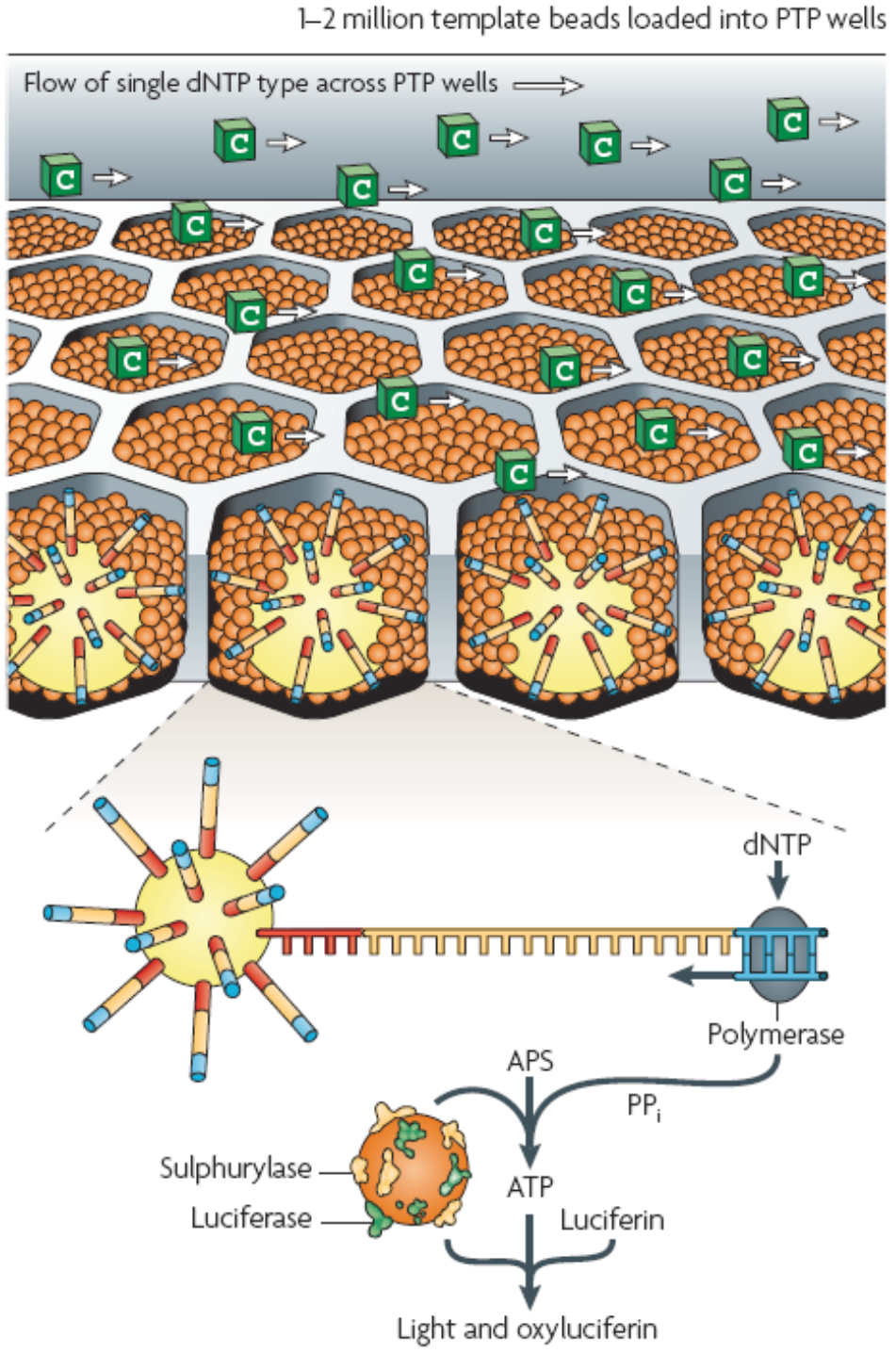
生物信息学

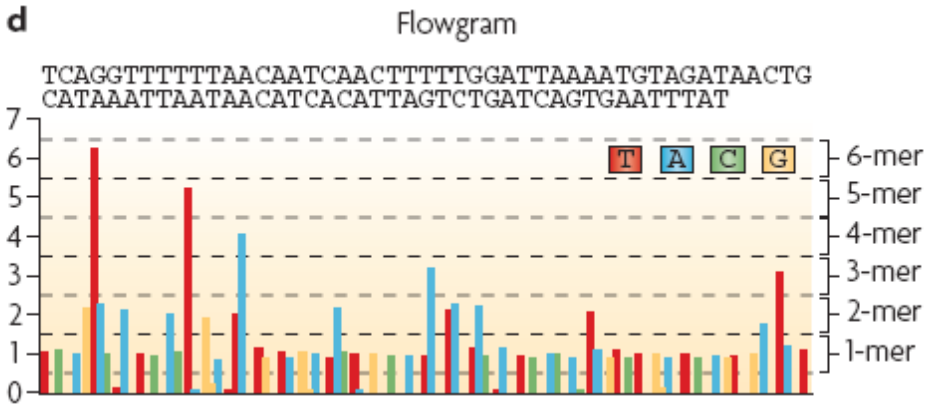
第十章 测序读段回贴

高通量DNA测序：生物信息学的挑战

Pyrosquencing

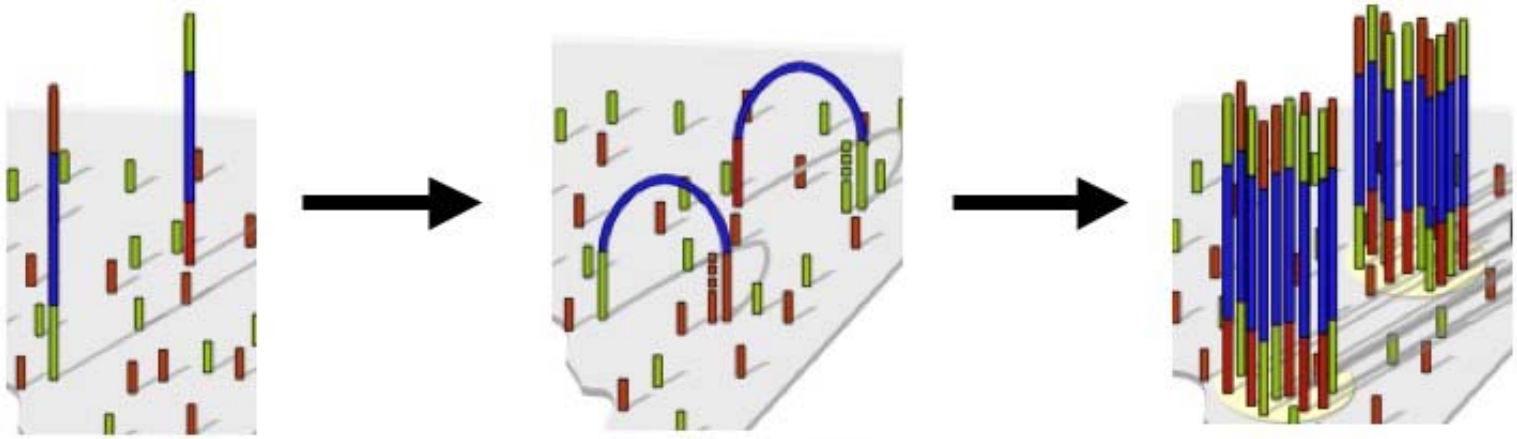
Roche 454

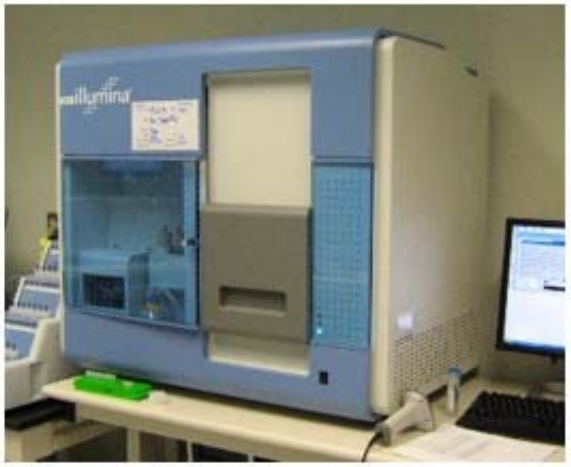


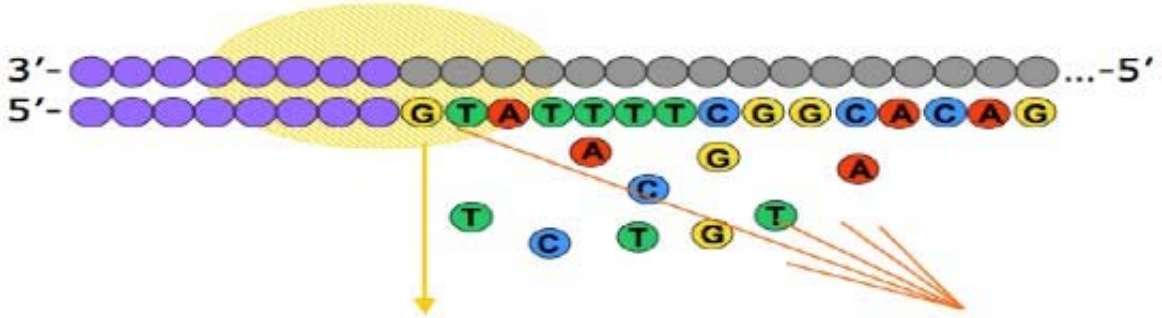


Sequencing by synthesis

Illumina Genome Analyzer



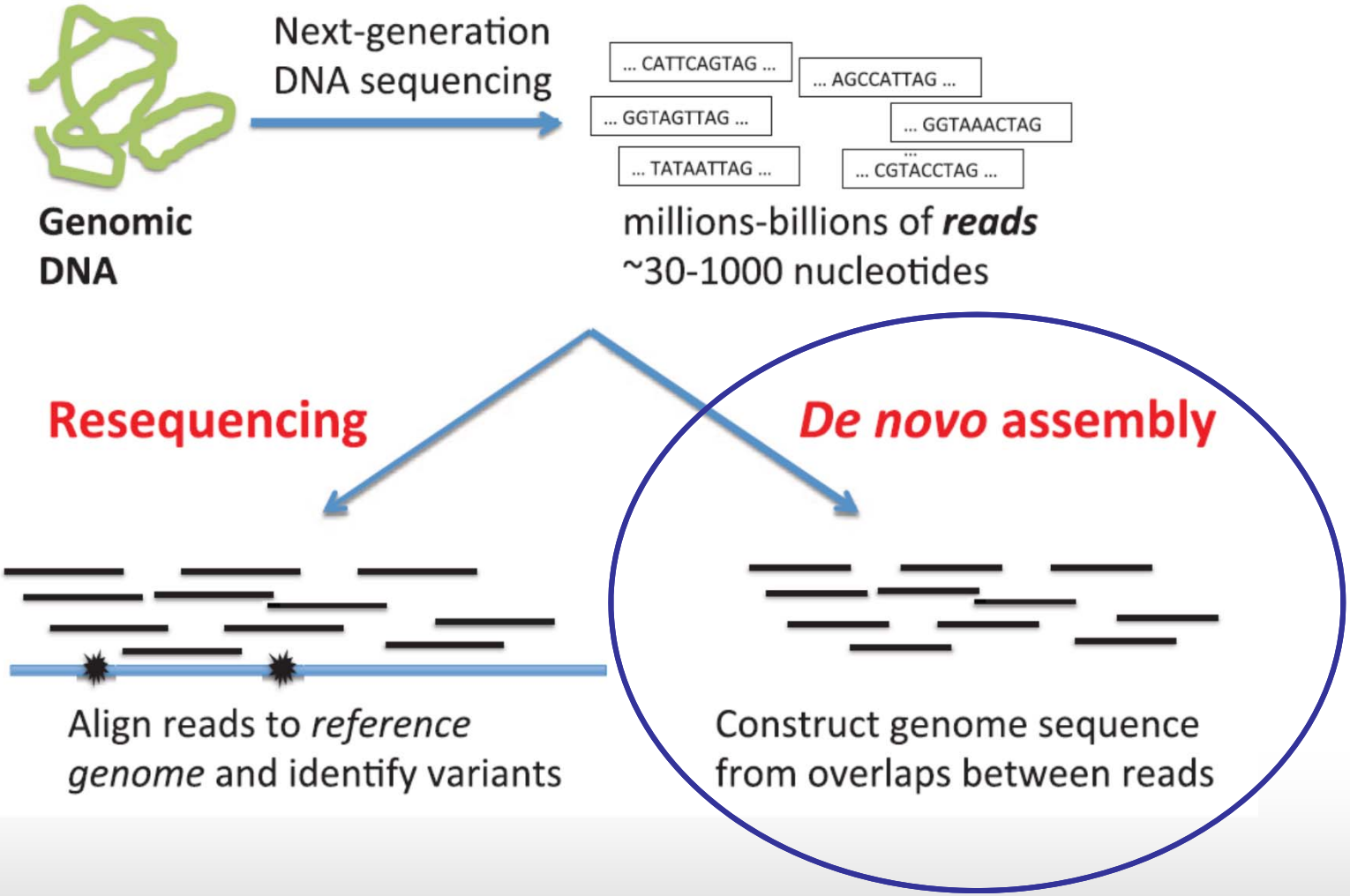




Cycle 1: Add sequencing reagents
First base incorporated
Remove unincorporated bases
Detect signal

Cycle 2-n: Add sequencing reagents and repeat

全基因组测序

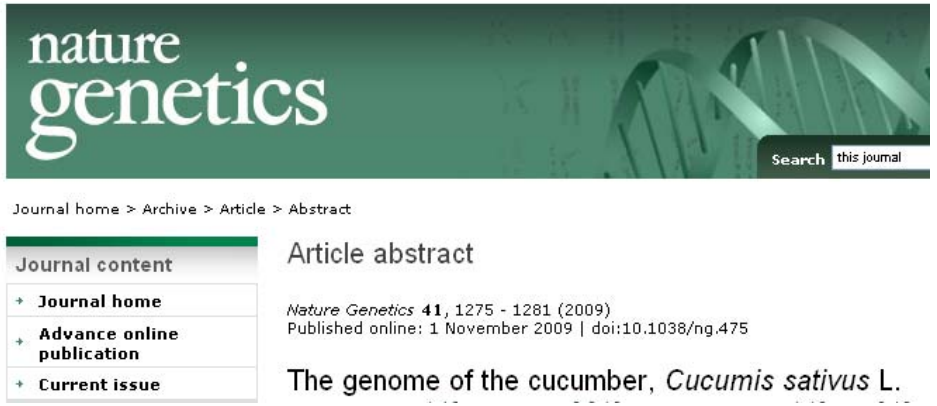

Next-generation
DNA sequencing
... CATTCAGTAG ...
... AGCCATTAG ...
... GGTAGTTAG ...
... GGTAAACTAG
... TATAATTAG ...
... CGTACCTAG ...
Genomic
DNA
millions-billions of reads
~30-1000 nucleotides
Resequencing
De novo assembly
Align reads to reference
genome and identify variants
Construct genome sequence
from overlaps between reads

从头测序

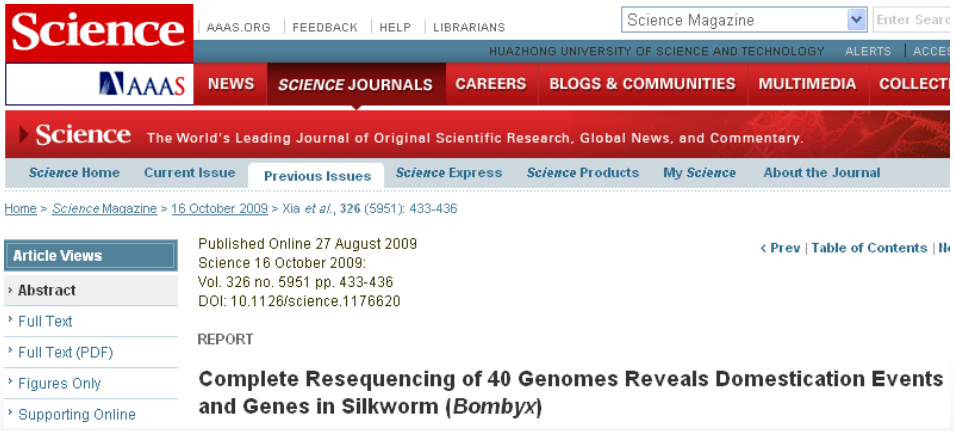
- ***De novo* sequencing**
  - 新的物种/品系
  - 短读段组装中的挑战：重复片段
- 突发事件的基因组学（Emergency Genomics）
  - SARS病毒、禽流感、猪流感、新冠病毒…
- 古代DNA（Ancient DNA）
  - 楼兰女尸，罗马军团，长沙马王堆，曹操墓…
  - 猛犸，乳齿象，恐龙

