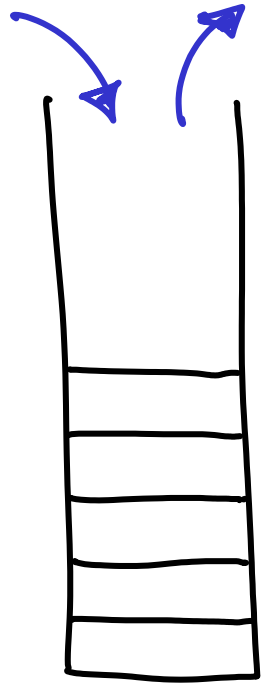
Amortization

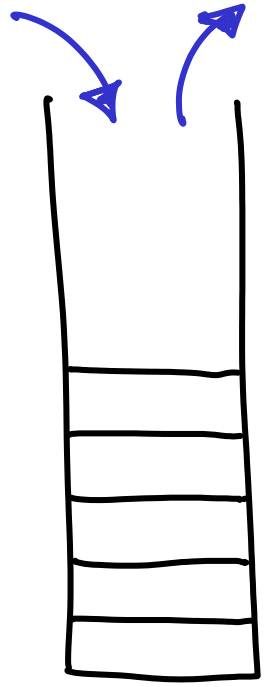
Stack operations: push pop

Cost = 1 Cost = 1



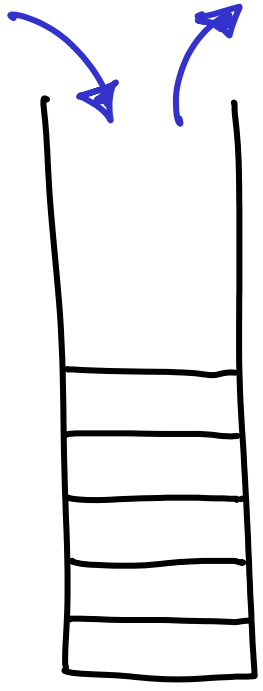
Stack operations:

push	pop	multipop(k)
Cost = 1	Cost = 1	Cost = k



Stack operations: push pop multipop(k)

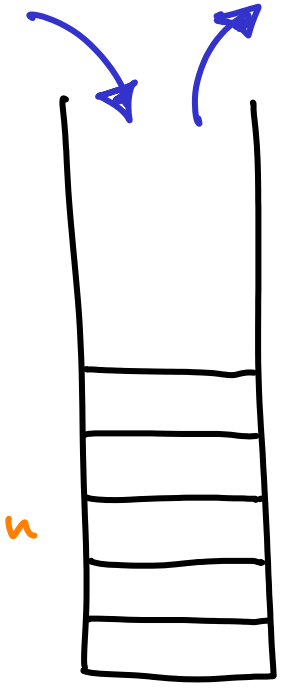
Cost = 1 Cost = 1 Cost = k



What will n operations cost ?

Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k

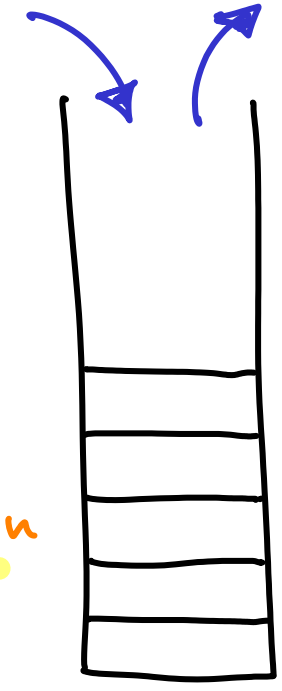


What will n operations cost ?

Worst case: n. (max individual op) = n.k = k per operation

Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k



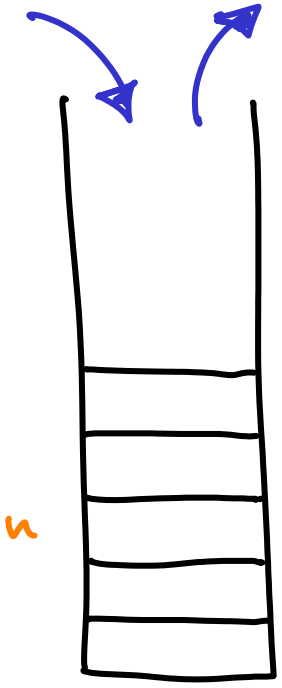
What will n operations cost ?

Worst case: n. (max individual op) = n.k = k per operation

Amortized = O(1) per operation

Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k



What will n operations cost ?

Worst case: n. (max individual op) = n.k = k per operation

Amortized = O(1) per operation

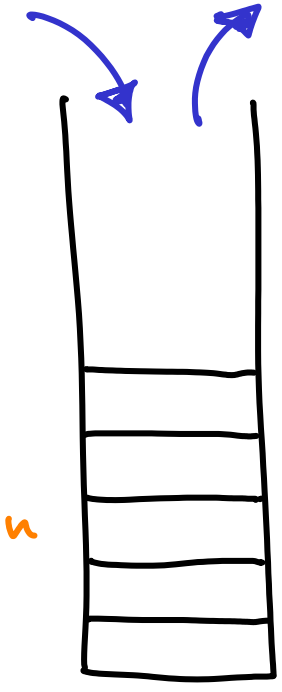
Intuitive: each element can be pushed precisely once

but also only popped or multipopped once

Contribution per element ≤ 2

Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k



What will n operations cost ?

Worst case: n. (max individual op) = n.k = k per operation

Amortized = O(1) per operation

Intuitive: each element can be pushed precisely once

but also only popped or multipopped once

Contribution per element ≤ 2 / #elements ≤ n / Total cost = O(n)

AMORTIZATION

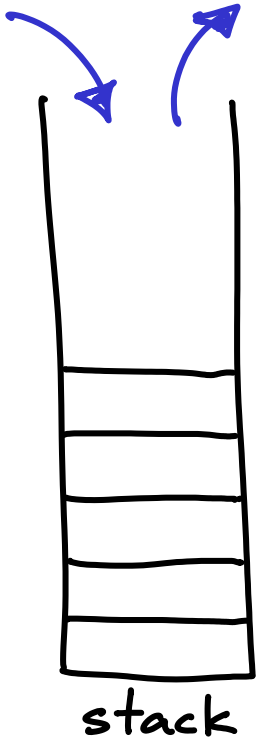
Applies to some problems that involve many operations.

If worst case time of operation k is $O(f(k))$,
try to show that n operations cost $o(n \cdot f(n))$

Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k

Pretend 2



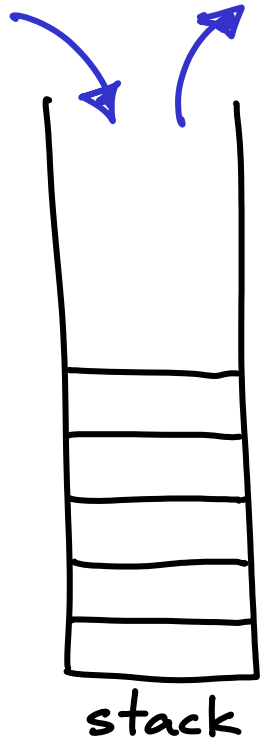
Accounting method

- Pretend some cheap operation type costs more

Stack operations: push pop multipop(k)

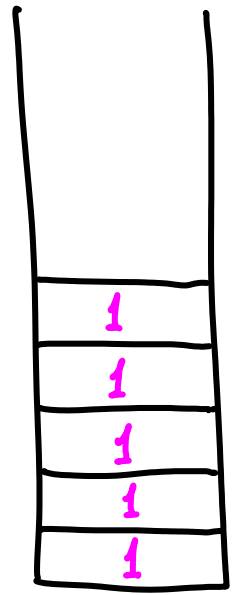
Cost = 1 Cost = 1 Cost = k

Pretend 2



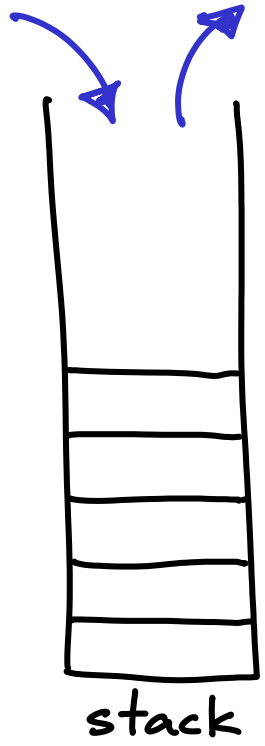
Accounting method

- Pretend some cheap operation type costs more
- Save the difference



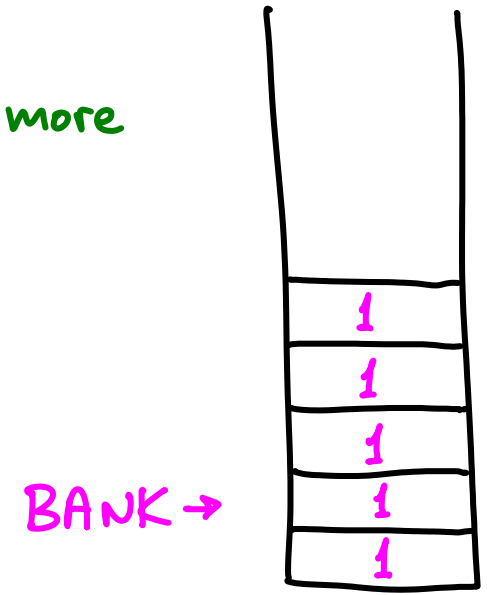
Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
	Pretend 2	use savings	
Amortized cost:	2	0	0



Accounting method

- Pretend some cheap operation type costs more
- Save the difference, use it later

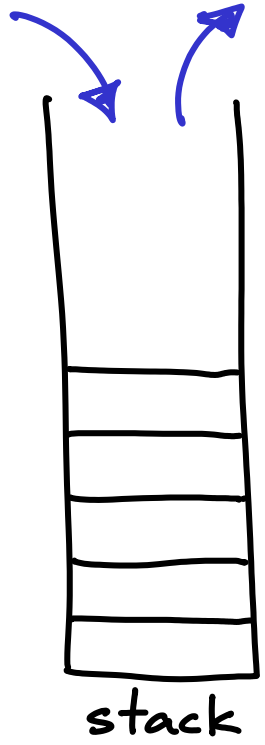


Stack operations: push pop multipop(k)

Cost = 1 Cost = 1 Cost = k

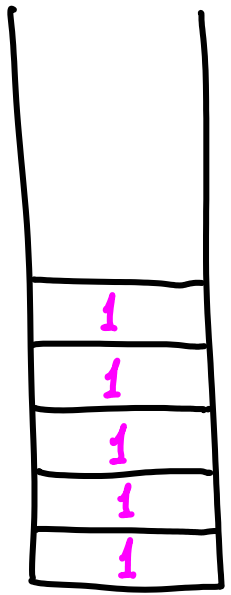
Pretend 2 use savings

Amortized cost: 2 0 0



ACCOUNTING METHOD

- Pretend some cheap operation type costs more
- Save the difference, use it later



True cost $\leq n \cdot$ (worst amortized cost)

