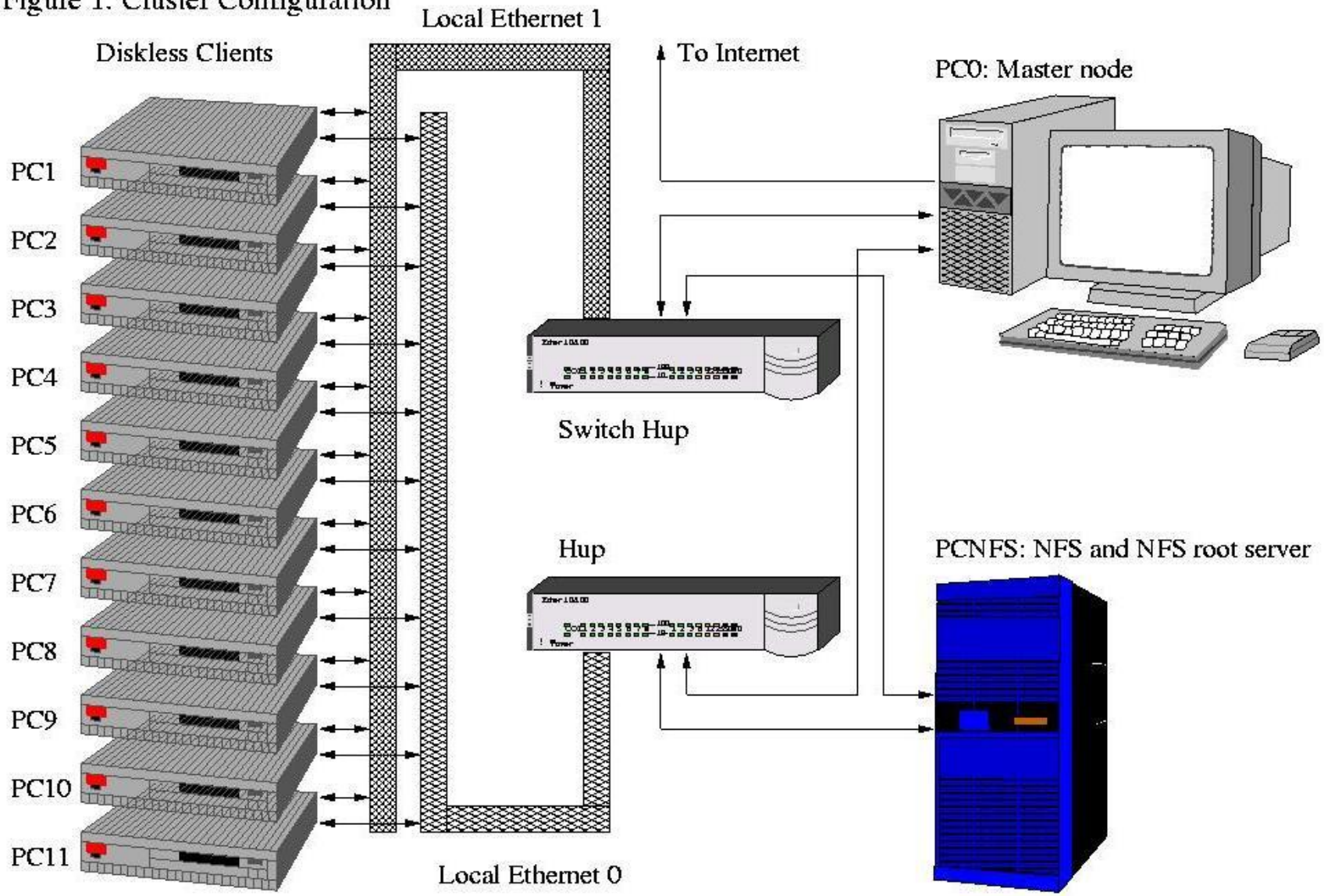
Asynchronous Pattern Matching

Amihood Amir

Yonatan Aumann, Gary Benson,Tzvika Hartman, Oren Kapah, Gadi Landau, Avivit Levy, Ohad Lipsky, Nisan Oz, Ely Porat, Steven Skiena, Uzi Vishne

BGU 2009

Figure 1. Cluster Configuration



Motivation

Motivation

In the "old" days: Pattern and text are given in correct sequential order. It is possible that the content is erroneous – hence, edit distance.

New paradigm: Content is exact, but the order of the pattern symbols may be scrambled.

Why? Transmitted asynchronously?
The nature of the application?

Example: Swaps

Tehse knids of typing mistakes are very common

So when searching for pattern These we are seeking the symbols of the pattern but with an order changed by swaps.

Surprisingly, pattern matching with swaps is easier than pattern matching with mismatches (ACHLP:01)

Example: Reversals

AAAGGCCCTTTGAGCCC

AAAGAGTTTCCCGGCCC

Given a DNA substring, a piece of it can detach and reverse.

This process still computationally tough.

Question: What is the minimum number of reversals necessary to sort a permutation of $\{1,\ldots,n\}$

Global Rearrangements?

Berman & Hannenhalli (1996) called this Global Rearrangement as opposed to Local Rearrangement (edit distance). Showed it is NP-hard.

Our Thesis: This is a special case of errors in the address rather than content.

Example: Transpositions

AAAGGCCCTTTGAGCCC

AATTTGAGGCCCAGCCC

Given a DNA substring, a piece of it can be transposed to another area.

Question: What is the minimum number of transpositions necessary to sort a permutation of $\{1,\ldots,n\}$?

Complexity?

Bafna & Pevzner (1998), Christie (1998), Hartman (2001): 1.5 Polynomial Approximation.

Not known whether efficiently computable.

This is another special case of errors in the address rather than content.

Example: Block Interchanges

AAAGGCCCTTTGAGCCC

AAGTTTAGGCCCAGCCC

Given a DNA substring, two non-empty subsequences can be interchanged.

Question: What is the minimum number of block interchanges necessary to sort a permutation of $\{1,\ldots,n\}$?

Christie (1996): $O(n^2)$

Summary

Biology: sorting permutations

Reversals — NP-hard
(Berman & Hannenhalli, 1996)

Transpositions — ?
(Bafna & Pevzner, 1998)

Block interchanges — $O(n^2)$
(Christie, 1996)

Pattern Matching:

Swaps — $O(n \log m)$
(Amir, Lewenstein & Porat, 2002)

Note: A swap is a block interchange simplification

1. Block size
2. ~~Only once~~
3. ~~Adjacent~~

Edit operations map

Reversal, Transposition, Block interchange:

1. arbitrary block size 2. not once 3. non adjacent
4. permutation 5. optimization

Interchange:

1. block of size 1 2. not once 3. non adjacent
4. permutation 5. optimization

Generalized-swap: ($O(1)$ time in parallel)

1. block of size 1 2. once 3. non adjacent
4. repetitions 5. optimization/decision

Swap:

1. block of size 1 2. once 3. adjacent
4. repetitions 5. optimization/decision

Models map

Pattern Matching:

slide pattern along text.

Nearest Neighbor:

pattern and text same size.

Permutation (Ulam):

no repeating symbols.

Definitions

S=abacb $\xrightarrow{\text{interchange}}$ F=bbaca

S=abacb $\xrightarrow[\text{matches}]{\text{interchange}}$ F=bbaac

S_1=bbaca

S_2=bbaac

S=abacb $\xrightarrow[\text{matches}]{\text{generalized-swap}}$ F=bcaba

S_1=bbaca

S_2=bcaba

O(1) time parallel

Generalized Swap Matching

INPUT: text T[0..n], pattern P[0..m]

OUTPUT: all i s.t. P generalized-swap matches T[i..i+m]

Reminder: Convolution

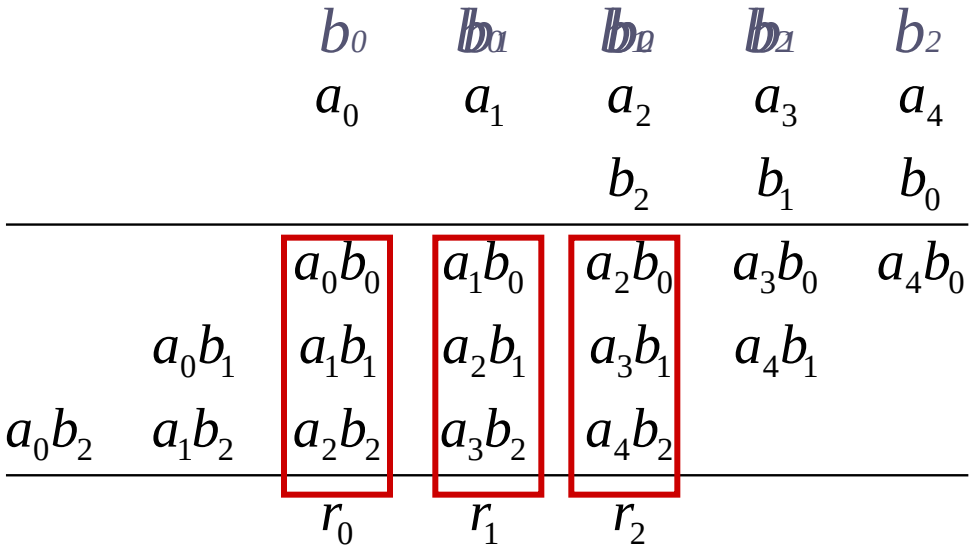
The convolution of the strings t[1..n] and p[1..m] is the string t*p such that:

$$(t*p)[i]=\sum_{k=1,m}(t[i+k-1]\cdot p[m-k+1]) \quad \text{for all } 1\leq i \leq n-m$$

Fact: The convolution of n-length text and m-length pattern can be done in $O(n \log m)$ time using FFT.

In Pattern Matching

Convolutions:



$O(n \log m)$ using FFT

Generalized Swap Matching: a Randomized Algorithm…

Idea: assign natural numbers to alphabet symbols, and construct:

T′: replacing the number a by the pair a^2,-a

P′: replacing the number b by the pair b, b^2.

Convolution of T′ and P′ gives at every location 2i:

$$\sum_{j=0..m} h(T'[2i+j],P'[j])$$

where h(a,b)=ab(a-b).

→ 3-degree multivariate polynomial.

Generalized Swap Matching: a Randomized Algorithm…

Since: $h(a,a)=0$

$h(a,b)+h(b,a)=ab(b-a)+ba(a-b)=0$,

a generalized-swap match $\Rightarrow$ 0 polynomial.

Example:

Text: ABCBAABBC

Pattern: CCAABABBB

1 -1, 4 -2, 9 -3,4 -2,1 -1,1 -1,4 -2,4 -2,9 -3

3 9, 3 9, 1 1,1 1,2 4, 1 1,2 4, 2 4,2 4

---

3 -9,12 -18,9 -3,4 -2,2 -4,1 -1,8 -8,8 -8,18 -12

Generalized Swap Matching: a Randomized Algorithm…

Problem: It is possible that coincidentally the result will be 0 even if no swap match.

Example: for text ace and pattern bdf we get a multivariate degree 3 polynomial:

$$a^2b - ab^2 + c^2d - cd^2 + e^2f - ef^2 = 0$$

We have to make sure that the probability for such a possibility is quite small.

Generalized Swap Matching: a Randomized Algorithm…

What can we say about the 0's of the polynomial?

By Schwartz-Zippel Lemma prob. of 0≤degree/|domain|.

Conclude:

Theorem: There exist an $O(n \log m)$ algorithm that reports all generalized-swap matches and reports false matches with prob.≤1/n.

Generalized Swap Matching:
De-randomization?

Can we detect 0's thus de-randomize the algorithm?

Suggestion: Take $h_1,\ldots h_k$ having no common root.

It won't work,

k would have to be too large !

Generalized Swap Matching: De-randomization?...

Theorem: $\Omega(m/\log m)$ polynomial functions are required to guarantee a 0 convolution value is a 0 polynomial.

Proof: By a linear reduction from word equality.

Given: m-bit words w1 w2 at processors P1 P2

Construct: T=w1,1,2,…,m P=1,2,…,m,w2.

Now, T generalized-swap matches P iff w1=w2.

P1 computes: w1 * (1,2,…,m) → log m bit result → P2 computes: (1,2,…,m) * w2

Communication Complexity:

word equality requires exchanging $\Omega(m)$ bits,

We get: $k \cdot \log m = \Omega(m)$, so k must be $\Omega(m/\log m)$.

Interchange Distance Problem

INPUT: text T[0..n], pattern P[0..m]

OUTPUT: The minimum number of interchanges s.t. T[i..i+m] interchange matches P.

