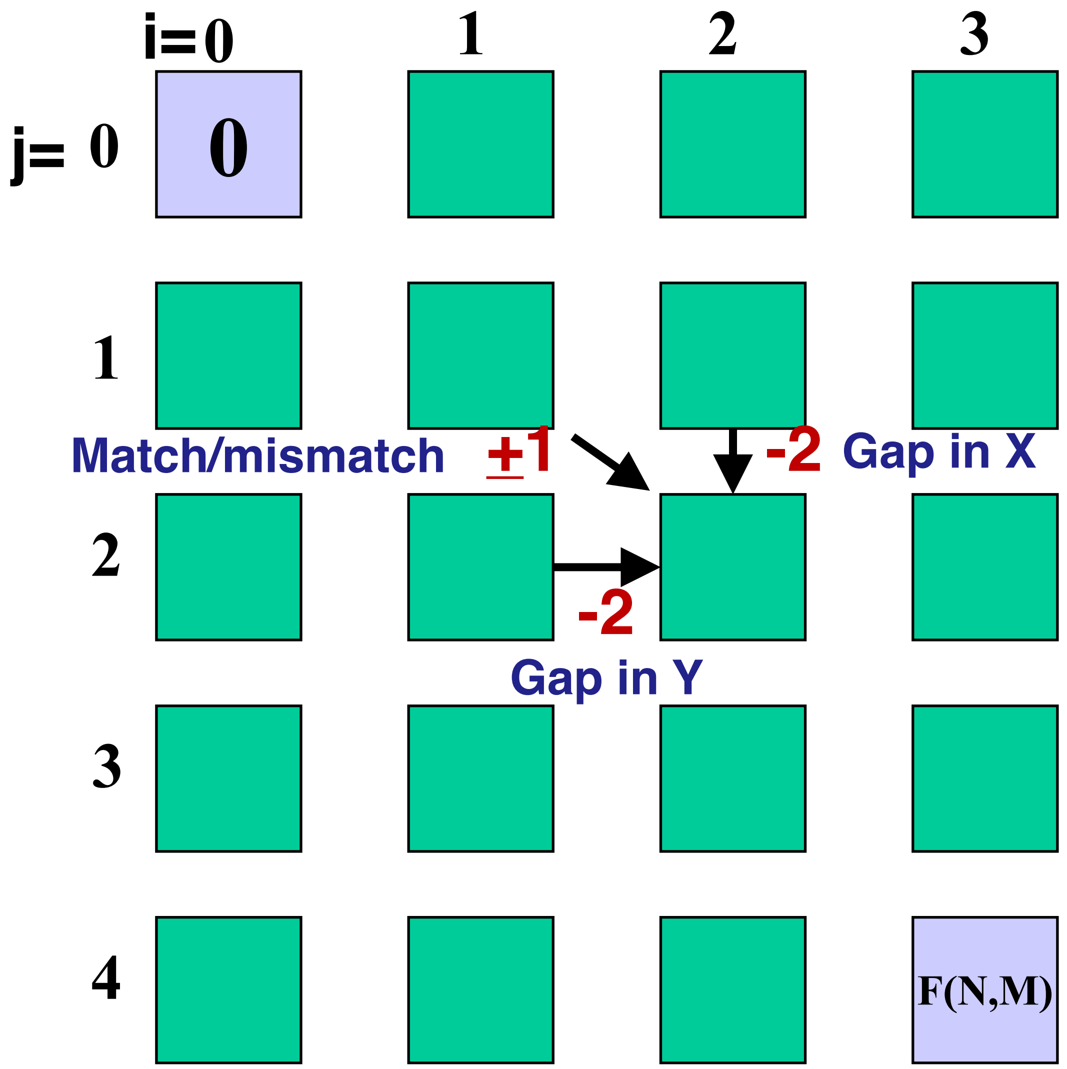
Sequence Alignment 2
- Alignment Variants
- Affine Gap Penalty
- State Transition Trellis

Lecture 4

September 17, 2019

Global Alignment



$$F(i,j) = \max \begin{cases} F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

F(i,j)

Score of <u>best</u> alignment between $X_1,\ldots,X_i$ and $Y_1,\ldots,Y_j$

Global Alignment

	i=0	1	2	3
j= 0	0	-2	-4	-6
1	-2			
2	-4			
3	-6			
4	-8			F(N,M)

$$F(i,j) = \max \begin{cases} F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

Fill in Boundary Conditions

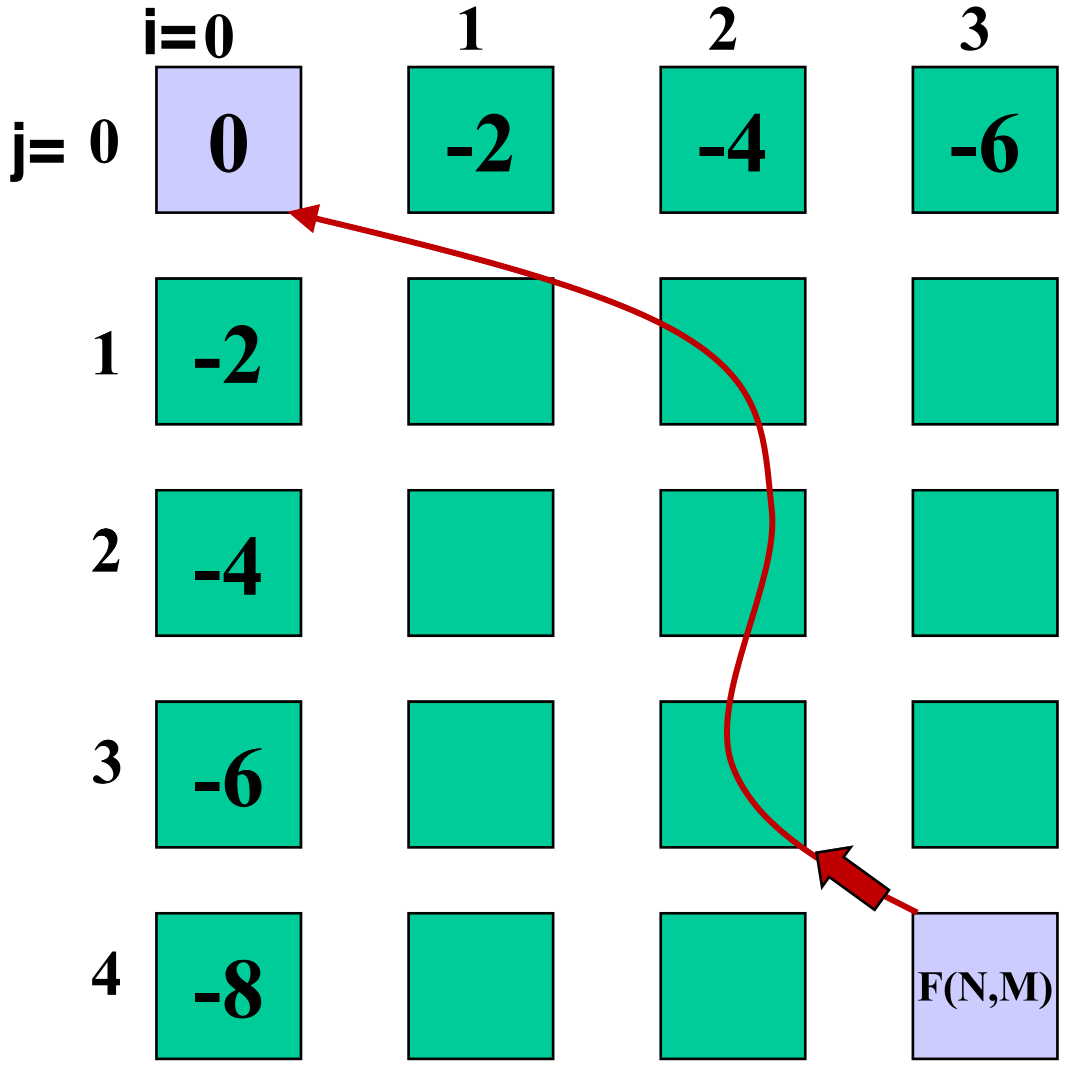
Global Alignment

	i=0	1	2	3
j= 0	0	-2	-4	-6
1	-2			
2	-4			
3	-6			
4	-8			F(N,M)

$$F(i,j) = \max \begin{cases} F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

F(N,M)
Score **of best global alignment**

Global Alignment



$$F(i,j) = \max\begin{cases} F(i-1,j-1) \pm 1 \\ F(i-1,j) - 2 \\ F(i,j-1) - 2 \end{cases}$$

F(N,M)
Score **of best global alignment**

Use *Tracebacks* to get actual alignments:

Store a pointer from F(i,j) to the cell used

Alignment Variants

- Bounded Alignments
- Semi-global Alignment
- Local Alignment

Bounded Sequence Alignment

What if X and Y are very similar?

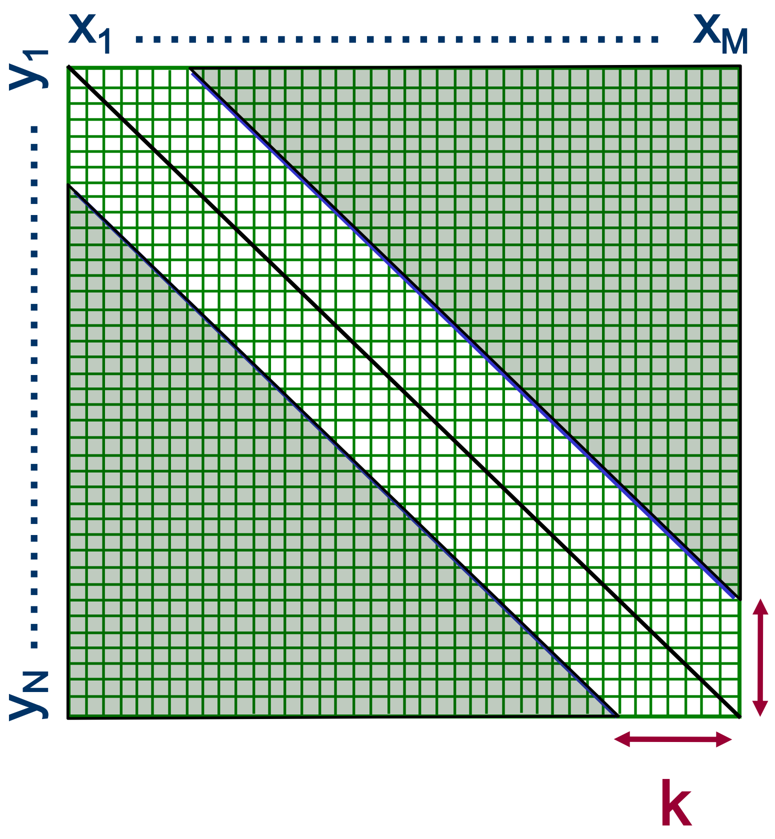
Assume that #gaps will be small (<k)

This means that if X_i matches Y_j, $|i-j|<k$

X_1	X_2	X_3	X_4	X_5	X_6
Y_1	-	-	Y_2	Y_3	Y_4

Can we use this to speed up alignment?

Bounded Dynamic Programming



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration Rule:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i - k \\ F(i-1, j) - d, \text{ if } j < i + k \end{cases}$$

Semi-Global Alignment

- Sometimes an entire sequence (X) is embedded in another sequence (Y)
- e.g. complete protein domain to a gene

We need to not penalize leading and trailing gaps

```
---CAGCGTGG---------
CAGCA-CTTGGATTCTCGG
```

