Max Pooling

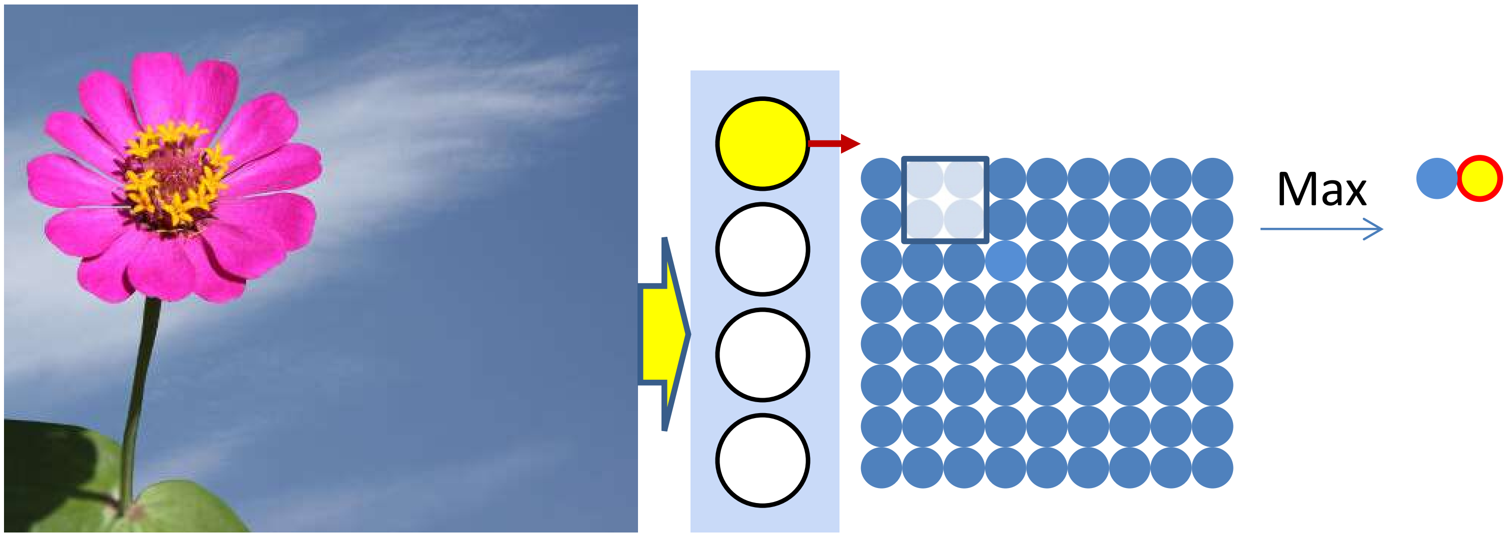


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

$$P(l,m,i,j) = \underset{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j,j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1,m,k,n)$$

