Mining Data Streams (Part 1)

海量数据集的挖掘

Queries over a (long) Sliding Window

Sliding Windows

- 一个有用的流处理模型是，查询是关于一个长度为***N*的*窗口***--最近收到的***N*个**元素。
- **有趣的情况：*N*是如此之大，以至于数据不能存储在内存中，甚至不能存储在磁盘上**
  - 或者，有如此多的流，无法存储所有的窗口

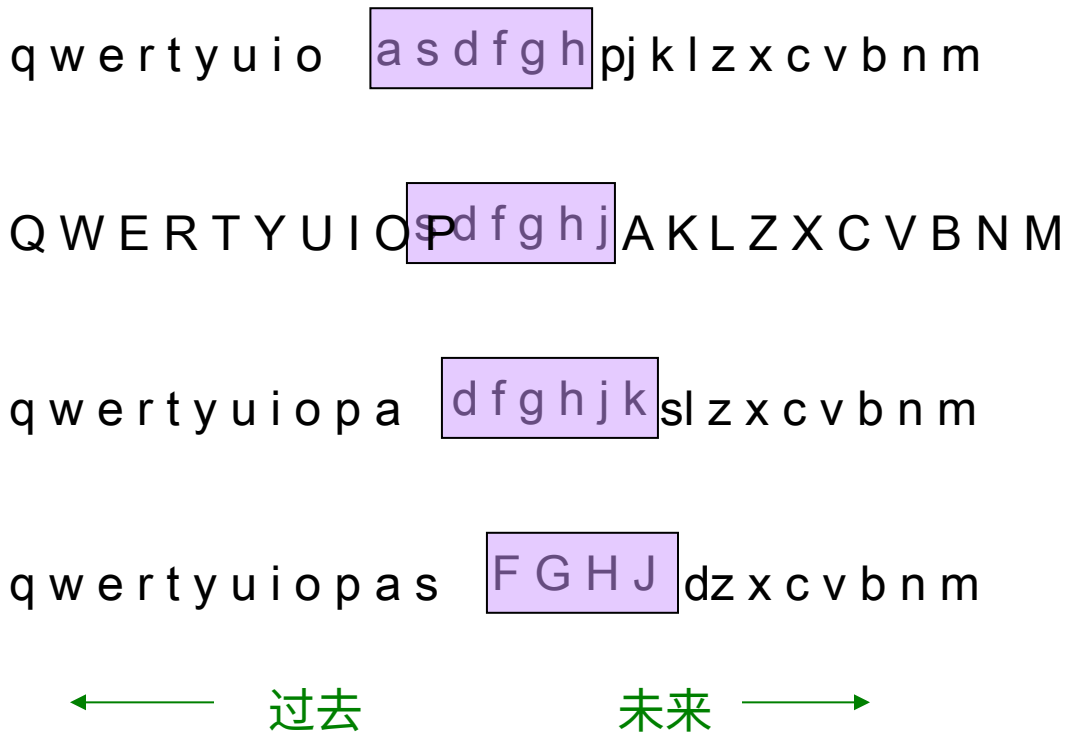
亚马逊的例子：

- 对于每个产品**X**，我们保持0/1的流，即该产品是否在**第n次**交易中被售出。
- 我们要回答的是，在过去的**k次**销售中，我们卖了多少次**X?**

Sliding Window: 1 Stream

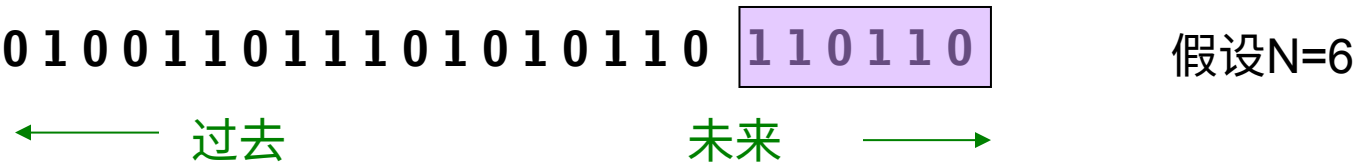
- **单流的滑动窗口：**

N = 6



Counting Bits (1)