Reminder: permutation cycle

The cycles (143) 3-cycle, (2) 1-cycle represent 3241.

Fact: The representation of a permutation as a product of disjoint permutation cycles is unique.

Interchange Distance Problem…

Lemma: Sorting a k-length permutation cycle requires exactly k-1 interchanges.

Proof: By induction on k. Cases: (1), (2 1), (3 1 2)

Theorem: The interchange distance of an m-length permutation π is $m-c(\pi)$, where $c(\pi)$ is the number of permutation cycles in π.

Result: An $O(nm)$ algorithm to solve the interchange distance problem.

Tighten connection between sorting by interchanges and generalized-swap matching…

Parallel Interchange Operations Problem

INPUT: text T[0..n], pattern P[0..m]

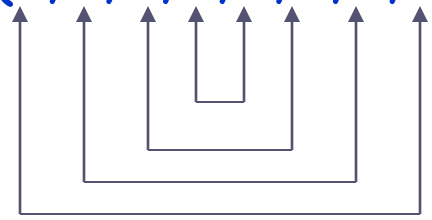
OUTPUT: The minimum number of parallel interchange operations s.t. T[i..i+m] interchange matches P.

Definition: Let $S=S_1,S_2,\ldots,S_k=F$, S_{l+1} derived from S_l via interchange I_l. A parallel interchange operation is a subsequence of $I_1,\ldots,I_{k-1}$ s.t. the interchanges have no index in common.

Parallel Interchange Operations Problem…

Lemma: Let σ be a cycle of length k>2. It is possible to sort σ in 2 parallel interchange operations (k-1 interchanges).

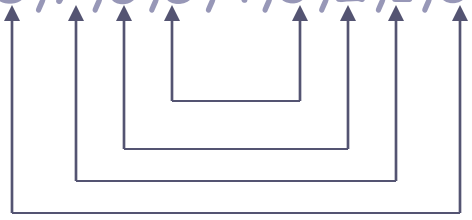
Example: (1,2,3,4,5,6,7,8,0)



generation 1:

(1,8),(2,7),(3,6),(4,5)

(8,7,6,5,4,3,2,1,0)



generation 2:

(0,8),(1,7),(2,6),(3,5)

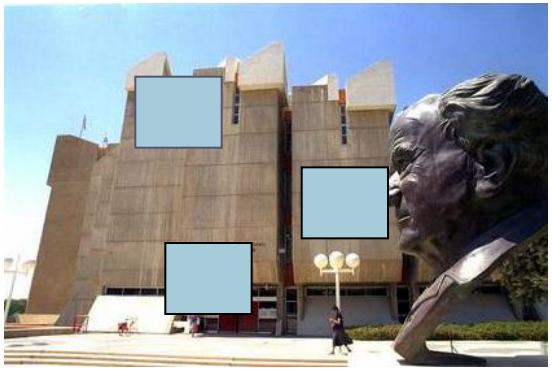
(0,1,2,3,4,5,6,7,8)

Parallel Interchange Operations Problem…

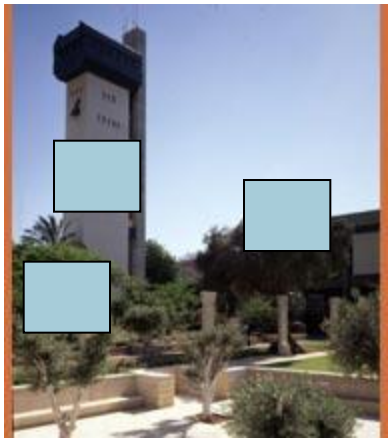
Theorem: Let maxl(π) be the length of the longest permutation cycle in an m-length permutation π. The number of parallel interchange operations required to sort π is exactly:

1. 0, if maxl(π)=1.
2. 1, if maxl(π)=2.
3. 2, if maxl(π)>2.

Error in Content:



Ben Gurion University



Bar-Ilan University

Error in Address:

Ben Gurion University

Bar-Ilan University

Motivation: Architecture.

Assume distributed memory.

Our processor has text and requests pattern of length m.

Pattern arrives in m asynchronous packets, of the form:

<symbol, addr>

Example: <A, 3>, <B, 0>, <A, 4>, <C, 1>, <B, 2>

Pattern: BCBAA

What Happens if Address Bits Have Errors?

In Architecture:

1. Checksums.
2. Error Correcting Codes.
3. Retransmits.

We would like…

To avoid extra transmissions.

For every text location compute the minimum number of address errors that can cause a mismatch in this location.

Our Model…

Text: T[0],T[1],…,T[n]

Pattern: P[0]=<C[0],A[0]>, P[1]=< C[1],A[1]>, …,
P[m]=<C[m],A[m]>;

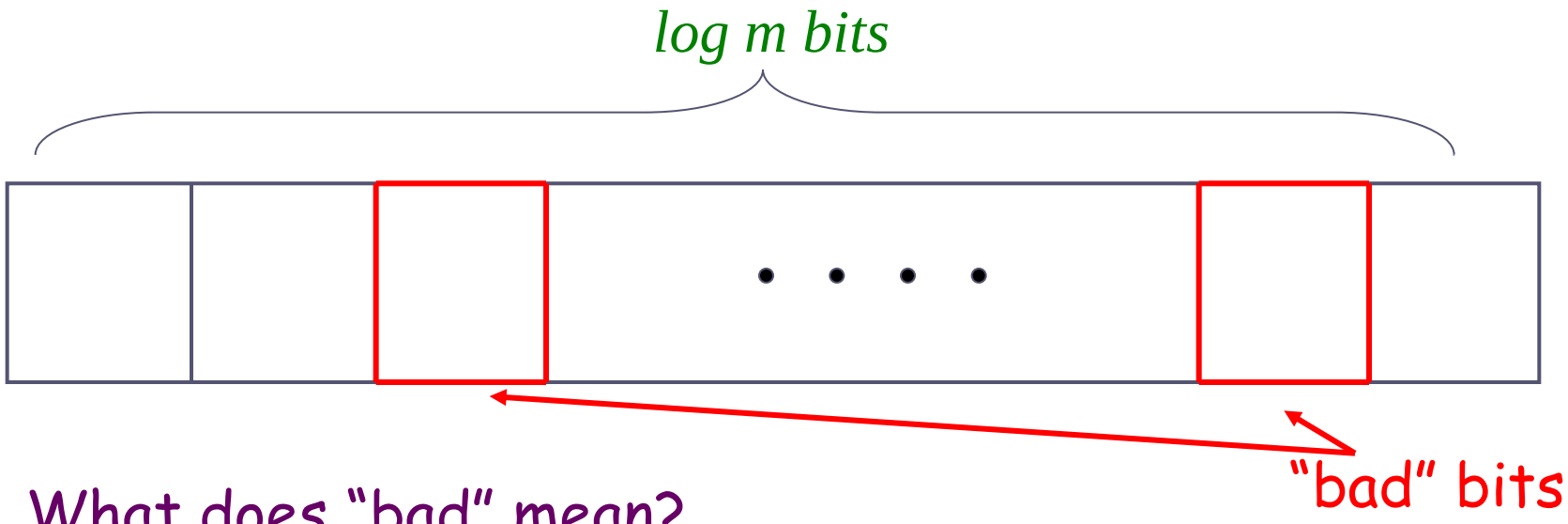
$C[i] \in \Sigma$, $A[i] \in \{1,\ldots,m\}$.

Standard pattern Matching: no error in A.

Asynchronous Pattern Matching: no error in C.

Eventually: error in both.

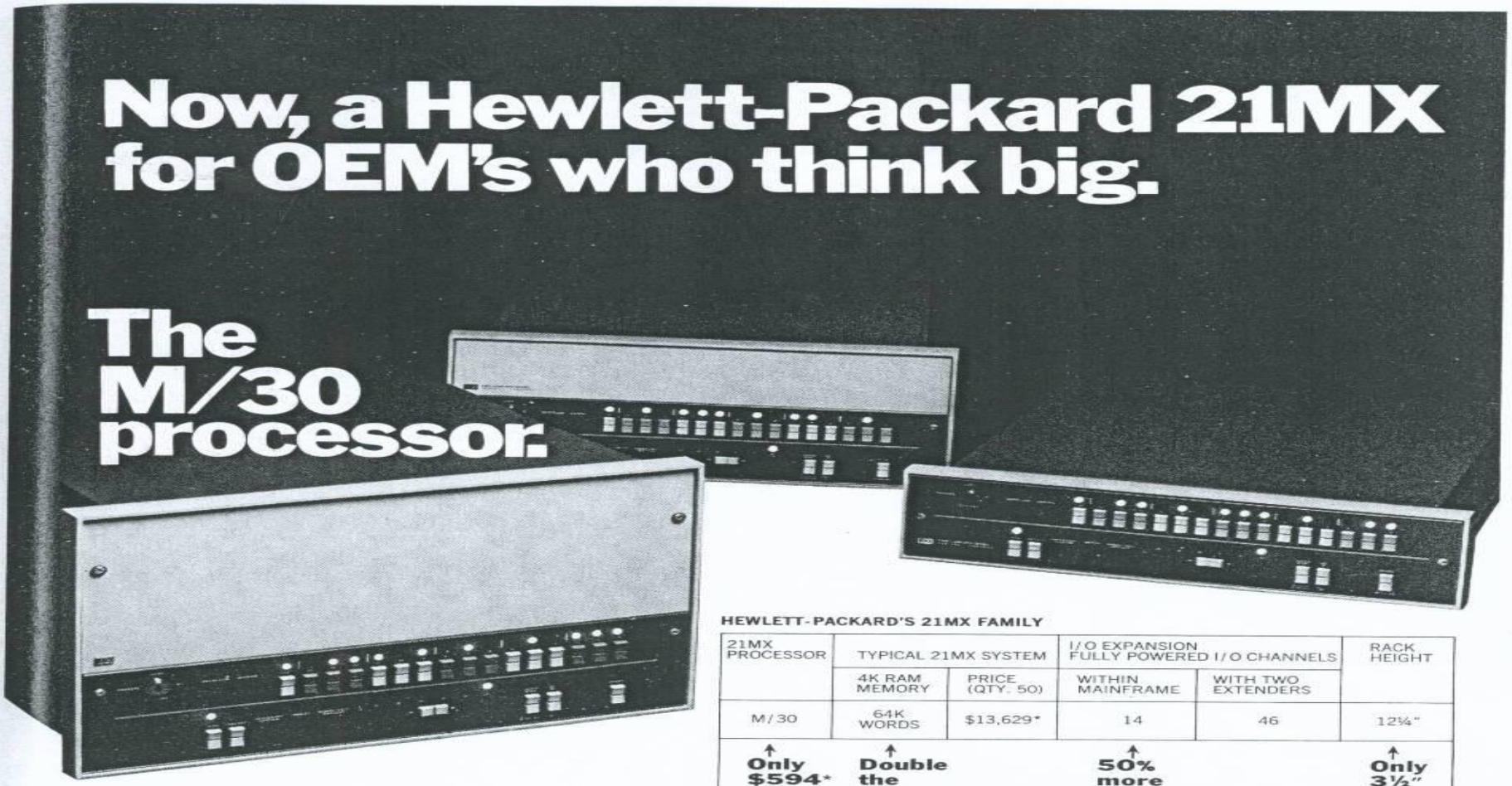
Address Register



What does "bad" mean?

1. bit "flips" its value.
2. bit sometimes flips its value.
3. Transient error.
4. "stuck" bit.
5. Sometimes "stuck" bit.

Bad Bits



Now Hewlett-Packard brings 4K RAM semiconductor technology to OEM's with large memory needs. Compatible with all previous 21MX minicomputers, the new M/30 offers the same modularity, the same flexible user microprogrammability.

Not only does 4K RAM technology bring you more reliability, you also benefit from our recent February 30% price reduction on memories. (Send for your free

HEWLETT-PACKARD'S 21MX FAMILY

21MX PROCESSOR	TYPICAL 21MX SYSTEM		I/O EXPANSION FULLY POWERED I/O CHANNELS		RACK HEIGHT
	4K RAM MEMORY	PRICE (QTY. 50)	WITHIN MAINFRAME	WITH TWO EXTENDERS	
M/30	64K WORDS	$13,629*	14	46	12¼"
	↑ Only $594* more	↑ Double the memory		↑ 50% more I/O slots	↑ Only 3½" more rack space
M/20	32K WORDS	$ 7,788*	9	41	8¾"
M/10	16K WORDS	$ 5,049*	4	36	5¼"

*Domestic USA OEM prices quantity 50.

