Dynamic programming in less time and space

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Department of Computer Science

Space usage revisited

We said the fill step requires $O(mn)$ space

	ϵ	T	A	T	G	T	C	A	T	G	C
ϵ	0	8	16	24	32	40	48	56	64	72	80
T	8	0	8	16	24	32	40	48	56	64	72
A	16	8	0	8	16	24	32	40	48	56	64
C	24	16	8	2	10	18	24	32	40	48	56
G	32	24	16	10	2	10	18	26	34	40	48
T	40	32	24	16	10	2	10	18	26	34	42
C	48	40	32	24	18	10	2	10	18	26	34
A	56	48	40	32	26	18	10	2	10	18	26
G	64	56	48	40	32	26	18	10	6	10	18
C	72	64	56	48	40	34	26	18	12	10	10

Can we do better?

Assume we're only interested in cost / score in lower right-hand cell

Space usage revisited

	ϵ	T	A	T	G	T	C	A	T	G	C
ϵ	**0**	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**	**72**	**80**
T	**8**	**0**	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**	**72**
A											
C											
G											
T											
C											
A											
G											
C											

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Space usage revisited

Idea: just store current and previous rows. Discard older rows as we go.
(Likewise for columns or antidiagonals.)

	ϵ	T	A	T	G	T	C	A	T	G	C
ϵ	0	8	16	24	32	40	48	56	64	72	80
T	8	0	8	16	24	32	40	48	56	64	72
A	?										
C											
G											
T											
C											
A											
G											
C											

← Discard this row...

← ...once we begin this row

Only keeping $O(1)$ rows at a time, space bound becomes $O(\min(n, m))$ -- linear space

Space usage revisited

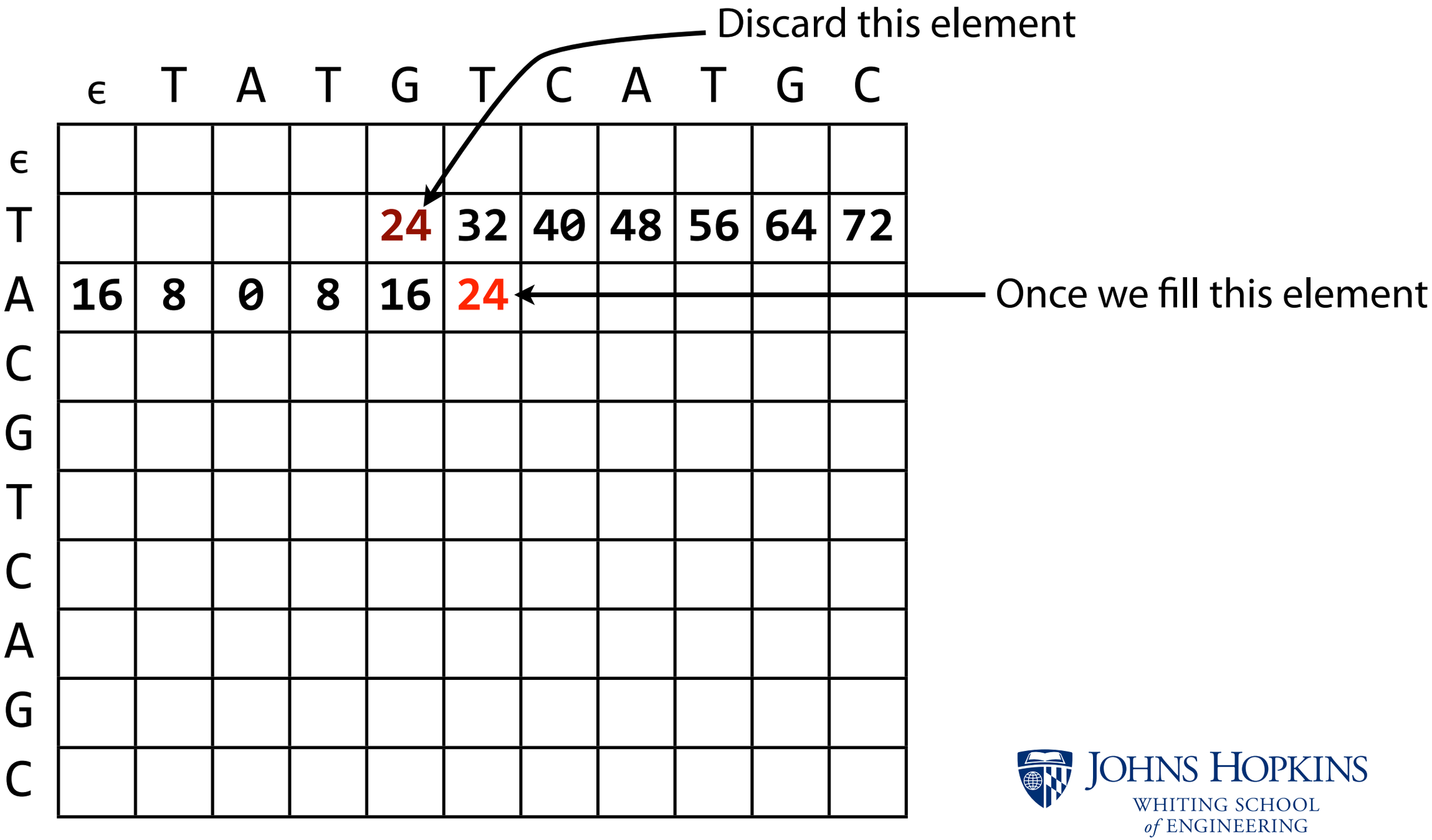
Idea: just store current and previous rows. Discard older rows as we go.
(Likewise for columns or antidiagonals.)