重要动植物

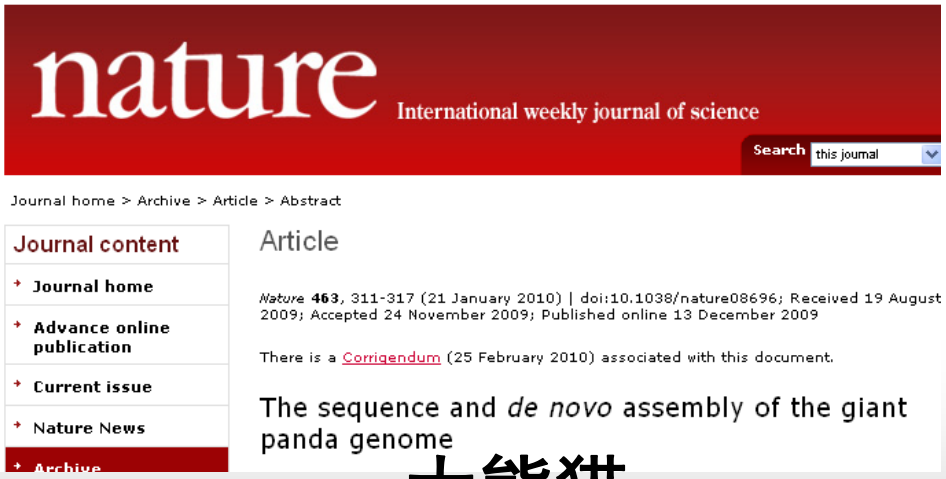
大豆



黄瓜



家蚕

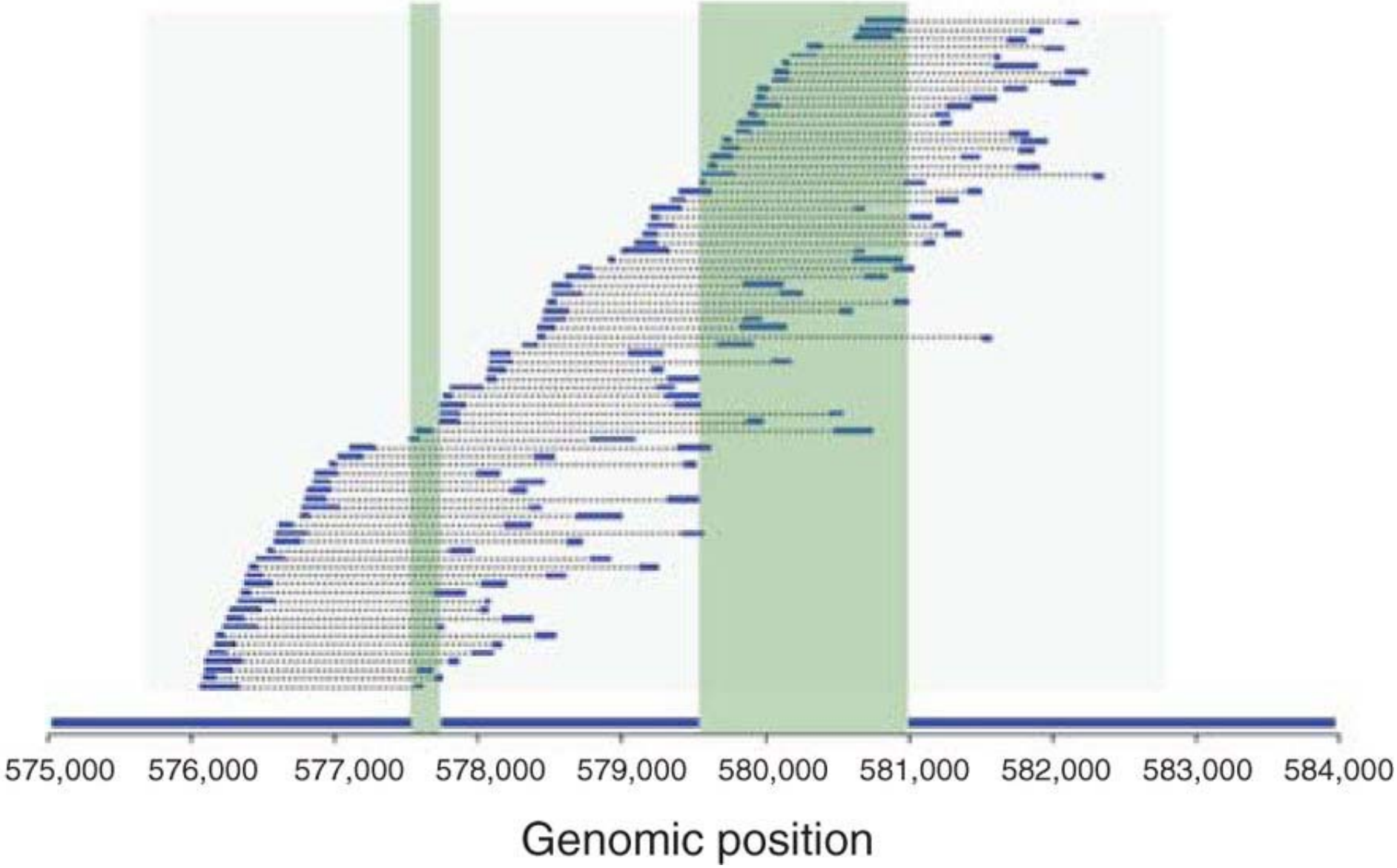


大熊猫

宏基因组（Metagenomics）

- 发现环境中，或者疾病样本中所有的物种
- 人类微生物组（Human Microbiome）
  - 口腔、肠道、皮肤的疾病 vs. 健康微生物的群落
- 全基因组测序
- 多物种转录本的完全检测
- 微生物群落中遗传变异的深度采样
  - 耐药
  - 毒素生成

读段如何拼装成基因组？


575,000
576,000
577,000
578,000
579,000
580,000
581,000
582,000
583,000
584,000
Genomic position

重测序（Resequencing）

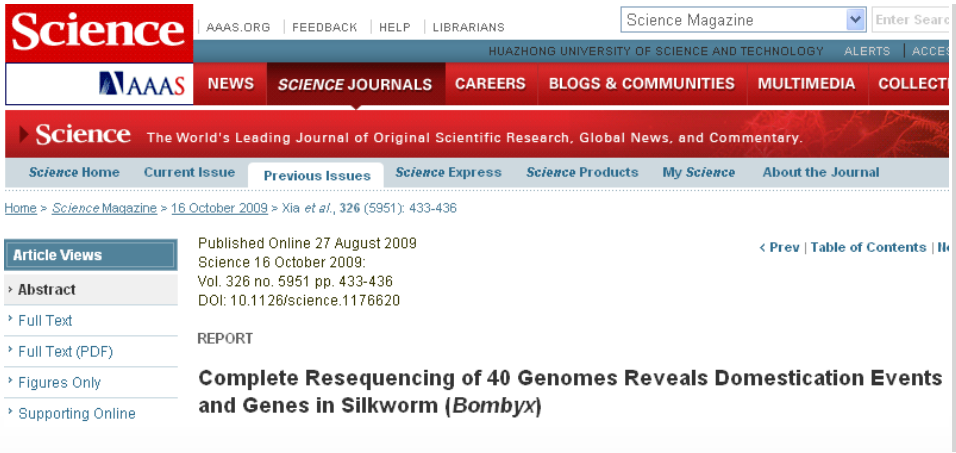
- 基因型和突变的发现
- 当前测序工作很大一部分主要是针对已有基因组物种的重测序
  - 单个人的基因组（1000 Genomes Project）
  - 模式生物：寻找遗传突变（Genetic variation）和拷贝数变异（Copy number variation）等
- 挑战：
  - 将数百万个短片段（Reads）快速的回贴到参考基因组上，并估算错误率
  - 准确的发现SNPs和indels

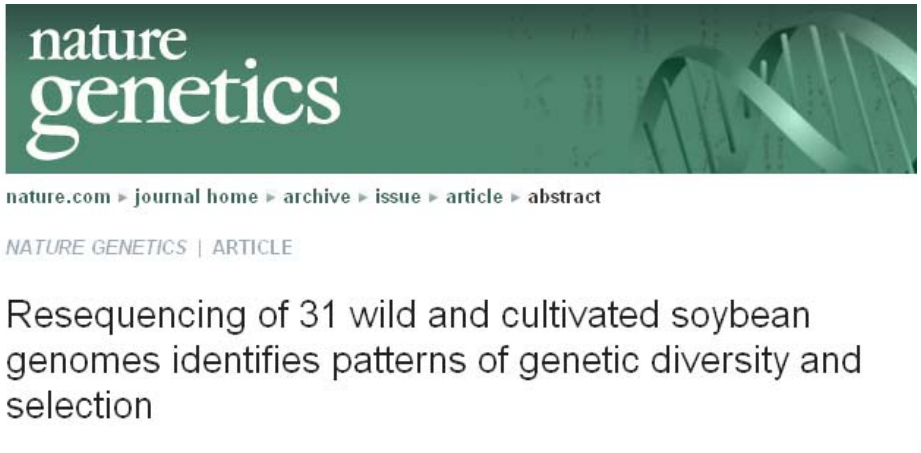
1000 Genomes Project

- 四个种群的179个人的全基因组测序
- 两组母亲-父亲-孩子的高覆盖度基因组测序
- 七个种群的697个人的外显子测序

深度测序（Deep Sequencing）

- 发现或者定量罕见的序列变异
  - 单个患者体内的HIV突变多样性
  - 肿瘤样本中的突变
  - 物种的单核甘酸多样性



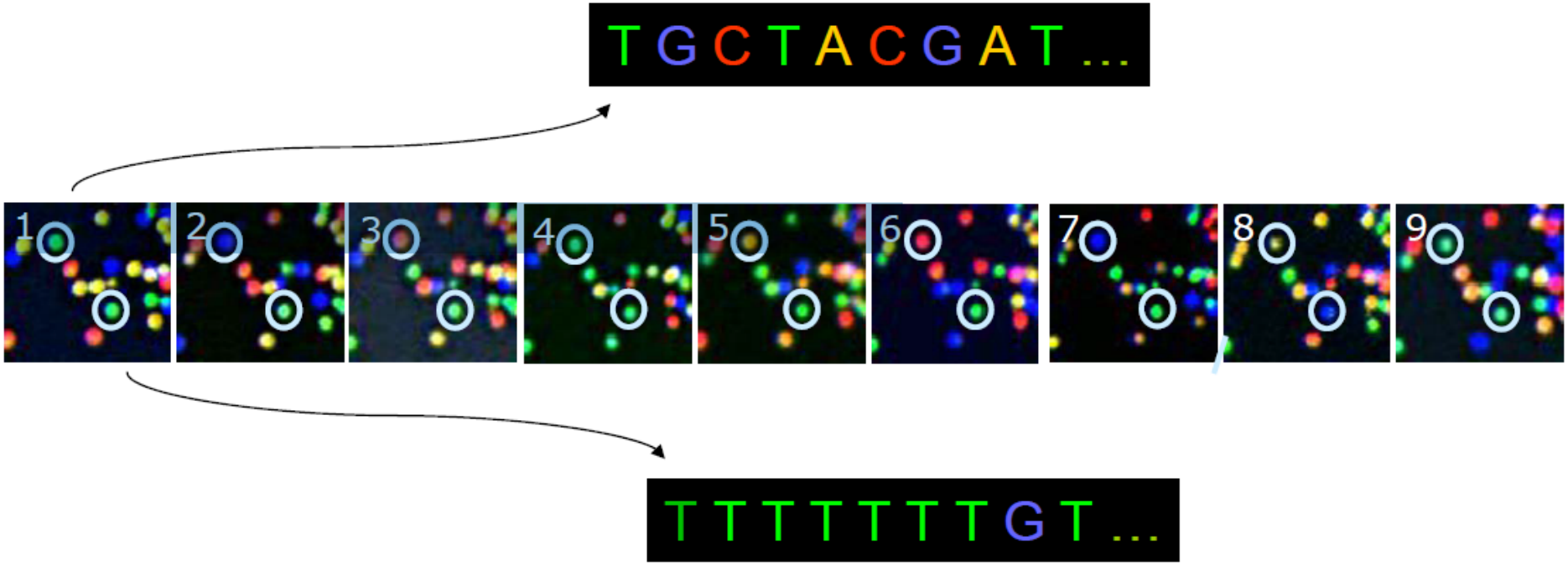


生物信息学挑战

- Base calling: 根据图像识别碱基
- 序列读段回贴到参考序列上
- 全基因组的组装
- 基因型（Genotype）和SNP的识别
- RNA-Seq数据分析：检测基因表达、可变剪接异构体等
- Chip-Seq数据中的Peak finding，以及受调控基因的回贴

碱基识别

- **Base calling: 从图像中识别碱基，组成序列**



数据格式：FASTQ

- FASTQ格式的序列一般包含四行
  - 第一行由‘@’开始，后面跟着序列的描述信息
  - 第二行是序列
  - 第三行由‘+’开始，后面也可以跟着序列的描述信息
  - 第四行是第二行序列的测序质量分值（quality values）

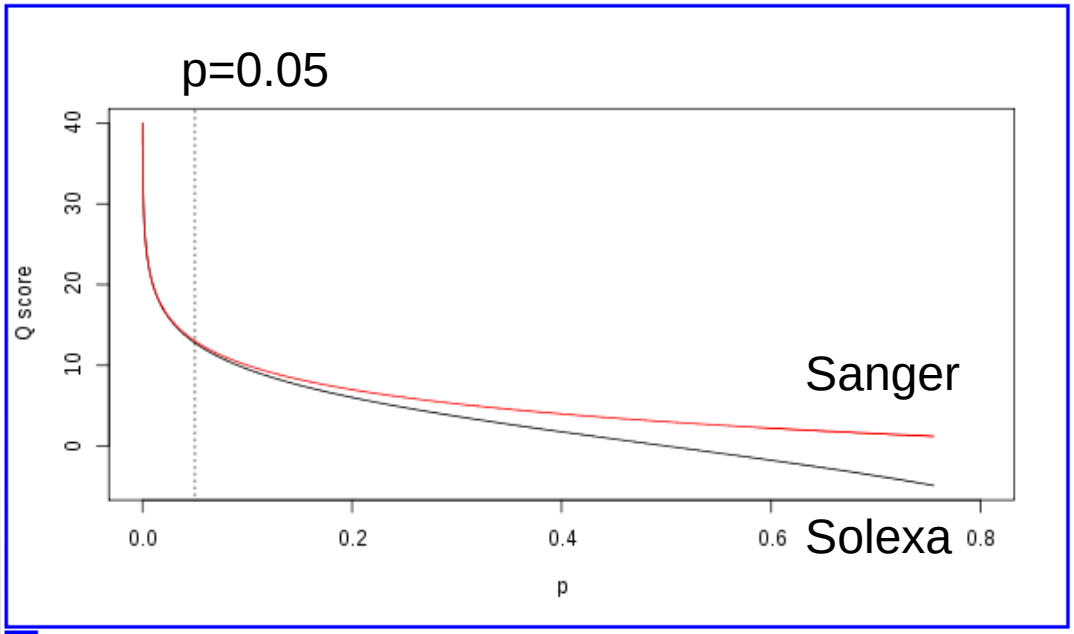
```
1  @
2  NATTAAACACTGGGTCAGGTCGTATGCCGTCTTCTGCTTGAAAAAAAAA
3  +
4  BNPPNWQQQR````_VIXXSSUUSQX\[\Z_`___BBBBBBBBBBBBB
5  @
6  NGAGGTAGTAGGTTGTATGGTTATCGTATGCCGTCTTCTGCTTGAAAAA
7  +
8  BLJJJPTTTTddddd[[[\]aaaa]^]^]^^BBBBBBBBBBBBBBBBB
9  @
10 NGAGGTAGTAGATTGTATAGTTTCGTATGCCGTCTTCTGCTTGAAAAAA
11 +
12 BRVMNTWWWVaaaaaddddddaaaaa^_^__^^_^_BBBBBBBBBBBBB
```

```
@HWI-EAS305:1:1:1:991#0/1
GCTGGAGGTTCAGGCTGGCCGGATTTAAACGTAT
+HWI-EAS305:1:1:1:991#0/1
MVXUWVRKTWWULRQQMMWWBBBBBBBBBBBBBB
@HWI-EAS305:1:1:1:201#0/1
AAGACAAAGATGTGCTTTCTAAATCTGCACTAAT
+HWI-EAS305:1:1:1:201#0/1
PXX[[[[XTXYXTTWYYY[XXWWW[TMTVXWBBB
@HWI-EAS305:1:1:1:70#0/1
ATGATAATAATACCTCTTGCAGTTTGCATCATGT
+HWI-EAS305:1:1:1:70#0/1
OYYYY[[Z[YZ[Y[[WYYP[YYYWZTZ[ZYBBBB
@HWI-EAS305:1:1:1:983#0/1
ACCCAATACCGGTACAGGAATTGCAGCAATCAAA
+HWI-EAS305:1:1:1:983#0/1
OYYXVVYYYYUVUYXVRSUYYUTUTPVYYYYVVT
@HWI-EAS305:1:1:1:1671#0/1
AATTACACAACAAAAGGAGATCAAAGGGATACAA
+HWI-EAS305:1:1:1:1671#0/1
OY[[[[YXXX[[[ZXUWXZZ[Y[[VTTUU[[[YX
@HWI-EAS305:1:1:1:1699#0/1
GTTGGCTCTACGATCACGTTGCTCACCATGTGGG
+HWI-EAS305:1:1:1:1699#0/1
JSUSSUWTWWQUUUVVUUUTTTUTUWBBBBBBBB
@HWI-EAS305:1:1:1:1616#0/1
ATTGGCGACGATATTCAGGTCCATGTTTCTTGCG
+HWI-EAS305:1:1:1:1616#0/1
NYYYVRVVVSVYYYYWUNNPWVYBBBBBBBBBB
```