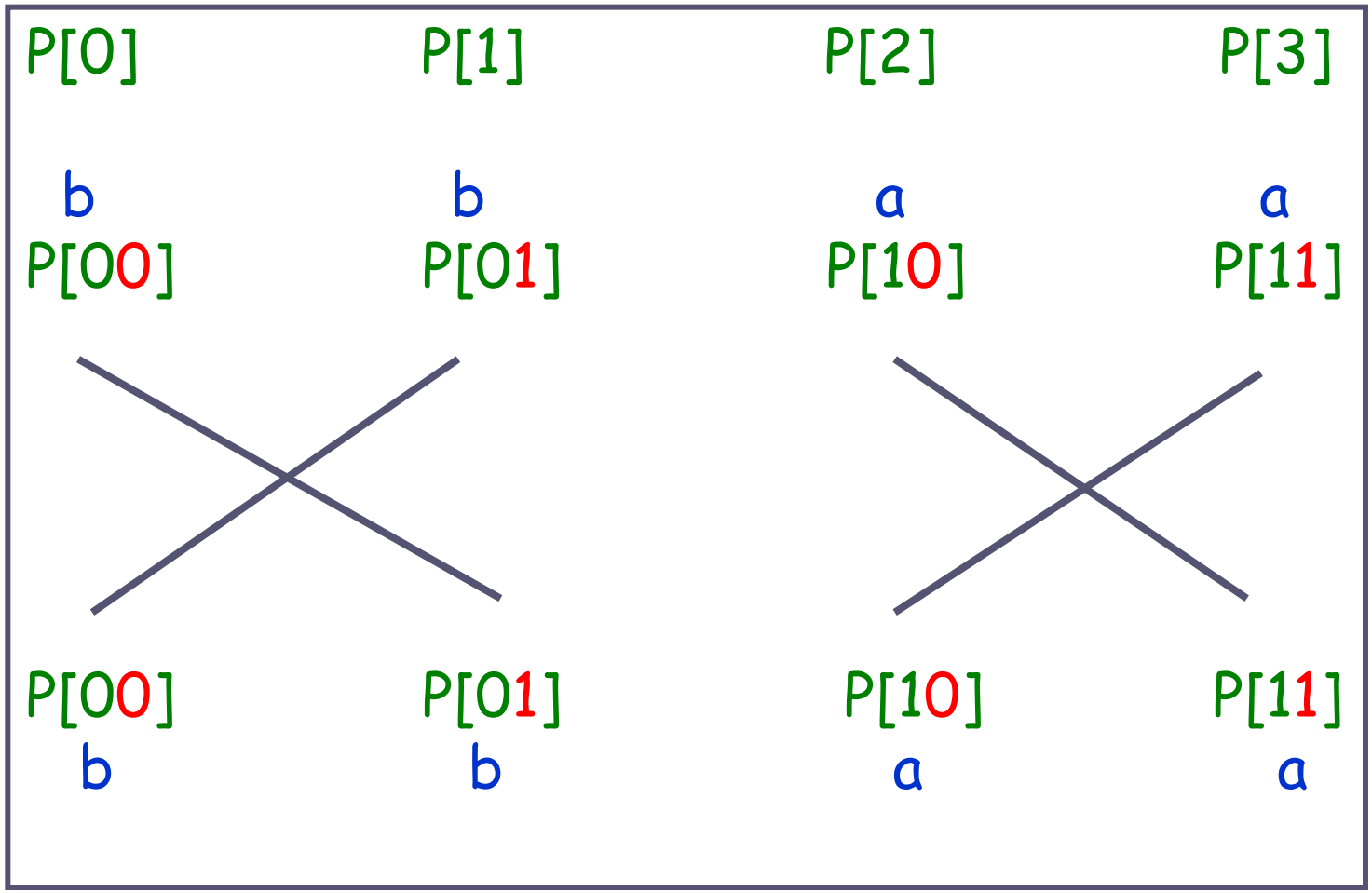
We will now concentrate on consistent bit flips