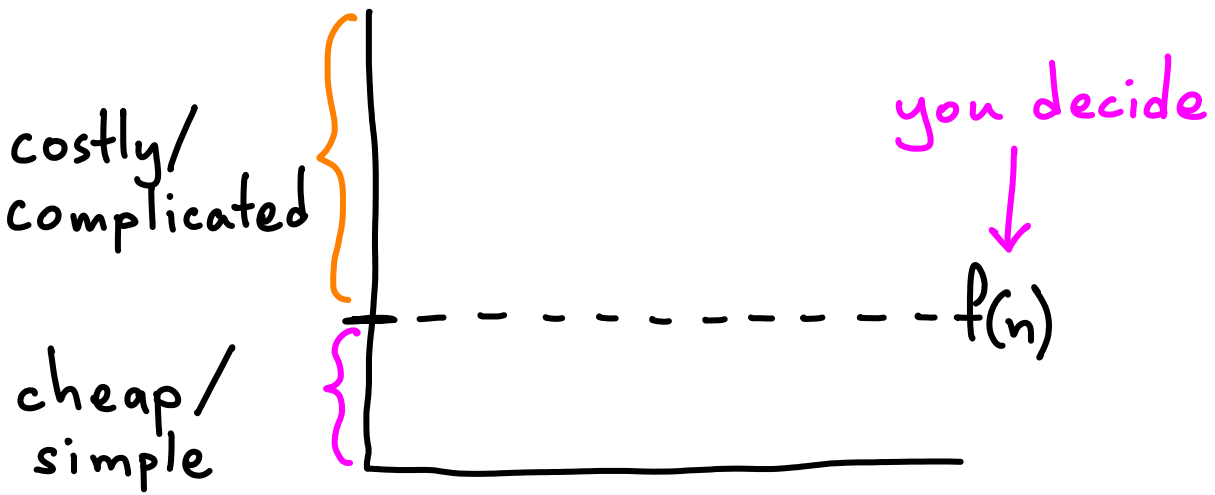
$= n \cdot 2$

ACCOUNTING : saving for a rainy day

ACCOUNTING : saving for a rainy day

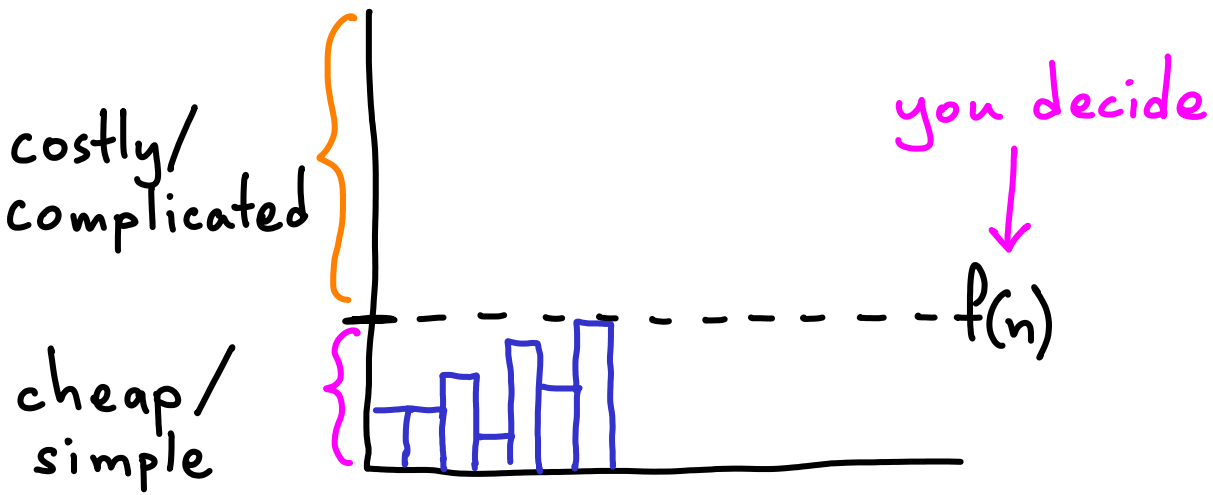
"simple" operations

"Complicated/costly" operations



ACCOUNTING : saving for a rainy day

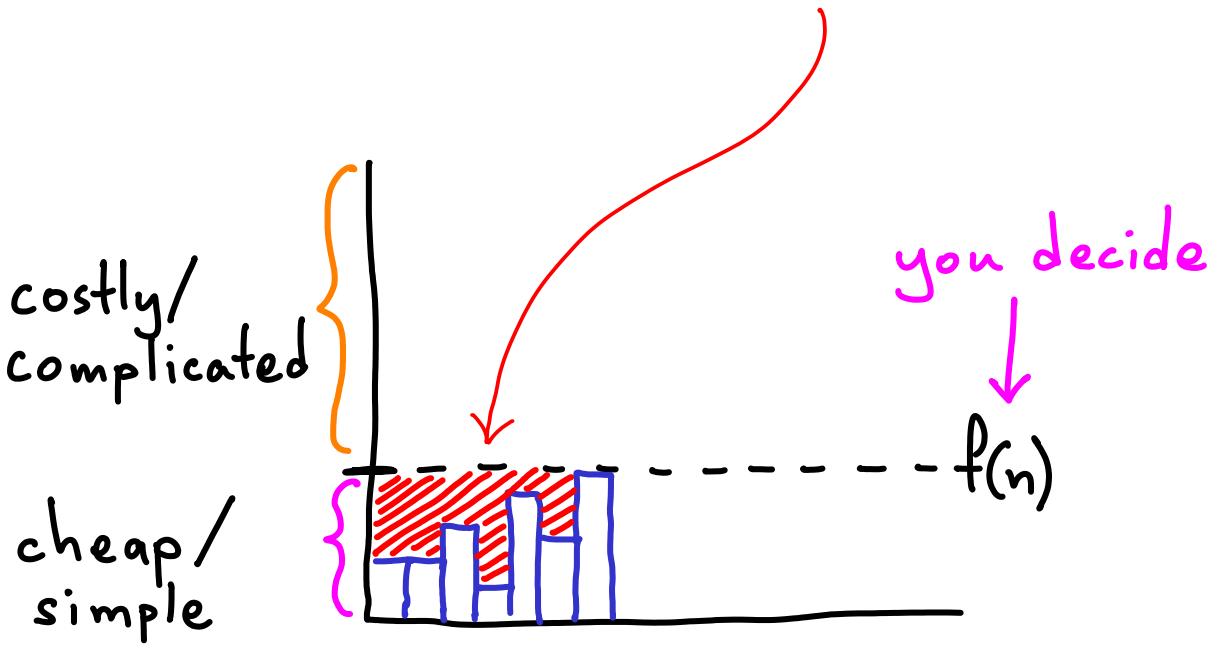
"simple" operations



Accounting : saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference

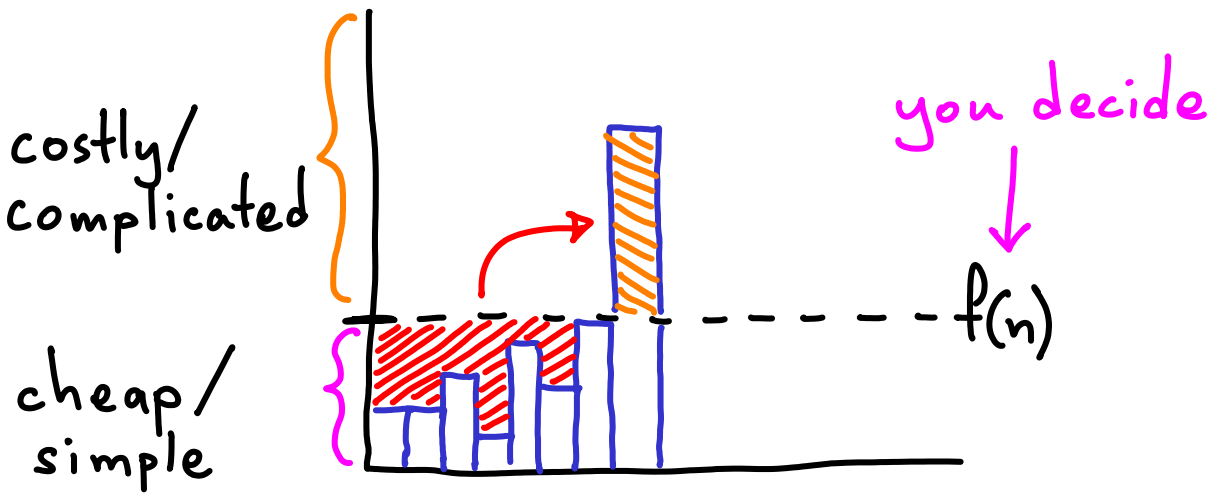


Accounting: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings

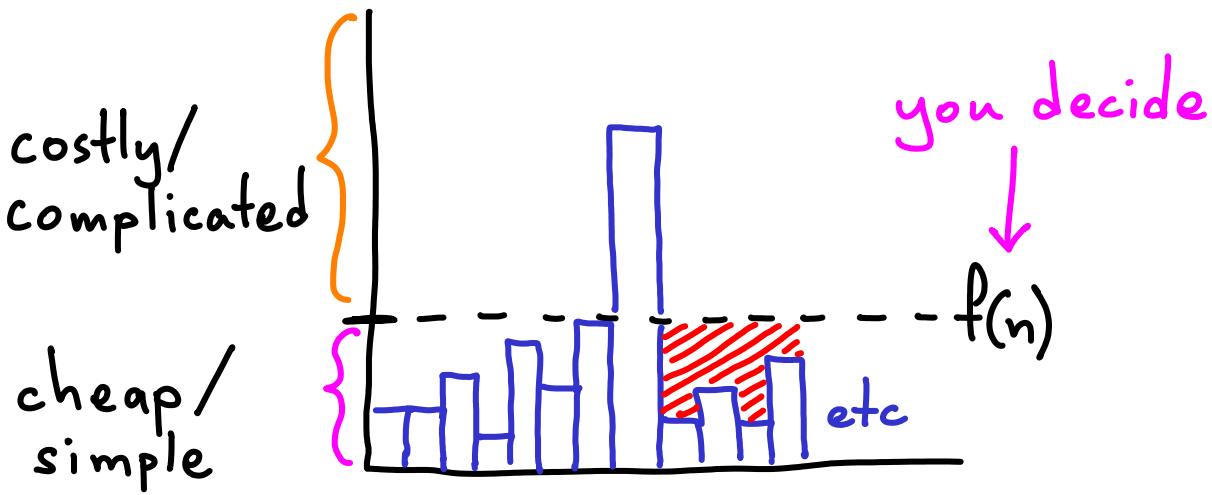


Accounting: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings

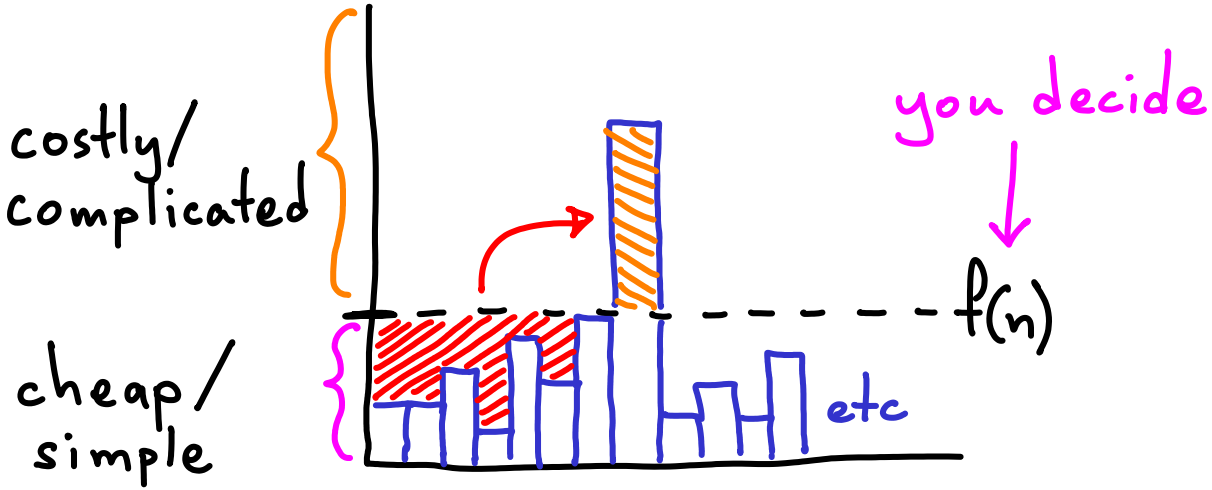


Accounting: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings



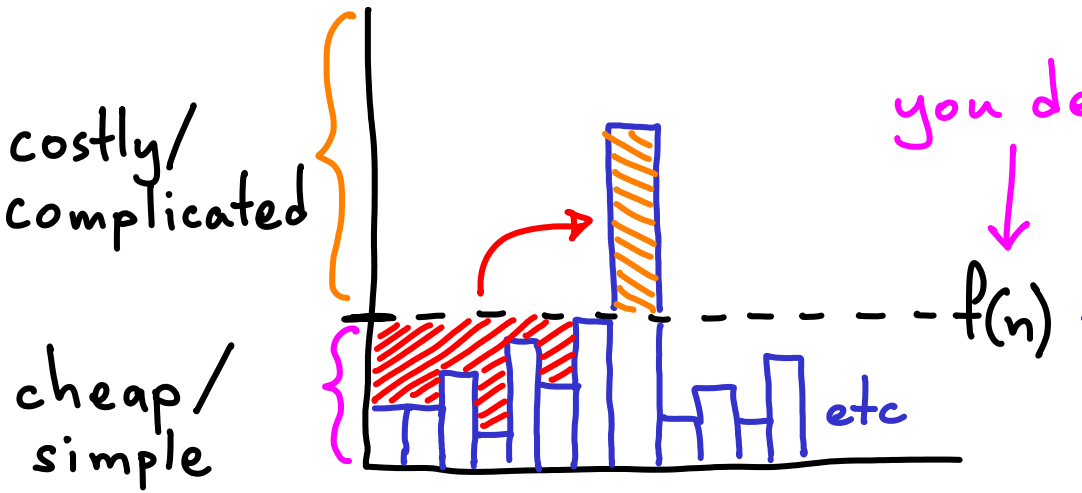
Rule: Never spend more than what you saved.

Accounting: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings



Rule: Never spend more than what you saved.

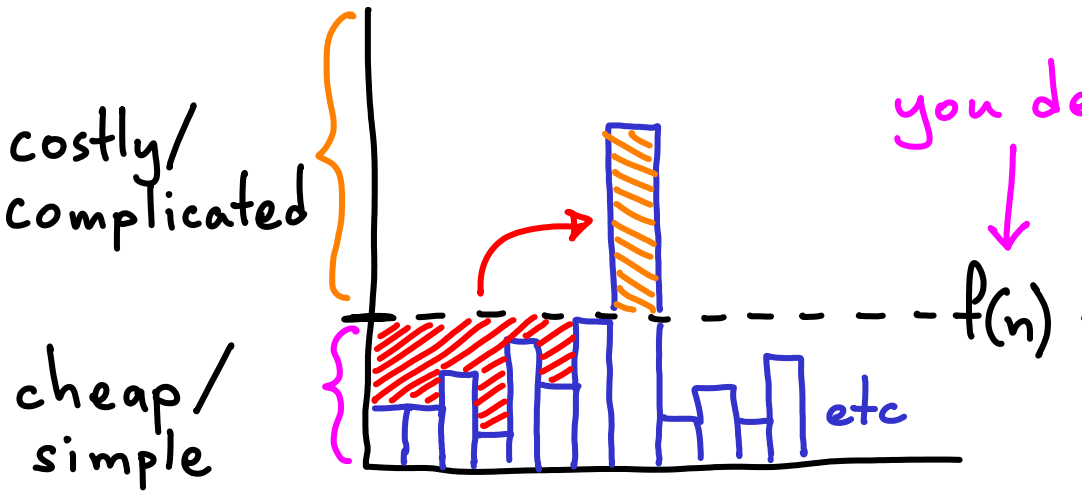
Amortized cost: $n \cdot f(n) \geq$ true cost

Accounting: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally Θ(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings



you decide

Rule: Never spend more than what you saved.