- **问题：**
  - 给出一个0和1的流
  - 准备好回答以下形式的询问
    最后*k*位有多少个1？ 其中***k*≤*N***
- **明显的解决方案：**
  存储最近的***N***位
  - 当有新位进入时，丢弃***第N+1位*** 位



Counting Bits (2)

- 如果不存储整个窗口，你就无法得到一个准确的答案

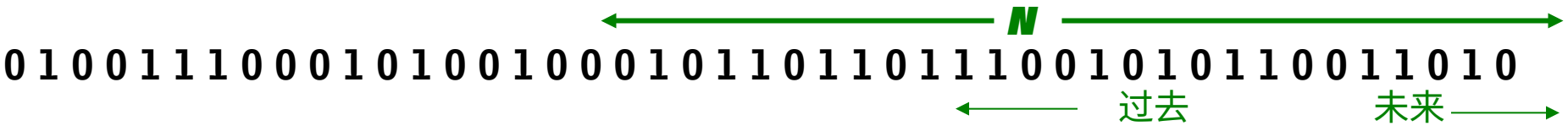
- **真正的问题：**
  如果我们没有能力存储*N* 个比特怎么办？
  - **例如**，我们正在处理10亿个数据流和 ***N*** **= 10亿**

- **但我们对一个近似的答案感到满意**

An attempt: Simple solution

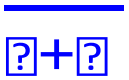
- **问：最后N位中有多少个1?**
- 一个简单的解决方案，并没有真正解决

  我们的问题：**统一性假设**



- **保持2个计数器：**
  - ***S***：从流开始的1的数量
  - ***Z***：从流开始的0的数量
- **最后N位有多少个1？**
- **但是，如果流是不均匀的呢?**

分布随时间变化怎么办?



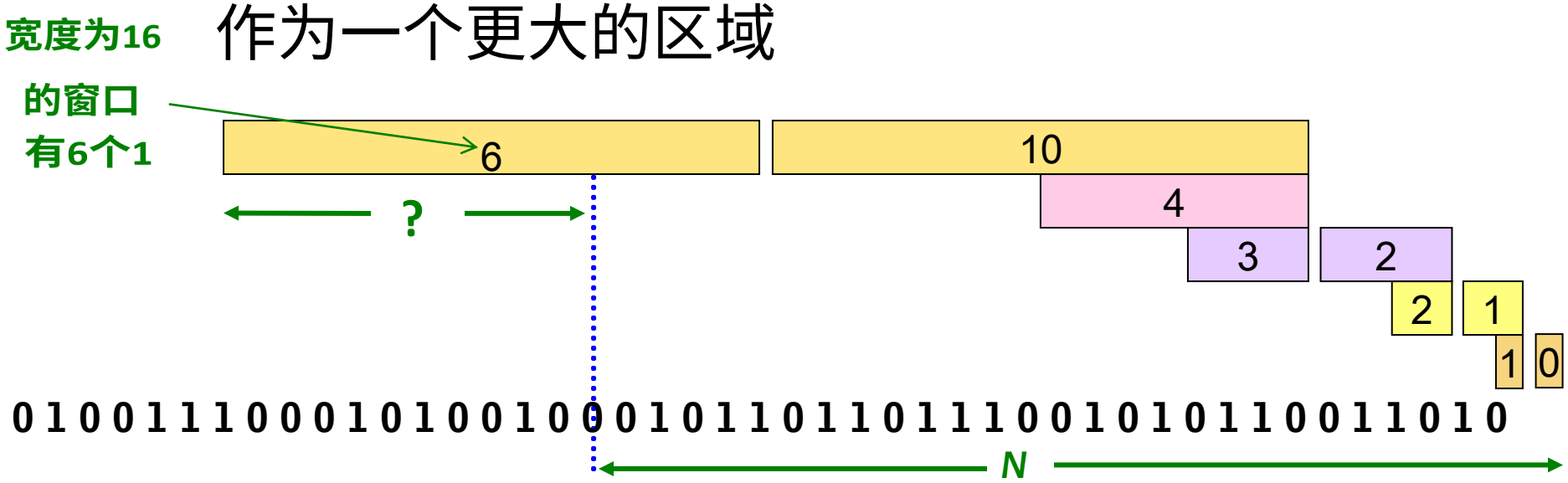
DGIM Method

[Datar, Gionis, Indyk, Motwani]。

- **不假设均匀性的DGIM解决方案**
- 我们在每个数据流中存储$O(\log^2 N)$位。
- **解决方案给出了近似的答案，但误差不超过50%。**
  - 误差系数可以减少到任何分数>0，但需要更复杂的算法和按比例更多的存储位