Example: Let $\Sigma=\{a,b\}$

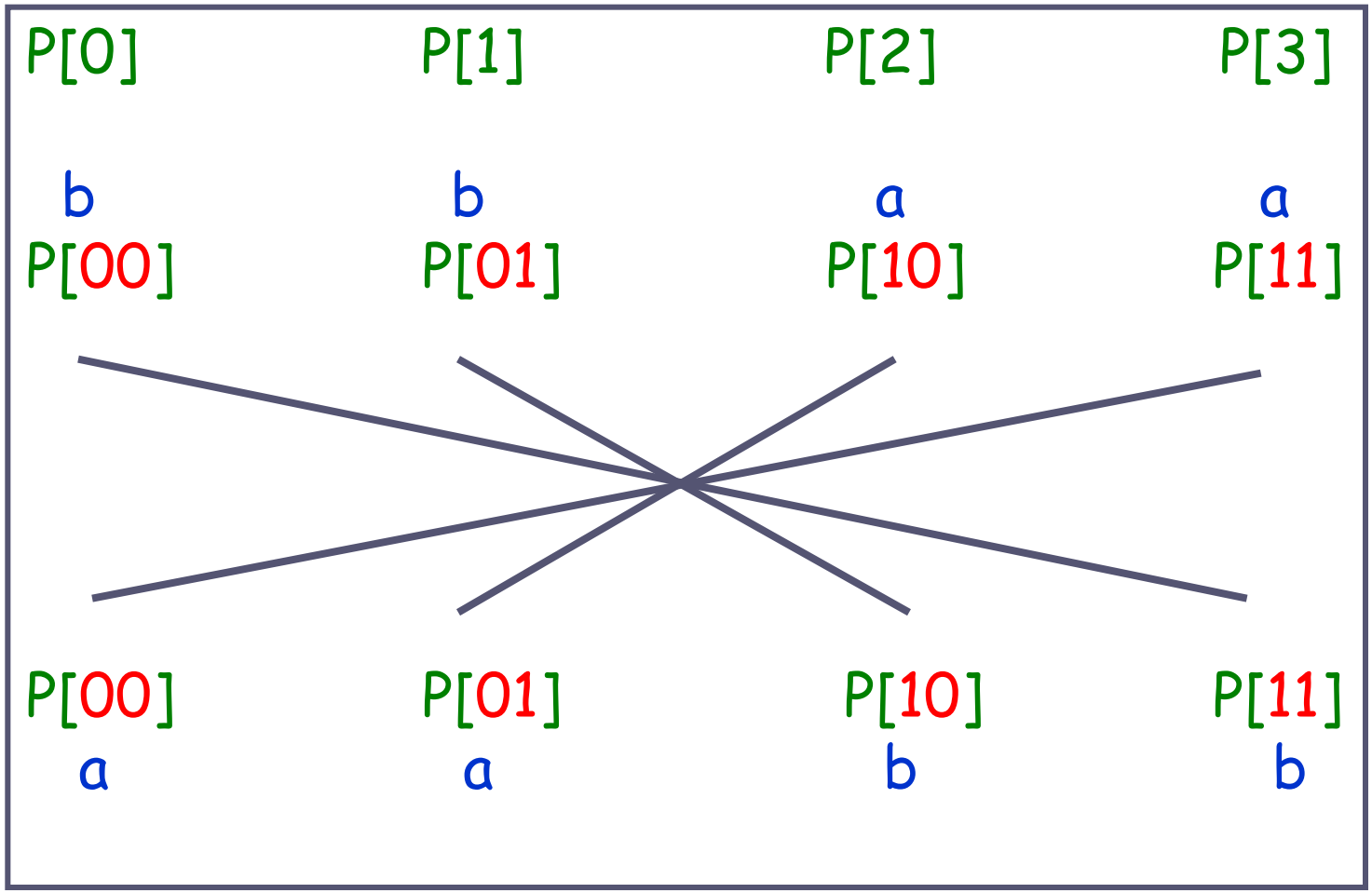
T[0]	T[1]	T[2]	T[3]
a	a	b	b

P[0]	P[1]	P[2]	P[3]
b	b	a	a

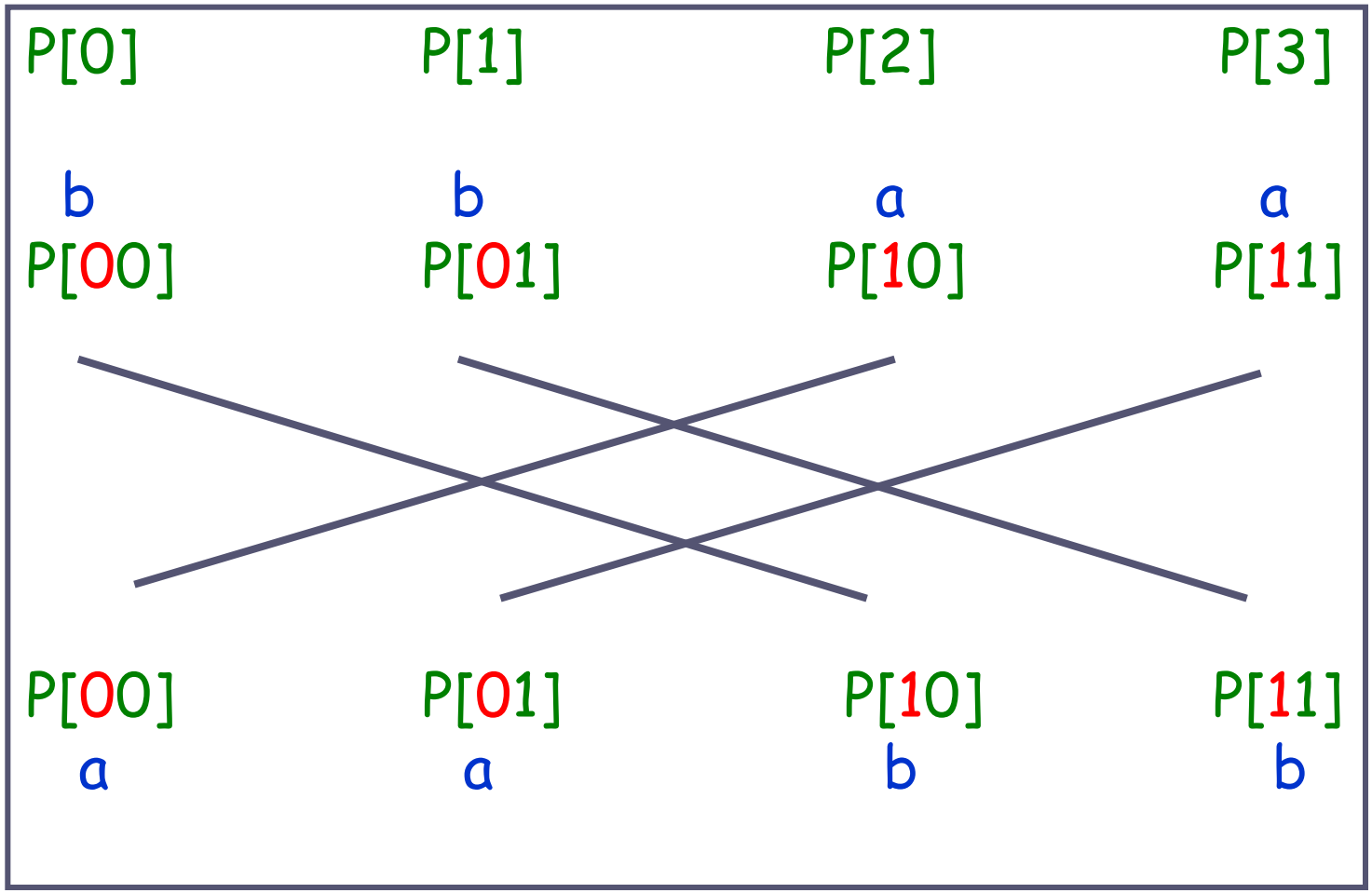
Example: BAD



Example: GOOD



Example: BEST



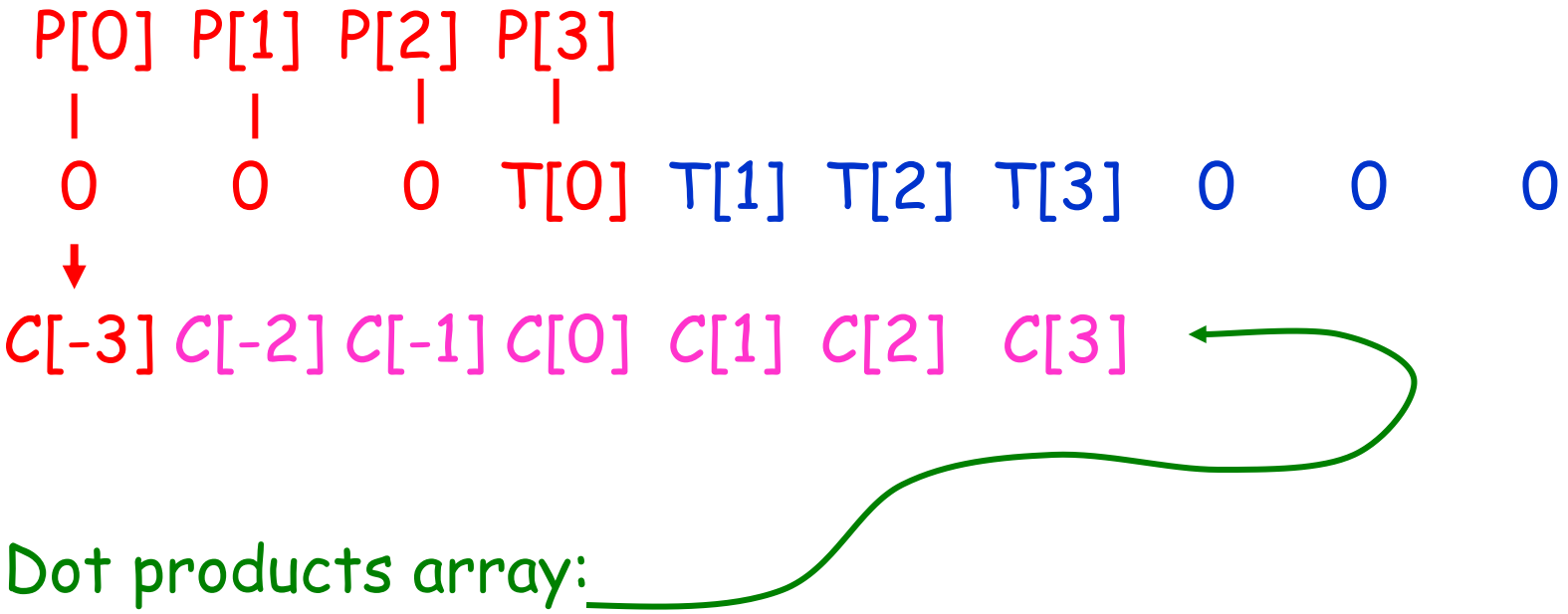
Naive *Algorithm*

For each of the $2^{\log m} = m$ different bit combinations try matching.

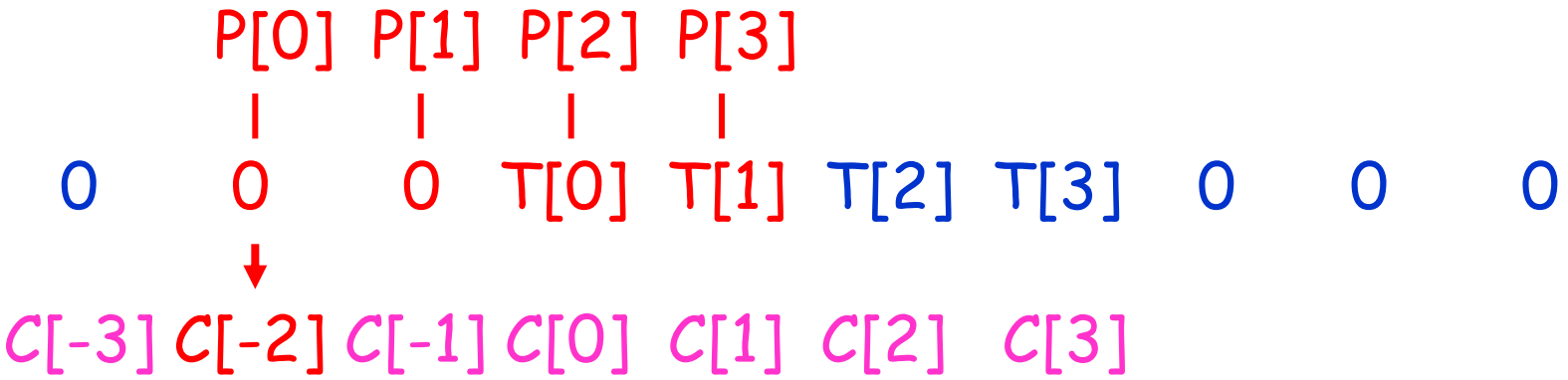
Choose match with minimum bits.

Time: $O(m^2)$.

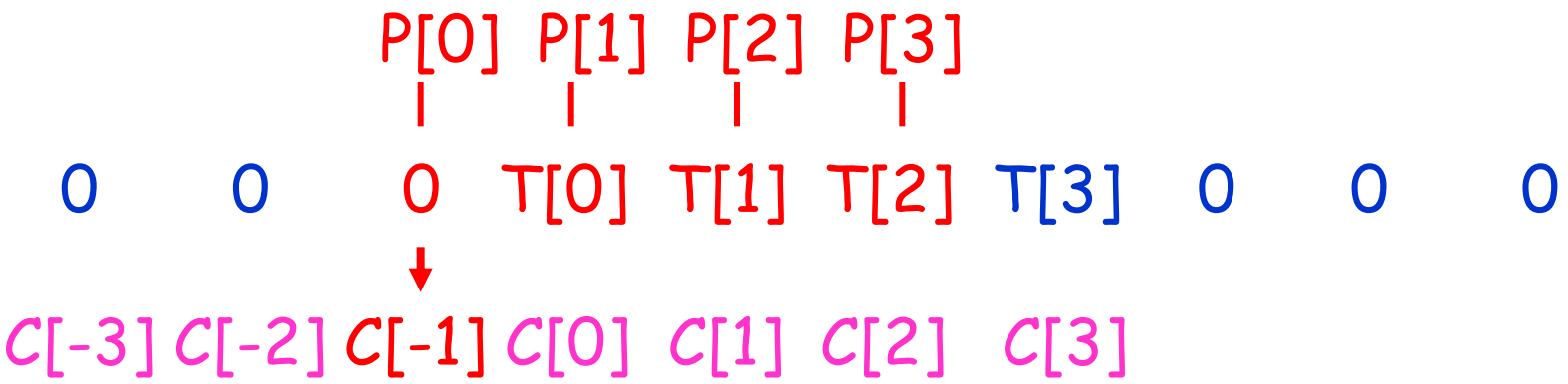
Polynomial multiplication - What Really Happened?



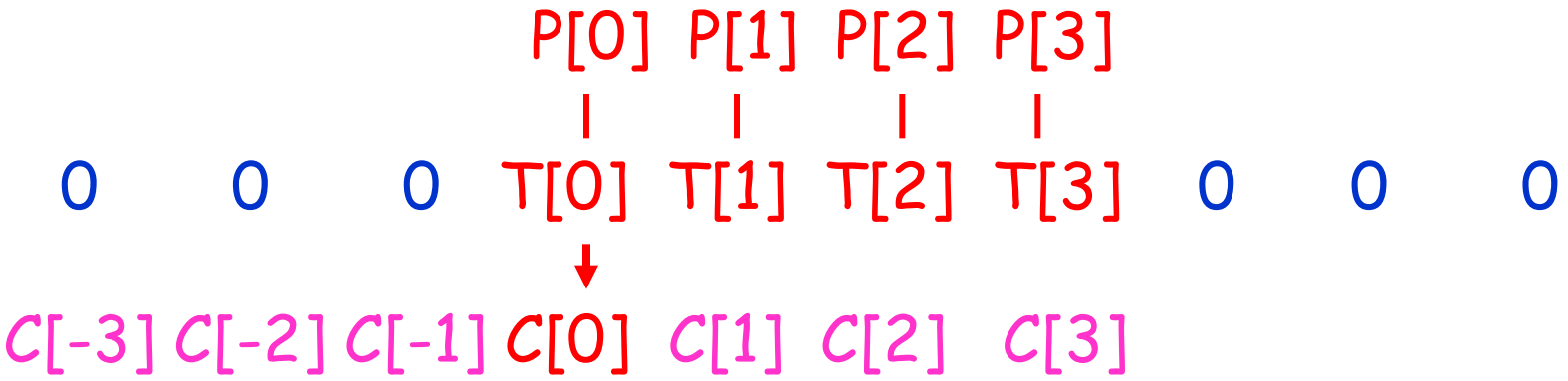
What Really Happened?



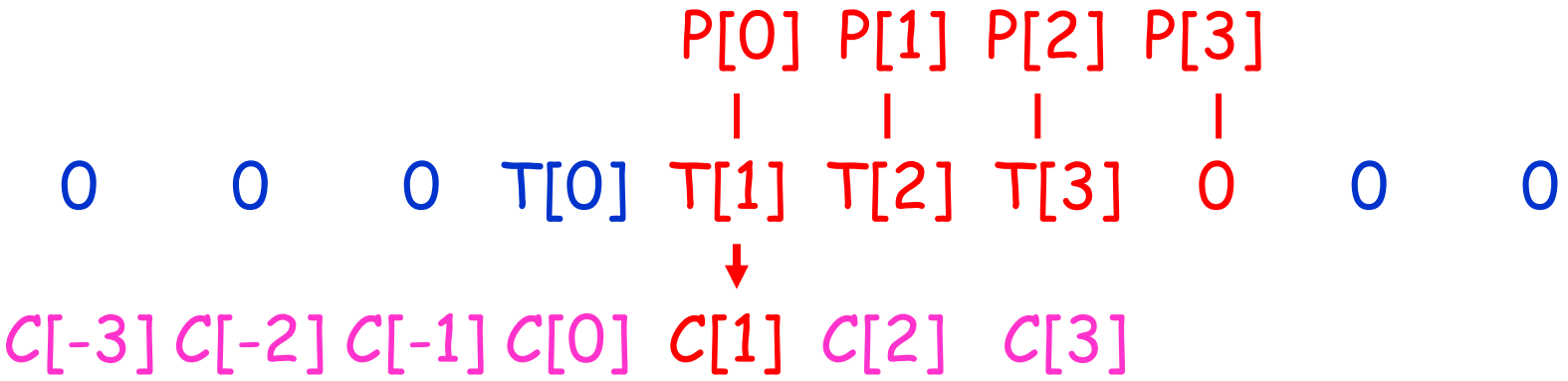
What Really Happened?



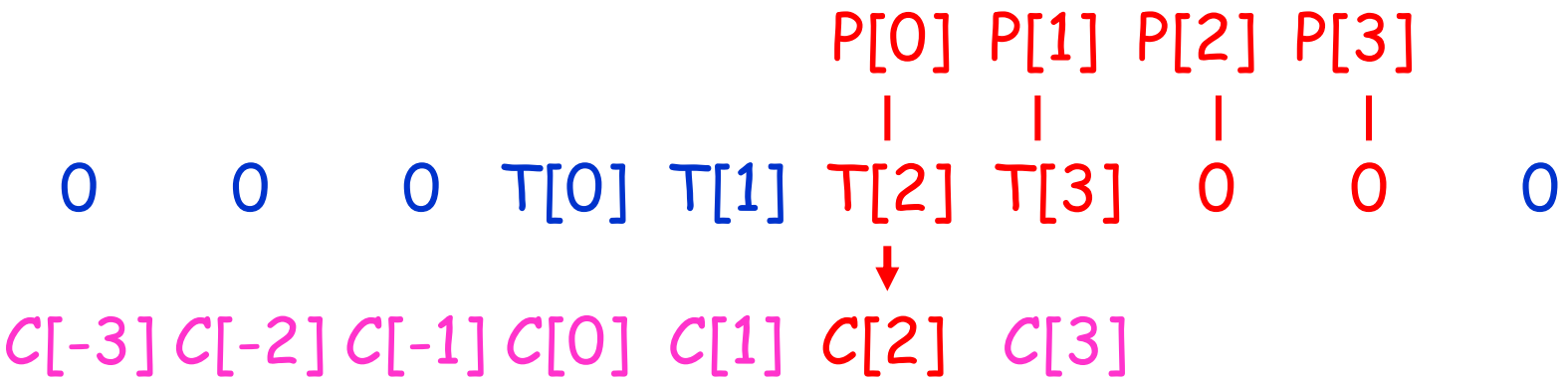
What Really Happened?