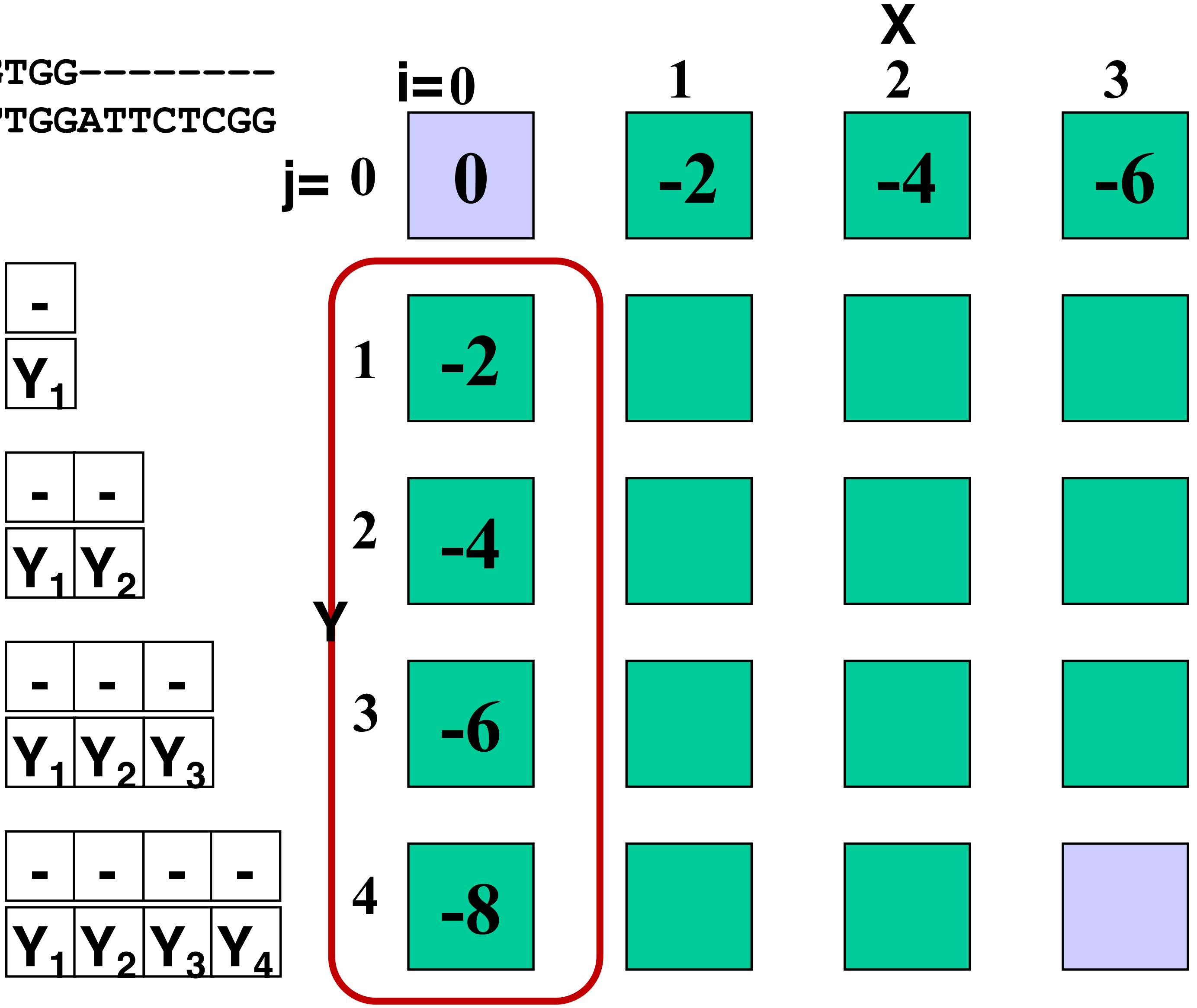
Nice

```
CAGC----G--T----GG
CAGCACTTGGATTCTCGG
```

Ugly

Semi-global Alignment

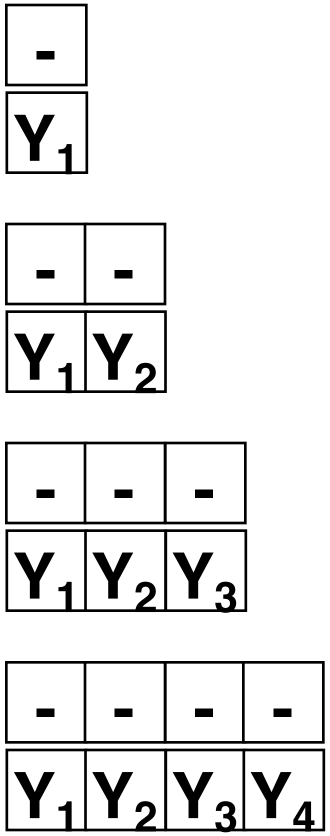
```
X ---CAGCGTGG--------
Y CAGCA-CTTGGATTCTCGG
```



Semi-global Alignment

```
X ---CAGCGTGG--------
Y CAGCA-CTTGGATTCTCGG
```

$$
\begin{array}{c|cccc}
 & i=0 & 1 & 2 & 3 \\
\hline
j=0 & 0 & -2 & -4 & -6 \\
1 & 0 & & & \\
2 & 0 & & & \\
3 & 0 & & & \\
4 & 0 & & & \\
\end{array}
$$



$\begin{array}{c} - \\ Y_1 \end{array}$
$\begin{array}{cc} - & - \\ Y_1 & Y_2 \end{array}$
$\begin{array}{ccc} - & - & - \\ Y_1 & Y_2 & Y_3 \end{array}$
$\begin{array}{cccc} - & - & - & - \\ Y_1 & Y_2 & Y_3 & Y_4 \end{array}$

First Row all zeros

Semi-global Alignment

```
X ---CAGCGTGG---------
Y CAGCA-CTTGGATTCTCGG
```

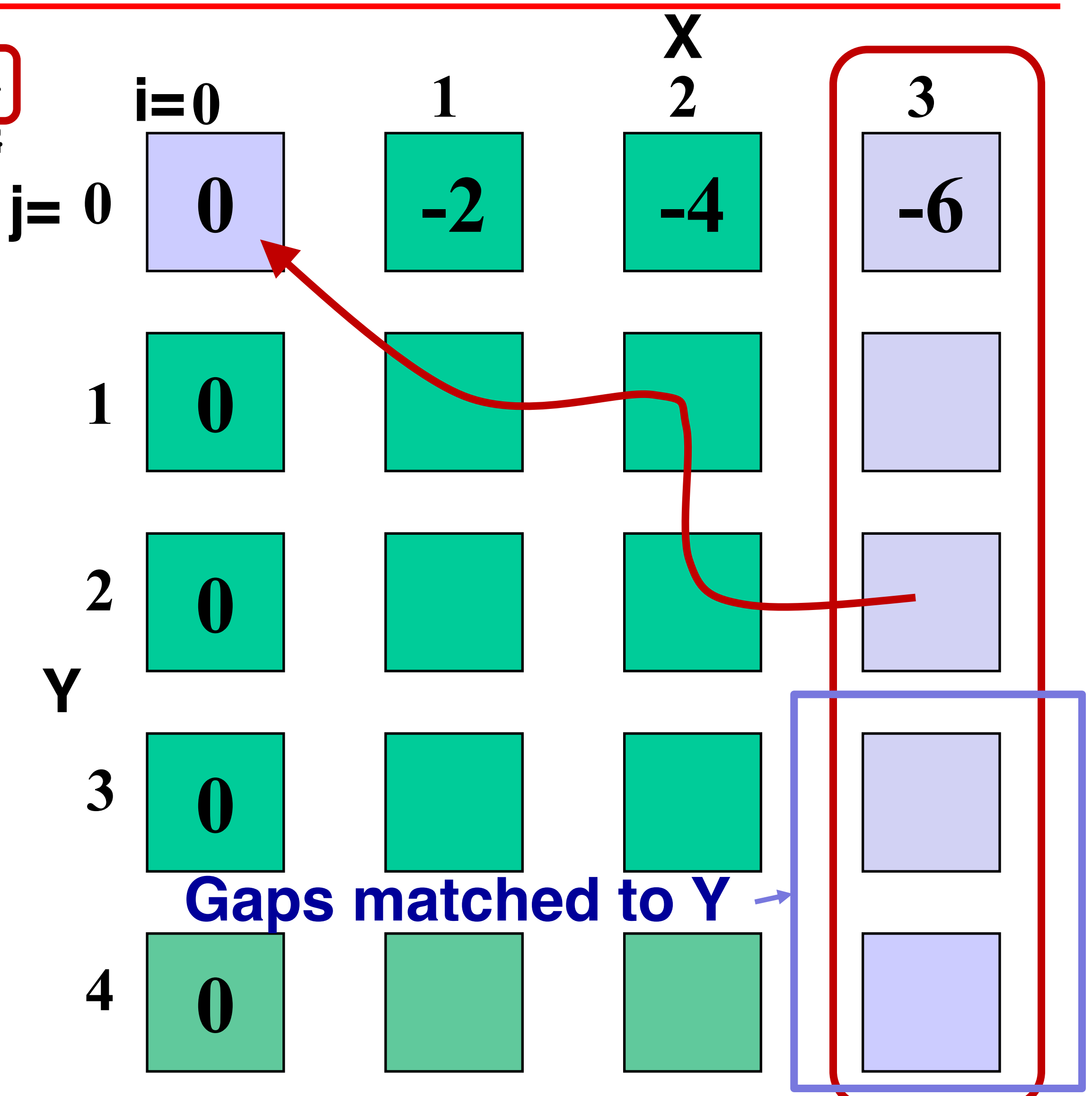
X

Y	i=0	1	2	3
j= 0	0	-2	-4	-6
1	0			
2	0			
3	0			
4	0			

Semi-global Alignment

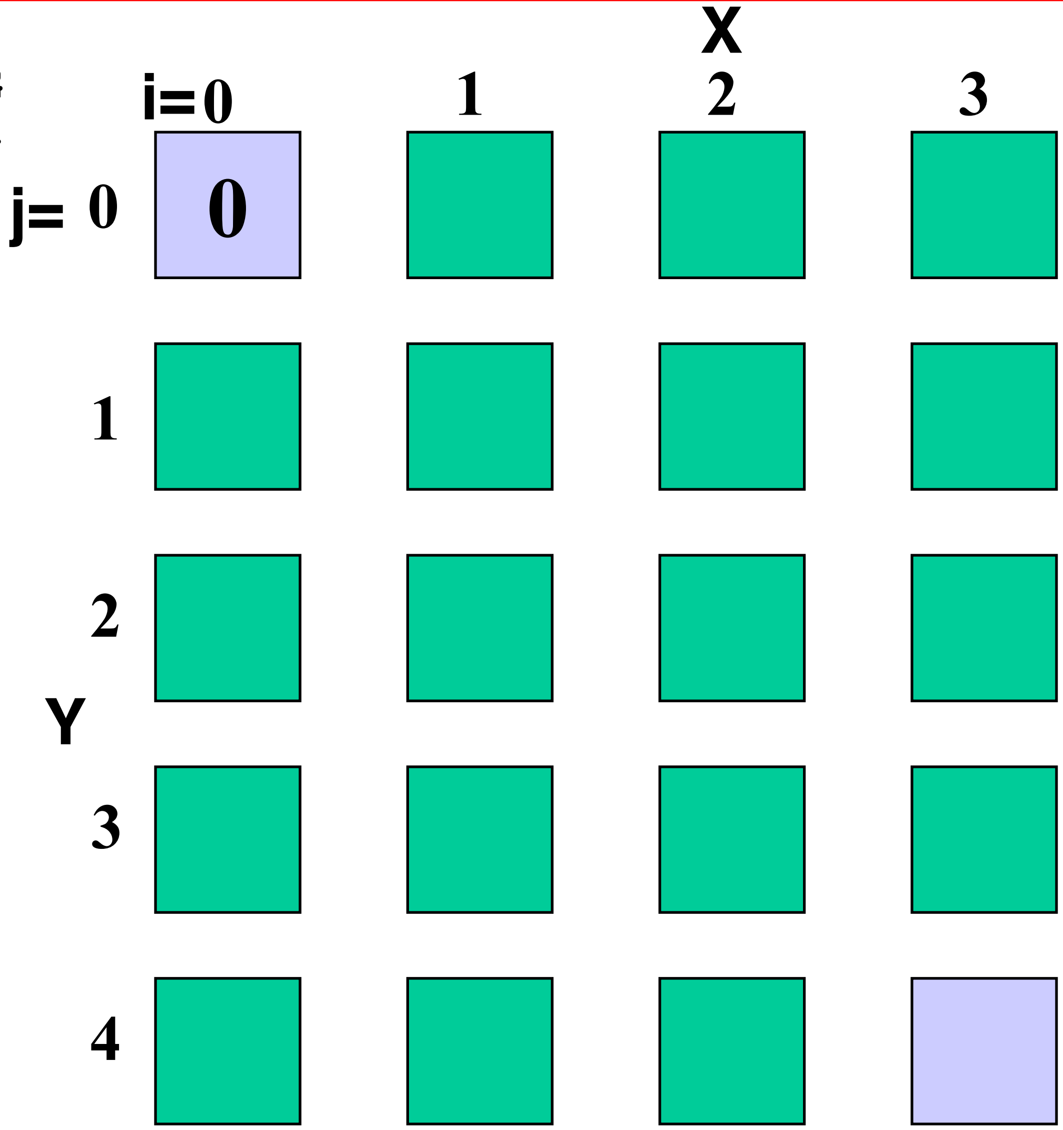
```
X ---CAGCGTGG--------
Y CAGCA-CTTGGATTCTCGG
```

