MIT 18.700/18.701/6.047/6.878/HST.507
Machine Learning in Computational Biology

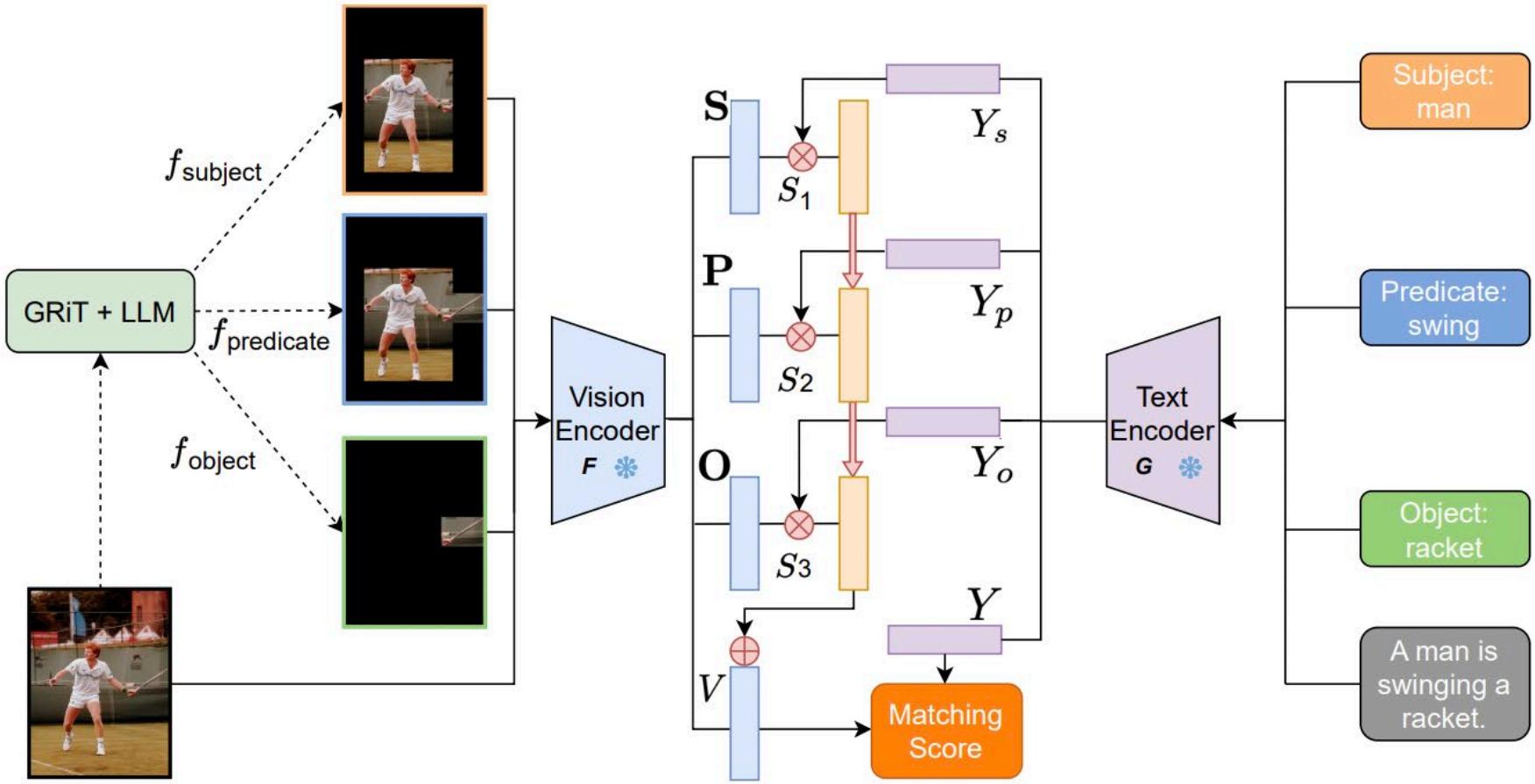
Lecture 4:
Sequential Pattern Matching
Alignment, DP, Hashing

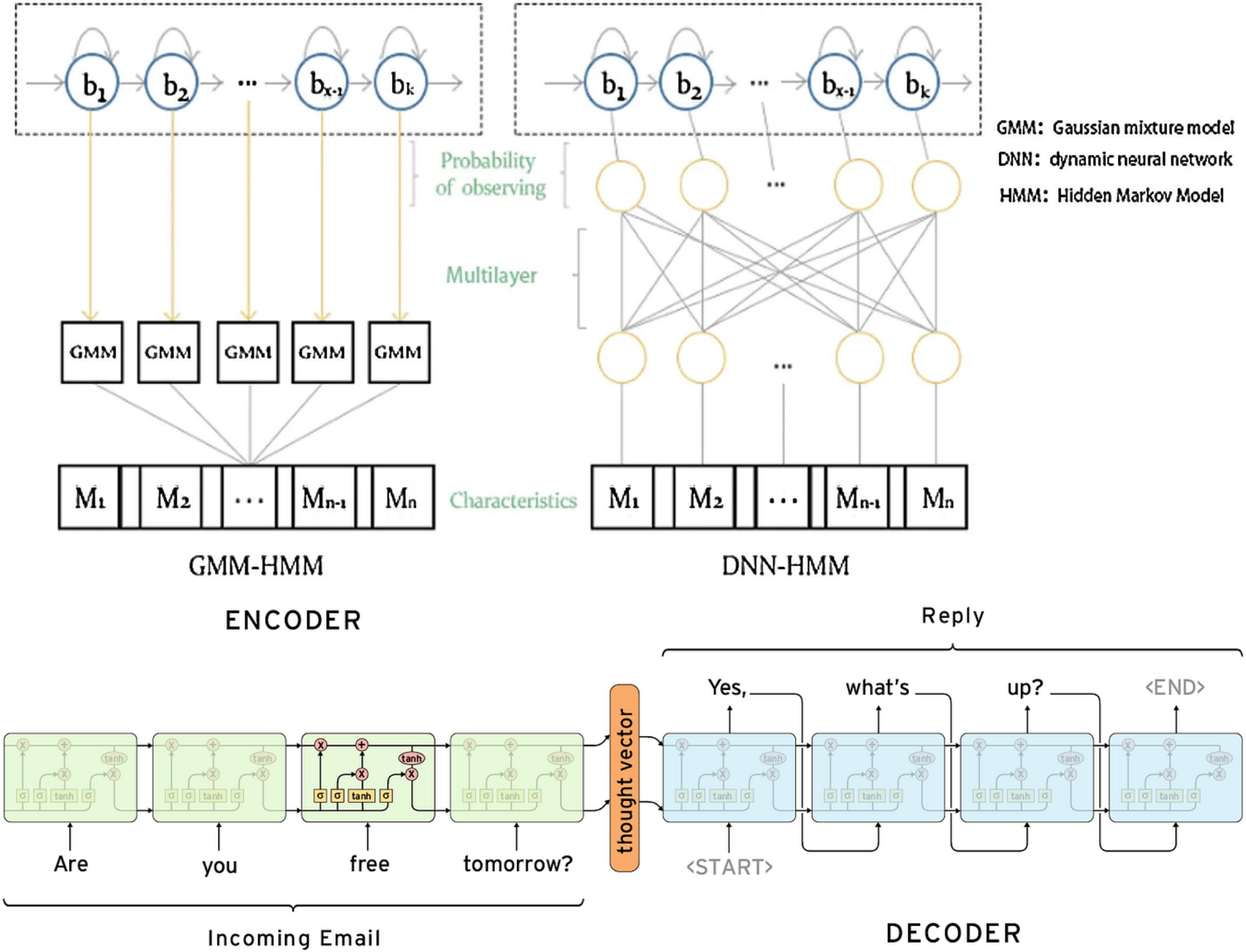
Comparative Genomics & Evolution
Computation Re-use, Dynamic Programming
Alignment Matrix, Paths, Traceback, $2^N \rightarrow N^2$
Local Alignment, Linear-Time, Linear Space
Hashing, Content-based Search, Expected Runtime
BLAST, Inexact Matching, PSI-BLAST
Probabilistic Foundations of Alignment Scores

Prof. Manolis Kellis – manoli@mit.edu – MIT Computer Science & AI Lab

MLCB - Machine Learning in Computational Biology - Fall 2024 - Profs. Manolis Kellis + Eric Alm - 6.8700/6.8701/20.s900/20.s948/HST.507

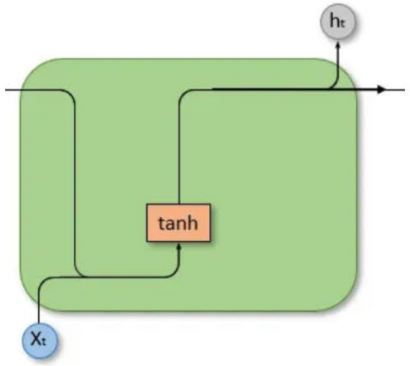
Homeworks	Project / Mentoring (Fri)	Wk	Date	Lec	Topic
			Introduction: Machine Learning, Deep Learning, Generative AI, and the Unification of Biology		
HW0 out Thu 9/5		1	Thu-Sep-05	L1	Course Overview, Machine Learning, Deep Learning, Inference, Genome, Proteins, Chemistry, Imaging
			Module 1: Genomics, Epigenomics, Single-Cell, Networks, Circuitry		
HW0 due Wed 9/11		2	Tue-Sep-10	L2	Expression Analysis, Clustering/Classification, Gaussian Mixture Models, K-means, Bayesian Inf, Gen-vs-DiscrML
HW1 out Thu 9/12	0=Self Introductions	2	Thu-Sep-12	L3	Single-cell genomics, sc-mutli-omics, non-linear embeddings, spatial transcriptomics, next-gen technologies
		3	Tue-Sep-17	L4	Sequential Data, Alignment, DynProg, Hidden Markov Models, Parsing, Posterior Decoding, HMM architectures
		3	Thu-Sep-19	L5	Epigenomics: Signal Modeling, Peak calling, Chromatin states, 3D structure, Hi-C, Genome Topology
		4	Tue-Sep-24	L6	Regulatory Genomics: Motifs, Information, ChIP, Gibbs Sampling, EM, CNNs for Genome Parsing
HW1 due Mon 9/30	1=Select previous paper(s)	4	Thu-Sep-26	L7	Regulatory Networks: Graphs, Linear Algebra, PCA, SVD, Dimentionality Reduction, TF-enhancer-gene circuitry
			Module 2: Protein Structure, Protein Language Models, Geometric Deep Learning		
HW2 out Thu 10/3		5	Tue-Oct-01	L8	Intro to structural biology
		5	Thu-Oct-03	L9	Protein structure and folding: Diffusion models, Cryo-EM, Protein design
		6	Tue-Oct-08	L10	Intro to transformers and Large Language Models LLMs
	2=Proposal+Feasibility	6	Thu-Oct-10	L11	Protein Language Models PLMs and Transfer Learning
		7	Tue-Oct-15	-	-- No Class -- Student holiday
HW2 due Mon 10/21		7	Thu-Oct-17	L12	DNA language models: Chromatin Structure
			Module 3: Chemistry, Therapeutics, Graph Neural Networks		
HW3 out Thu 10/24		8	Tue-Oct-22	L13	Overview of drug development
	3=OffHrs|Update|Feedback	8	Thu-Oct-24	L14	Intro to small molecules
		9	Tue-Oct-29	L15	Representation of small molecules: Graphs, GNNs, Transformers, RDKit
		9	Thu-Oct-31	L16	Docking: Small molecule - proteins docking
		10	Tue-Nov-05	L17	Disease Association Mapping, genetics, GWAS, linkage analysis, disease circuitry, variant-to-function
HW3 due Tue 11/12	4=OffHrs|Update|Feedback	10	Thu-Nov-07	L18	Quantitative trait mapping, molecular traits, eQTLs, mediation analysis, iMWAS, multi-modal QTLs
			Module 4: Electronic Health Records, Imaging, Evolution, Metabolism		
No HW		11	Tue-Nov-12	L19	Electronic Health Records, AllOfUs, UKBioBank, Medical Genomics, Pop-Scale Cohorts, Multi-Ancestry [not quiz'd]
		11	Thu-Nov-14	L20	**-- In-class Quiz**
		12	Tue-Nov-19	L21	Imaging methods for biological applications
	5=Midcourse report	12	Thu-Nov-21	L22	Comparative genomics, Conservation, Evolutionary signatures, PhyloCSF, RNA structure, Motif BLS2conf
		13	Tue-Nov-26	L23	Evolution, Phylogenetics, Phylogenomics, Duplication, RNA world, RNA folding, lincRNAs, RNA modifications, m6A
		13	Thu-Nov-28	-	-- No Class -- Thanksgiving Holiday
		14	Tue-Dec-03	L24	Modeling metabolism: Flux balance analysis
	6=WriteUp, Slides Due	14	Thu-Dec-05	L25	Measuring metabolism: Metabolomics and Deep Learning
			Final Projects		
	7=In Class Presentations	15	Tue-Dec-10	L26	**Project Presentations (6-8 mins/team). Report due Fri@11.59p, Slides due Mon@11.59p, Present Live Tue**


GRiT + LLM
f_{subject}
$f_{\text{predicate}}$
f_{object}
Vision Encoder
F
S
P
O
S_1
S_2
S_3
Y_s
Y_p
Y_o
Y
V
Matching Score
Text Encoder
G
Subject: man
Predicate: swing
Object: racket
A man is swinging a racket.

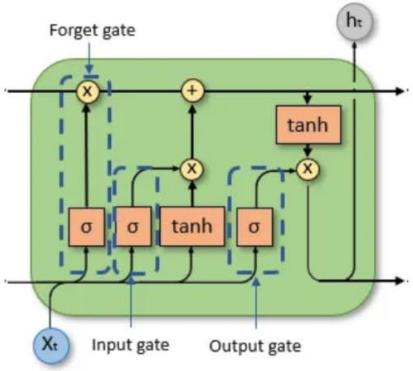

GMM: Gaussian mixture model
DNN: dynamic neural network
HMM: Hidden Markov Model
Probability of observing
Multilayer
Characteristics
GMM-HMM
DNN-HMM
ENCODER
Reply
thought vector
Are
you
free
tomorrow?
<START>
Yes,
what's
up?
<END>
Incoming Email
DECODER

Sequential models in the age of deep learning

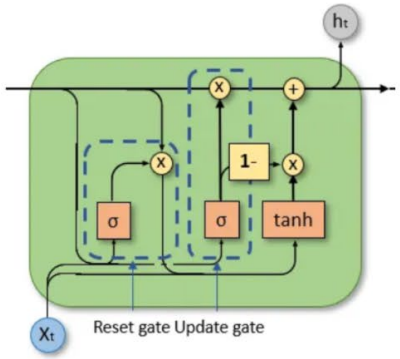
RNN



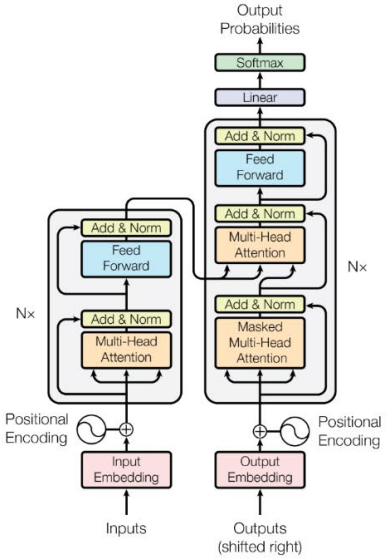
LSTM



GRU



Transformers



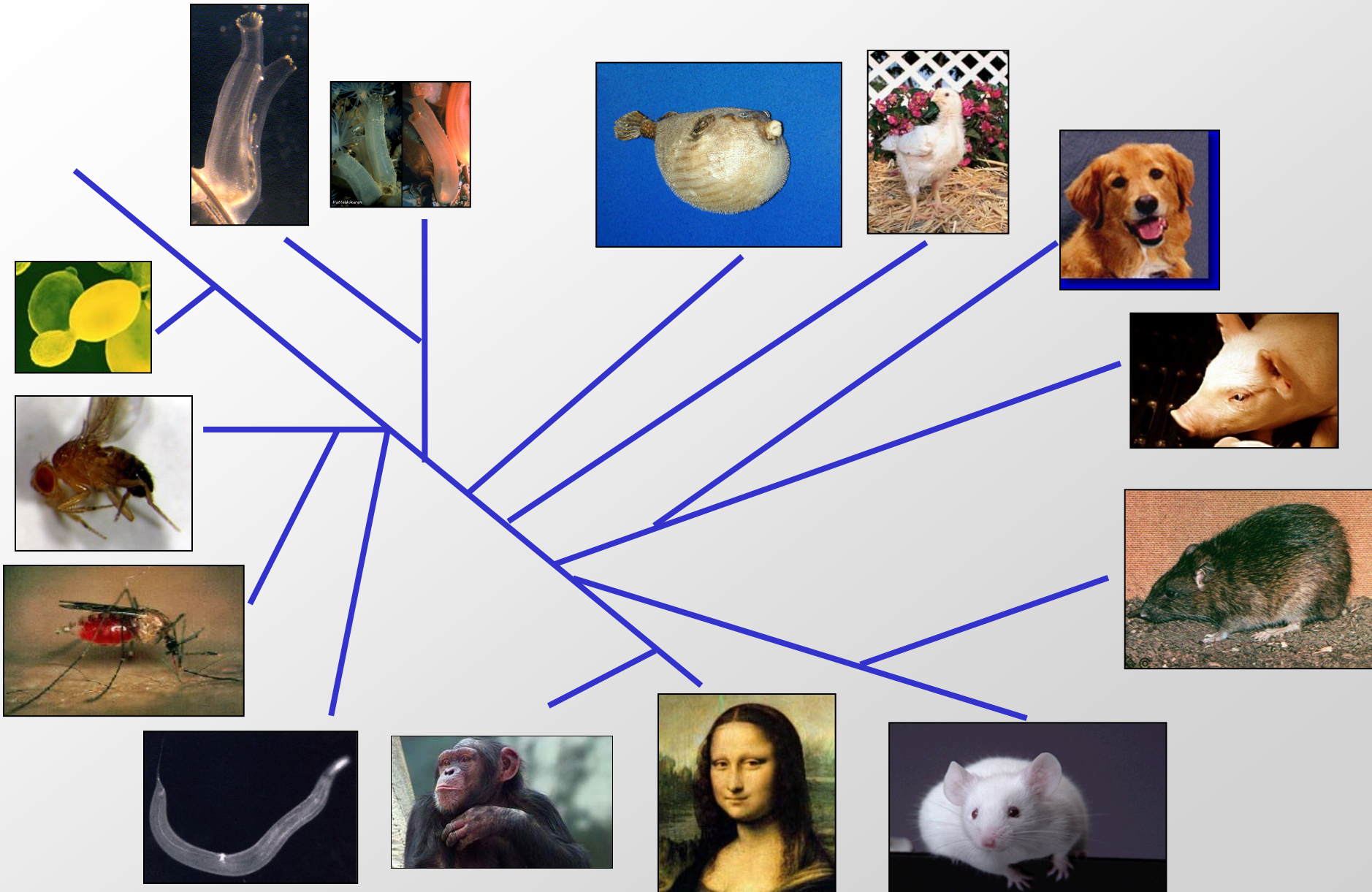
Recurrent Neural Network (RNN)	Long Short Term Memory (LSTM)	Gated Recurrent Unit (GRU)	Transformers
RNNs are foundational sequence models that process sequences iteratively, using the output from the previous step as an input to the current step.	LSTMs are an enhancement over standard RNNs, designed to better capture long-term dependencies in sequences.	GRUs are a variation of LSTMs with a simplified gating mechanism.	Transformers move away from recurrence and focus on self-attention mechanisms to process data in parallel

Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

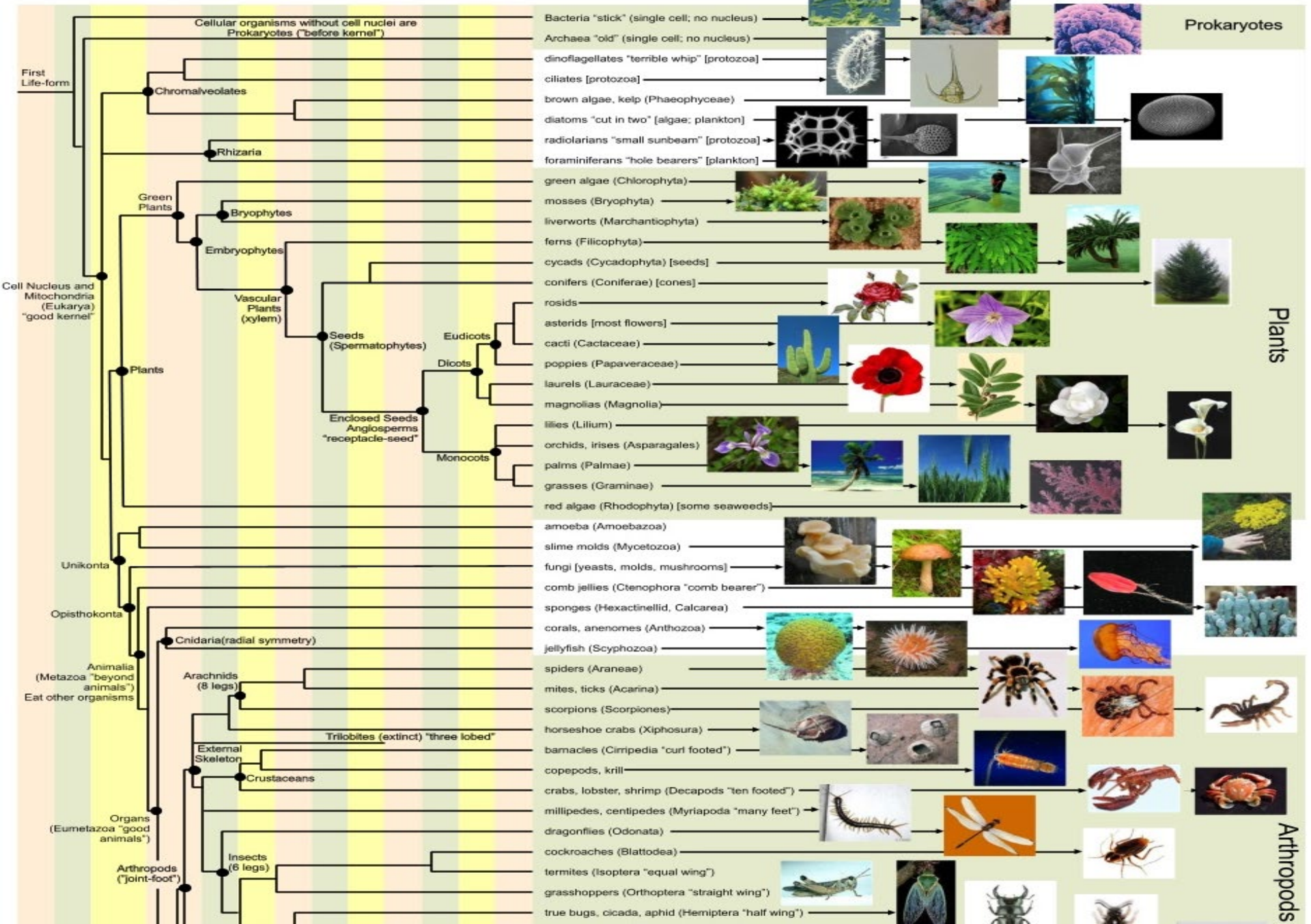
Comparative Genomics and Evolutionary Signatures