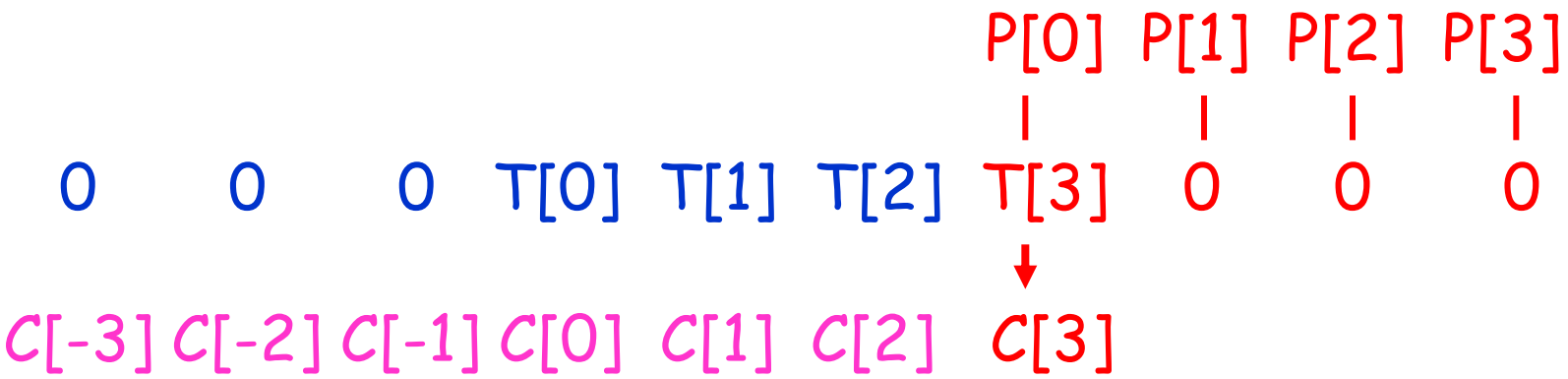
What Really Happened?



What Really Happened?



What Really Happened?



Another way of defining the convolution:

$$C(T,P)[j] = \sum_{i=0}^{m} T[i] \cdot P[i-j]; \quad j = -m,\ldots, m$$

Where we define: $P[x]=0$

for $x<0$ and $x>m$.

FFT solution to the "shift" convolution:

1. Compute $F^m(\overline{X}) = \overline{V}$ in time $O(m \log m)$ (values of X at roots of unity).

2. For polynomial multiplication $\overline{A} \times \overline{B}$ compute values of product polynomial at roots of unity $F^m(\overline{A}) \times F^m(\overline{B}) = \overline{V}$ in time $O(m \log m)$.

3. Compute the coefficient of the product polynomial, $(F^m)^{-1}(\overline{V})$ again in time $O(m \log m)$.

A General Convolution C_f

Bijections f_j ; $j=1,\ldots,O(m)$

$$f_j : \{0,\ldots,m\} \to \{0,\ldots,m\}$$

$$C_f(T,P)[j] = \sum_{i=0}^{m} T[i] \cdot P[f_j(i)]; \quad j = 1,\ldots,O(m)$$

Consistent bit flip as a Convolution

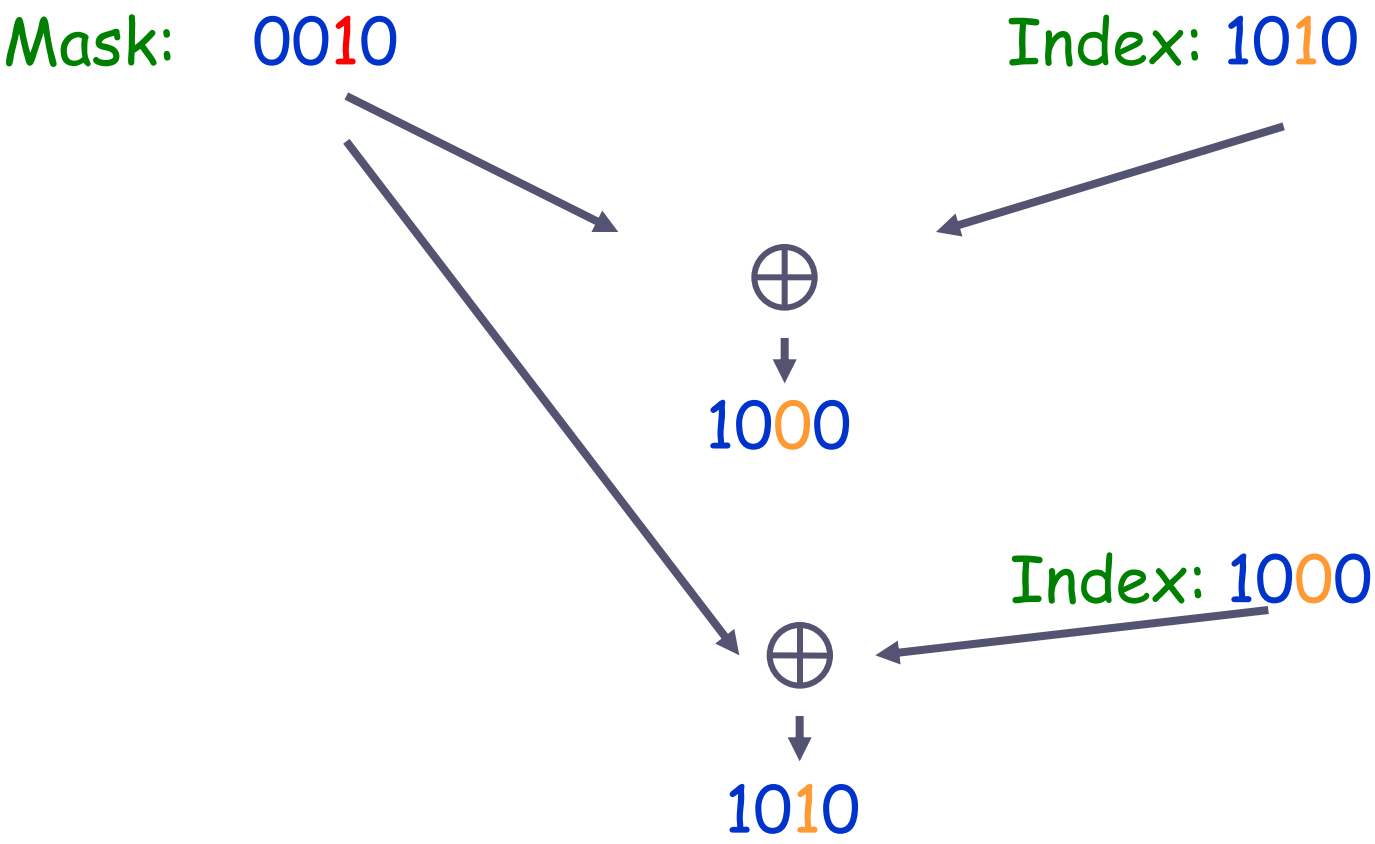
Construct a mask of length $\log m$ that has 0 in every bit except for the bad bits where it has a 1.

Example: Assume the bad bits are in indices $i,j,k \in \{0,\ldots,\log m\}$. Then the mask is

$$\begin{array}{c} \quad i \quad\quad j \quad\quad\quad k \\ 0000010000100001000 \end{array}$$

An exclusive OR between the mask and a pattern index Gives the target index.

Example:



Our Case:

Denote our convolution by: $T \otimes P$

Our convolution: For each of the $2^{\log m} = m$ masks, let $j \epsilon \{0,1\}^{\log m}$

$$T \otimes P[j] = \sum_{i=0}^{m} T[i] \cdot P[j \oplus i]$$

To compute min bit flip:

Let T,P be over alphabet $\{0,1\}$:
For each j, $P[j \oplus 0],\ldots, P[j \oplus m]$ is a permutation of P.

Thus, only the j's for which

$$T \otimes P[j] = \text{number of } 1\text{'s in } T$$

are valid flips.

Since for them all 1's match 1's and all 0's match 0's.

Choose valid j with minimum number of 1's.

Time

All convolutions can be computed in time $O(m^2)$
After preprocessing the permutation functions as tables.

Can we do better? (As in the FFT, for example)

Idea – Divide and Conquer- Walsh Transform

1. Split T and P to the length $m/2$ arrays:

$$T^+, T^-, P^+, P^-$$

2. Compute $T^+ \otimes P^+,\ T^- \otimes P^-$

3. Use their values to compute $T \otimes P$ in time $O(m)$.

Time: Recurrence: $t(m)=2t(m/2)+m$

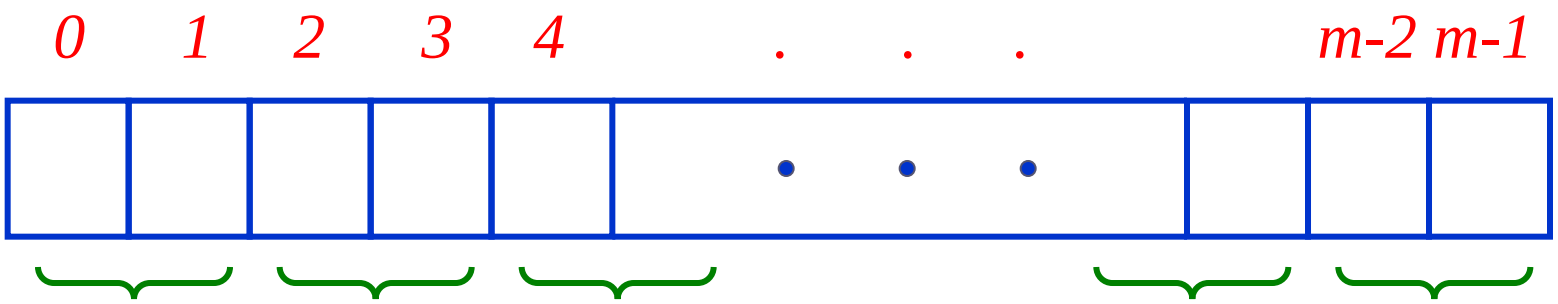
Closed Form: $t(m)=O(m \log m)$

Details

Constructing the Smaller Arrays V^+, V^-

Note: A mask $i \in \{0,1\}^{\log m}$ can also be viewed as a number $i=0,\ldots,m-1$. For $i \in \{0,1\}^{\log m-1}$:

$$V^+[i] = V[i0] + V[i1] \qquad V^-[i] = V[i0] - V[i1]$$



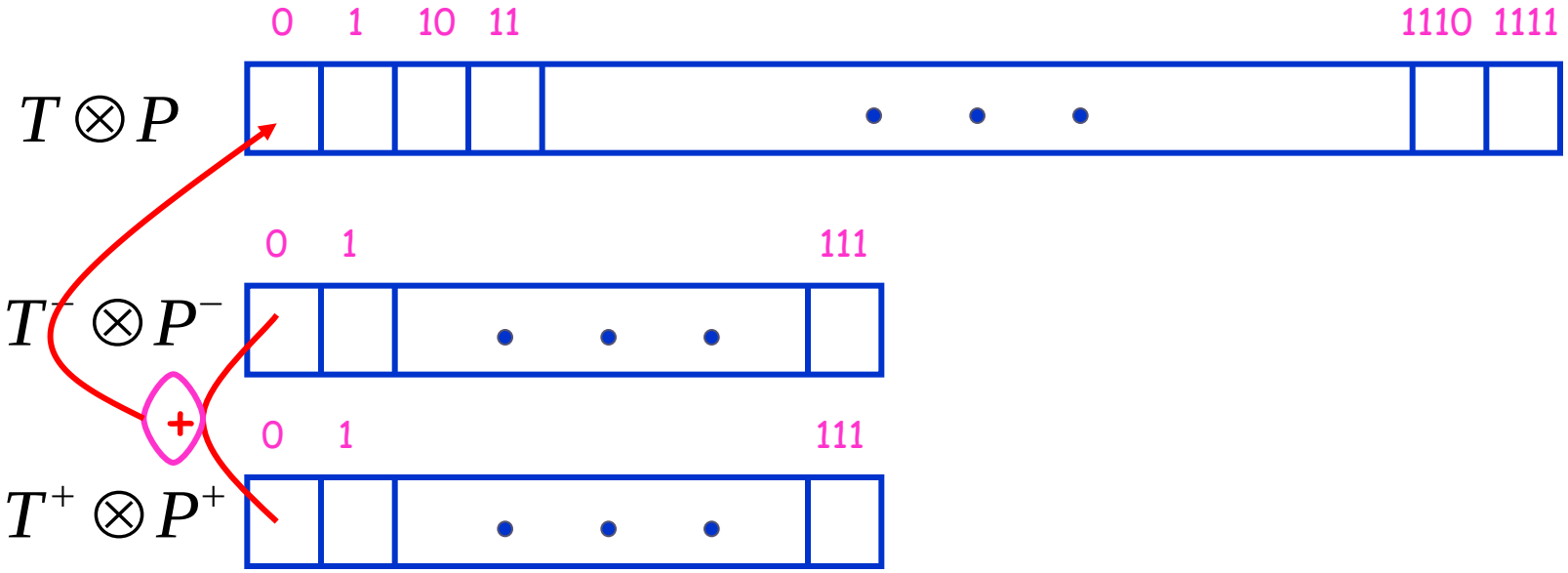
$V^+ = V[0]+V[1],\ V[2]+V[3],\ \ldots\ ,V[m-2]+V[m-1]$