Look for <u>maximum anywhere in last column</u>



Semi-global Alignment

```
X  CAGCA-CTTGGATTCTCGG
Y  ---CAGCGTGG--------
```



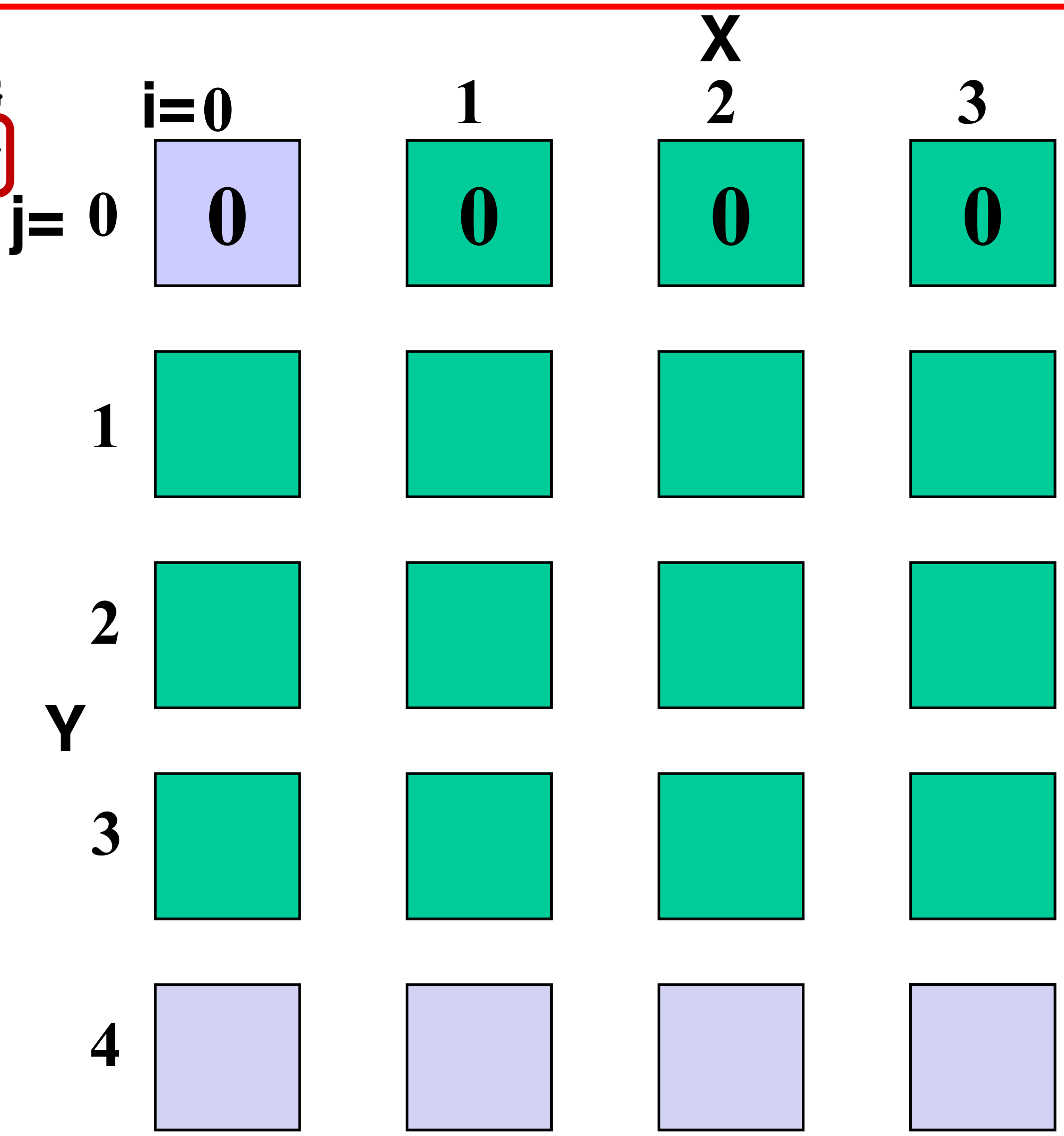
Semi-global Alignment

```
X  CAGCA-CTTGGATTCTCGG
Y  ---CAGCGTGG--------
```

		X		
	i=0	1	2	3
j= 0	0	0	0	0
1				
2				
Y 3				
4				

Semi-global Alignment

```
X CAGCA-CTTGGATTCTCGG
Y ---CAGCGTGG--------
```



Alignment Variations

- Summary of end space scoring procedures:

Place where spaces are not penalized for	**Action**
Before 1st sequence	**Initialize 1st column with zeros**
After 1st sequence	**Look for max in last column**
Before 2nd sequence	**Initialize 1st row with zeros**
After 2nd sequence	**Look for max in last row**

Semi-global Alignment

Allow gaps at either end of each sequence

But this is still a <u>global</u> alignment

X

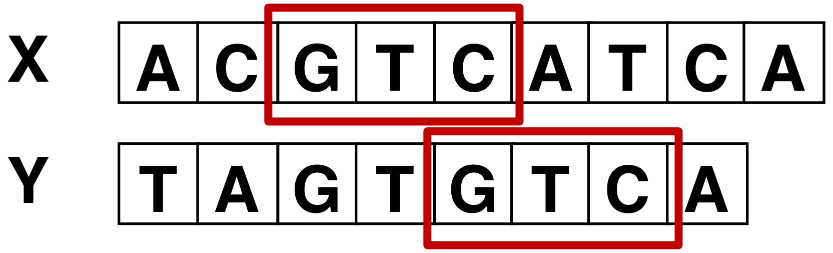
	i=0	1	2	3
j= 0	0	0	0	0
1	0			
2	0			
3	0			
4	0			

Y

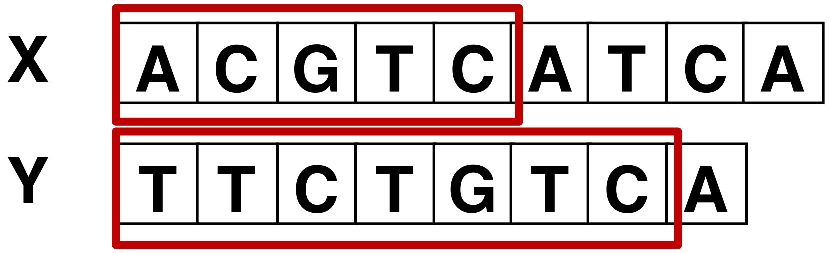
Local Alignment

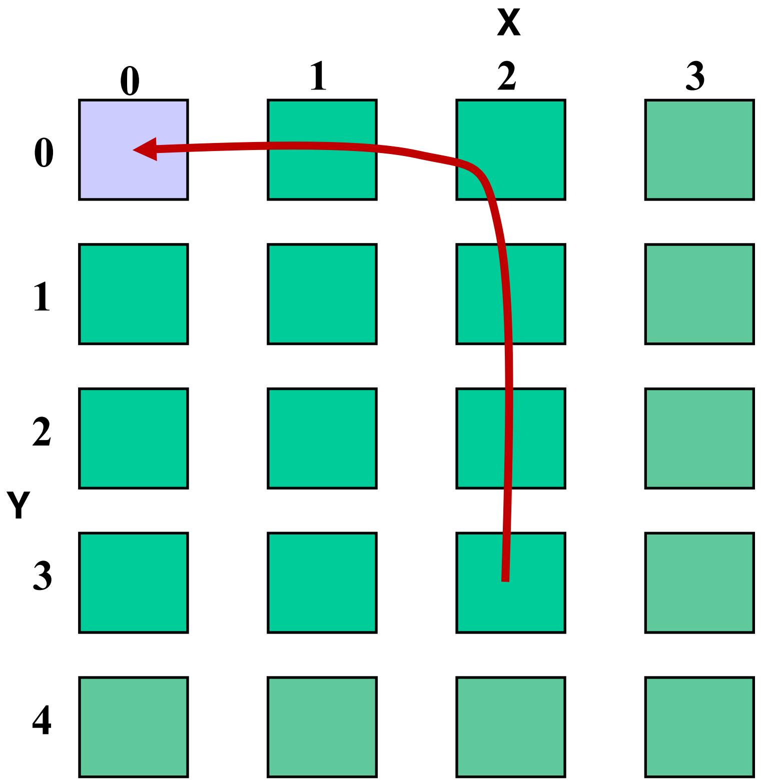
Smith-Waterman Algorithm

- Sometimes we just want an alignment between part of X and part of Y
- e.g. conserved protein domain between two complete genes
- This means finding the highest scoring alignment of <u>any</u> subsequences of X & Y



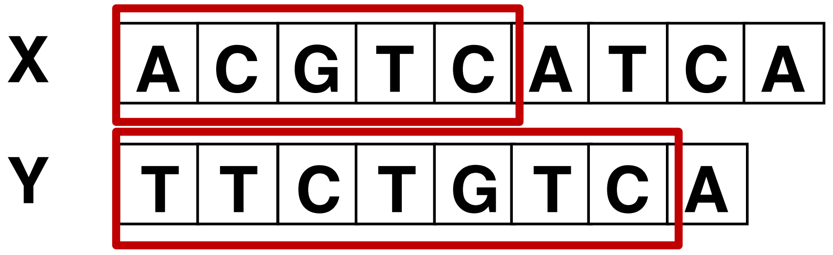
A Slightly Easier Problem

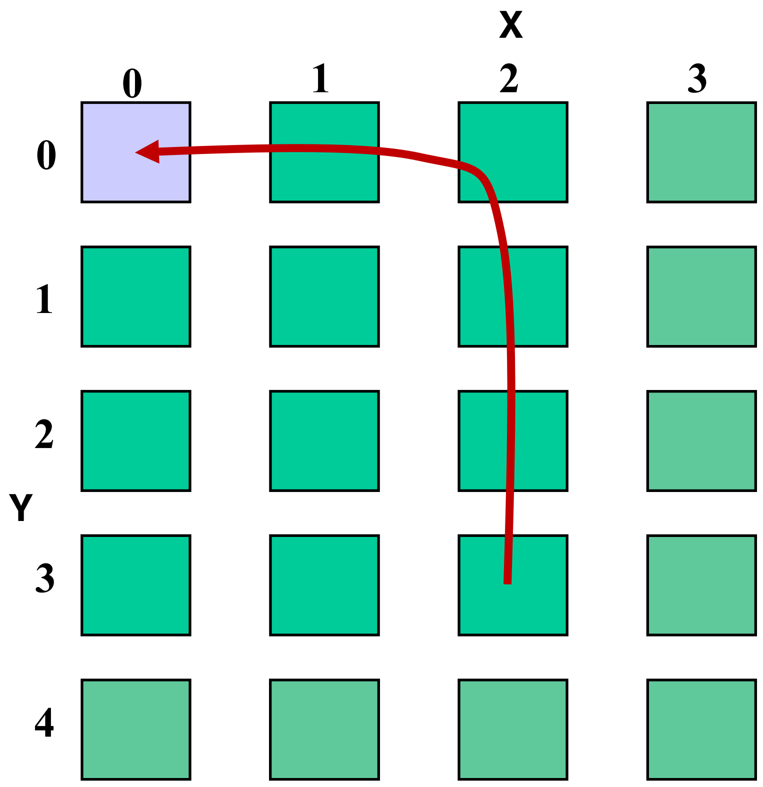




What if we want the highest score alignment of any two substrings that start at X_0,Y_0