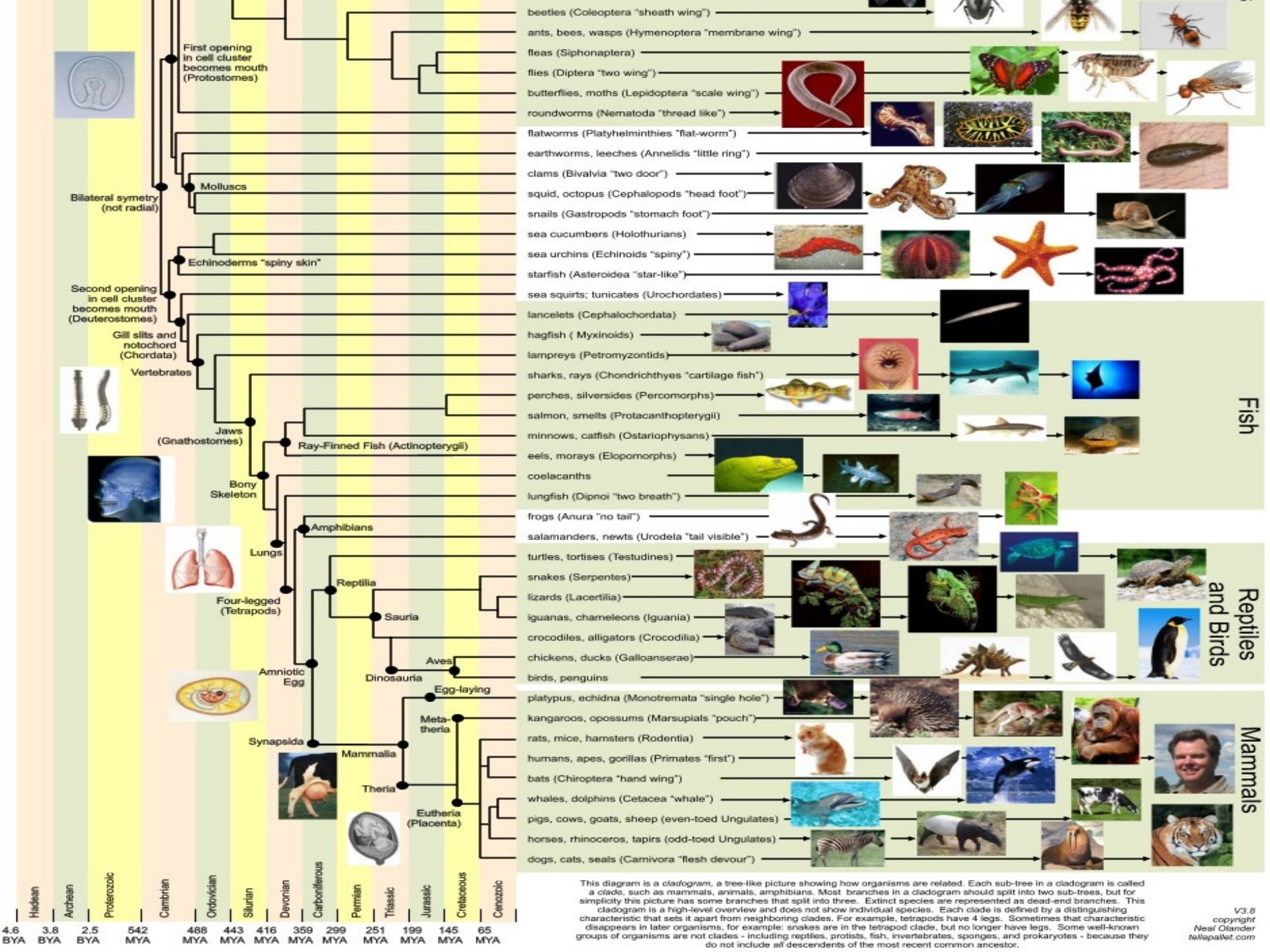
Alignment: all species/genes share common ancestry



Slide credit: Serafim Batzoglou

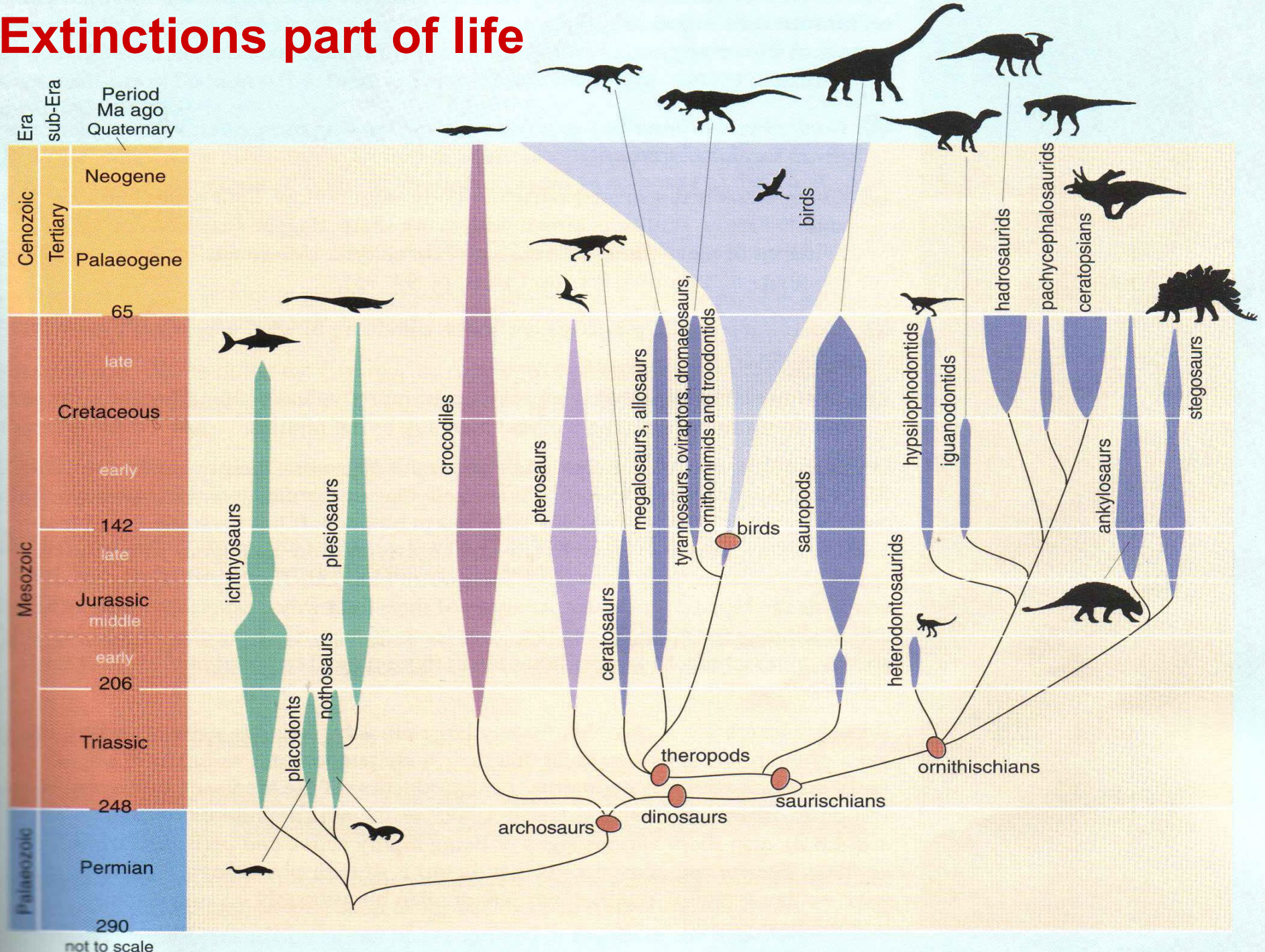
Tree of Life


Cellular organisms without cell nuclei are Prokaryotes ("before kernel")
First Life-form
Chromalveolates
Rhizaria
Green Plants
Bryophytes
Embryophytes
Vascular Plants (xylem)
Seeds (Spermatophytes)
Eudicots
Dicots
Enclosed Seeds Angiosperms "receptacle-seed"
Monocots
Cell Nucleus and Mitochondria (Eukarya) "good kernel"
Plants
Unikonta
Opisthokonta
Animalia (Metazoa "beyond animals") Eat other organisms
Cnidaria(radial symmetry)
Arachnids (8 legs)
Trilobites (extinct) "three lobed"
External Skeleton
Crustaceans
Organs (Eumetazoa "good animals")
Arthropods ("joint-foot")
Insects (6 legs)
Bacteria "stick" (single cell; no nucleus)
Archaea "old" (single cell; no nucleus)
dinoflagellates "terrible whip" [protozoa]
ciliates [protozoa]
brown algae, kelp (Phaeophyceae)
diatoms "cut in two" [algae; plankton]
radiolarians "small sunbeam" [protozoa]
foraminiferans "hole bearers" [plankton]
green algae (Chlorophyta)
mosses (Bryophyta)
liverworts (Marchantiophyta)
ferns (Filicophyta)
cycads (Cycadophyta) [seeds]
conifers (Coniferae) [cones]
rosids
asterids [most flowers]
cacti (Cactaceae)
poppies (Papaveraceae)
laurels (Lauraceae)
magnolias (Magnolia)
lilies (Lilium)
orchids, irises (Asparagales)
palms (Palmae)
grasses (Graminae)
red algae (Rhodophyta) [some seaweeds]
amoeba (Amoebazoa)
slime molds (Mycetozoa)
fungi [yeasts, molds, mushrooms]
comb jellies (Ctenophora "comb bearer")
sponges (Hexactinellid, Calcarea)
corals, anenomes (Anthozoa)
jellyfish (Scyphozoa)
spiders (Araneae)
mites, ticks (Acarina)
scorpions (Scorpiones)
horseshoe crabs (Xiphosura)
barnacles (Cirripedia "curl footed")
copepods, krill
crabs, lobster, shrimp (Decapods "ten footed")
millipedes, centipedes (Myriapoda "many feet")
dragonflies (Odonata)
cockroaches (Blattodea)
termites (Isoptera "equal wing")
grasshoppers (Orthoptera "straight wing")
true bugs, cicada, aphid (Hemiptera "half wing")
Prokaryotes
Plants
Arthropods

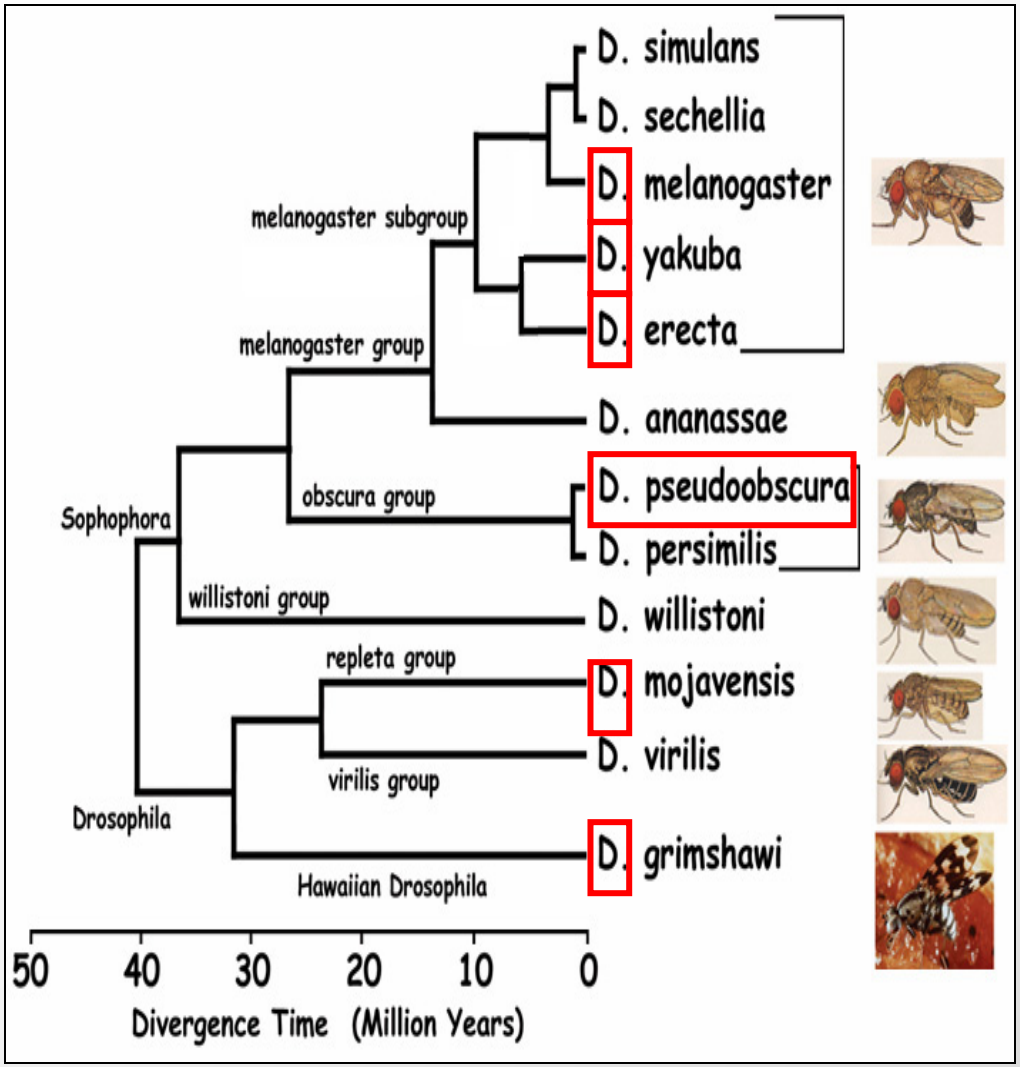


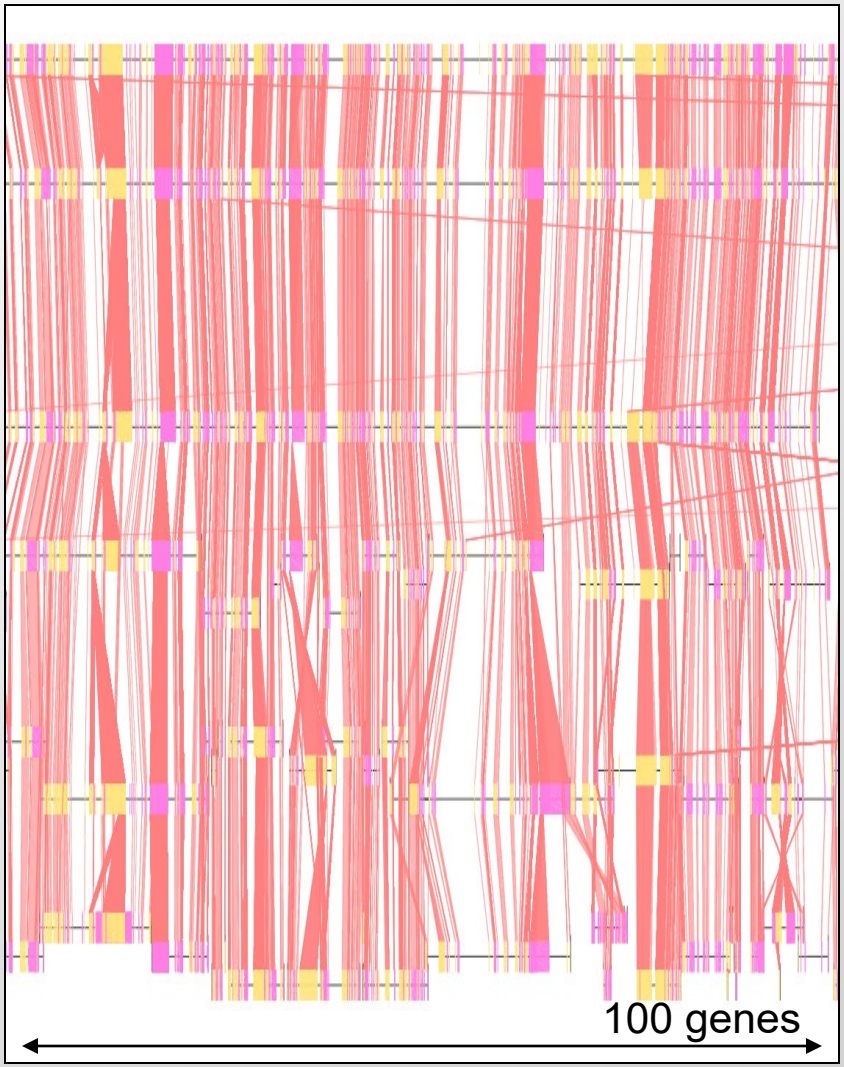
This diagram is a *cladogram*, a tree-like picture showing how organisms are related. Each sub-tree in a cladogram is called a *clade*, such as mammals, animals, amphibians. Most branches in a cladogram should split into two sub-trees, but for simplicity this picture has some branches that split into three. Extinct species are represented as dead-end branches. This cladogram is a high-level overview and does not show individual species. Each clade is defined by a distinguishing characteristic that sets it apart from neighboring clades. For example, tetrapods have 4 legs. Sometimes that characteristic disappears in later organisms, for example: snakes are in the tetrapod clade, but no longer have legs. Some well-known groups of organisms are not clades - including reptiles, protists, fish, invertebrates, sponges, and prokaryotes - because they do not include all descendents of the most recent common ancestor.

Extinctions part of life



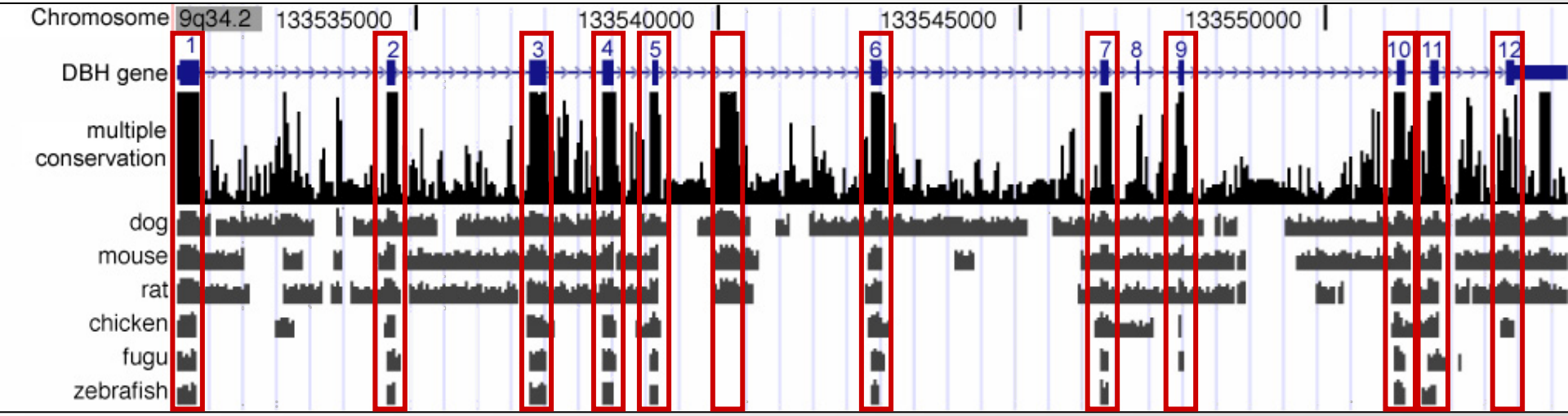
Genome-wide alignments reveal orthologous segments





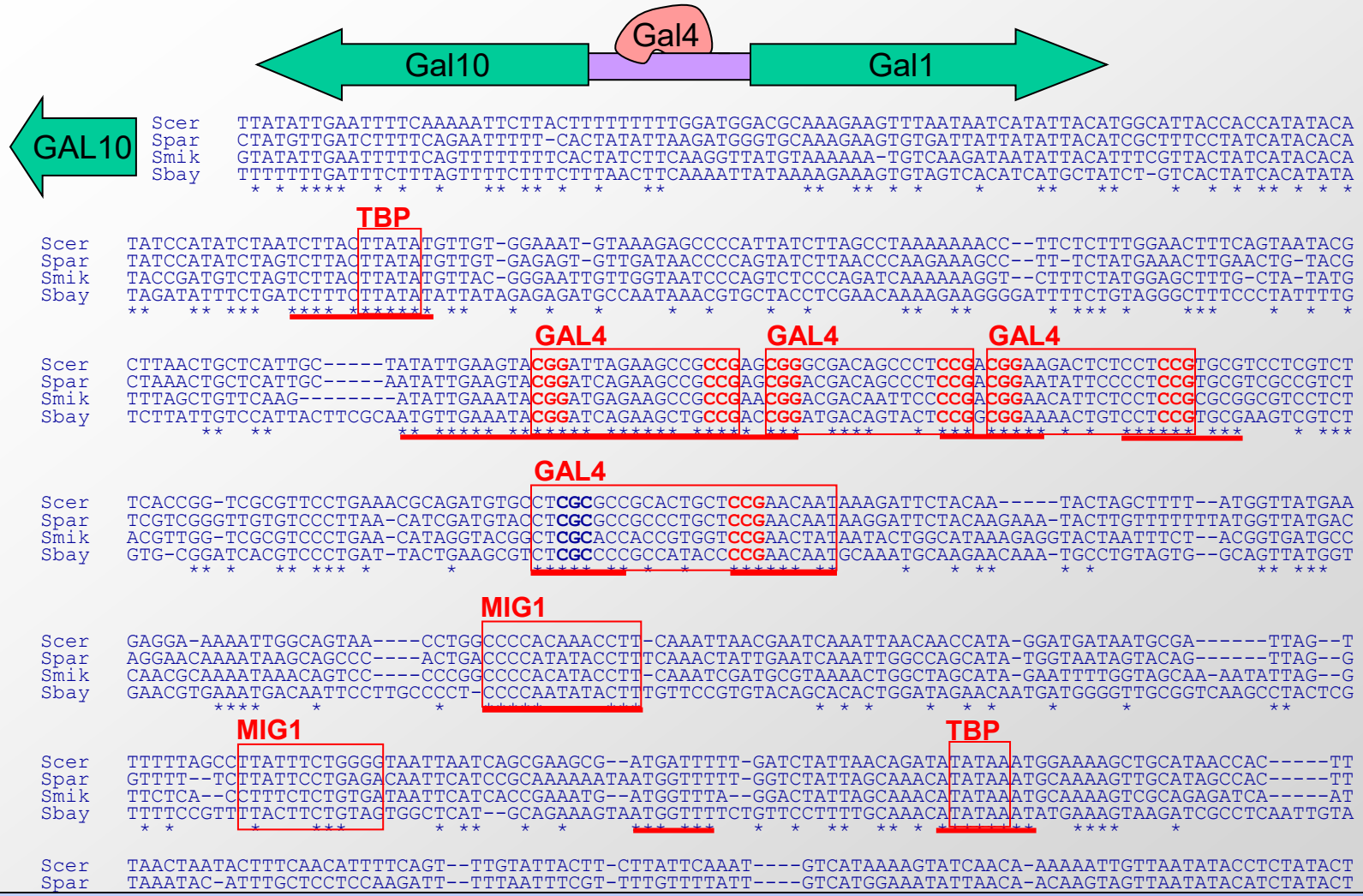
- Genome-wide alignments span entire genome
- Comparative identification of functional elements

Comparative genomics reveals conserved regions



- **Comparative genomics can reveal functional elements**
  - For example: exons are deeply conserved to mouse, chicken, fish
  - Many other elements are also strongly conserved: exons / regulatory?
- **Develop methods for estimating the level of constraint**
  - Count the number of edit operations, number of substitutions and gaps
  - Estimate the number of mutations (including estimate of back-mutations)
  - Incorporate information about neighborhood: conservation ‘windows’
  - Estimate the probability of a constrained ‘hidden state’: HMMs next week
  - Use phylogeny to estimate tree mutation rate, or ‘rejected substitutions’
  - Allow different portions of the tree to have different rates: phylogenetics

Alignment: Evolution preserves functional elements!