Amortized cost: $n \cdot f(n) \geq$ true cost

Goal: exaggerate / save as little as possible. } minimize $f(n)$

POTENTIAL METHOD

aka Physicist's method

POTENTIAL METHOD

Start with data structure D_0

Operation i : $D_{i-1} \rightarrow D_i$ cost : c_i

POTENTIAL METHOD

Start with data structure D_0

Operation i : $D_{i-1} \to D_i$ cost : c_i

Potential function Φ_i maps $D_i \to \mathbb{R}$: potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

POTENTIAL METHOD

Start with data structure D_0

Operation i : $D_{i-1} \to D_i$ cost : c_i

Potential function Φ_i maps $D_i \to \mathbb{R}$: potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

POTENTIAL METHOD

Start with data structure D_0

Operation i : $D_{i-1} \to D_i$ cost : c_i

Potential function Φ_i maps $D_i \to \mathbb{R}$: potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

If $\Delta\Phi_i > 0$, $\hat{c}_i > c_i$: storing potential ... "work" in D_i

POTENTIAL METHOD aka Physicists' method

Start with data structure D_0

Operation i : $D_{i-1} \to D_i$ cost : c_i

Potential function Φ_i maps $D_i \to \mathbb{R}$: potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

If $\Delta\Phi_i > 0$, $\hat{c}_i > c_i$: storing potential ... "work" in D_i

$\Delta\Phi_i < 0$, $\hat{c}_i < c_i$: release work.

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i) \quad = \; ?$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i \quad \geq \sum_{1}^{n} c_i$$

$\Phi_n \geq 0$, $\Phi_0 = 0$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i \quad \geq \sum_{1}^{n} c_i$$

$\Phi_n \geq 0$, $\Phi_0 = 0$

so we know that the amortized cost will not underestimate real cost.

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

($\Phi_n \geq 0$, $\Phi_0 = 0$)

so we know that the amortized cost will not underestimate real cost.

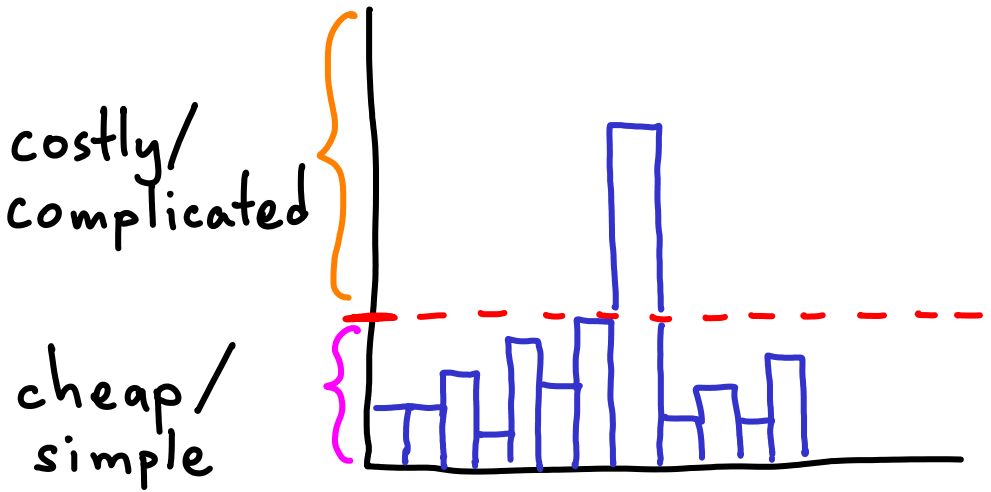
now figure out worst case for any individual $\hat{c}_i$

↳ Ideally this will give a good (and easy) bound for total cost

$$\sum c_i \leq \sum \hat{c}_i \leq n \cdot \max \hat{c}_i$$

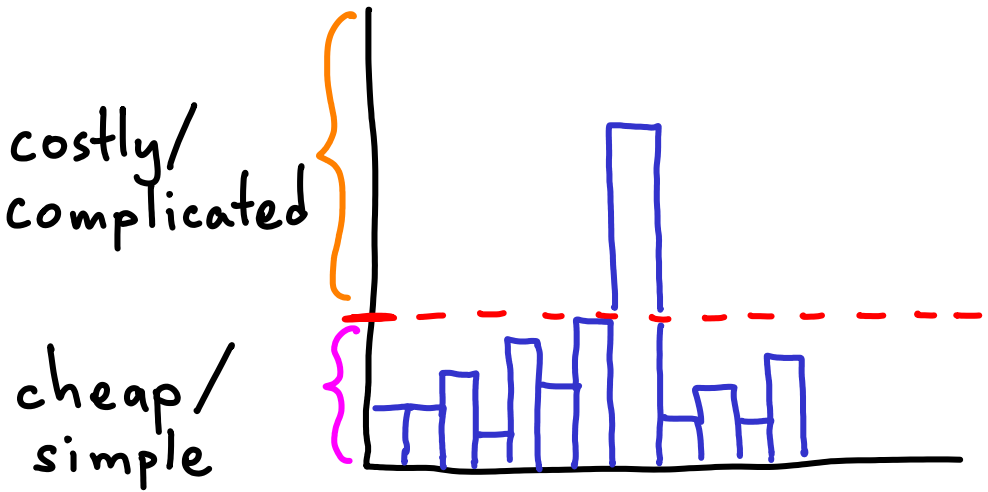
How this works

- (Subjectively) define what a complicated / costly operation (c_i) is.



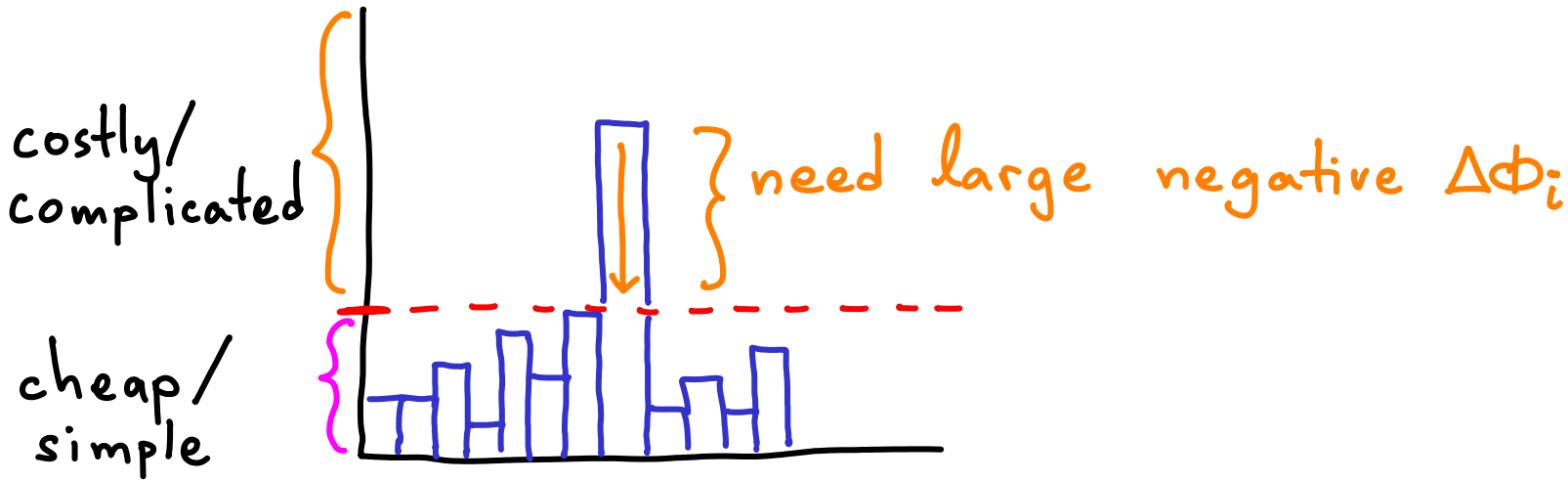
How this works

- (Subjectively) define what a complicated / costly operation (c_i) is.
- Find something that changes a lot in the data structure in such cases.



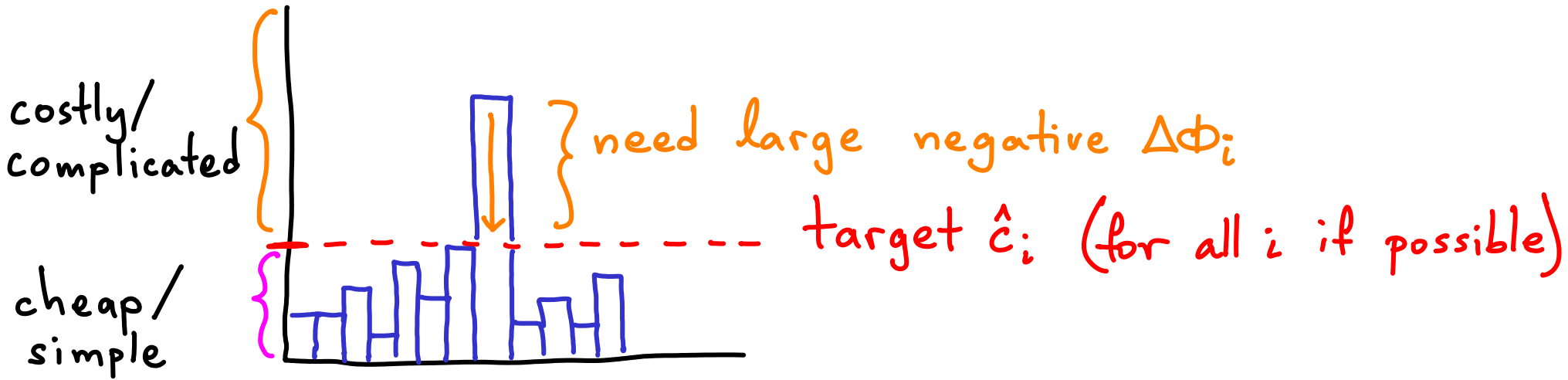
How this works

- (Subjectively) define what a complicated / costly operation (c_i) is.
- Find something that changes a lot in the data structure in such cases.
  - ↪ Quantify this change as $\Delta\Phi_i$: let it "kill" c_i : $\hat{c}_i = c_i + \Delta\Phi_i$
    - ↳ obtain low $\hat{c}_i$



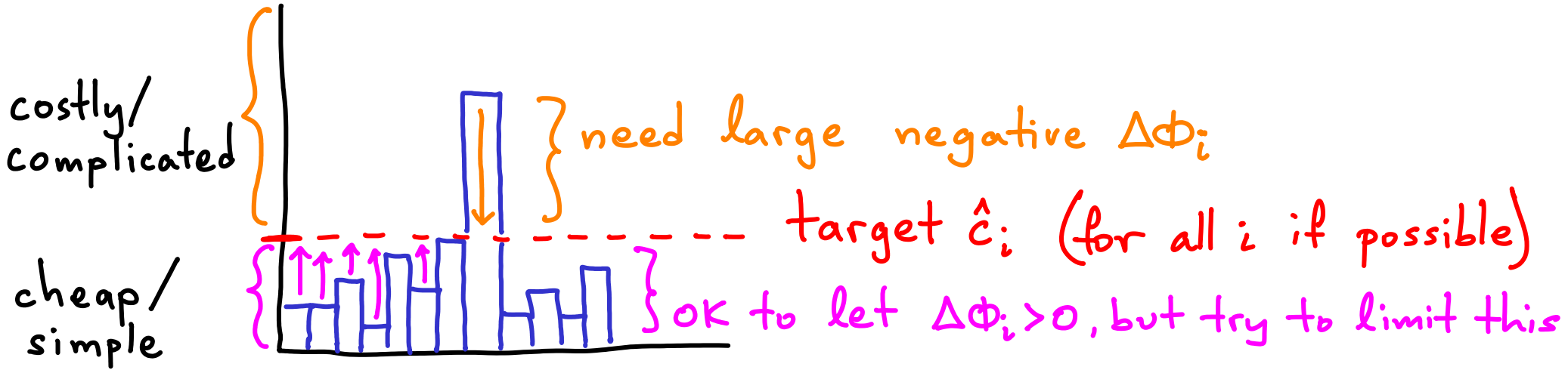
How this works

- (Subjectively) define what a complicated / costly operation (c_i) is.
- Find something that changes a lot in the data structure in such cases.
  - ↪ Quantify this change as $\Delta\Phi_i$: let it "kill" c_i : $\hat{c}_i = c_i + \Delta\Phi_i$
    - ↳ obtain low $\hat{c}_i$



How this works

- (Subjectively) define what a complicated / costly operation (c_i) is.
- Find something that changes a lot in the data structure in such cases.
  - ↳ Quantify this change as $\Delta\Phi_i$: let it "kill" c_i : $\hat{c}_i = c_i + \Delta\Phi_i$
  - ↳ Invent your Φ accordingly ↳ obtain low $\hat{c}_i$
  - ↳ Make sure $\Delta\Phi_i$ doesn't add much to c_i, when c_i is cheap.



