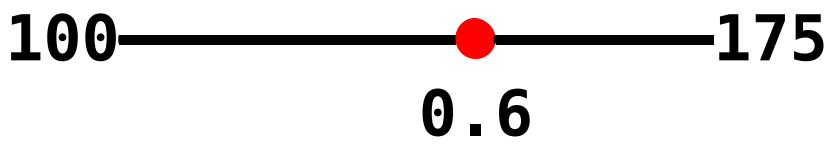
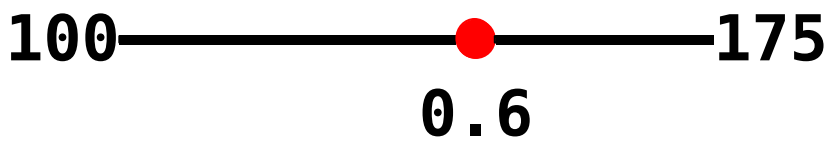
The Rank-Polymorphic Array-Oriented Programming Model

Justin Slepak

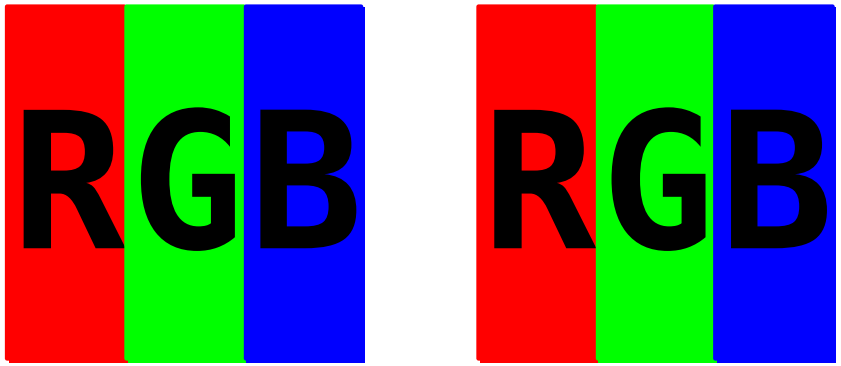
In collaboration with Olin Shivers and Panagiotis Manolios



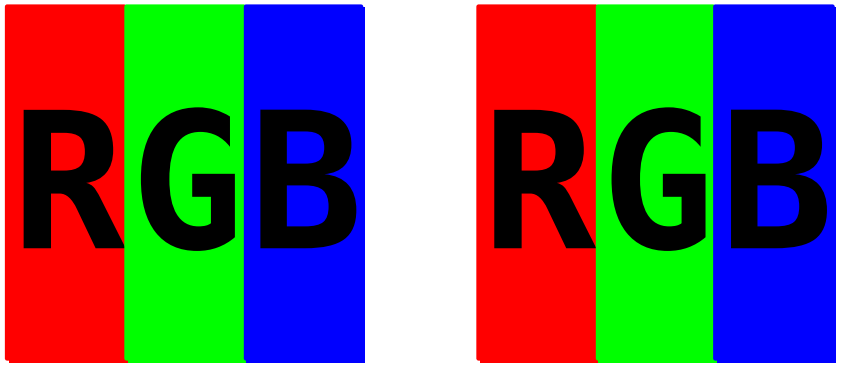

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```



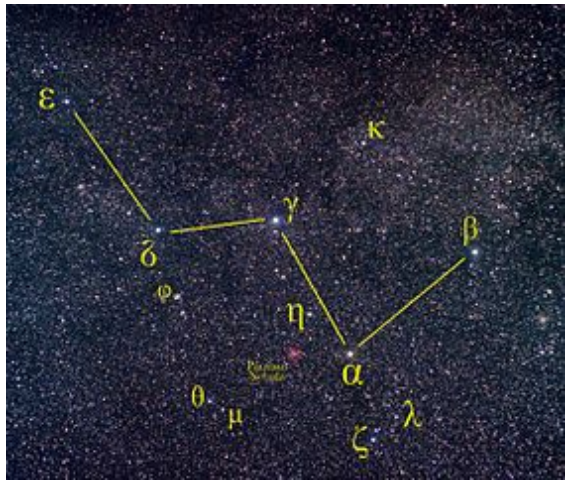

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 100
 175
 0.6)
```




```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

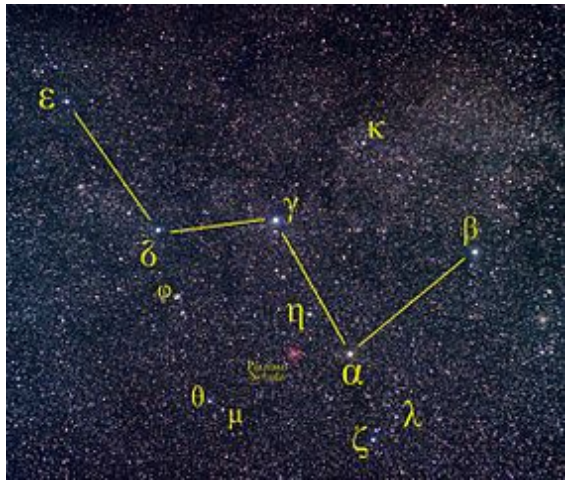



```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1 ; 3 channels
 rgb2 ; 3 channels
 0.6) ; scalar
```



Credit: Wikimedia user Sadalsuud

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```



Credit: Wikimedia user Sadalsuud

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 sky        ; row × col × chan
 labels     ; row × col × chan
 img-mask) ; row × col × chan
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 film      ; time × row × col × chan
 audience  ; time × row × col × chan
 vid-mask) ; time × row × col × chan
```

Credit: Wikimedia user Thetawave

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```


```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 [0.0 ... 1.0]) ; time
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

Polymorphic in dimensionality

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \\ 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \\ 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

$$\begin{bmatrix} 0 \end{bmatrix}.$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

atoms: non-aggregate elements

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \\ 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

$$[0].$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

atoms: non-aggregate elements

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \\ 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

shape: size in each dimension

$$[0].$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7} \quad 2$$

atoms: non-aggregate elements

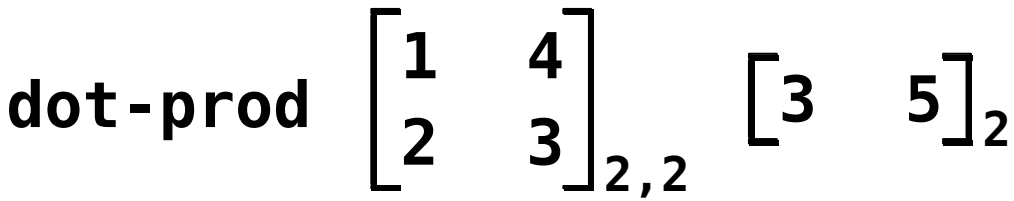
$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \\ 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2} \quad 3$$