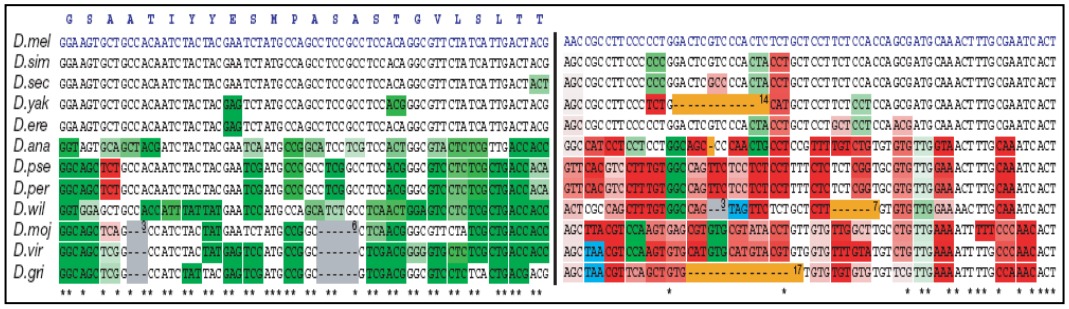


We can ‘read’ evolution to reveal functional elements

Conservation island

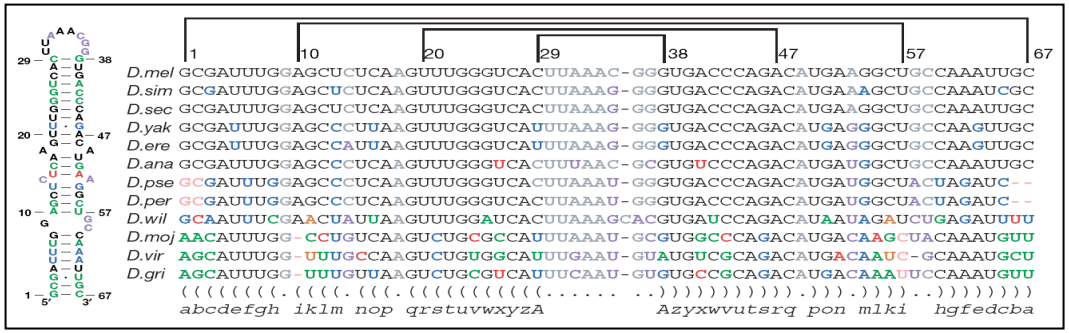
Yeast (Kellis et al, Nature 2003), Mammals (Xie, Nature 2005), Fly (Stark et al, Nature 07)

Evolutionary signatures for diverse functions



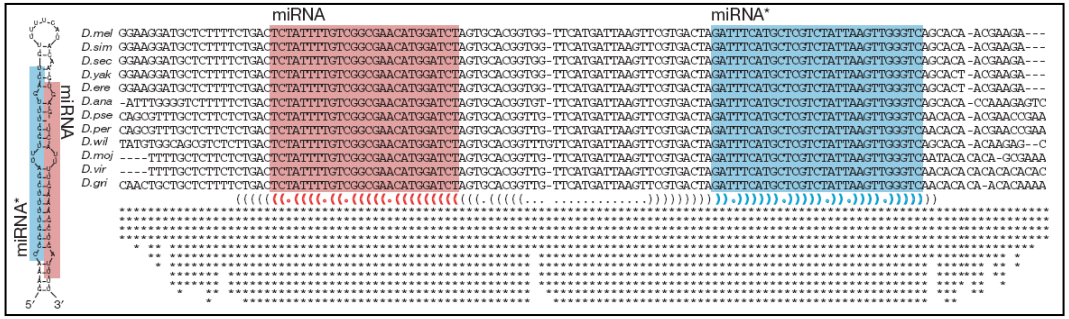
Protein-coding genes

- Codon Substitution Frequencies
- Reading Frame Conservation



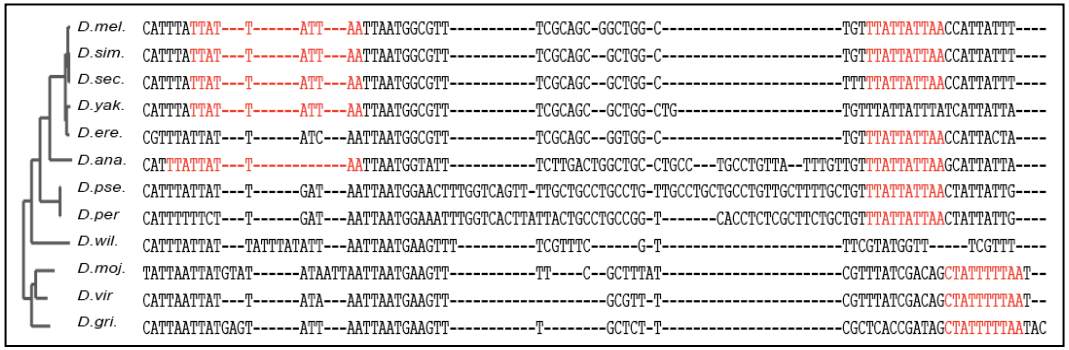
RNA structures

- Compensatory changes
- Silent G-U substitutions



microRNAs

- Shape of conservation profile
- Structural features: loops, pairs
- Relationship with 3'UTR motifs

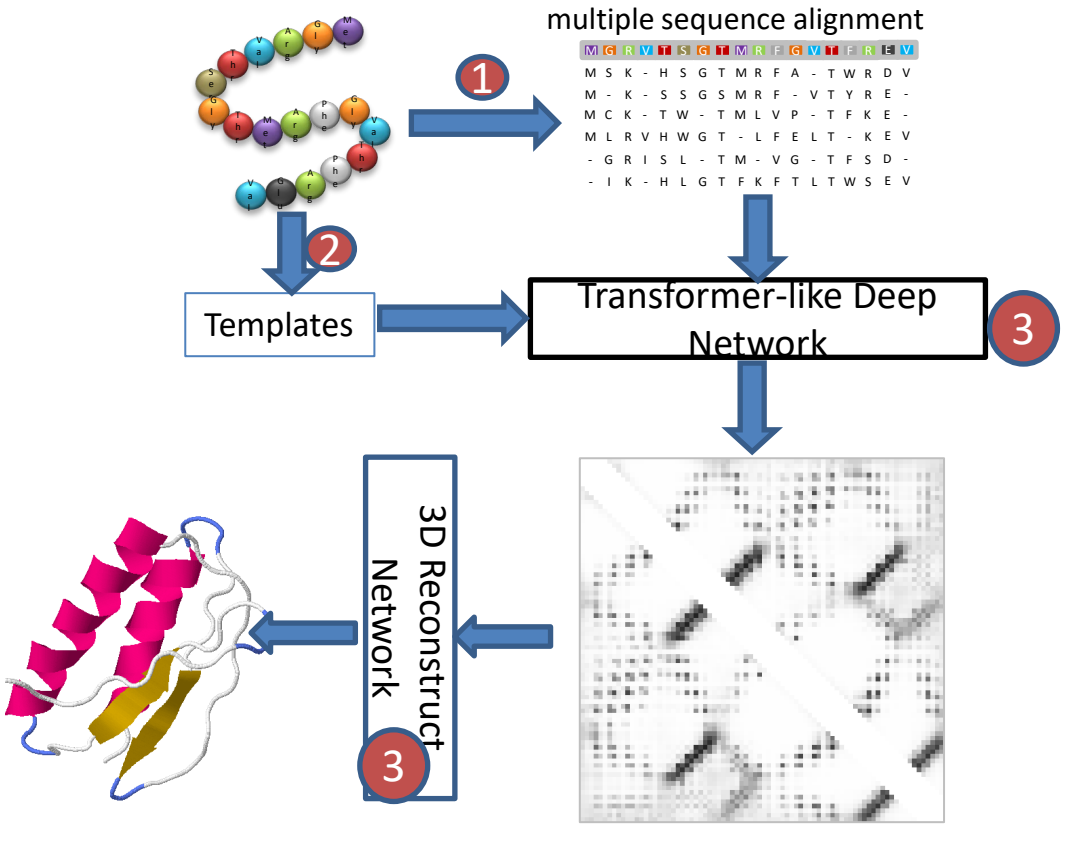


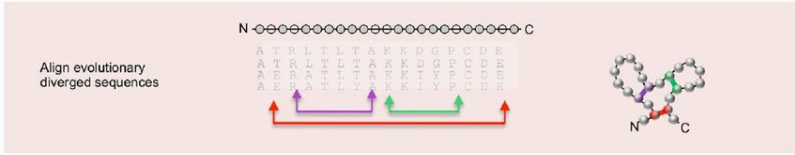
Regulatory motifs

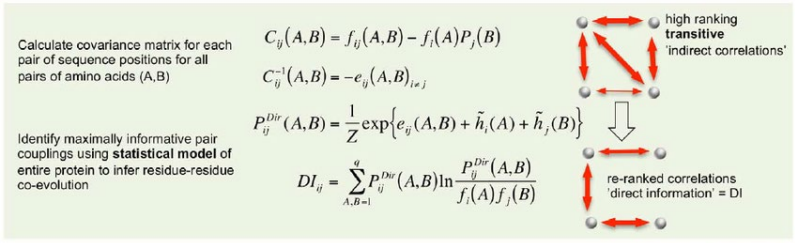
- Mutations preserve consensus
- Increased Branch Length Score
- Genome-wide conservation

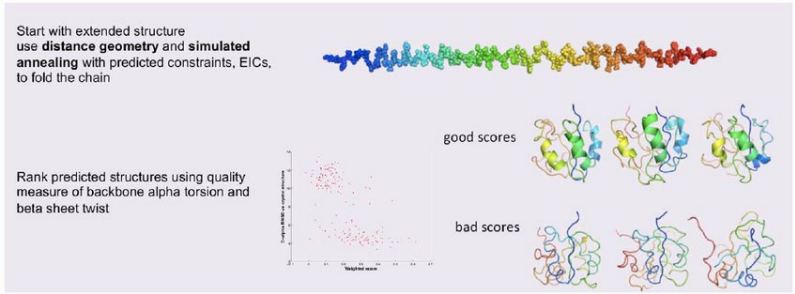
Stark et al, Nature 2007

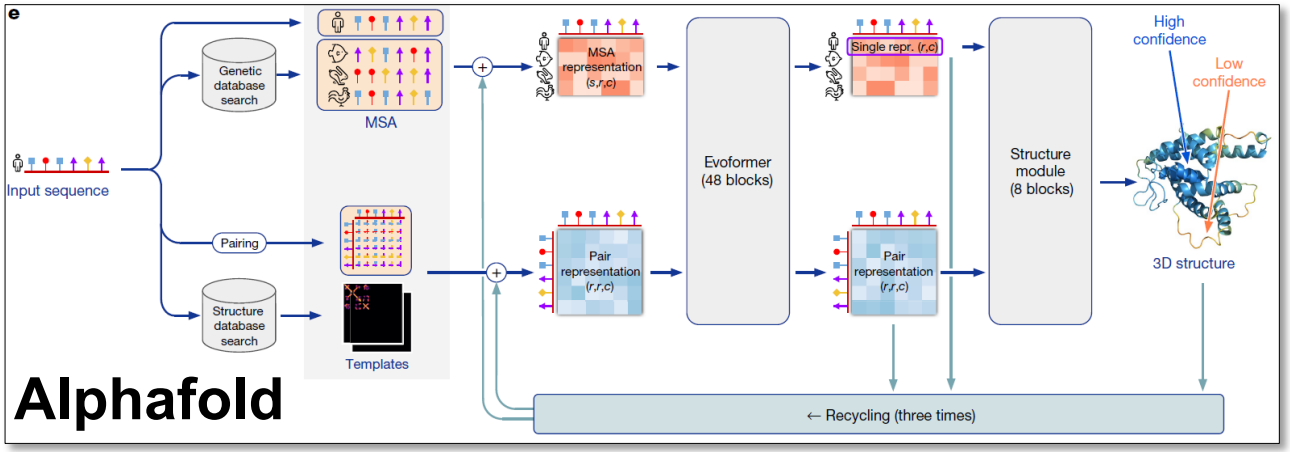
Protein Folding: AA pairs co-evolution evidence

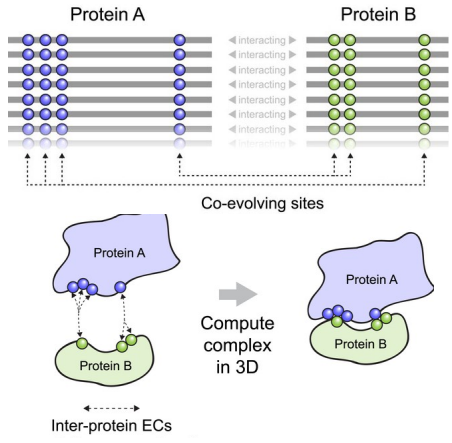








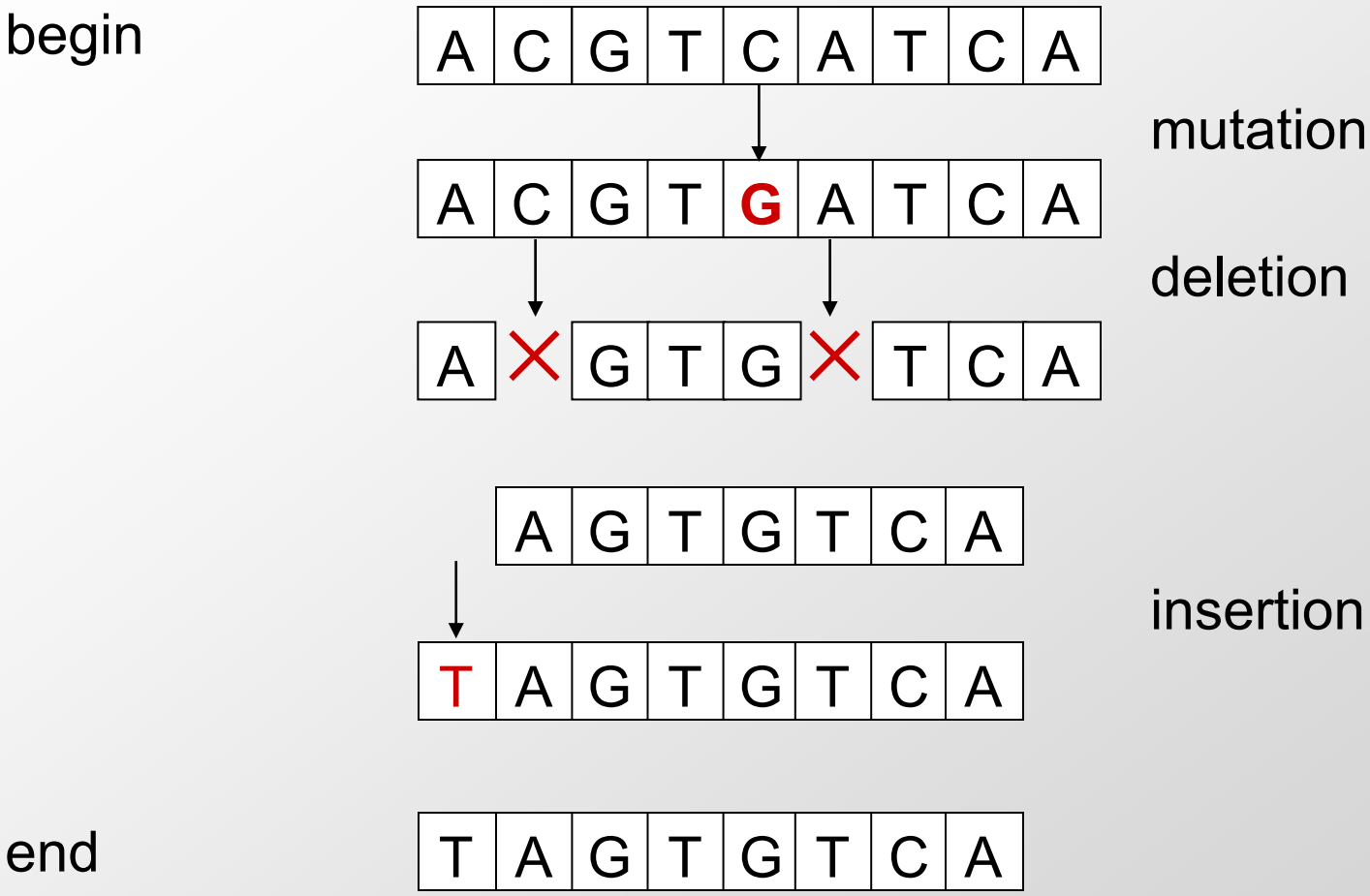




Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

Genomes change over time



Goal of alignment: Infer edit operations

begin

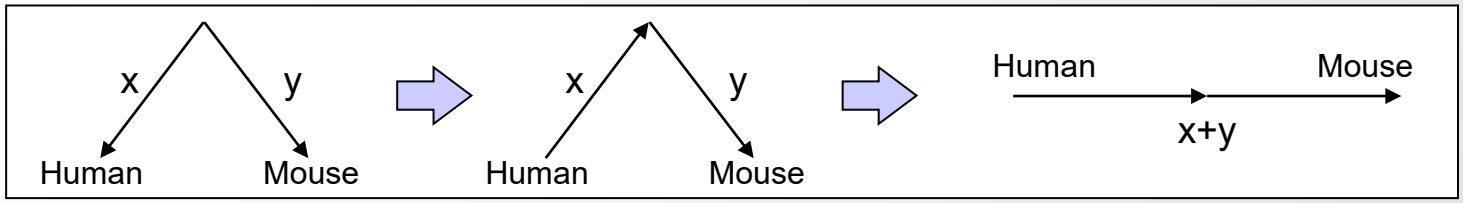
A	C	G	T	C	A	T	C	A

end

T	A	G	T	G	T	C	A

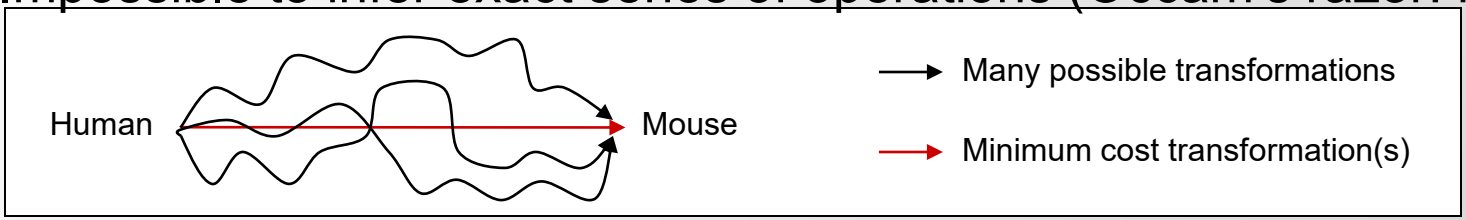
From Bio to CS: Formalizing the problem

- Define set of evolutionary operations (insertion, deletion, mutation)
  - Symmetric operations allow time reversibility (part of design choice)

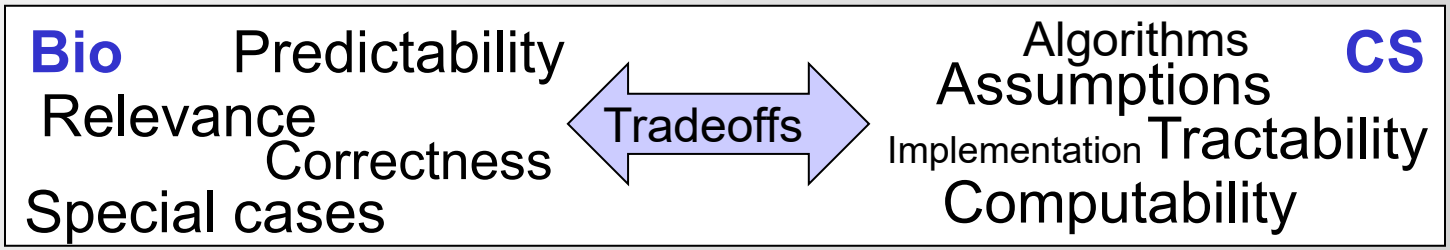


(Exception: methylated CpG dinucleotides → TpG/CpA non-symmetric)

- Define optimality criterion (min number, min cost)
  - Impossible to infer exact series of operations (Occam's razor: find min)



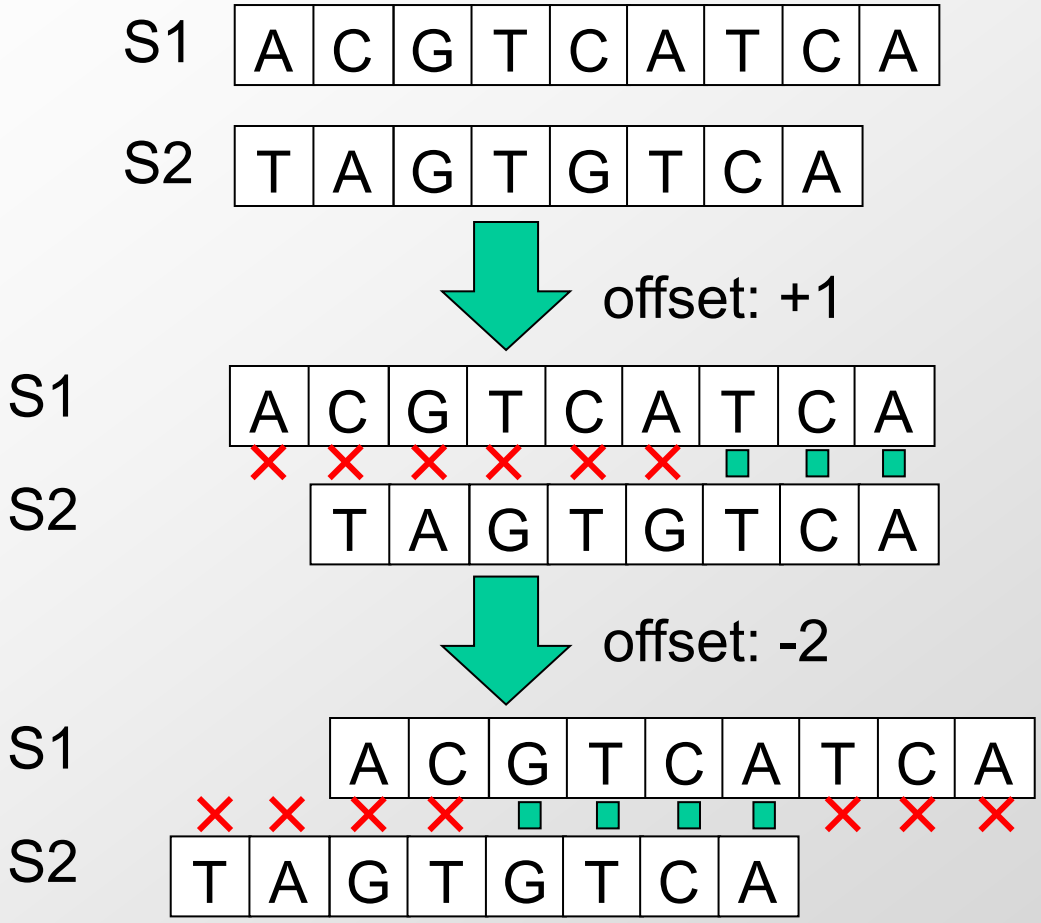
- Design algorithm that achieves that optimality (or approximates it)
  - Tractability of solution depends on assumptions in the formulation



Note: Not all decisions are conflicting (some are both relevant and tractable) (e.g. Pevzner vs. Sankoff and directionality in chromosomal inversions)

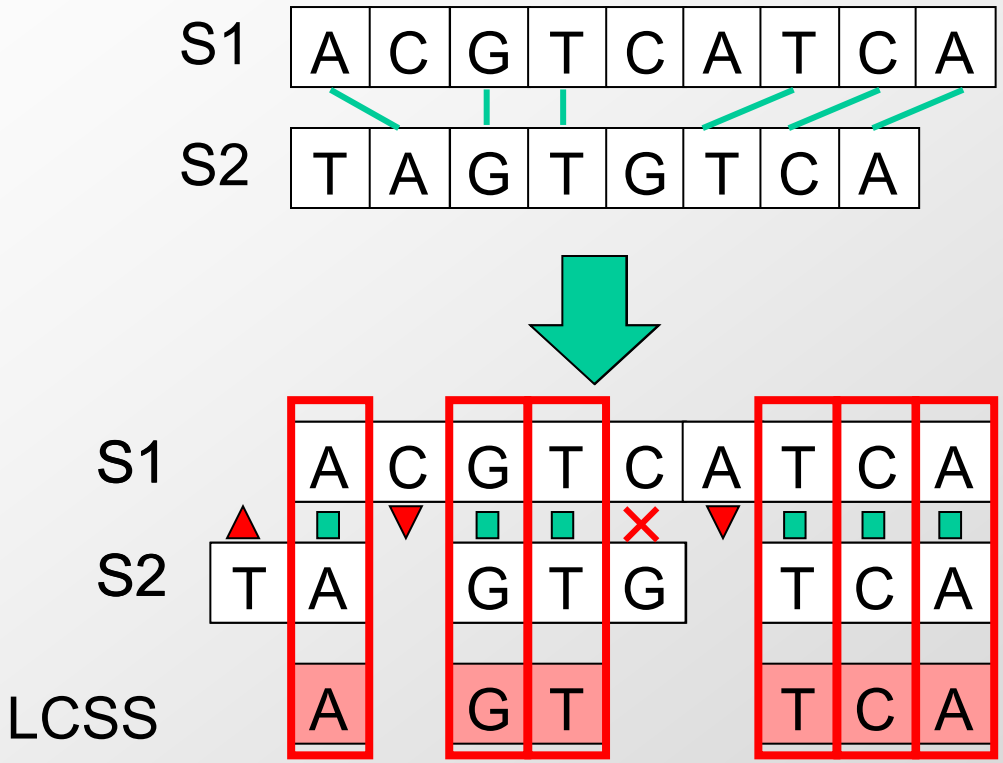
Formulation 1: Longest common substring

- Given two possibly related strings S1 and S2
  - What is the longest common substring? (no gaps)



Formulation 2: Longest common subsequence

- Given two possibly related strings S1 and S2
  - What is the longest common subsequence? (gaps allowed)



Related to:
Edit distance:

- Number of changes needed for S1→S2
- Uniform scoring function

Formulation 3: Sequence alignment

- Allow gaps (fixed penalty)
  - Insertion & deletion operations
  - Unit cost for each character inserted or deleted
- Varying penalties for edit operations
  - Transitions (Pyrimidine⇔Pyrimidine, Purine⇔Purine)
  - Transversions (Purine ⇔ Pyrimidine changes)
  - Polymerase confuses Aw/G and Cw/T more often
- Varying penalties for different Amino-Acid Pairs
  - BLOSUM62/PAM matrices (see last section)

Scoring function:

Match(x,x) = +1

Mismatch(A,G)= -½

Mismatch(C,T)= -½

Mismatch(x,y) = -1

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W

	purine		pyrimid.	
	A	G	T	C
A	+1	-½	-1	-1
G	-½	+1	-1	-1
T	-1	-1	+1	-½
C	-1	-1	-½	+1

Transitions:

A⇔G, C⇔T common (lower penalty)

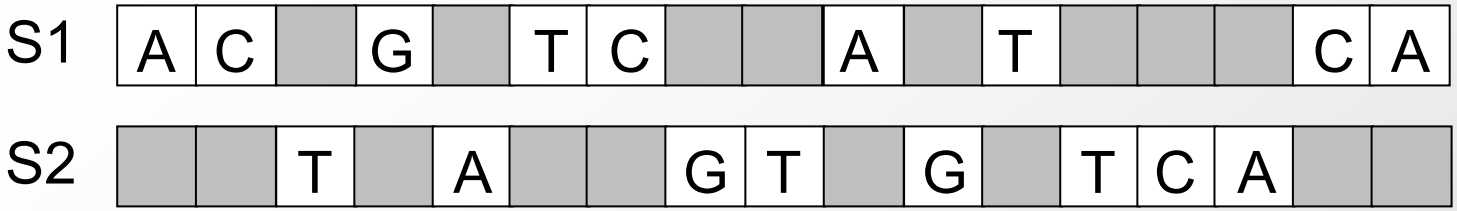
Transversions:

All other operations

Formulation 4: Varying gap cost models

1. Linear gap penalty
   - Same as before
2. Affine gap penalty
   - Big initial cost for starting or ending a gap
   - Small incremental cost for each additional character
3. General gap penalty
   - Any cost function
   - No longer computable using the same model
4. Frame-aware gap penalty
   - Multiples of 3 disrupt coding regions
5. Seek duplicated regions, rearrangements, …
   - Etc

How many alignments are there?