Stack operations:

push	pop	multipop(k)
Cost = 1	Cost = 1	Cost = k

Potential method

Let Φ = # elements in stack

Stack operations:

push	pop	multipop(k)
Cost = 1	Cost = 1	Cost = k

Potential method

Let Φ = # elements in stack

Stack operations:	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

Stack operations:	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i$$

Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \Phi_n - \Phi_0$$

Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \overbrace{\Phi_n - \Phi_0}^{\text{if} \geq 0}$$

then $\sum_{i=1}^{n} c \leq \sum_{i=1}^{n} \hat{c}_i$

Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \Phi_n - \Phi_0$$

if ≥ 0 then $\sum_{i=1}^{n} c \leq \sum_{i=1}^{n} \hat{c}_i$

$\Phi_n \geq 0$

$\Phi_0 = 0$ if stack starts empty

Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \Phi_n - \Phi_0$$

if ≥ 0 then $\sum_{i=1}^{n} c \leq \sum_{i=1}^{n} \hat{c}_i$

≥ 0 ; $\Phi_0 = 0$ if stack starts empty

Stack operations:

	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

$$\sum_{i=1}^{n} \hat{c}_i \leq n \cdot \max \hat{c}_i = n \cdot 2$$

Potential method

Let Φ = # elements in stack

Let $\hat{c}_i = c + \Delta\Phi_i$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c + \sum_{i=1}^{n} \Delta\Phi_i = \sum_{i=1}^{n} c + \Phi_n - \Phi_0$$

if ≥ 0 then $\sum_{i=1}^{n} c \leq \sum_{i=1}^{n} \hat{c}_i$

$\Phi_n \geq 0$

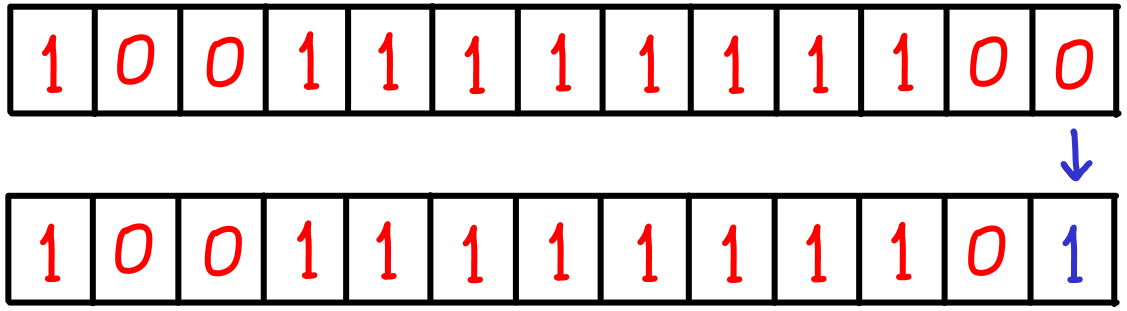
$\Phi_0 = 0$ if stack starts empty

Stack operations:

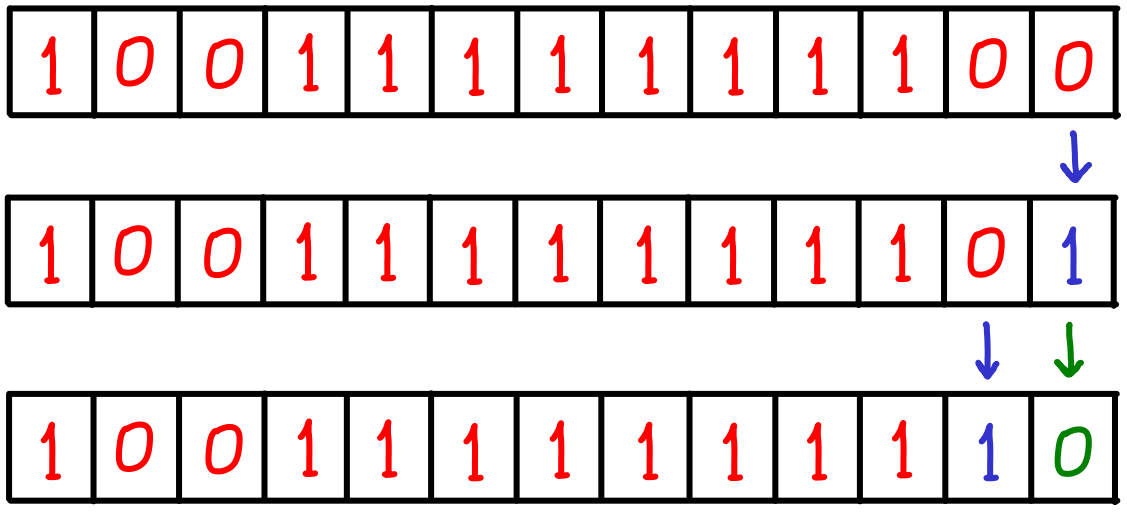
	push	pop	multipop(k)
	Cost = 1	Cost = 1	Cost = k
$\Delta\Phi$	+1	-1	-k
$\hat{c}_i$ =	1 + 1	1 - 1	k - k

$$\sum_{i=1}^{n} \hat{c}_i \leq n \cdot \max \hat{c}_i = n \cdot 2 \rightarrow \sum_{i=1}^{n} c \leq 2n$$