Quality values

- 质量评价分值Q
- p: 该位置碱基不正确的可能性
- p=0.05时，Q=13
- p=0.001，Q=30
- 编码：ASCII码

$$Q_{\text{sanger}} = -10 \log_{10} p$$



ASCII标准表

ASCII表

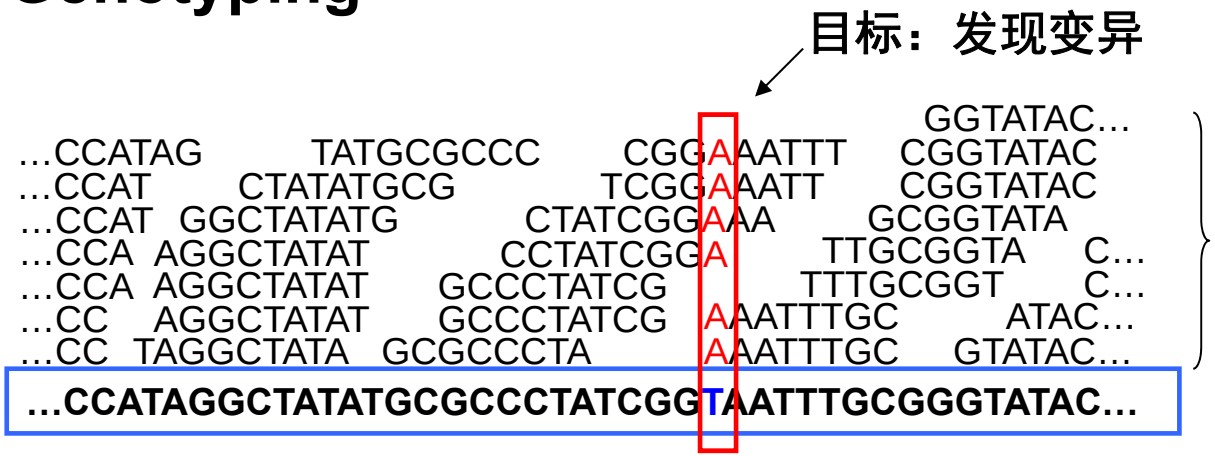
（ American Standard Code for Information Interchange 美国标准信息交换代码 ）

高四位 / 低四位		ASCII控制字符												ASCII打印字符													
		0000						0001						0010		0011		0100		0101		0100		0111			
		0						1						2		3		4		5		6		7			
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl	
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p		
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q		
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r		
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s		
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t		
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u		
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v		
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w		
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x		
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y		
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z		
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{		
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|		
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}		
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~		
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	^Backspace 代码：DEL	

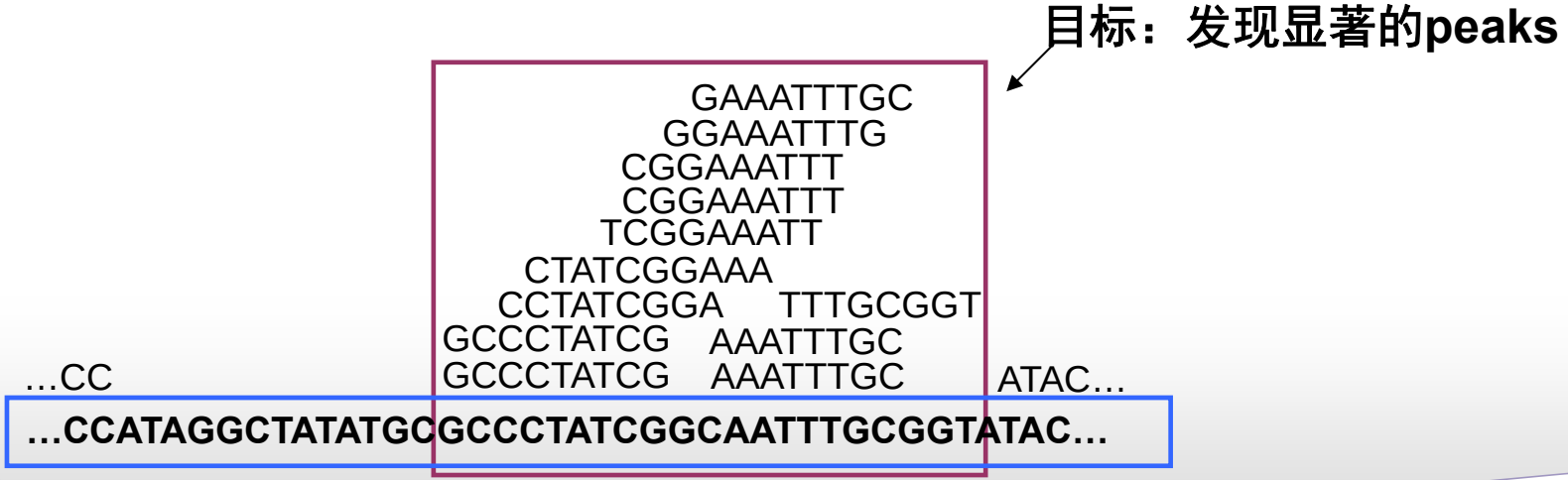
注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。　　制作：MHL　QQ:1208980380　2013/08/08

读段回贴

Genotyping

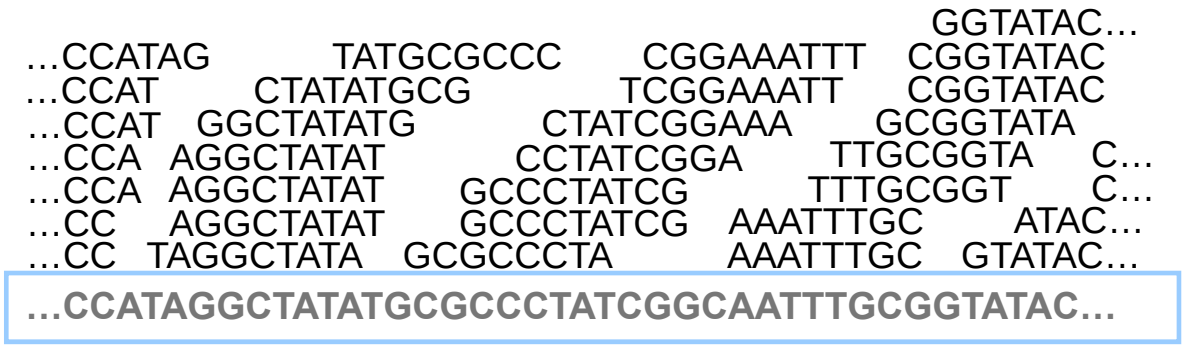


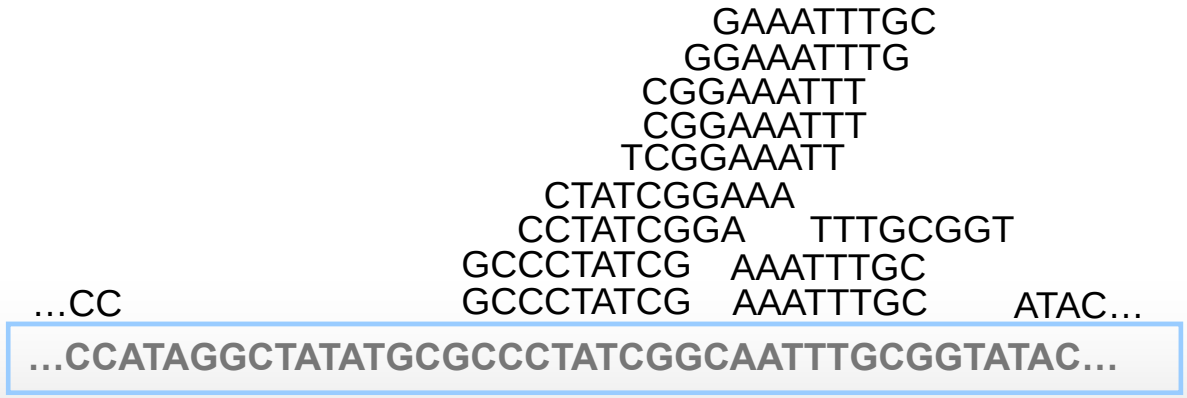
RNA-seq, ChIP-seq, Methyl-seq



读段映射

- 瓶颈：“最优的”比对结果？
  - 准确性 *vs.* 速度 *vs.* 运算空间





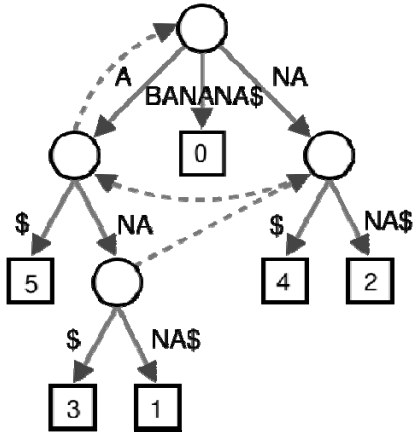
读段比对

- **Short Read Alignment**
  - 给定参考序列和一系列读段
  - 每个读段：发现至少一个“好的”局部比对
- “好的”比对
  - 错配越少越好
  - 质量低的碱基错配率高
  - 质量高的碱基错配率低

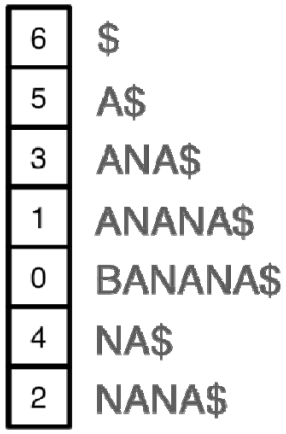
```
...TGATCATA...      better than   ...TGATCATA...
   ||||| |                           || ||
   GATCAA                            GAGAAT

...TGATATTA...      better than   ...TGATcaTA...
   ||| |                             | |||
   GATcaT                            GTACAT
```

索引（Indexing）

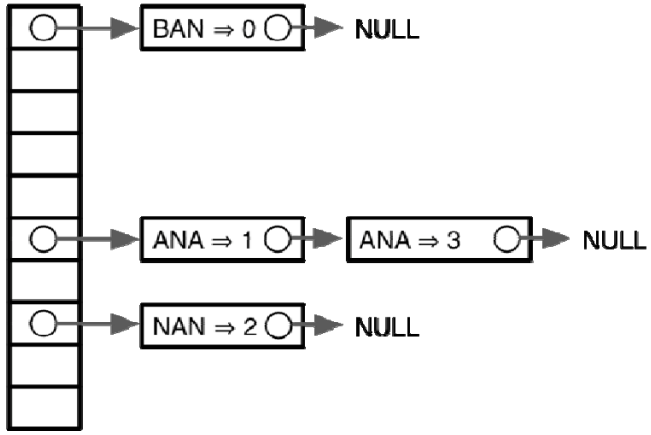
- 基因组和读段数据规模太大：不能直接用动态规划算法
- 索引是必须的: **BANANA$**



后缀树 (Suffix tree)



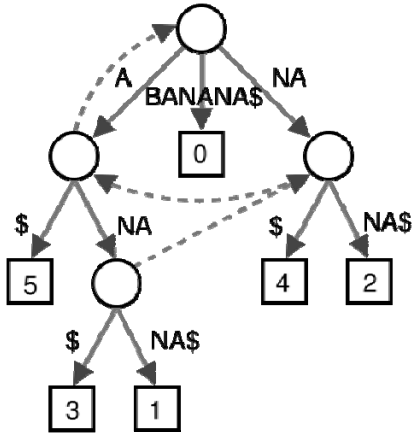
后缀数组 (Suffix array)



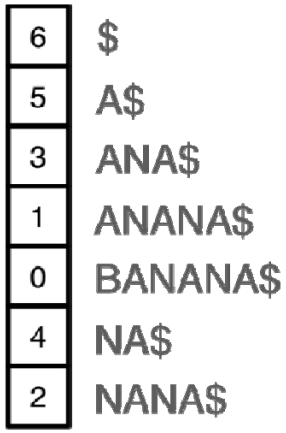
种子哈希表 (Seed hash tables

索引（Indexing）

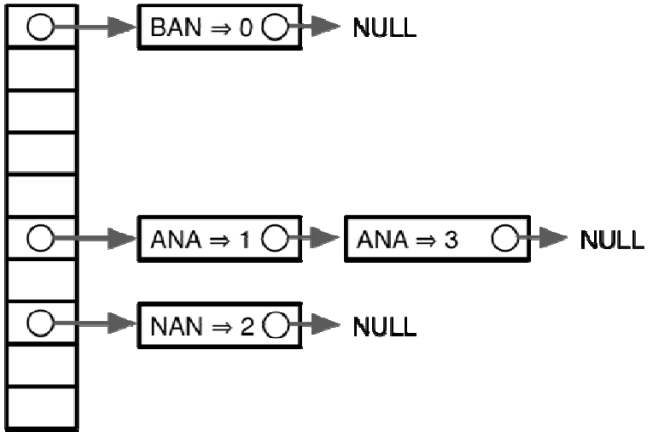
- 人类基因组的索引规模



> 35 GBs



> 12 GBs



> 12 GBs

- 合适的索引大小对计算性能很重要
- 大的索引：需要更多的内存空间

SOAP

- 2008年，华大基因
  - Short oligonucleotide alignment program
- “Seed and hash look-up table”
  - 先搜索种子序列：完全匹配
  - 再根据索引匹配剩余序列
  - 允许最多两个错配或空位 (gap)
- 种子序列（seed）：
  - 酵母：L=12Mb，S=10bp, 200 Mb RAM
  - 人类：L=3Gb，S=12 bp, 14Gb RAM

Program	Time consumed (s)	Reads aligned (%)
blastn (−F F −W 11)	165 780	85.47
blastn (−F F −W 15)	150 660	84.66
Blat (−tileSize = 8)	22 032	85.07
Eland	166	88.53
Maq	458	88.39
Soap	134	88.46
Soap iterative	161	90.9
Soap iterative + gapped	486	91.15