- Longest 'non-boring' alignment: n+m entries
  - Otherwise a gap will be aligned to a gap→condense
- Alignment is equivalent to gap placement
  - (n+m choose n) ways to choose S1 placement
  - At each position yes/no answer of placing character
  - Exponential number of possible placements
- Exponential number of sequence alignment
  - Enumerating and scoring each of them not an option
  - Need faster solution for finding best alignment

Need **polynomial** algorithm to find best alignment amongst an **exponential** number of possible alignments!

→ DP

Alignment, Dynamic Programming, Hashing, BLAST

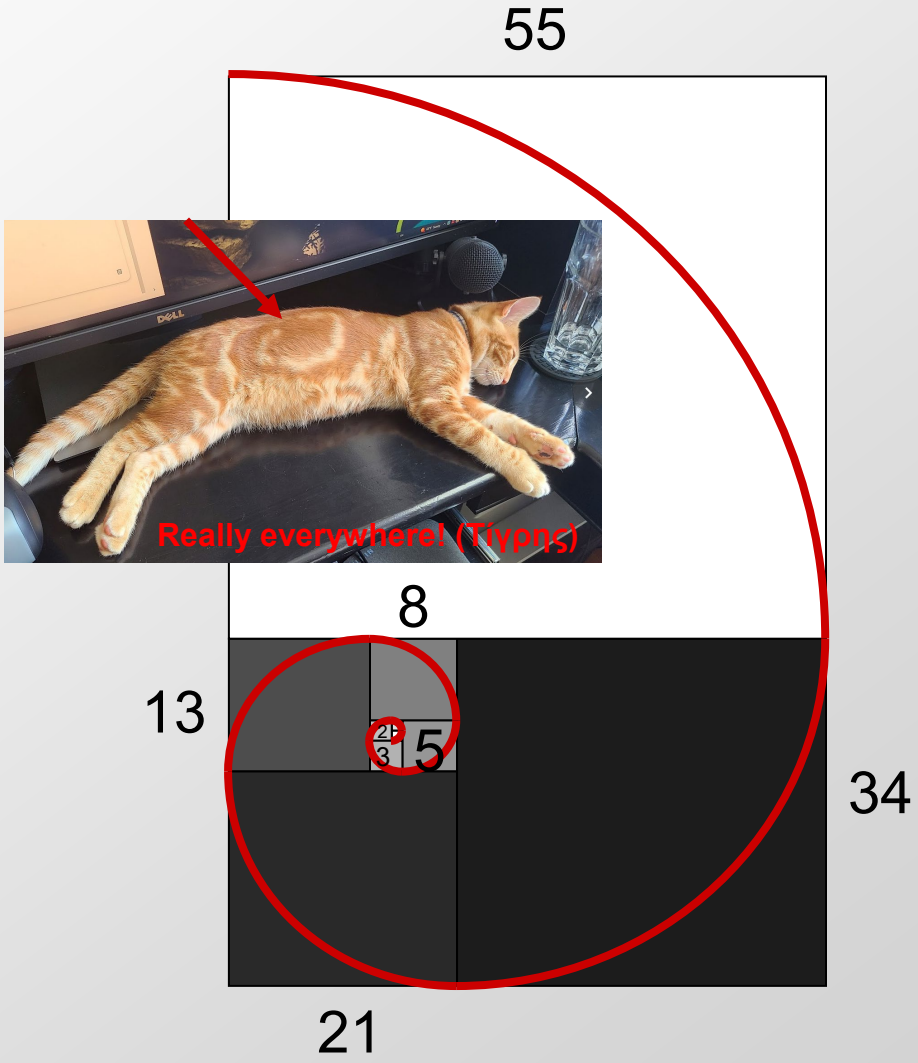
1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

A simple introduction to the principles of Dynamic Programming

Turning exponentials into polynomials

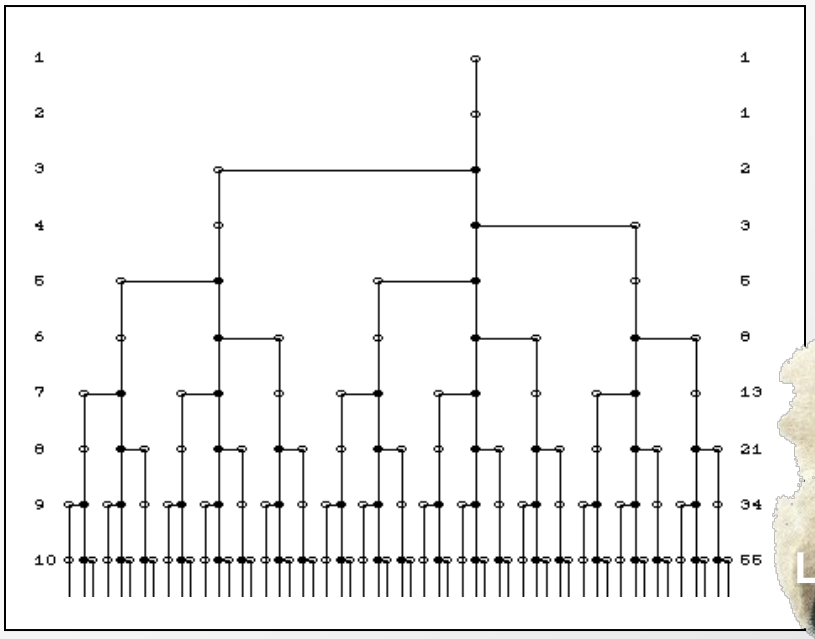
Computing Fibonacci Numbers

- Fibonacci numbers



F6=F5+F4=(F4+F3)+(F3+F2))=….=(3+2)+(2+1)=5+3=8

Fibonacci numbers are ubiquitous in nature

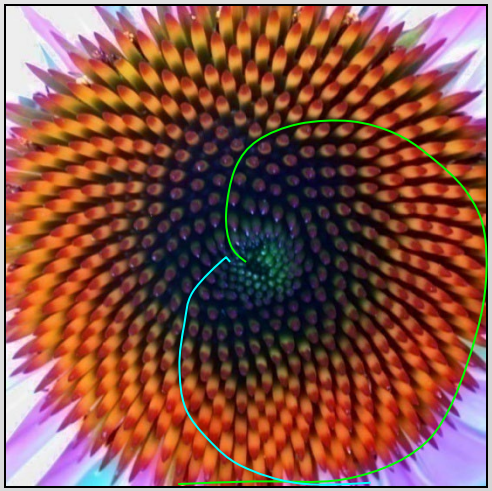


Rabbits per generation

Leaves per height

Romanesque spirals

Nautilus size



Coneflower spirals

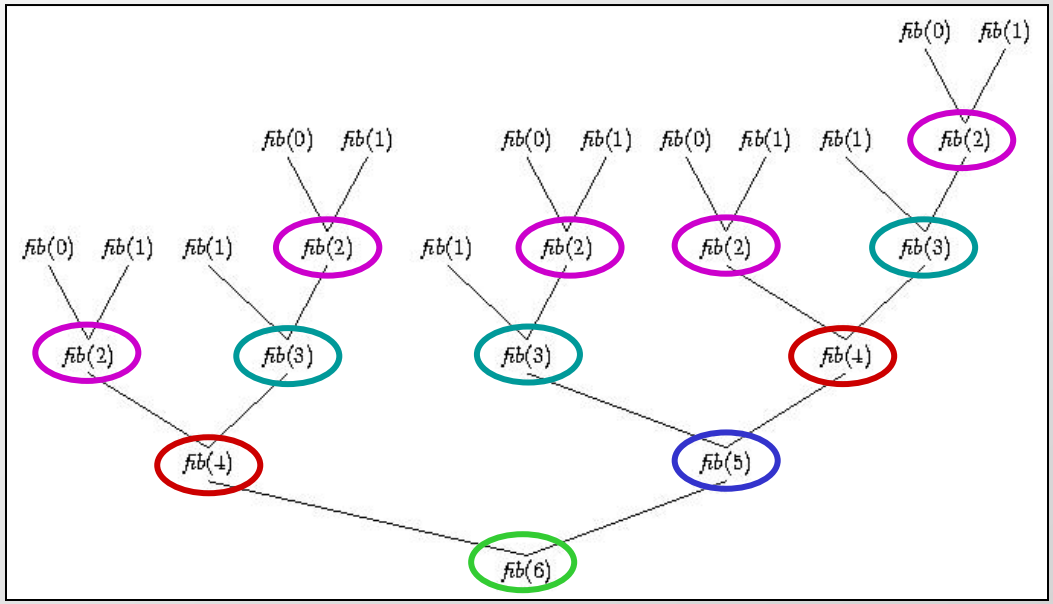
Leaf ordering

Computing Fibonacci numbers: Top down

- Fibonacci numbers are defined recursively:
  - Python code

```
def fibonacci(n):
    if n==1 or n==2: return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Goal: Compute n^{th} Fibonacci number.
  - F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2)
  - 1,1,2,3,5,8,13,21,34,55,89,144,233,377,…
- Analysis:
  - T(n) = T(n-1) + T(n-2) = (…) = $O(2^n)$



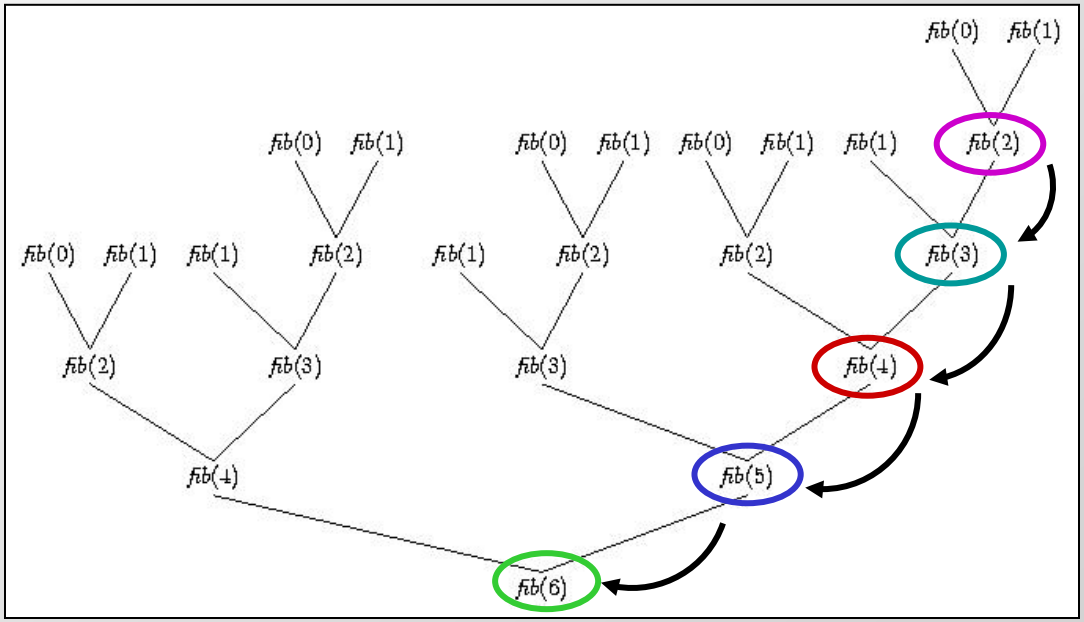
Computing Fibonacci numbers: Bottom up

- Bottom up approach
  - Python code

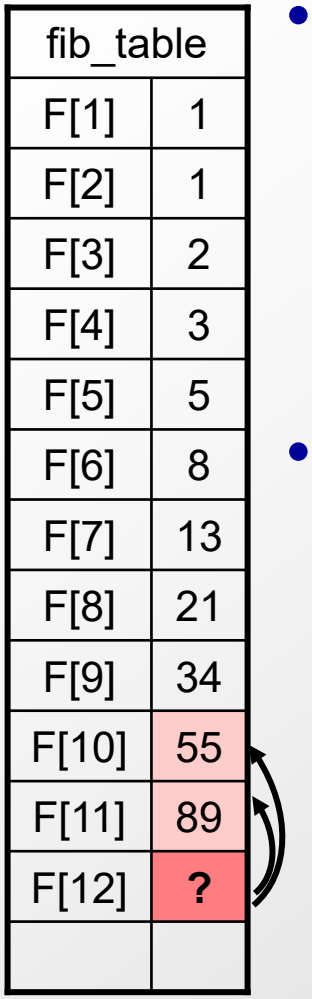
fib_table	
F[1]	1
F[2]	1
F[3]	2
F[4]	3
F[5]	5
F[6]	8
F[7]	13
F[8]	21
F[9]	34
F[10]	55
F[11]	89
F[12]	?