shape: size in each dimension

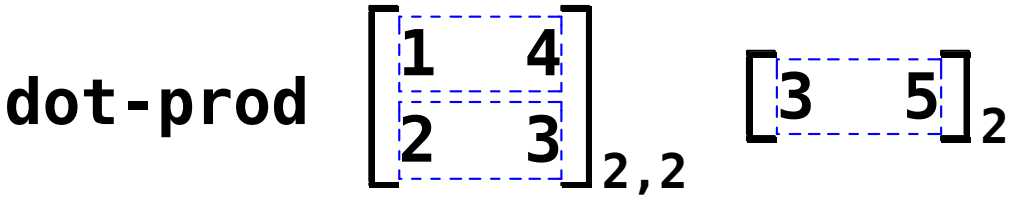
$$[0]. \quad 0$$

rank: length of shape
i.e., number of dimensions

Programming model

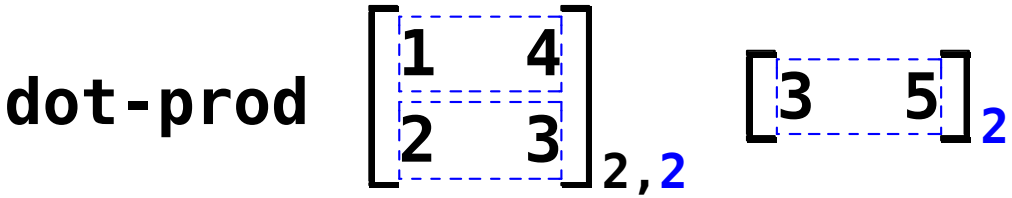


Programming model



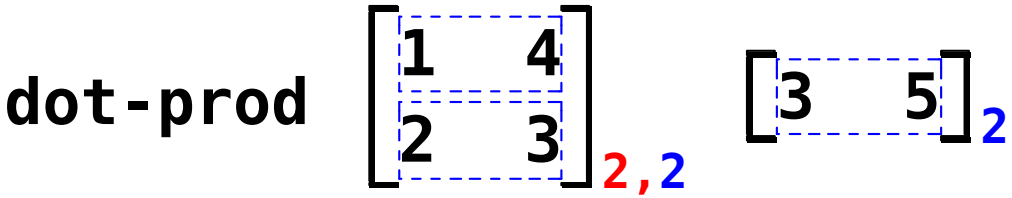
cell: basic unit function operates on

Programming model



cell: basic unit function operates on

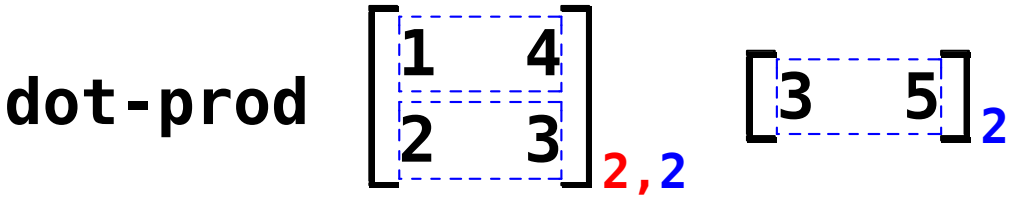
Programming model



cell: basic unit function operates on

frame: structure around cells

Programming model

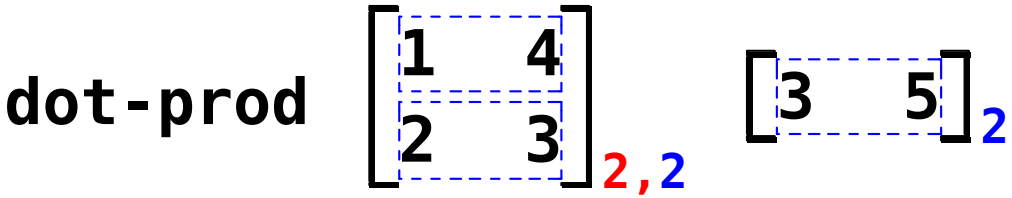


cell: basic unit function operates on

frame: structure around cells

function applied to each cell

Programming model



cell: basic unit function operates on

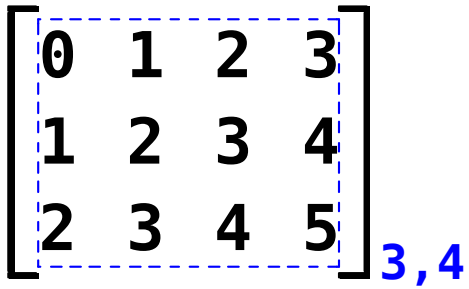
frame: structure around cells

function applied to each cell

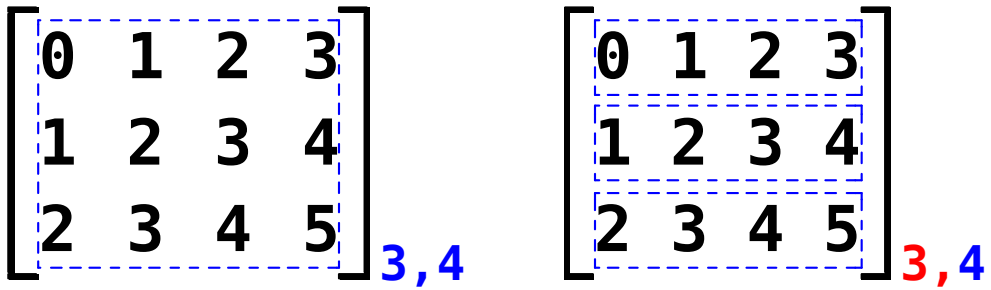
results reassembled in frame

Rank n array can split n+1 ways

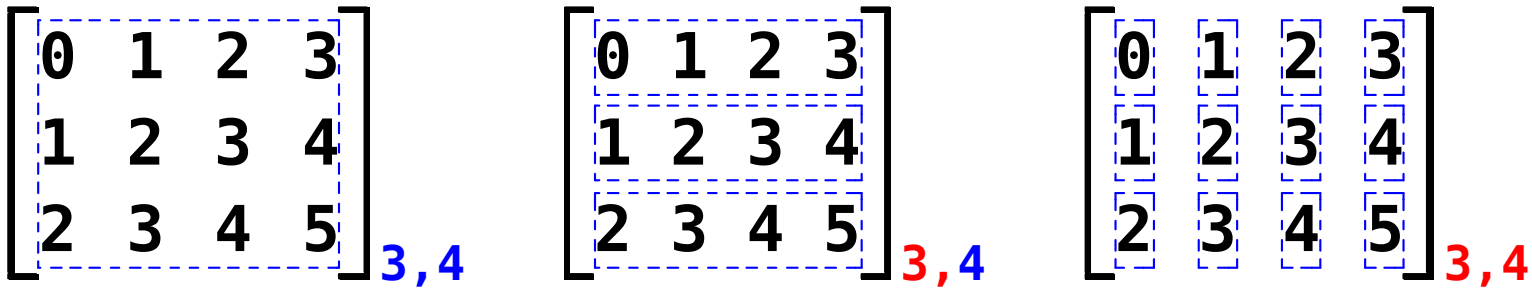
Rank n array can split n+1 ways



Rank n array can split n+1 ways



Rank n array can split n+1 ways

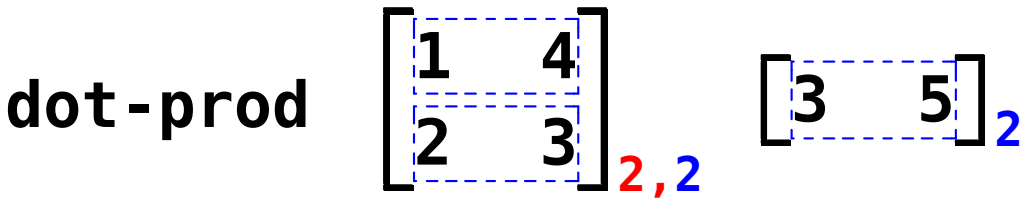


Programming model

Frame-of-cells split chosen by function

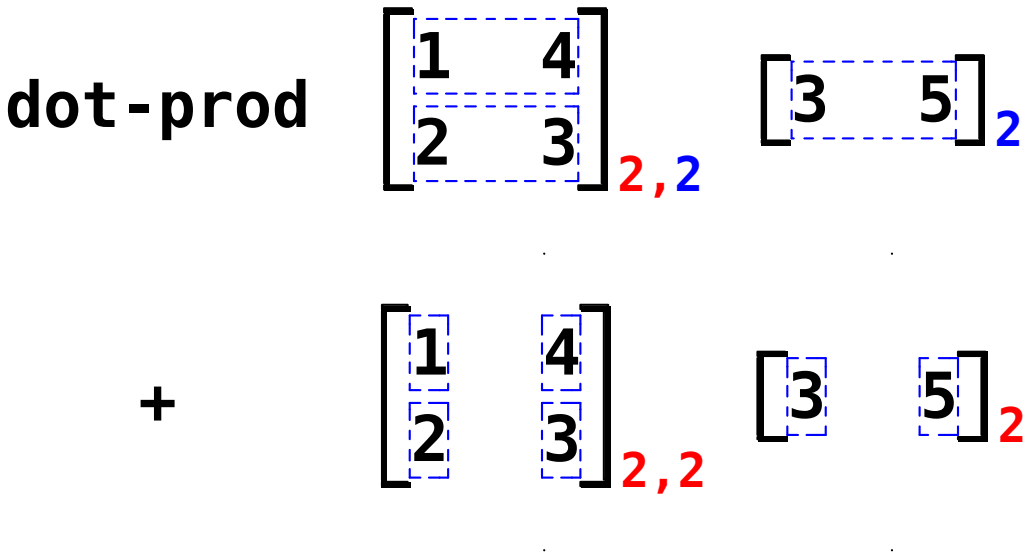
Programming model

Frame-of-cells split chosen by function



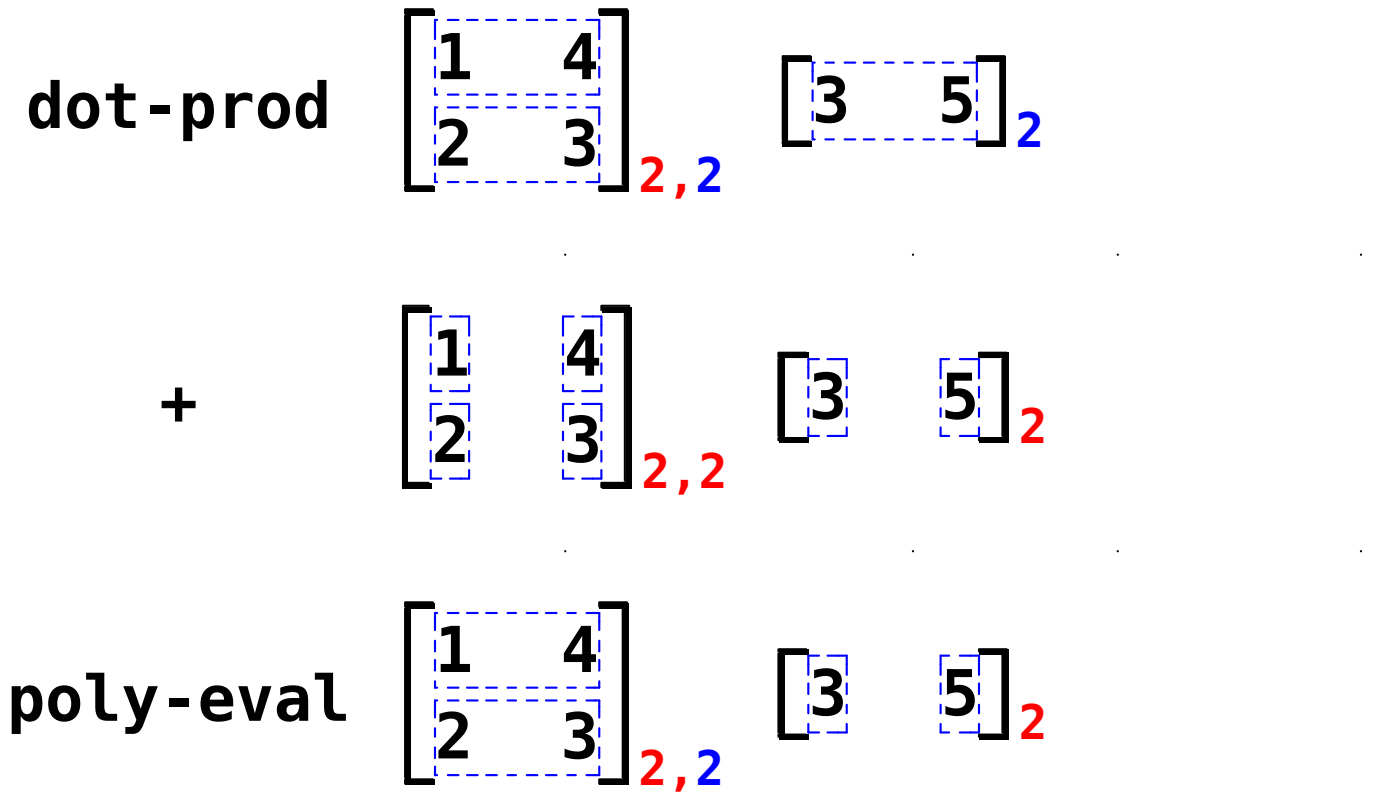
Programming model

Frame-of-cells split chosen by function



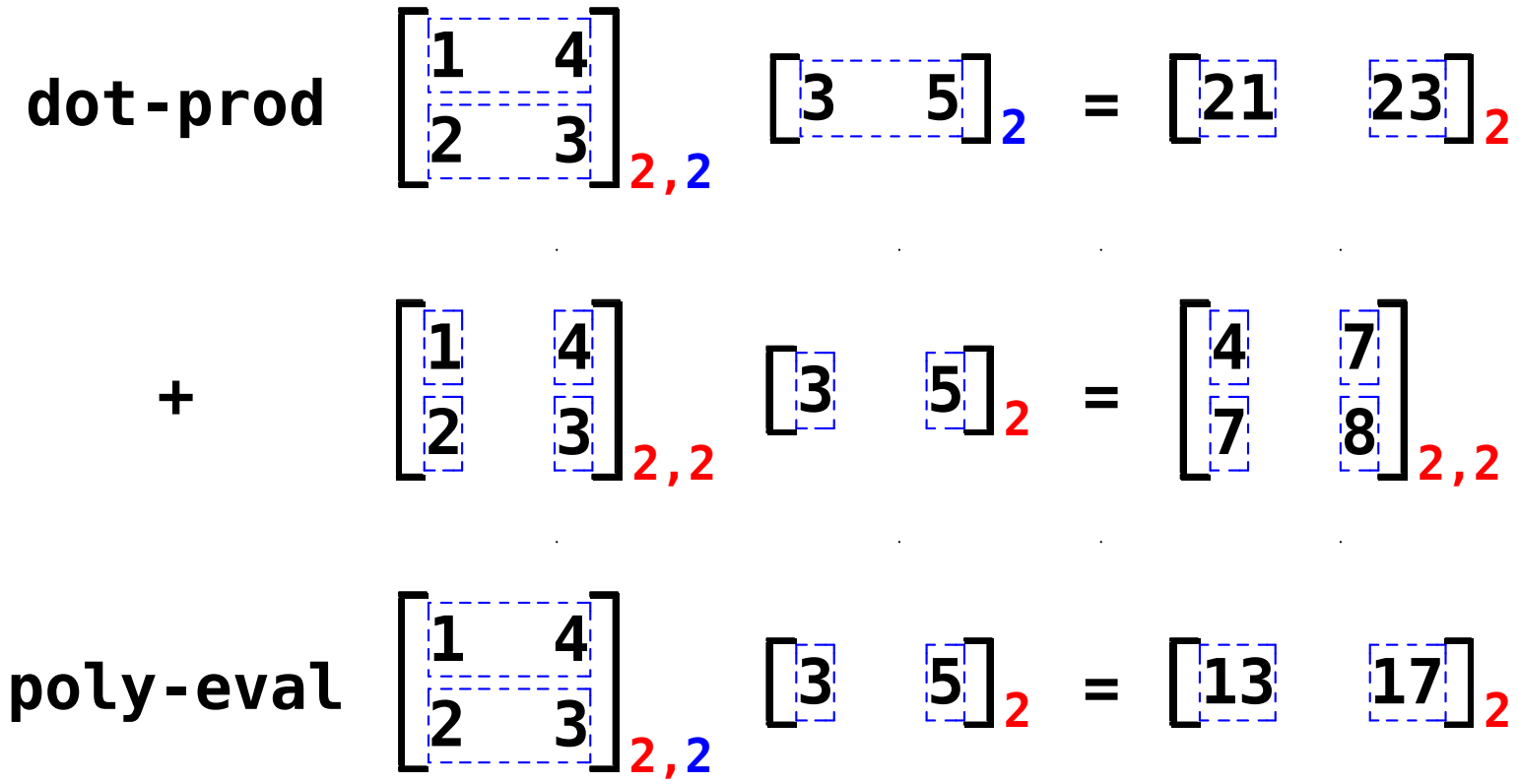
Programming model

Frame-of-cells split chosen by function

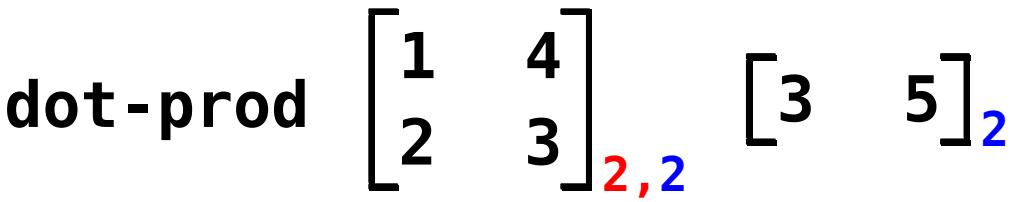


Programming model

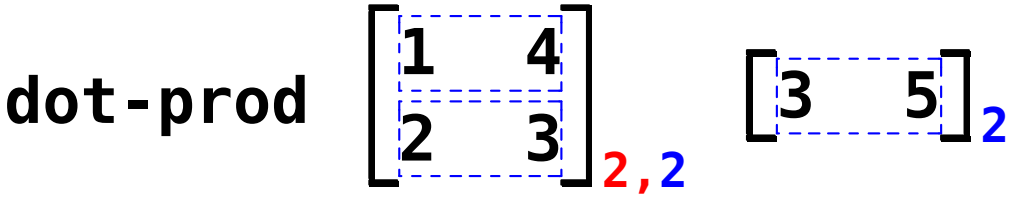
Frame-of-cells split chosen by function



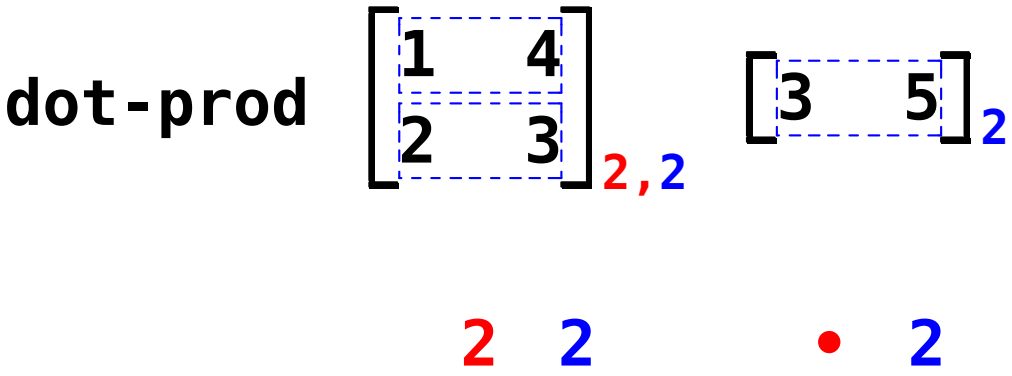
Evaluation



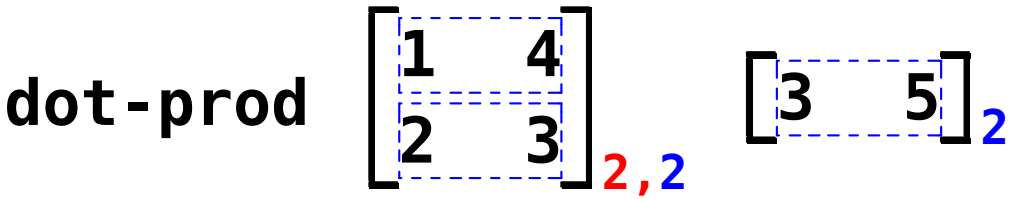
Evaluation



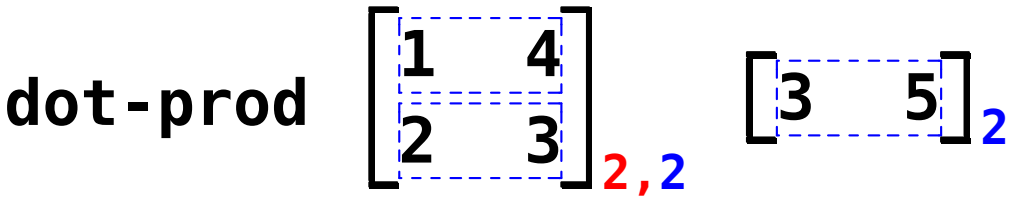
Evaluation



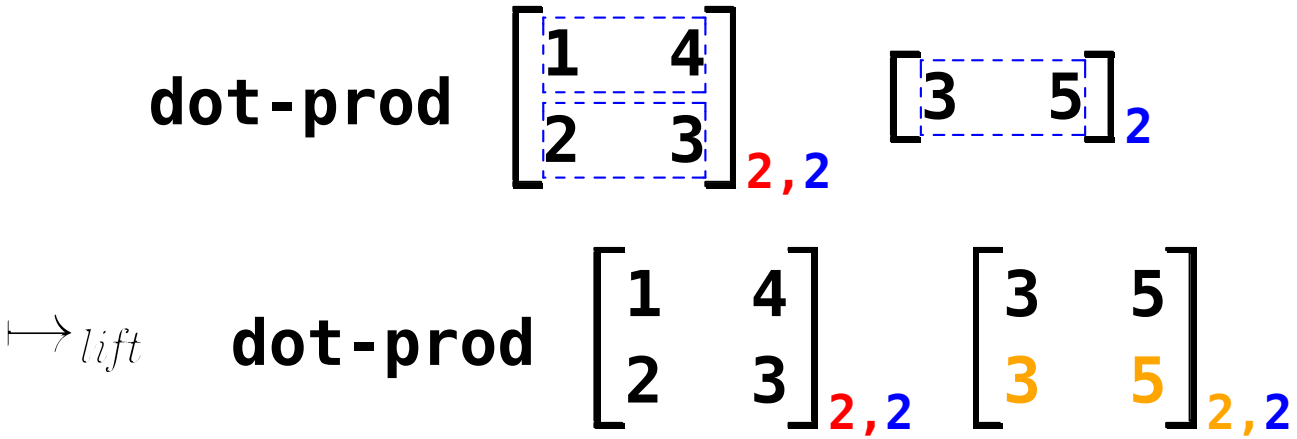
Evaluation



Evaluation



Evaluation



Raise into principal frame by replication

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix}\texttt{1} & \texttt{4}\\ \texttt{2} & \texttt{3}\end{bmatrix}_{\texttt{2,2}}\ \begin{bmatrix}\texttt{3} & \texttt{5}\\ \texttt{3} & \texttt{5}\end{bmatrix}_{\texttt{2,2}}$$

$$\mapsto_{map}\ \begin{bmatrix}(\texttt{dot-prod}\ [\texttt{1}\ \ \texttt{4}]_{\texttt{2}}\ [\texttt{3}\ \ \texttt{5}]_{\texttt{2}})\\ (\texttt{dot-prod}\ [\texttt{2}\ \ \texttt{3}]_{\texttt{2}}\ [\texttt{3}\ \ \texttt{5}]_{\texttt{2}})\end{bmatrix}_{\texttt{2}}$$