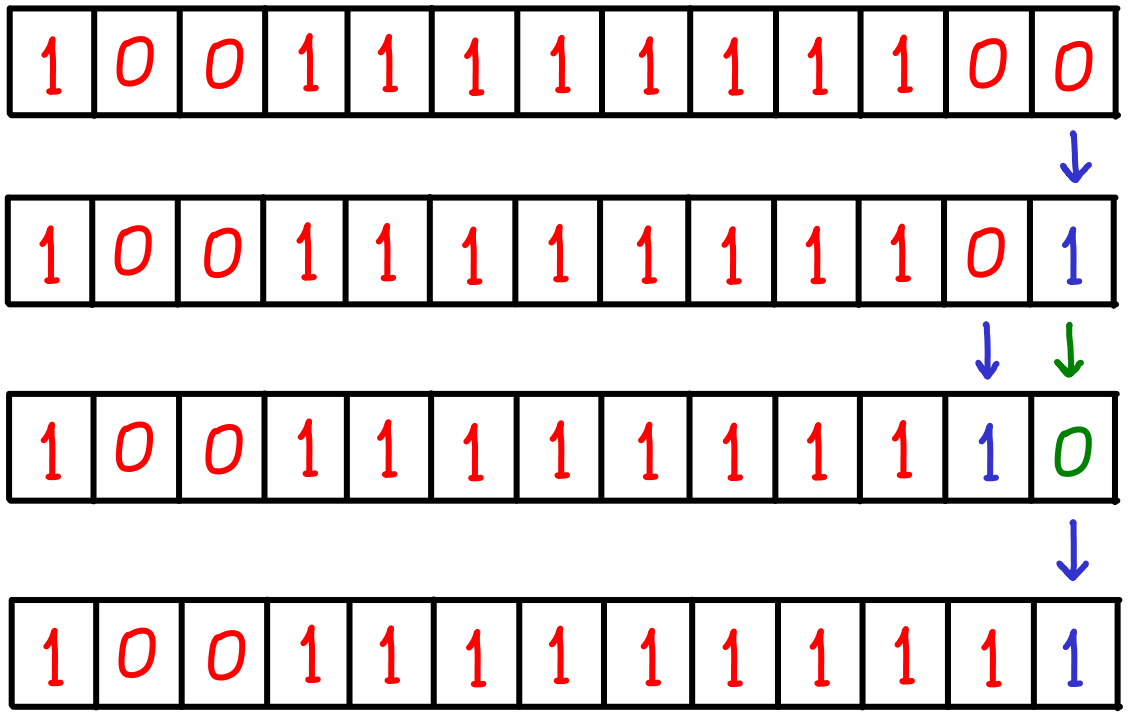
Incrementing a k-bit counter



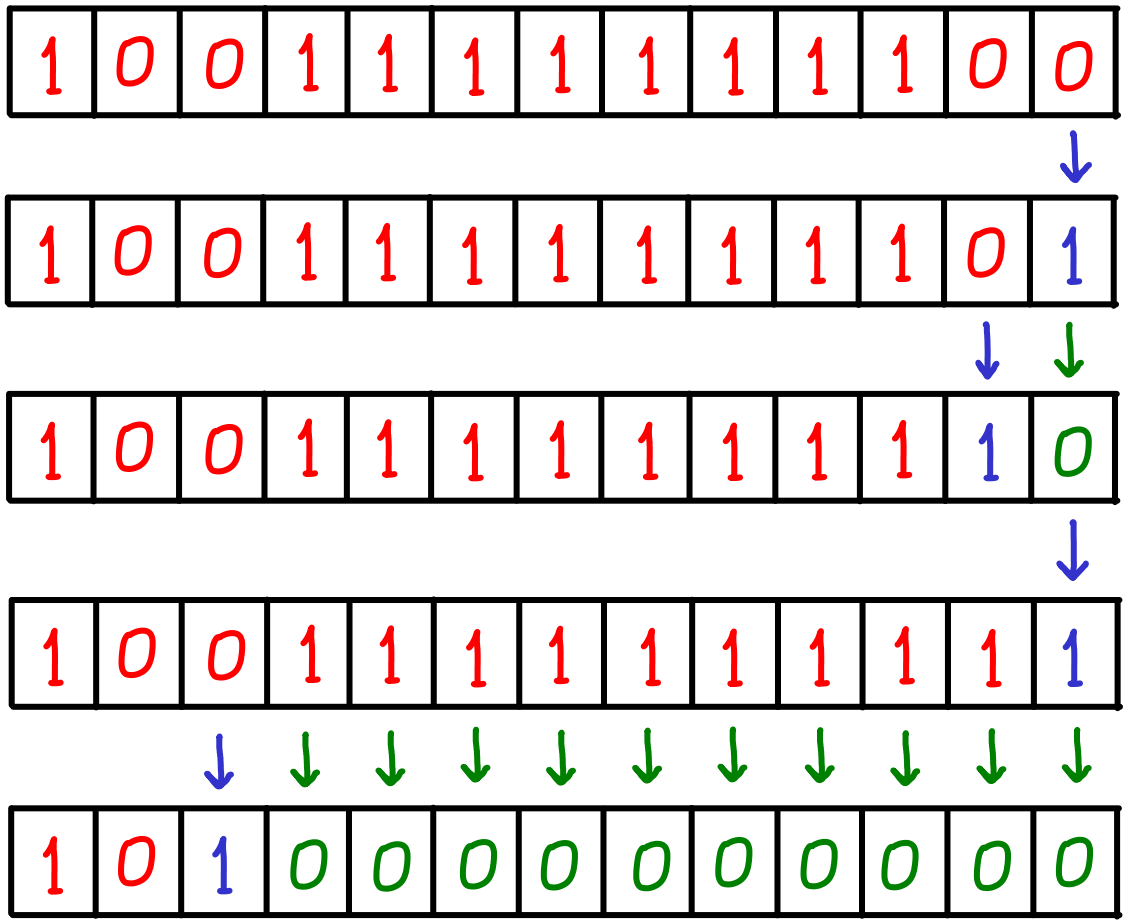
Incrementing a k-bit counter



Incrementing a k-bit counter

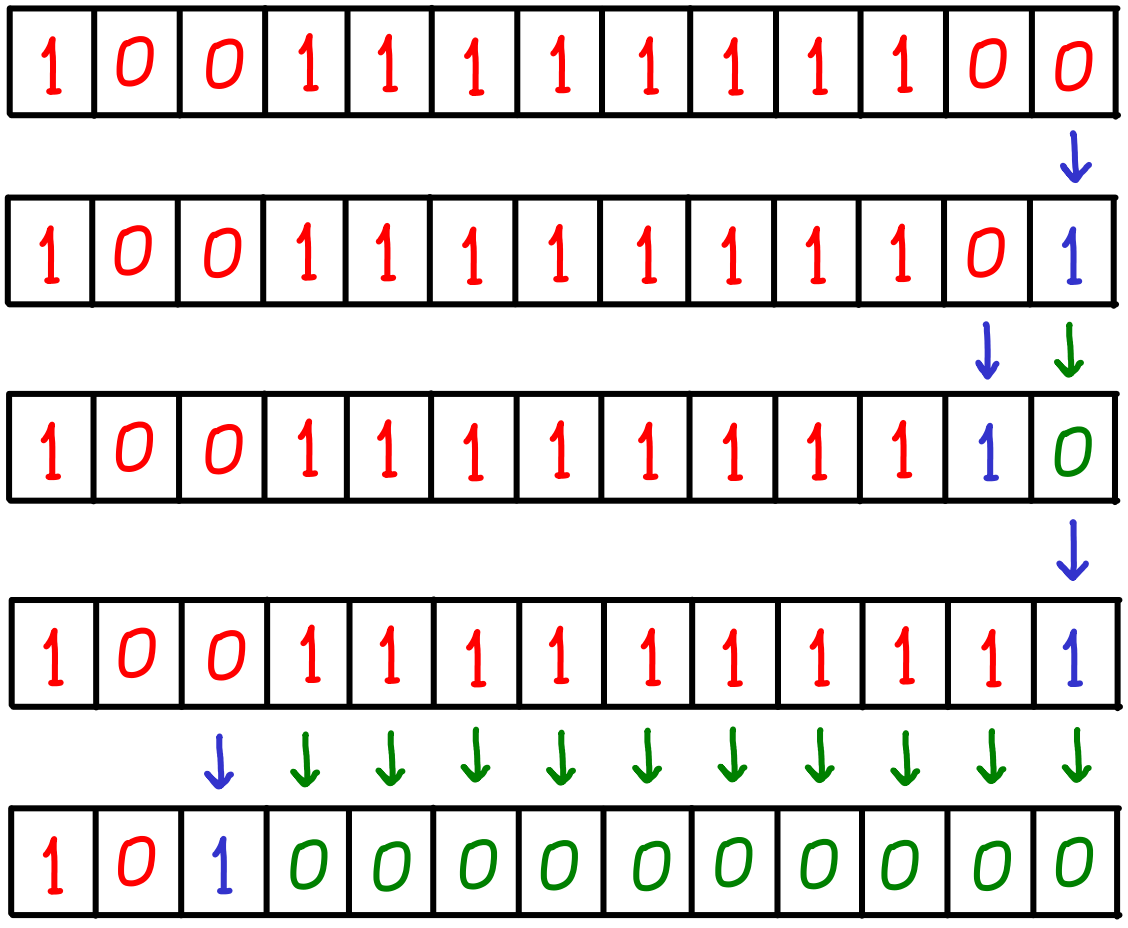


Incrementing a k-bit counter



Incrementing a k-bit counter

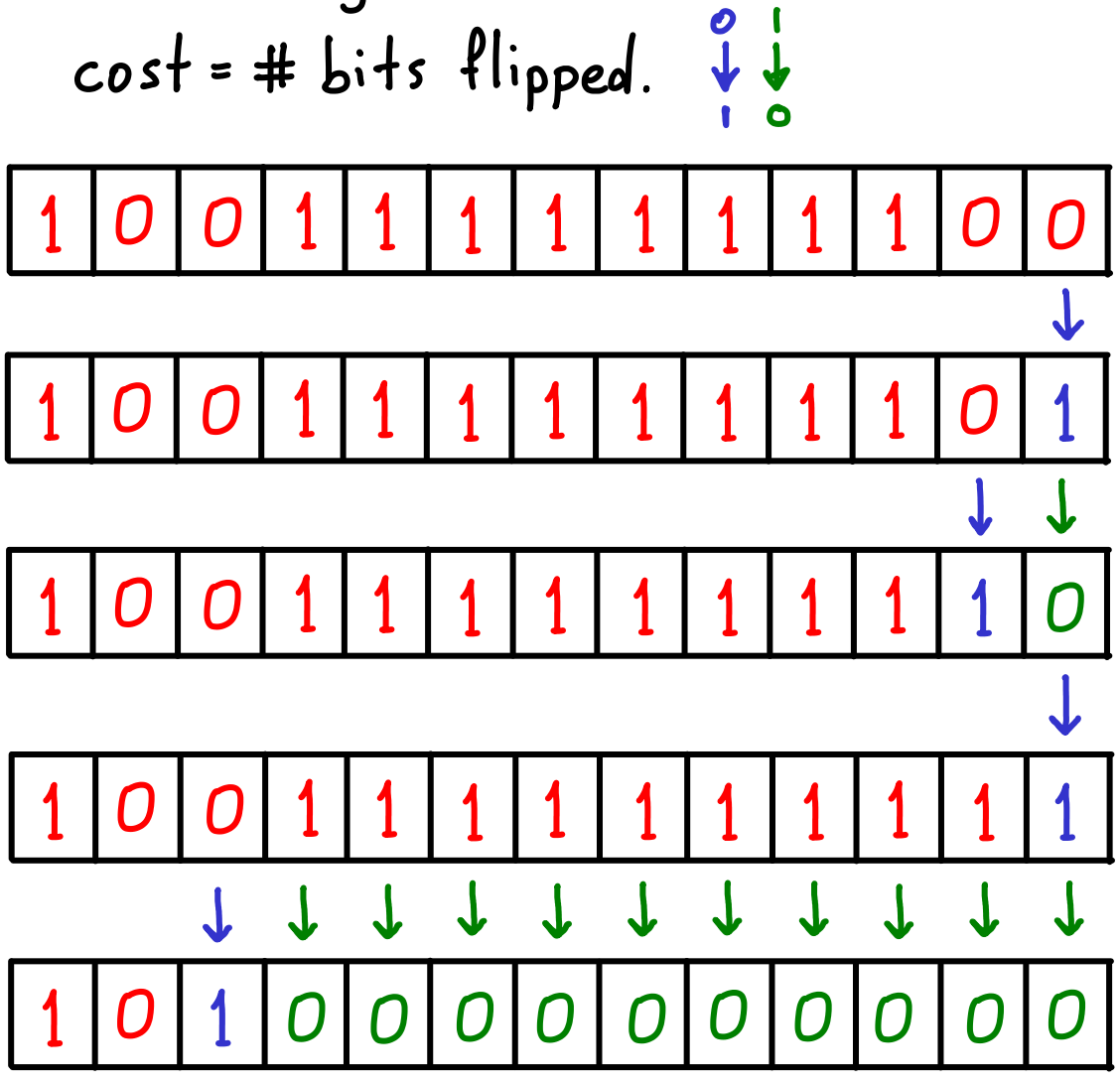
cost = # bits flipped.



Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

($< 2^k$)

cost = # bits flipped.

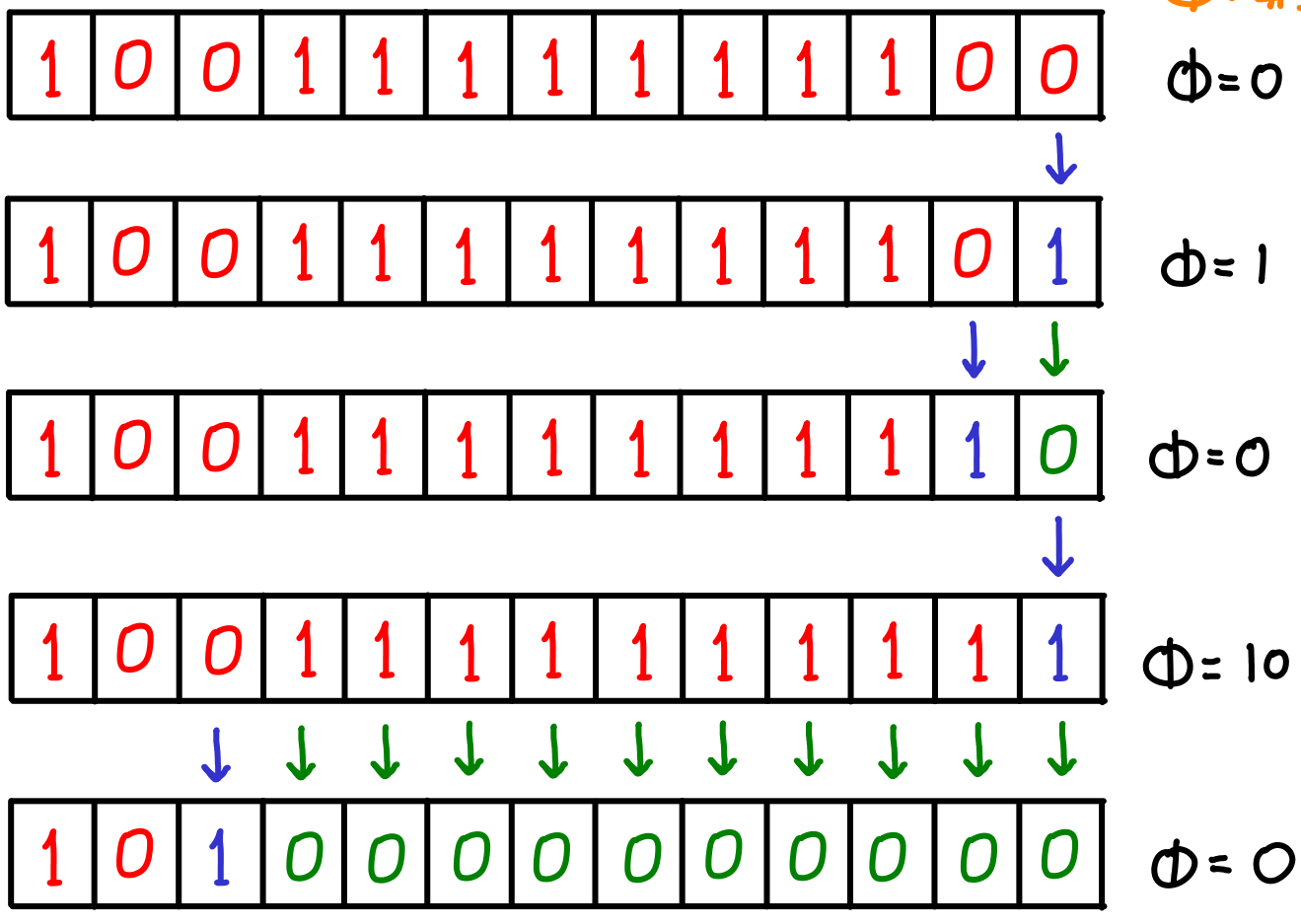


Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.

($< 2^k$)

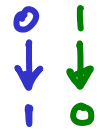
Φ: #leading 1's ?



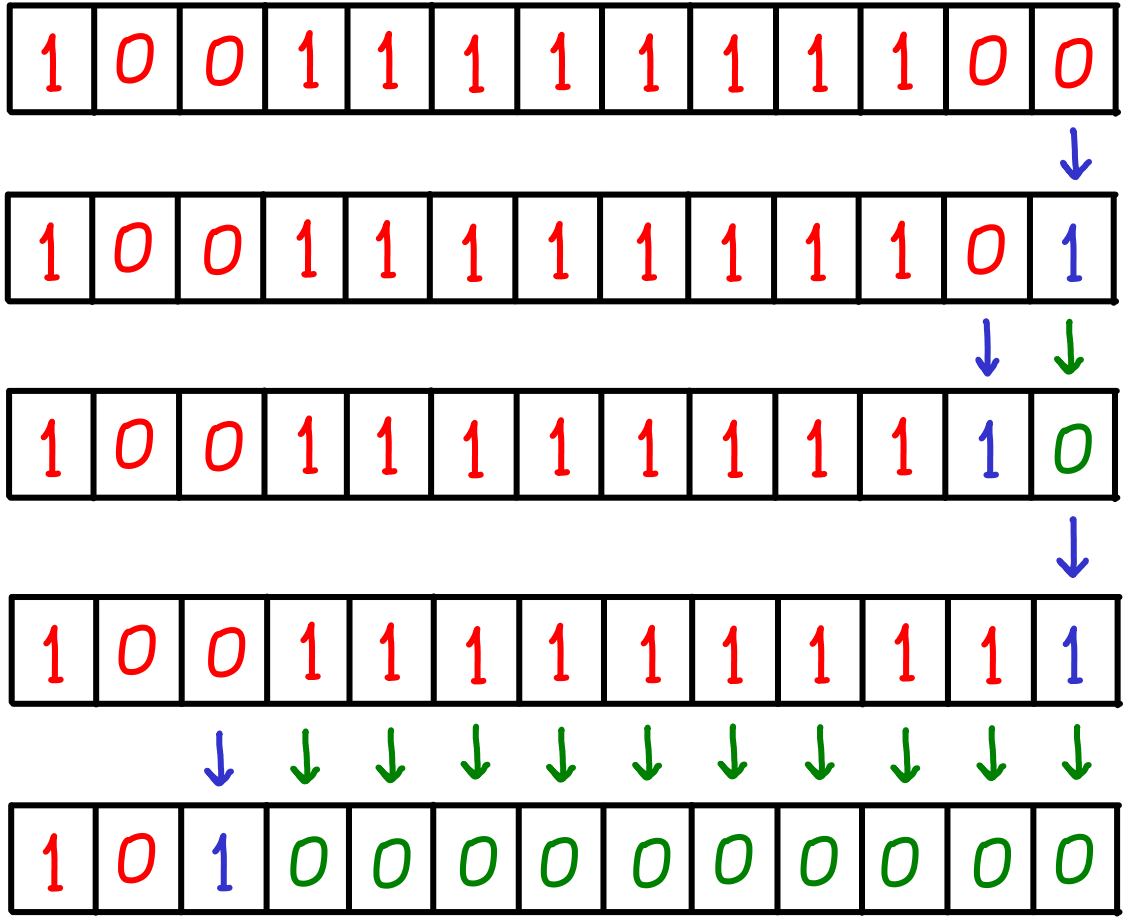
Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.

($< 2^k$)

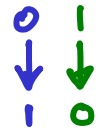


Φ: #leading 1's



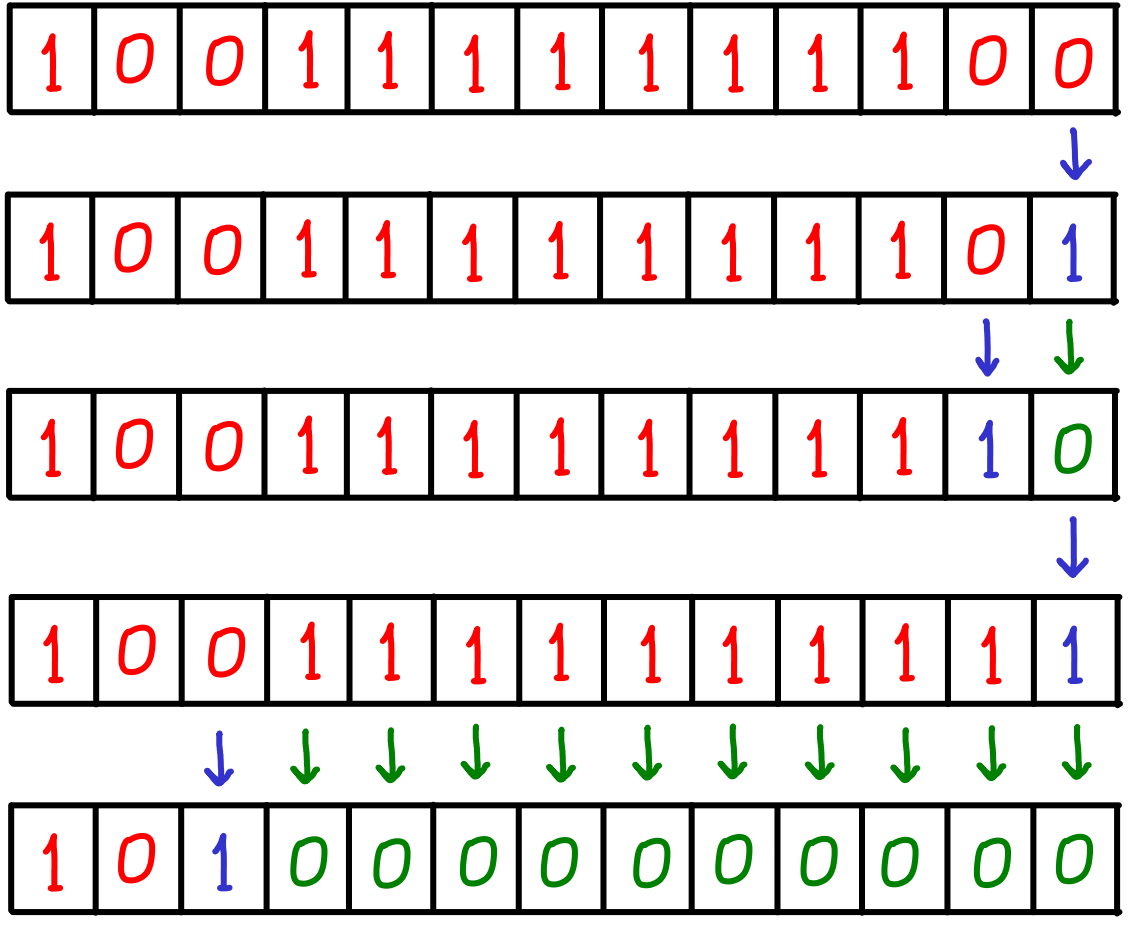
Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.