```
def fibonacci(n):
        fib_table[1] = 1
        fib_table[2] = 1
        for i in range(3,n+1):
          fib_table[i] = fib_table[i-1]+fib_table[i-2]
        return fib_table[n]
```

  - Analysis: T(n) = O(n)



Lessons from iterative Fibonacci algorithm



- What did the iterative solution do?
  - Reveal identical sub-problems
  - Order computation to enable result reuse
  - Systematically filled-in table of results
  - Expressed larger problems from their subparts
- Ordering of computations matters
  - Naïve top-down approach very slow
    - results of smaller problems not available
    - repeated work
  - Systematic bottom-up approach successful
    - Systematically solve each sub-problem
    - Fill-in table of sub-problem results in order.
    - Look up solutions instead of recomputing

Traversal order – Memoization vs. Dynamic Programming

Memoization

- Create a big dictionary, indexed by aligned seqs
  - When you encounter a new pair of sequences
  - If it is in the dictionary:
    - Look up the solution
  - If it is not in the dictionary
    - Compute the solution
    - Insert the solution in the dictionary
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!

Top down approach

Dynamic Programming

- Create a big table, indexed by (i,j)
  - Fill it in from the beginning all the way till the end
  - You know that you'll need every subpart
  - Guaranteed to explore entire search space
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!
- Very simple computationally!

Bottom-up approach

Dynamic Programming in Theory

- Hallmarks of Dynamic Programming
  - **Optimal substructure:** Optimal solution to problem (instance) contains optimal solutions to sub-problems
  - **Overlapping subproblems:** Limited number of distinct subproblems, repeated many many times
- Typically for optimization problems (unlike Fib example)
  - Optimal choice made locally: max(subsolution score)
  - Score is typically added through the search space
  - Traceback common, find optimal path from indiv. choices
- Middle of the road in range of difficulty
  - Easier: greedy choice possible at each step
  - DynProg: requires a traceback to find that optimal path
  - Harder: no opt. substr., e.g. subproblem dependencies

Hallmarks of optimization problems

Greedy algorithms	Dynamic Programming

1. Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

2. Overlapping subproblems

A recursive solution contains a "small" number of distinct subproblems repeated many times.

3. Greedy choice property

Locally optimal choices lead to globally optimal solution

Greedy Choice is not possible

Globally optimal solution requires trace back through many choices

Dynamic Programming in Practice

- Setting up dynamic programming
  1. Find ‘matrix’ parameterization (# dimensions, variables)
  2. Make sure sub-problem space is finite! (not exponential)
     - If not all subproblems are used, better off using memoization
     - If reuse not extensive, perhaps DynProg is not right solution!
  3. Traversal order: sub-results ready when you need them
     - Computation order matters! (bottom-up, but not always obvious)
  4. **Recursion formula: larger problems = F(subparts)**
  5. Remember choices: typically F() includes min() or max()
     - Need representation for storing pointers, is this polynomial !
- Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution

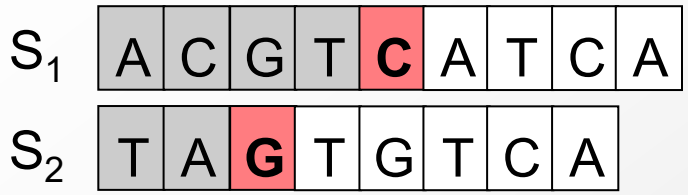
Alignment, Dynamic Programming, Hashing, BLAST

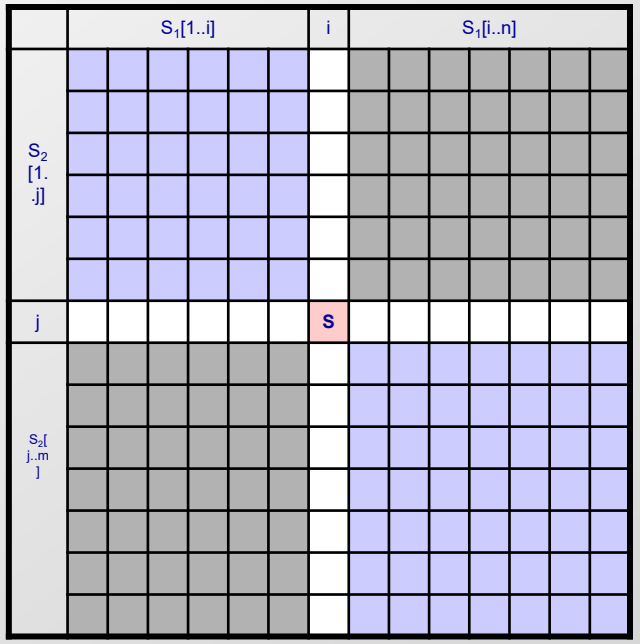
1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

How do we apply dynamic programming

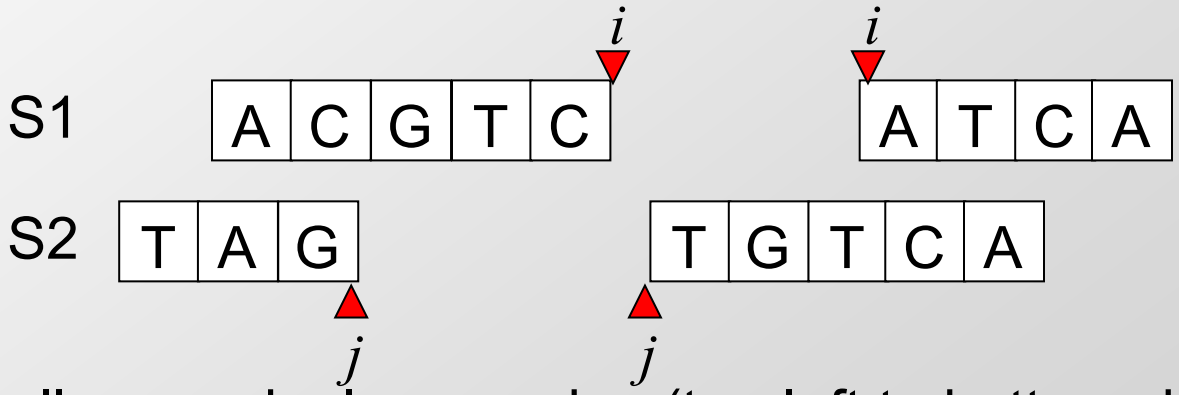
to sequence alignment ?

Key insight #1: Score is additive, smaller to larger



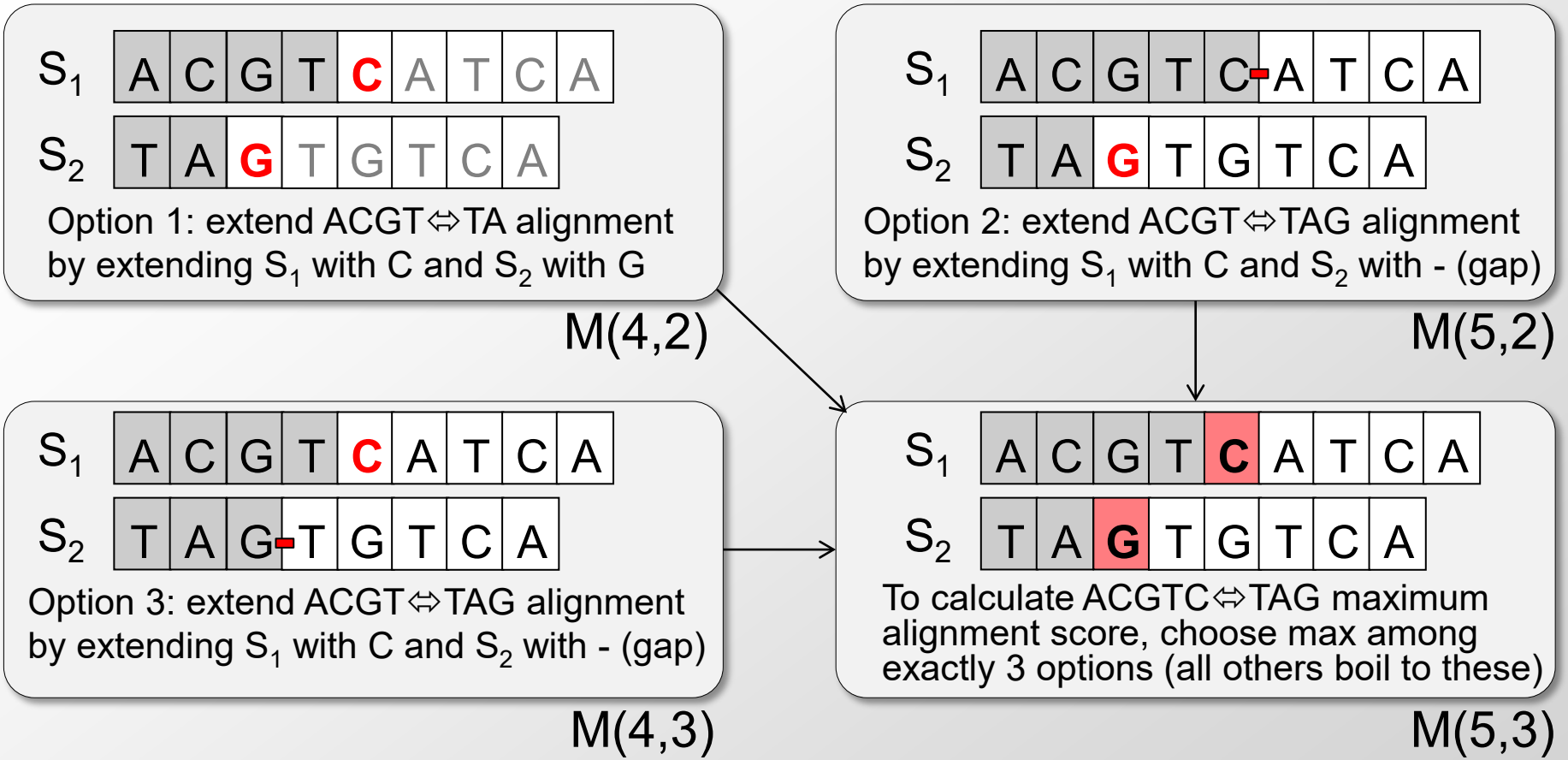


- Compute best alignment recursively
  - The best alignment going through $S_1[i] \Leftrightarrow S_2[j]$
    - Best alignment of $S_1[1..i]$ vs. $S_2[1..j]$
    - \+ Best alignment of $S_1[\ \ i..n]$ and $S_2[\ \ j..m]$
  - **Proof**: cut-and-paste argument (see 6.046)
    - If pasted sub-part was not best, then can replace with better one, and get better overall score
    - Thus, overall above solution couldn't have been best



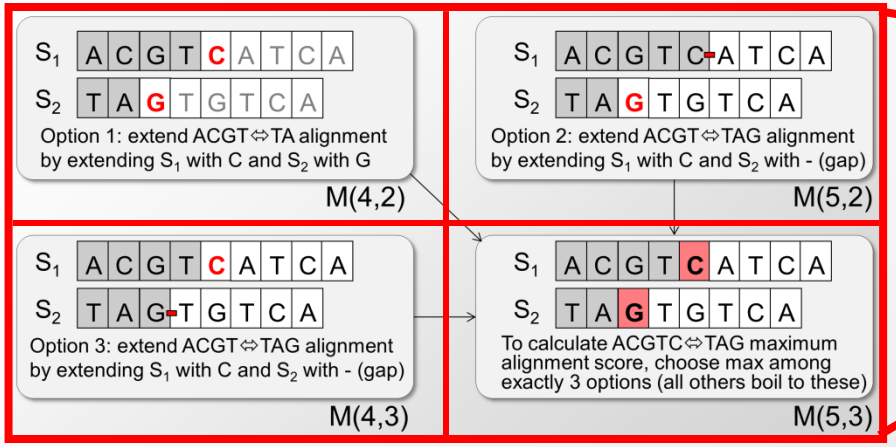
This allows a single recursion (top-left to bottom-right) instead of two recursions (middle-to-outside top-down)

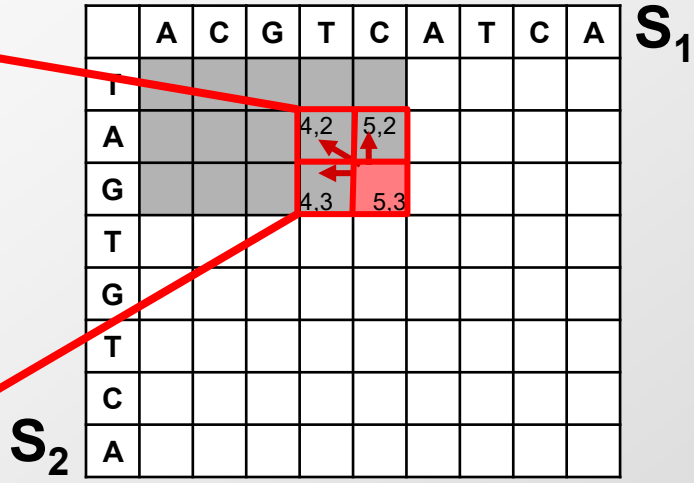
Compute optimal score based on smaller problems



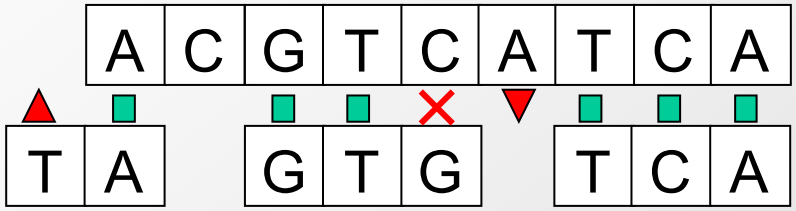
- Key idea: Calculate maximum alignment score of longer sequences based on previously-computed scores of shorter sequences
  - Assume all shorter alignments are already computed
  - Show that you can use them to compute longer alignment using them
  - By induction, can compute alignments of any length this way
- Implementation: Store all these scores in a matrix M(S_1 prefix, S_2 prefix)
  - Fill the matrix systematically from the top left to the bottom right

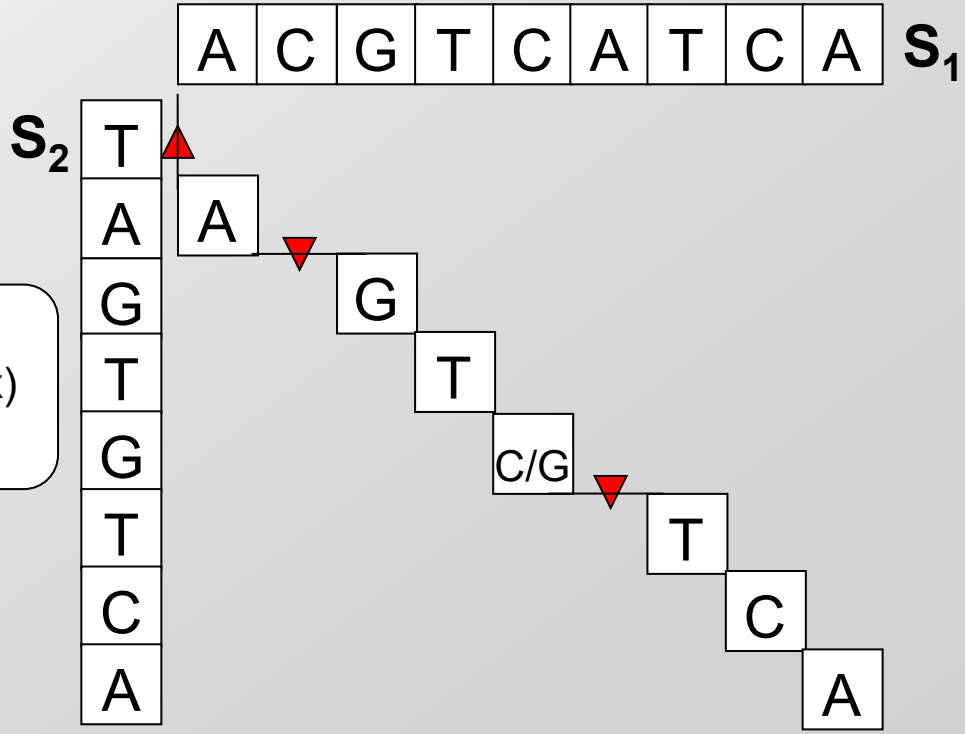
Can store all max alignment scores in a matrix M[i,j]





Path through the matrix ⇔ Alignment of S_1 and S_2

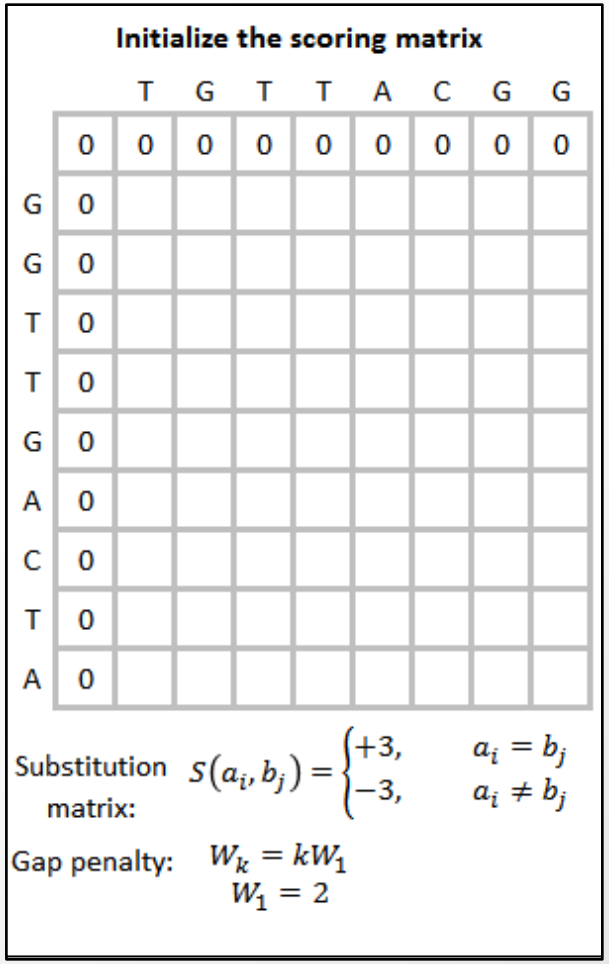


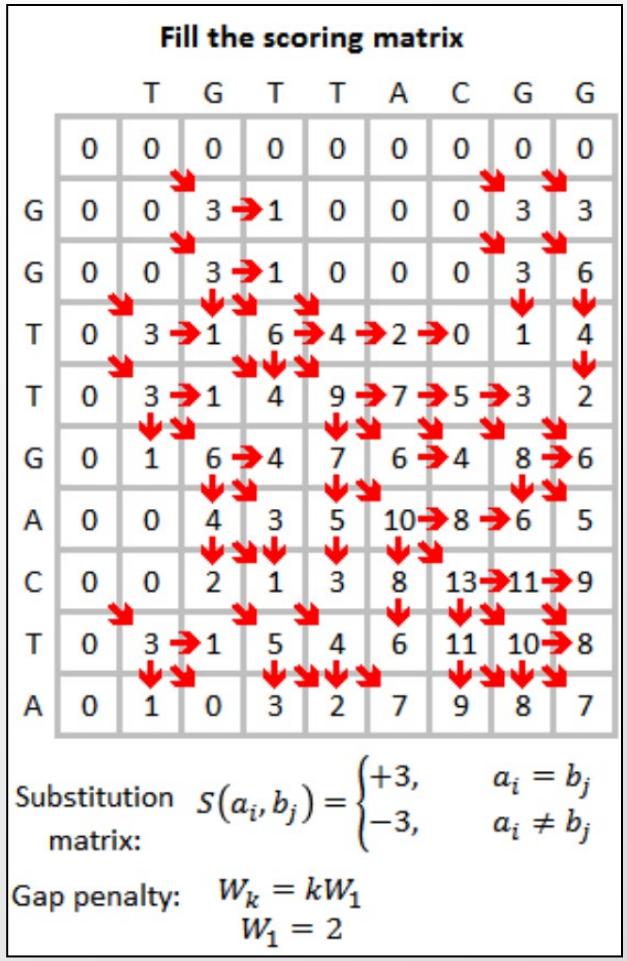


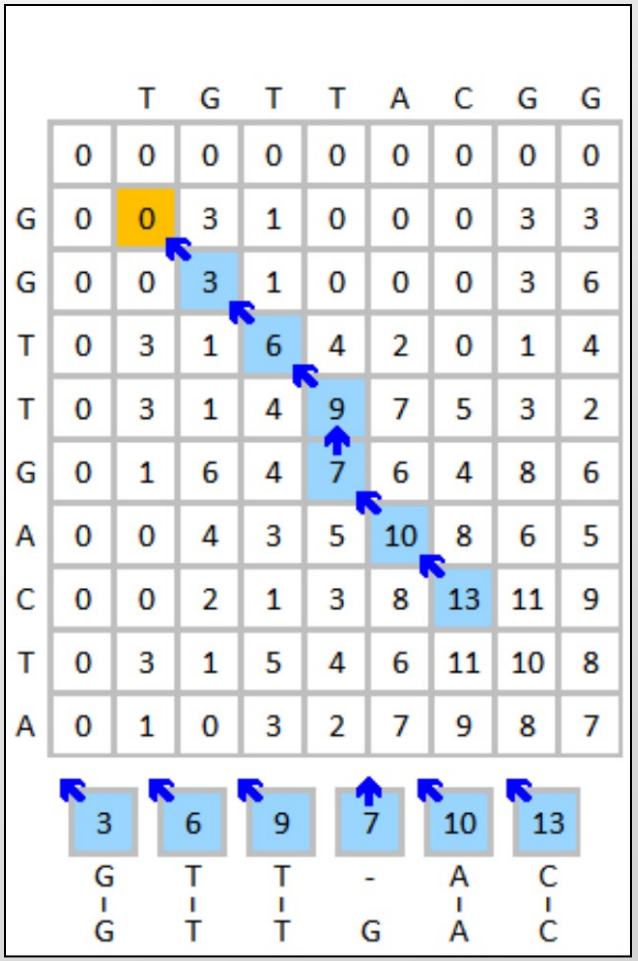
- **Each entry M[i,j]** ⇔ Best alignment score between S_1[1..i] (i^{th} prefix) vs. S_2[1..j] (j^{th} prefix)
- **Best alignment** ⇔ Best path through matrix

Goal: Find best path through the matrix

Animation: Filling in the matrix, traceback







Sequences

Sequence 1: T G T T A C G G

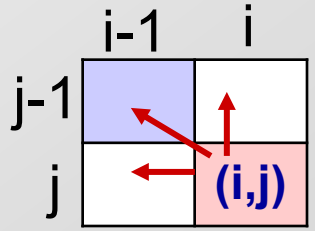
Sequence 2: G G T T G A C T A

- **Initialize the matrix**
- **Fill the matrix**
- **(remembering the max pointers)**
- **Find the maximum score**
- **Traceback max pointers to build alignment**
- **Return alignment of min length for max score**

Animation credit: Wikipedia

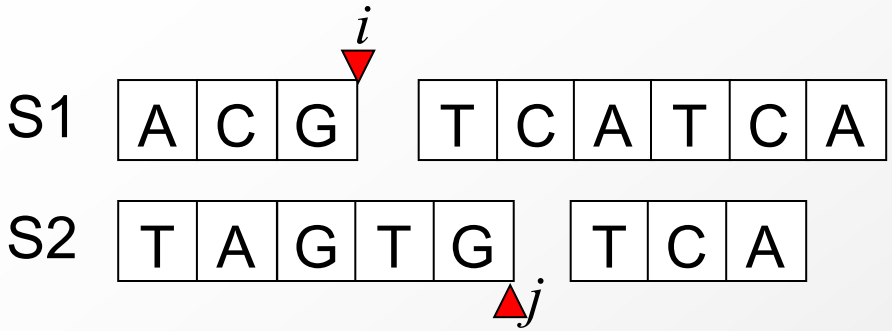
Computing alignments recursively: M[i,j]=F(smaller)

- Local update rules, only look at neighboring cells:
  - Compute next alignment based on previous alignment
  - Just like Fibonacci numbers: F[i] = F[i-1] + F[i-2]
  - Table lookup avoids repeated computation
- Computing the score of a cell from smaller neighbors
  - $M(i,j) = \max\{ M(i-1, j) - \text{gap},\ M(i-1, j-1) + \text{score},\ M(i, j-1) - \text{gap} \}$



  - Only three possibilities for extending by one nucleotide: a gap in one species, a gap in the other, a (mis)match
- Compute scores for prefixes of increasing length
  - Start with prefixes of length 1, extend by one each time, until all prefixes have been computed
  - When you reach bottom right, alignment score of $S_1[1..m]$ and $S_2[1..n]$ is alignment of full S_1 and full S_2
  - (Can then trace back to construct optimal path to it)

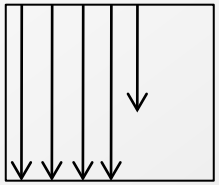
DP approach: iteratively grow best alignment soltn



- Compute all alignment scores from the bottom up
  - Define M[i,j] prefix alignment score of S_1[1..i] and S_2[1..j]
  - Fill up table recursively from smaller to bigger alignments
- Express alignment of S_1[1..i+1] and S_2[1..j+1] ➔ M[i+1,j+1]
  - One of three possibilities: (1) extend alignment from M[i,j] (2) extend from M[i-1,j], or (3) extend from M[i,j-1]
  - Only a local computation, takes O(1) time!
- Proof of correctness (cut-and-paste argument from 6.006)
  - Best alignment of S_1[1..i+1] and S_2[1..j+1] must be composed of best alignments of smaller prefix
  - Proof: otherwise could replace sub and get better overall

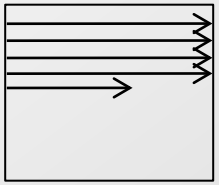
Dynamic Programming for sequence alignment

- Setting up dynamic programming
  1. Find 'matrix' parameterization
     - Prefix parameterization. Score(S_1[1..i],S_2[1..j]) ➔ M(i,j)
     - (i,j) only prefixes vs. (i,j,k,l) all substrings ➔ simpler 2-d matrix
  2. Make sure sub-problem space is finite! (not exponential)
     - It's just n^2, quadratic (which is polynomial, not exponential)
  3. Traversal order: sub-results ready when you need them



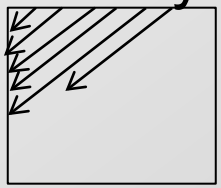
Cols

L→R



Rows

top→bot



Diags

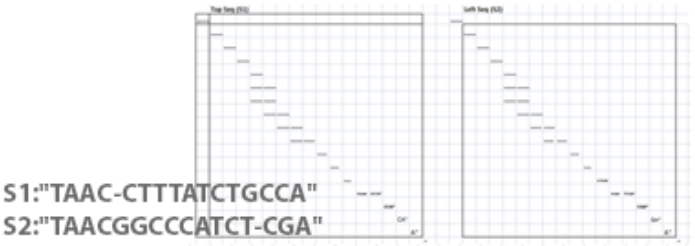
topR→botL

  4. Recursion formula: larger problems = Func(subparts)
     - Need formula for computing M[i,j] as function of previous results
     - Single increment at a time, only look at M[i-1,j], M[i,j-1], M[i-1,j-1] corresponding to 3 options: gap in S_1, gap in S_2, char in both
     - Score in each case depends on gap/match/mismatch penalties
  5. Remember choice: F() typically includes min() or max()
     - Remember which of three cells (top,left,diag) led to maximum

Genome alignment in an excel spreadsheet

	A	G	C	T	Gap
A	1.0	0.0	-1.0	-1.0	-1.0
G	0.0	1.0	-1.0	-1.0	
C	-1.0	-1.0	1.0	0.0	
T	-1.0	-1.0	0.0	1.0	

S1[1..i]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Prefixes	T	TA	TAA	TAAC	TAACC	TAACCT	TAACCTT	TAACCTTT	TAACCTTTA	TAACCTTTAT	TAACCTTTATC	TAACCTTTATCT	TAACCTTTATCTG	TAACCTTTATCTGC	TAACCTTTATCTGCC	TAACCTTTATCTGCCA



S2[1..j]	Prefixes	S2:"	S1:"	T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
1	T	T		1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
2	TA	A		-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
3	TAA	A		-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
4	TAAC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
5	TAACG	G		-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
6	TAACGG	G		-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
7	TAACGGC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
8	TAACGGCC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
9	TAACGGCCC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
10	TAACGGCCCA	A		-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
11	TAACGGCCCAT	T		1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
12	TAACGGCCCATC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
13	TAACGGCCCATCT	T		1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
14	TAACGGCCCATCTC	C		0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
15	TAACGGCCCATCTCG	G		-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
16	TAACGGCCCATCTCGA	A		-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
T	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
A	-2	0	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
A	-3	-1	1	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-4	-2	0	2	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
G	-5	-3	-1	1	3	3	2	1	0	0	-1	-2	-3	-3	-4	-5	-6
G	-6	-4	-2	0	2	2	2	1	0	0	-1	-2	-3	-2	-3	-4	-5
C	-7	-5	-3	-1	1	3	2	2	1	0	0	0	-1	-2	-1	-2	-3
C	-8	-6	-4	-2	0	2	3	2	2	1	0	1	0	-1	-1	0	-1
C	-9	-7	-5	-3	-1	1	2	3	2	1	1	1	1	0	0	0	-1
A	-10	-8	-6	-4	-2	0	1	2	2	3	2	1	0	1	0	-1	1
T	-11	-9	-7	-5	-3	-1	1	2	3	2	4	3	2	1	1	0	0
C	-12	-10	-8	-6	-4	-2	0	1	2	2	3	5	4	3	2	2	1
T	-13	-11	-9	-7	-5	-3	-1	1	2	1	3	4	6	5	4	3	2
C	-14	-12	-10	-8	-6	-4	-2	0	1	1	2	4	5	5	6	5	4
G	-15	-13	-11	-9	-7	-5	-3	-1	0	1	1	3	4	6	5	5	5
A	-16	-14	-12	-10	-8	-6	-4	-2	-1	1	0	2	3	5	5	4	6

	T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
T	\	--	--	--	--	--\	--\	--\	--	--\	--	--\	--	--	--	--
A	|	\	--\	--	--	--	--	--	--\	--	--	--	--	--	--	--\
A	|	|	\	--	--	--	--	--	--\	--	--	--	--	--	--	--\
C	|	|	|	\	--\	--	--	--	--	--	--\	--	--	--\	--\	--
G	|	|	|	|	\	--\	--\	--\	\	--	--	--	\	--	--	--
G	|	|	|	|	|	\	--\	--\	\	--\	--\	--\	\	--	--	--\
C	|	|	|	|	\	--\	\	--\	--	\	\	--	--	\	--\	--
C	|	|	|	|	|	\	--\	\	--	--\	\	--\	--	\	\	--
C	|	|	|	|	|	|	\	--\	--\	\	\	\	--	\	\	--\
A	|	|	|	|	|	|	|	\	\	--	--	|--\	\	--	|--\	\
T	|	|	|	|	|	\	\	\	|--	\	--	--\	--	\	--\	|
C	|	|	|	|	|	|	|	|	\	|	\	--	--	--\	\	--
T	|	|	|	|	|	|	\	\	|--\	\	|	\	--	--	--	--
C	|	|	|	|	|	|	|	|	\	|	\	|	\	\	--\	--
G	|	|	|	|	|	|	|	|	\	|	|	|	\	|--	\	\
A	|	|	|	|	|	|	|	|	\	|--\	|	|	\	|	|--\	\

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

https://www.dropbox.com/s/ksh4qfl5eb182p6/Lecture02_DP%20Alignment%20In%20Excel.xlsx?dl=0

S1 : "TAAAGGGTTAAAGGG-----G"

S2 : "C-----CCCAAAGGGGGCCCC"

	A	G	C	T	Gap
A	1.0	0.0	-1.0	-1.0	-1.0
G	0.0	1.0	-1.0	-1.0	
C	-1.0	-1.0	1.0	0.0	
T	-1.0	-1.0	0.0	1.0	

S1[1..i]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Prefixes	T	TA	TAA	TAAA	TAAAG	TAAAGG	TAAAGGG	TAAAGGGT	TAAAGGGTT	TAAAGGGTTA	TAAAGGGTTAA	TAAAGGGTTAAA	TAAAGGGTTAAAG	TAAAGGGTTAAAGG	TAAAGGGTTAAAGGG	TAAAGGGTTAAAGGGG

S1:"	T	A	A	A	G	G	G	T	T	A	A	A	G	G	G	G

S2[1..j]	Prefixes	S2:"
1	C	C
2	CC	C
3	CCC	C
4	CCCC	C
5	CCCCA	A
6	CCCCAA	A
7	CCCCAAA	A
8	CCCCAAAG	G
9	CCCCAAAGG	G
10	CCCCAAAGGG	G
11	CCCCAAAGGGG	G
12	CCCCAAAGGGGG	G
13	CCCCAAAGGGGGC	C
14	CCCCAAAGGGGGCC	C
15	CCCCAAAGGGGGCCC	C
16	CCCCAAAGGGGGCCCC	C

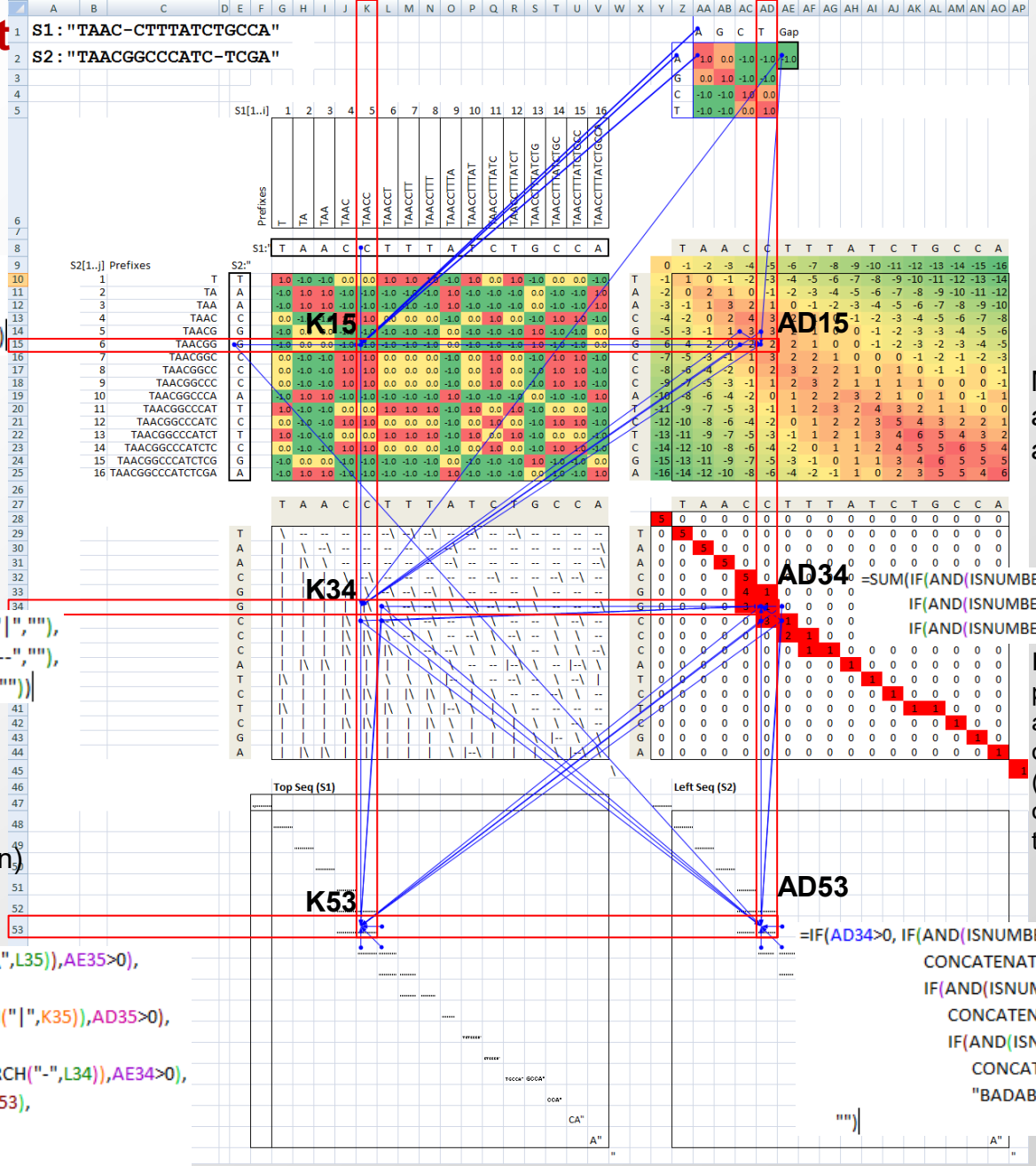
		T	A	A	A	G	G	G	T	T	A	A	A	G	G	G	G
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
C	-1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15
C	-2	-1	-1	-2	-3	-4	-5	-6	-6	-7	-8	-9	-10	-11	-12	-13	-14
C	-3	-2	-2	-2	-3	-4	-5	-6	-6	-6	-7	-8	-9	-10	-11	-12	-13
C	-4	-3	-3	-3	-3	-4	-5	-6	-6	-6	-7	-8	-9	-10	-11	-12	-13
A	-5	-4	-2	-2	-2	-3	-4	-5	-6	-7	-5	-6	-7	-8	-9	-10	-11
A	-6	-5	-3	-1	-1	-2	-3	-4	-5	-6	-6	-4	-5	-6	-7	-8	-9
A	-7	-6	-4	-2	0	-1	-2	-3	-4	-5	-5	-5	-3	-4	-5	-6	-7
G	-8	-7	-5	-3	-1	1	0	-1	-2	-3	-4	-5	-4	-2	-3	-4	-5
G	-9	-8	-6	-4	-2	0	2	1	0	-1	-2	-3	-4	-3	-1	-2	-3
G	-10	-9	-7	-5	-3	-1	1	3	2	1	0	-1	-2	-3	-2	0	-1
G	-11	-10	-8	-6	-4	-2	0	2	2	1	1	0	-1	-1	-2	-1	1
G	-12	-11	-9	-7	-5	-3	-1	1	1	1	1	1	0	0	0	-1	0
C	-13	-12	-10	-8	-6	-4	-2	0	1	1	0	0	0	-1	-1	-1	-1
C	-14	-13	-11	-9	-7	-5	-3	-1	0	1	0	-1	-1	-1	-2	-2	-2
C	-15	-14	-12	-10	-8	-6	-4	-2	-1	0	0	-1	-2	-2	-2	-3	-3
C	-16	-15	-13	-11	-9	-7	-5	-3	-2	-1	-1	-1	-2	-3	-3	-3	-4

	T	A	A	A	G	G	G	T	T	A	A	A	G	G	G	G
C	\	--	--	--	--	--	--	--\	--\	--	--	--	--	--	--	--
C	|	\	--\	--\	--\	--\	--\	\	--\	--	--	--	--	--	--	--
C	|	|	\	--\	--\	--\	--\	\	\	--	--	--	--	--	--	--
C	|	|	|	\	--\	--\	--\	\	\	--\	--\	--\	--\	--\	--\	--\
A	|	\	\	\	--\	--\	--\	--	|--\	\	--\	--\	--	--	--	--
A	|	|	\	\	--\	--\	--\	--	--	|	\	--\	--	--	--	--
A	|	|	|	\	--\	--\	--\	--	--	\	|	\	--	--	--	--
G	|	|	|	|	\	--\	--\	--	--	--	--\	|	\	--\	--\	--\
G	|	|	|	|	|	\	--\	--	--	--	--	--	|	\	--\	--\
G	|	|	|	|	|	|	\	--	--	--	--	--	--\	|	\	--\
G	|	|	|	|	|	|	|	\	--\	\	--\	--\	\	--\	|	\
G	|	|	|	|	|	|	|	|	\	\	\	--\	\	\	--\	|
C	|	|	|	|	|	|	|	\	\	|--\	|	\	|--\	|	\	|
C	|	|	|	|	|	|	|	|	\	--\	|--\	|	\	|--\	|	|
C	|	|	|	|	|	|	|	|	|	\	--\	|--\	|	\	|--\	|
C	|	|	|	|	|	|	|	|	|	|	\	--\	|--\	|	\	|--\

		T	A	A	A	G	G	G	T	T	A	A	A	G	G	G	G
	55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	25	55	25	20	15	10	5	0	0	0	0	0	0	0	0	0	0
C	25	25	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0
C	25	25	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
C	0	25	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
A	0	0	25	0	0	0	0	0	0	0	5	0	0	0	0	0	0
A	0	0	0	25	0	0	0	0	0	0	0	5	0	0	0	0	0
A	0	0	0	0	25	0	0	0	0	0	0	0	5	0	0	0	0
G	0	0	0	0	0	25	0	0	0	0	0	0	1	4	0	0	0
G	0	0	0	0	0	0	25	0	0	0	0	0	0	1	3	0	0
G	0	0	0	0	0	0	0	25	25	25	15	10	6	0	1	2	0
G	0	0	0	0	0	0	0	0	0	0	10	5	4	6	0	1	1
G	0	0	0	0	0	0	0	0	0	0	0	5	1	4	6	0	1
C	0	0	0	0	0	0	0	0	0	0	0	0	4	1	3	3	1
C	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	2	1
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	1
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

https://www.dropbox.com/s/5g3uzr62wvkagov/DP_InExcel4_coolBifurcation2.xlsx?dl=0

Genome alignment in an excel spreadsheet



K15

