Fast Fourier transform

Prof. Richard Vuduc

Georgia Institute of Technology

CSE/CS 8803 PNA, Spring 2008

[L.11] Tuesday, February 12, 2008

Sources for today's material

- CS 267 (Yelick & Demmel, UCB)
- *Applied Numerical Linear Algebra*, by Demmel
- Xiaoye (Sherry) Li (LBNL)
- Van Loan (Cornell)
  - "The ubiquitous Kronecker product"
  - *Computational frameworks for the FFT*
- Heath (UIUC)

Review: Sparse direct solvers

Anatomy of a sparse direct solver

$$P_r \cdot A \cdot P_c^T = LU$$

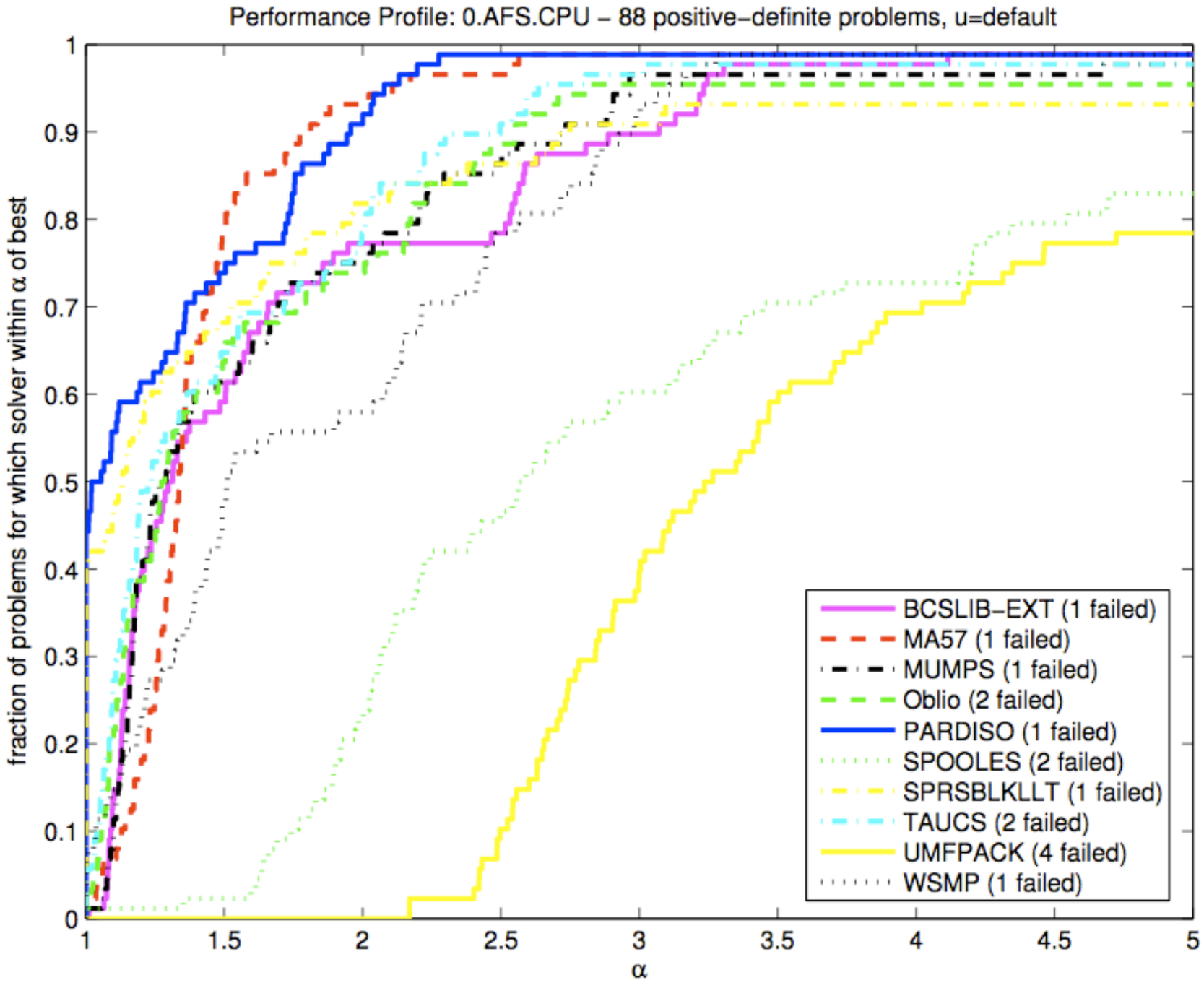
- Order equations & variables to minimize fill in L, U [Combinatorial]
- Symbolic factorization [Allocate memory for factorization]
- Numerical factorization [Dominates run-time]
- Triangular solves [Small fraction, unless many RHS]

Sparse LU software

See also: http://www.netlib.org/utk/people/JackDongarra/la-sw.html

- LU
  - SuperLU [Xiaoye “Sherry” Li @ LBNL]
  - MUMPS [Amestoy, *et al*., INPT-ENSEEIHT-IRIT, France]
  - UMFPACK [Davis @ U. Florida]
- Cholesky
  - PSPASES / WSMP [Gupta @ IBM]
  - TAUCS [Toledo @ Tel Aviv U.]
  - Oblio [Dobrian, formerly @ Columbia / ODU]
  - Also: MUMPS, UMFPACK

$p(\alpha)$



Source: Gould, Yu, Scott (2005); http://www.numerical.rl.ac.uk/reports/reports.shtml