$$Y(l,m,i,j) = Y(l-1,m,P(l,m,i,j))$$

Max pooling

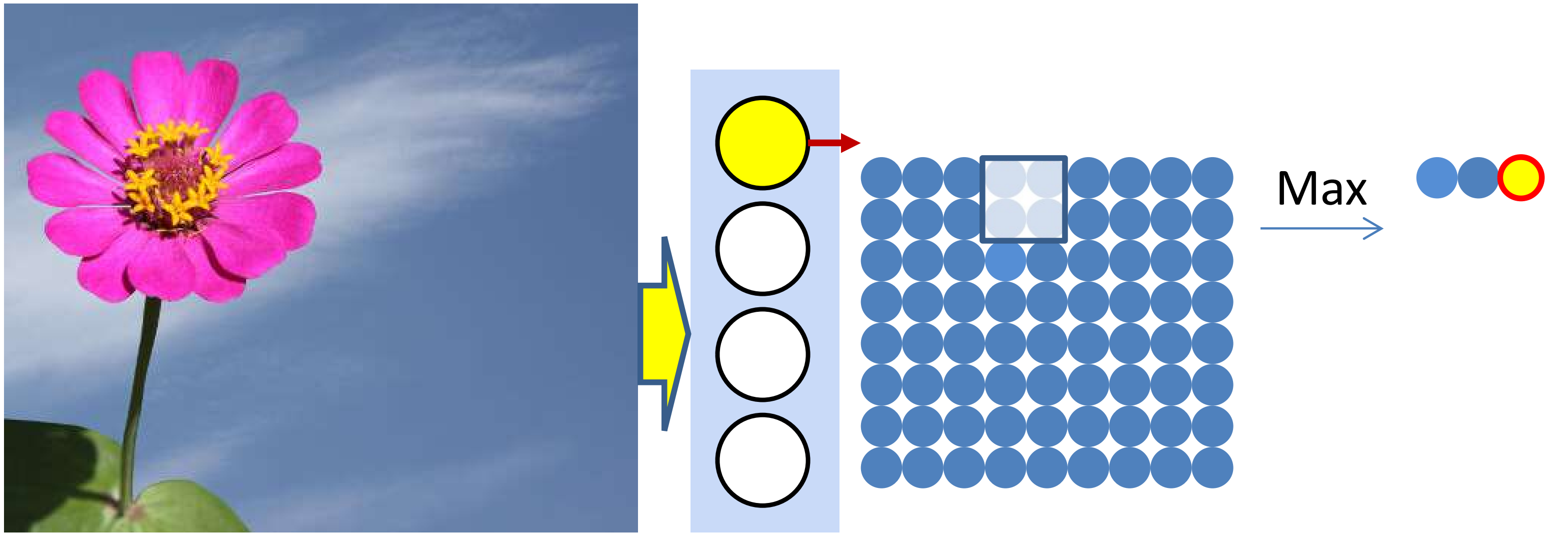


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

$$P(l,m,i,j) = \underset{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j,j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1,m,k,n)$$

$$Y(l,m,i,j) = Y(l-1,m,P(l,m,i,j))$$

Max pooling

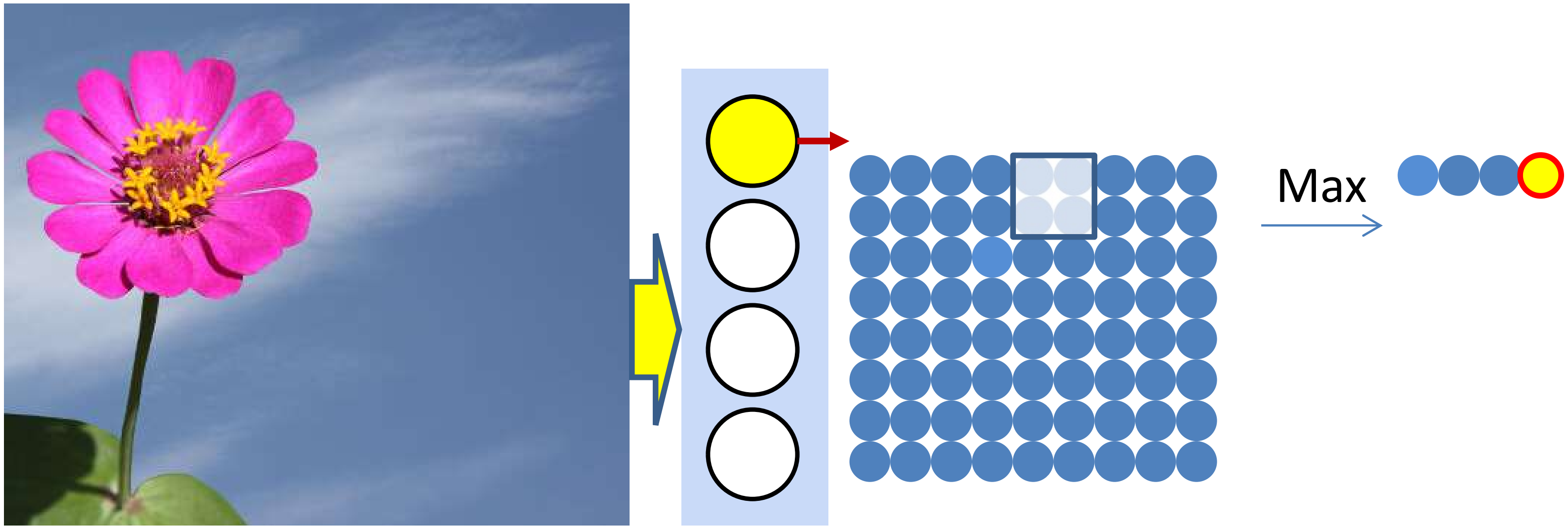


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

$$P(l,m,i,j) = \underset{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j, j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1,m,k,n)$$