```
=INDEX($AA$2:$AD$5,
        MATCH(K$8,$Z$2:$Z$5,0),
        MATCH($E15,$AA$1:$AD$1,0))
```

Local score of matching characters $S_1[i]$ and $S_2[j]$

AD15

```
=MAX(AD14+$AE$2,
     AC15+$AE$2,
     AC14+K15)
```

Max alignment score of aligning prefix $S_1[1..i]$ and prefix $S_2[1..j]$

K34

```
=CONCATENATE(IF(AD15=AD14+$AE$2,"|",""),
             IF(AD15=AC15+$AE$2,"--",""),
             IF(AD15=AC14+K15,"\",""))
```

Is the max alignment score coming from the top ("|"), from the left ("--") or from the diagonal up ("\") (show all of them, cuz we can)

AD34

```
=SUM(IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),AD35,0),
     IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),AE35,0),
     IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),AE34,0))
```

Is the [i,j] part of an optimal path? (i.e. are chars $S_1[i]$ and $S_2[j]$ aligned to each other in an optimal path) (also count number of optimal paths/alignment through [i,j], cuz we can)

K53

```
=IF(AD34>0, IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),
               CONCATENATE(K$8,L54),
               IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),
                  CONCATENATE("-",K54),
                  IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),
                     CONCATENATE(K$8,L53),
                     "BADABOOM!"))),
   "")
```

Construct the optimal alignment for sequence S_1 by adding in characters or gaps to increasingly large suffixes (and arbitrarily choose one path when multiple using nested if's)

AD53

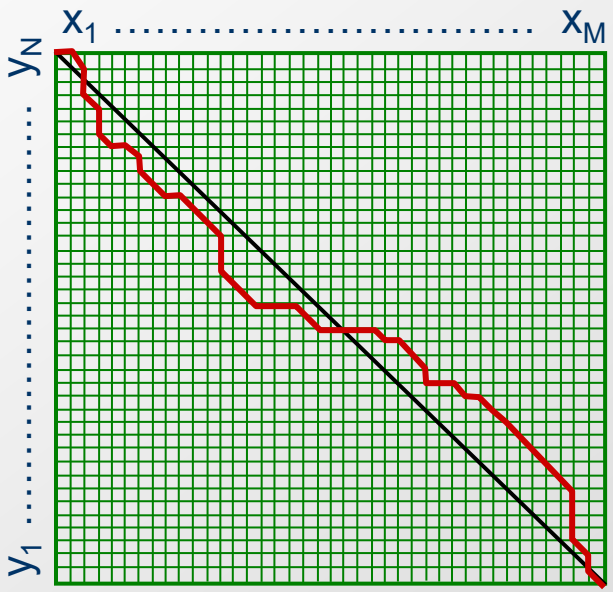
```
=IF(AD34>0, IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),
               CONCATENATE($E15,AE54),
               IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),
                  CONCATENATE($E15,AD54),
                  IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),
                     CONCATENATE("-",AE53),
                     "BADABOOM!"))),
   "")
```

https://www.dropbox.com/s/ksh4cf5eb182c6/Lecture02_DP%20Alignment%20In%20Excel.xlsx?dl=0

Construct the optimal alignment for sequence S_2 similarly to S_1

What is missing? (5) Returning the actual path!

- We know how to compute the best score
  - Simply the number at the bottom right entry
- But we need to remember where it came from
  - Pointer to the choice we made at each step
- Retrace path through the matrix
  - Need to remember all the pointers



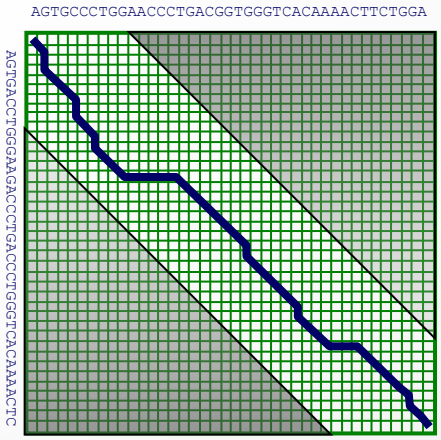
Time needed: O(m*n)

Space needed: O(m*n)

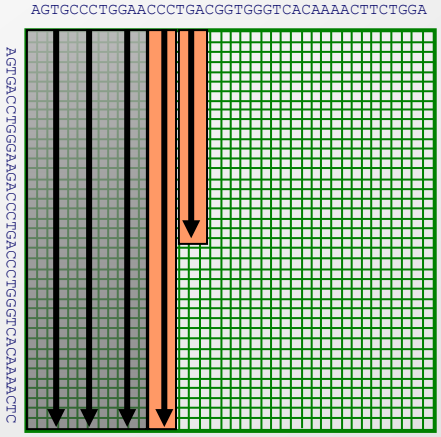
Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

Algorithmic variations (save time and/or space)



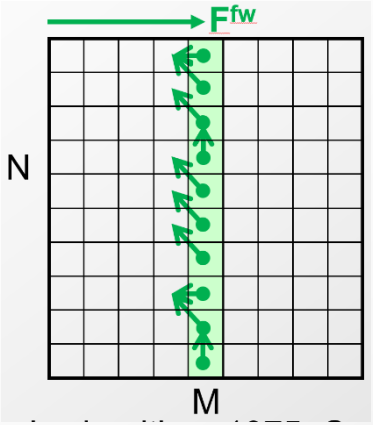
- Save time: Bounded-space computation
  - Space: O(k*m)
  - Time: O(k*m), where k = radius explored
  - Heuristic
    - Not guaranteed optimal answer
    - Works very well in practice
  - Practical interest



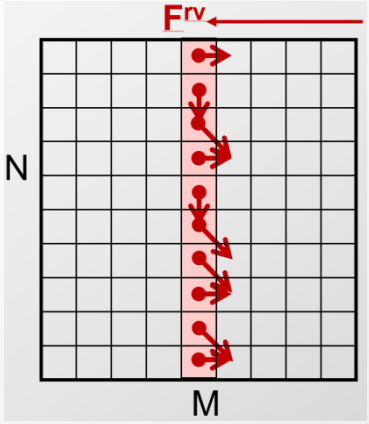
- Save space: Linear-space computation
  - Save only one col / row / diag at a time
  - Computes optimal score easily
  - Theoretical interest
    - Effective running time slower
    - Optimal answer guaranteed
  - Recursive call modification allows traceback

Finding optimal path using only linear space

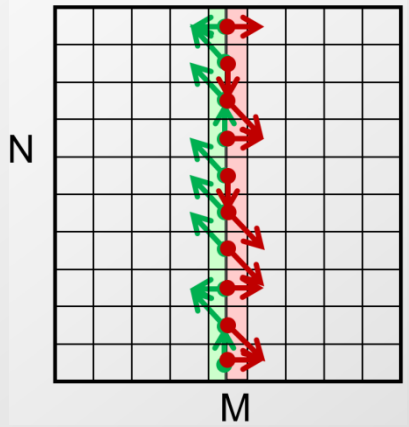
Incoming scores



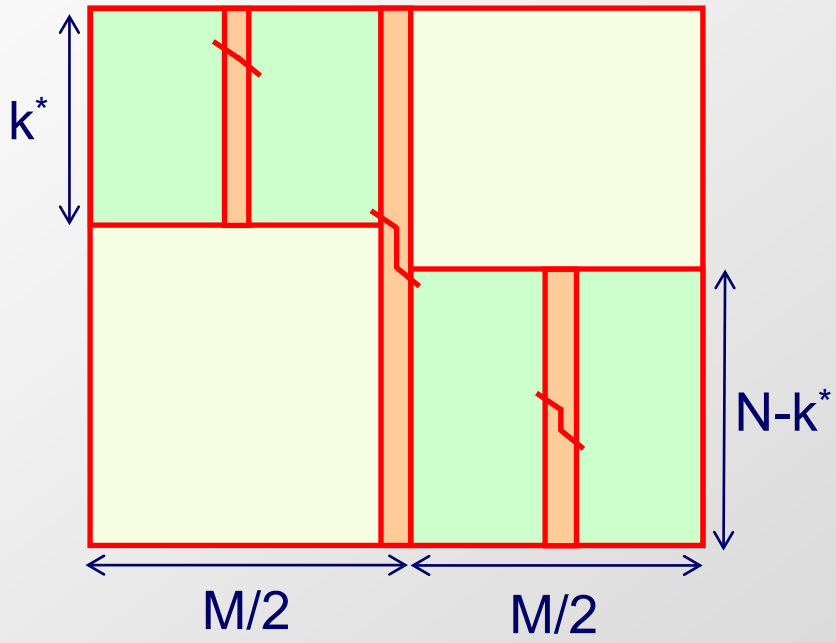
Outgoing scores

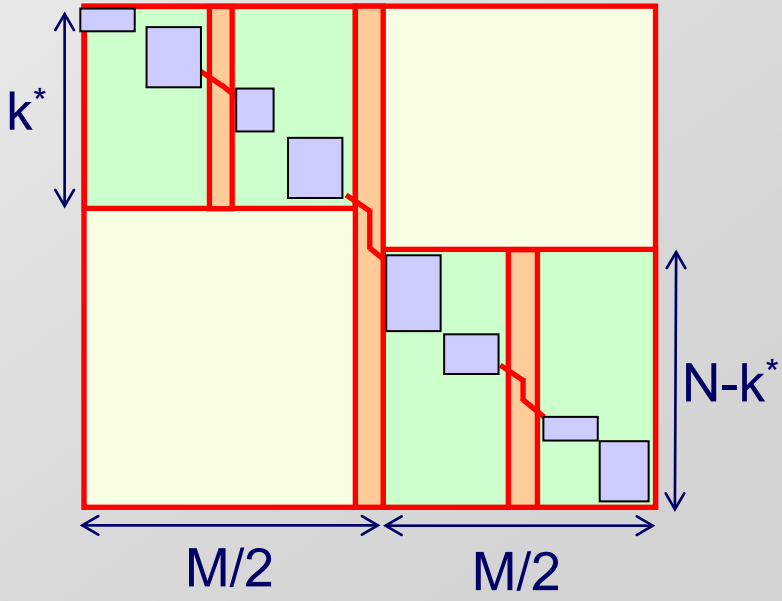


Sum the two → best transition



$F(M/2, k)$ $F^r(M/2, N-k)$ $\text{Max } F(M/2, k) + F^r(M/2, N-k)$



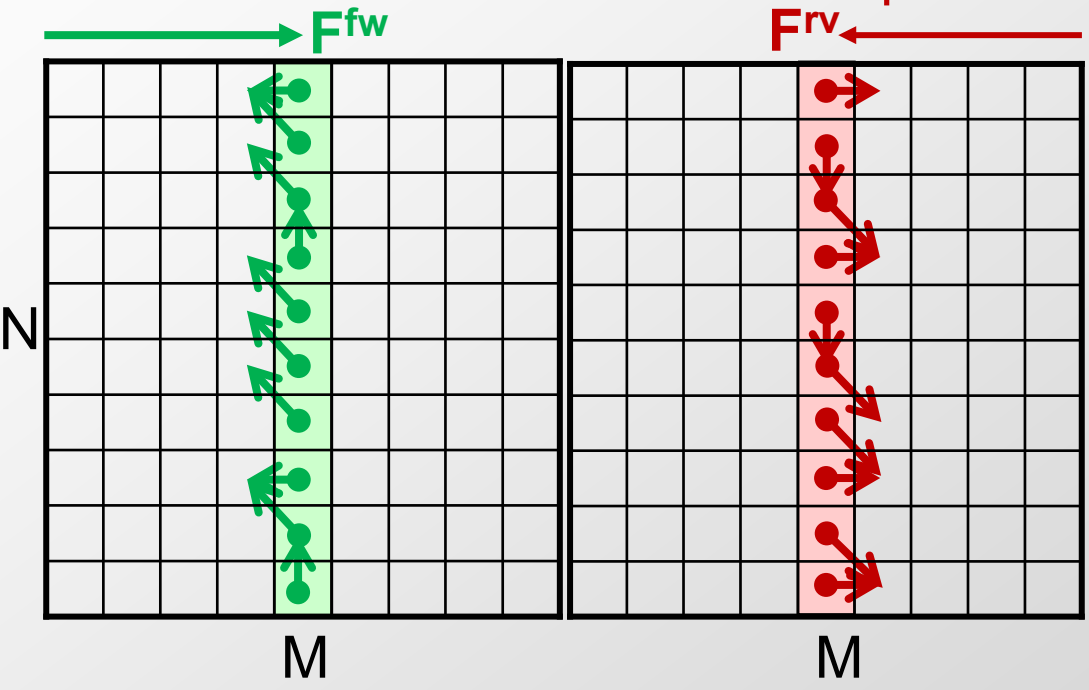


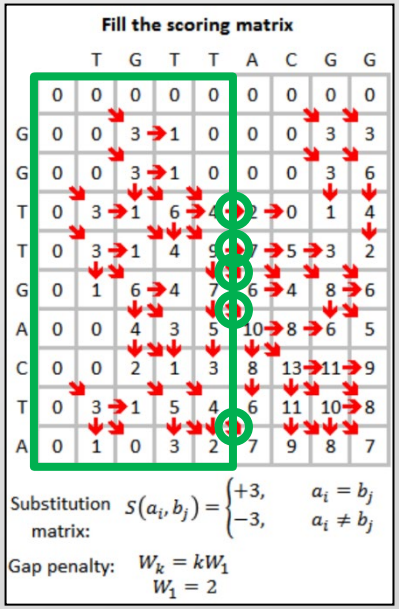
Iterate procedure in corner quadrants

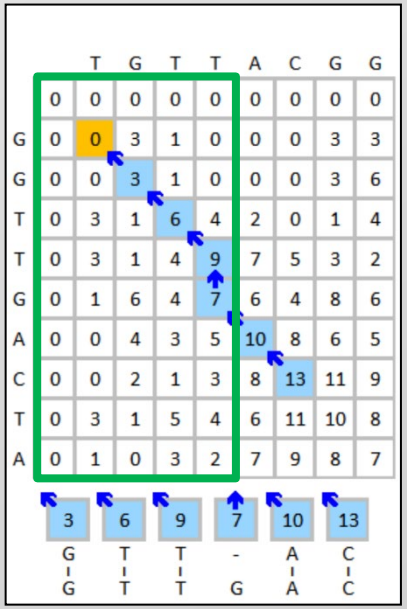
Total cost $MN(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots)\leq 2MN$

Finding the best back-pointer for current column

- Forward: Now, using 2 columns of space, we can compute for k = 1…N, $F^{fw}(M/2, k)$
  PLUS the *back*-pointers
- Reverse: Also, using 2 columns of space, we can compute backwards the suffix alignment for k = 1…N, $F^{rv}(M/2, N-k)$
  PLUS the *forward* pointers



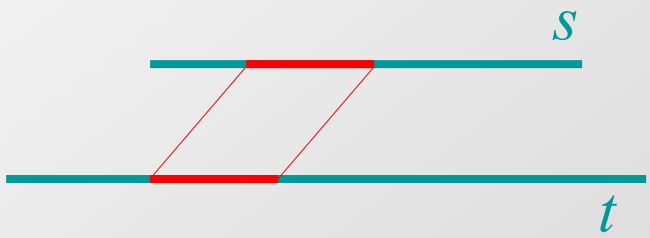




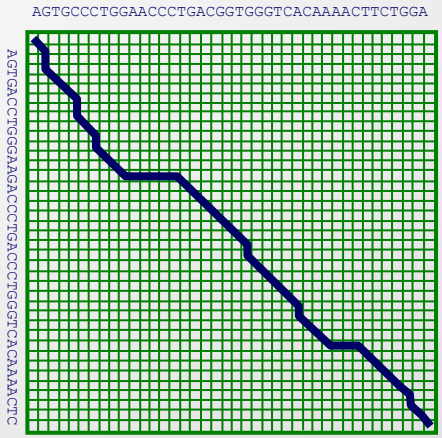
Hirschberg's algorithm, 1975, Communications of ACM 18(6):341-343

Intro to Local Alignments

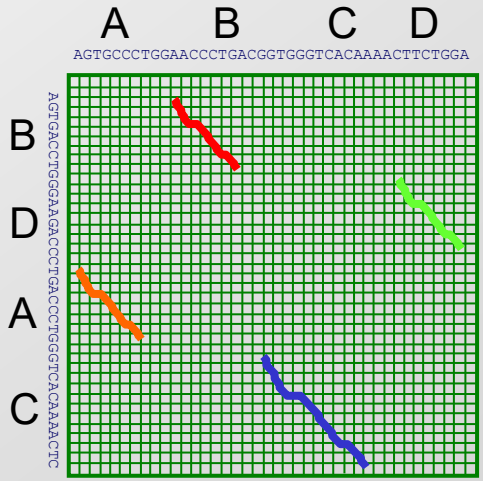
- Statement of the problem
  - A *local alignment* of strings *s* and *t* is an alignment of a substring of *s* with a substring of *t*



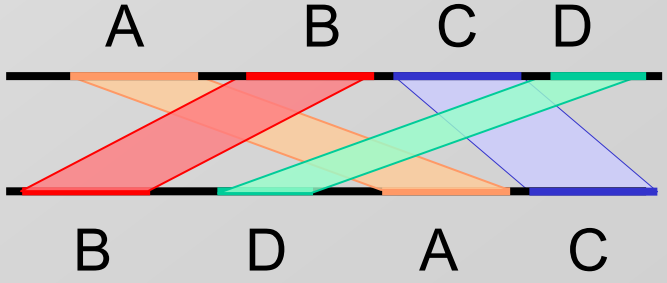
- Why local alignments?
  - Small domains of a gene may be only conserved portions
  - Looking for a small gene in a large chromosome (search)
  - Large segments often undergo rearrangements



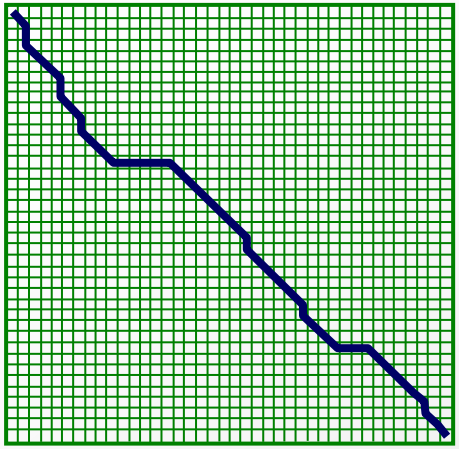
Global alignment

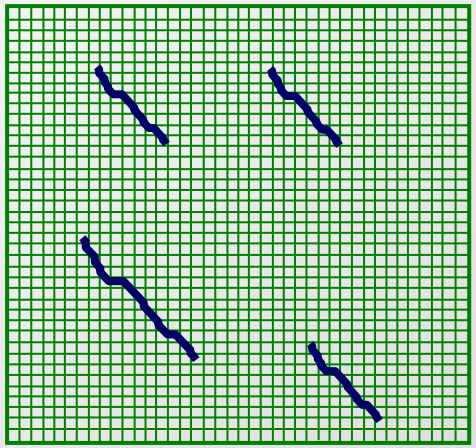


Local alignment



Global Alignment vs. Local alignment





Needleman-Wunsch algorithm

Initialization: $F(0, 0) = 0$

Iteration:

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

Termination: Bottom right

Smith-Waterman algorithm

Initialization: $\mathbf{F(0, j) = F(i, 0)} = 0$

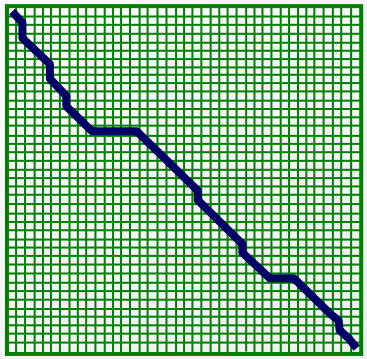
Iteration:

$$F(i, j) = \max \begin{cases} \mathbf{0} \\ F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + s(x_i, y_j) \end{cases}$$

Termination: **Anywhere**

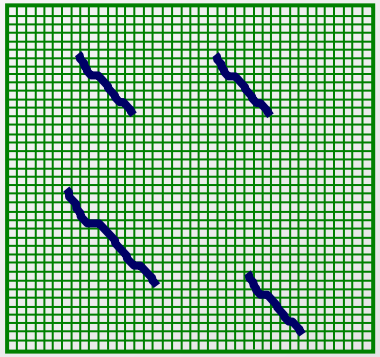
More variations on the theme: semi-global alignment

- Sequence alignment variations



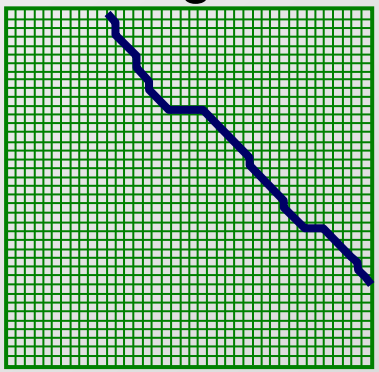
Global

Complete alignment



Local

Stretches of similarity



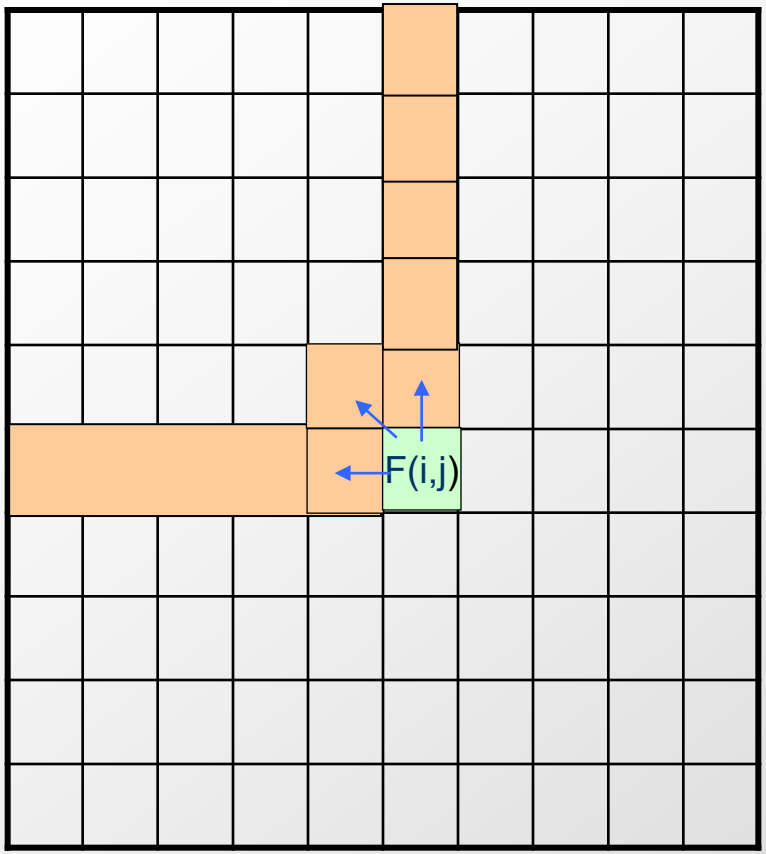
Semi-global

No end-gap penalty

	Global	Local	Semi-global
Initialization	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

Sequence alignment with generalized gap penalties

- **Implementing a generalized gap penalty function F(gap_length)**



Initialization: same

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ \max_{k=0\ldots i-1} F(k,j) - \gamma(i-k) \\ \max_{k=0\ldots j-1} F(i,k) - \gamma(j-k) \end{cases}$$

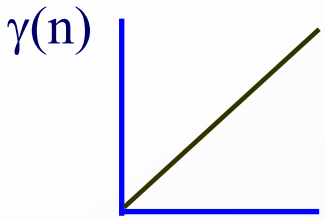
Termination: same

Running Time: $O(N^2M)$ (cubic)

Space: $O(NM)$

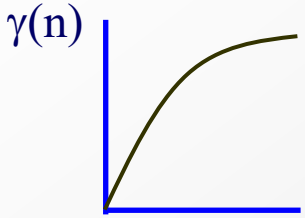
Do we have to be so general?

Algorithmic trade-offs of varying gap penalty functions



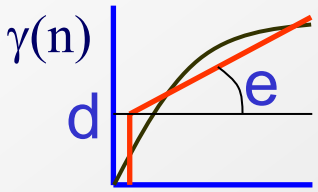
Linear gap penalty: $w(k) = k*p$

- State: Current index tells if in a gap or not
- Achievable using quadratic algorithm (even w/ linear space)



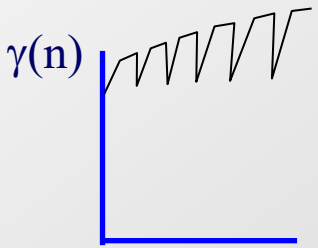
Quadratic: $w(k) = p+q*k+rk^2$.

- State: needs to encode the length of the gap, which can be O(n)
- To encode it we need O(log n) bits of information. Not feasible



Affine gap penalty: $w(k) = p + q*k$, where $q<p$

- State: add binary value for each sequence: starting a gap or not
- Implementation: add second matrix for already-in-gap (recitation)



Length (mod 3) gap penalty for protein-coding regions

- Gaps of length divisible by 3 are penalized less: conserve frame
- This is feasible, but requires more possible states
- Possible states are: starting, mod 3=1, mod 3=2, mod 3=0

Alignment, Dynamic Programming, Hashing, BLAST

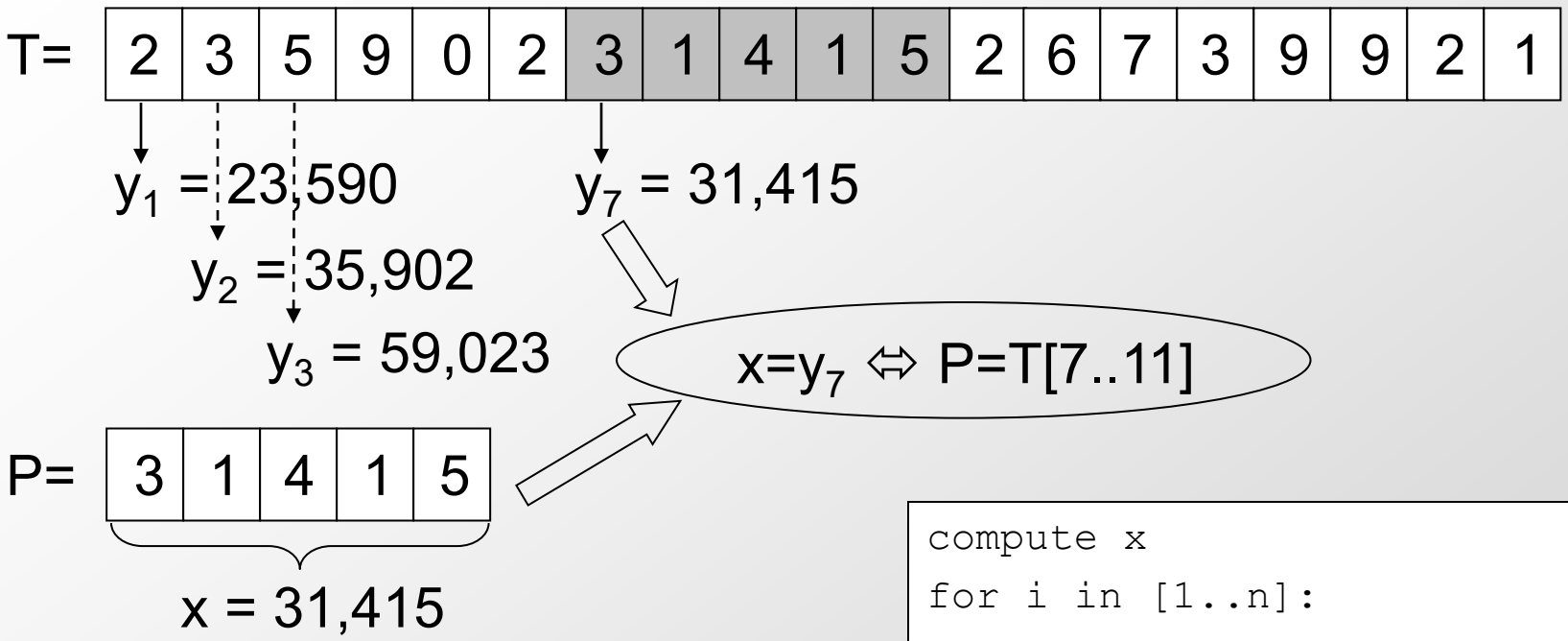
1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

Linear-time string matching

- When looking for exact matches of a pattern (no gaps)

- Karp-Rabin algorithm (probabilistic linear time):
  - Interpret String numerically
  - Start with ‘broken’ version of the algorithm
  - Progressively fix it to make it work
- Deterministicc linear-time solutions exist (not this term):
  - Z-algorithm / fundamental pre-processing, Gusfield
  - Boyer-Moore and Knuth-Morris-Pratt algorithms are earliest instantiations, similar in spirit
  - Suffix trees: beautiful algorithms, many different variations and applications, limited use in CompBio
  - Suffix arrays: practical variation, Gene Myers

Karp-Rabin algorithm

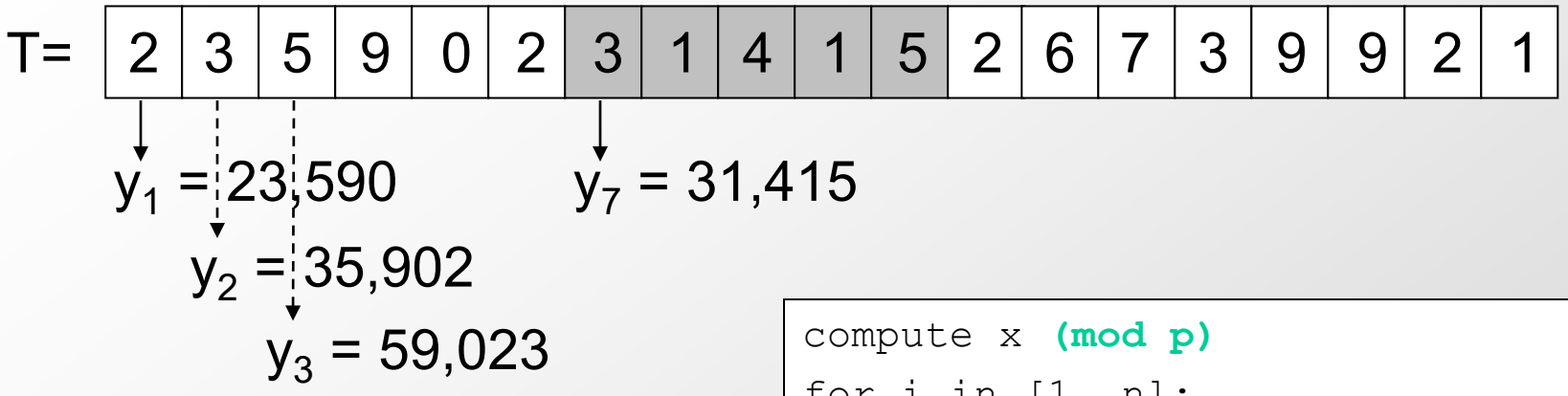



