# Amortized Analysis of a Binary Counter

Andreas Klappenecker

# Binary Counter

A binary k-bit counter can be implemented with a k-element binary array. The counter is initially 0.

The only operation is increment(A), which adds 1 to the current number in the counter.

The increment operation can be implemented using the grade-school ripple-carry algorithm.

# Aggregate Method

The worst case running time occurs when all k bits are flipped, so increment(A) has running time O(k).

In a sequence of n increment operations, few increments will cause that many bits to flip. Indeed,

bit 0 flips with every increment

bit 1 flips with every 2nd increment

bit 2 flips with every 4th increment, ...

# Aggregate Method

Total number of bit flips in n increment operations is

$$n + n/2 + n/4 + ... + n/2^k < n(1/(1-1/2)) = 2n$$

So total cost of the sequence is O(n).

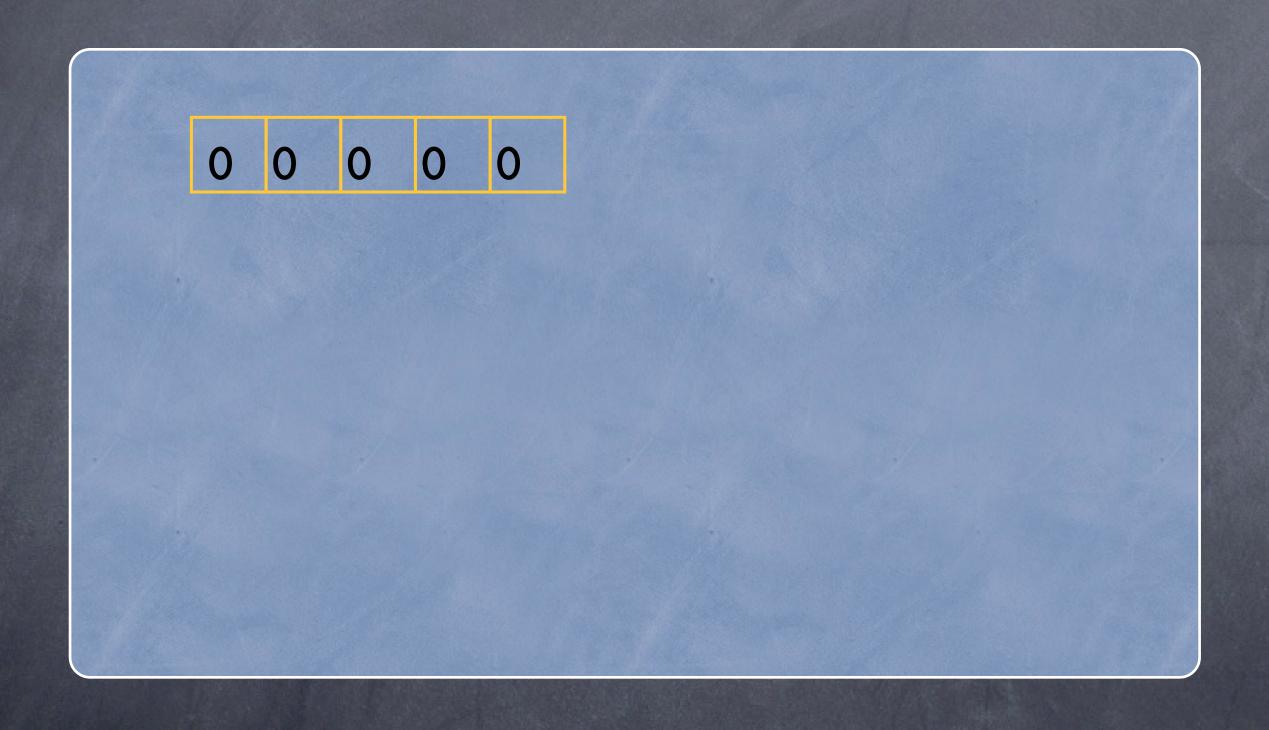
Amortized cost per operation is O(n)/n = O(1).