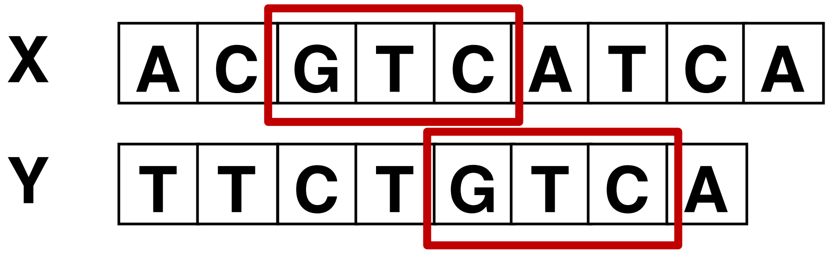
A Slightly Easier Problem

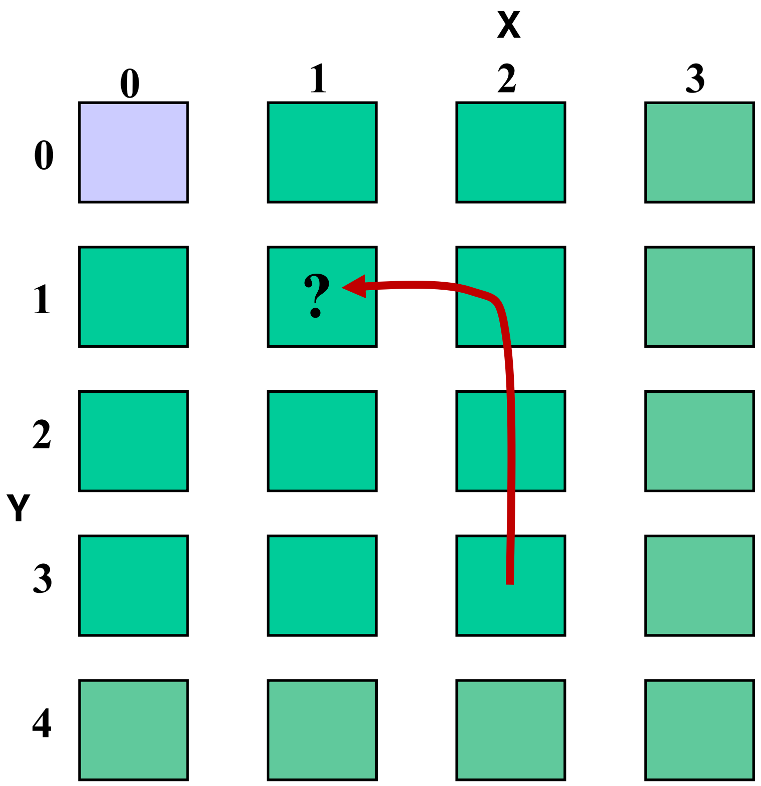




A	-	C	-	G	T	C
T	T	C	T	G	T	C
-1	-2	1	-2	1	1	1

Smith-Waterman Algorithm



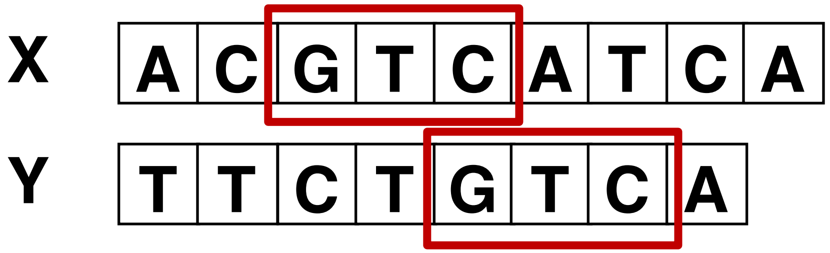


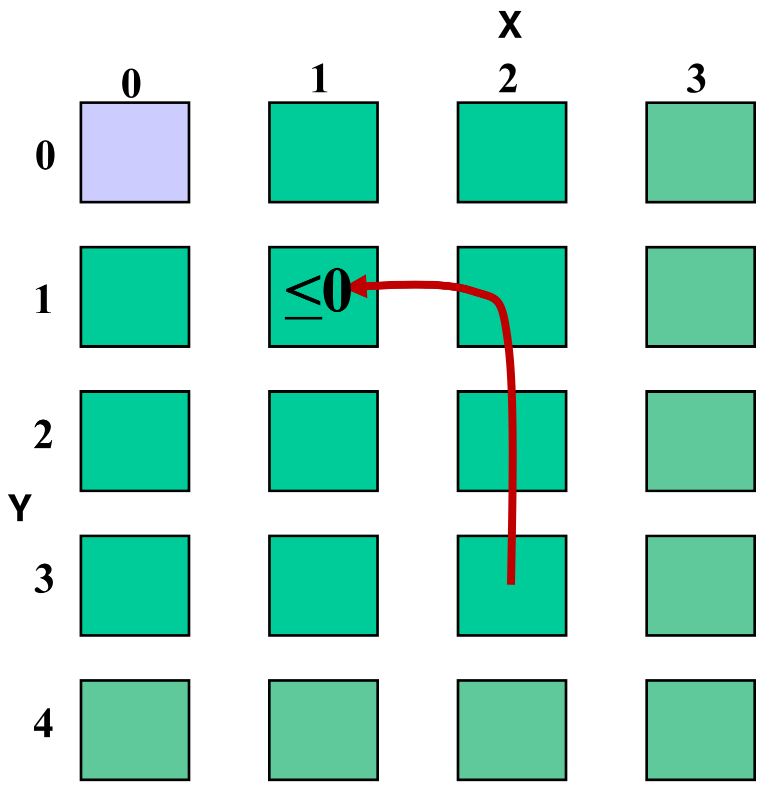
A - C - G T C

T T C T G T C

-1 -2 1 -2 1 1 1

Smith-Waterman Algorithm

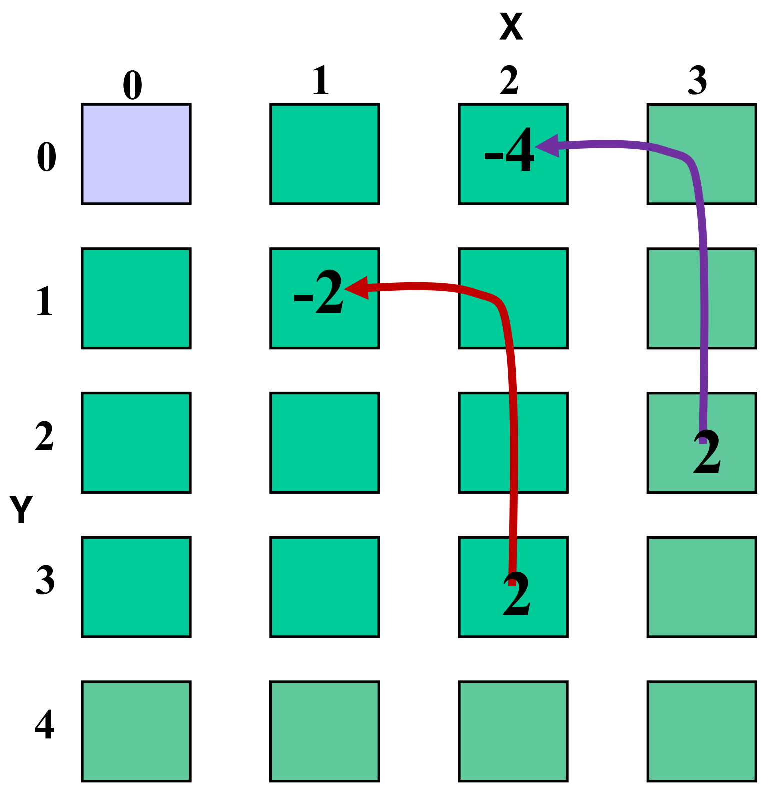




Stop the alignment when what came before had more bad than good

Smith-Waterman Algorithm

But imagine this hypothetical scenario
Is the blue alignment really just as good as the red?

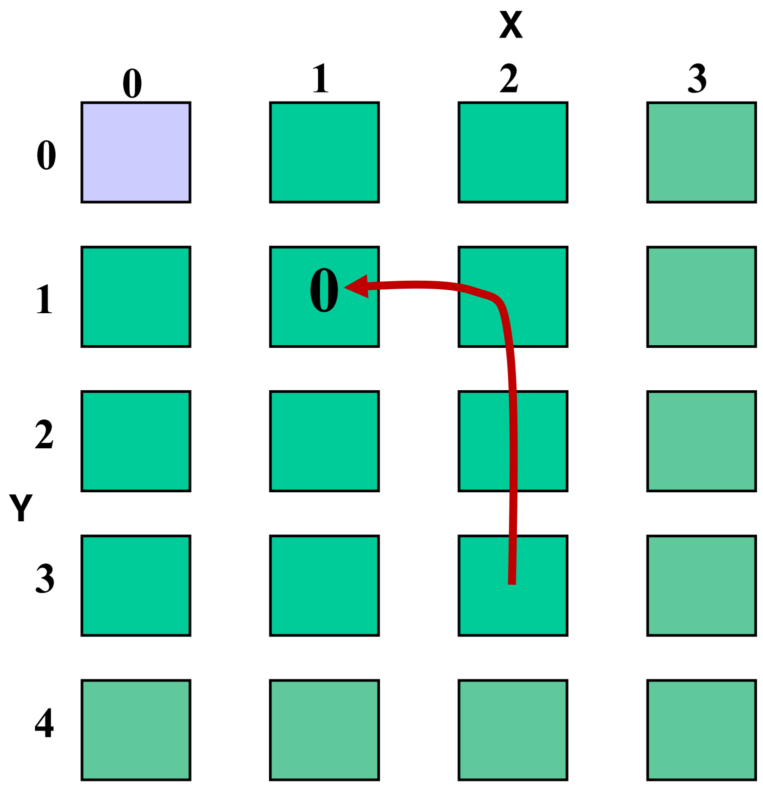


No, because it started at a lower negative to reach 2!

Smith-Waterman Algorithm

$$F(i,j) = \max \begin{cases} \boxed{0} \\ F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!

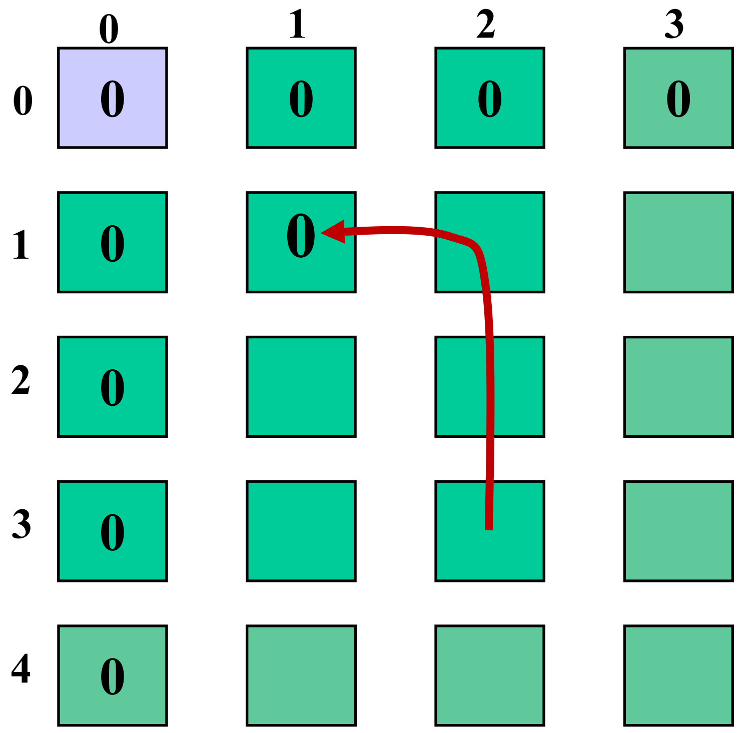


Start alignment at highest F(i,j) anywhere in matrix – traceback to first zero

Smith-Waterman Algorithm

$$F(i,j) = \max \begin{cases} \boxed{0} \\ F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!



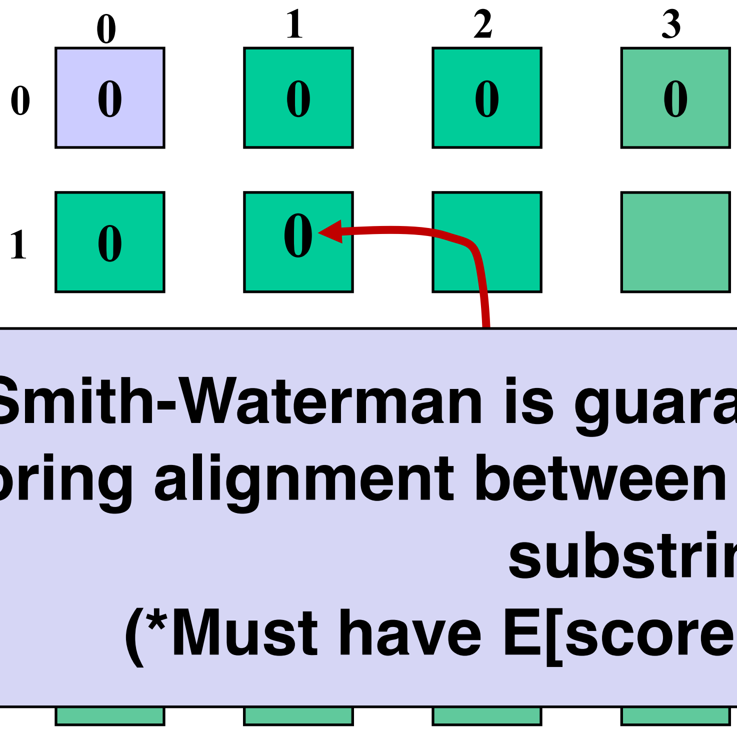
Initialize first row and column to zeros

Start alignment at highest F(i,j) anywhere in matrix – traceback to first zero

Smith-Waterman Algorithm

$$F(i,j) = \max \begin{cases} \boxed{0} \\ F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!



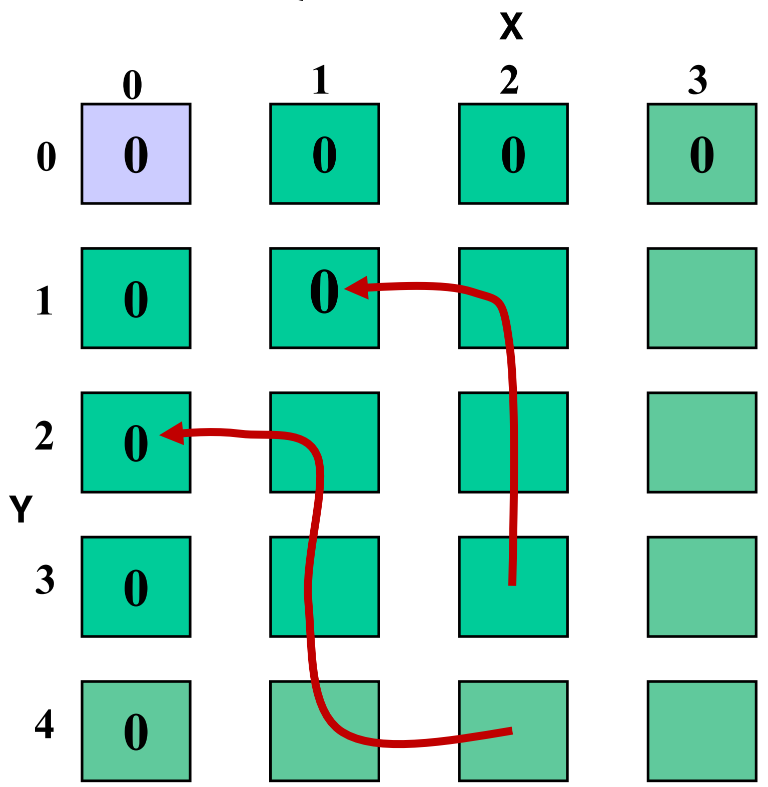
Initialize first row and column to zeros

Smith-Waterman is guaranteed to find the highest scoring alignment between *any* substring of X and *any* substring of Y*
(*Must have E[score of random seqs]<0)

Smith-Waterman Algorithm

$$F(i,j) = \max\begin{cases} \boxed{0} \\ F(i-1,j-1) \pm 1 \\ F(i-1,j) - 2 \\ F(i,j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!