Apply function cell-wise

Evaluation

$$\left[\begin{array}{l}\left(\texttt{dot-prod}\ \left[\begin{array}{cc}1 & 4\end{array}\right]_2\ \left[\begin{array}{cc}3 & 5\end{array}\right]_2\right)\\ \left(\texttt{dot-prod}\ \left[\begin{array}{cc}2 & 3\end{array}\right]_2\ \left[\begin{array}{cc}3 & 5\end{array}\right]_2\right)\end{array}\right]_2$$

$$\mapsto_{\delta^2} \left[\begin{array}{l}[23].\\ [21].\end{array}\right]_2$$

Compute each result cell

Evaluation

$$\left[\left[23\right].\left[21\right].\right]_2$$

$$\mapsto_{collapse} \left[\begin{matrix}23\\21\end{matrix}\right]_2$$

Collapse frame-of-cells to single array

$$\texttt{dot-prod}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2} \mapsto_{lift}$$

$$\texttt{dot-prod}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 3 & 5\end{bmatrix}_{2,2} \mapsto_{map}$$

$$\begin{bmatrix}(\texttt{dot-prod}\ \begin{bmatrix}1 & 4\end{bmatrix}_{2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2})\\ (\texttt{dot-prod}\ \begin{bmatrix}2 & 3\end{bmatrix}_{2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2})\end{bmatrix}_{2} \mapsto^{2}_{\delta}$$

$$\begin{bmatrix}[23].\\ [21].\end{bmatrix}_{2} \mapsto_{collapse} \begin{bmatrix}23\\ 21\end{bmatrix}_{2}$$

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1
 rgb2
 0.6)
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1
 rgb2
 0.6)
```

$\mapsto_{lift}$

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1
 rgb2
 [0.6   0.6   0.6]3)
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1
 rgb2
 0.6)
```

$\mapsto_{lift}$

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1
 rgb2
 [0.6  0.6  0.6]3)
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 ; time
 [0.0 ... 1.0])
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 ; time
 [0.0 ... 1.0])
```

$\mapsto_{lift}$

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1
 scene2
 ; time × row × col × chan
 [0.0 ... 0.0 ... 1.0 ... 1.0])
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 ; time
 [0.0 ... 1.0])
```

$\mapsto_{lift}$

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1
 scene2
 ; time × row × col × chan
 [0.0 ... 0.0 ... 1.0 ... 1.0])
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