		ϵ	T	A	T	G	T	C	A	T	G	C
	ϵ											
T												
A												
C												
G												
T												
C												
A												
G		64	56	48	40	32	26	18	10	6	10	18
C		72	64	56	48	40	34	26	18	12	10	10

We get desired value / score, by looking in the lower right cell (global alignment)

Space usage revisited

More savings: discard *elements* as soon as they're no longer needed



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Space usage revisited

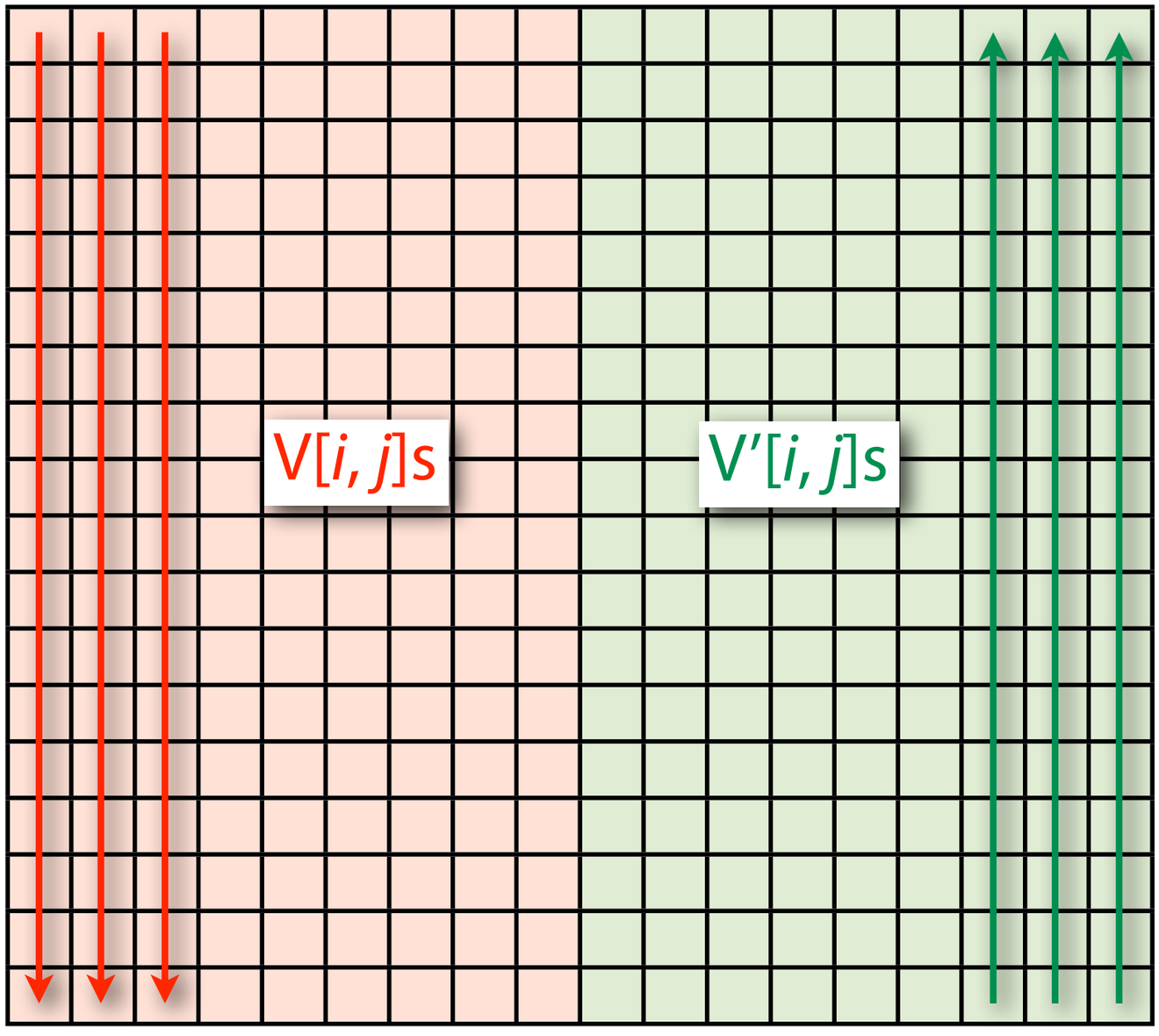
Can we get both the optimal score *and the alignment* in linear space?