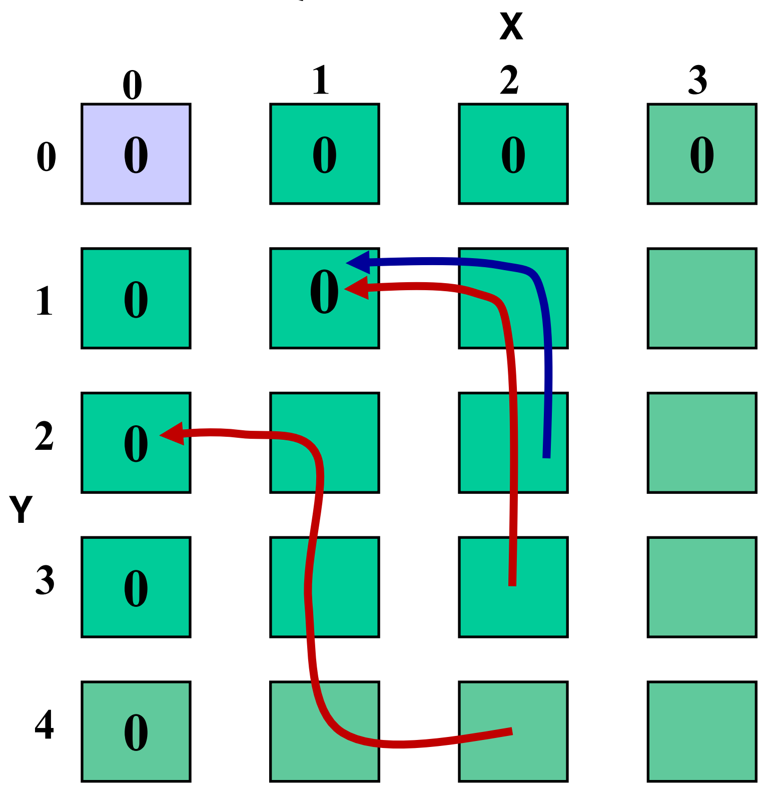


What about the next highest scoring subsequence alignment?

Smith-Waterman Algorithm

$$F(i,j) = \max \begin{cases} \boxed{0} \\ F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!

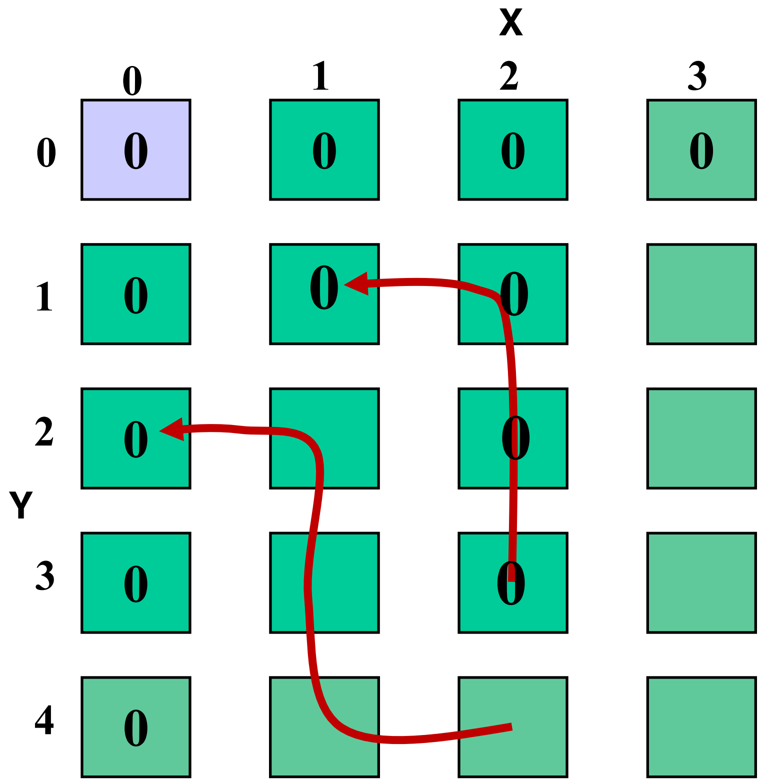


But how do we avoid this trivial solution?

Smith-Waterman Algorithm

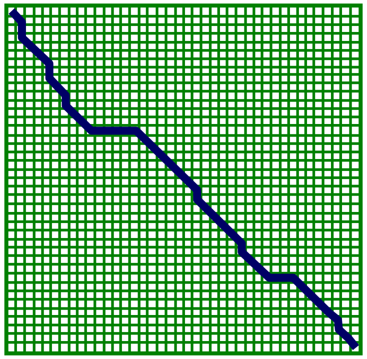
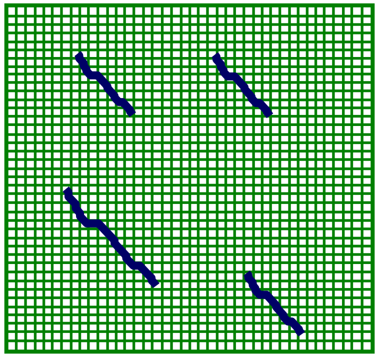
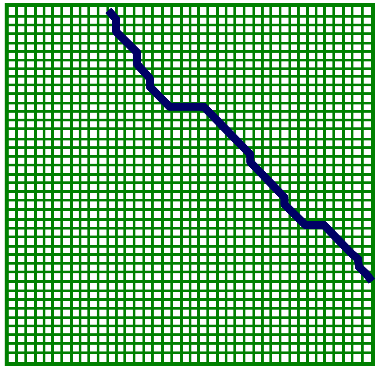
$$F(i,j) = \max \begin{cases} \boxed{0} \\ F(i-1, j-1) \pm 1 \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases}$$

During recursion, rather than going < 0, we reset F to zero!



But how do we avoid this trivial solution?

Zero out found alignments

Global	Local	Semi-global
		
Complete alignment	Stretches of similarity	No end-gap penalty

Initialization to zero	Top left	Top row/left col.	Top row or left column
Iteration:max	$F(i-1, j) - d$ * $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$	0 $F(i-1, j) - d$ $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$	$F(i-1, j) - d$ $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$
Termination	Bottom right	Anywhere	Bottom row or right column

Exercise for You

- Often gaps come in bunches
- The probability of getting 10 gaps in a row is higher than 10 different 1 gap events
- Solution: Affine Gap Penalty
- g(n) is penalty for gap of length n:

$$g(n)= \underset{\nearrow}{(-d)} - (n-1)\, \underset{\nearrow}{e}$$

Gap-open penalty (d) **Gap-extension penalty (e)**

What is the new update rule for F(i,j)?

Key Idea

To compute optimal alignment, at position (i, j), need to "remember" both best score if gap is open AND if gap is not open

E.g. Two scenarios for X_i aligned to gap

x_{i-1} aligns to y_{j-1}

A	**A**	**X_{i-1}**	**X_i**
A	**A**	**Y_{i-1}**	**-**

Score with d
(gap open)

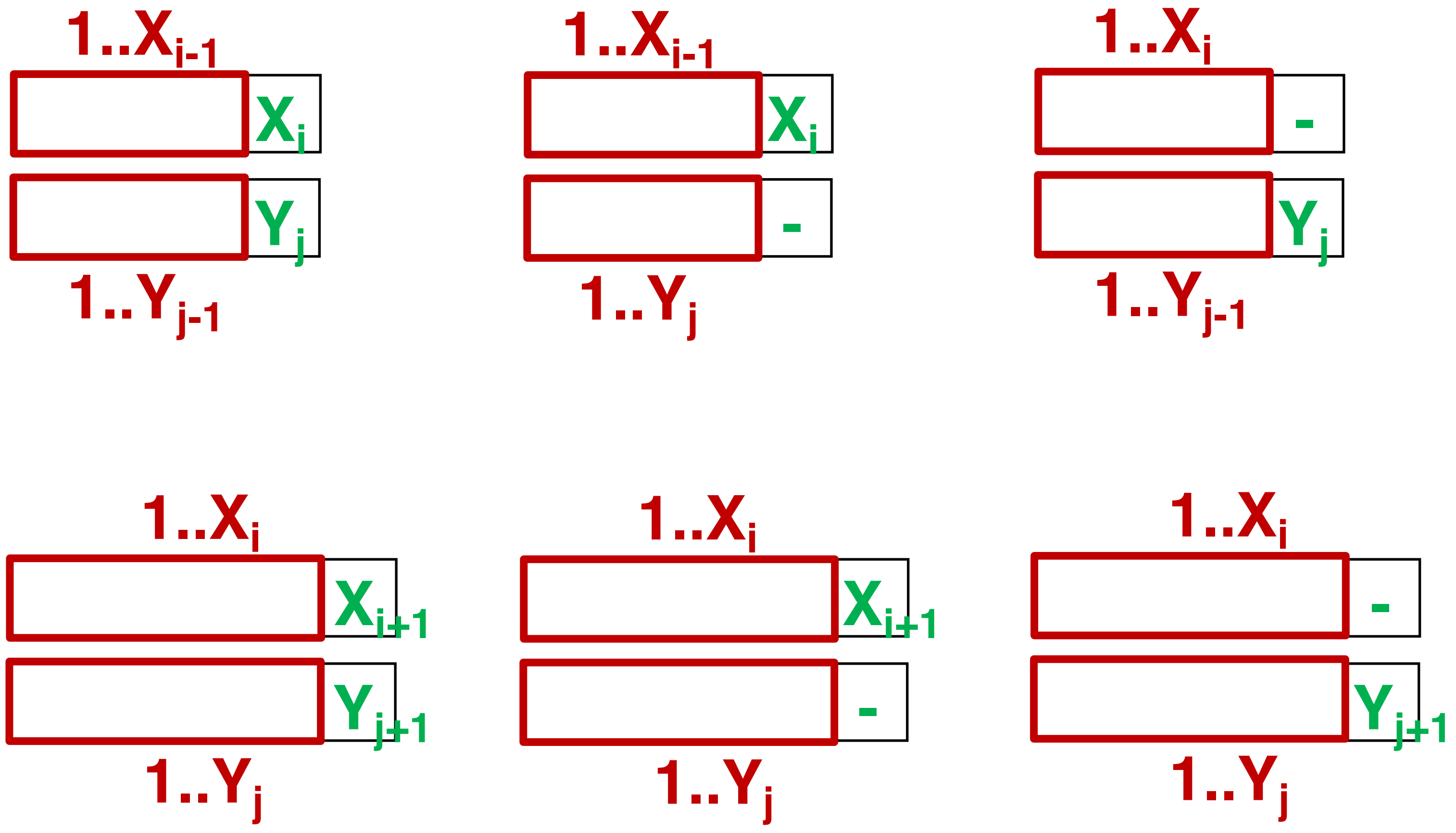
x_{i-1} aligns to gap

A	**A**	**X_{i-1}**	**X_i**
A	**Y_{i-1}**	**-**	**-**

Score with e
(gap extension)

Getting to (i+1,j+1) from (i,j)

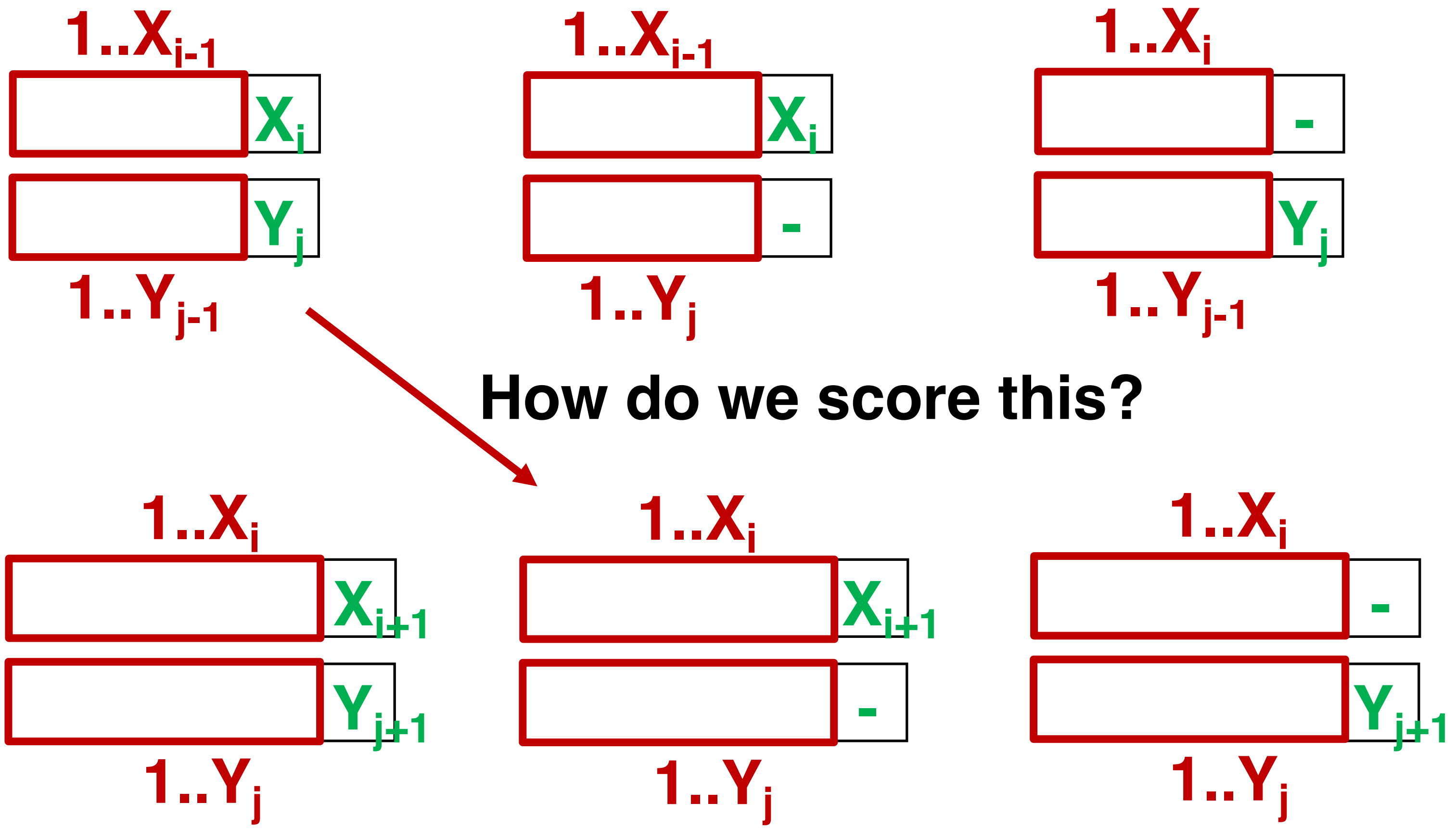
Still three ways that an alignment of (i,j) or (i+1,j+1) can end