Frame mismatch: need to align arguments differently

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$+^{1,1}$

Reranking

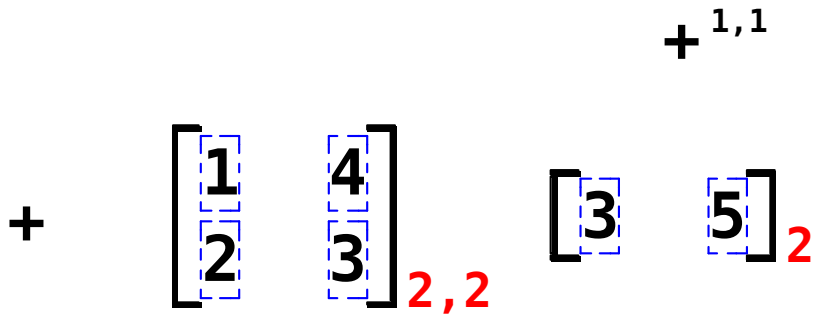
```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

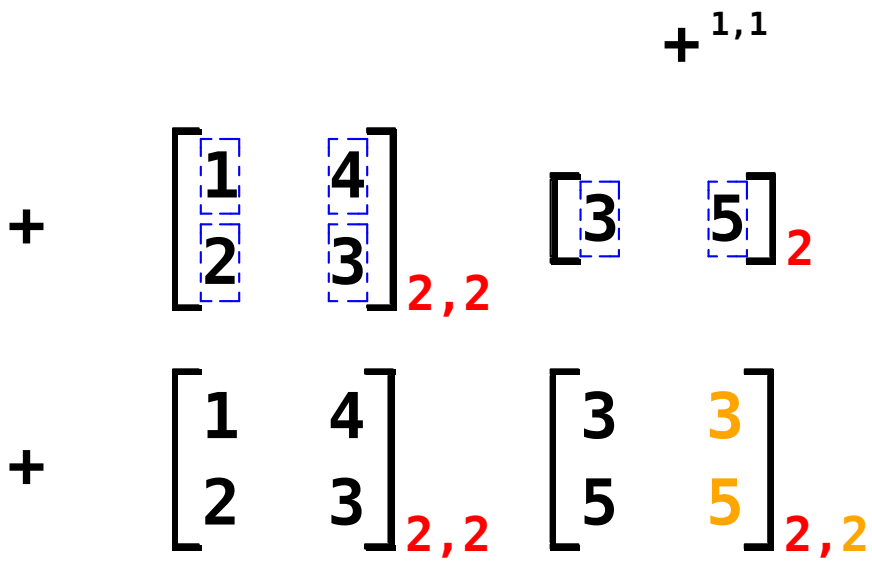
Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \quad \begin{bmatrix} 3 & 3 \\ 5 & 5 \end{bmatrix}_{2,2} = \begin{bmatrix} 4 & 7 \\ 7 & 8 \end{bmatrix}_{2,2}$$

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

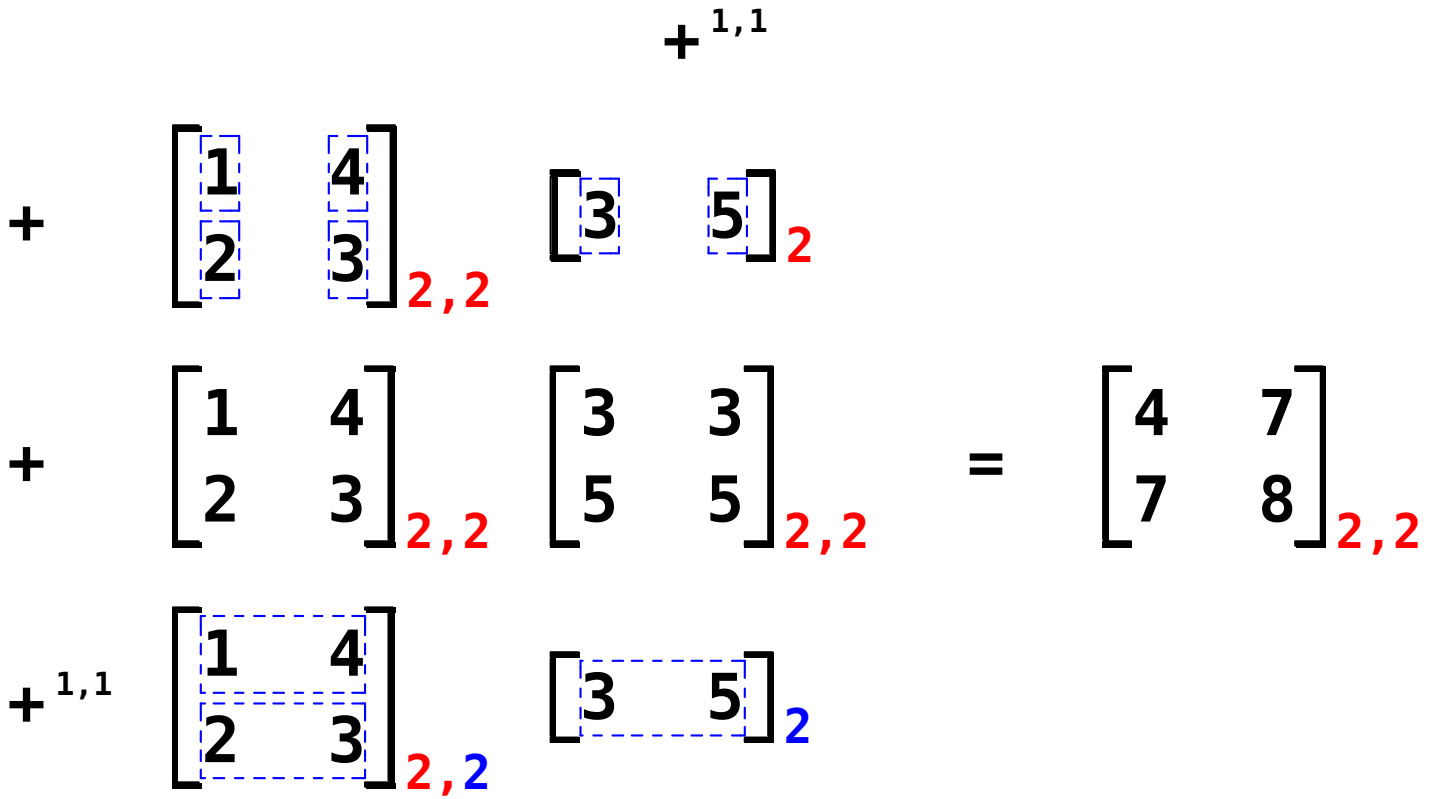
$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 3 \\ 5 & 5 \end{bmatrix}_{2,2} = \begin{bmatrix} 4 & 7 \\ 7 & 8 \end{bmatrix}_{2,2}$$

$$+^{1,1} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

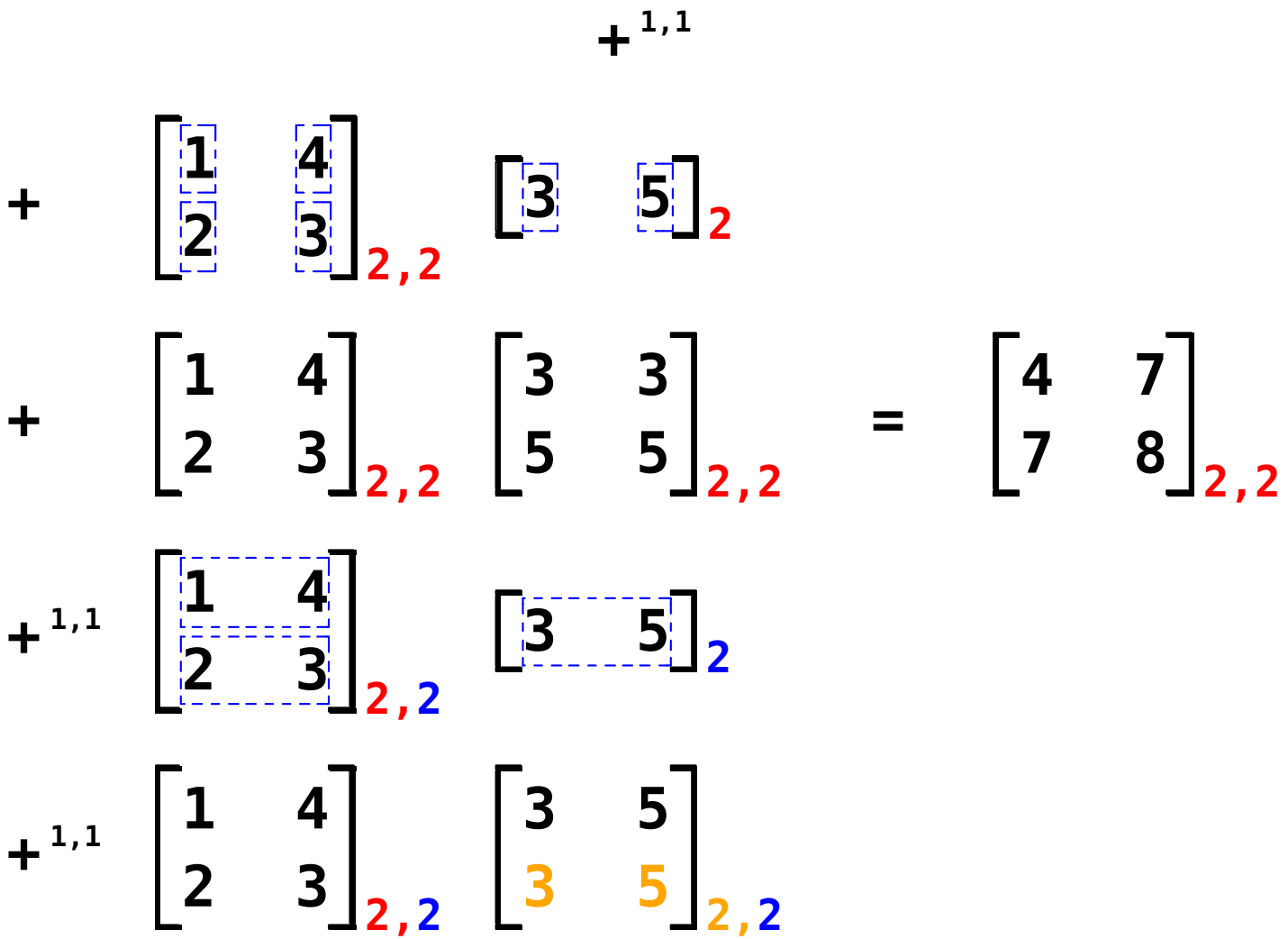
Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

$$+ \begin{bmatrix} \boxed{1} & \boxed{4} \\ \boxed{2} & \boxed{3} \end{bmatrix}_{2,2} \begin{bmatrix} \boxed{3} & \boxed{5} \end{bmatrix}_{2}$$

$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 3 \\ 5 & 5 \end{bmatrix}_{2,2} = \begin{bmatrix} 4 & 7 \\ 7 & 8 \end{bmatrix}_{2,2}$$

$$+^{1,1} \begin{bmatrix} \boxed{1} & \boxed{4} \\ \boxed{2} & \boxed{3} \end{bmatrix}_{2,2} \begin{bmatrix} \boxed{3} & \boxed{5} \end{bmatrix}_{2}$$

$$+^{1,1} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \\ 3 & 5 \end{bmatrix}_{2,2} = \begin{bmatrix} 4 & 9 \\ 5 & 8 \end{bmatrix}_{2,2}$$

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* lo α) (* hi (- 1 α)))))
```

```
((λ [(lo ?) (hi ?) (α ?)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* lo α) (* hi (- 1 α)))))
```

```
((λ [(lo 1) (hi ?) (α ?)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* lo α) (* hi (- 1 α)))))
```

```
((λ [(lo 1) (hi 1) (α ?)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* lo α) (* hi (- 1 α)))))
```

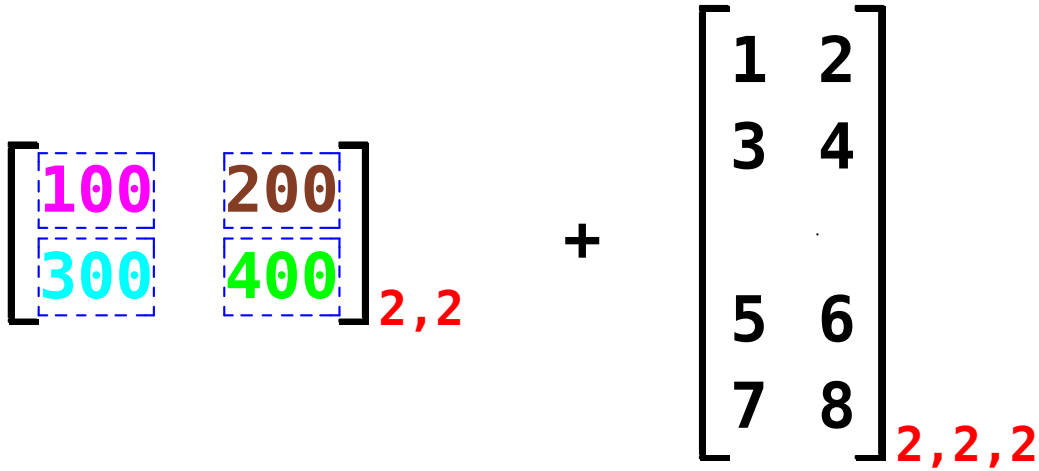
```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* lo α) (* hi (- 1 α)))))
```

```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo          ; row × col × chan
 rgb-expanded ; row × col × chan
 [0.6 ... 0.6]) ; row × col
```

Axis alignment



Axis alignment

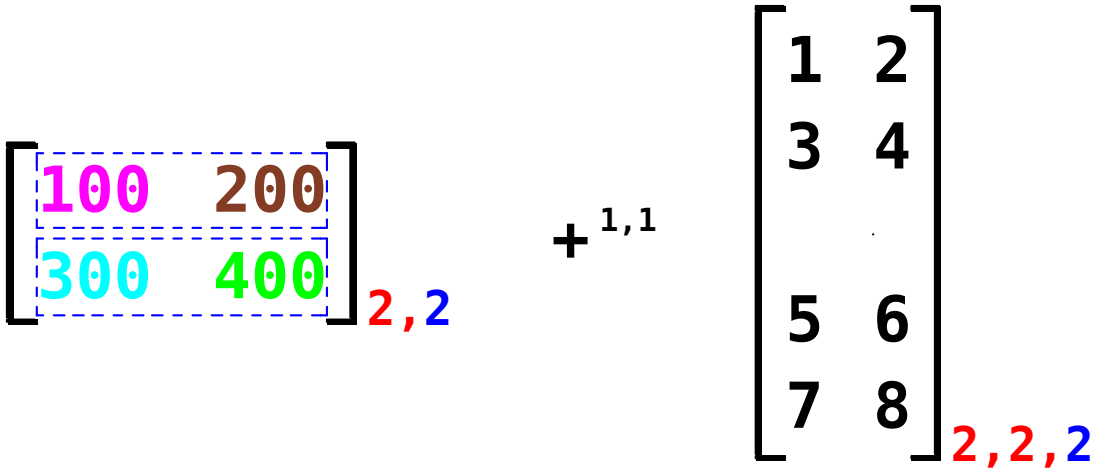
$$\begin{bmatrix} 100 & 100 \\ 200 & 200 \\ \\ 300 & 300 \\ 400 & 400 \end{bmatrix}_{2,2,2} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

Axis alignment

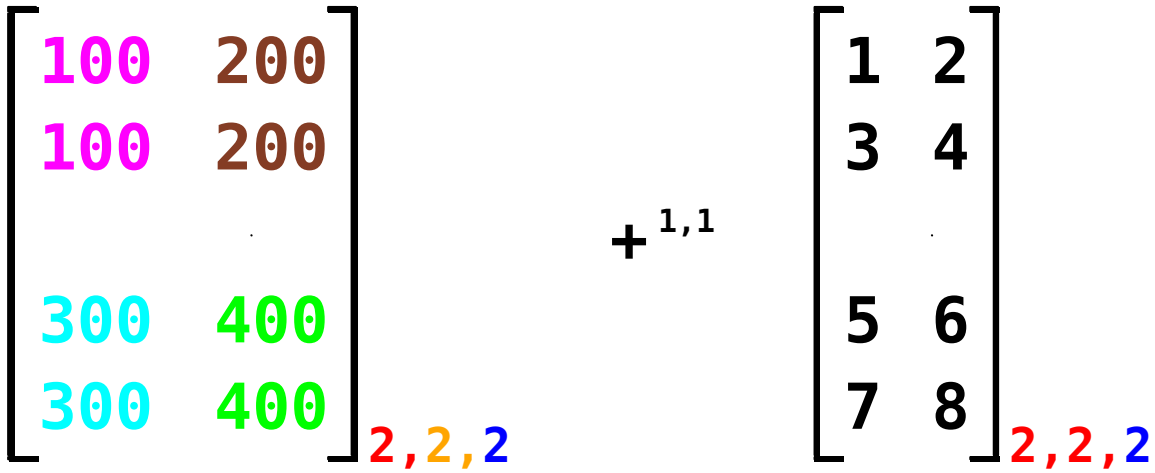
$$\begin{bmatrix} 100 & 100 \\ 200 & 200 \\ \\ 300 & 300 \\ 400 & 400 \end{bmatrix}_{2,2,2} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

$$\begin{bmatrix} 101 & 102 \\ 203 & 204 \\ \\ 305 & 306 \\ 407 & 408 \end{bmatrix}_{2,2,2}$$