Idea: Exponential Windows

- **解决方案，不（相当）有效：**
  - 向后看，总结出**指数级增长**区域的流向
  - 如果小区域在同一点上开始，则放弃它们作为一个更大的区域



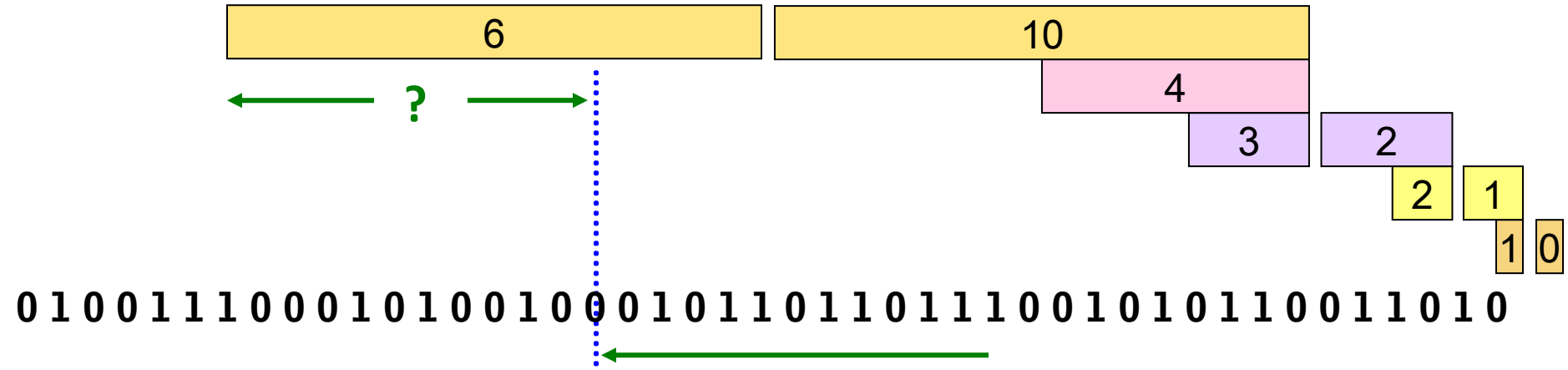
我们可以重建最后***N*** **个**比特的计数，只是我们不确定最后的6**个1中有多少**是包含在***N***

What's Good?

- **只存储O(*log2N*)比特**
  - O (对数 ?) 对数$_2$?位的计数。
- **随着更多比特的进入，易于更新**
- 计数的误差不超过 "**未知** "区域的**1的**数量

What's Not So Good?

- 只要 "1 "的分布相当均匀，由未知区域引起的误差就很小--**不超过50%**。
- 但可能是所有的1都在末尾的未知区域
- 在这种情况下，**错误是无界的!**



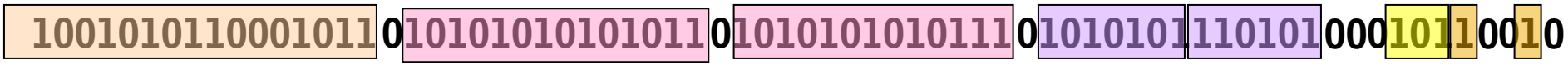

N

Fixup: DGIM method

[Datar, Gionis, Indyk, Motwani]。

- **思路：**与其总结固定长度的块，不如总结有特定数量**1的**块：
  - 让区块***大小***（1的数量）以指数形式增加

- **当窗口中很少有1时，块的大小保持很小，所以误差也小。**




N

DGIM: Timestamps

- 流中的每个比特都有一个***时间戳***，从**1，2，……**开始。
- 记录时间戳的模数为***N***（**窗口大小**），所以我们可以表示任何**相关的**时间戳，单位为$\log_2 N$）位数