```
compute x
for i in [1..n]:
   compute yi
   if x == yi:
        print "match at S[i]"
```

(this does not actually work)

- Key idea:
  - Interpret strings as numbers: fast comparison

Karp-Rabin algorithm




```
compute x (mod p)
for i in [1..n]:
   compute yi (mod p) (using yi-1)
   if x == yi:
      if P==S[i..]:
         print "match at S[i]"
      else:
         (spurious hit)
```

(this actually works)

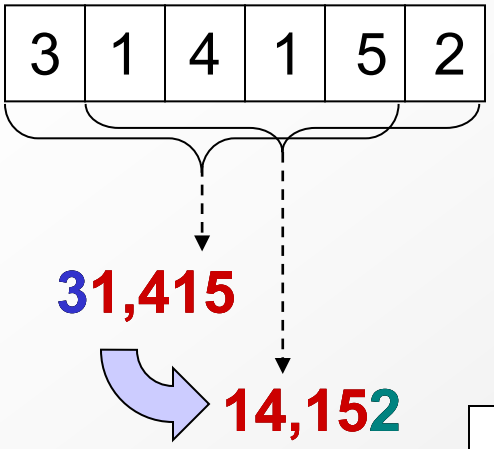
- Key idea:
  - Interpret strings as numbers: fast comparison
- To make it work:

  (a) Compute next number based on previous one → O(1)

  (b) Hashing (mod p) → keep the numbers small → O(1)

  (c) Deal with spurious hits due to hashing collisions

(a) Computing t_{s+1} based on t_s in constant time



left shift

new low-order digit

old high-order bit

14,152 = (31,415 - **3** * 10,000) * **10** + **2**

14,152 =? function (31,415)

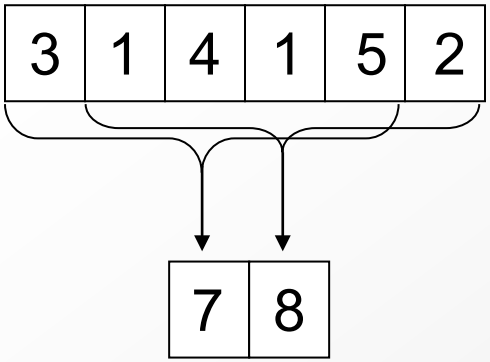
- Middle digits of the number are already computed
  Shift them to the left ←
- Remove the high-order bit
- Add the low-order bit

- General case:

$$t_s = T[s+1]2^{m-1} + T[s+2]2^{m-2} + \ldots + T[s+m]2^0$$

$$t_{s+1} = T[s+2]2^{m-1} + T[s+3]2^{m-2} + \ldots + T[s+m+1]2^0$$

(b) Dealing with long numbers in constant time



old high-order bit — left shift — new low-order digit

14,152 = (**31,415** - 3 * **10,000**) * 10 + 2 **(mod 13)**

= (**7**-3***3**)*10+2 **(mod 13)**

= **8** **(mod 13)**

Problem:

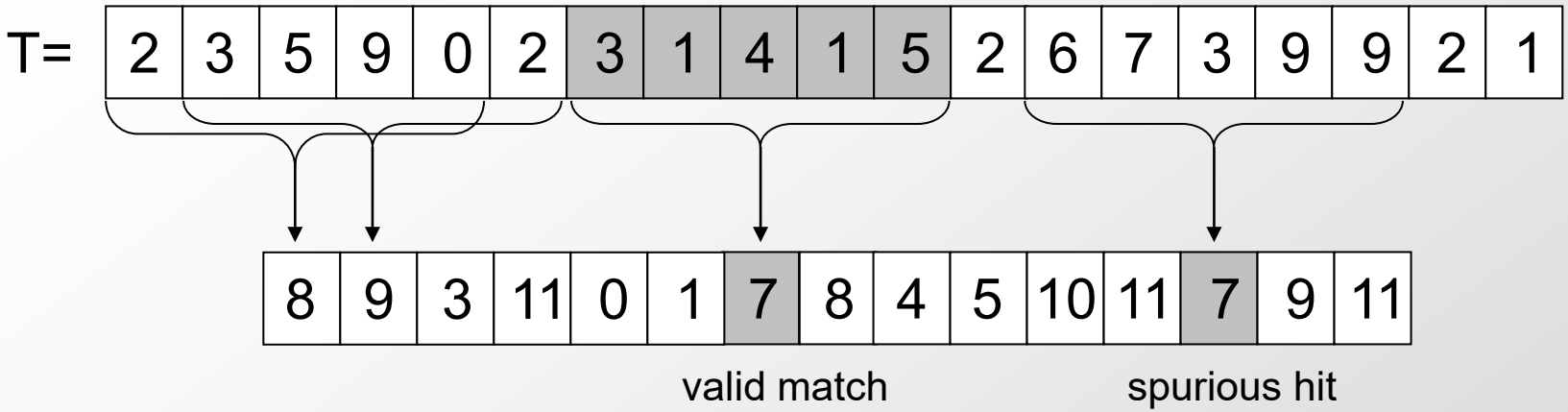
- To get O(n) time, need to perform each operation in O(1) time
- But if arguments are m-bit long (2^m range), can take O(m) time
- Need to reduce number range to something more manageable

Solution:

- Hashing: Mapping keys ***k*** from large universe U (of strings/numbers) into the '**hash**' of each key ***h(k)***, in smaller space [1..m]
- Many hash functions possible, w/ theoretical & practical properties:
  - Reproducibility: x=y➔h(x)=h(y) (hash of x always the same)
  - Uniform output distrib: x≠y➔P(h(x)=h(y))=1/m, for any input dist

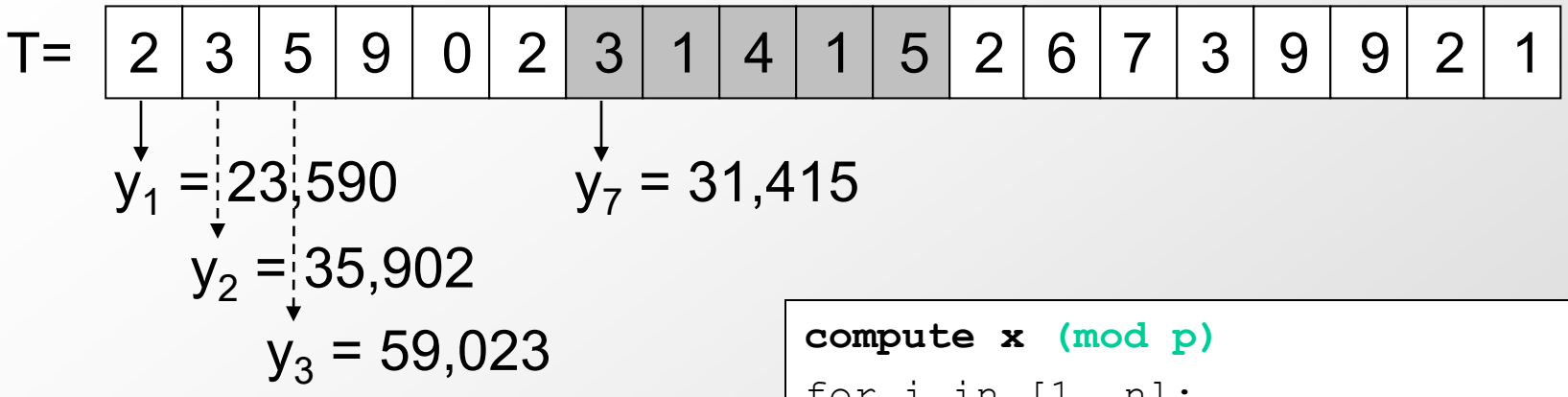
New problem: Collisions

(c) Dealing with collisions, due to hashing



- Consequences of (mod p) 'hashing'
  - Good: Enable fast computation (use small numbers)
  - Bad: Leads to spurious hits (collisions)
- Dealing with the bad:
  1. Verify that a **hit** correspond to valid **match**
     → re-compute equality for entire string (not just hash)
  2. Avoid worst-case behavior of many collisions w/ bad m
     → Choose **random m**
- Algorithm and its analysis becomes more complex:
  1. Compute expected run time, include expected cost of verification
  2. Show probability of spurious hits is small, expected run time is linear

Karp-Rabin algorithm: Putting it all together