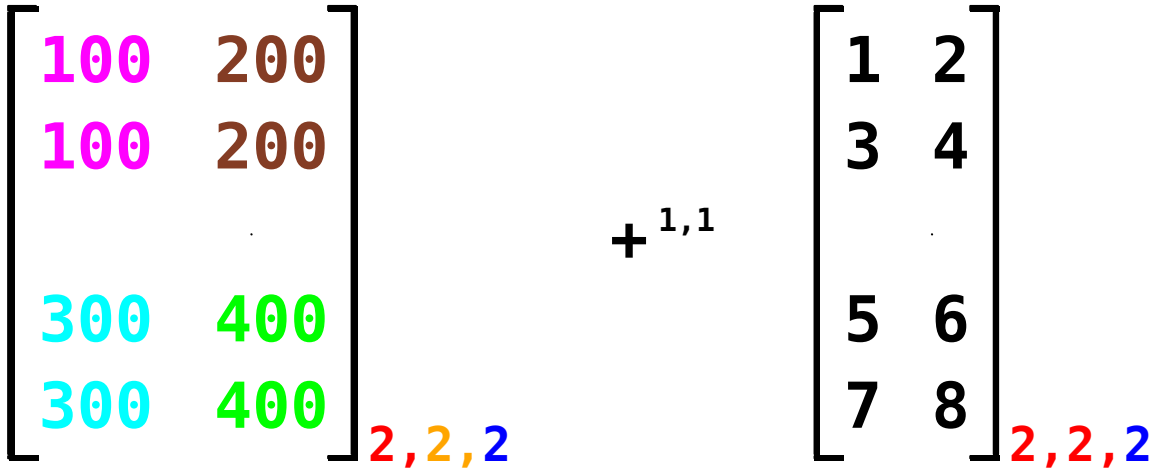
Axis alignment

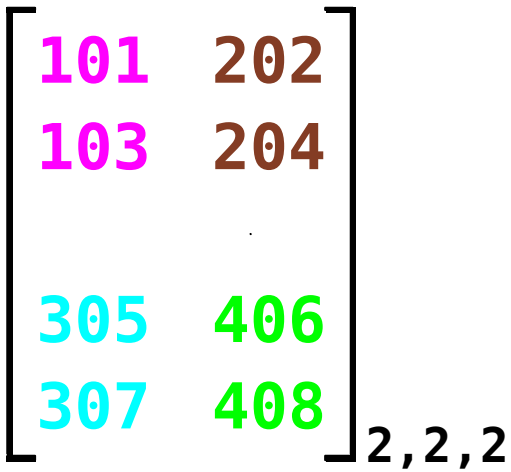


Axis alignment

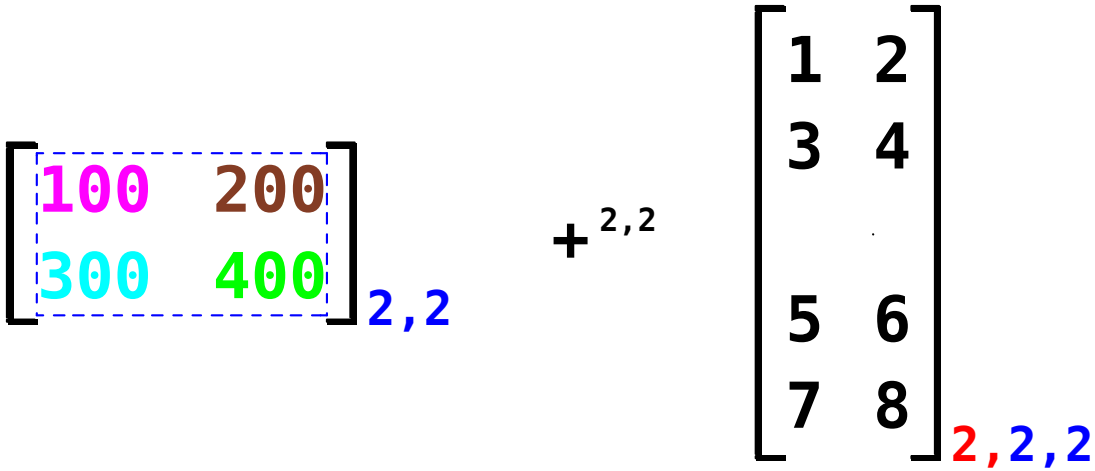


Axis alignment





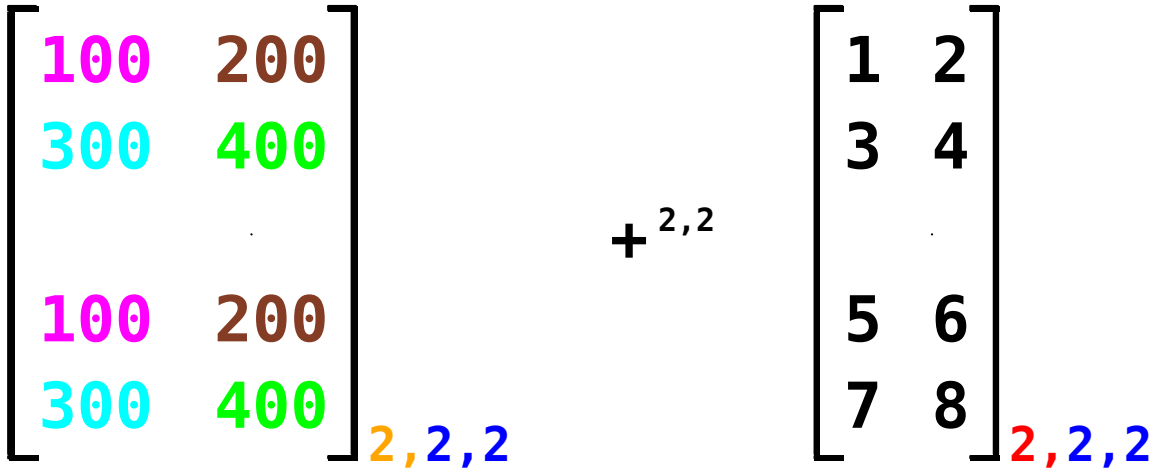
Axis alignment

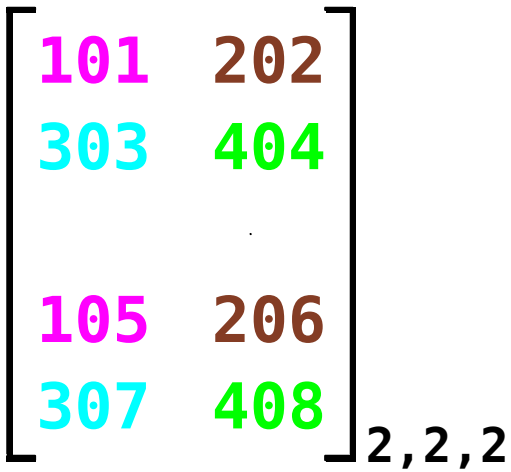


Axis alignment

$$\begin{bmatrix} 100 & 200 \\ 300 & 400 \\ \\ 100 & 200 \\ 300 & 400 \end{bmatrix}_{2,2,2} \overset{2,2}{+} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

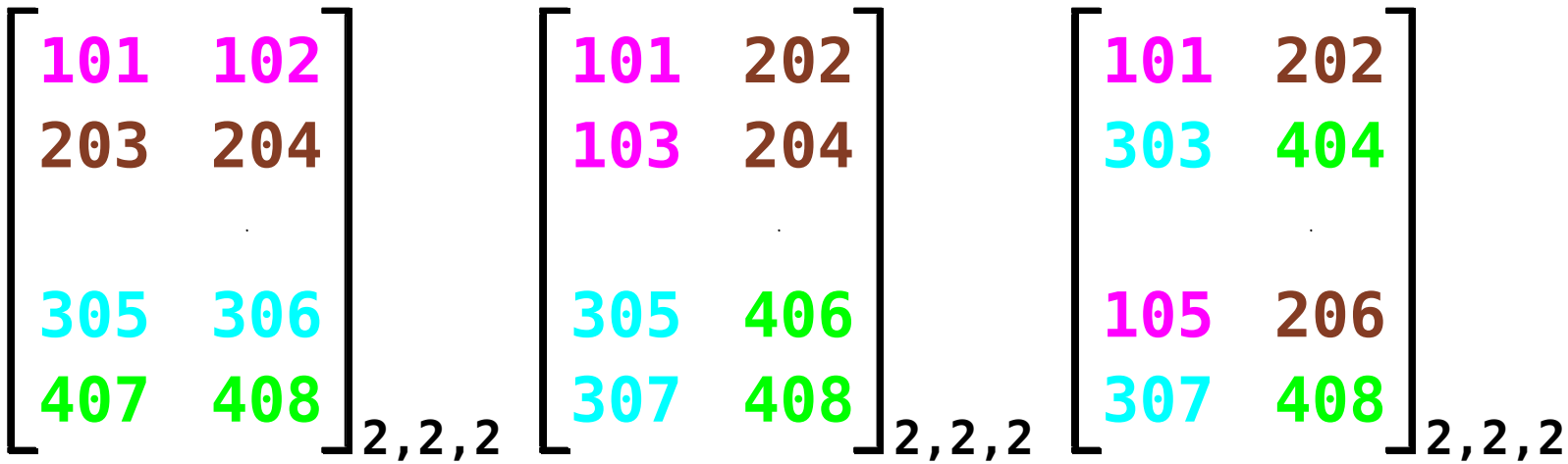
Axis alignment





Axis alignment

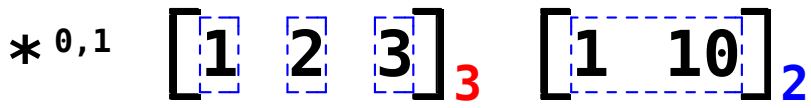
different expected rank leads to different lifting behavior