$$Y(l,m,i,j) = Y(l-1,m,P(l,m,i,j))$$

Max pooling

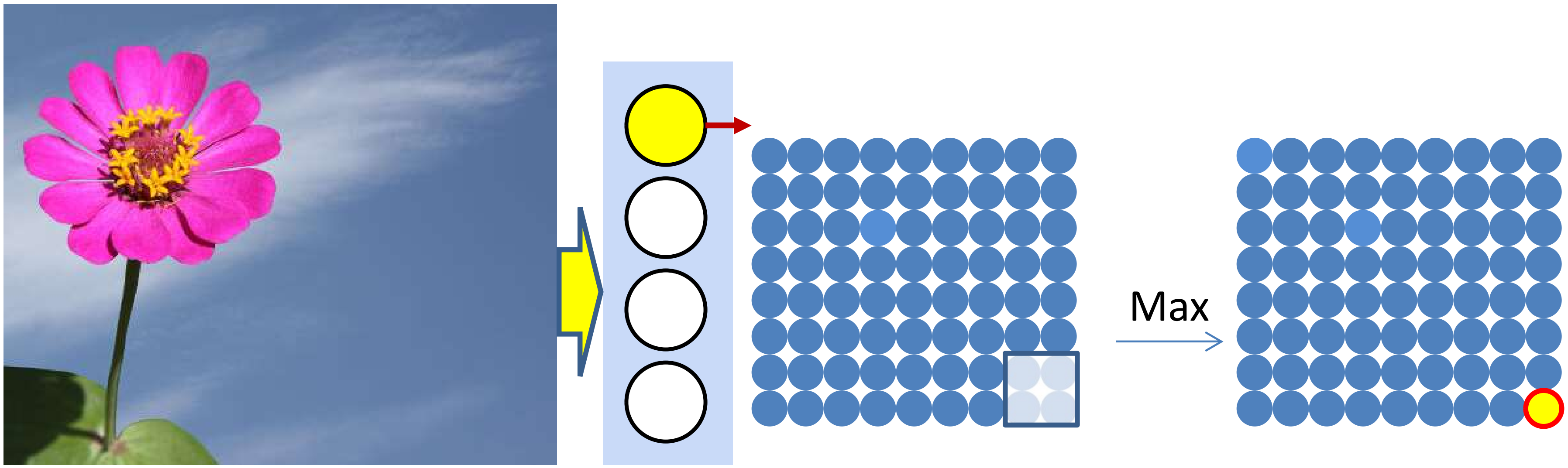


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

$$P(l, m, i, j) = \underset{\substack{k \in \{i,\, i+K_{lpool}-1\},\\ n \in \{j, j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1, m, k, n)$$