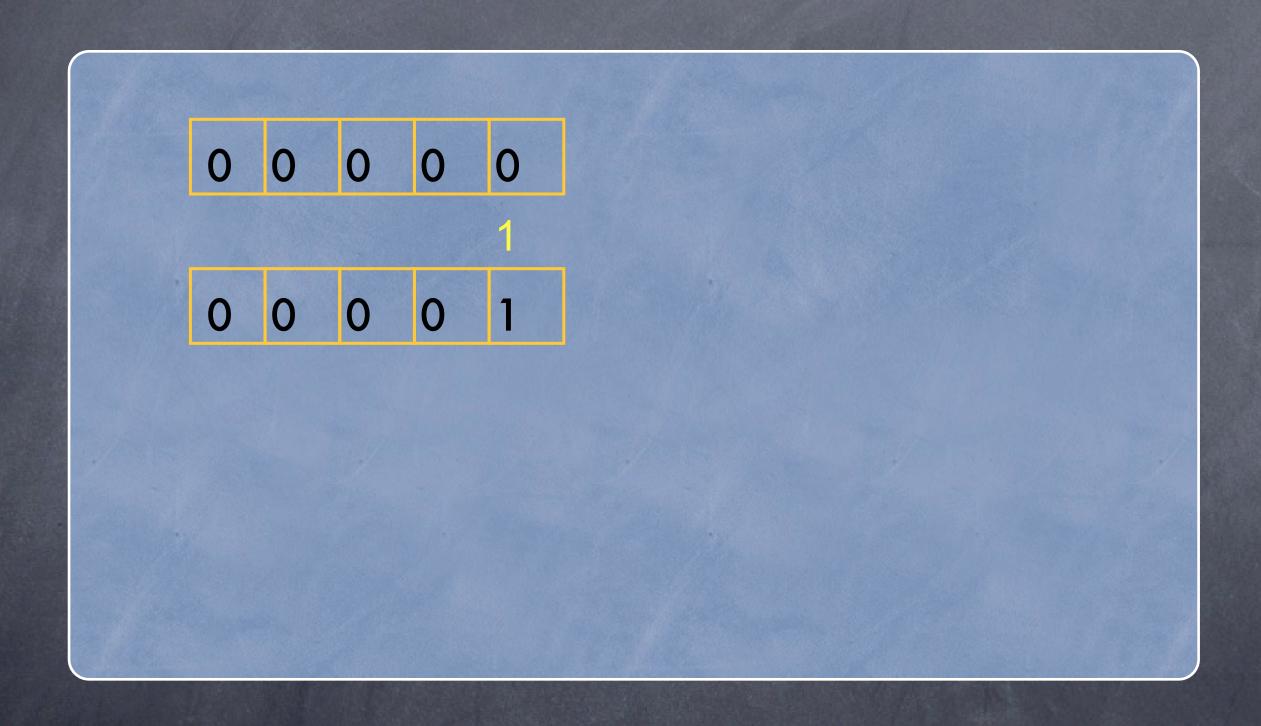
# Accounting Method for k-Bit

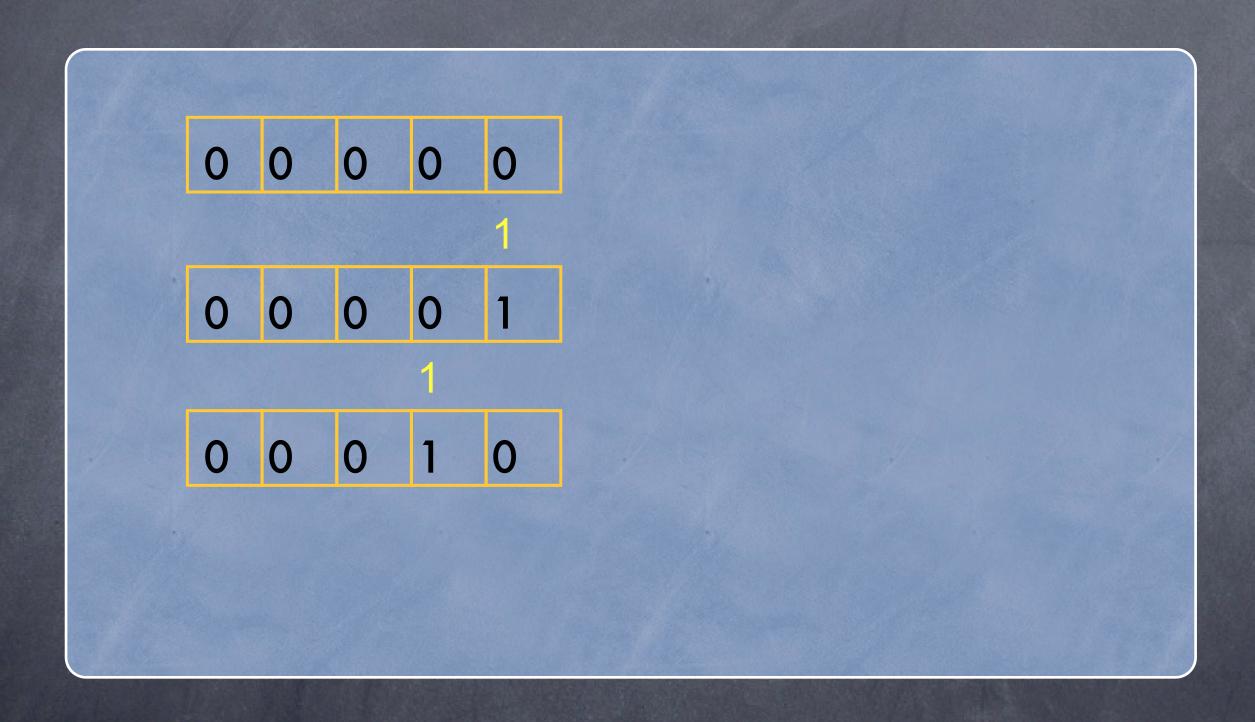
The actual cost for an increment operation is the number of bits flipped.