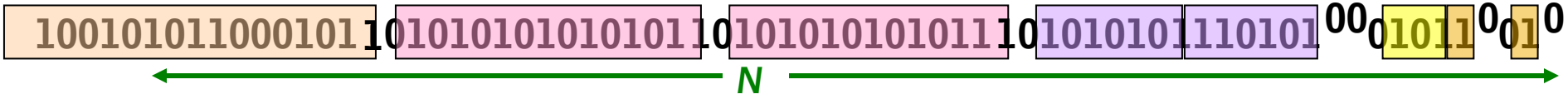
DGIM: Buckets

- 在DGIM方法中，***一个桶***是一条记录，

  由以下内容组成：
  - **(A) 其结束的时间戳**[O(log *N*)位]
  - **(B) 其开始和结束之间的1的数量**[O(log log *N*)位]

- **对桶的限制：**

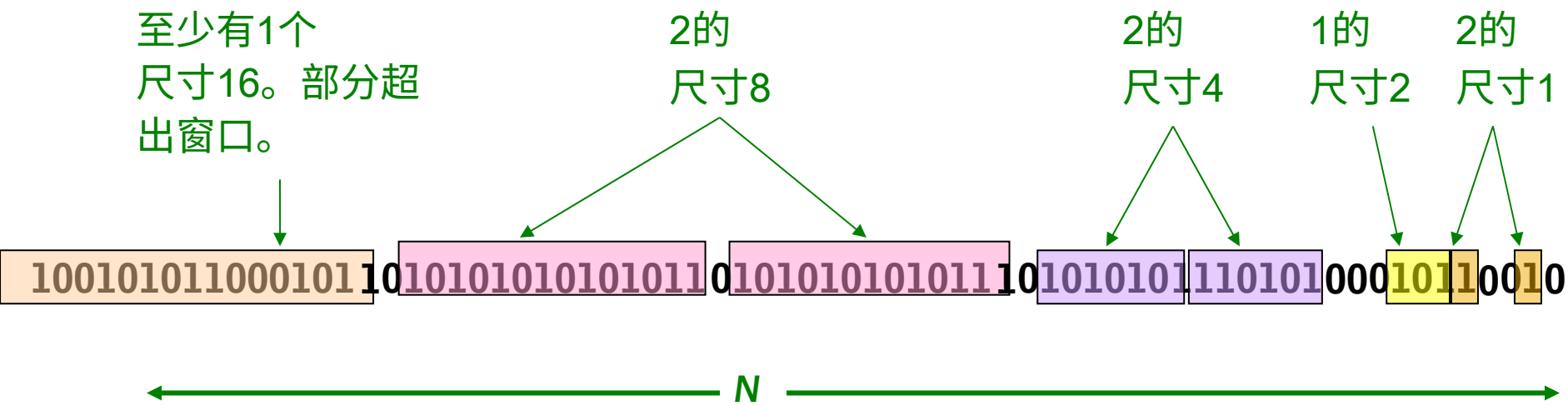
  1的数量必须是**2**的倍数
  - 这就解释了**上面（B）**中的**O（log log *N*）。**


1001010110001011010101010101011010101010101110101010111010100010110010
N

Representing a Stream by Buckets

- 无论是一个还是两个桶，都有相同的
  1的2次方数
- **桶中的时间戳不重合**
- **桶是按大小排序的**
  - 早期的水桶不比后期的水桶小
- 水箱在其
  结束时间是>过去的***N*** **个**时间单位

Example: Bucketized Stream



被维护的水桶的三个属性：

- **一个**或**两个**具有相同**1的2次方**数量的桶
- 桶中的时间戳不重合
- 桶是按大小排序的

Updating Buckets (1)

- 当一个新的位进来时，如果最后一个（最老的）桶的结束时间在当前时间之前的***N*个**时间单位之前，则放弃该桶。

- **2种情况：** 当前位为**0**或**1**

- **如果当前位是0：**

不需要其他改动

Updating Buckets (2)