($< 2^k$)

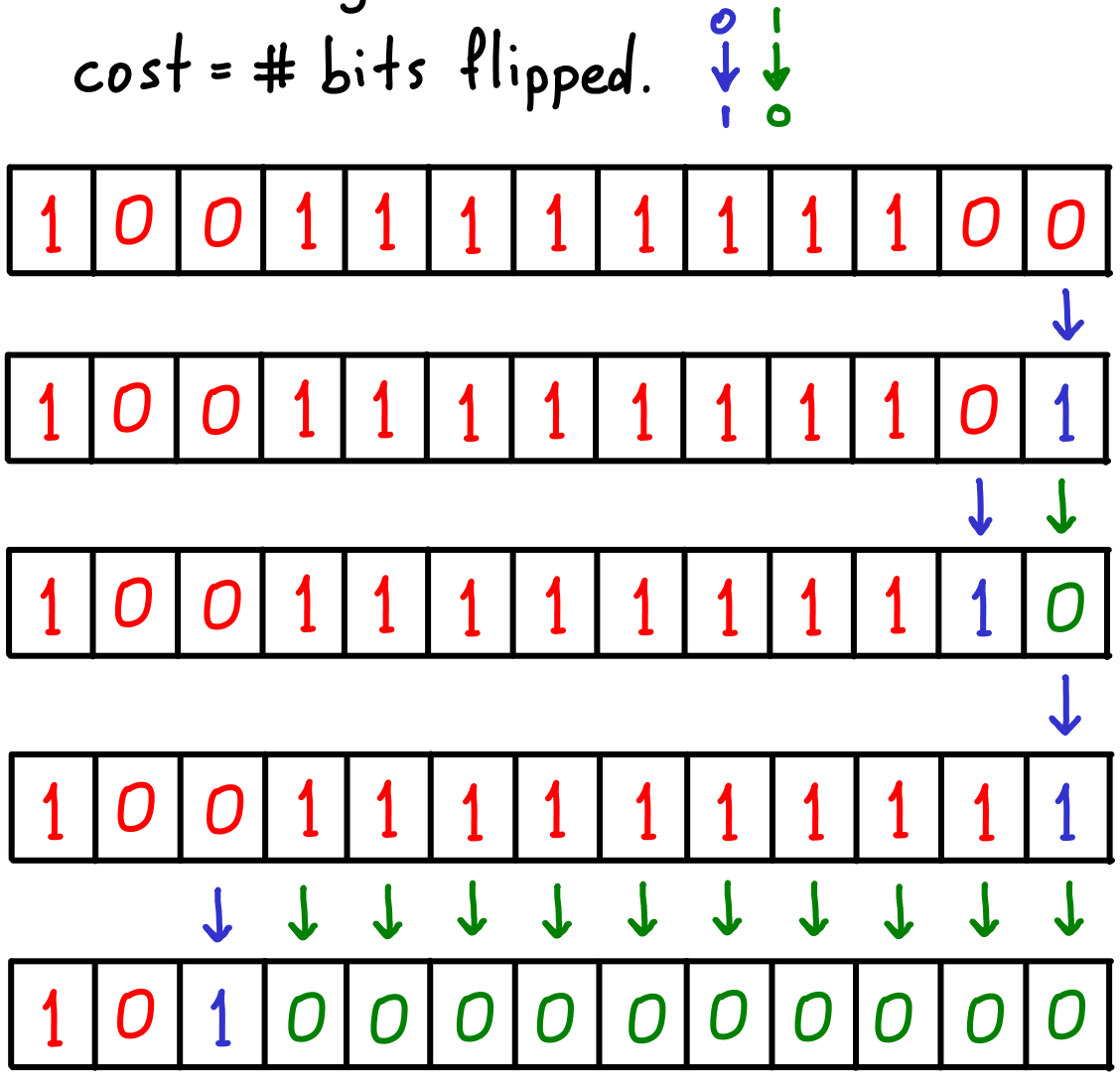
Φ: #leading 1's



Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.

($< 2^k$)



Φ: #leading 1's

$\Phi = 0$

$\hat{c} = 1 + (1 - 0) = 2$

$\Phi = 1$

$\hat{c} = 2 + (0 - 1) = 1$

$\Phi = 0$

$\hat{c} = 1 + (10 - 0) = 11$ → k-2

☹

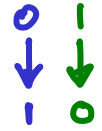
$\Phi = 10$

$\hat{c} = 11 + (0 - 10) = 1$

$\Phi = 0$

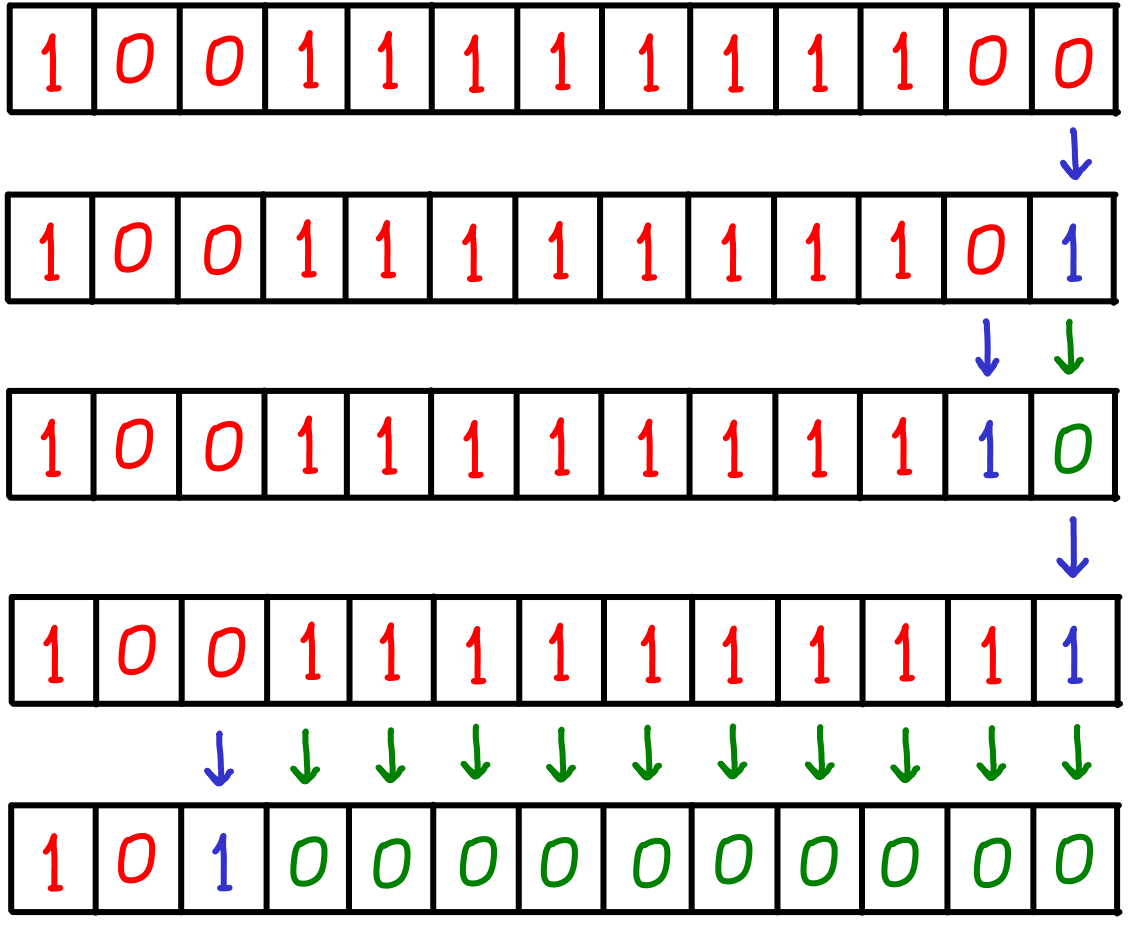
Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.



($< 2^k$)

Φ: #leading 1's



$\Phi = 0$

$\hat{c} = 1 + (1 - 0) = 2$

$\Phi = 1$

$\hat{c} = 2 + (0 - 1) = 1$

$\Phi = 0$

$\hat{c} = 1 + (10 - 0) = 11$ → allowed small c to grow a lot ☹

$\Phi = 10$

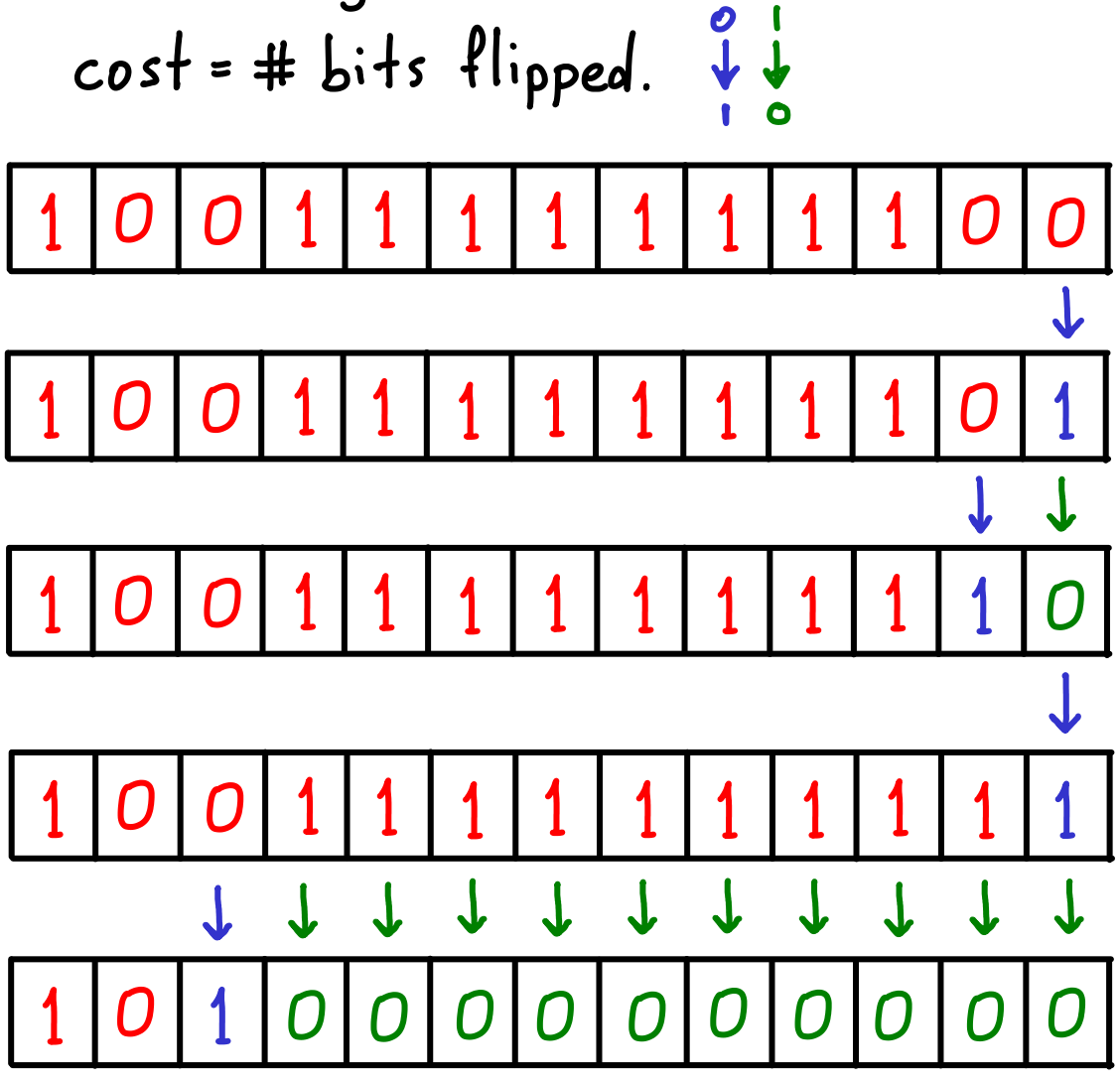
$\hat{c} = 11 + (0 - 10) = 1$ → killed large c ☺

$\Phi = 0$

Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.

($< 2^k$)



Φ: #leading 1's

Φ = 0

$\hat{c} = 1 + (1 - 0) = 2$

Φ = 1

$\hat{c} = 2 + (0 - 1) = 1$

Φ = 0

$\hat{c} = 1 + (10 - 0) = 11$ ☹

Φ = 10

$\hat{c} = 11 + (0 - 10) = 1$

Φ = 0

Φ = total #1's

Φ = 9

Φ = 10

Φ = 10

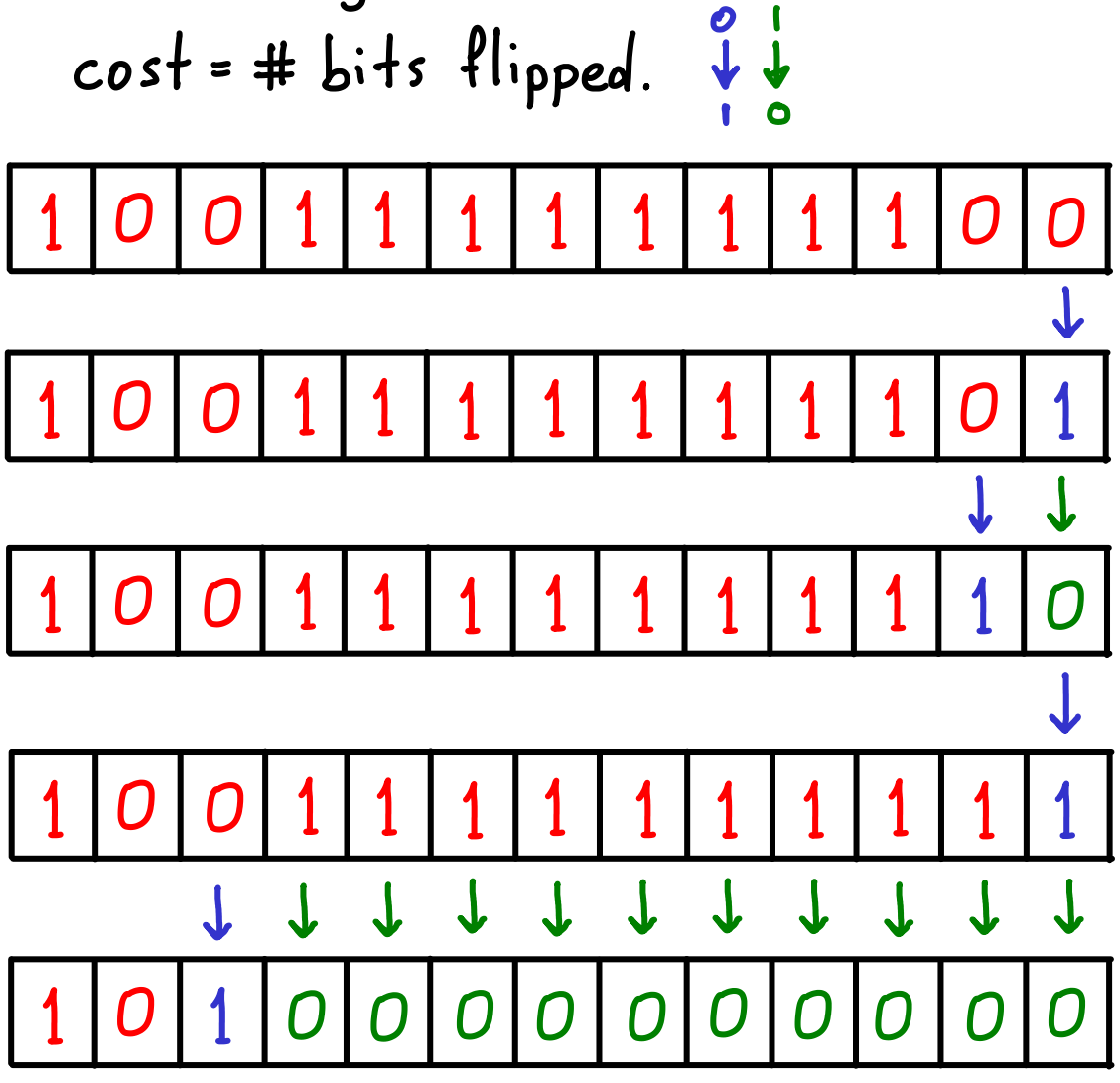
Φ = 11

Φ = 2

Incrementing a k-bit counter: cost of n increments? (assume max = 111111...)

cost = # bits flipped.