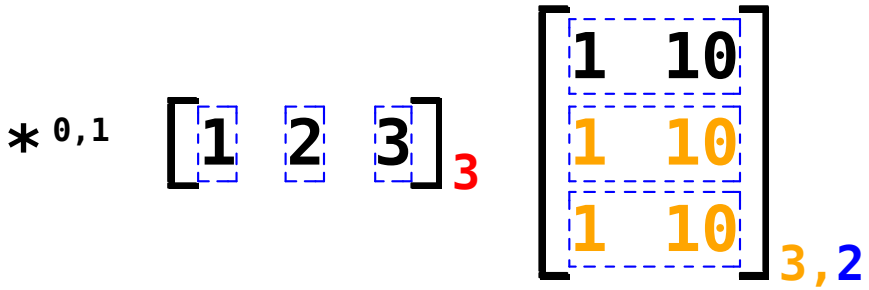
Axis alignment

$$*^{0,1}\ [1\ \ 2\ \ 3]_3\ \ [1\ \ 10]_2$$

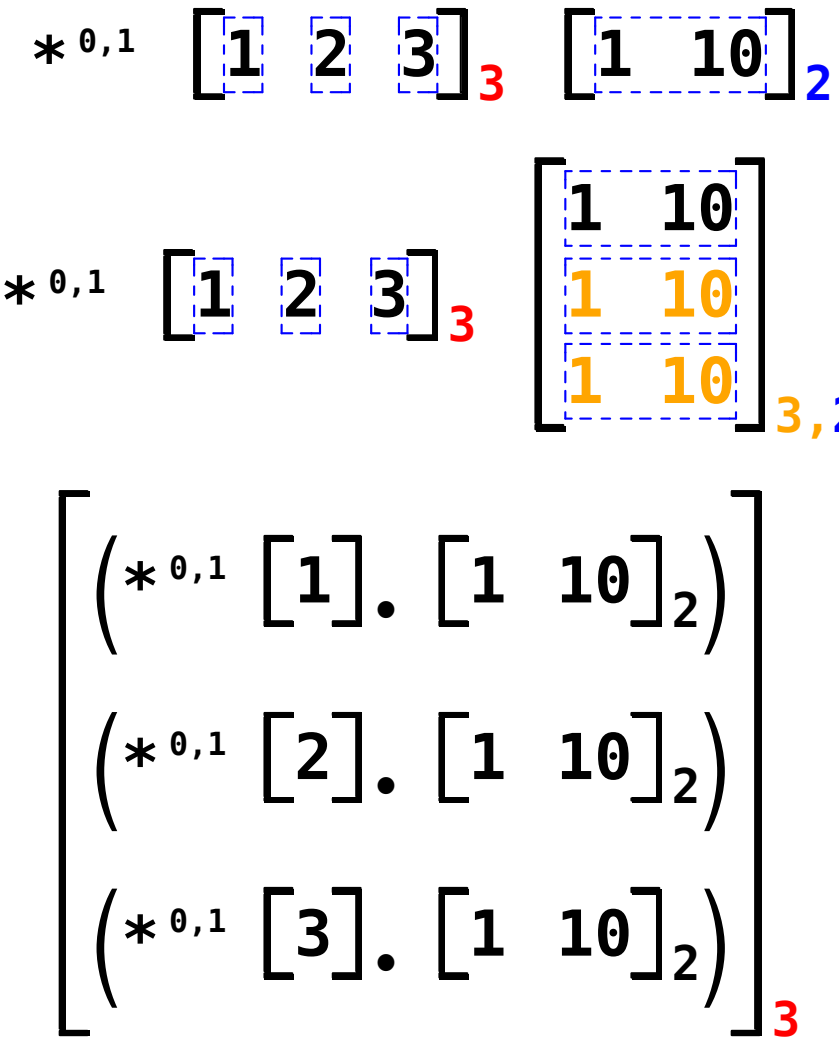
Axis alignment



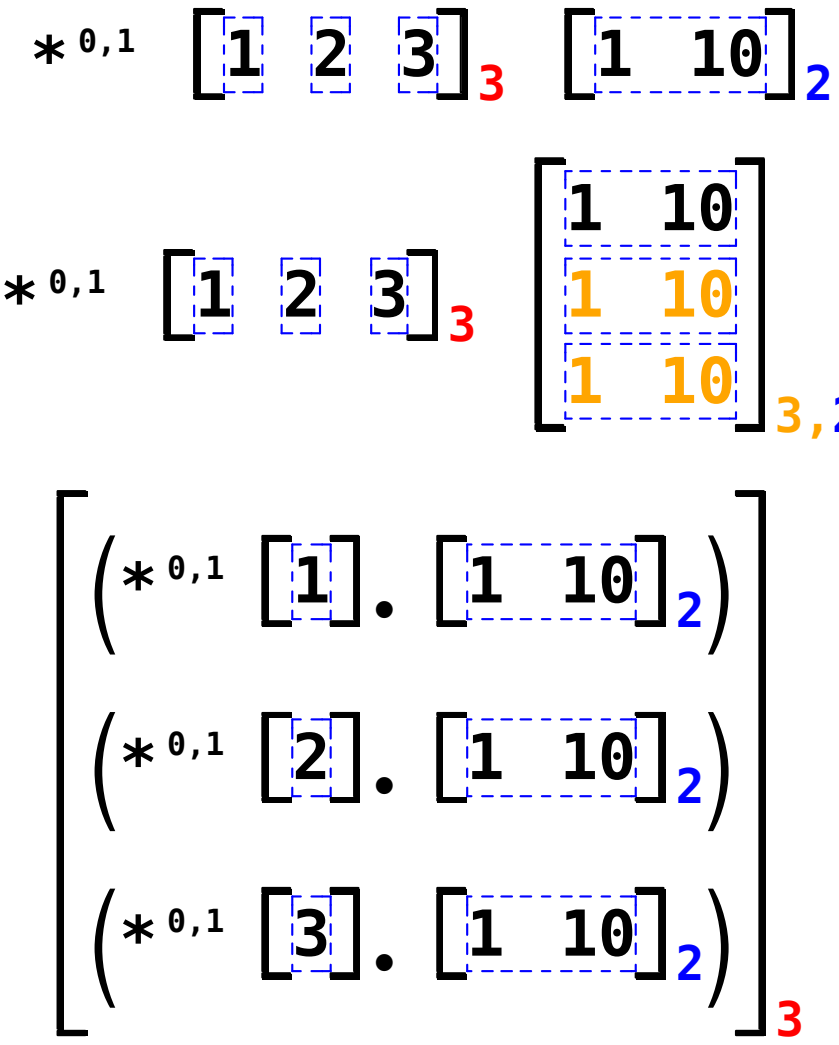
Axis alignment



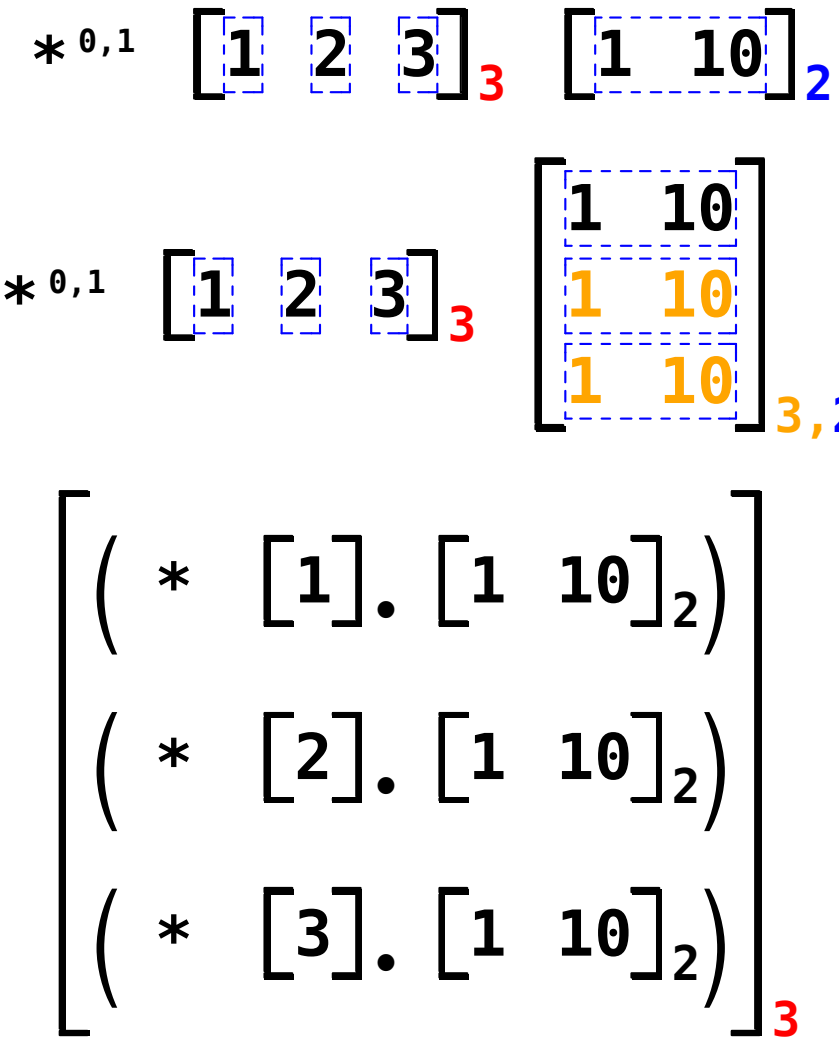
Axis alignment



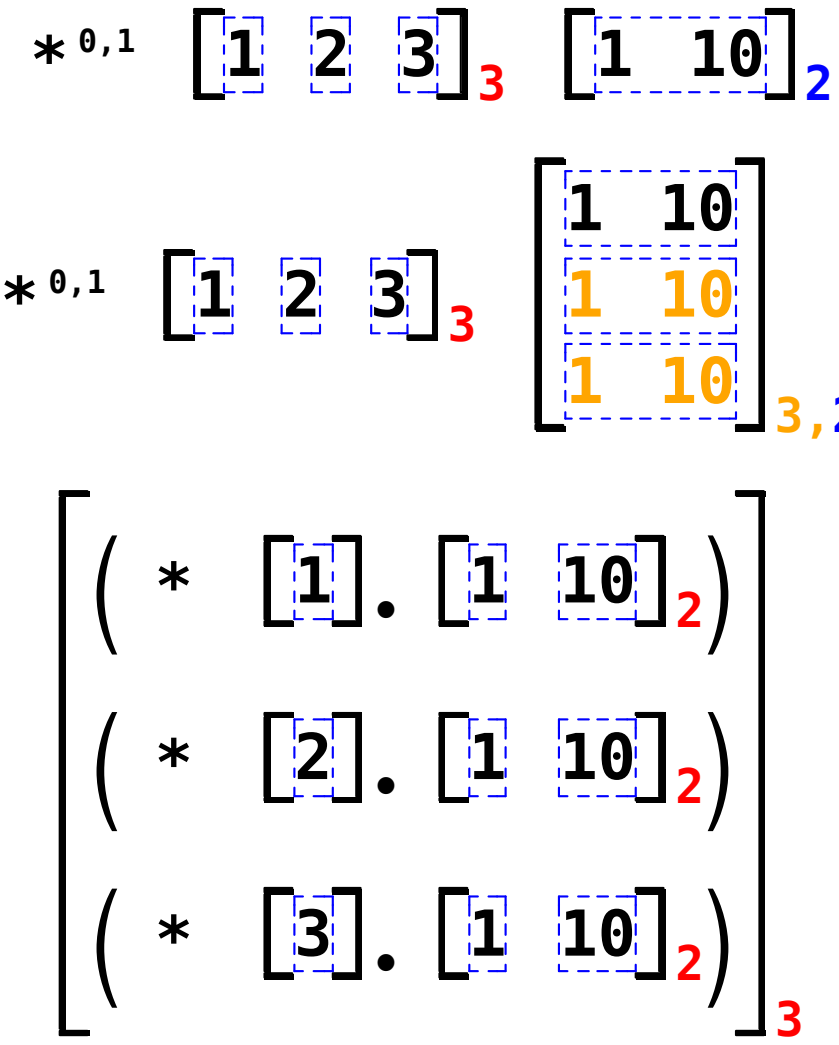
Axis alignment



Axis alignment



Axis alignment



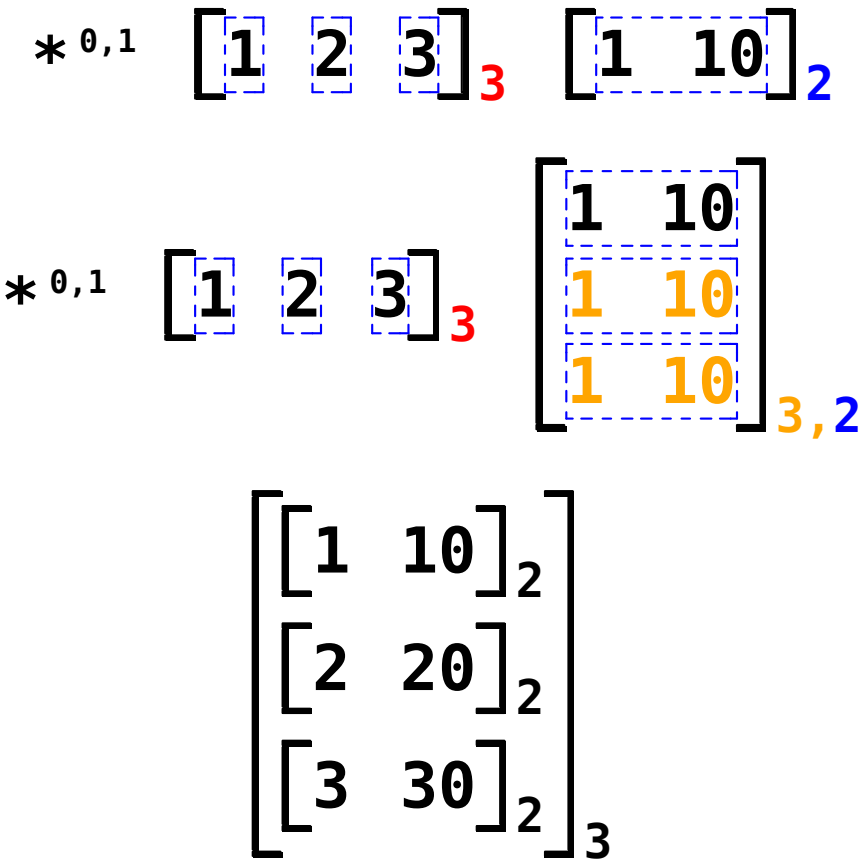
Axis alignment

$$*^{0,1}\ [1\ \ 2\ \ 3]_3\ \ [1\ \ 10]_2$$

$$*^{0,1}\ [1\ \ 2\ \ 3]_3\ \ \begin{bmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{bmatrix}_{3,2}$$

$$\begin{bmatrix} (*\ [1\ \ 1]_2\ [1\ \ 10]_2) \\ (*\ [2\ \ 2]_2\ [1\ \ 10]_2) \\ (*\ [3\ \ 3]_2\ [1\ \ 10]_2) \end{bmatrix}_3$$