$$Y(l, m, i, j) = Y(l-1, m, P(l, m, i, j))$$

Max pooling

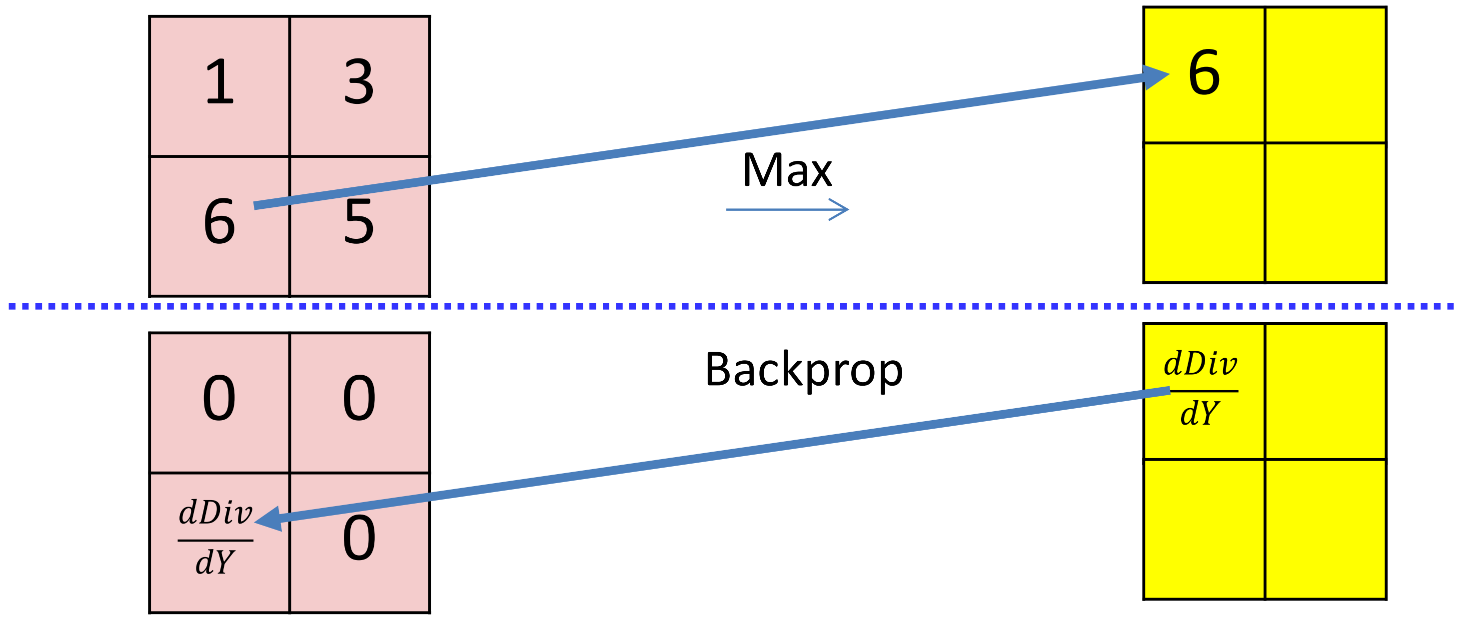


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

$$P(l,m,i,j) = \underset{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j, j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1,m,k,n)$$

$$Y(l,m,i,j) = Y(l-1,m,P(l,m,i,j))$$

Derivative of Max pooling



$$\frac{dDiv}{dy(l-1,m,k,l)} = \begin{cases} \frac{dDiv}{dy(l,m,i,j)} \; if \; (k,l) = P(l,m,i,j) \\ 0 \; otherwise \end{cases}$$

- Max pooling selects the largest from a pool of elements

$$P(l,m,i,j) = \underset{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j, j+K_{lpool}-1\}}}{\operatorname{argmax}} Y(l-1,m,k,n)$$

$$y(l,m,i,j) = y(l-1,m,P(l,m,i,j))$$

Max Pooling layer at layer l

a) Performed separately for every map (j).
*) *Not* combining multiple maps within a single max operation.
b) Keeping track of location of max

Max pooling

```
for j = 1:D_l
  for x = 1:W_(l-1)-K_l+1
    for y = 1:H_(l-1)-K_l+1
      pidx(l,j,x,y) = maxidx(y(l-1,j,x:x+K_l-1,y:y+K_l-1))
      y(l,j,x,y) = y(l-1,j,pidx(l,j,x,y))
```