- **如果当前位是1：**
  - **(1)** 创建一个大小为**1**的新桶，只为这个位。
    - **结束时间戳=当前时间**
  - **(2)** 如果现在有**三个大小为1的桶，将最老的两个桶合并为一个大小为2的桶。**
  - **(3)** 如果现在有**三个大小为2的桶，将最老的两个桶合并为一个大小为4的桶。**
  - **(4)以此类推……。**

Example: Updating Buckets

流的当前状态：

1001010110001011010101010101011010101010101110101010111010100010110010

数值为1的比特到达

0010101100010110101010101010110101010101011101010111010100010110010 1

两个橙色的桶被合并成一个黄色的桶

0010101100010110101010101010110101010101011101010111010100010110010 1

下一个位1到了，新的橙色桶被创建，然后是0来了，然后是1：

0101100010110101010101010110101010101011101010111010100010110010 1 1

桶被合并了…

合并后的桶的状态


0101100010110101010101010110101010101010111010101011101010001011001011101

How to Query?

- **要估计最近的*N* 个比特中的1的数量：**

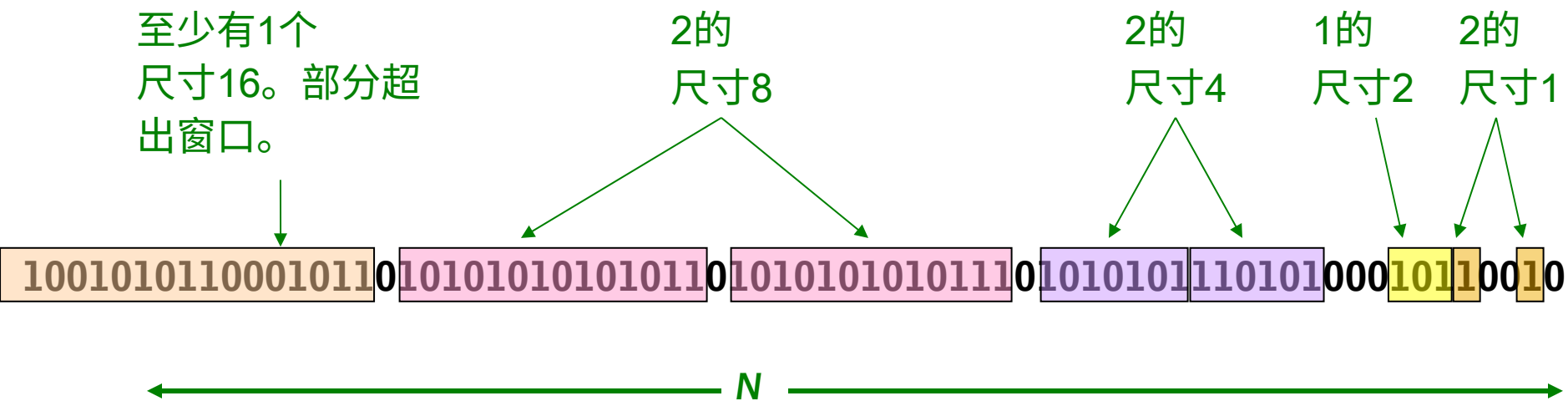
1. **将所有桶的大小相加，但最后一个桶除外。**

   (注意 "大小 "是指桶中的1的数量)

2. **加入上一个桶的一半大小**

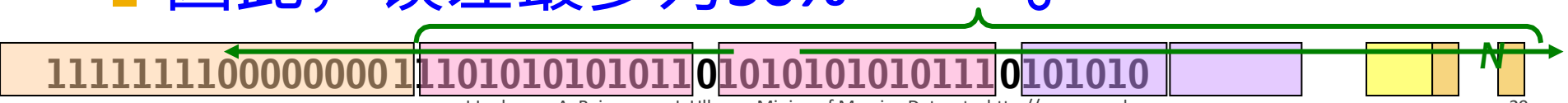
- **请记住：**我们不知道最后一个桶里有多少个1还在想要的窗口内。

Example: Bucketized Stream



Error Bound: Proof

- **为什么误差是50%？让我们来证明一下！**
- 假设最后一个桶的大小为2^r
- 那么，通过假设其2^r-1（即一半）的1仍在窗口内，我们最多只会产生$2r-1$的误差
- 由于每个大小的桶中至少有一个小于2^r的桶，真正的总和至少是$1+2+4+...+2r-1 = 2^r - 1$
- 因此，误差最多为50%。