Axis alignment



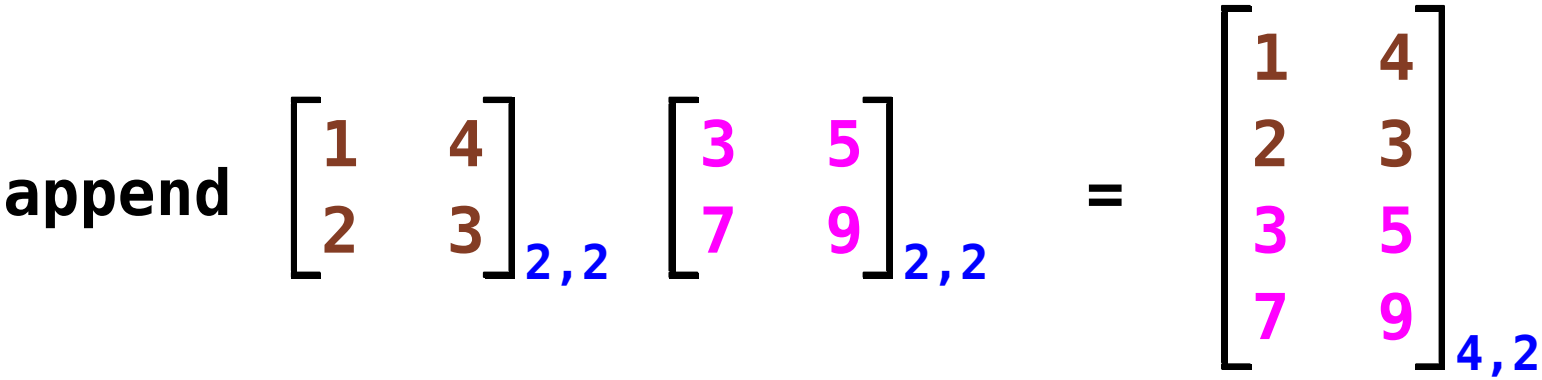
Axis alignment

$$*^{0,1}\ \underbrace{\begin{bmatrix}1 & 2 & 3\end{bmatrix}}_{3}\ \underbrace{\begin{bmatrix}1 & 10\end{bmatrix}}_{2}$$

$$*^{0,1}\ \underbrace{\begin{bmatrix}1 & 2 & 3\end{bmatrix}}_{3}\ \underbrace{\begin{bmatrix}1 & 10\\ 1 & 10\\ 1 & 10\end{bmatrix}}_{3,2}$$

$$\begin{bmatrix}1 & 10\\ 2 & 20\\ 3 & 30\end{bmatrix}_{3,2}$$

Axis alignment

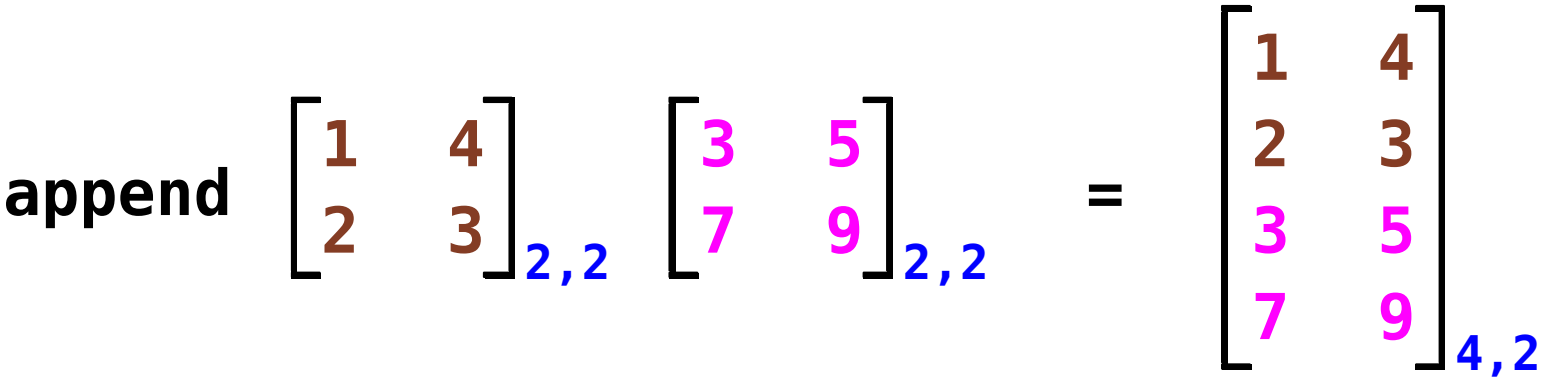


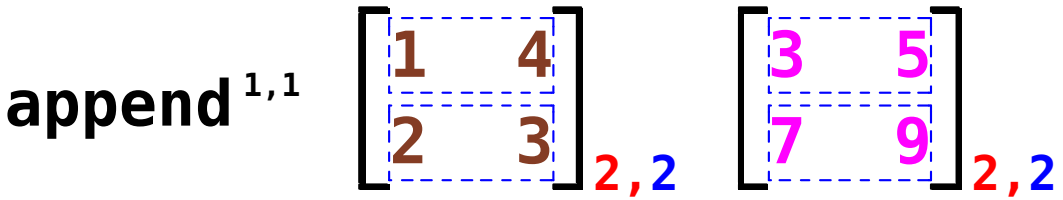
Axis alignment

$$\textbf{append}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2} = \begin{bmatrix}1 & 4\\ 2 & 3\\ 3 & 5\\ 7 & 9\end{bmatrix}_{4,2}$$

$$\textbf{append}^{1,1}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

Axis alignment





Axis alignment

$$\mathbf{append}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2} = \begin{bmatrix}1 & 4\\ 2 & 3\\ 3 & 5\\ 7 & 9\end{bmatrix}_{4,2}$$

$$\mathbf{append}^{1,1}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$= \begin{bmatrix}\left(\mathbf{append}^{1,1}\ \begin{bmatrix}1 & 4\end{bmatrix}_{2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2}\right)\\ \left(\mathbf{append}^{1,1}\ \begin{bmatrix}2 & 3\end{bmatrix}_{2}\ \begin{bmatrix}7 & 9\end{bmatrix}_{2}\right)\end{bmatrix}_{2}$$

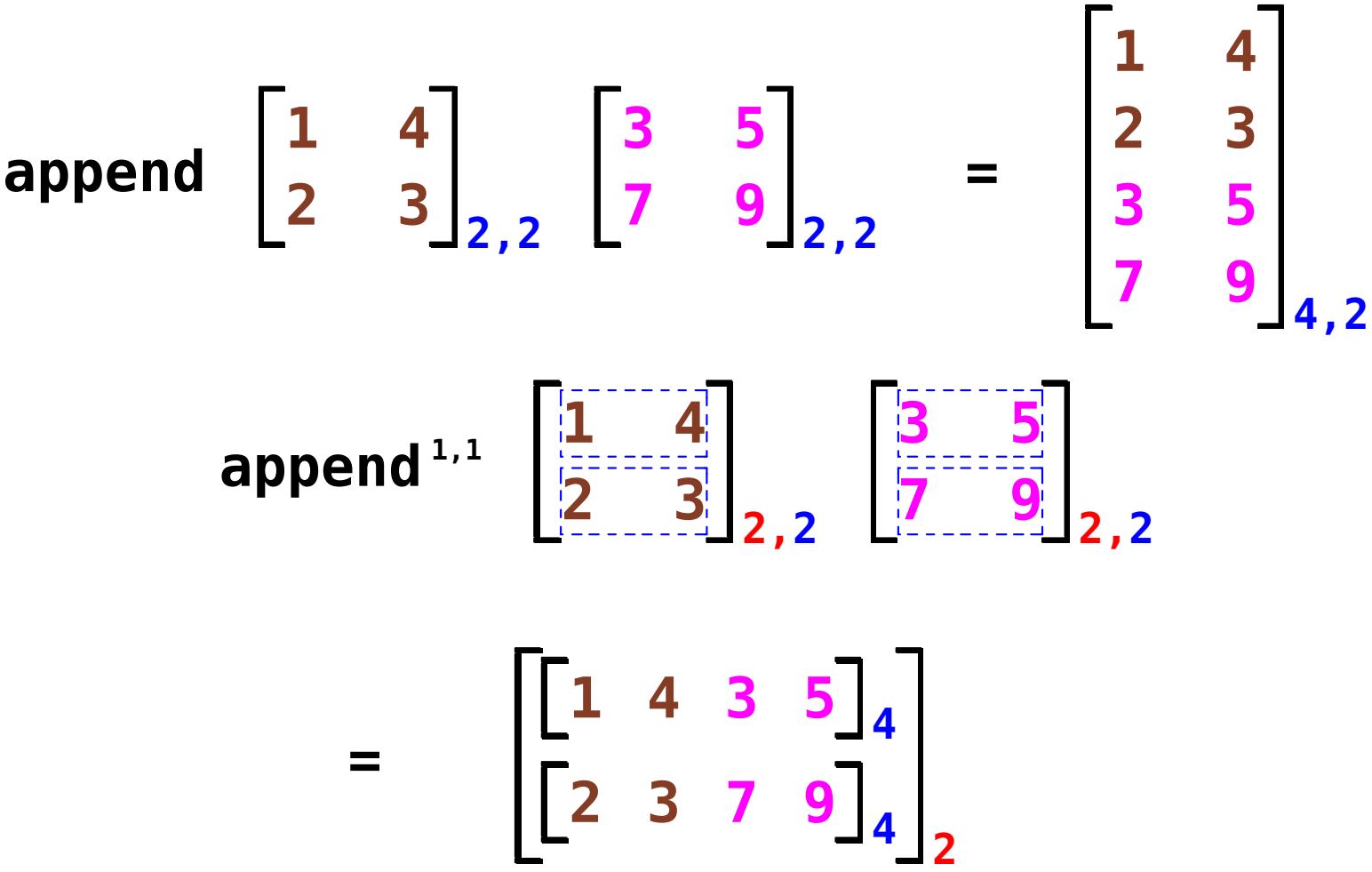
Axis alignment

$$\textbf{append}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2} = \begin{bmatrix}1 & 4\\ 2 & 3\\ 3 & 5\\ 7 & 9\end{bmatrix}_{4,2}$$

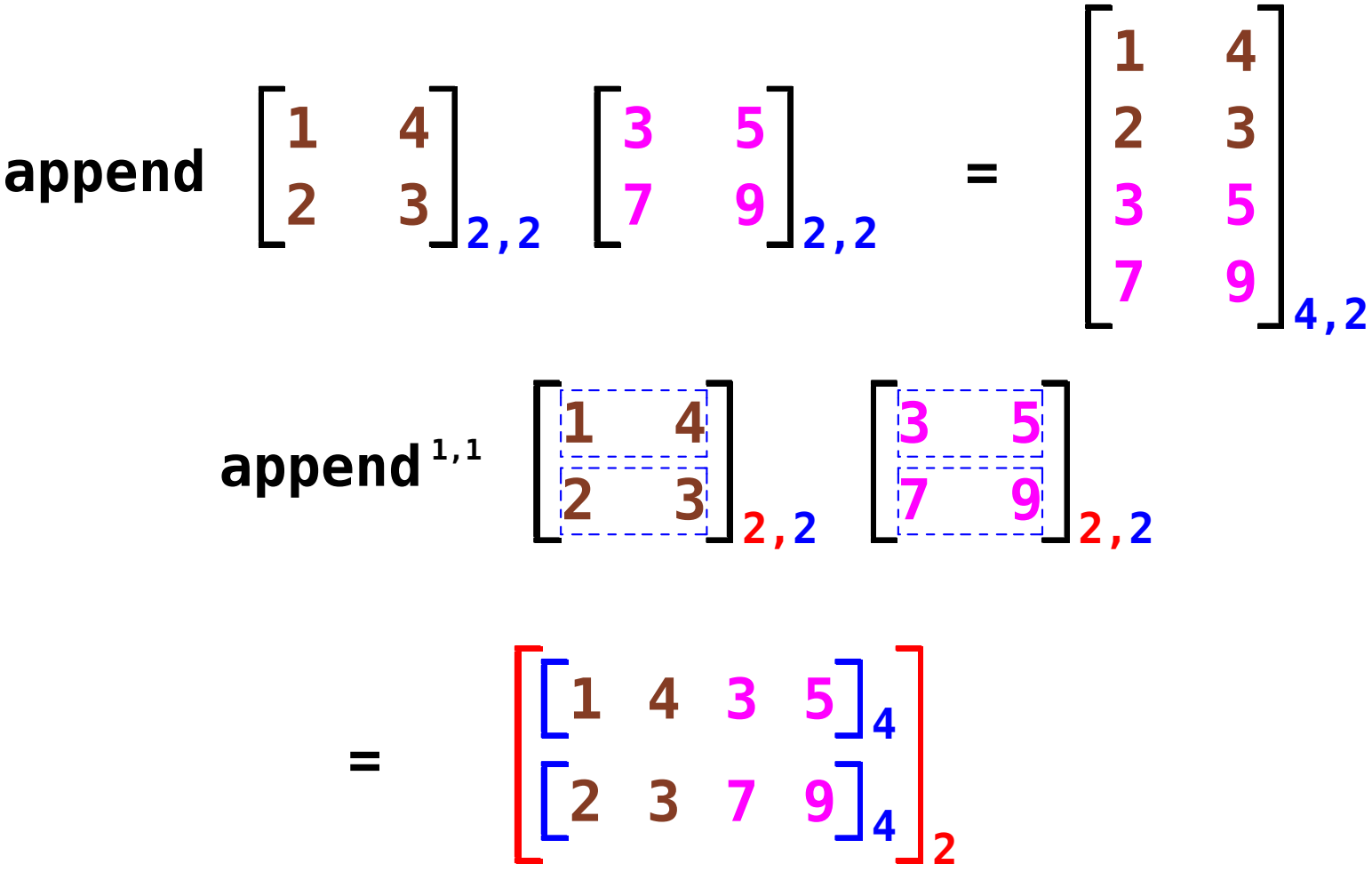
$$\textbf{append}^{1,1}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$= \begin{bmatrix}\left(\textbf{append}\ \begin{bmatrix}1 & 4\end{bmatrix}_{2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2}\right)\\ \left(\textbf{append}\ \begin{bmatrix}2 & 3\end{bmatrix}_{2}\ \begin{bmatrix}7 & 9\end{bmatrix}_{2}\right)\end{bmatrix}_{2}$$