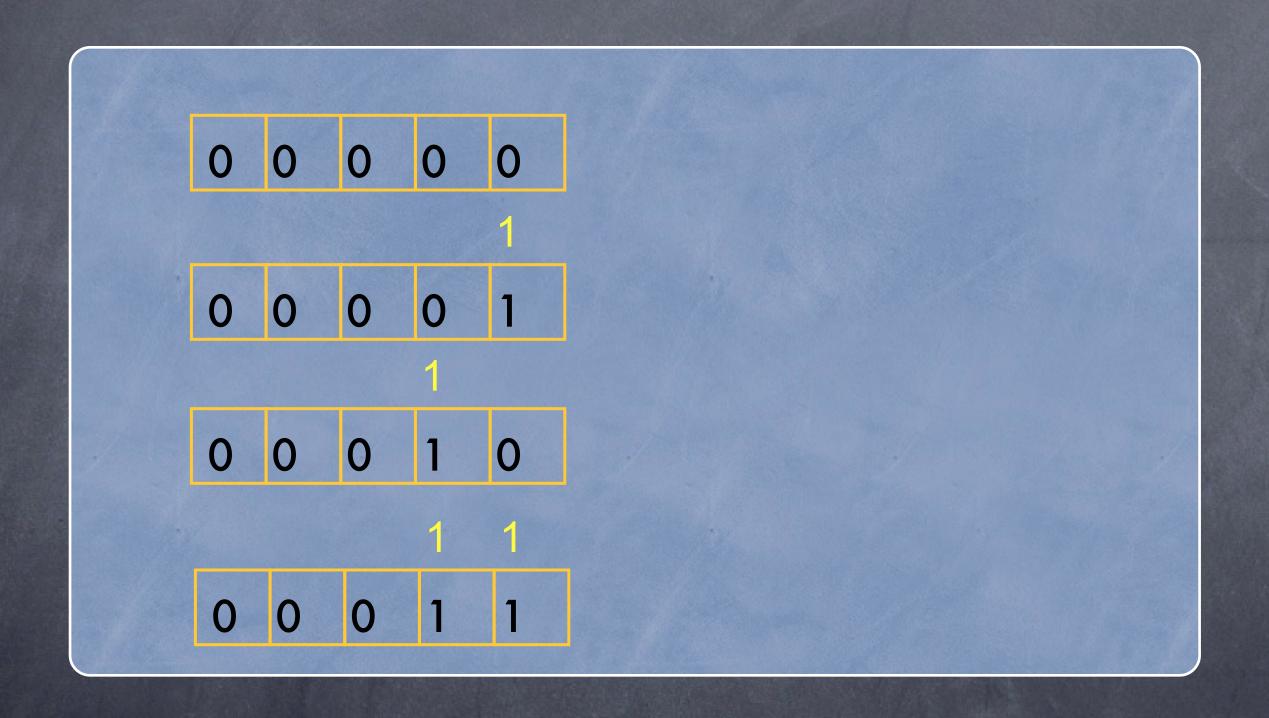
We can assign an amortized cost of 2 for each increment operation. The main idea is to use 1 to flip the bit from 0 to 1 and store 1 credit to flip it back to 0 later.