For global alignment, we can...

Space usage revisited: subdividing matrix

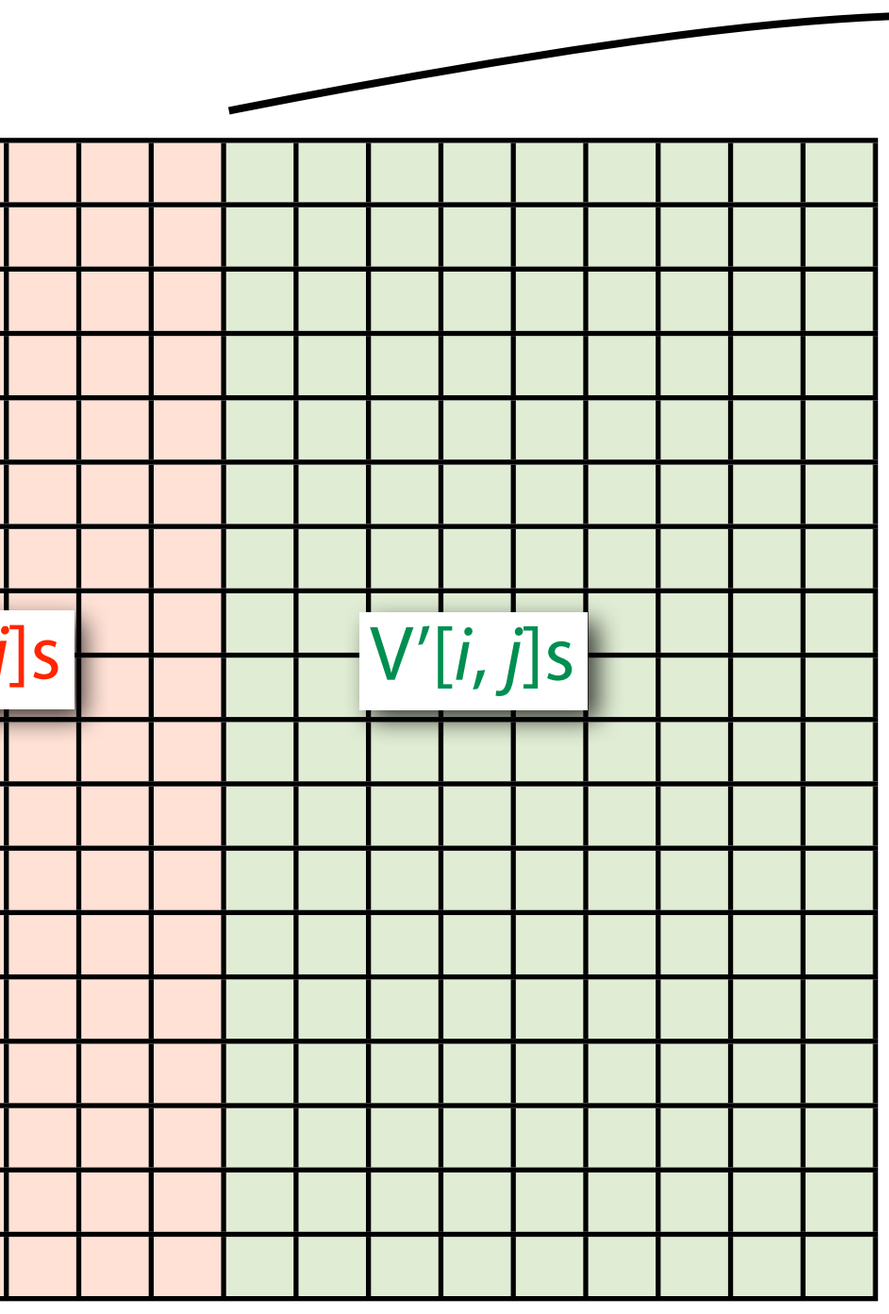
Assume global alignment. Idea: Split matrix into left half, filled as usual, and right half, filled "backwards." In both halves, only store current, previous columns.