至少16个1

00 0 0 1 1 0 1 0 1 0 1 1 0 1 1 0 1

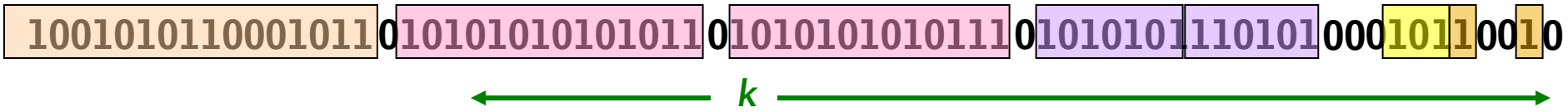
Further Reducing the Error

- 我们不保持每个大小的桶的1或2，而是允许*r*-1或*r*个桶（*r*>2）。
  - 除了最大尺寸的桶，我们可以有1到*r*之间的任何数量的桶。
- **误差最多为*O*(1/r)**
- 通过适当地选择*r*，我们可以在我们存储的比特数和

  错误

Extensions

- 我们能否用同样的技巧来回答询问在**最后的*k*里有多少个1？** 其中***k*<*N***?
  - **A:** 找到与***k***重合的最早的桶**B**，**1**的数量是**最近的桶的大小之和+B的½大小**



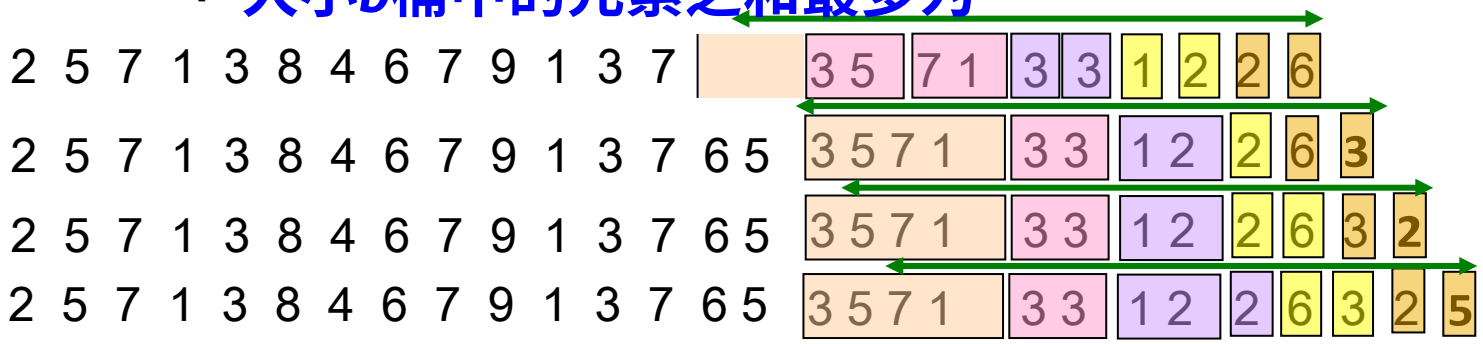
- **我们能否处理这样的情况，即流是不是比特，而是整数，而我们想要的是最**

后k个元素的总和?

Extensions

- **正整数流**
- **我们希望最后*k*个元素的总和**
  - **亚马逊：**最近**k次**销售的平均价格
- **解决方案：**
  - **(1) 如果你知道所有的人最多有*m*个比特**
    - 将每个整数的***m***位作为一个单独的流处理
    - 使用DGIM来计算每个整数中的**1**
    - 总和为 $= \sum_{i=0}^{m-1} c_i 2^i$
  - **(2) 使用桶来保存部分的和**
    - **大小*b*桶中的元素之和最多为2^b**

c_i …第i位的估计计数



思想：每个桶中的总和最多为2^b（除非桶中只有**1个**整数）

桶的大小：

Summary

- **计算最后N个元素中的1的数量**
  - 指数级增长的窗口
  - 延长：
    - 任何最后k（k<N）个元素中的1的数量
    - 最后N个元素中的整数之和