$V^- = V[0]-V[1],\ V[2]-V[3],\ \ldots\ ,V[m-2]-V[m-1]$

Putting it Together

$$T \otimes P[i0] = \frac{T^+ \otimes P^+[i] + T^- \otimes P^-[i]}{2}$$

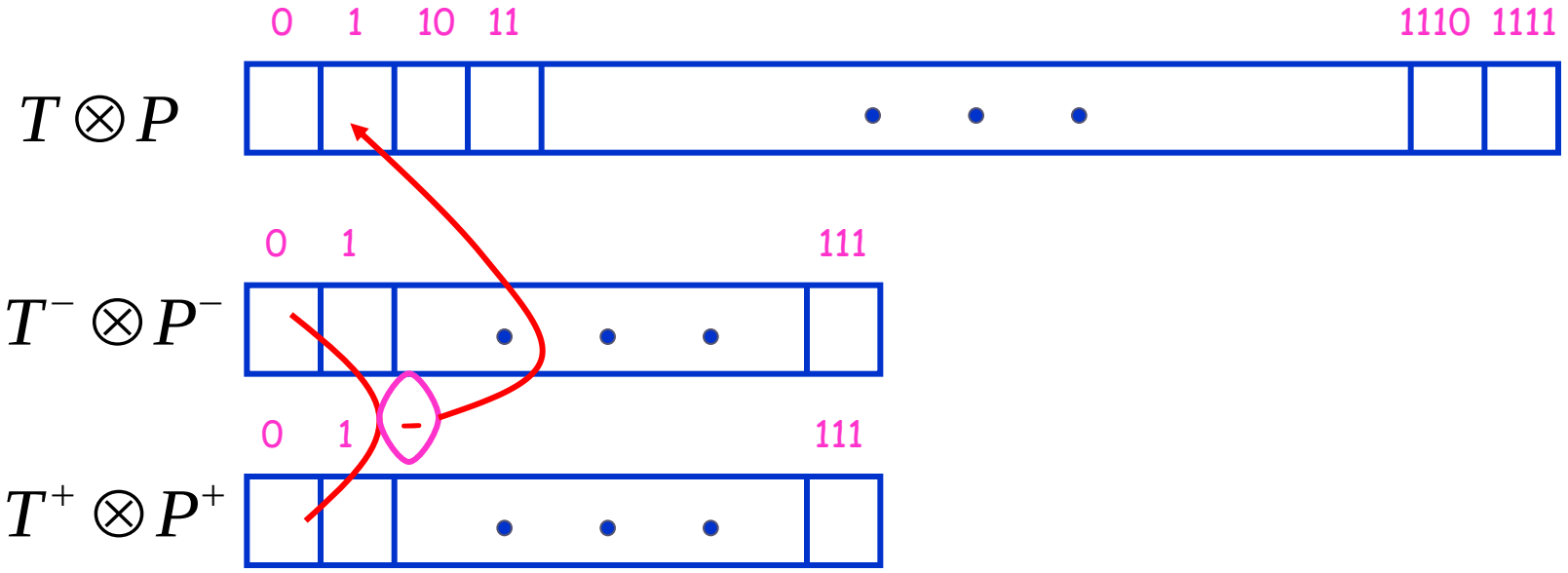
$$T \otimes P[i1] = \frac{T^+ \otimes P^+[i] - T^- \otimes P^-[i]}{2}$$



Putting it Together

$$T \otimes P[i0] = \frac{T^+ \otimes P^+[i] + T^- \otimes P^-[i]}{2}$$

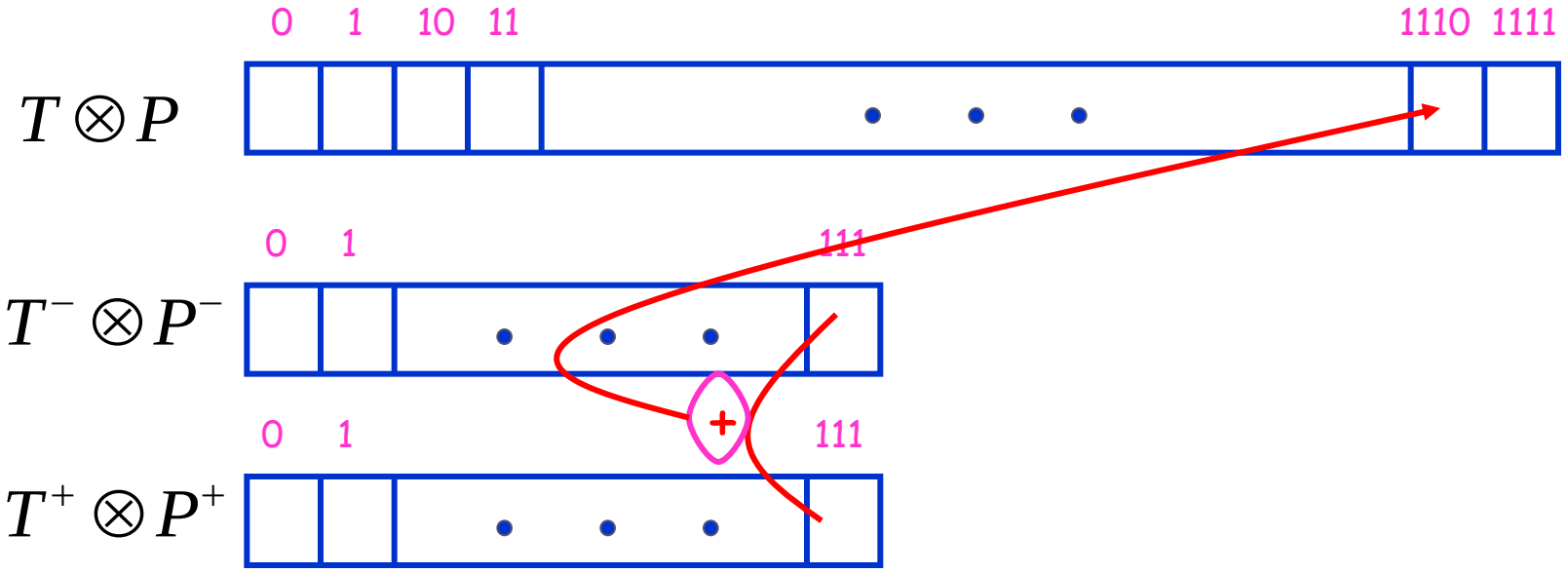
$$T \otimes P[i1] = \frac{T^+ \otimes P^+[i] - T^- \otimes P^-[i]}{2}$$



Putting it Together

$$T \otimes P[i0] = \frac{T^{+} \otimes P^{+}[i] + T^{-} \otimes P^{-}[i]}{2}$$

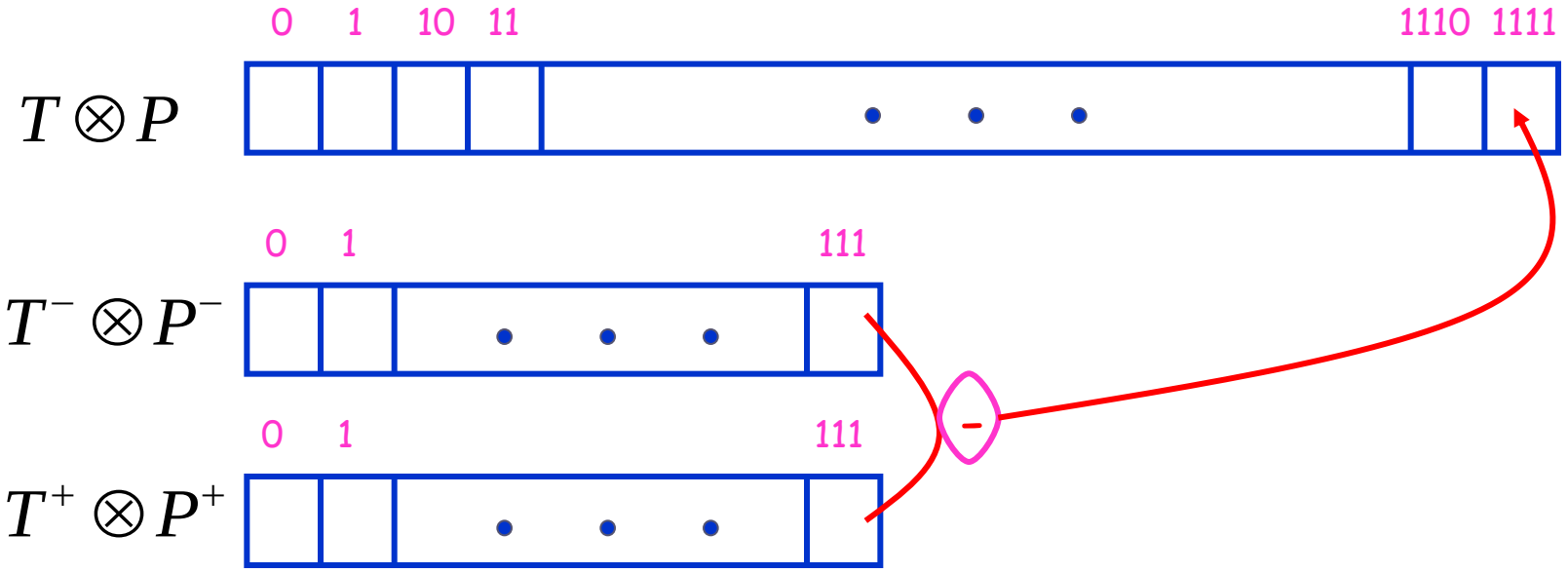
$$T \otimes P[i1] = \frac{T^{+} \otimes P^{+}[i] - T^{-} \otimes P^{-}[i]}{2}$$



Putting it Together

$$T \otimes P[i0] = \frac{T^+ \otimes P^+[i] + T^- \otimes P^-[i]}{2}$$

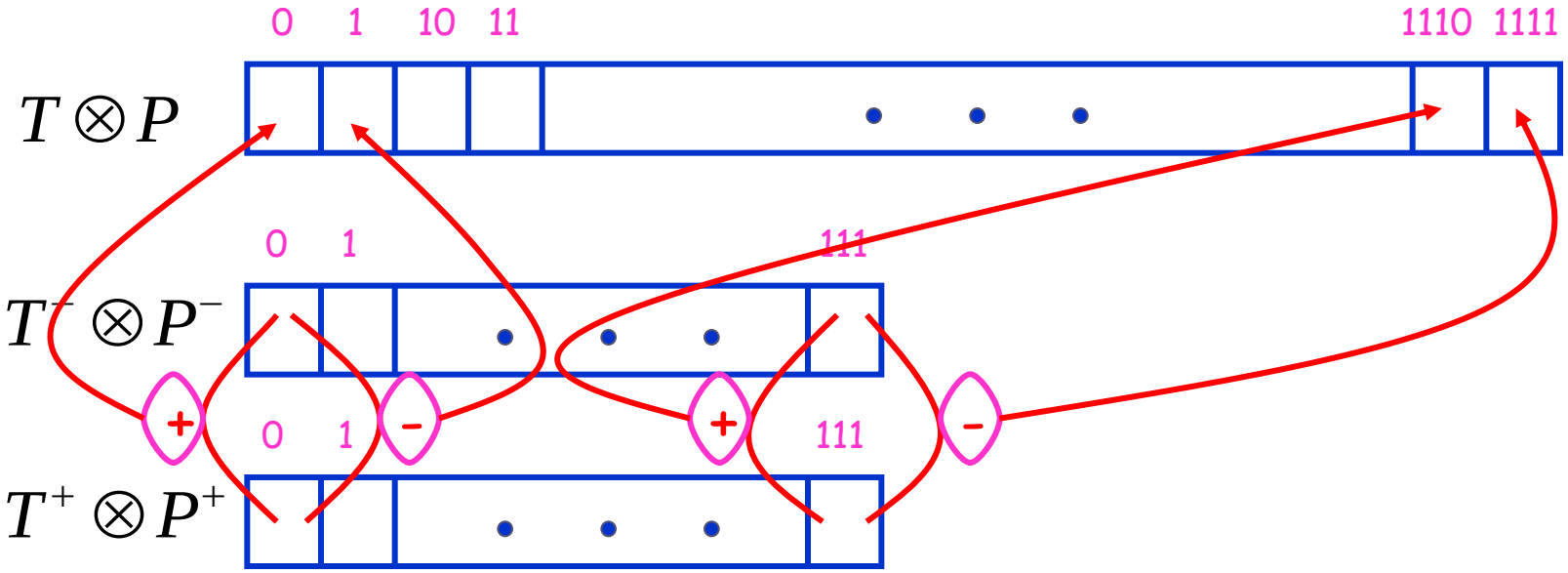
$$T \otimes P[i1] = \frac{T^+ \otimes P^+[i] - T^- \otimes P^-[i]}{2}$$