Find V[*i*, *j*]s for successively longer prefixes of *x* and *y*



Find V′[*i*, *j*]s for successively longer suffixes of *x* and *y*

Space usage revisited: subdividing matrix



After fill, we have the center 2 columns

5	4
5	5
4	6
2	6
4	3
5	2
1	3
1	4
4	1
5	4
5	5
3	6
4	7
4	1
2	3
4	3
5	3
5	3

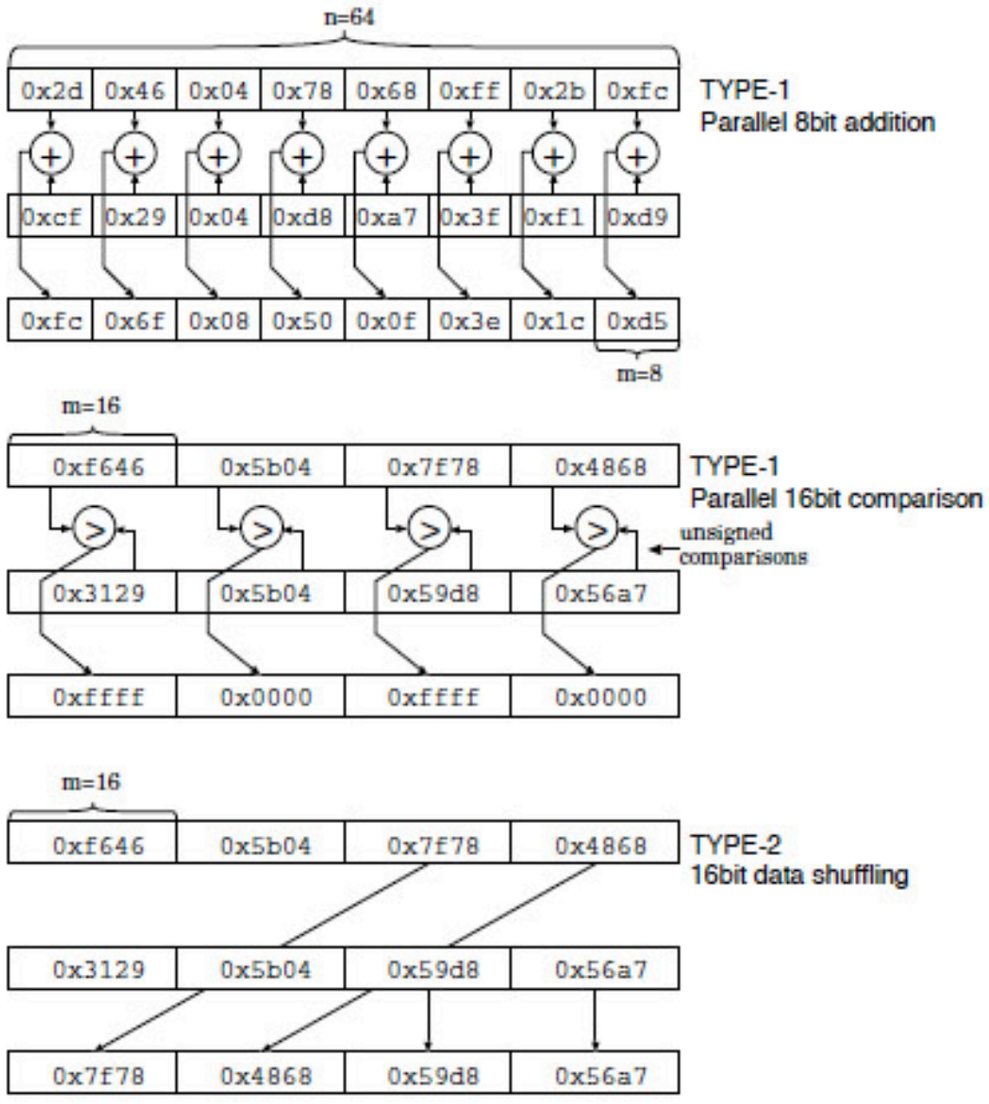
Given nearby cells from each column, we can calculate value for optimal global alignment *passing through those cells*