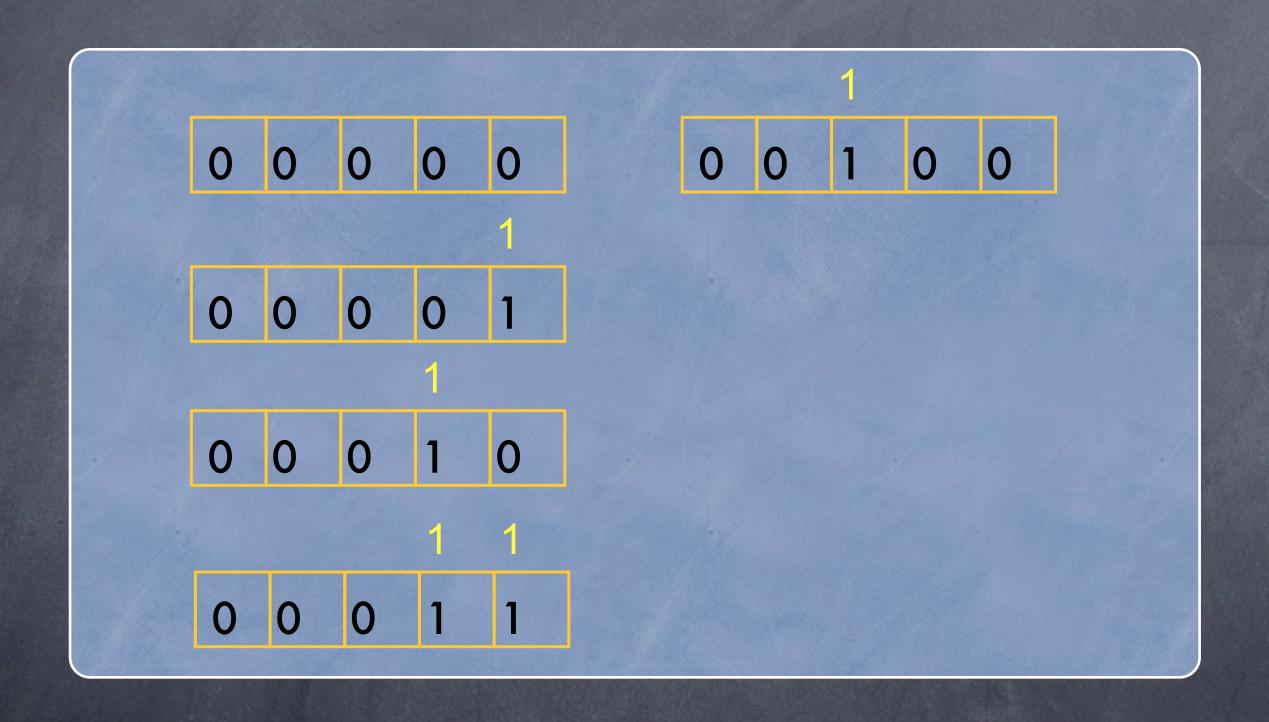


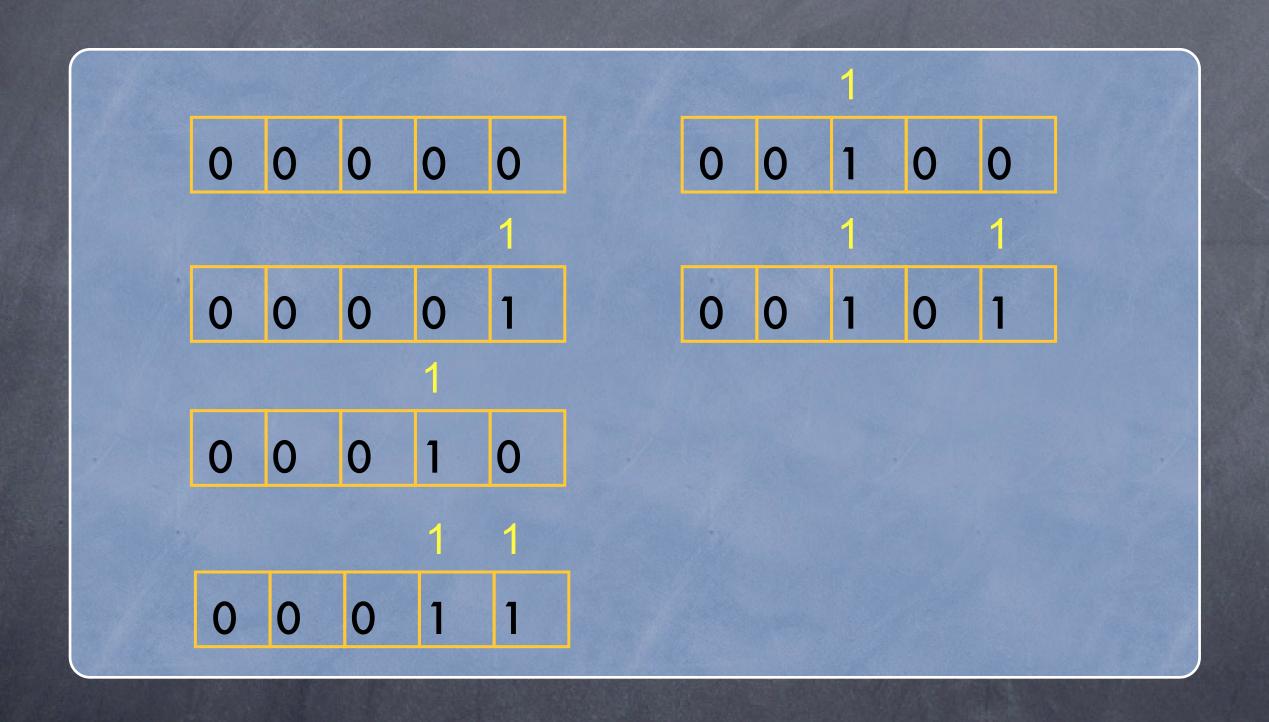


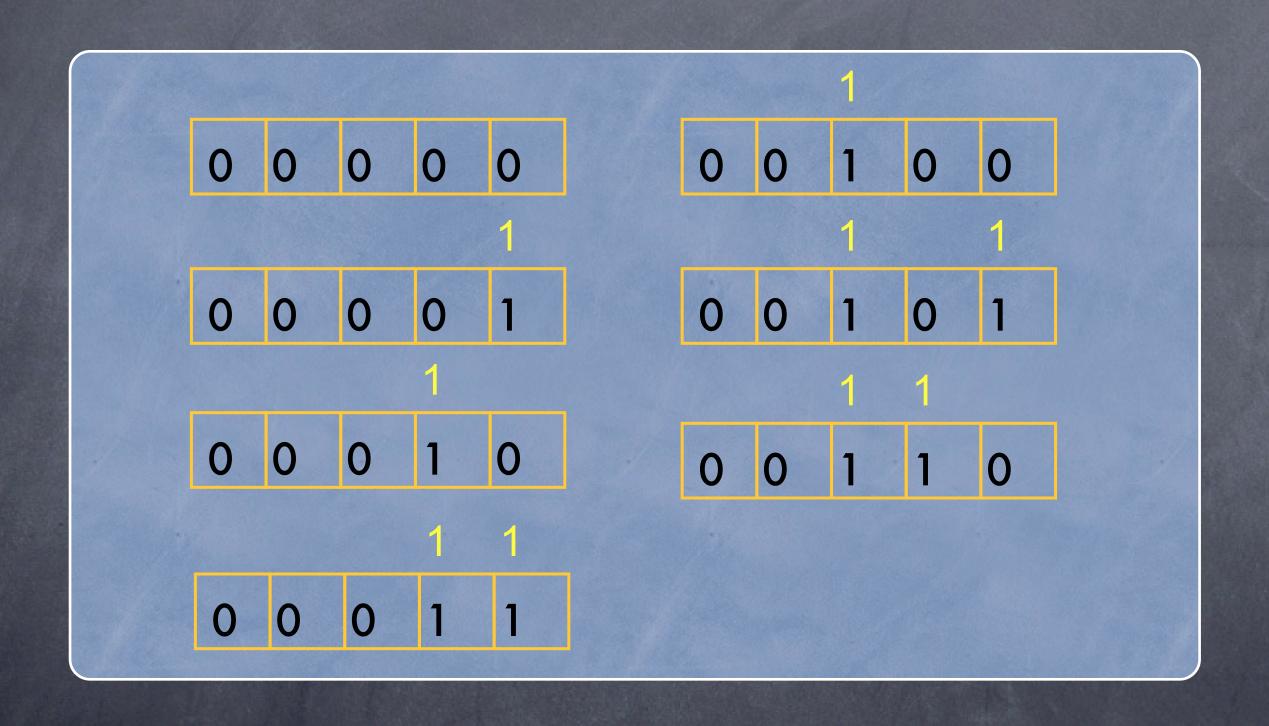