Burrows-Wheeler Transform (BWT)

- 1994年，Michael Burrows和David Wheeler发明
- 数据压缩算法（bzip2）
- 聚类相似的字符：输出能够更容易压缩
- 每一轮将最后一列的字符移到第一列
- 输出：矩阵的最后一列

	矩阵	排序	最后一列	
acaacg$	$acaacg	$acaacg	$acaacg	gc$aaac
	g$acaac	aacg$ac	aacg$ac	
	cg$acaa	acaacg$	acaacg$	
	acg$aca	acg$aca	acg$aca	
	aacg$ac	caacg$a	caacg$a	
	caacg$a	cg$acaa	cg$acaa	
	acaacg$	g$acaac	g$acaac	
T				BWT(T)

Burrows-Wheeler Transform

- BWT *vs.* 后缀数组（Suffix array, SA）

BWM(T)	SA(T)
$ a c a a c g	6 $
a a c g $ a c	2 a a c g $
a c a a c g $	0 a c a a c g $
a c g $ a c a	3 a c g $
c a a c g $ a	1 c a a c g $
c g $ a c a a	4 c g $
g $ a c a a c	5 g $

Burrows-Wheeler Transform

- T-ranking: 给T中每一个字符赋秩（Rank）

$a_0\ c_0\ a_1\ a_2\ c_1\ g_0\ \$$

$\$\ a_0\ c_0\ a_1\ a_2\ c_1\ g_0$
$a_1\ a_2\ c_2\ g_0\ \$\ a_0\ c_0$
$a_0\ c_0\ a_1\ a_2\ c_1\ g_0\ \$$
$a_2\ c_1\ g_0\ \$\ a_0\ c_0\ a_1$
$c_0\ a_1\ a_2\ c_1\ g_0\ \$\ a_0$
$c_1\ g_0\ \$\ a_0\ c_0\ a_1\ a_2$
$g_0\ \$\ a_0\ c_0\ a_1\ a_2\ c_1$

Burrows-Wheeler Transform

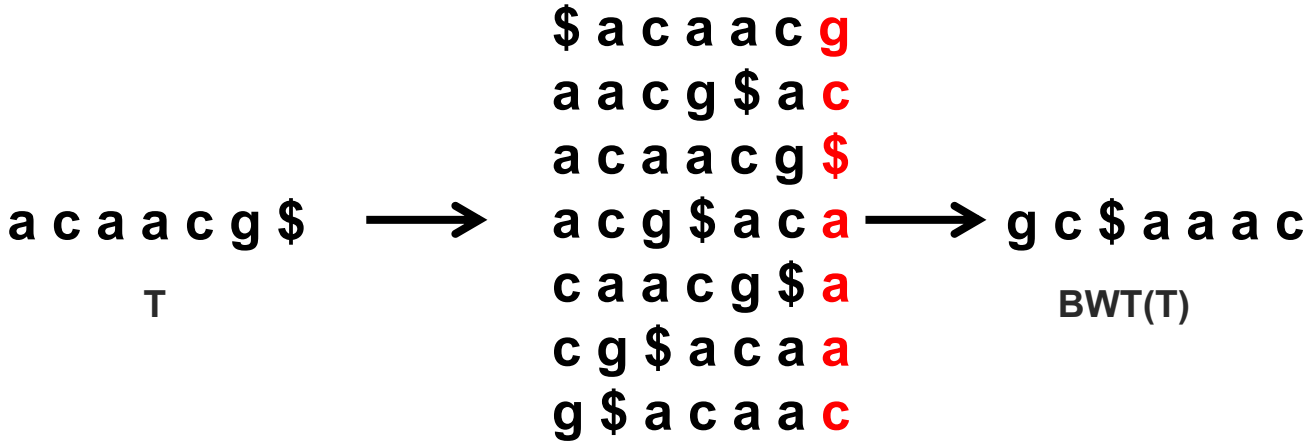
- BWT(T)的可逆性
  - 最后一列第i^{th}个出现的字符，是第一列第i^{th}个出现的字符

$\$\ a_0\ c_0\ a_1\ a_2\ c_1\ g_0$
$a_1\ a_2\ c_2\ g_0\ \$\ a_0\ c_0$
$a_0\ c_0\ a_1\ a_2\ c_1\ g_0\ \$$
$a_2\ c_1\ g_0\ \$\ a_0\ c_0\ a_1$
$c_0\ a_1\ a_2\ c_1\ g_0\ \$\ a_0$
$c_1\ g_0\ \$\ a_0\ c_0\ a_1\ a_2$
$g_0\ \$\ a_0\ c_0\ a_1\ a_2\ c_1$

$\$\ a_0\ c_0\ a_1\ a_2\ c_1\ g_0$
$a_1\ a_2\ c_2\ g_0\ \$\ a_0\ c_0$
$a_0\ c_0\ a_1\ a_2\ c_1\ g_0\ \$$
$a_2\ c_1\ g_0\ \$\ a_0\ c_0\ a_1$
$c_0\ a_1\ a_2\ c_1\ g_0\ \$\ a_0$
$c_1\ g_0\ \$\ a_0\ c_0\ a_1\ a_2$
$g_0\ \$\ a_0\ c_0\ a_1\ a_2\ c_1$

Burrows-Wheeler Transform

- 便于压缩：相同的字符在一起
- 怎样可逆？
- 怎样做索引？



Run-length encoding

- 可逆的数据压缩算法
  - WWWWWWWWWWWWBWWWWWWWWWWWWBBB WWWWWWWWWWWWWWWWWWWWWWWWBWWW WWWWWWWWWWW
  - -> 12W1B12W3B24W1B14W
- a c a a c g -> 1a1c2a1cg

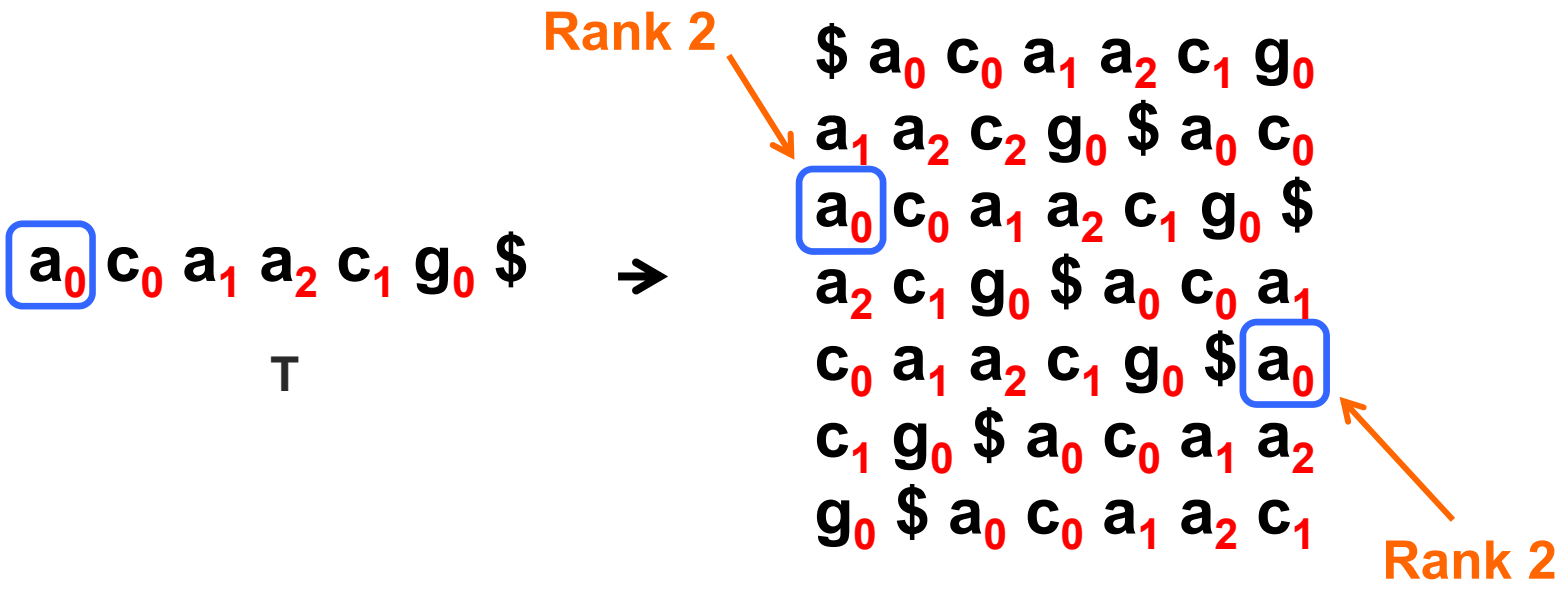
Move-to-front transform

- 数据编码的方式
- 字符->数字
- “Recently used symbols”
- 可逆转换：从后往前

a c a a c g	1	(abcdefghijklmnopqrstuvwxyz)
a c a a c g	1,3	(cabdefghijklmnopqrstuvwxyz)
a c a a c g	1,3,2	(acbdefghijklmnopqrstuvwxyz)
a c a a c g	1,3,2,1	(acbdefghijklmnopqrstuvwxyz)
a c a a c g	1,3,2,1,2	(cabdefghijklmnopqrstuvwxyz)
a c a a c g	1,3,2,1,2,7	(gcabdefhijklmnopqrstuvwxyz)
Final	1,3,2,1,2,7	(gcabdefhijklmnopqrstuvwxyz)

Burrows-Wheeler Transform

- “Last first mapping”：BWT(T)的可逆性
  - 最后一列第i^{th}个出现的字符，是第一列第i^{th}个出现的字符



BWT(T) -> T

从第一行开始：

$ -> g_0

a_0 -> c_1

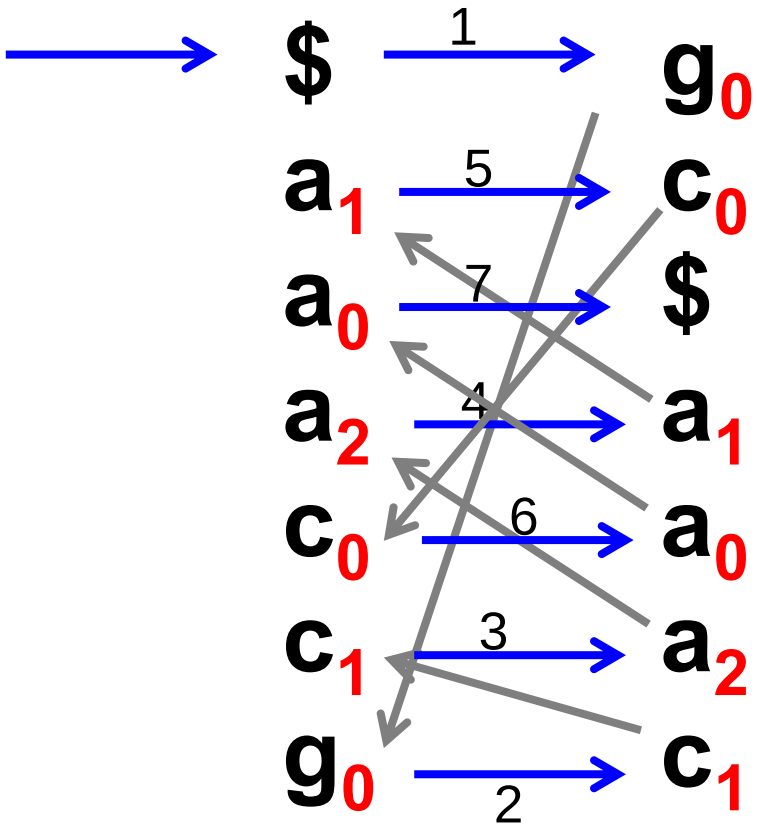
c_0 -> a_2

a_2 -> a_1

a_1 -> c_0

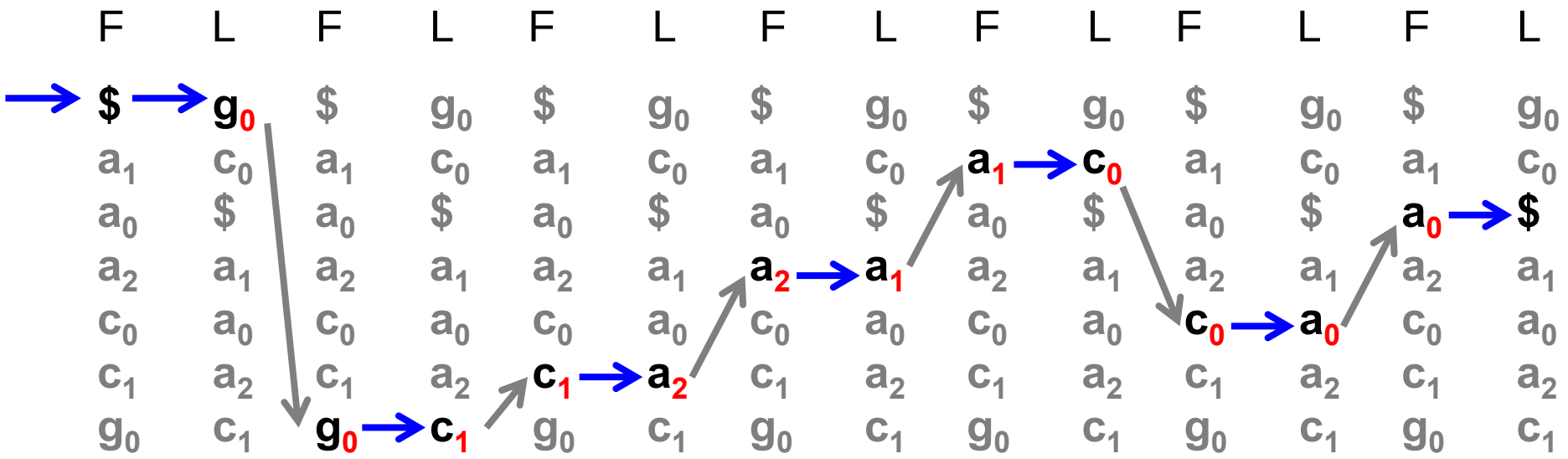
c_0 -> a_0

a_0 -> $



T: a_0 c_0 a_1 a_2 c_1 g_0 $

BWT(T) -> T



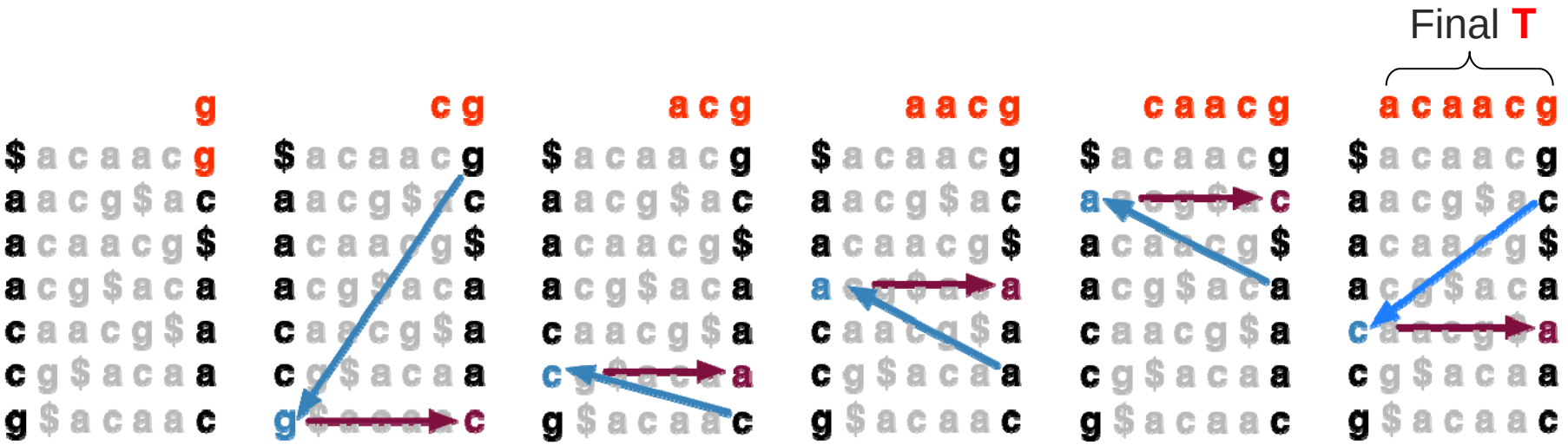
T: a_0 c_0 a_1 a_2 c_1 g_0 $

BWT(T) -> T

- 从BWT(T)重建T, 重复利用下列规则:

  T = BWT[LF(i)] + T; i = LF(i)

  - LF(i) 根据第i行找到第一个字符与第i行匹配的行



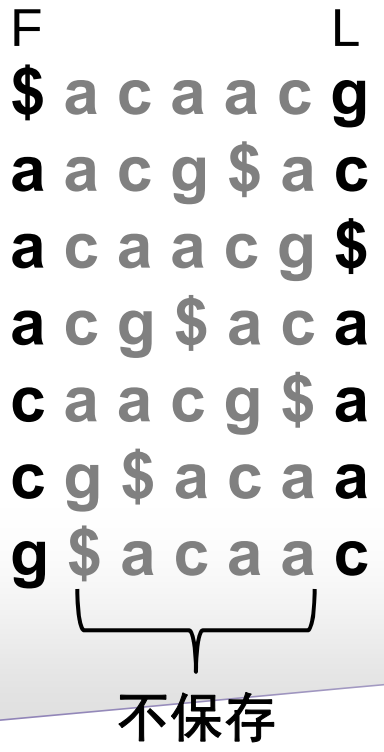
FM索引

- “FM Index”:
  - “FM”: Ferragina & Manzini
  - Full-text index in Minute space（极小空间里的全文索引）
  - 基于BWT转换前后的字符建立的矩阵
- 索引的核心部分: F, L

F可以用数字来代替，例如300, 400, 500, 600，可以表示A_{300}, C_{400}, G_{500}, T_{600}

L可以被压缩

因此占用的空间较小

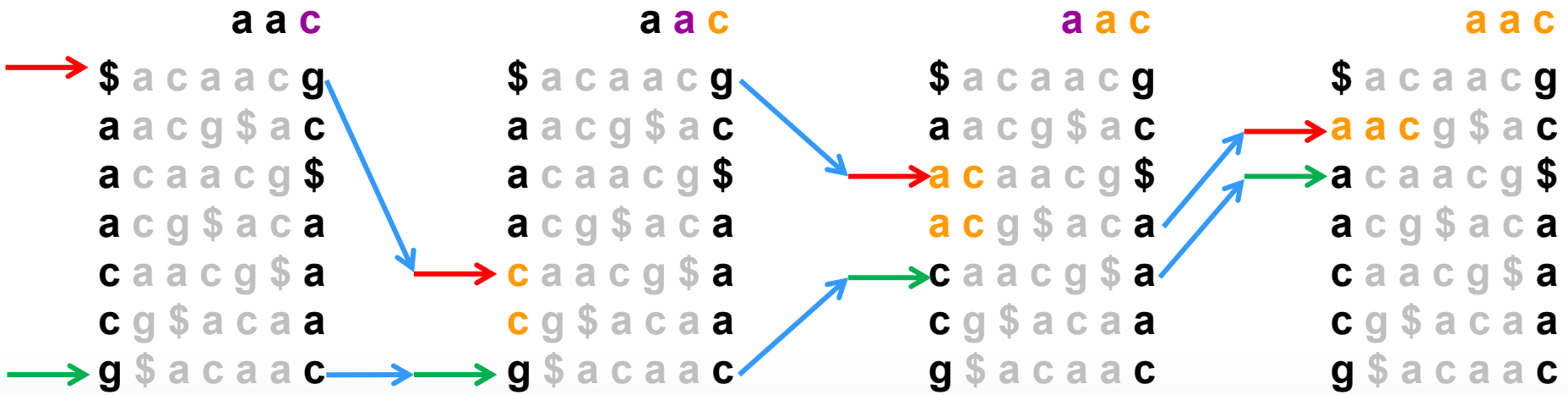


基于FM索引的精确匹配

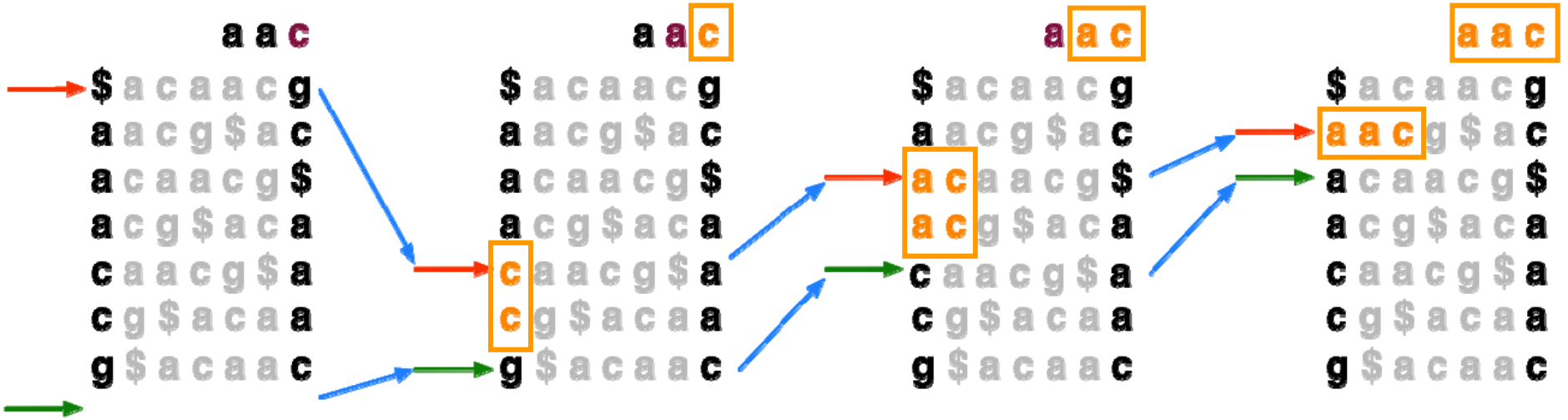
- 重复利用规则：用BWT(T)在T中匹配字符串Q

  top = LF(top, qc); bot = LF(bot, qc)

