13-C5

串

KMP算法：分摊分析

失之东隅，收之桑榆

幸好过了一个冬天那女人又来了，两个人仍是逆时针绕着园子走，一长一短两个身影恰似钟表的两支指针

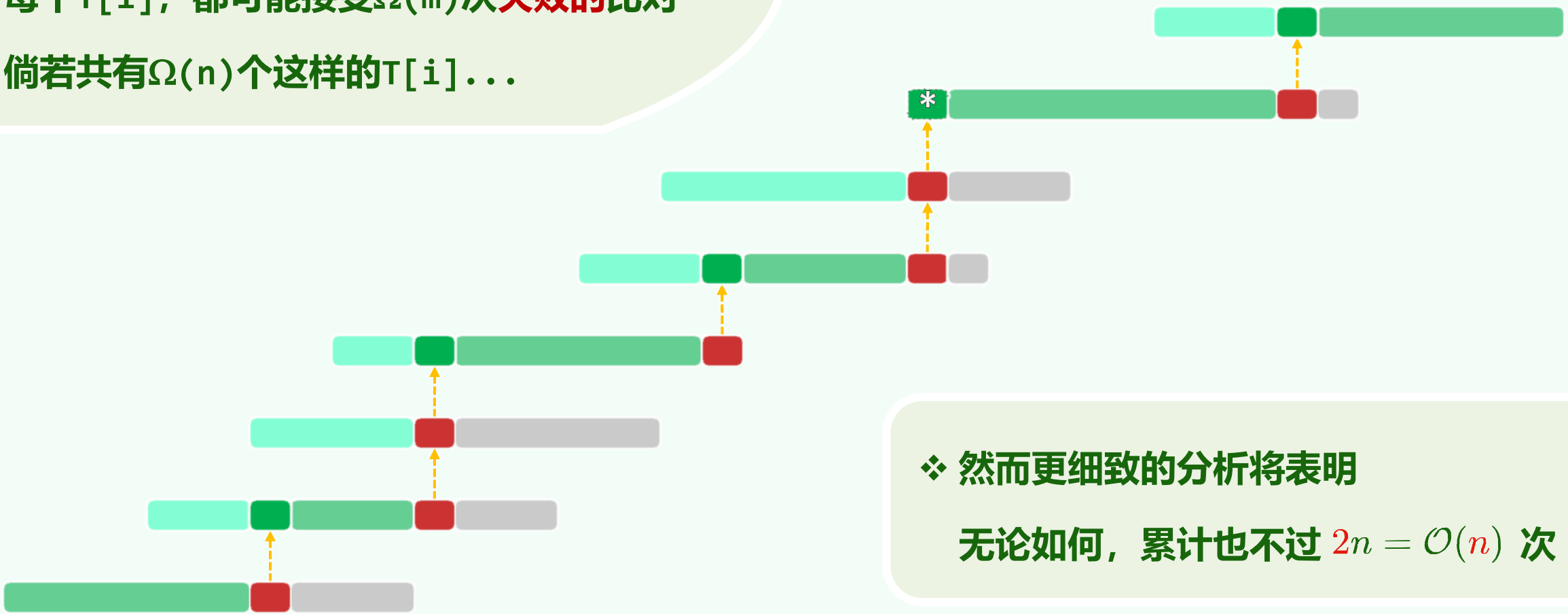
邓俊辉

deng@tsinghua.edu.cn

Ω(n*m)?

❖ 每个T[i]，都可能接受Ω(m)次失败的比对

倘若共有Ω(n)个这样的T[i]...



❖ 然而更细致的分析将表明

无论如何，累计也不过 $2n = \mathcal{O}(n)$ 次

O(n + m) by Aggregate

❖ 令: k = 2*i - j //虽欠精准，但还算够用的计步器

```
while ( (j < m) && (i < n) ) //k必随迭代而单调递增，故也是迭代步数的上界
```

```
if ( 0 > j || T[i] == P[j] )
   { i ++;  j ++; } //k恰好加1
```

```
else
   j = next[j]; //k至少加1
```

❖ 初始: k = 0

算法结束时: k = 2*i - j ≤ 2(n - 1) - (-1) = 2n - 1

O(n + m) by Accounting (1/2)