Derivative of max pooling layer at layer l

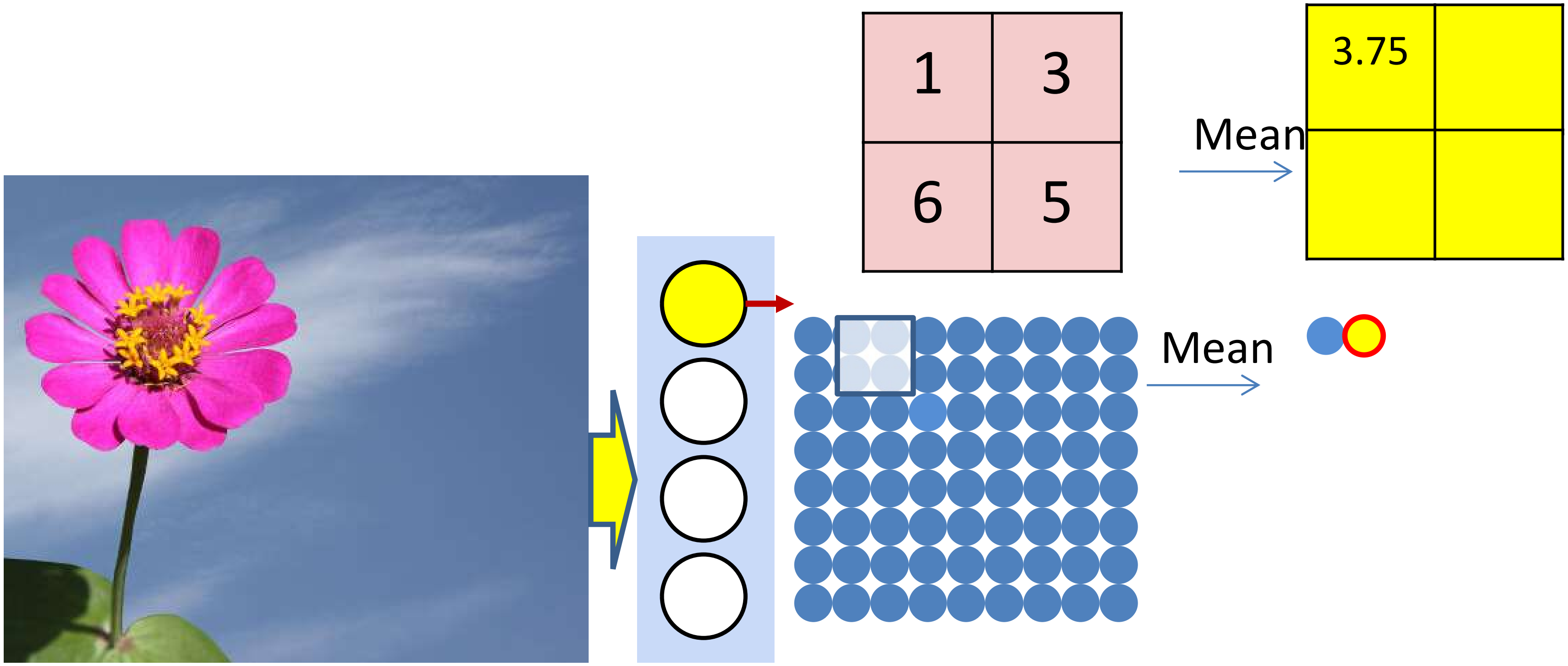
a) Performed separately for every map (j).
   *) *Not* combining multiple maps within a single max operation.
b) Keeping track of location of max

Max pooling

```
dy(:,:,:) = zeros(D_l x W_l x H_l)
for j = 1:D_l
    for x = 1:W_l
        for y = 1:H_l
            dy(l-1,j,pidx(l,j,x,y)) += dy(l,j,x,y)
```