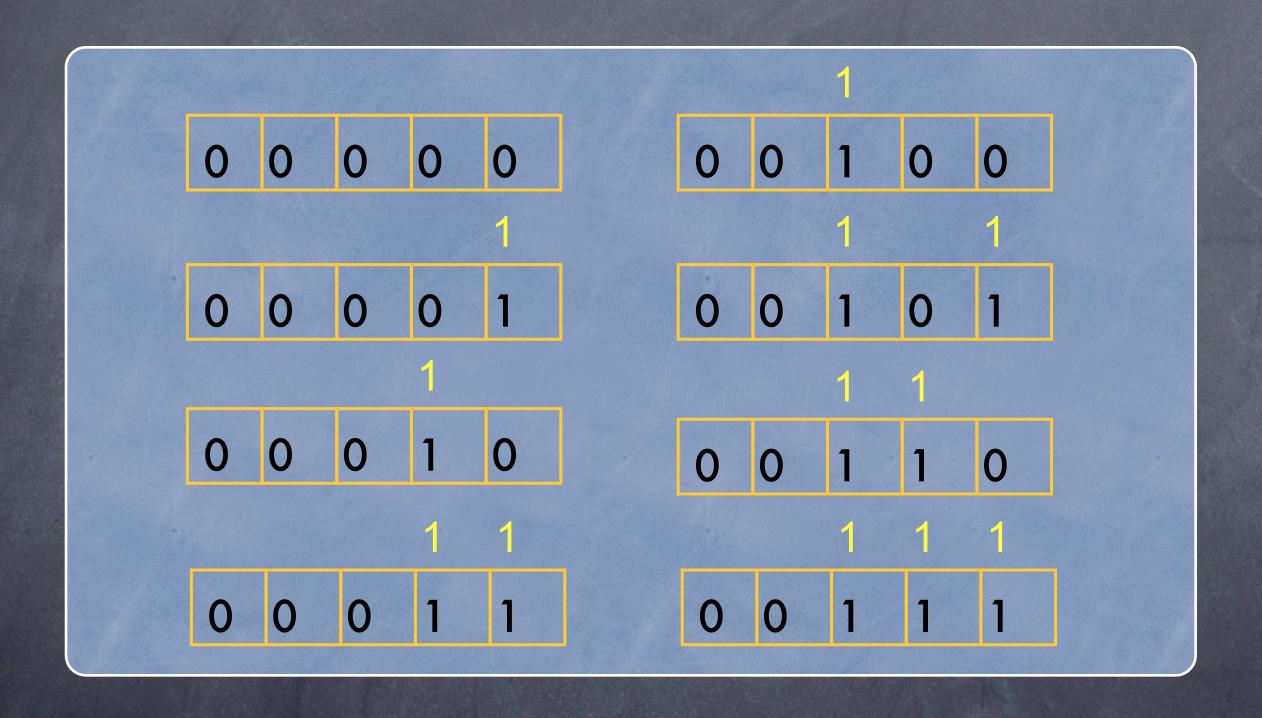












# Accounting Method

All changes from 1 to 0 are paid for with previously stored credit (never go into red)

The amortized time per operation is O(1)