Opimal such value indicates where alignment crosses the center

Repeat recursively to solve the entire problem, including backtrace, in $O(mn)$ time and linear space

Hirschberg's algorithm See Gusfield 12.1

Data parallelism: SIMD operations



http://www.coins-project.org/international/COINSdoc.en/simd/simd.html

SIMD: Single Instruction, Multple Data

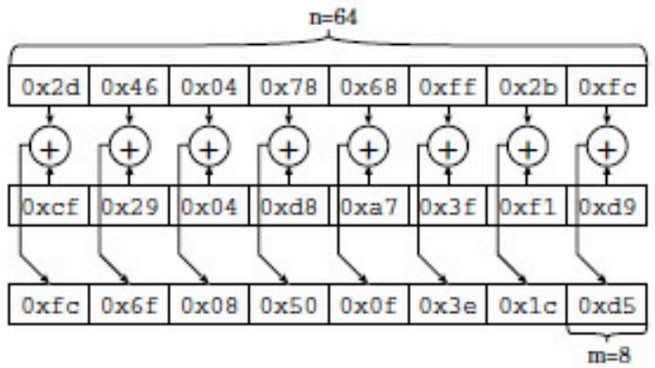
A SIMD operation performs several operations at once on *vectors* of operands

One instruction on a modern CPU can add two vectors of 8 16-bit numbers quickly:

134	45	14	73	86	782	67	36
+							
7	952	65	33	6	56	5	3
=							
141	997	79	106	92	838	72	39

Data parallelism

Can we take advantage of these operations when filling the matrix?



	-	T	A	T	G	T	C	A	T	G	C
-	0	8	16	24	32	40	48	56	64	72	80
T	8										
A	16										
C	24										
G	32										
T	40										
C	48										
A	56										
G	64										
C	72										

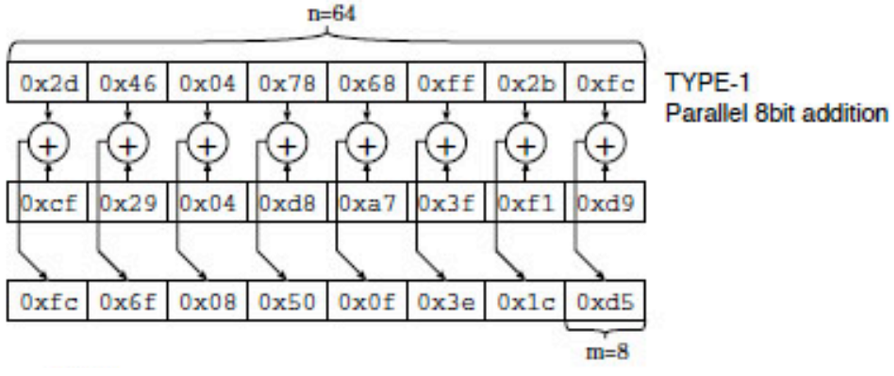
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Data parallelism

Yes, dynamic programming has "data parallelism"

E.g. cells in **red** are calculated in the same way: different inputs but same operations. None depend on the others.

Things we do when filling in a cell: add, max, etc, can be packed into vectors and done for many cells in parallel:



	-	T	A	T	G	T	C	A	T	G	C
-	0	8	16	24	32	40	48	56	64	72	80
T	8	0	8	16	24	32					
A	16	8	0	8	16						
C	24	16	8	2							
G	32	24	16								
T	40	32									
C	48										
A	56										
G	64										
C	72										