  - qc是Q (从右到左) 的下一个字符
  - LF(i, qc) 将第i行映射到其最后一个字符与*qc* 相同的行

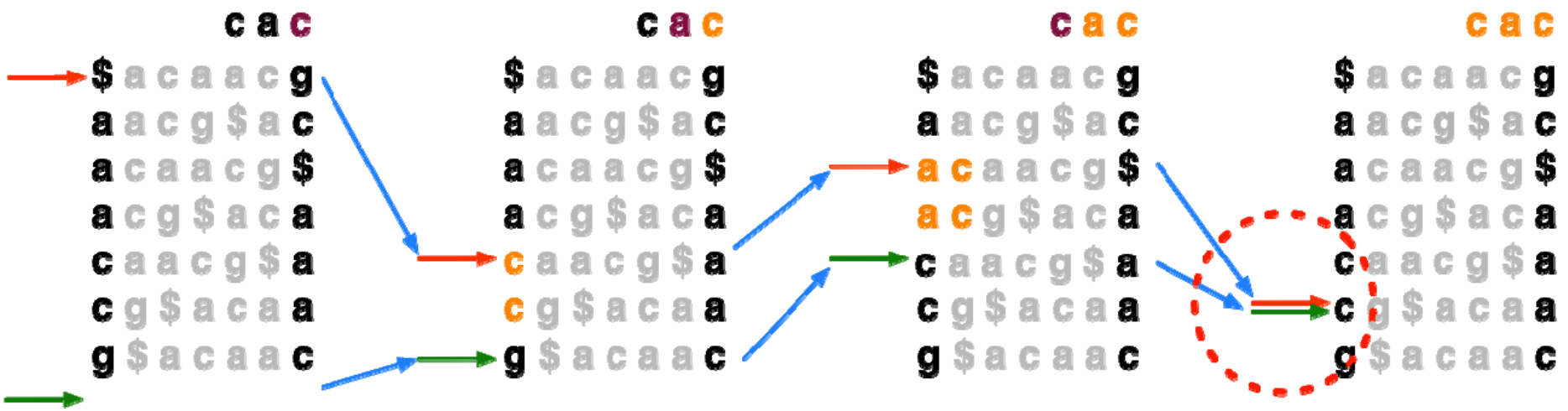


基于FM索引的精确匹配



- “渐进顺序”，top & bot 限制搜索的范围

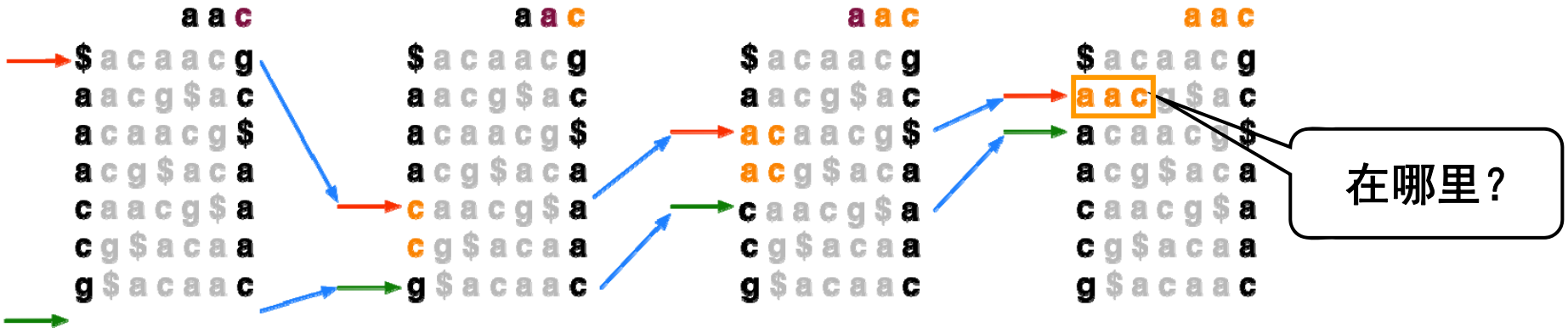
基于FM索引的精确匹配



- 当可选范围为空值 (top = bot)，则表明不能匹配

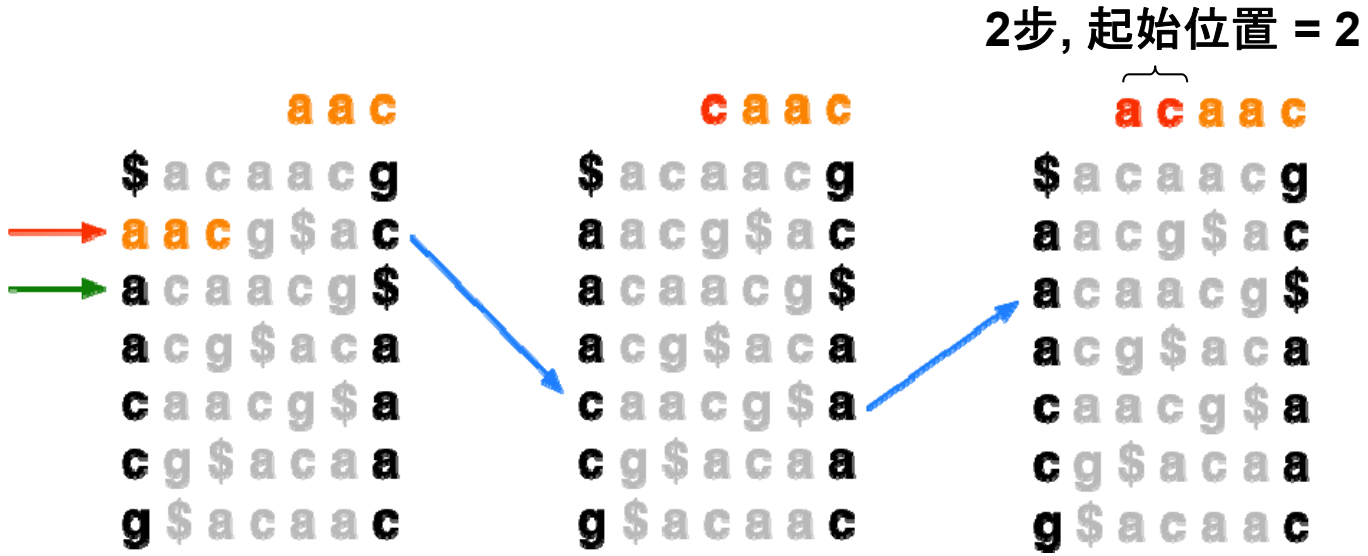
读段在参考序列的位置

- 当比对结束后，如何确定读段在参考序列上的位置？



读段在参考序列的位置

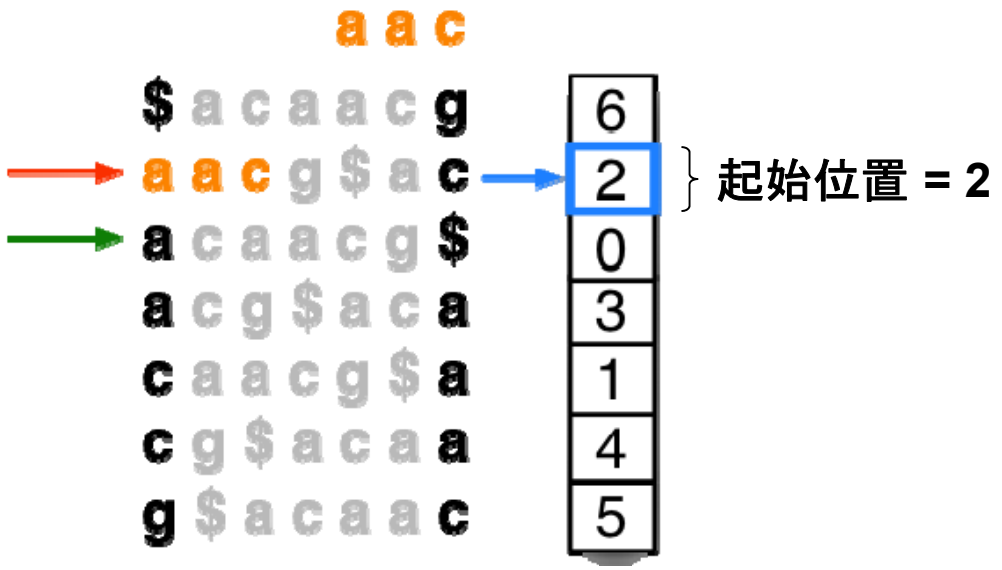
- 方案1：从当前位置一直回读到文本的起始
- 步数 = 起始位置



- 时间复杂度~O(N)，与文本长度线性相关 - 太慢

读段在参考序列的位置

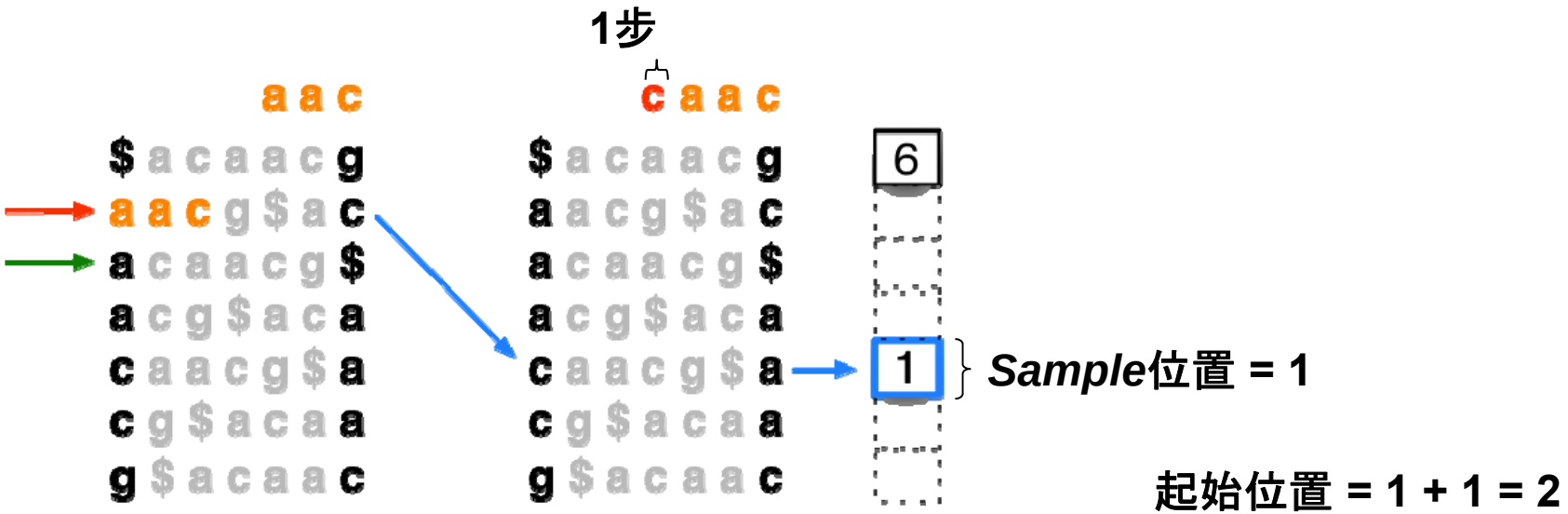
- 方案2：将全部的后缀数组载入到内存，直接从数组中找到参考序列上的起始位置



- 人类基因组的后缀数组约为12Gb – 太大

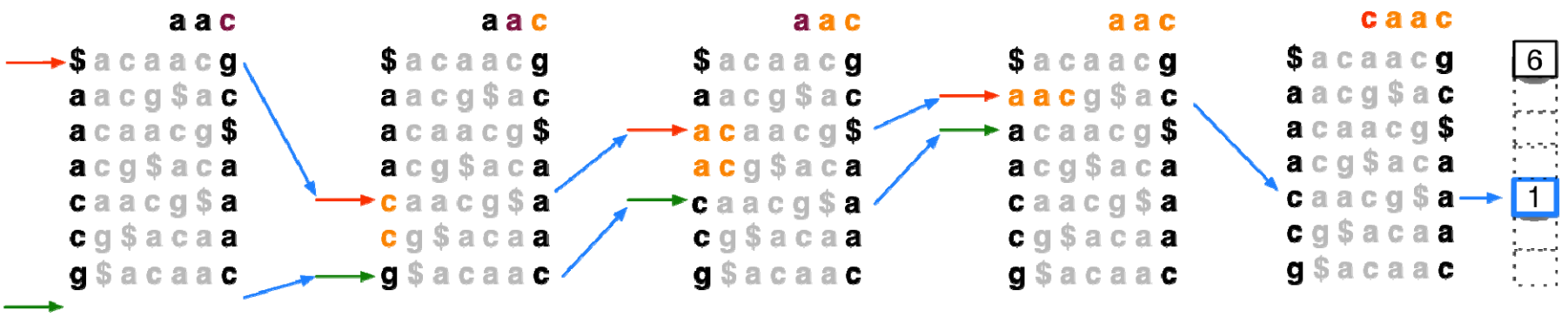
读段在参考序列的位置

- 混合方案：保留部分后缀数组的内容 – “*Sample*”
- 仅需要回读到*sample*位置



- Bowtie: 每32行设置一个*sample*位置值
- 内存开销：12Gb/32 =~ 400Mb

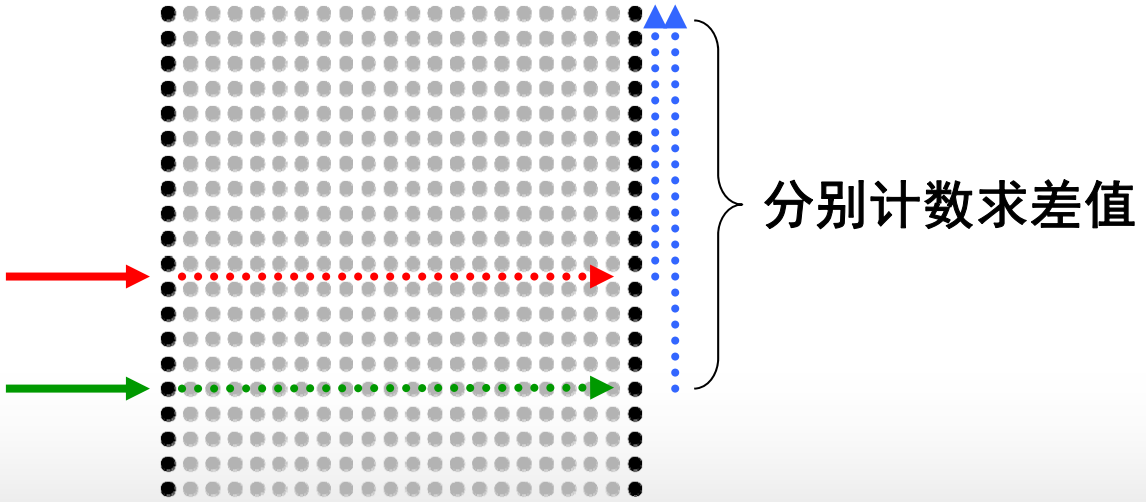
合并



- 算法确定: “aac”在“acaacg”参考序列上的起始位置为2

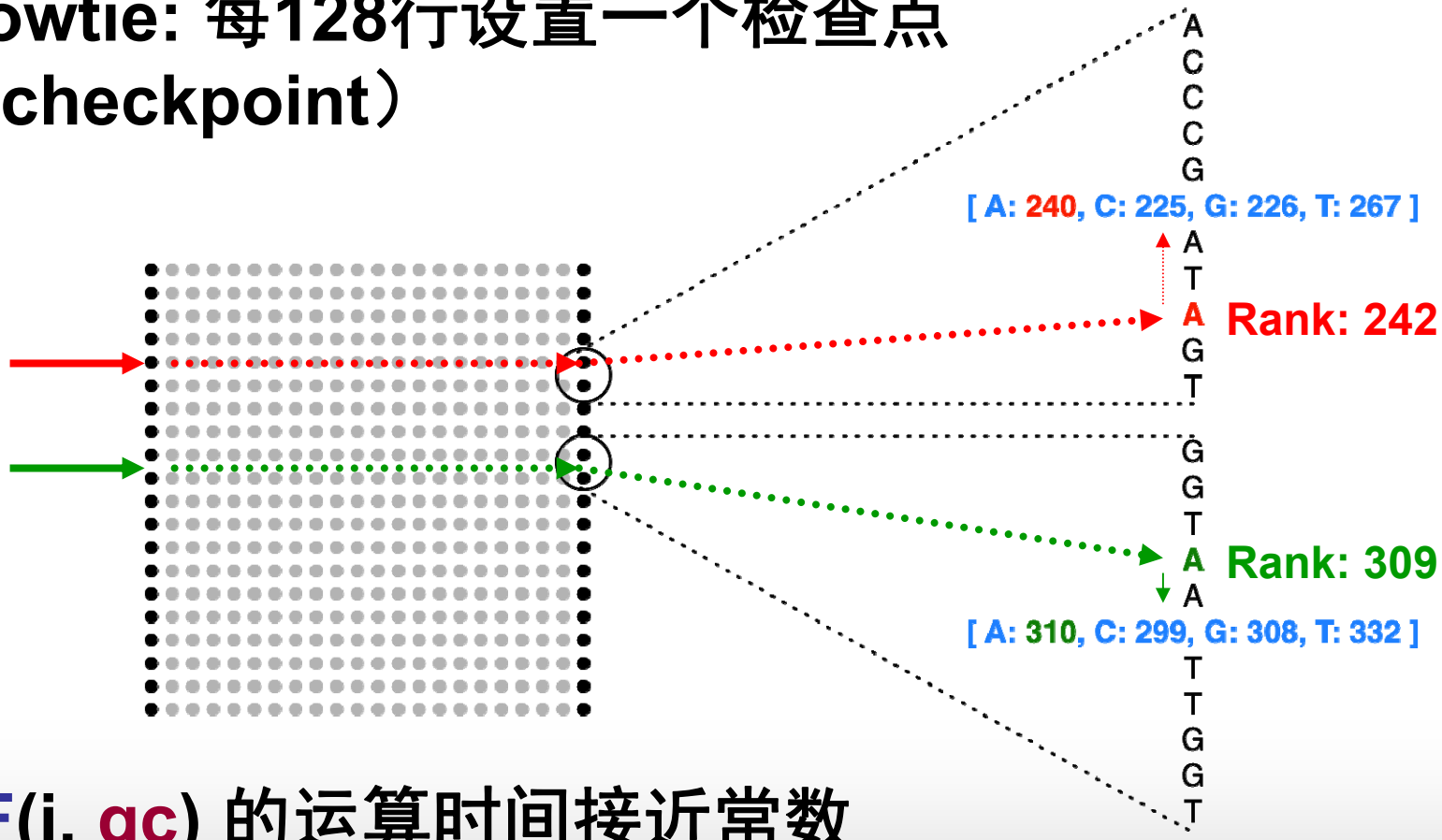
FM索引的检查点

- LF(i, qc)：需要确定第i行qc的次序
- 计数：先数top, qc的次序，再数bot, qc的次序，求差值
- LF(i, qc)的时间复杂度~O(N)，与文本长度线性相关 - 太慢



FM索引的检查点

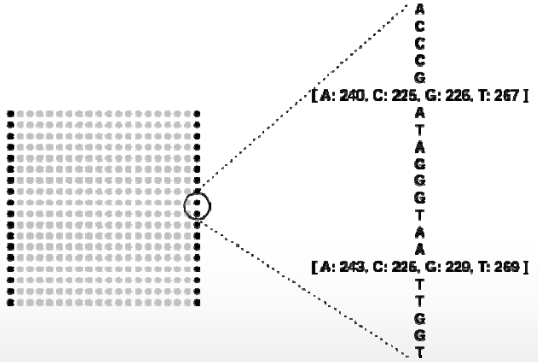
- 方案：预先算好累积的A/C/G/T个数
- Bowtie: 每128行设置一个检查点（checkpoint）



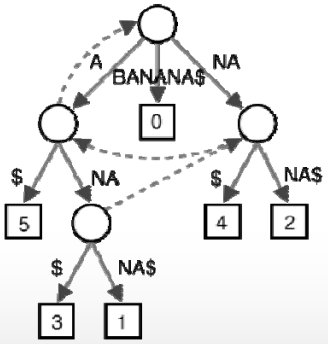
- LF(i, qc) 的运算时间接近常数

FM索引需要的空间较小

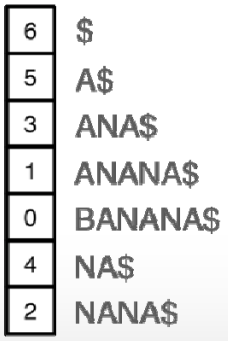
- 人类基因组 ： $3*10^9$字符
  - First column (F)：4 * 4 integer = 16 bytes
  - Last column (L)/BWT：2 bits * $3*10^9$ = 1/4 bytes $3*10^9$ = 750 Mb
  - SA sample： 4 bytes * $3*10^9$ * 1/32 = ~400Mb
  - Checkpoints：4 bytes * $3*10^9$ * 1/128 = ~100Mb
- Total: ~1.65x the size of T (~ 1.25Gb)



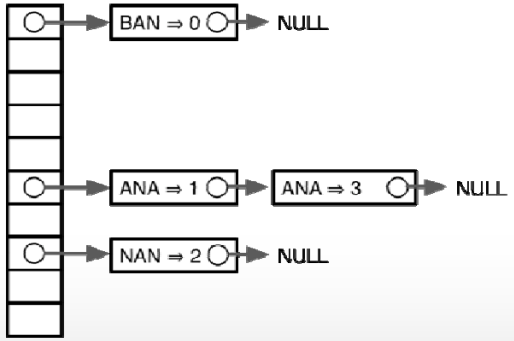
~1.65x



>45x



>15x



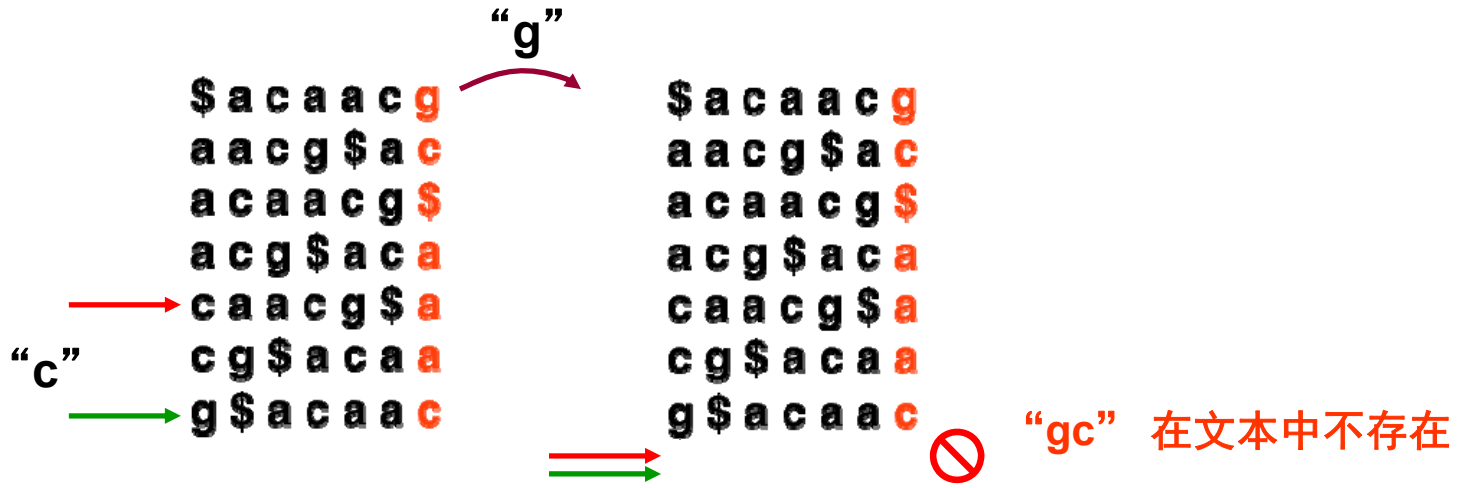
>15x

读段比对

- FM索引：快速获得精确的序列匹配，占用内存小
- 读段比对还需要：
  - 允许错配
  - 考虑质量分值（Quality value）
- Smith-Waterman算法？
  - 比BLAST慢
- BLAST思想：“seed-and-extend”
  - 快一点儿
- Bowtie的解决方案
  - 考虑质量分值的回溯（*backtracking*）搜索