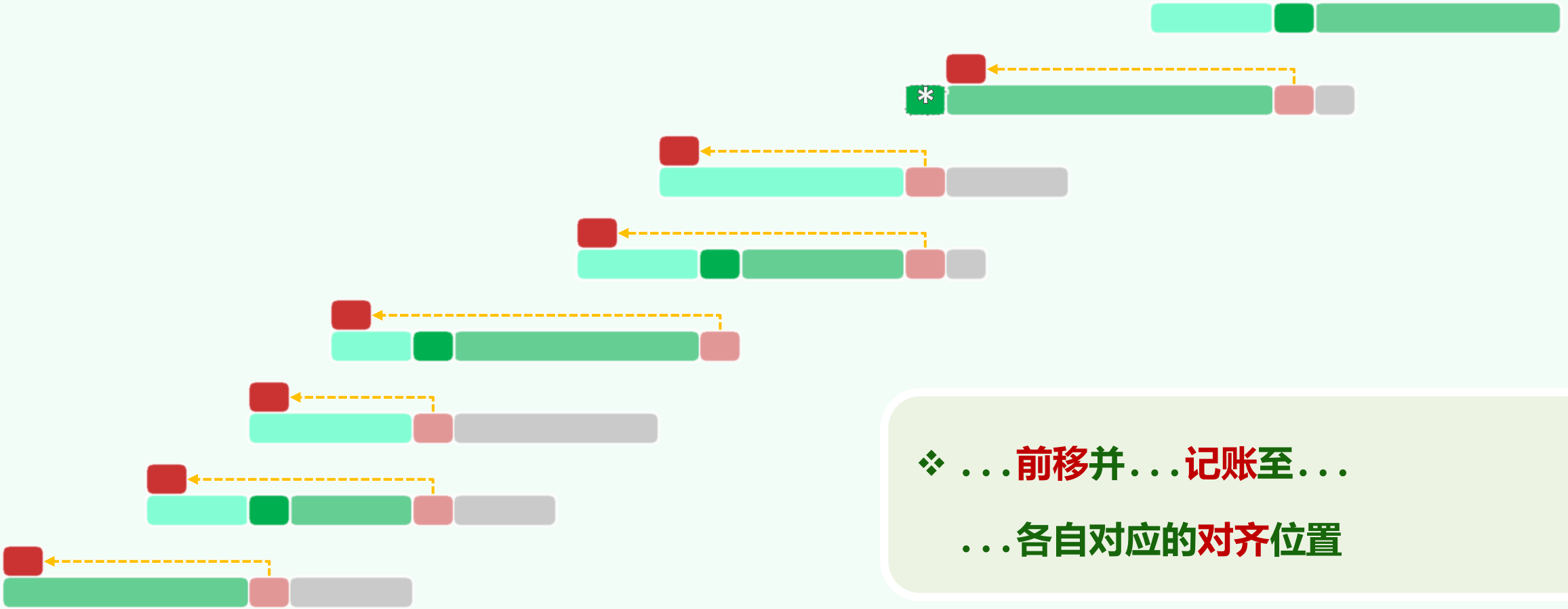
❖ 成功的比对：恰好覆盖T串，共计O(n)次

❖ 失败的比对：在同一位置i或有多个

不能通过覆盖简明计数

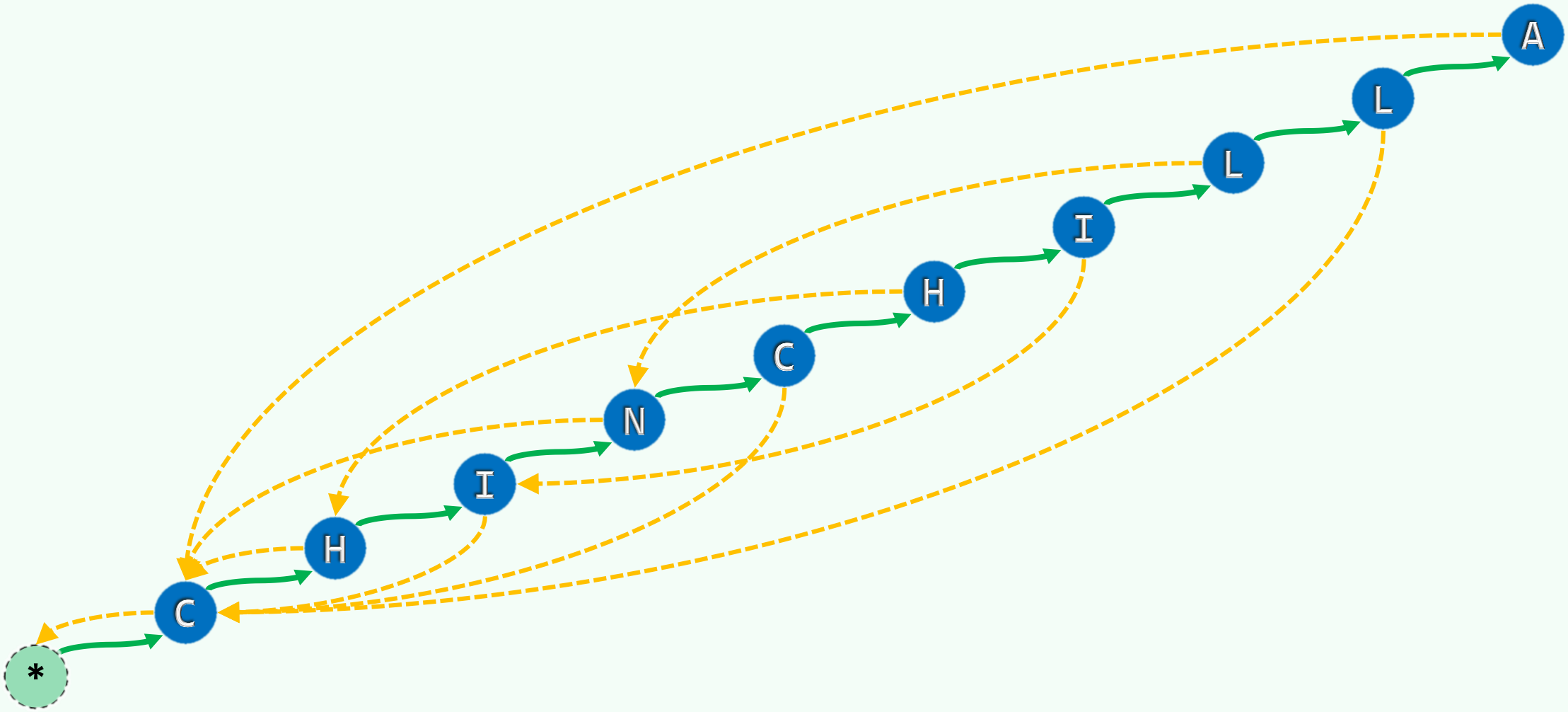
但可以...

$O(n + m)$ by Accounting (2/2)



- . . . 前移并 . . . 记账至 . . .
  . . . 各自对应的对齐位置

每一个P串，都是一台自动机



模式串 ~ 匹配算法

```
int match( char * T ) { //对任一模式串（比如P = chinchilla），可自动生成如下代码
   int n = strlen(T); int i = -1; //文本串对齐位置
s_: ++i;                                                          // ↕
s0: (T[i] != 'C') ? goto s_ : if (n <= ++i) return -1;            // *          ~ ↕
s1: (T[i] != 'H') ? goto s0 : if (n <= ++i) return -1;            // *C         ~ *
s2: (T[i] != 'I') ? goto s0 : if (n <= ++i) return -1;            // *CH        ~ *
s3: (T[i] != 'N') ? goto s0 : if (n <= ++i) return -1;            // *CHI       ~ *
s4: (T[i] != 'C') ? goto s0 : if (n <= ++i) return -1;            // *CHIN      ~ *
s5: (T[i] != 'H') ? goto s1 : if (n <= ++i) return -1;            // *CHINC     ~ *C
s6: (T[i] != 'I') ? goto s2 : if (n <= ++i) return -1;            // *CHINCH    ~ *CH
s7: (T[i] != 'L') ? goto s3 : if (n <= ++i) return -1;            // *CHINCHI   ~ *CHI
s8: (T[i] != 'L') ? goto s0 : if (n <= ++i) return -1;            // *CHINCHIL  ~ *
s9: (T[i] != 'A') ? goto s0 : if (n <= ++i) return -1;            // *CHINCHILL ~ *
   return i - 10;                                                 // *CHINCHILLA
}
```