"+=" because this entry may be selected in multiple adjacent overlapping windows

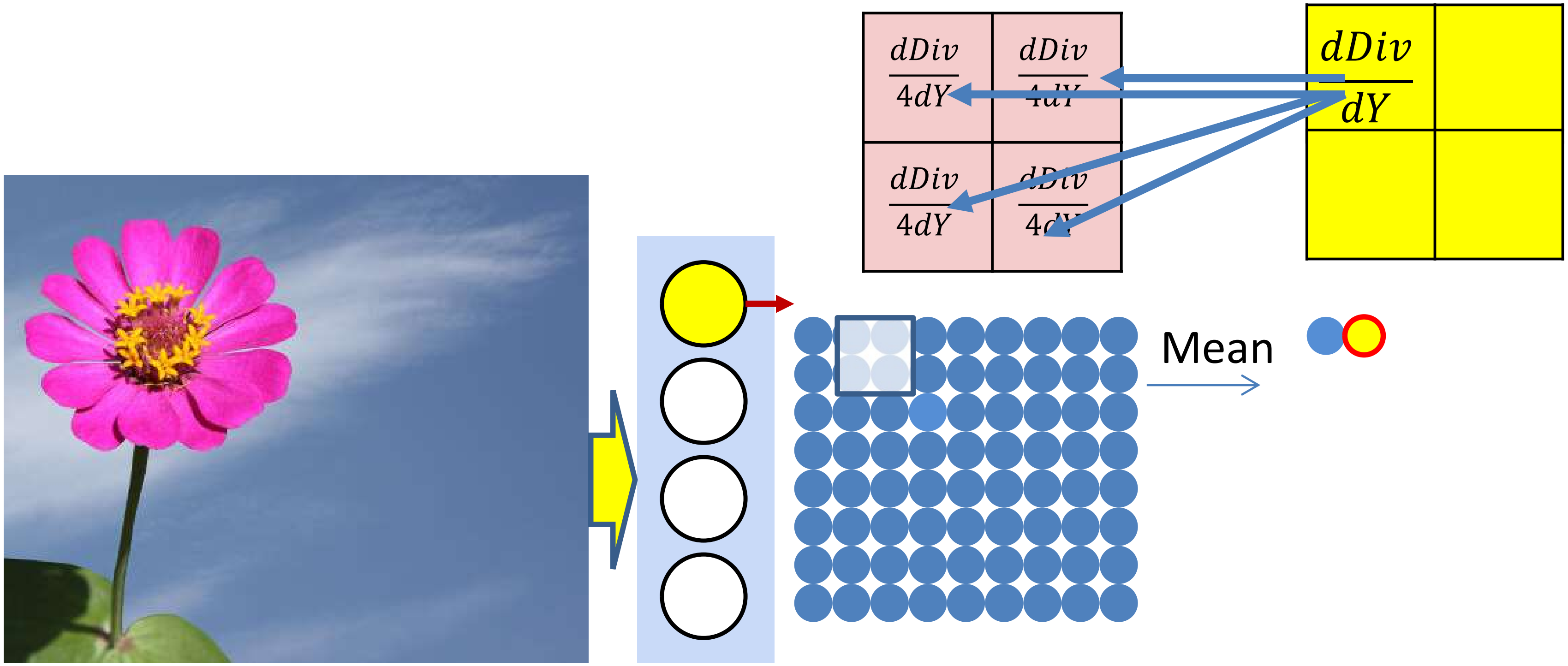
Mean pooling



- Mean pooling compute the mean of a pool of elements
- Pooling is performed by "scanning" the input

$$y(l,m,i,j) = \frac{1}{K_{lpool}^2} \sum_{\substack{k\in\{i,\, i+K_{lpool}-1\},\\ n\in\{j, j+K_{lpool}-1\}}} y(l-1,m,k,n)$$

Derivative of mean pooling



- The derivative of mean pooling is distributed over the pool

$$k \in \{i, i + K_{lpool} - 1\},\ n \in \{j, j + K_{lpool} - 1\}\quad dy(l-1, m, k, n) \mathrel{+}= \frac{1}{K_{lpool}^2} dy(l, m, k, n)$$

Mean Pooling layer at layer l

Mean pooling

```
for j = 1:D_l  #Over the maps
    for x = 1:W_l-1-K_l+1 #K_l = pooling kernel size
        for y = 1:H_l-1-K_l+1
            y(l,j,x,y) = mean(y(l-1,j,x:x+K_l-1,y:y+K_l-1))
```