But now it matters which way (i,j) ended to score (i+1,j+1)

Getting to (i+1,j+1) from (i,j)

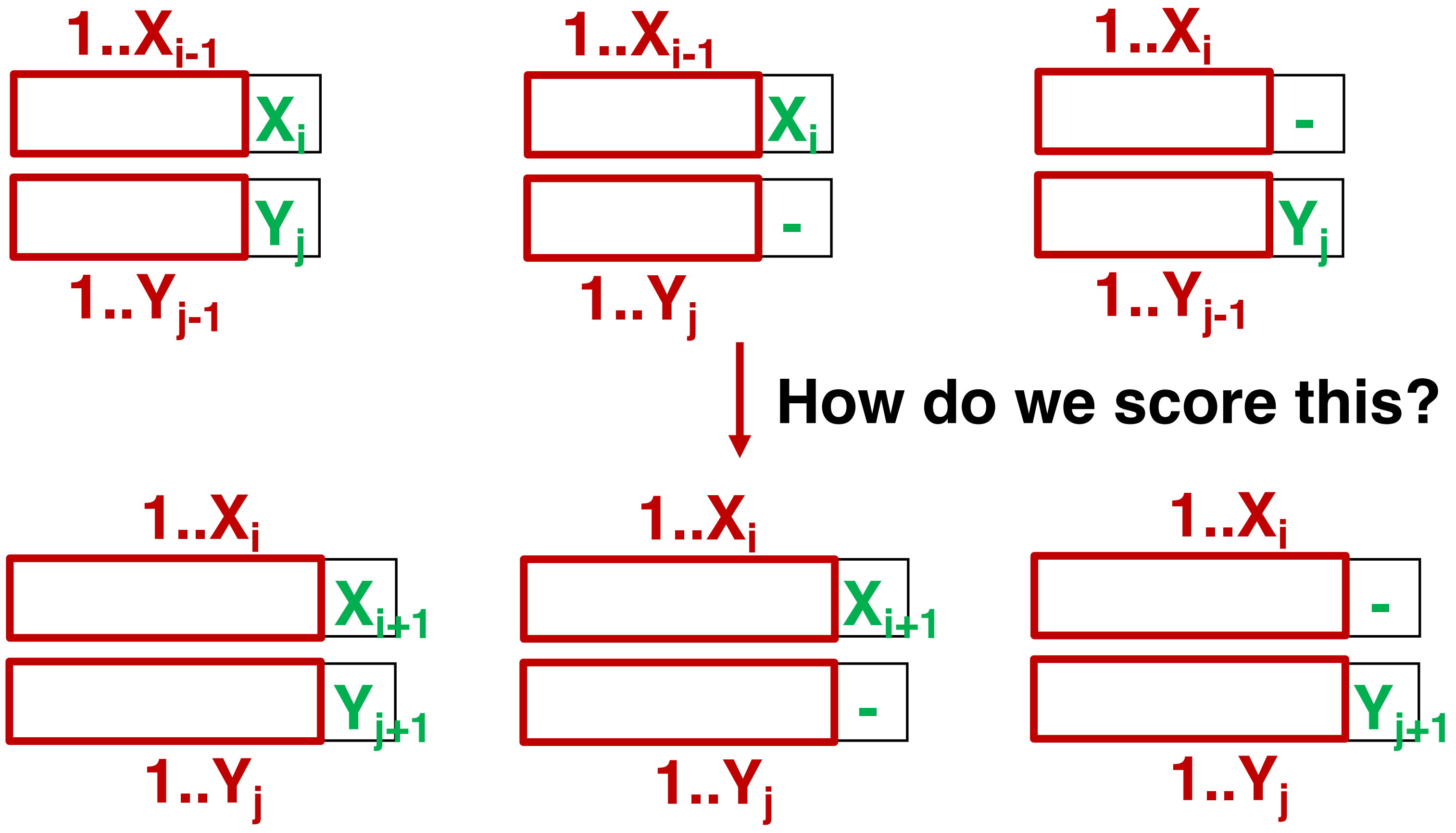
Still three ways that an alignment of (i,j) or (i+1,j+1) can end



But now it matters which way (i,j) ended to score (i+1,j+1)

Getting to (i+1,j+1) from (i,j)

Still three ways that an alignment of (i,j) or (i+1,j+1) can end



But now it matters which way (i,j) ended to score (i+1,j+1)

Update Rule

Define 3 Update Quantities

$M(i,j)$ – best score up to (i,j) with x_i aligned to y_i

$I_x(i,j)$ – best score up to (i,j) with x_i aligned to gap

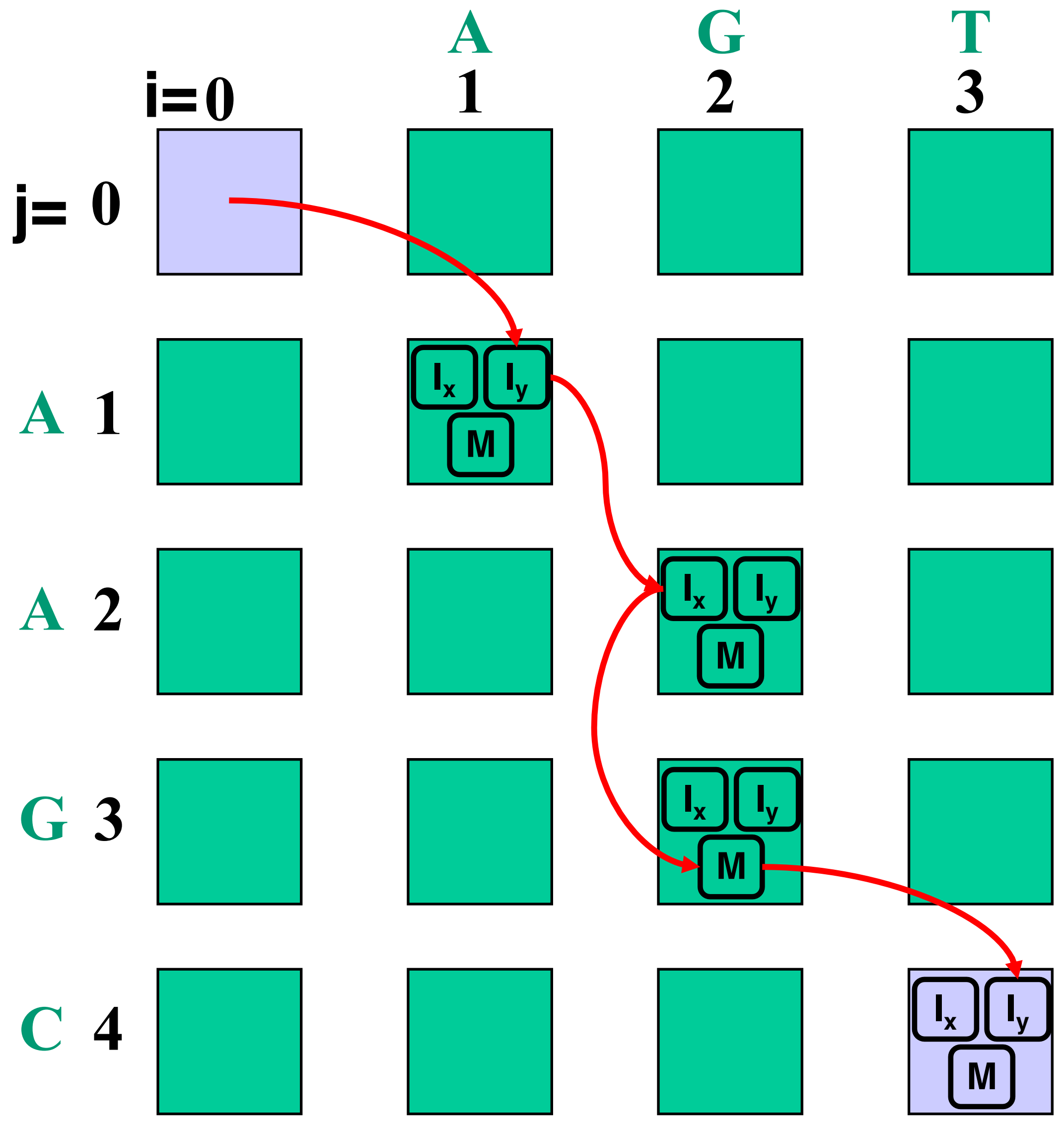
$I_y(i,j)$ – best score up to (i,j) with y_j aligned to gap

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j) \\ I_x(i-1,j-1) + s(x_i, y_j) \\ I_y(i-1,j-1) + s(x_i, y_j) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

Expanding the F Matrix

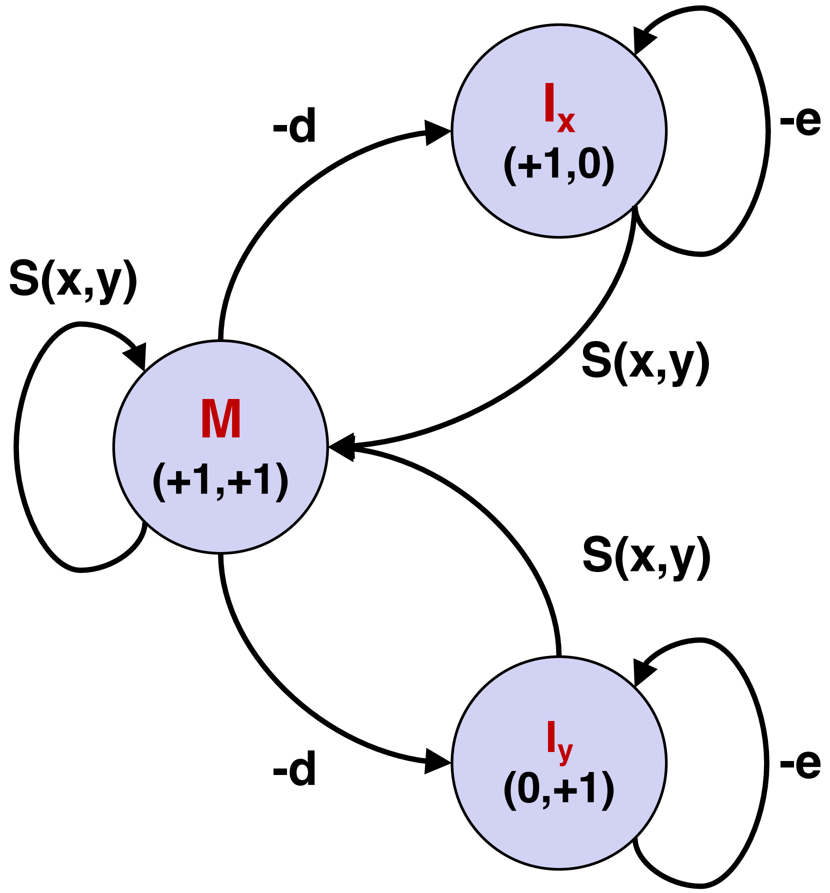


Instead of storing a single value F(i,j) in each square

We need to store three values

These can be considered 3 *states* at each (i,j)