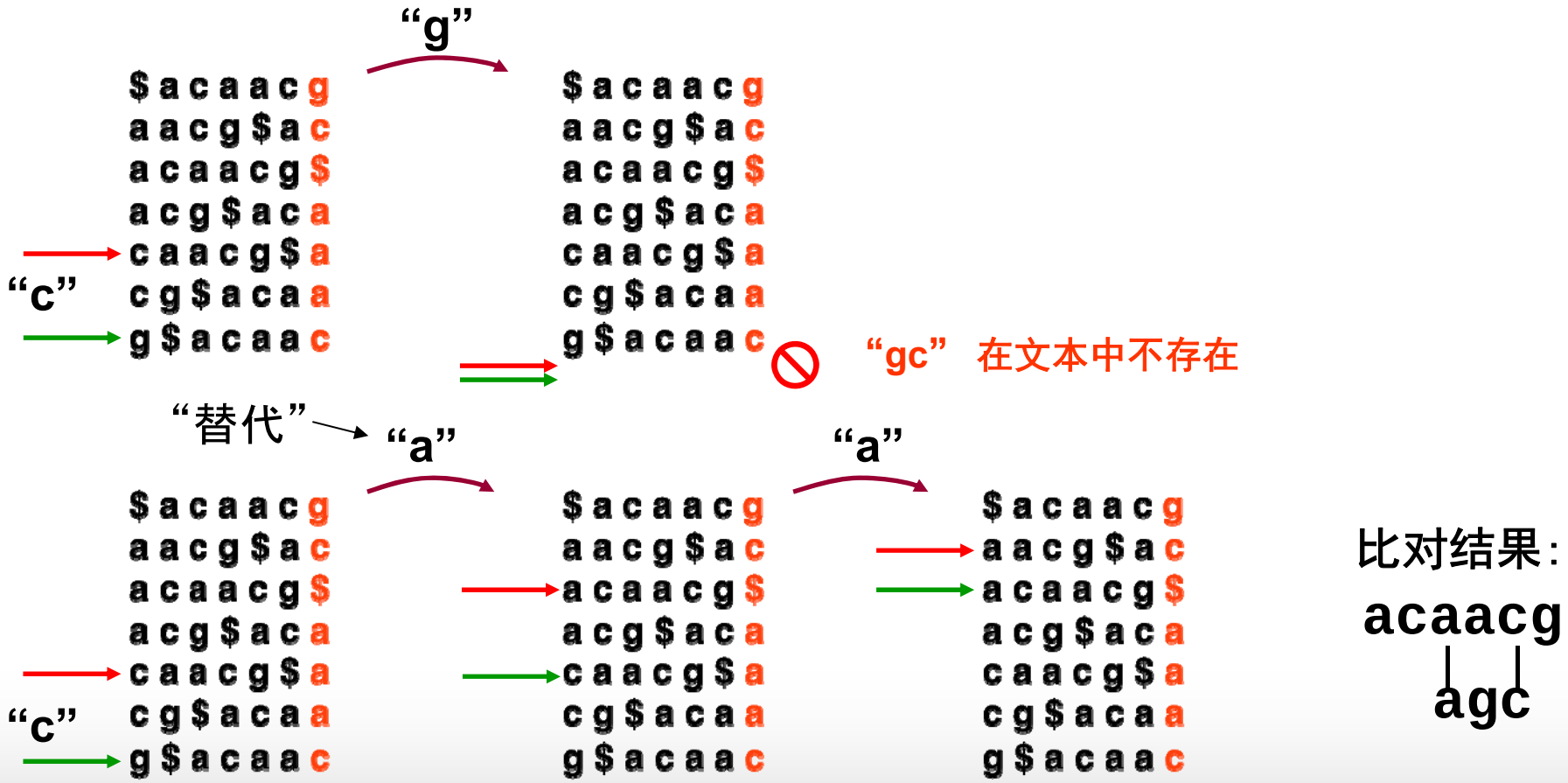
不精确匹配：回溯

- 例如在T = “acaacg” 中搜索Q = “agc”
- “回溯”到之前的位置，并尝试不同的碱基



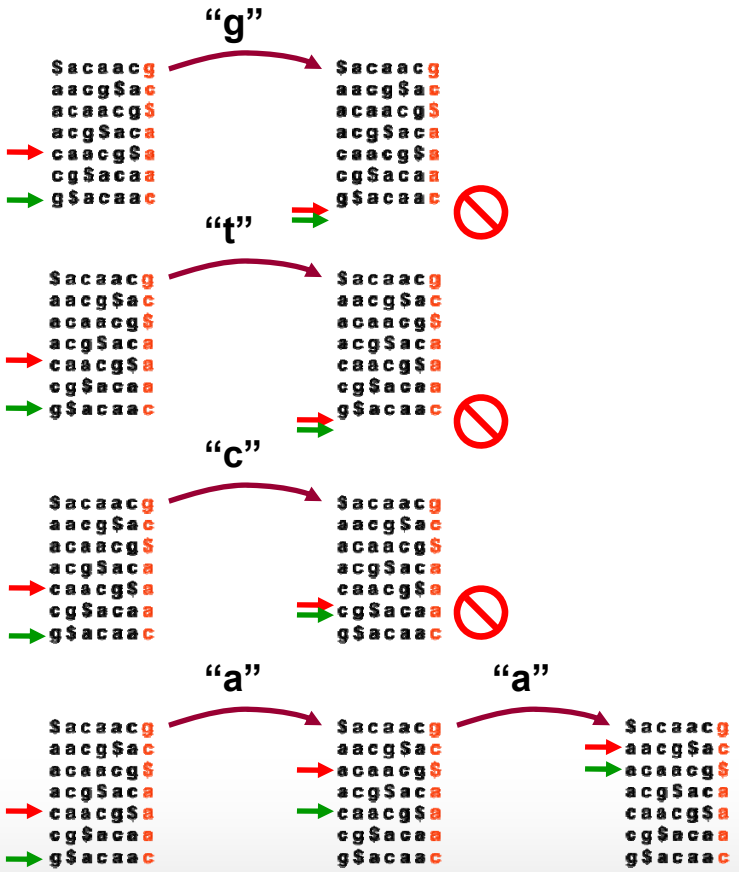
回溯

- 回溯尝试：Q = “agc”, T = “acaacg”:



回溯

- 可能需要多次的回溯尝试



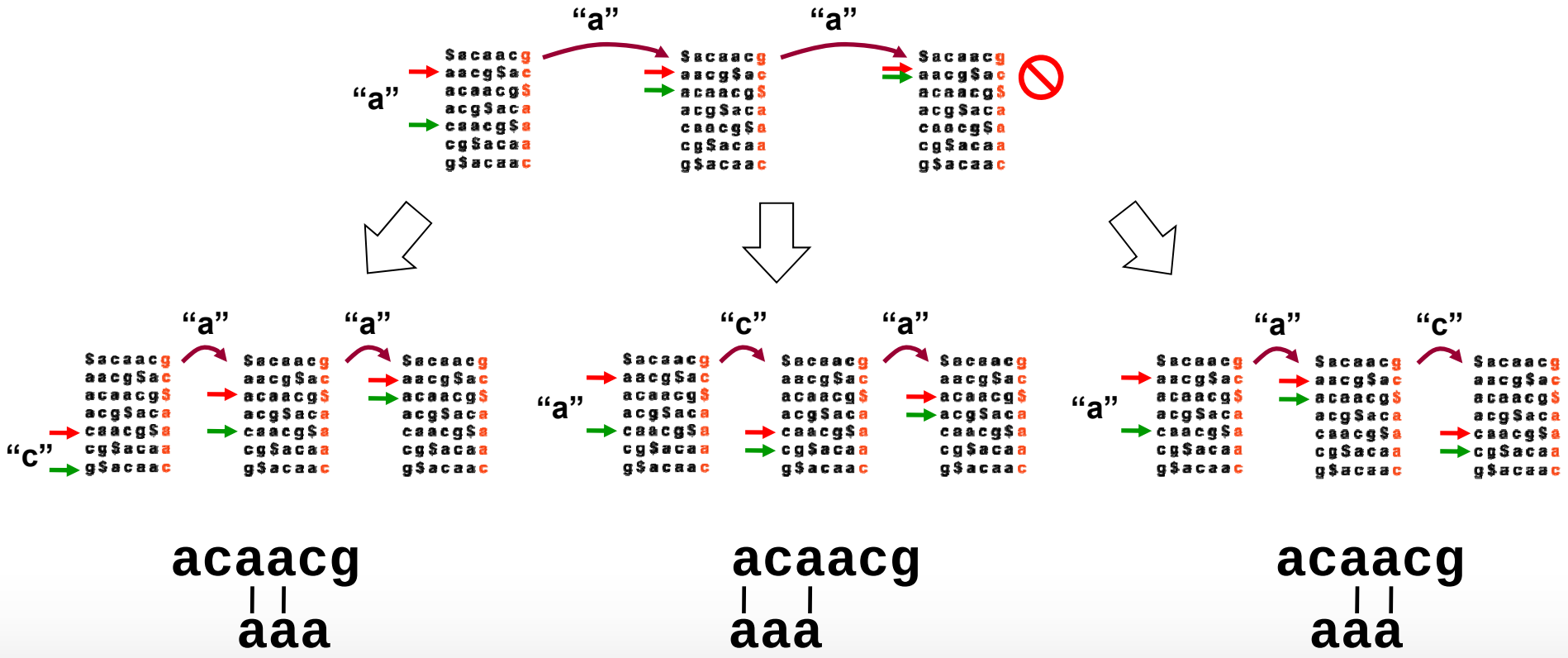
最终的比对结果：

acaacg

agc

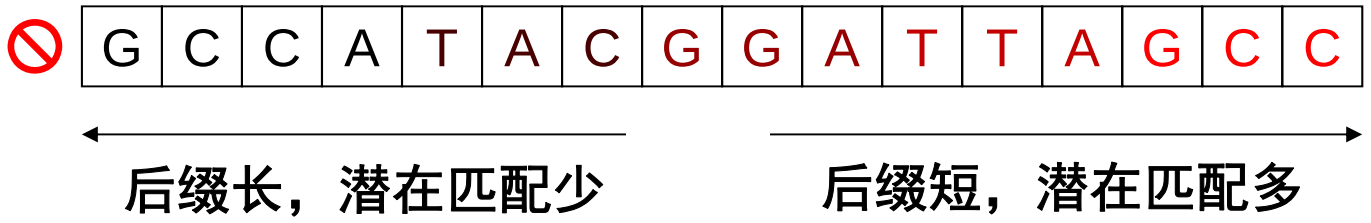
回溯

- 可能存在多个比对结果
  - E.g., Q = “aaa”, T = “acaacg”