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Data parallelism

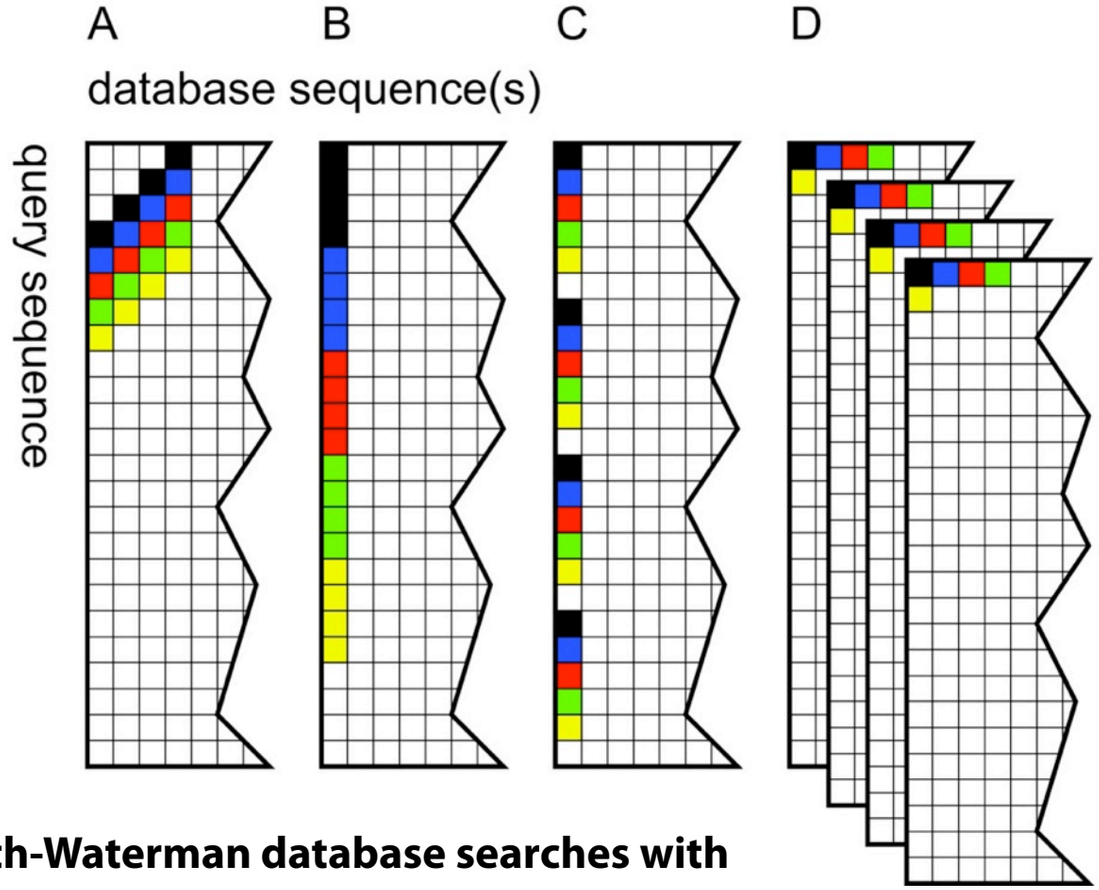
Variations on this idea are quite practical and used a lot in practice

A Wozniak A. **Using video-oriented instructions to speed up sequence comparison**. *Comput Appl Biosci*. 1997 Apr;13(2):145-50.

B Rognes T, Seeberg E. **Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors**. *Bioinformatics*. 2000 Aug;16(8):699-706.

C Farrar M. **Striped Smith-Waterman speeds database searches six times over other SIMD implementations**. *Bioinformatics*. 2007 Jan 15;23(2): 156-61.

D Rognes T. **Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation**. *BMC Bioinformatics*. 2011 Jun 1;12:221.



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Figure from: Rognes T. **Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation**. *BMC Bioinformatics*. 2011 Jun 1;12:221.

Dynamic programming summary

- Edit distance is harder to calculate than Hamming distance, but there is a *O*(*mn*) time dynamic progamming algorithm
- Global alignment generalizes edit distange to use a cost function
- Slight tweaks to global alignment turn it into an algorithm for:
  - Longest Common Subsequence
  - Finding approximate occurrences of P in T
- Local alignment also has a *O*(*mn*)-time dynamic programming solution
- Further efficiencies are possible:
  - If no alignment is needed, global/local alignment can be made linear-space
  - Even if alignment is needed, global alignment can be made linear-space with Hirschberg
  - SIMD instructions can fill in chunks of cells at a time

More ideas: http://en.wikipedia.org/wiki/Smith-Waterman_algorithm#Accelerated_versions