```
compute x (mod p)
for i in [1..n]:
    compute y_i (mod p) (using y_{i-1})
    if x == y_i:
        if P==S[i..]:
            print "match at S[i]"
        else:
            (spurious hit)
```

- Key idea: Semi-numerical computation (this actually works)
  - **Idea: Interpret strings as numbers => fast comparison**
    (other semi-numerical methods: Fast Fourier Transform, Shift-And)
- To make it work:
  (a) Compute next number based on previous one → O(1)
  (b) Hashing (mod p) → keep the numbers small → O(1)
  (c) Dealing with collisions → Randomized p, expected run time → O(1) exp

Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

Sequence Alignment vs. Sequence Database Search

- Sequence Alignment
  - Assumes sequences have some common ancestry
  - Finding the “right” alignment between two sequences
  - Evolutionary interpretation: min # events, min cost
- Sequence Database Search
  - Given a query (new seq), and target (many old seqs), ask: which sequences (if any) are related to the query
  - Individual alignments need not be perfect: Once initial matches are reported, we can fine-tune them later
  - Query must be very fast for a new sequence
  - Most sequences will be completely unrelated to query
- Exploit distinct nature of database search problem

Speeding up your searches in dB setting

- Exploit nature of the problem (many spurious hits)
  - If you're going to reject any match with idperc <= 90, then why bother even looking at sequences which don't have a stretch of 10 nucleotides in a row.
  - Pre-screen sequences for common long stretches
- Put the speed where you need it (pre-processing)
  - Pre-processing the database is off-line.
  - Once the query arrives, must act fast
- Solution: content-based indexing and BLAST
  - Example: index 10-mers.
  - Only one 10-mer in 4^{10} will match, one in a million (even with 500 k-mers, only 1 in 2000 will match).
  - Additional speedups are possible

BLAST

Basic local **alignment search** tool

SF **Altschul**, W Gish, W Miller, EW Myers… - Journal of molecular …, 1990 - Elsevier

A new approach to rapid sequence comparison, basic local **alignment search** tool (BLAST), directly approximates **alignments** that optimize a measure of local similarity, the maximal …

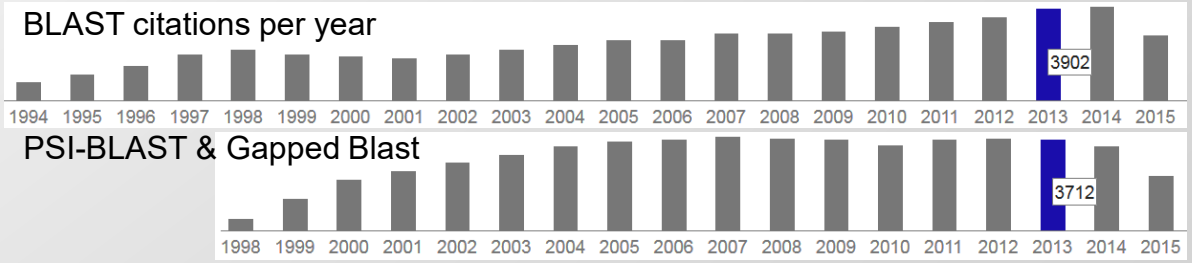
☆ Save Cite Cited by 107580 Related articles All 67 versions

Gapped BLAST and PSI-**BLAST**: a new generation of protein database search programs

SF **Altschul**, TL Madden, AA Schäffer… - Nucleic acids …, 1997 - academic.oup.com Paperpile

… **Myers** during the evaluation of the original **BLAST** algorithm. It was not included in the publicly distributed code primarily because the then current strategy of extending every hit …

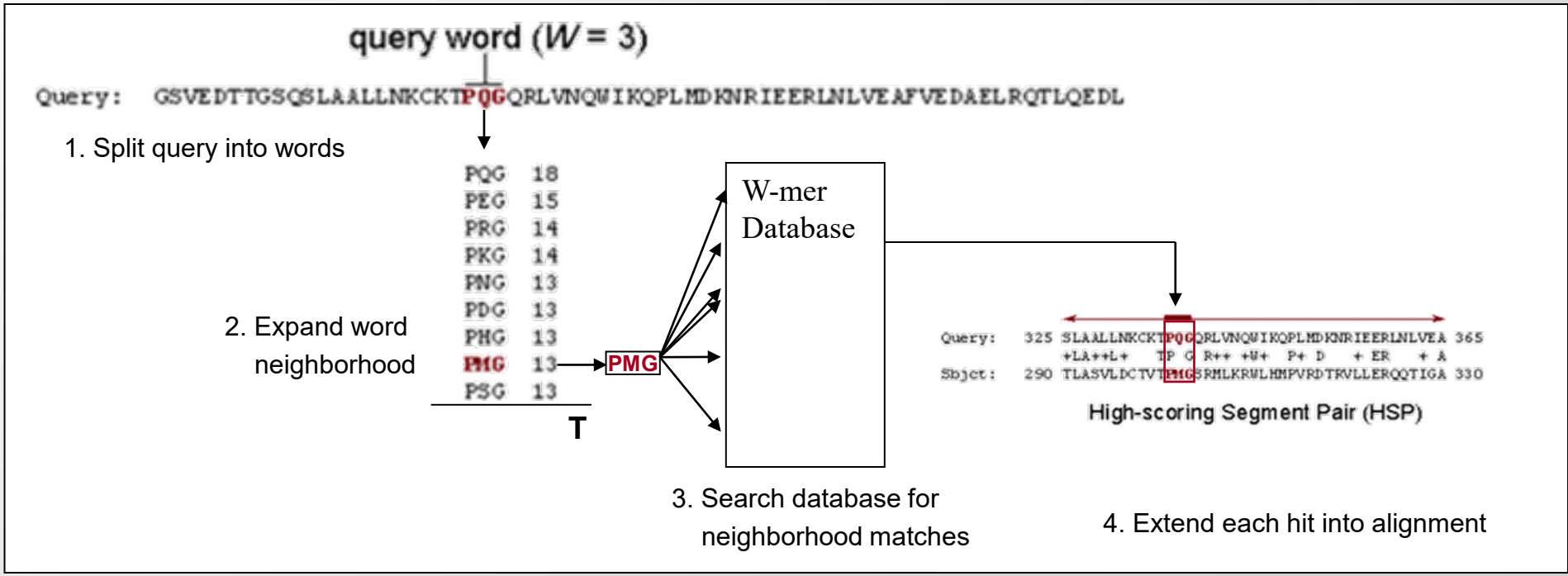
☆ Save Cite Cited by 83254 Related articles All 81 versions



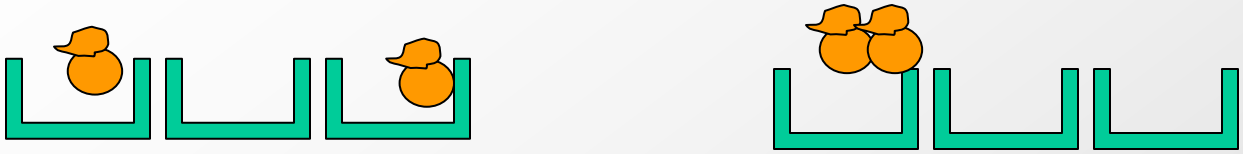
- Two key insights:
- Hashing:
  - Like Karp-Rabin, semi-numerical string matching
- Neighborhood search:
  - Can find hits even when no exact k-mer matches

Blast Algorithm Overview

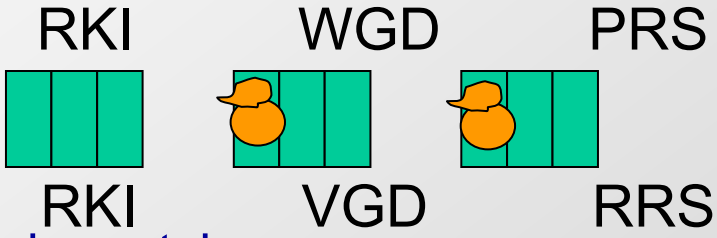
- Receive query
  1. Split query into overlapping words of length W
  2. Find neighborhood words for each word until threshold T
  3. Look in table where these neighbor words occur: seeds S
  4. Extend seeds S until score drops off under X
- Report significance and alignment of each match



Why BLAST works(1): Pigeonhole and W-mers



- Pigeonhole principle (flipped)
  - If you have 2 pigeons and 3 holes, there must be at least one hole with no pigeon



- Pigeonholing mis-matches
  - Two sequences, each 9 amino-acids, with 7 identities
  - There is a stretch of 3 amino-acids perfectly conserved
- In general:
  - Sequence length: n
  - Identities: t
  - Can use W-mers for W= [n/(n-t+1)]

Extensions to the basic algorithm

- Ideas beyond W-mer indexing? Desirata:
  - Faster
  - Better sensitivity (fewer false negatives)

1) Filtering: Low complexity regions cause spurious hits
   - Filter out low complexity in your query
   - Filter most over-represented items in your database
2) Two-hit BLAST
   - Two smaller W-mers are more likely than one longer one
   - Therefore it's a more sensitive searching method to look for two hits instead of one, with the same speed.
   - Improves sensitivity for any speed, speed for any sensitivity
3) Beyond W-mers, hashing with non-consecutive k-mers (combs)
   - Next slide

Extension 3: Combs and Random Projections

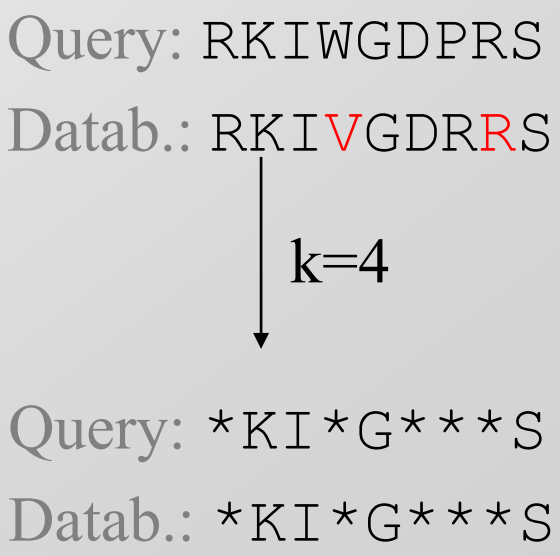
Key idea:

- No reason to use only consecutive symbols
- Instead, we could use combs, e.g.,

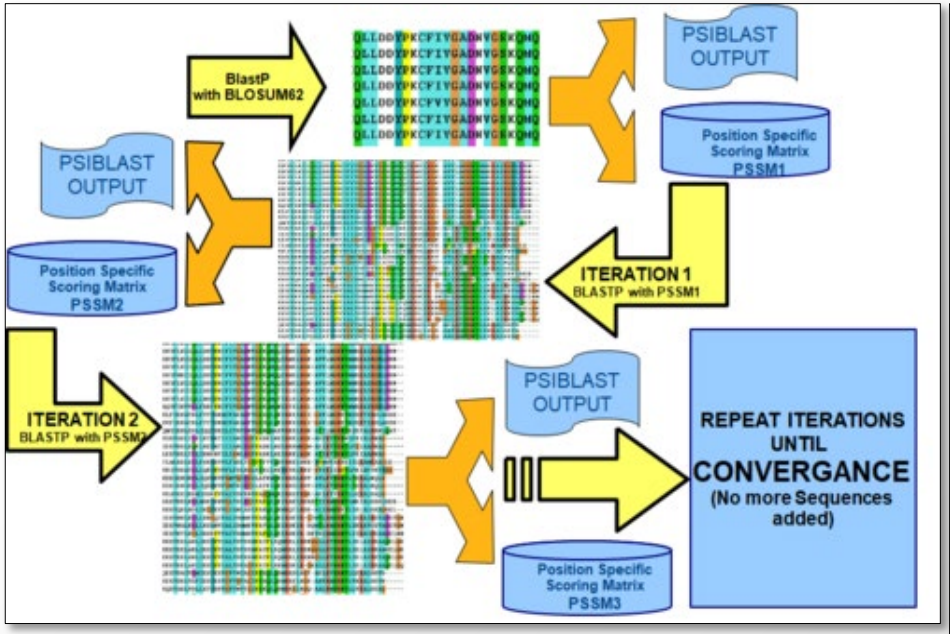
  RGIKW → R*IK* , RG**W, …
- Indexing same as for W-mers:
  - For each comb, store the list of positions in the database where it occurs
  - Perform lookups to answer the query
- How to choose the combs? At random
  - Random projections: Califano-Rigoutsos'93, Buhler'01, Indyk-Motwani'98
  - Choose the positions of * at random
  - Analyze false positives and false negatives

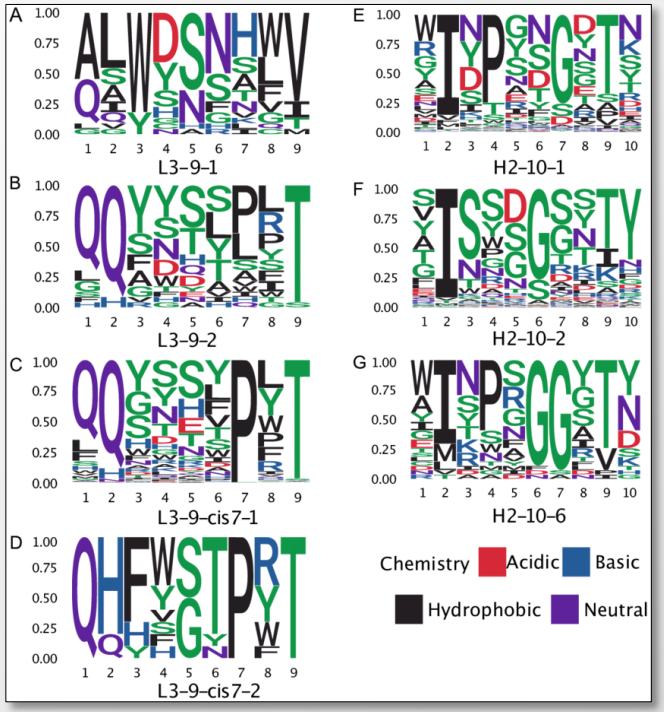
Performance Analysis:

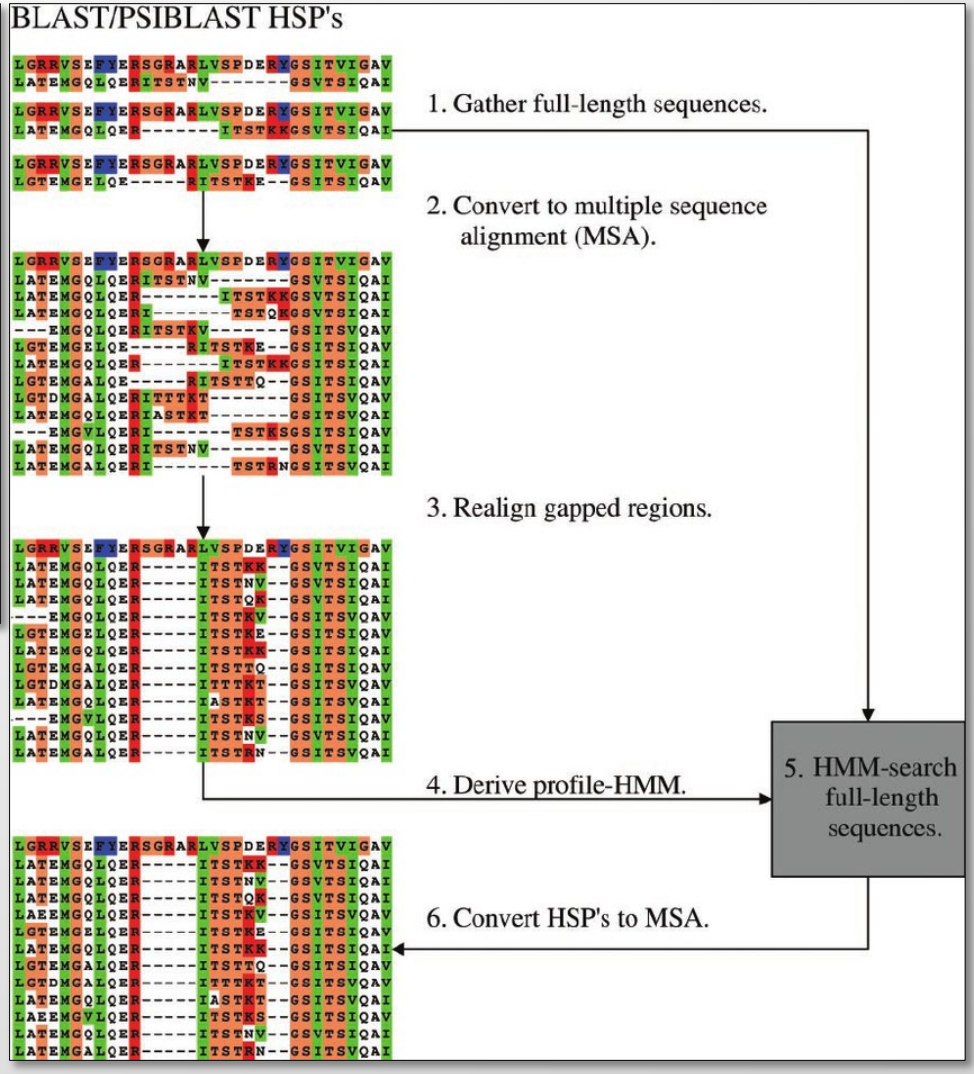
- Assume we select k positions, which do not contain *, at random with replacement
- What is the probability of a false negative ?
  - At most: 1-idperc^k
  - In our case: $1-(7/9)^4 = 0.63...$
- What is we repeat the process l times, independently ?
  - Miss prob. = 0.63^l
  - For l=5, it is less than 10%



Extension 4: Repeated Searches With Match Profiles







PSI-BLAST

Position-specific iterative BLAST

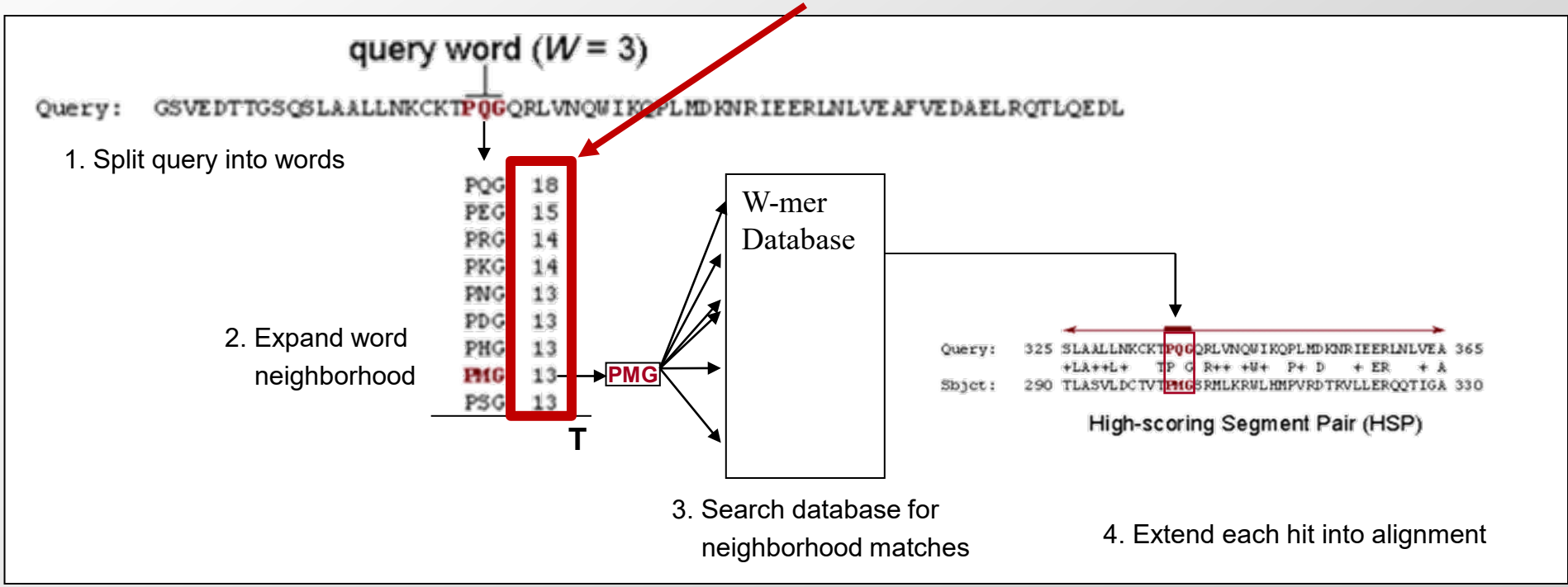
Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices

Recall: BLAST Algorithm Overview

- Receive query
  1. Split query into overlapping words of length W
  2. Find neighborhood words for each word until threshold T
  3. Look into the table where these neighbor words occur: seeds S
  4. Extend seeds S until score drops off under X
- Report significance and alignment of each match

Where did these scores come from?



Varying scores/penalties for matches/mismatches

Nucleotide space:

	A	G	T	C
A	+1	-½	-1	-1
G	-½	+1	-1	-1
T	-1	-1	+1	-½
C	-1	-1	-½	+1

purine pyrimid.

Transitions:

A⇔G, C⇔T common (lower penalty)

Transversions:

All other operations

Protein space: amino-acid match scores

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W

BLOSUM matrix of AA similarity scores

- Where do these scores come from?
- Are two aligned sequences actually related?

What should the alignment score represent?

- P that two ‘similar’ sequences are ‘homologous’
- Likelihood ratio between two hypotheses
  - Hypothesis 1: alignment due to chance (unrelated)
  - Hypothesis 2: due to common ancestry (related)
- Calculate probability of observing an alignment according to each hypothesis
  - Pr(x,y|U): P of alignment x,y by model U (unrelated)
  - Pr(x,y|R): P of alignment x,y by model R (related)
- Alignment score: likelihood ratio between the two
  - P that aligmt not due to chance = P(x,y|R) / P(x,y|U)
  - Score = log(P)
- a-ha moment: additive matrices are exactly that!

Model for Unrelated Sequences

- we'll assume that each position in the alignment is sampled randomly from some distribution of amino acids
- let q_a be the probability of amino acid a
- the probability of an n-character alignment of x and y is given by

$$\Pr(x, y \mid U) = \prod_{i=1}^{n} q_{x_i} \prod_{i=1}^{n} q_{y_i}$$

Model for Related Sequences

- we'll assume that each pair of aligned amino acids evolved from a common ancestor
- let p_{ab} be the probability that evolution gave rise to amino acid a in one sequence and b in another sequence
- the probability of an alignment of x and y is given by

$$\Pr(x, y \mid R) = \prod_{i=1}^{n} p_{x_i y_i}$$

Likelihood ratio of the two models

- How can we decide which possibility (U or R) is more likely?
- one principled way is to consider the relative likelihood of the two possibilities (the odds ratio)

$$\frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} q_{y_i}}$$

- taking the log, we get

$$\log \frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)} = \sum_i \log \left(\frac{p_{x_i y_i}}{q_{x_i} q_{y_i}} \right)$$

➢ **Product of 'column' probabilities**

Additive score (in log space)

- the score for an alignment is thus given by:

$$S = \sum_i s(x_i, y_i) = \log \frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)}$$

- the substitution matrix score for the pair a, b should thus be given by:

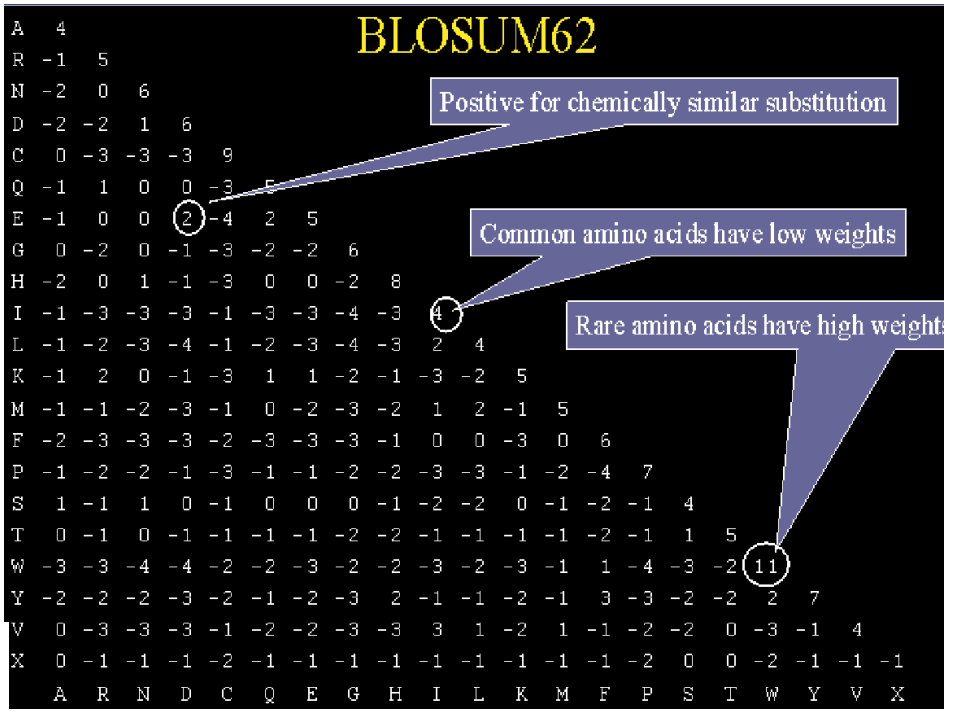
$$s(a, b) = \log \left(\frac{p_{ab}}{q_a q_b} \right)$$

➢ **Sum of log 'column' scores**

Where score matrices come from

Substitution Score Matrices

- two popular sets of matrices for protein sequences
  - PAM matrices [Dayhoff *et al.*, 1978]
  - BLOSUM matrices [Henikoff & Henikoff, 1992]
- both try to capture the the relative substitutability of amino acid pairs in the context of evolution

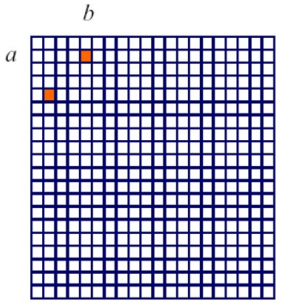


Create a trusted matrix of scores

- the substitution matrix score for the pair a, b is given by:

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right)$$

- given: a set of sequences in a block
- fill in matrix A with number of observed substitutions (we won't worry about details of some normalization that happens here)



$$p_{ab} = \frac{A_{ab}}{\sum_{c,d} A_{cd}} \qquad q_a = \frac{\sum_b A_{ab}}{\sum_{c,d} A_{cd}}$$

Adjusting for 'age' of divergence

- but how do we get values for p_{ab} (probability that a and b arose from a common ancestor)?
- it depends on how long ago sequences diverged
  - diverged recently: $p_{ab} \approx 0$ for $a \neq b$
  - diverged long ago: $p_{ab} \approx q_a q_b$

Empirically-derived subst. freq's

- key idea: trusted alignments of related sequences provide information about biologically permissible mutations
- [Henikoff & Henikoff, *PNAS* 1992]
- probabilities estimated from "blocks" of sequence fragments that represent structurally conserved regions in proteins
- transition frequencies observed directly by identifying blocks that are at least
  - 45% identical (BLOSUM-45)
  - 50% identical (BLOSUM-50)
  - 62% identical (BLOSUM-62)
  - etc.

Alignment, Dynamic Programming, Hashing, BLAST

1. **Comparative Genomics**, Conservation, Mutational Processes
   - Reading Evolution's Notebook: Using evolution to understand genomes
2. **Modeling evolution**, inferring mutation path, edit distance
   - From Bio to CS, problem formulation, evolutionary operations, optimality
   - Substring matching, subseq. matching, varying costs, searching 2^N alignments
3. **Principles of Dynamic Programming**: Ordering Computation
   - Fibonacci, top-down vs. bottom-up, repeated sub-problems, lookup. $O(2^N) \rightarrow O(N)$
   - DP recipe: Matrix, Sub-problem space, traversal order, recursion, trace-back
4. **Sequence alignment:** Why and How Dynamic Programming applies
   - Additive score, build from smaller, prefix matrix, N^2 subproblems, 2^N paths
   - Duality: matrix entry $\Leftrightarrow$ alignment score (N^2); traversal path $\Leftrightarrow$ alignment (2^N)
5. **Power of DP matrix abstraction**: Linear Time, Linear Space, Local, Gaps
   - Linear-time bounded DP (heuristic). Linear-space DP: Hirschberg algorithm.
   - Local alignment, semi-global. Vary gap penalties, impact on runtime
6. **Hashing**: Linear-time exact string matching (expected time, no mismatches)
   - Local substring matching: Karp-Rabin algorithm, semi-numerical methods
   - Hash functions, content-based look-up. Randomized algorithms, expected runtime
7. **The BLAST algorithm**: inexact matching
   - Hashing, neighborhood search, properties of good hits
   - Algorithmic speed-ups, two-hit blast, hashing with combs, repeated searches
8. **Probabilistic foundations** of sequence alignment and scoring matrices
   - Related vs. unrelated model, likelihood ratios, $\log(\Pi)=\Sigma(\log)$, additive scores
   - Building mismatch score/penalty matrix, empirical derivations, BLOSUM matrices