额外回溯

- **Bowtie：回溯后希望获得top ≠ bot**
  - 越靠近右端的位置，后缀越短，匹配的潜在空间越大

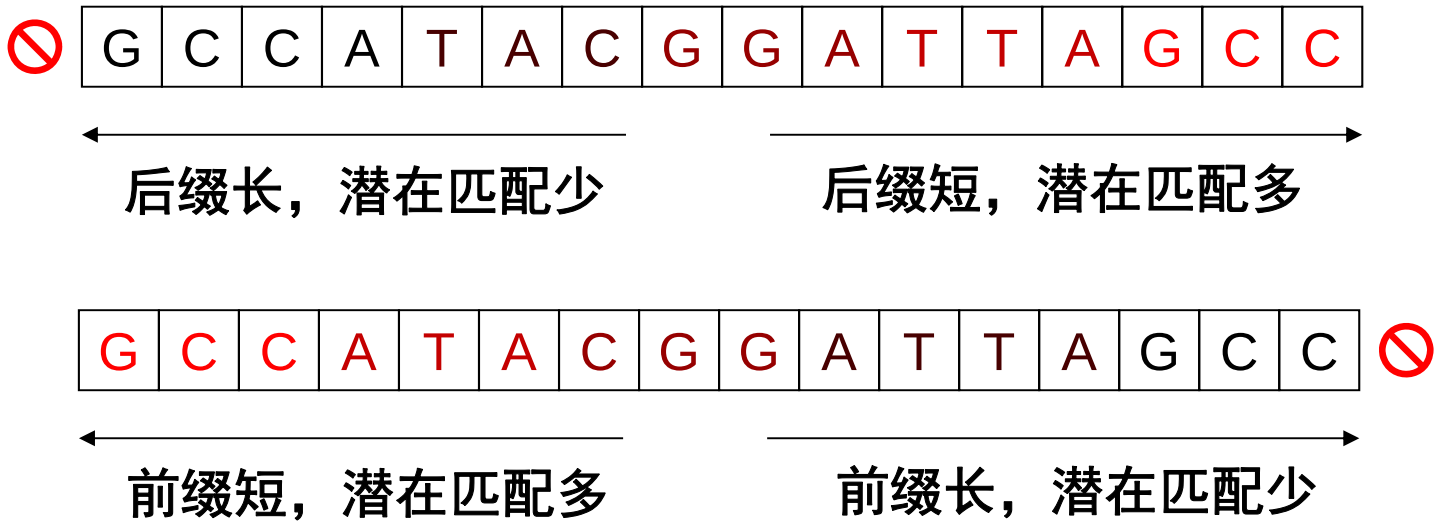


- **当允许 >1 错配时，回溯的速度慢**

额外回溯

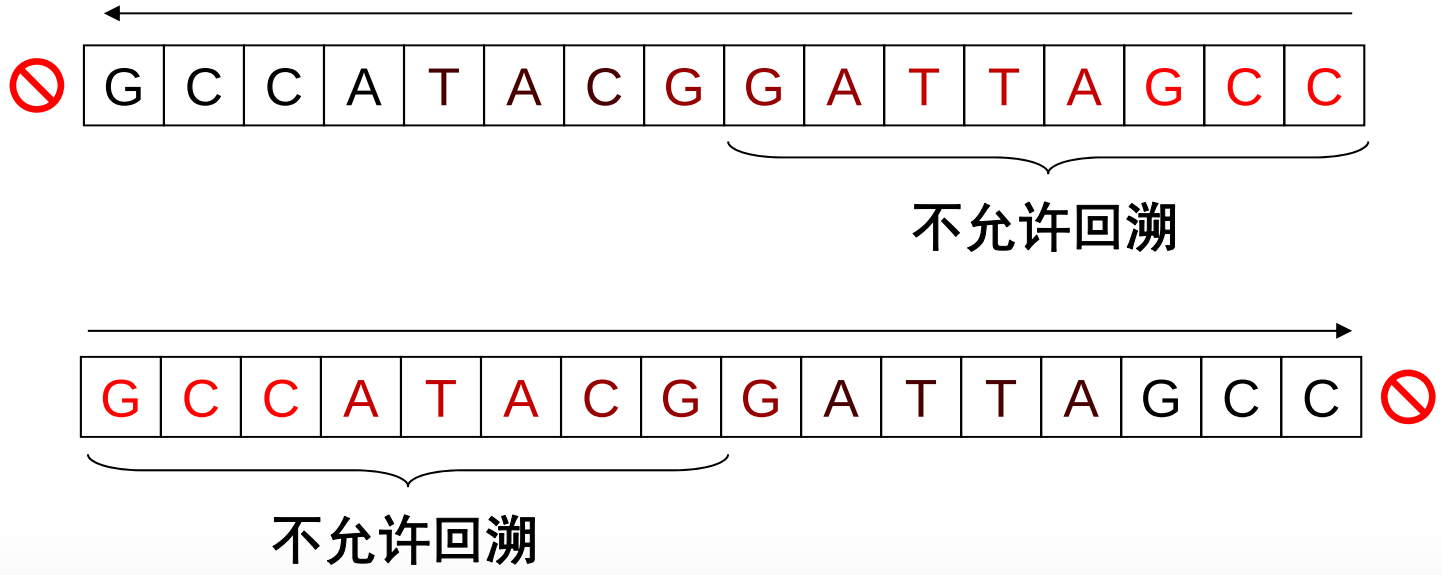
❒ 解决方案：双索引

- 既考虑从右到左的匹配，也考虑从左到右的匹配



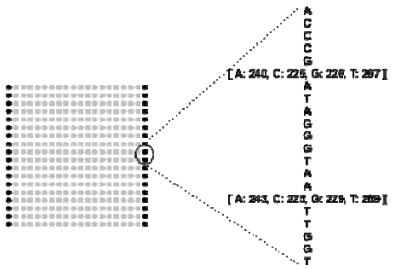
额外回溯

- 限制红色区域的回溯
  - 每一个方向最多允许一个错配
  - 最红色的区域不允许回溯

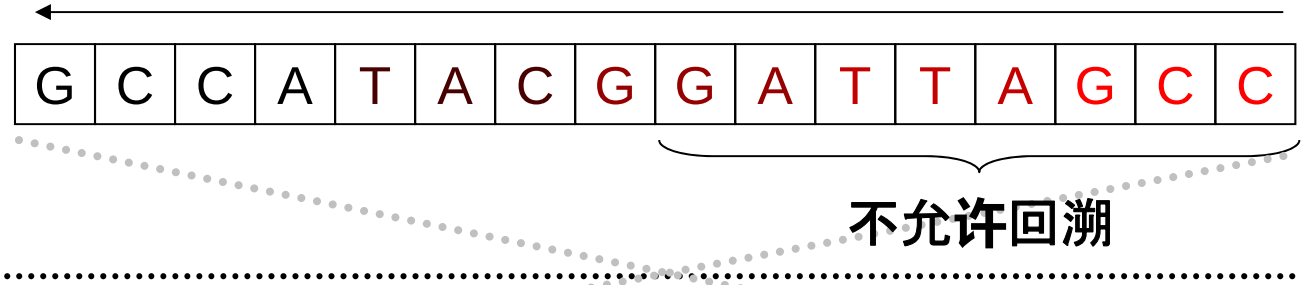


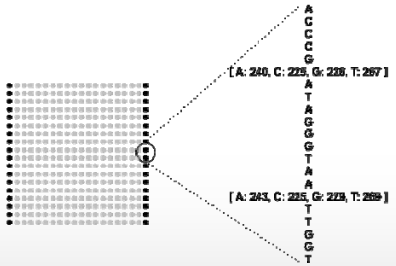
额外回溯

- 如何匹配left-to-right?
- 双索引:
  - 反转读段，利用“镜像索引”：参考序列反转的索引

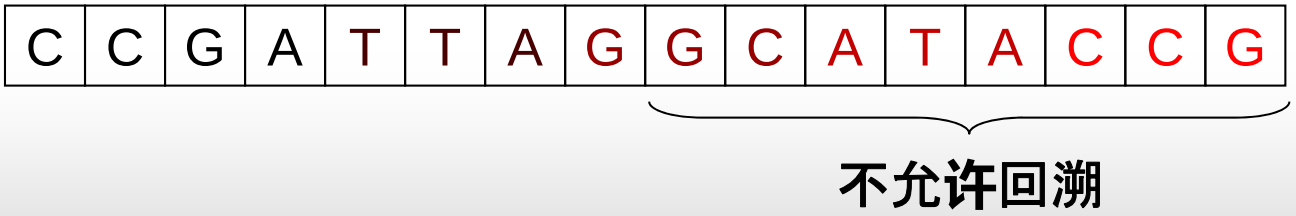


正向索引



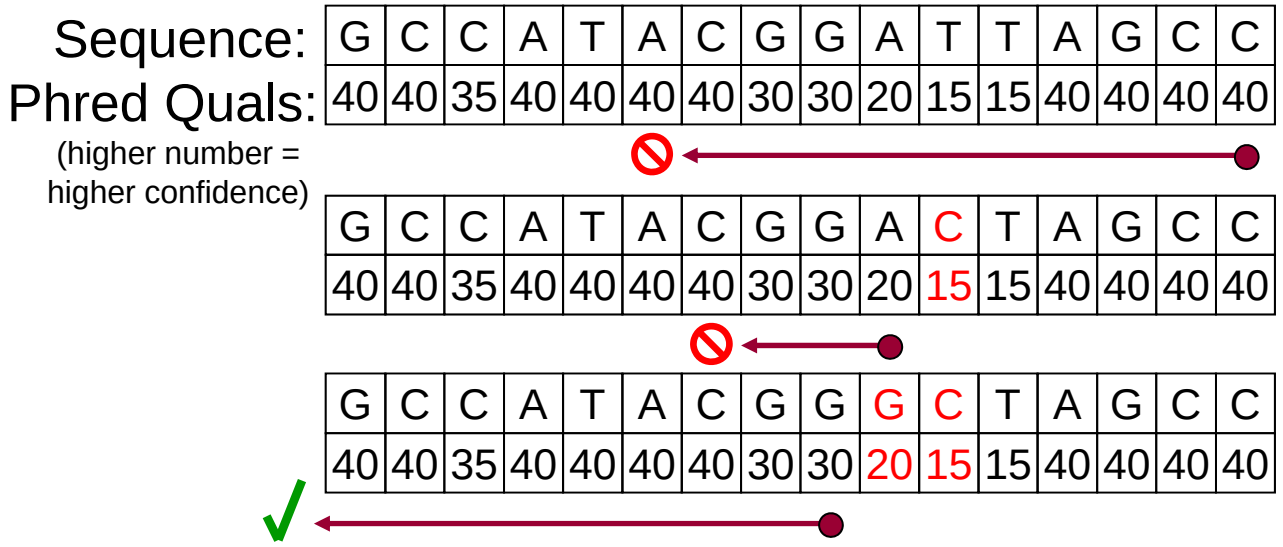


镜像索引



质量分值

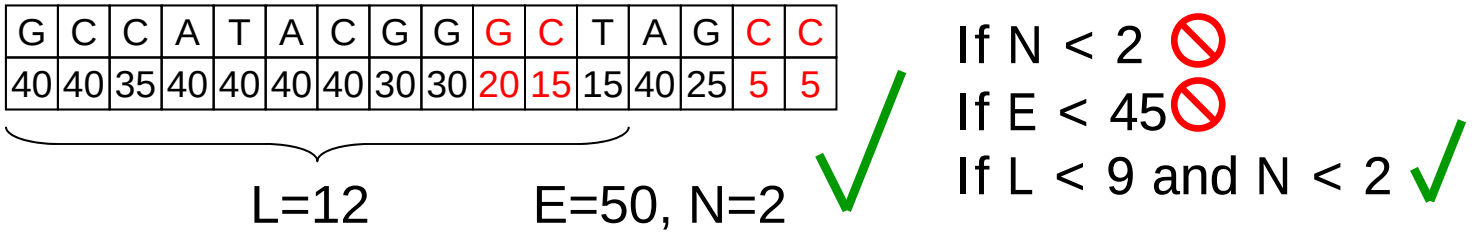
- 若需要回溯：Bowtie从最靠近左边的、质量分值最低的位置开始



质量分值

Bowtie的比对原则

- 允许错配 ≤ N 个
- 在左端 ≤ L个碱基中
- 错配分值之和 ≤ E：尽量让错配发生在质量分值低的区域
- N, L, E: -n, -l, -e



- Bowtie的缺省模式 (N=2, L=28, E=70)

http://bowtie-bio.sourceforge.net/index.shtml

- 使用SeqAn* 库（http://www.seqan.de）
- 利用POSIX 实现并行化

Bowtie
An ultrafast memory-efficient short read aligner

JOHNS HOPKINS UNIVERSITY

Bowtie is an ultrafast, memory-efficient short read aligner. It aligns short DNA sequences (reads) to the human genome at a rate of over 25 million 35-bp reads per hour. Bowtie indexes the genome with a Burrows-Wheeler index to keep its memory footprint small: typically about 2.2 GB for the human genome (2.9 GB for paired-end).

OSI certified

» Recent news

» Lighter released

- Lighter is an extremely fast and memory-efficient program for c
  on how error correction can help improve the speed and accura
  Biology. Source and software available at GitHub.

» 1.1.1 - 10/1/2014

- Fixed a compiling linkage problem related with Mac OS X Maver
- Improved performance for cases where the reference contains
- Some minor automatic tests updates.

» 1.1.0 - 7/19/2014

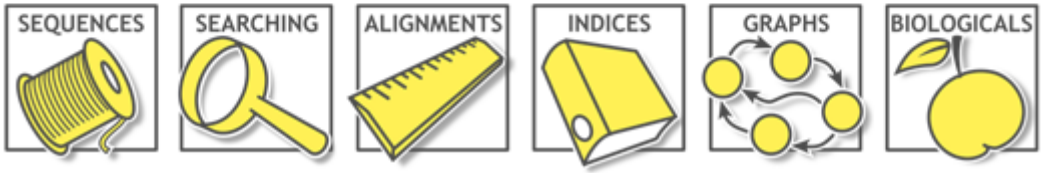
- Added support for large and small indexes, removing 4-billion-n
  genomes of any size.
- No longer releasing 32-bit binaries. Simplified manual and Makef
- Phased out CygWin support.
- Improved efficiency of index files loading.
- Fixed a bug that made `bowtie-inspect` fail in some situations.
- (This release was briefly given version number 2.0.0, but we ch

SeqAn » Home

Home

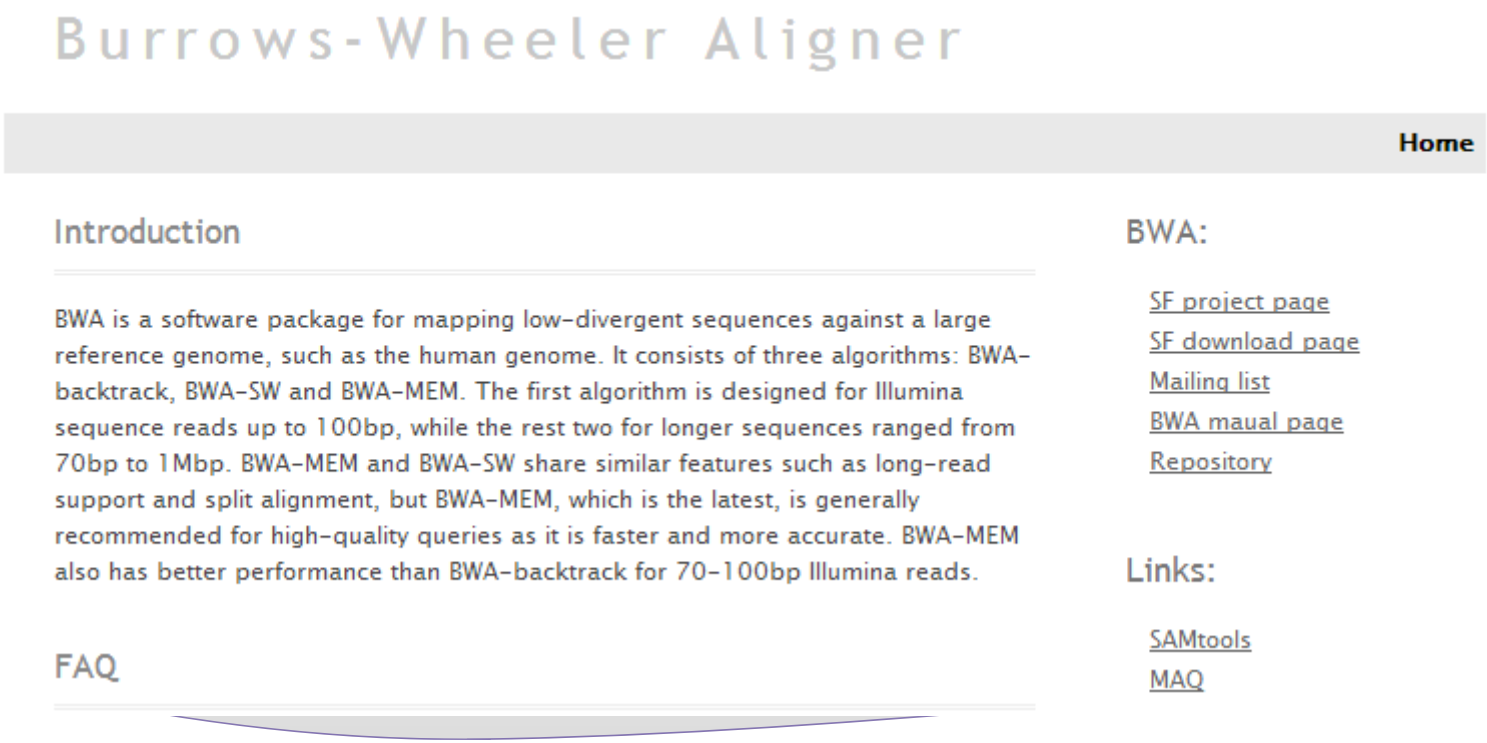
What is SeqAn?

SeqAn is an open source C++ library of efficient algorithms and data structures for the analysis of sequences with the focus on biological data. Our library applies a unique generic design that guarantees high performance, generality, extensibility, and integration with other libraries. SeqAn is easy to use and simplifies the development of new software tools with a minimal loss of performance.



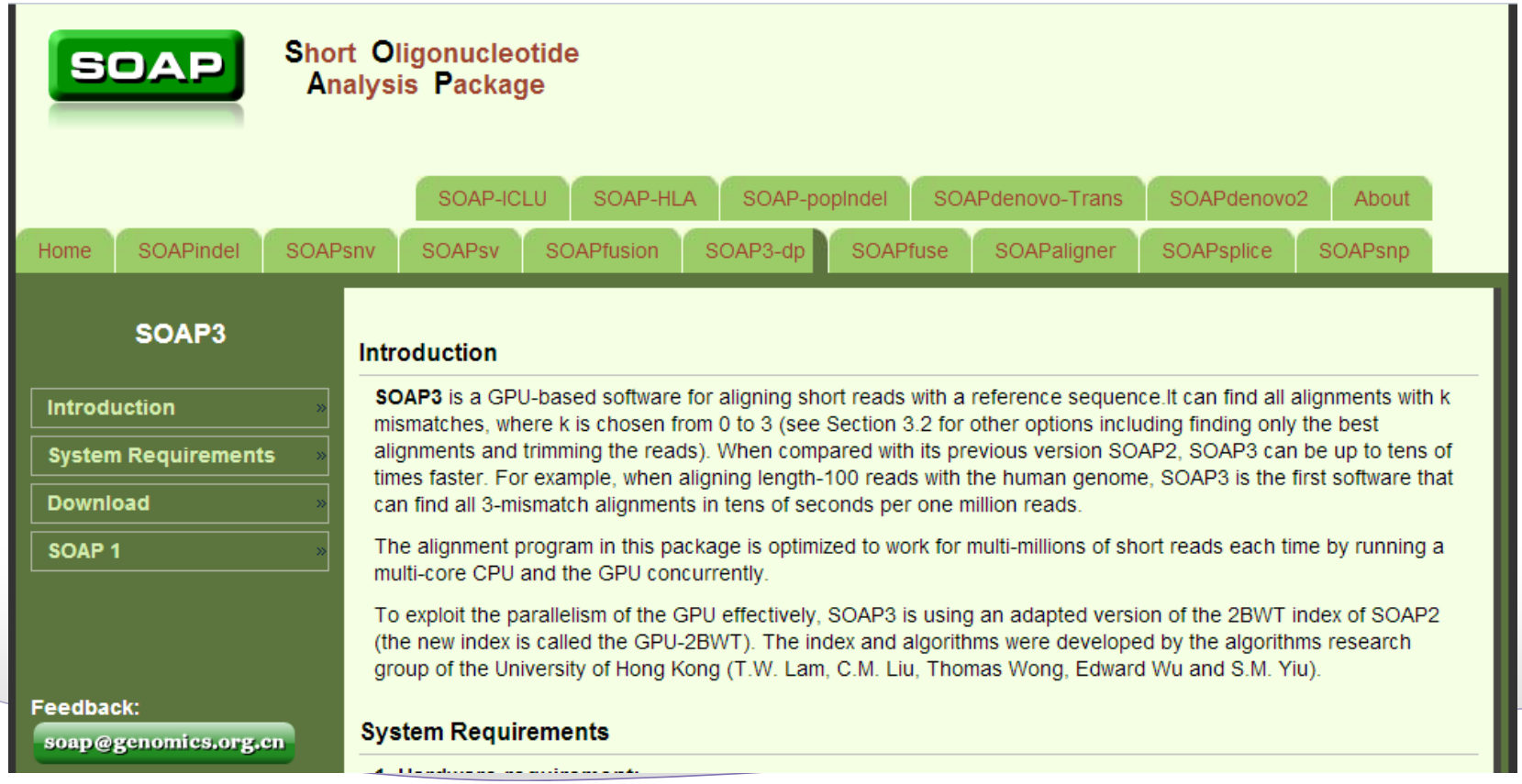
Burrows-Wheeler Aligner (BWA)

- http://bio-bwa.sourceforge.net
- BWT + 后缀矩阵
- 最多3个错配



Short Oligonucleotide Analysis Package (SOAP3)

- 利用GPU实现并行化计算
- 最多4个错配
- BWT + FM索引



性能比较

- 读段的Mapping率SOAP3最高
- 运算时间：
  - 3/4个错配：SOAP3比BWA快 >6倍，Bowtie最慢
  - 0/2个错配：Bowtie比BWA快，SOAP3比Bowtie快

Table 1. Running time and percentage of aligned reads of SOAP3, BWA and Bowtie on two paired-end datasets with 70.7 M (HiSeq 2000) and 25.3 M (Genome Analyzer II) read pairs. Each time reported below includes index loading time, read loading time (259 sec for 70.7M; 107 sec for 25.3M) and the alignment time. For BWA, we opt for faster speed by disabling the gapped alignment.

		4 mismatches		3 mismatches		2 mismatches		1 mismatch		0 mismatch	
Dataset		seconds	%	seconds	%	seconds	%	seconds	%	seconds	%
70.7 M read pairs	SOAP3	1839	81.46%	1019	79.43%	695	76.48%	521	70.94%	452	53.11%
	BWA	13756	81.03%	10590	79.19%	8920	76.38%	6064	70.90%	5272	53.11%
	Bowtie	not supported		29178	79.42 %	2082	76.47%	1570	70.93%	1216	53.10%
25.3 M read pairs	SOAP3	735	86.58%	453	85.34%	356	83.46%	291	79.17%	266	60.30%
	BWA	4700	86.17%	3803	85.12%	3298	83.38%	2352	79.17%	1800	60.30%
	Bowtie	not supported		9486	85.33%	1431	83.45%	617	79.17%	484	60.13%