Putting it Together

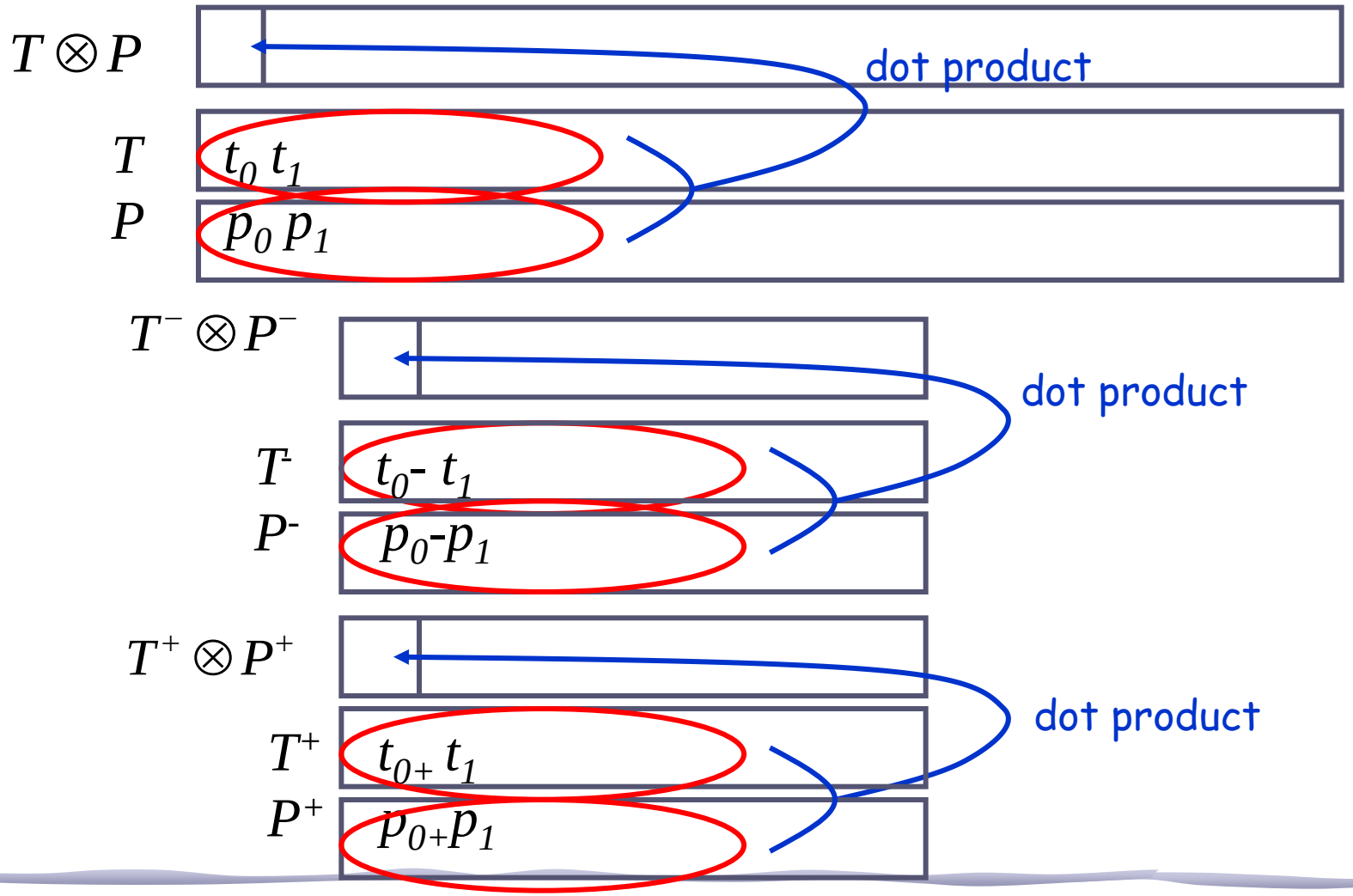
$$T \otimes P[i0] = \frac{T^{+} \otimes P^{+}[i] + T^{-} \otimes P^{-}[i]}{2}$$

$$T \otimes P[i1] = \frac{T^{+} \otimes P^{+}[i] - T^{-} \otimes P^{-}[i]}{2}$$

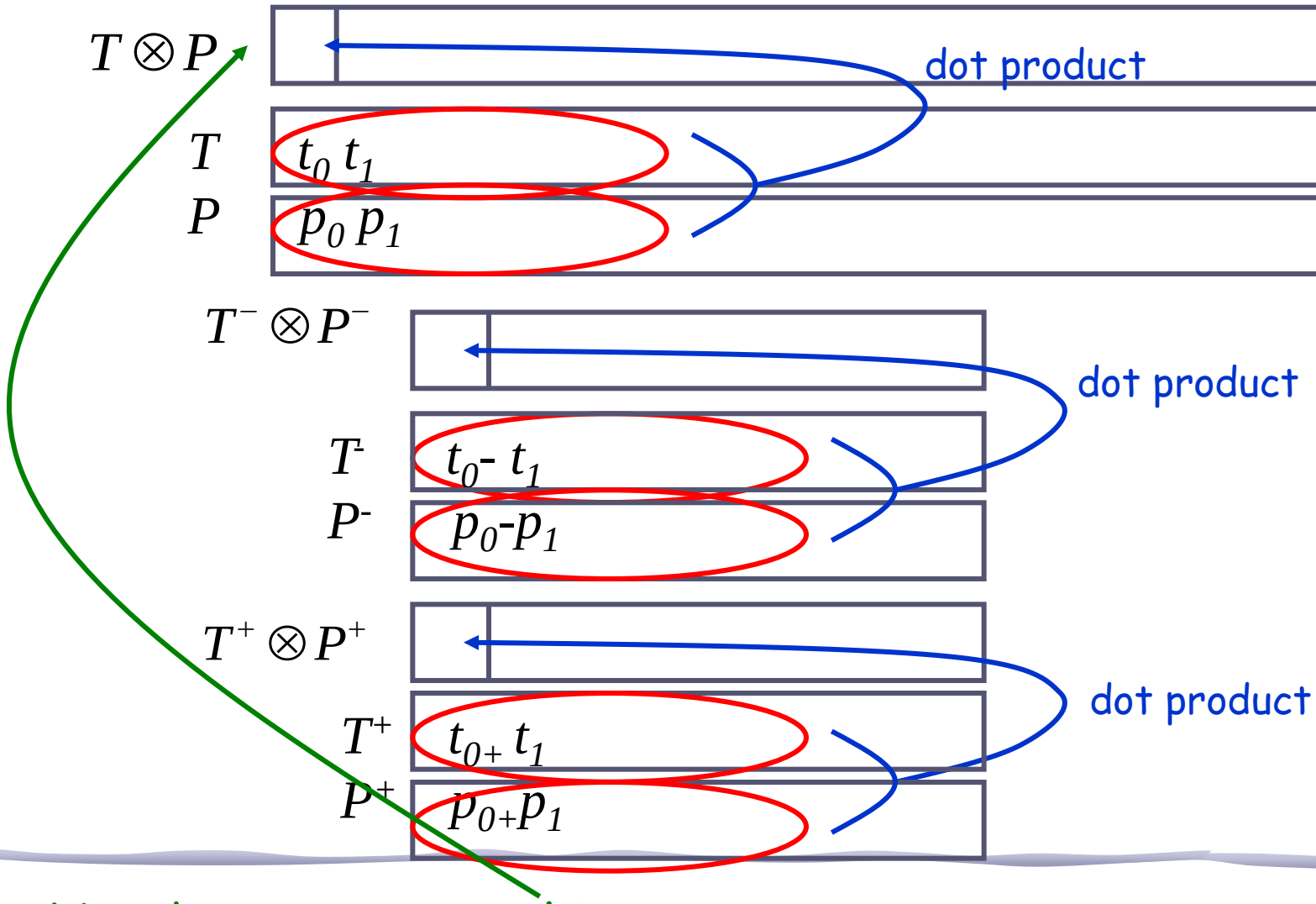


Why does it work ????

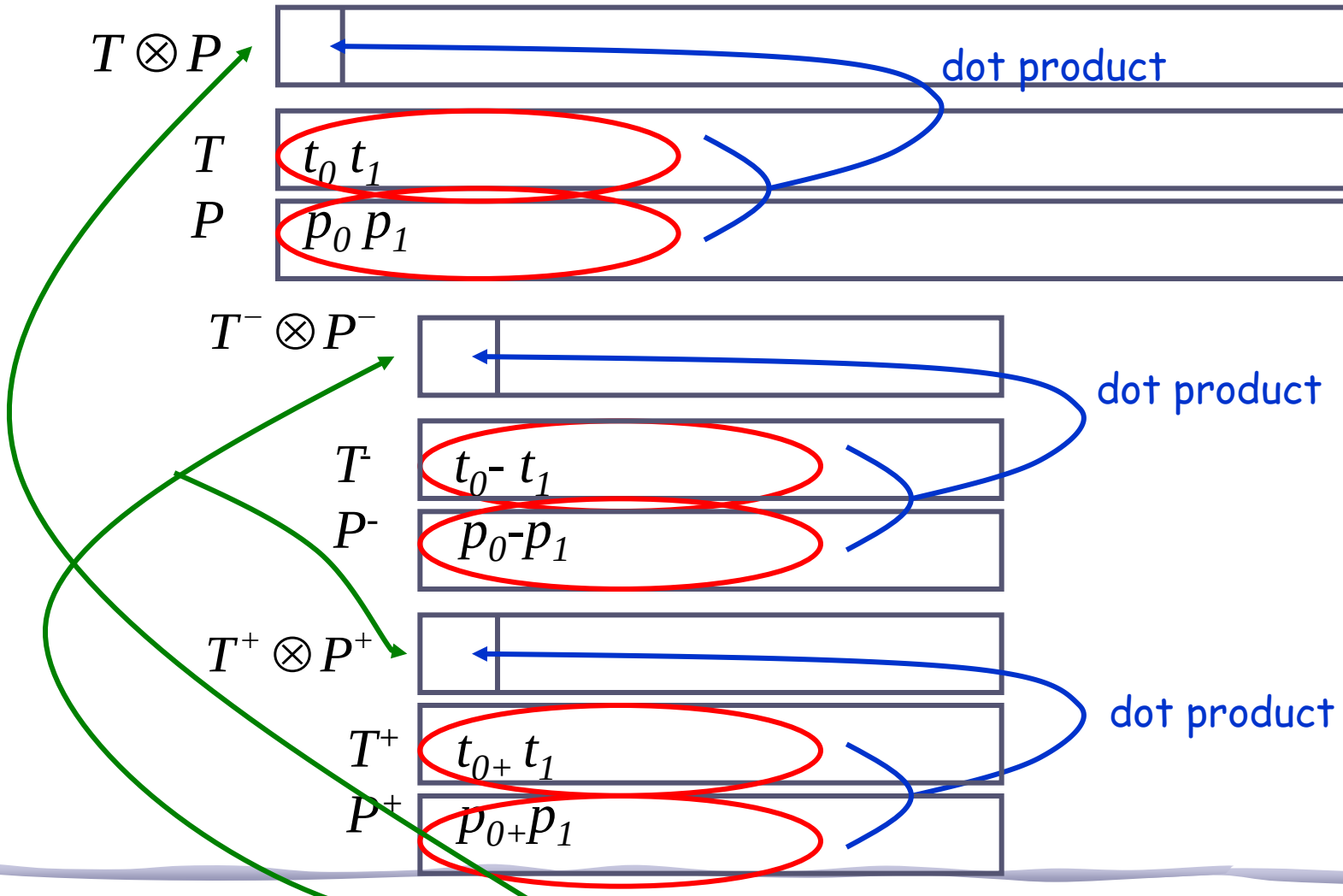
Consider the case of i=0



Consider the case of i=0



Consider the case of i=0



Need a way to get this from these…

Lemma:

T	a	c
P	b	d

To get the dot product: ab+cd

T^+	a+c
P^+	b+d

T^-	a-c
P^-	b-d

from: (a+c)(b+d) and (a-c)(b-d)

Add: (a+c)(b+d) = ab + cd + cb + ad

(a-c)(b-d) = ab + cd – cb – ad

Get: 2ab+2cd

Divide by 2: ab + cd

Because of distributivity it works for entire dot product.