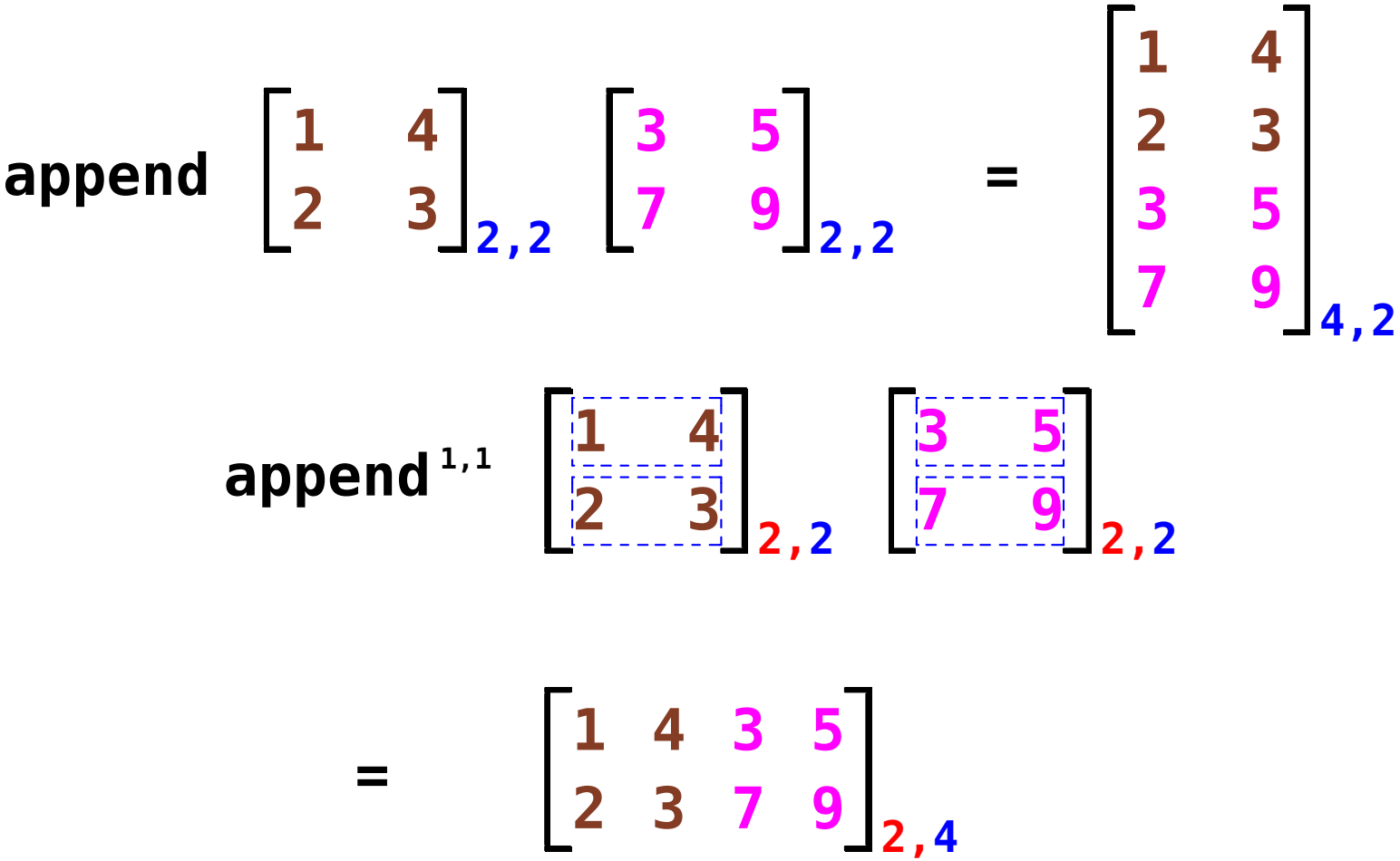
Axis alignment



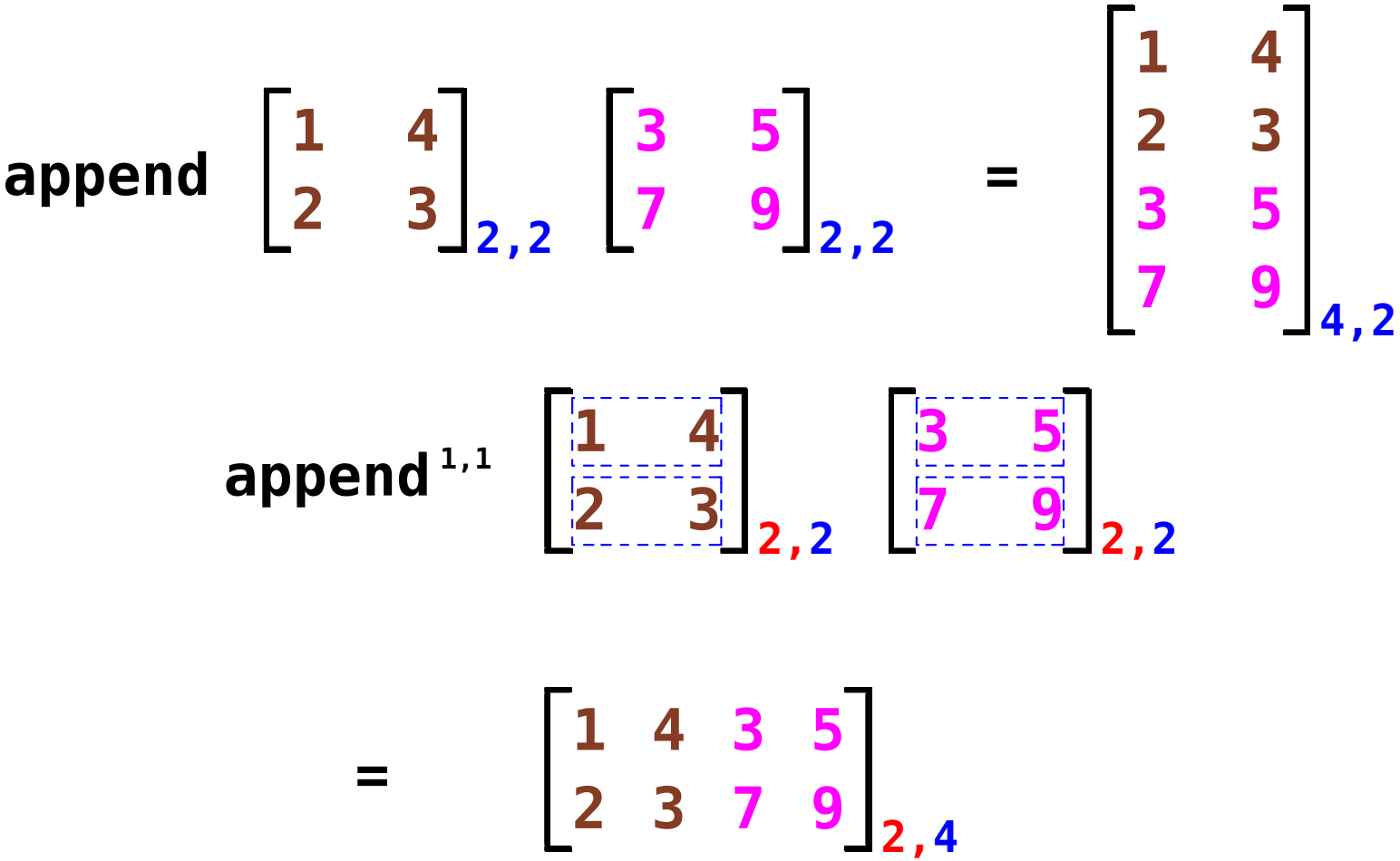
Axis alignment



Axis alignment



Axis alignment



First-class functions

```
repeat [3]. [inc  dec]2
```

First-class functions

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

First-class functions

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

$$(\ \texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2\ )\ [10].$$

First-class functions

repeat [3]. [inc dec]$_2$ ↦ [(+3) (-3)]$_2$

[(+3) (-3)]$_2$ [10].

First-class functions

```
repeat [3]. [inc  dec]₂ ↦ [(+3)  (-3)]₂
```

```
[(+3)  (-3)]₂  [10].
```

array in function position

First-class functions

```
repeat [3]. [inc  dec]₂ ↦ [(+3)  (-3)]₂
```

```
apply   [(+3)  (-3)]₂   [10].
```

array in function position

First-class functions

```
apply [(+3) (-3)]₂ [10].
```

what is **apply**'s expected rank?

```
0, ? ...
```

First-class functions

```
apply [(+3) (-3)]₂ [10].
```

what is **apply**'s expected rank?

```
0, ? ...
```

fill in with ranks expected by **(+3)** and **(-3)**

First-class functions

```
apply [(+3) (-3)]₂ [10].
```

what is **apply**'s expected rank?

```
0, ? ...
```

fill in with ranks expected by **(+3)** and **(-3)**

```
0, 0
```

First-class functions

```
[(+3)  (-3)]2 [10].
```

First-class functions

```
[(+3)  (-3)]2 [10].
```

```
0, 0
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2].
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2].                 0, 0, 0
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0

[vec-norm  vec-sum]2 [1  3  5]3
```

First-class functions

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0

[vec-norm  vec-sum]2 [1  3  5]3            0, 1
```

First-class functions

```
[(+3)  (-3)]₂ [10  10]₂                    0, 0

[+  -  *]₃ [1  3  5]₃ [2  2  2]₃           0, 0, 0

[vec-norm  vec-sum]₂ ⎡1  3  5⎤             0, 1
                     ⎣1  3  5⎦₂,₃
```

First-class functions

```
[(+3)  (-3)]₂ [10  10]₂                    0, 0

[+  -  *]₃ [1  3  5]₃ [2  2  2]₃           0, 0, 0

[vec-norm  vec-sum]₂ ⎡1  3  5⎤             0, 1
                     ⎣1  3  5⎦₂,₃

[vec-norm  mat-inv]₂ ⎡1  3⎤
                     ⎣2  5⎦₂,₂
```

First-class functions

```
[(+3)  (-3)]₂ [10  10]₂                    0, 0

[+  -  *]₃ [1  3  5]₃ [2  2  2]₃           0, 0, 0

[vec-norm  vec-sum]₂ [1  3  5]             0, 1
                     [1  3  5]₂,₃

[vec-norm  mat-inv]₂ [1  3]                not allowed
                     [2  5]₂,₂             (expected ranks
                                           do not match)
```

Prototype implementations

Available on GitHub

`https://github.com/jrslepak/Remora`

`semantics`	Semantic model • step-by-step reductions • type system (more next week)
`remora`	Embedded DSL • built as Racket **`#lang`**