($< 2^k$)



Φ: #leading 1's

Φ = 0

$\hat{c} = 1 + (1 - 0) = 2$

Φ = 1

$\hat{c} = 2 + (0 - 1) = 1$

Φ = 0

$\hat{c} = 1 + (10 - 0) = 11$ ☹

Φ = 10

$\hat{c} = 11 + (0 - 10) = 1$

Φ = 0

Φ = total #1's

Φ = 9

$\hat{c} = 1 + (10 - 9) = 2$

Φ = 10

$\hat{c} = 2 + (10 - 10) = 2$

Φ = 10

$\hat{c} = 1 + (11 - 10) = 2$

Φ = 11

$\hat{c} = 11 + (2 - 11) = 2$ ☺

Φ = 2

Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment

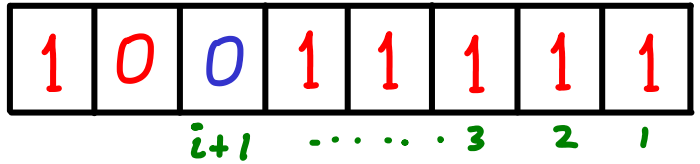
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

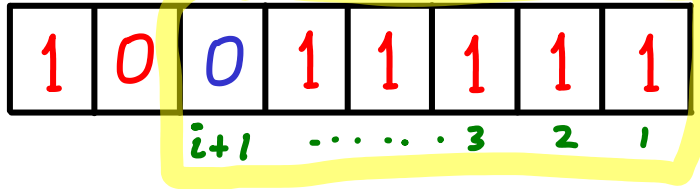
$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1

$c = i+1$



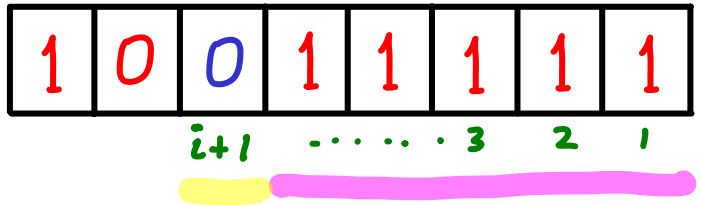
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1 - i$

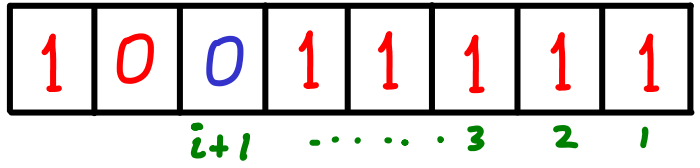
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1-i$

$\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

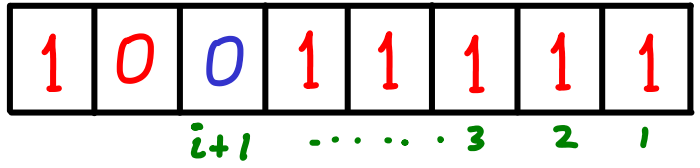
Φ: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1-i$

$\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\rightarrow$ what do we check?

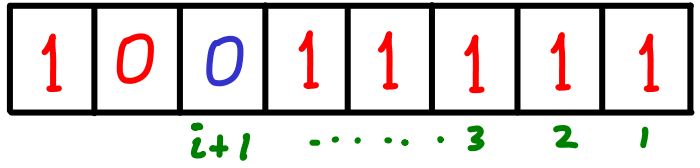
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1 - i$

$\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

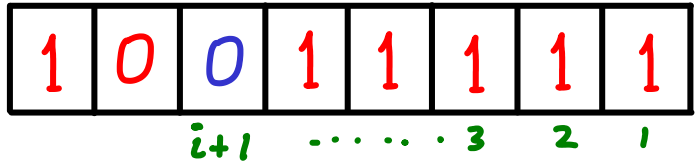
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1 - i$

$\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ×

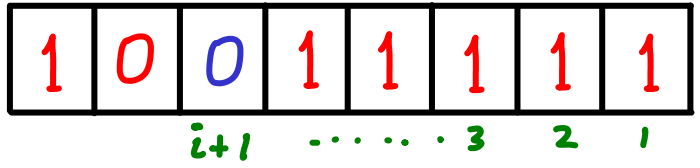
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1 - i$

} $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

$\hookrightarrow = 0$ if we start count up from zero ✓

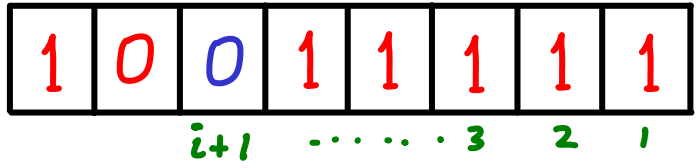
Φ: #leading 1's $\longrightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

↳ $\leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$
$\Delta\Phi = 1 - i$
} $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

↳ = 0 if we start count up from zero ✓

$\Phi \leq k$

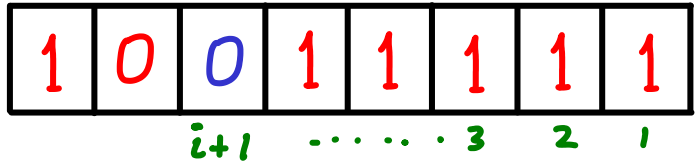
Φ: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$

$\Delta\Phi = 1-i$

$\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

$\hookrightarrow = 0$ if we start count up from zero ✓

$\Phi \leq k \rightarrow \Phi_n - \Phi_0 \geq -k$

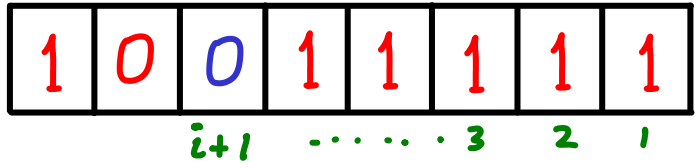
Φ: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$
$\Delta\Phi = 1-i$
} $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

$\hookrightarrow = 0$ if we start count up from zero ✓

$\Phi \leq k \rightarrow \Phi_n - \Phi_0 \geq -k \rightarrow \sum \hat{c} \geq \sum c - k$

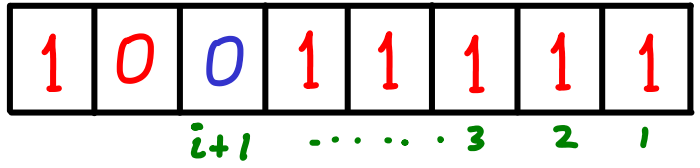
Φ: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

↳ $\leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$
$\Delta\Phi = 1-i$
} $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

↳ = 0 if we start count up from zero ✓

$\Phi \leq k \rightarrow \Phi_n - \Phi_0 \geq -k \rightarrow \sum \hat{c} \geq \sum c - k \rightarrow \sum c \leq k + \sum \hat{c}$

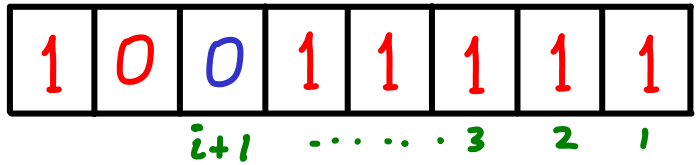
Φ: #leading 1's $\rightarrow$ $\hat{c}$: 2, 1, 11, 1 ... (up to k-1)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

$\hookrightarrow \leq k-1$ per increment ☹

Φ = total #1's

First 0 at position i+1



$c = i+1$
$\Delta\Phi = 1-i$
} $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

$\hookrightarrow = 0$ if we start count up from zero ✓

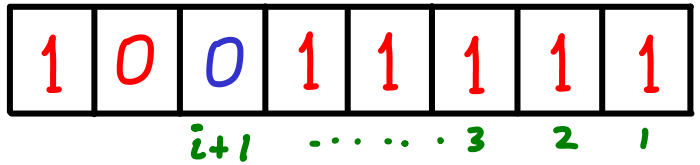
$\Phi \leq k \rightarrow \Phi_n - \Phi_0 \geq -k \rightarrow \sum \hat{c} \geq \sum c - k \rightarrow \sum c \leq k + \sum \hat{c} \rightarrow$

$\rightarrow \sum c \leq k + 2n$

Φ: #leading 1's → $\hat{c}$: 2, 1, 11, 1 ... (up to k-1) $\sum \hat{c} \leq n \cdot \max(\hat{c}) = n(k-1)$

↳ $\leq k-1$ per increment ☹

Φ = total #1's First 0 at position i+1



$c = i+1$, $\Delta\Phi = 1-i$ } $\hat{c}$: 2 (always)

$\sum \hat{c} \leq n \cdot \max(\hat{c}) = 2n$ ☺

But is $\sum c \leq \sum \hat{c}$? $\left(\sum \hat{c} = \sum c + \Phi_n - \Phi_0\right)$

$\Phi_n \geq 0$ but Φ_0 not necessarily 0, so $\Phi_n - \Phi_0$ could be negative ✗

↳ = 0 if we start count up from zero ✓

$\Phi \leq k \rightarrow \Phi_n - \Phi_0 \geq -k \rightarrow \sum \hat{c} \geq \sum c - k \rightarrow \sum c \leq k + \sum \hat{c} \rightarrow$

$\rightarrow \sum c \leq k + 2n \rightarrow$ if $n \geq k$, $\sum c \leq 3n$

Incrementing a k-bit counter

Accounting

Incrementing a k-bit counter

Accounting

Every $0 \to 1$ will cost $\hat{c} = 2$ (instead of $c = 1$)

Use the extra 1 to pay for $1 \to 0$ for the same bit (later)

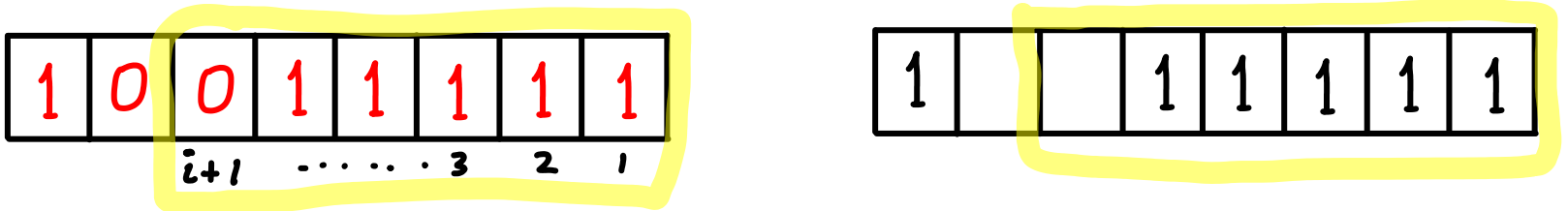
Incrementing a k-bit counter

Accounting

Every $0 \to 1$ will cost $\hat{c} = 2$ (instead of $c = 1$)

Use the extra 1 to pay for $1 \to 0$ for the same bit (later)

↳ $1 \to 0$ will cost $\hat{c} = 0$ ... paid for with savings



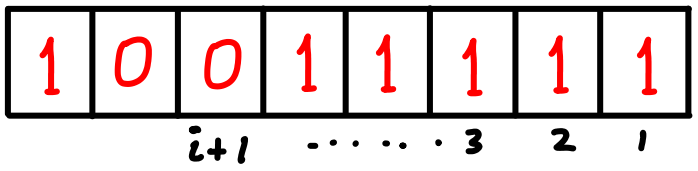
Incrementing a k-bit counter

Accounting

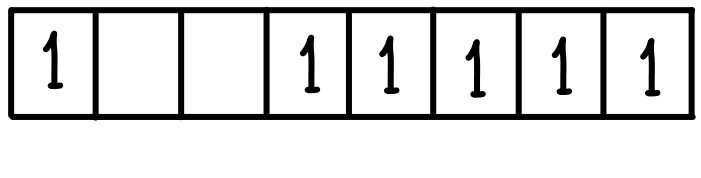
Every $0 \to 1$ will cost $\hat{c} = 2$ (instead of $c = 1$)

Use the extra 1 to pay for $1 \to 0$ for the same bit (later)