If mask is 00001:

T	a	c
P	b	d

To get the dot product: ad+cb

T^+	a+c
P^+	b+d

from: (a+c)(b+d) and (a-c)(b-d)

T^-	a-c
P^-	b-d

Subtract: (a+c)(b+d) = ab + cd + cb + ad

(a-c)(b-d) = ab + cd – cb – ad

Get: 2cb+2ad

Divide by 2: cb + ad

Because of distributivity it works for entire dot product.

What happens when other bits are bad?

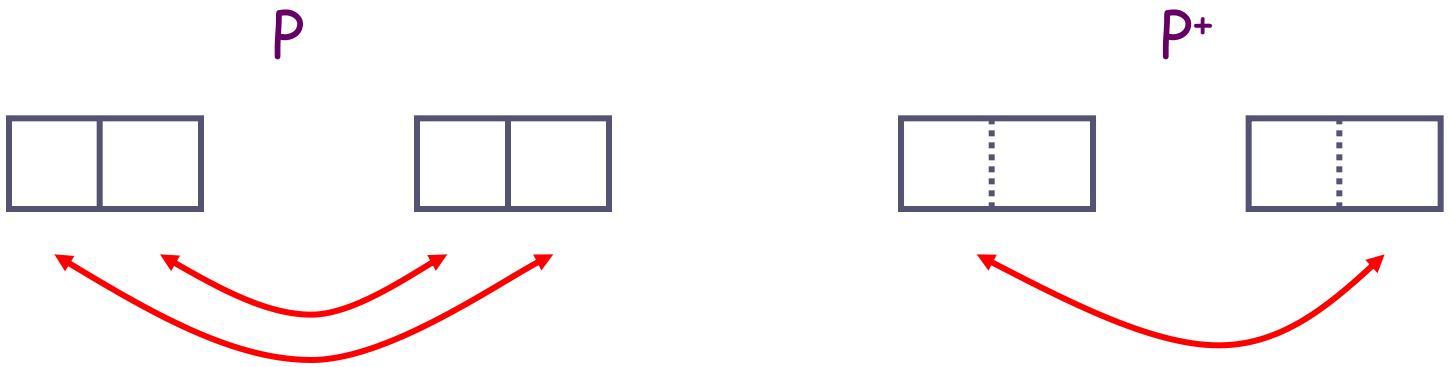
If LSB=0 , mask i0 on $T \otimes P$
is mask i on $T^+ \otimes P^+$ and $T^- \otimes P^-$
meaning, the "bad" bit is at half the index.

What it means is that appropriate pairs are multiplied , and single products are extracted from pairs as seen in the lemma.

If Least Significant Bit is 1

If LSB=1 , mask i1 on
is mask i on

meaning, the "bad" bit is at half the index. But there Is an additional flip within pairs.



What it means is that appropriate pairs are multiplied , and single products are extracted from pairs as seen in the lemma for the case of flip within pair.

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into $\log m$ strings:

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S =$ A_0 A_1 A_2 . . . A_m

$a_{00}\ a_{01}\ a_{02}\ \ldots$ $a_{10}\ a_{11}\ a_{12}\ \ldots$ $a_{20}\ a_{21}\ a_{22}\ \ldots$ $a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 =$ a_{00}

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots\quad a_{10}\ a_{11}\ a_{12}\ \ldots\quad a_{20}\ a_{21}\ a_{22}\ \ldots\qquad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\quad a_{10}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \quad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \qquad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}\ \ldots\ a_{m0}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

S = A_0 A_1 A_2 . . . A_m

a_{00} a_{01} a_{02} … a_{10} a_{11} a_{12} … a_{20} a_{21} a_{22} … a_{m0} a_{m1} a_{m2} …

S_0 = a_{00} a_{10} a_{20} . . . a_{m0}

S_1 = a_{01}

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \qquad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\quad a_{10}\quad a_{20}\quad \ldots\quad a_{m0}$

$S_1 = a_{01}\quad a_{11}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \qquad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}\ \ldots\ a_{m0}$

$S_1 = a_{01}\ a_{11}\ a_{21}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S =$	A_0	A_1	A_2	. . .	A_m
	$a_{00}\ a_{01}\ a_{02}\ \ldots$	$a_{10}\ a_{11}\ a_{12}\ \ldots$	$a_{20}\ a_{21}\ a_{22}\ \ldots$		$a_{m0}\ a_{m1}\ a_{m2}\ \ldots$
$S_0 =$	a_{00}	a_{10}	a_{20}	. . .	a_{m0}
$S_1 =$	a_{01}	a_{11}	a_{21}	. . .	a_{m1}

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \quad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}\ \ldots\ a_{m0}$

$S_1 = a_{01}\ a_{11}\ a_{21}\ \ldots\ a_{m1}$

. . .

$S_{\log m} = a_{0\ \log m}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots \quad a_{10}\ a_{11}\ a_{12}\ \ldots \quad a_{20}\ a_{21}\ a_{22}\ \ldots \quad a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}\ \ldots\ a_{m0}$

$S_1 = a_{01}\ a_{11}\ a_{21}\ \ldots\ a_{m1}$

. . .

$S_{\log m} = a_{0\ \log m}\ a_{1\ \log m}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

$S = A_0\ A_1\ A_2\ \ldots\ A_m$

$a_{00}\ a_{01}\ a_{02}\ \ldots\ \ a_{10}\ a_{11}\ a_{12}\ \ldots\ \ a_{20}\ a_{21}\ a_{22}\ \ldots\ \ a_{m0}\ a_{m1}\ a_{m2}\ \ldots$

$S_0 = a_{00}\ a_{10}\ a_{20}\ \ldots\ a_{m0}$

$S_1 = a_{01}\ a_{11}\ a_{21}\ \ldots\ a_{m1}$

. . .

$S_{\log m} = a_{0\ \log m}\ a_{1\ \log m}\ a_{2\ \log m}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

S =	A_0	A_1	A_2	. . .	A_m
	$a_{00}\ a_{01}\ a_{02}\ \ldots$	$a_{10}\ a_{11}\ a_{12}\ \ldots$	$a_{20}\ a_{21}\ a_{22}\ \ldots$		$a_{m0}\ a_{m1}\ a_{m2}\ \ldots$
S_0 =	a_{00}	a_{10}	a_{20}	. . .	a_{m0}
S_1 =	a_{01}	a_{11}	a_{21}	. . .	a_{m1}
. . .					
$S_{\log m}$ =	$a_{0\ \log m}$	$a_{1\ \log m}$	$a_{2\ \log m}$	. . .	$a_{m\ \log m}$

General Alphabets

1. Sort all symbols in T and P.

2. Encode $\{0,\ldots,m\}$ in binary, i.e. $\log m$ bits per symbol.

3. Split into log m strings:

S =	A_0	A_1	A_2	. . .	A_m
	$a_{00}\ a_{01}\ a_{02}\ \ldots$	$a_{10}\ a_{11}\ a_{12}\ \ldots$	$a_{20}\ a_{21}\ a_{22}\ \ldots$		$a_{m0}\ a_{m1}\ a_{m2}\ \ldots$
S_0 =	a_{00}	a_{10}	a_{20}	. . .	a_{m0}
S_1 =	a_{01}	a_{11}	a_{21}	. . .	a_{m1}
. . .					
$S_{\log m}$ =	$a_{0\ \log m}$	$a_{1\ \log m}$	$a_{2\ \log m}$	. . .	$a_{m\ \log m}$

General Alphabets

4. For each S_i: Write list of masks that achieves minimum flips.

5. Merge lists and look for masks that appear in all.

Time: $O(m \log m)$ per bit.
$O(m \log^2 m)$ total.

Other Models

1. Minimum "bad" bits (occasionally flip).
2. Minimum transient error bits?
3. Consistent flip in string matching model?
4. Consistent "stuck" bit?
5. Transient "stuck" bit?

Note: The techniques employed in asynchronous pattern matching have so far proven different from traditional pattern matching.

Results

- $|T|=|P|=m$, flipped bits problem : $O(m \log m)$.

FFT over Z_2

- $|T|=|P|=m$, faulty bits problem: deterministically $O(|S|m^{\log 3})$, randomly $O(m \log m)$.

Formal polynomials

- $|T|=|P|=m$, faulty bits problem: deterministically *approximated* to $c>1$ $O(|S|m^{\log 3}/\log^{c-1} m)$.

Probabilistic proof

- $|T|=n$, $|P|=m=2^k$, faulty bits problem: deterministically $O(|S|nm \log m)$.

The problem we have seen

- Interchange Rearrangement Problem:

  INPUT: input string S and target string T.

  GOAL: Rearrange S to T by interchanges.

Example:

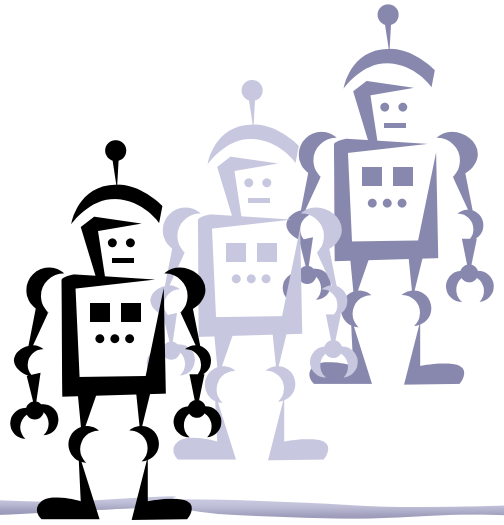
S=abacb → bbaca → bbaac=T

T=bbaac

- Cost of rearrangement.

The problem:

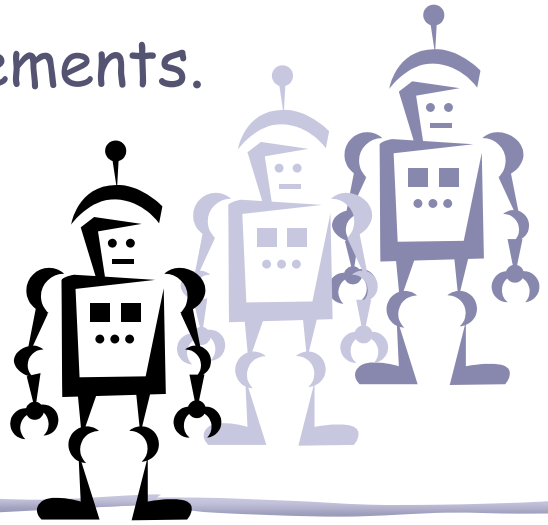
Find distance (=minimum cost).



History

- The interchange rearrangement problem is classical and well-studied.
- The mathematician Cayley studied this problem back in 1849.
- Focused on permutation strings case: strings with no repetitions of elements.
- General strings case is an

open question since 1849!



Our Challenge

- Study the interchange rearrangement problem for general strings (possibly repeating symbols).

Solve the open problem of Cayley

- Generalize the study under various length-weighted cost models.

  Recently, interest in such cost models, e.g. [Bender et al., SODA04].

Our results

- $\mathcal{NP}$-**hard** for general strings in unit cost model, BUT
- **polynomial time** comput./approx. in various other cost models.

A summary of results for L^{α}-interchange distance problem

α value	Binary Strings	Permutations	General strings
$\alpha=0$	$O(m)$	$O(m)$	$\mathcal{NP}$-hard $O(m)$ 1.5-approx.
$0<\alpha\leq 1/\log m$	$O(m^3)$	$O(m)$ 2-approx.	$O(m)$ 3-approx.
$1/\log m<\alpha<1$	$O(m^3)$	$O(m)$ 2-approx.	$O(m^3)$ $\lvert\Sigma\rvert$-approx.
$\alpha=1$	$O(m)$	$O(m)$	$O(m)$
$1<\alpha\leq\log 3$	$O(m)$	$O(m)$ 2-approx.	$O(m)$ 2-approx.
$\alpha>\log 3$	$O(m)$	$O(m)$	$O(m)$

D-type

I-type

Thank You