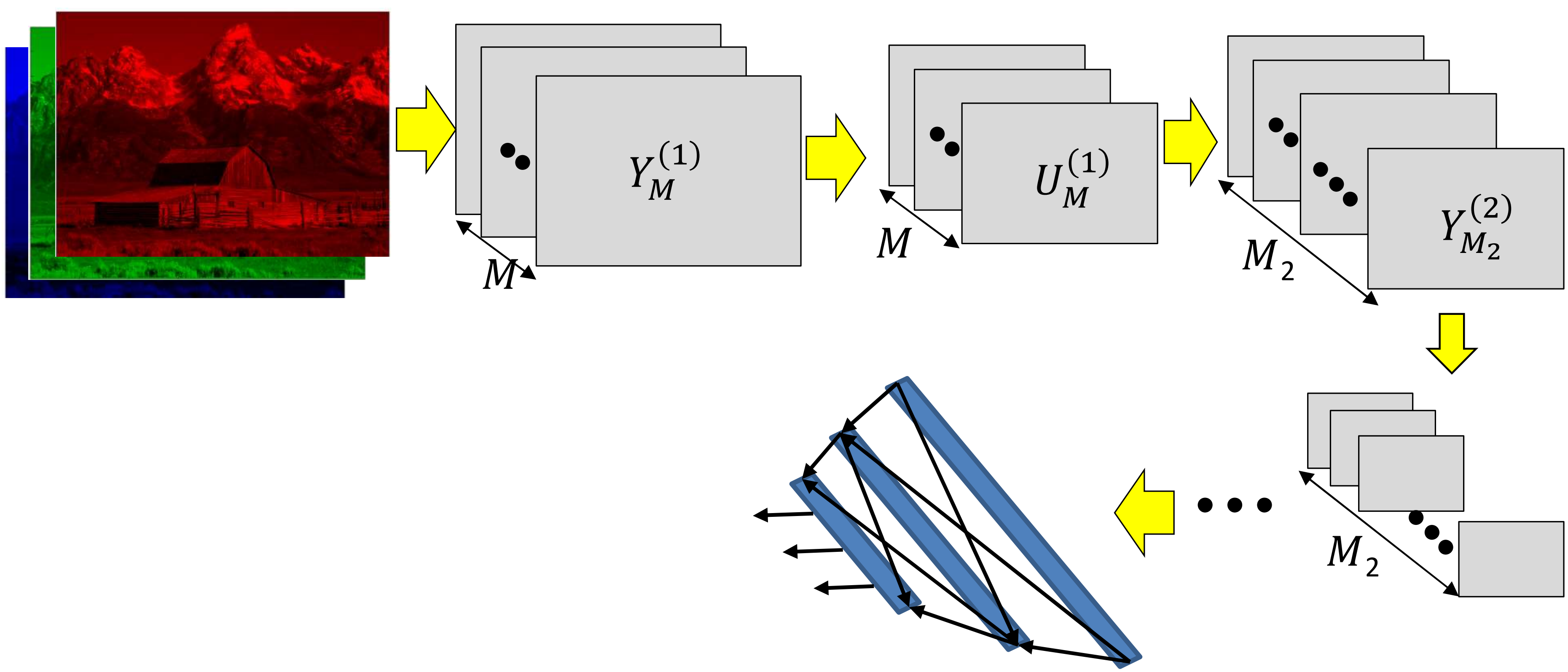
Derivative of mean pooling layer at layer *l*

Mean pooling

```
dy(:,:,:) = zeros(Dl x Wl x Hl)
for k = 1:Dl
  for x = 1:Wl
    for y = 1:Hl
       for i = 1:Klpool
          for j = 1:Klpool
              dy(l-1,k,p,x+i,y+j) += (1/K2lpool)dy(l,k,x,y)
```

"+=" because adjacent windows may overlap

Learning the network



- Have shown the derivative of divergence w.r.t every intermediate output, and every free parameter (filter weights)
- Can now be embedded in gradient descent framework to learn the network
- Still missing one component… resampling
  - Next class

Story so far

- The CNNis a supervised version of a computational model of mammalian vision
- It includes
  - Convolutional layers comprising learned filters that scan the outputs of the previous layer, followed by an activation function
  - Pooling layers that operate over groups of outputs from the convolutional layer for jitter invariance
  - Optional resizing operations, typically performed through strides greater than 1 (for downsampling) or interpolation of zeros (for upsampling)

- The parameters of the network can be learned through regular back propagation
  - Computing derivatives for convolutional layers is simply also a convolution operation
    - Convolve transposed filters with derivative maps of output for derivatives of input maps
    - Convolve input maps with derivative maps of the output for derivatives of filters
  - Backprop of pooling operations redistributes the derivatives at the output over all the inputs that actively contributed to it