↳ $1 \to 0$ will cost $\hat{c} = 0$

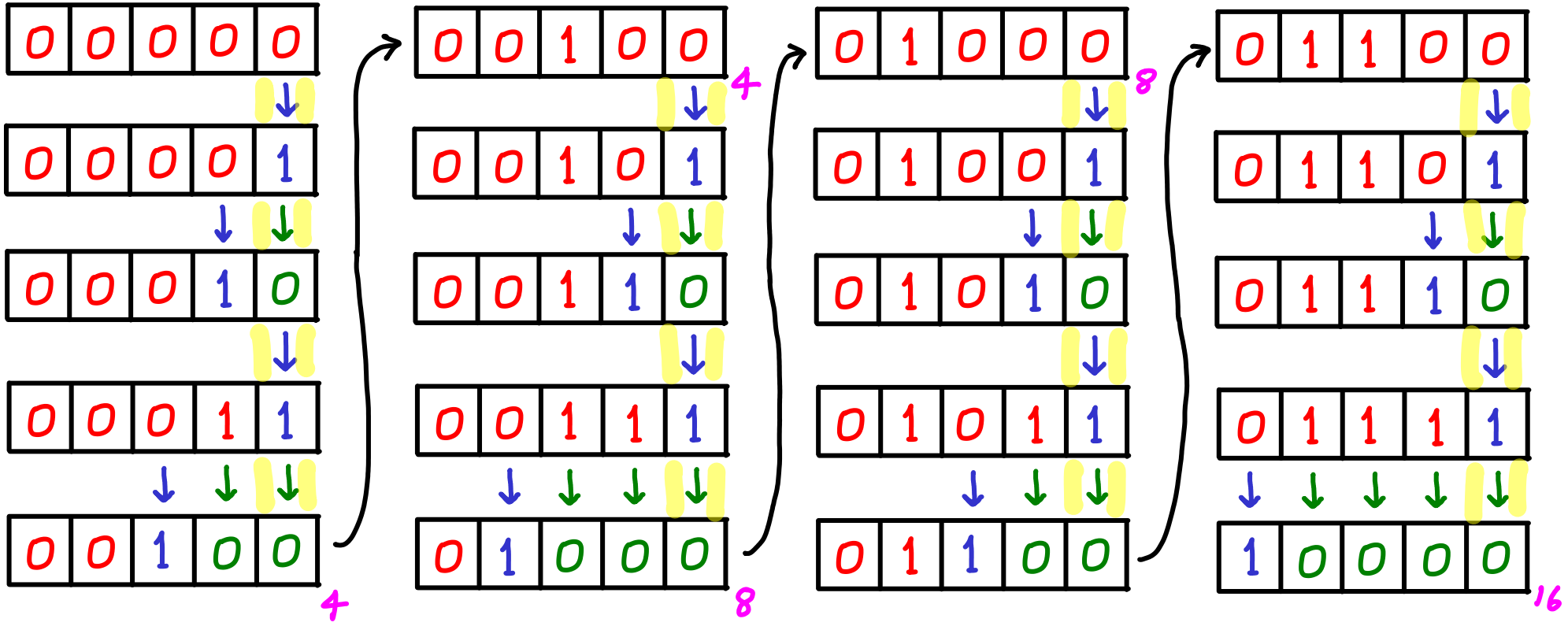


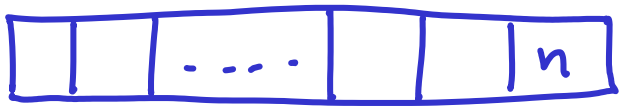
BANK



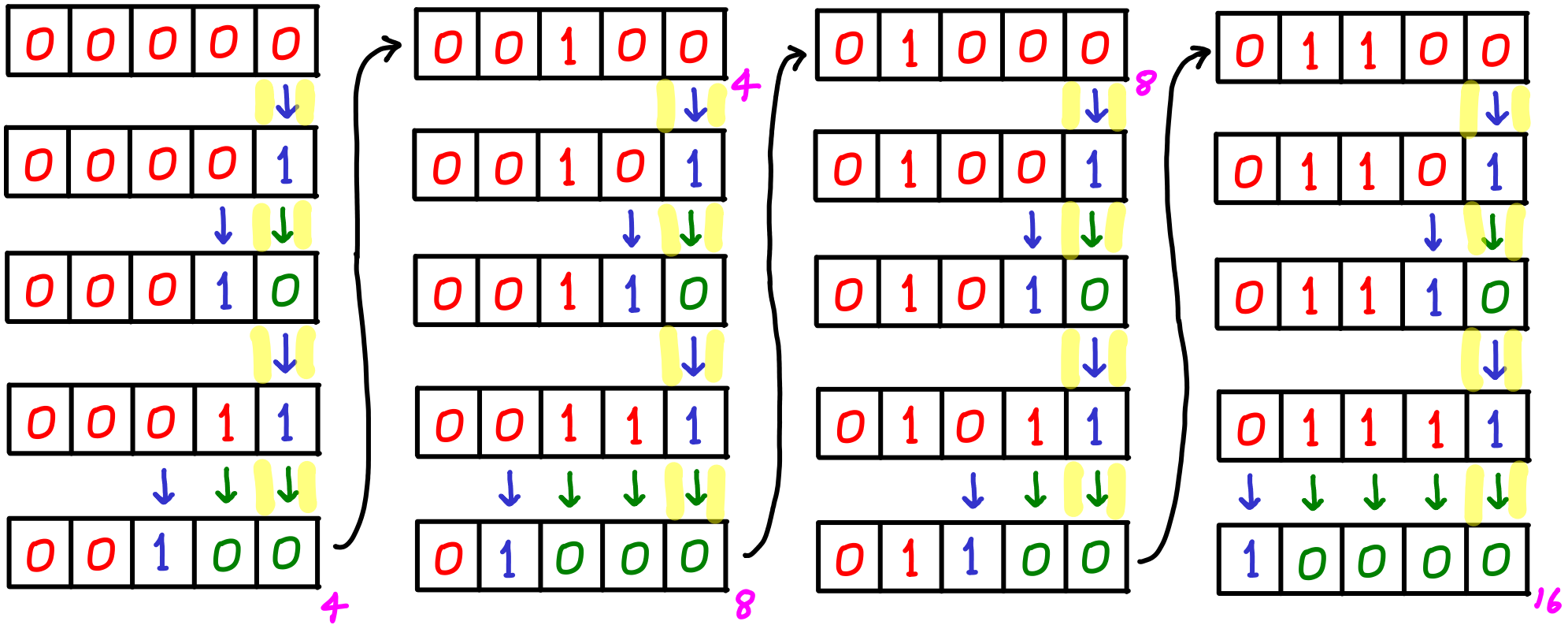
Aggregate method: Just count everything

* relies on full understanding of operation types, frequencies, costs
* not always possible

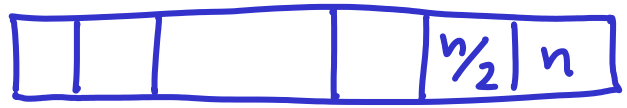


Incrementing a k-bit counter (n times) – aggregate: cost = 

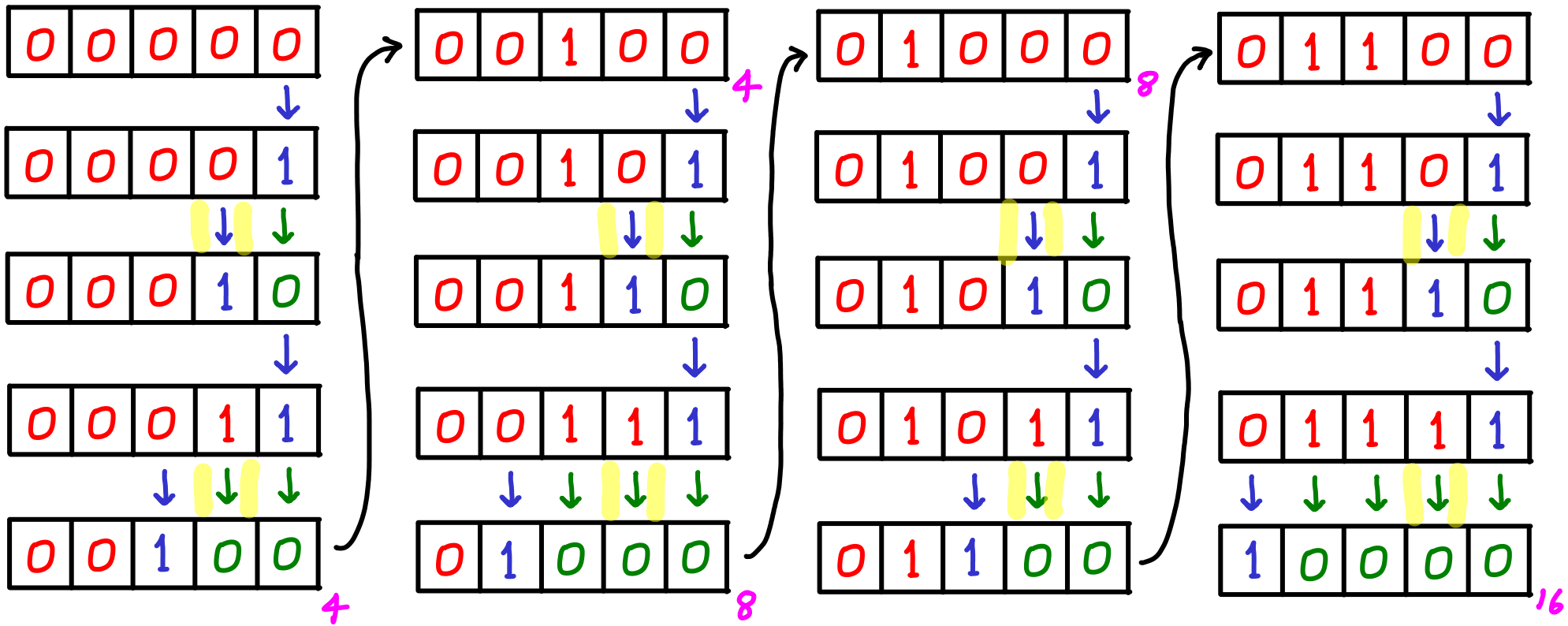
index 0 gets flipped every iteration



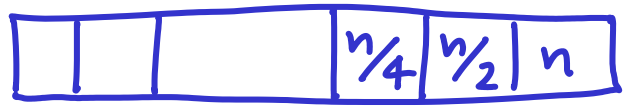
Incrementing a k-bit counter (n times) – aggregate: cost =



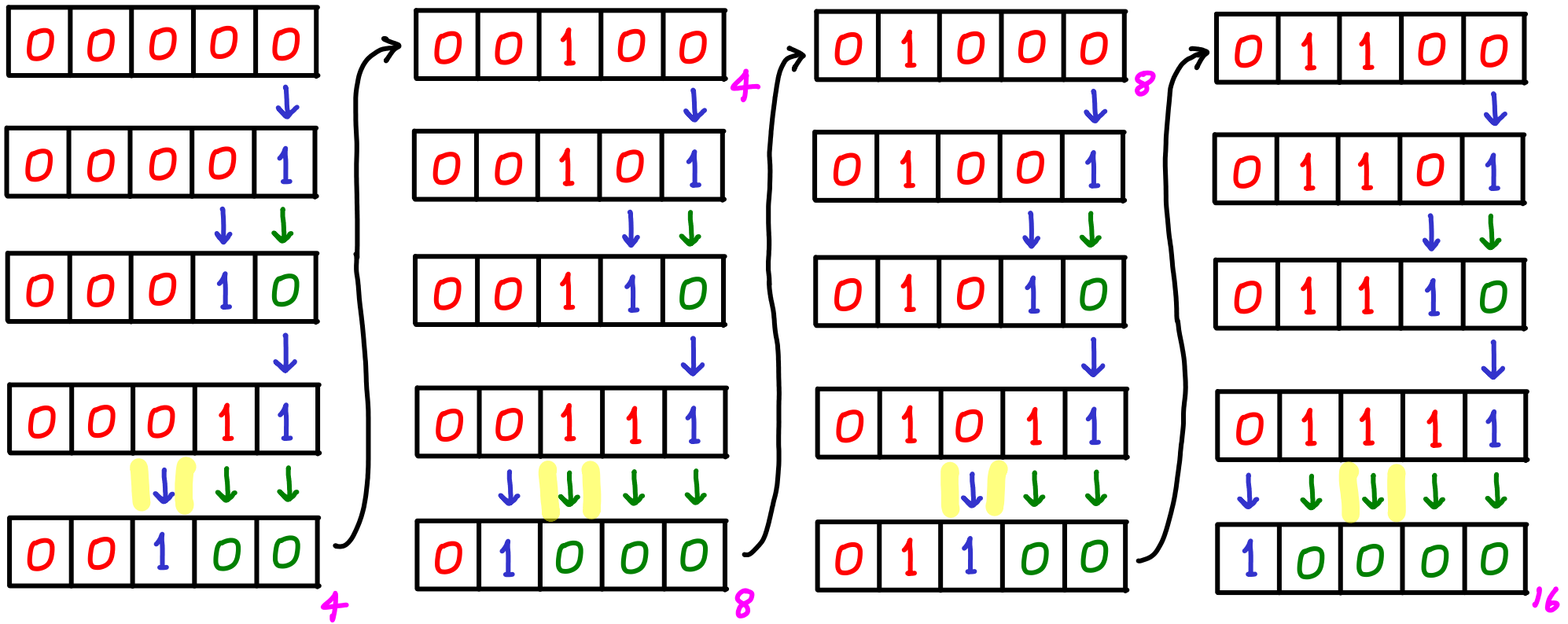
index 1 gets flipped every 2 iterations



Incrementing a k-bit counter (n times) – aggregate: cost =



index 2 gets flipped every 2^2 iterations



Incrementing a k-bit counter (n times) – aggregate: cost = $n/2^i$ | ... | $n/4$ | $n/2$ | n = $2n$

index i gets flipped every 2^i iterations: total $\sum_{i=0}^{n} \frac{n}{2^i}$