Finite State Machine

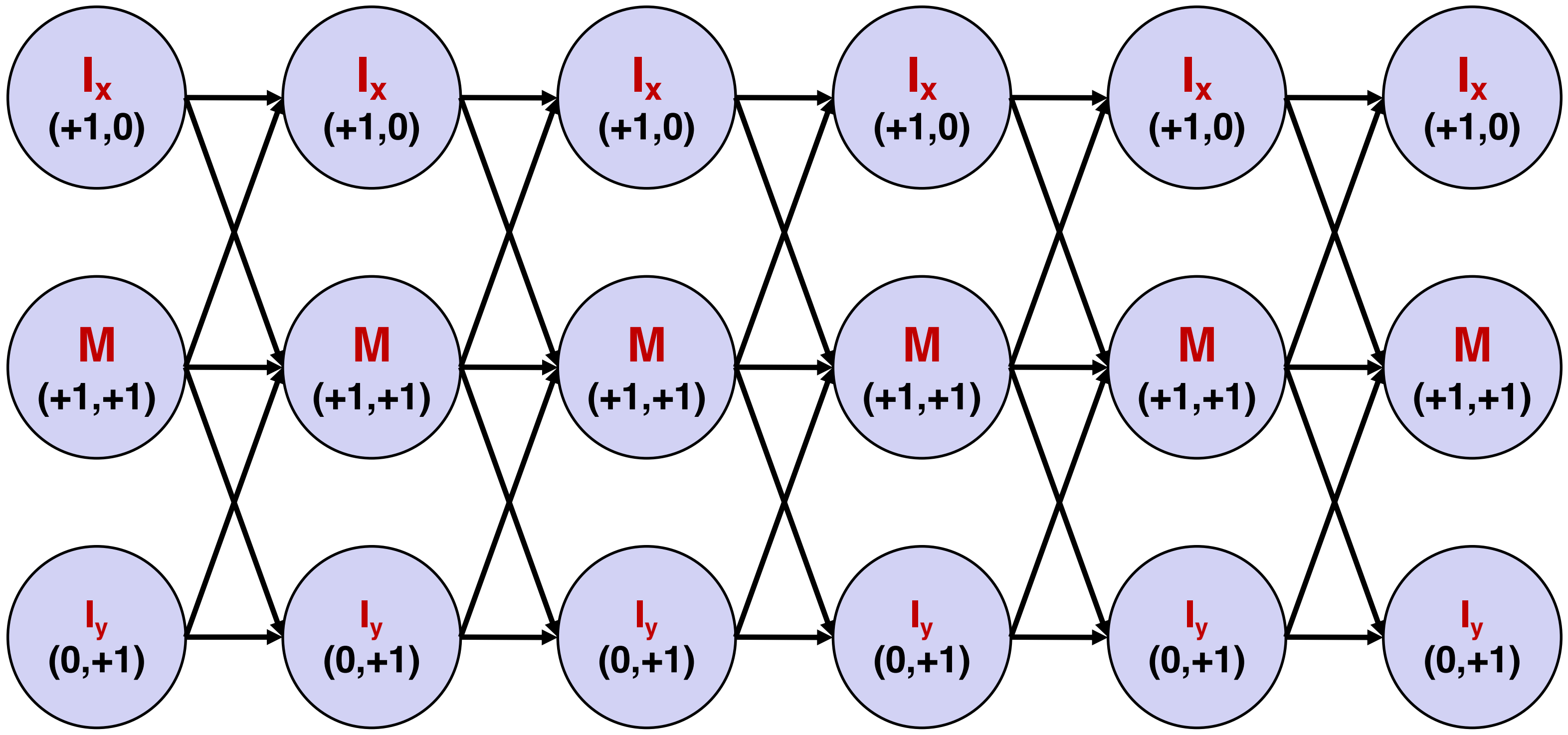


Any given alignment is a series of state transitions through this state machine

X	A	T	G	C	-	T
Y	A	-	-	C	A	T

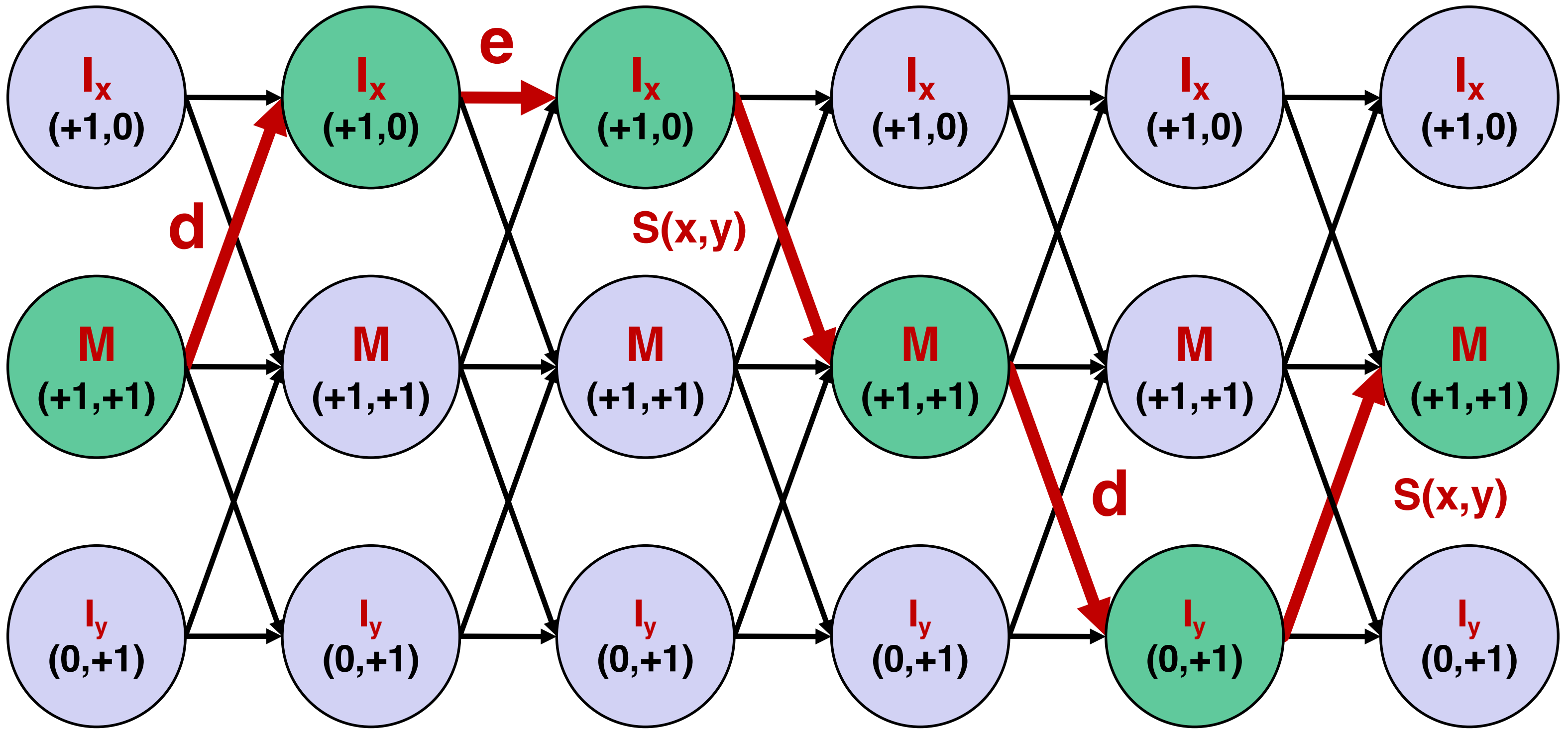
And we can score an alignment by scoring transitions

State Transition Trellis



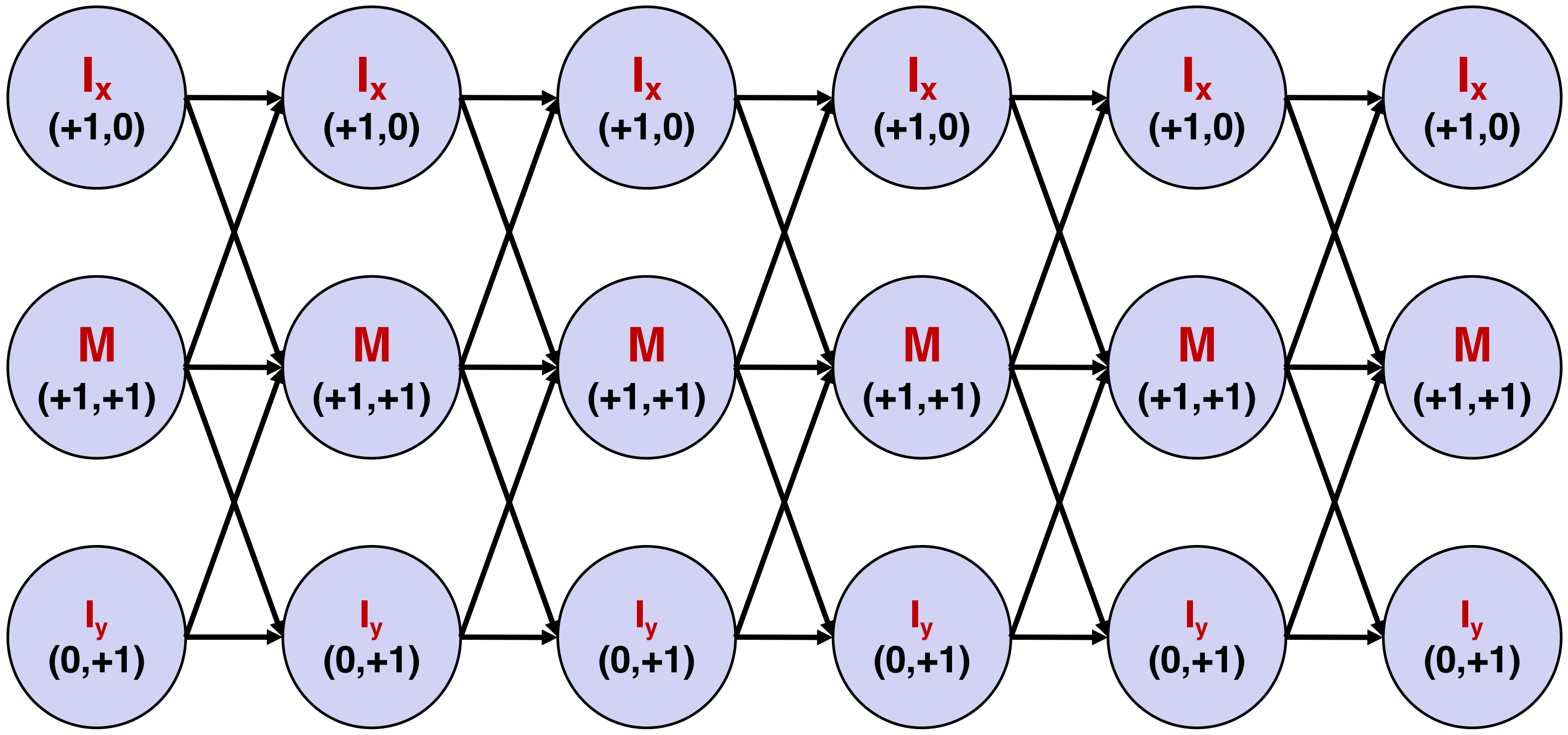
State Transition Trellis

X	A	T	G	C	-	T
Y	A	-	-	C	A	T



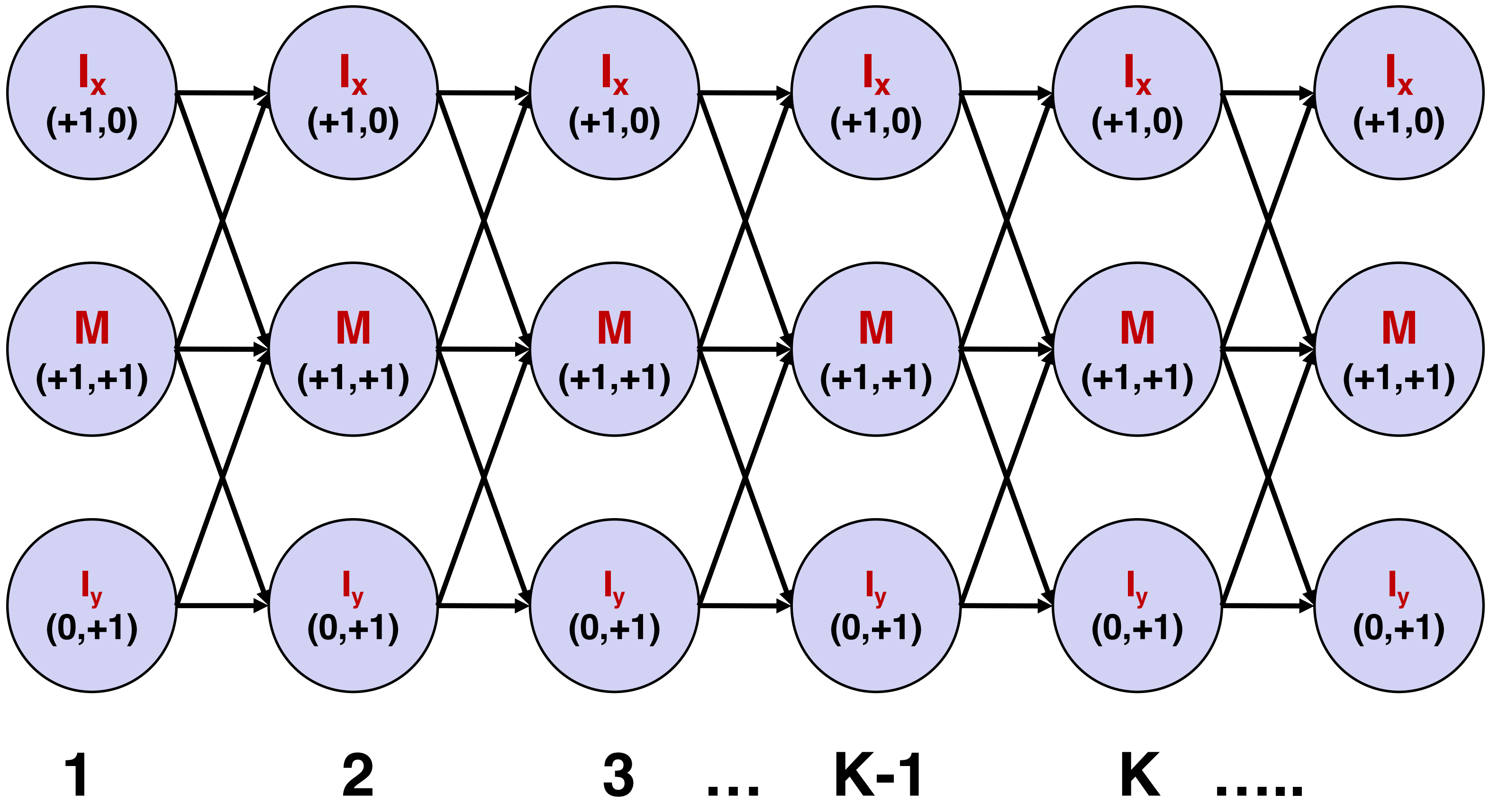
State Transition Trellis

But how do we find the best alignment?



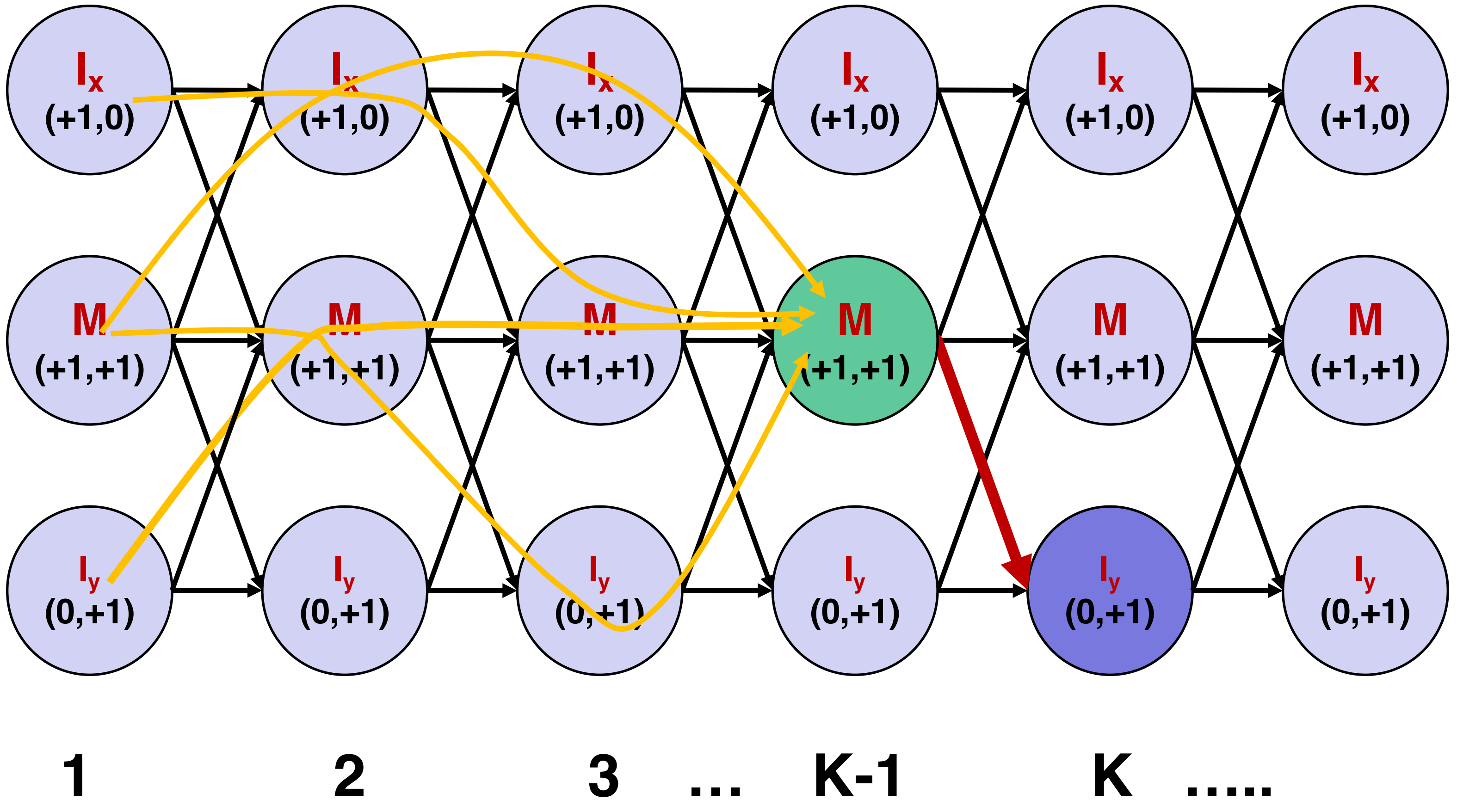
State Transition Trellis

Define $\begin{matrix} I_{x,k} \\ I_{y,k} \\ M_k \end{matrix}$ as score of best alignment to *position* K ending in state $\begin{matrix} I_x \\ I_y \\ M \end{matrix}$



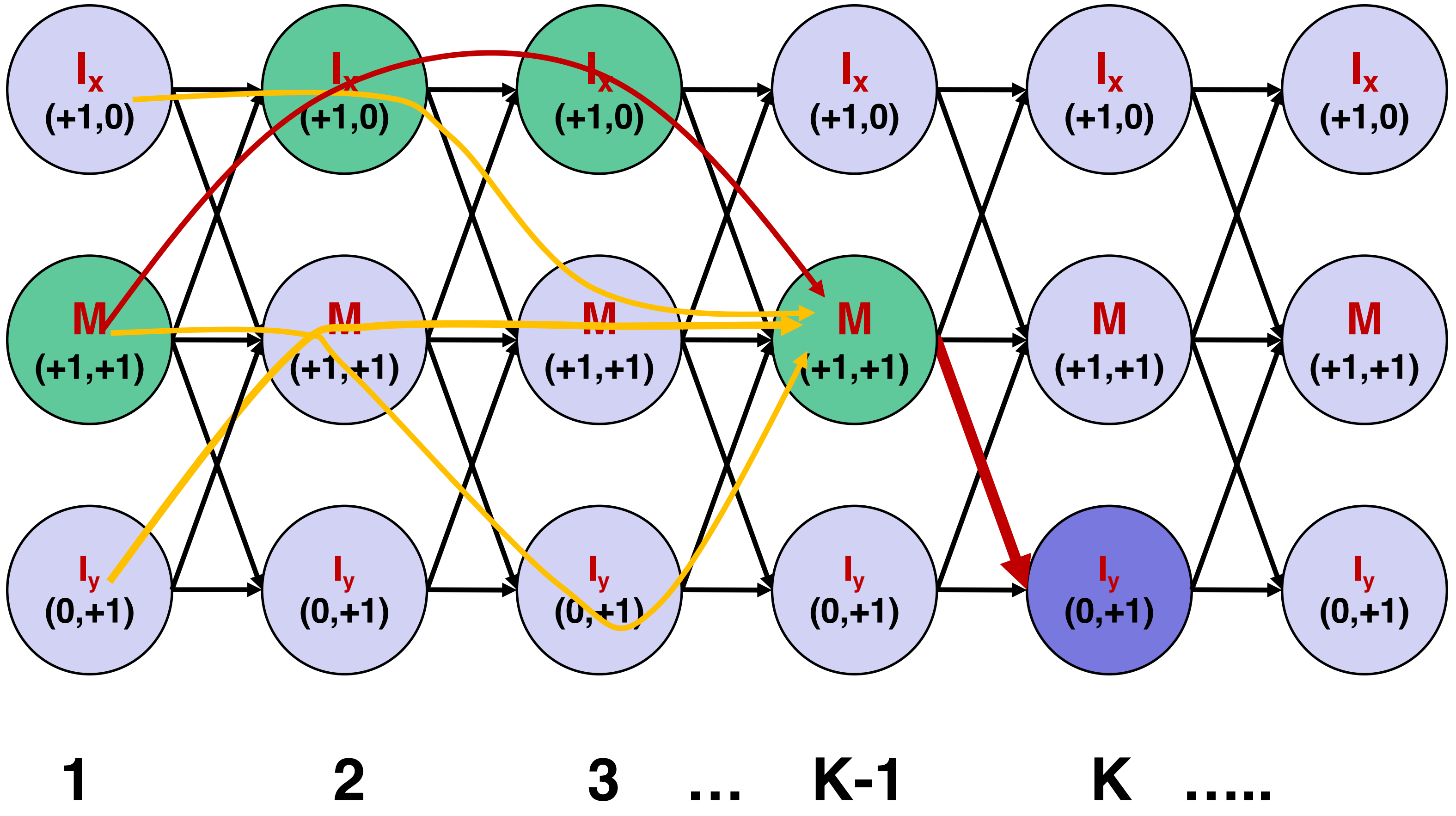
Optimality Substructure Revisted

Define $\begin{matrix} I_{x,k} \\ I_{y,k} \\ M_k \end{matrix}$ as score of best alignment to *position* K ending in state $\begin{matrix} I_x \\ I_y \\ M \end{matrix}$



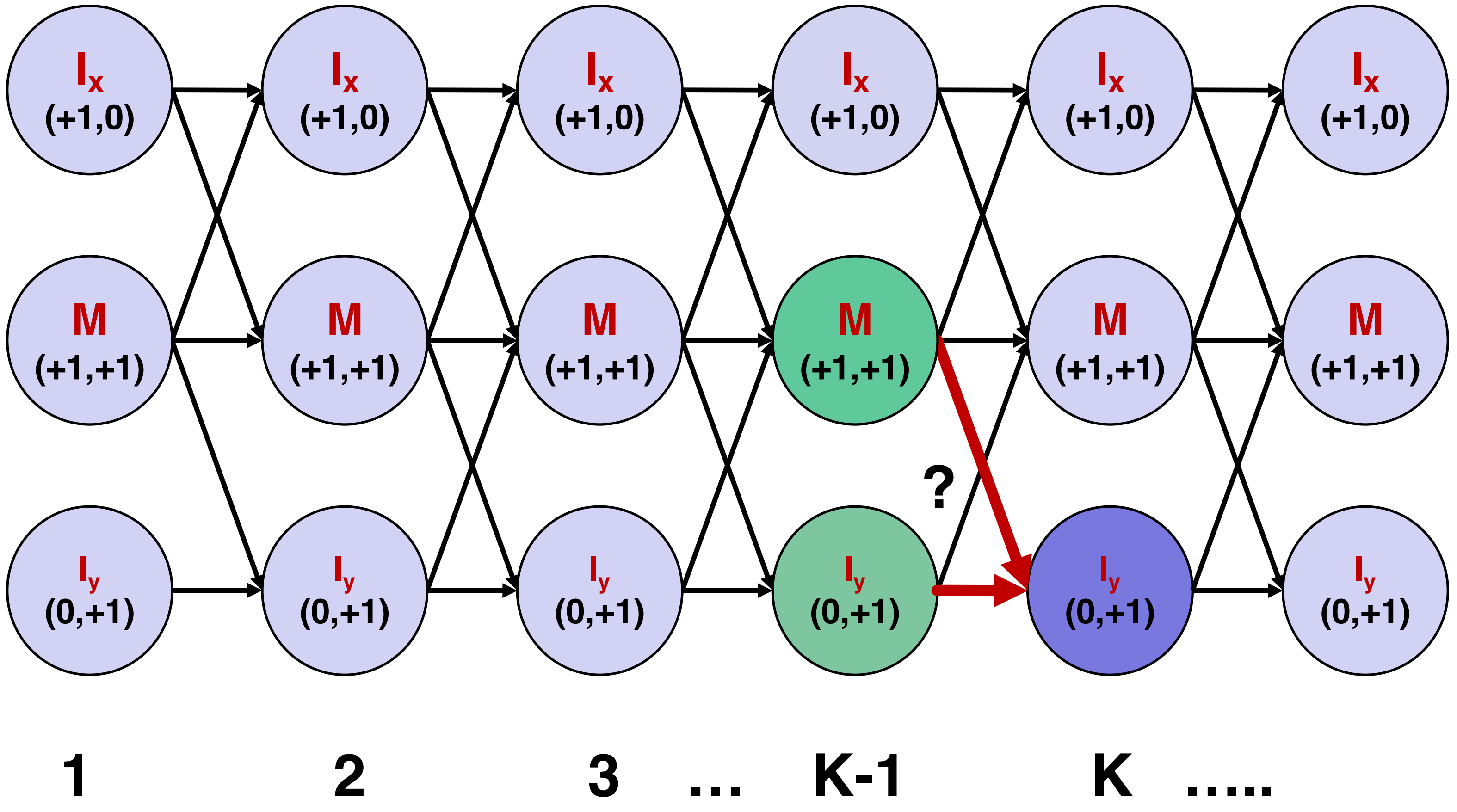
Optimality Substructure Revisted

$$\text{Define}\ \begin{matrix} I_{x,k} \\ I_{y,k} \\ M_k \end{matrix}\ \text{as score of best alignment to } \textit{position}\ K\ \text{ending in state}\ \begin{matrix} I_x \\ I_y \\ M \end{matrix}$$



Optimality Substructure Revisted

Define $\begin{matrix} I_{x,k} \\ I_{y,k} \\ M_k \end{matrix}$ as score of best alignment to *position* K ending in state $\begin{matrix} I_x \\ I_y \\ M \end{matrix}$



Finite State Machine

Trellis

$$M_k = \max\begin{cases} M_{k-1} + s(x,y) \\ I_{x,k-1} + s(x,y) \\ I_{y,k-1} + s(x,y) \end{cases}$$

$$I_{x,k} = \max\begin{cases} M_{k-1} - d \\ I_{x,k-1} - e \end{cases}$$

$$I_{y,k} = \max\begin{cases} M_{k-1} - d \\ I_{y,k-1} - e \end{cases}$$

F Matrix

$$M(i,j) = \max\begin{cases} M(i-1,j-1) + s(x_i,y_j) \\ I_x(i-1,j) + s(x_i,y_j) \\ I_y(i,j-1) + s(x_i,y_j) \end{cases}$$

$$I_x(i,j) = \max\begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases}$$

$$I_y(i,j) = \max\begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

Equivalent ways of looking at the same problem.
Parameterized over different spaces: (i,j) vs (k)