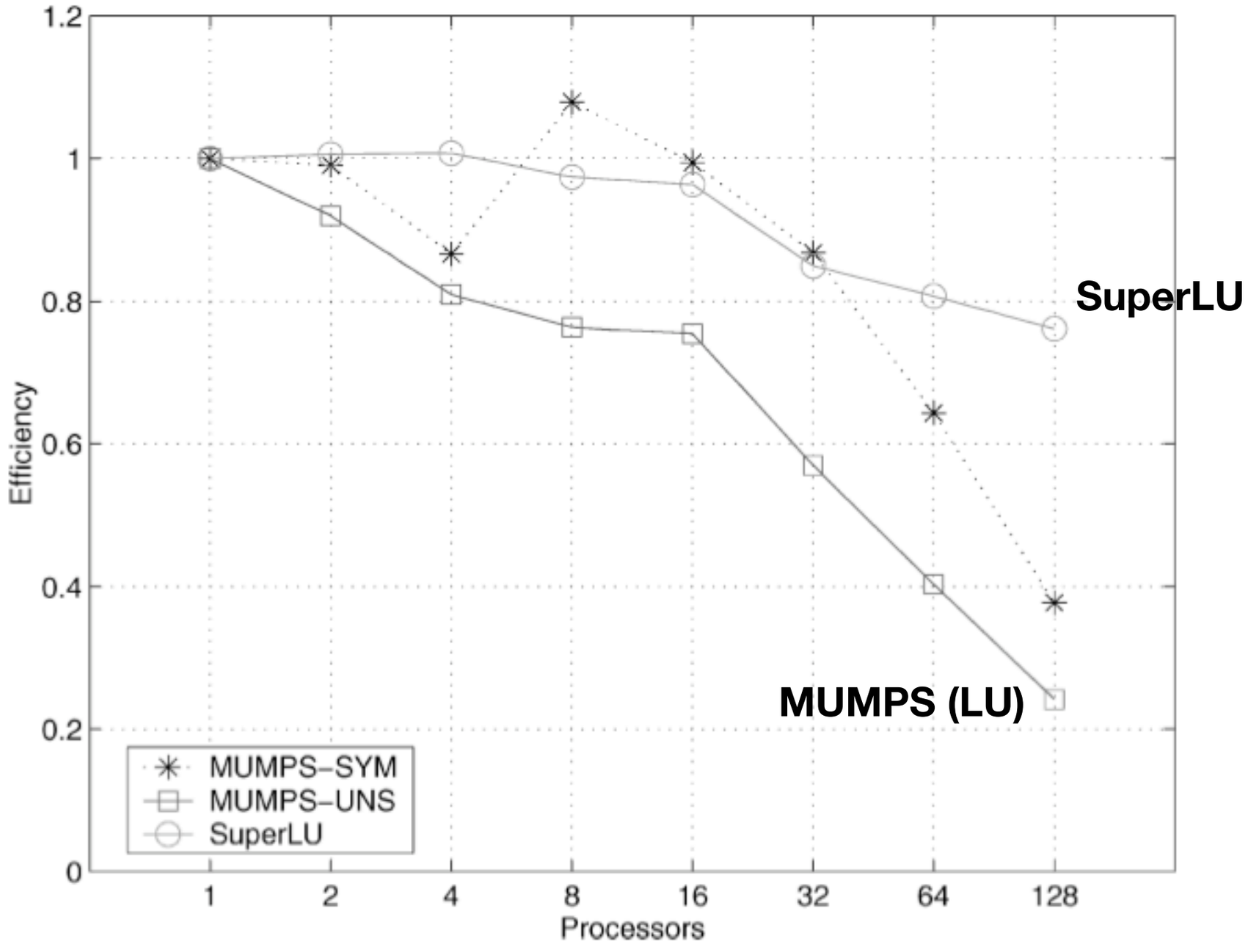


Fig. 8. Parallel efficiency (cubic grids, nested dissection).

Source: Amestoy, Duff, L'Excellent, & Li (2001). http://mumps.enseeiht.fr/doc.html

Table XII. Solve Time (in Seconds) for Large Matrices on the CRAY T3E. "+IR" Shows the Time Spent Improving the Initial Solution Using Iterative Refinement

Matrix	Ordering	Solver	Number of processors								
			1	4	8	16	32	64	128	256	512
BBMAT	AMD	MUMPS	—	0.52	0.37	0.29	0.28	0.28	0.31	0.33	0.38
		SuperLU	—	1.33	1.16	0.75	0.71	0.53	0.51	0.50	0.44
		SuperLU+IR	—	1.80	1.48	1.01	0.93	0.72	0.68	0.62	0.57
	ND	MUMPS	—	0.37	0.35	0.24	0.26	0.26	0.28	0.30	0.32
		SuperLU	—	1.99	1.60	1.07	0.93	0.76	0.65	0.59	0.44
		SuperLU+IR	—	2.43	1.95	1.34	1.16	0.96	0.86	0.78	0.62
ECL32	AMD	MUMPS	—	0.64	0.46	0.37	0.38	0.36	0.42	0.46	0.53
		SuperLU	—	1.72	1.60	1.09	1.13	0.75	0.79	0.66	0.56
	ND	MUMPS	—	0.47	0.32	0.28	0.26	0.24	0.28	0.31	0.36
		SuperLU	—	1.52	1.57	1.02	0.72	0.68	0.68	0.56	0.49
INVEXTR1	ND	MUMPS	0.57	0.30	0.18	0.14	0.14	0.13	0.16	0.18	0.21
		MUMPS+IR	1.50	0.16	0.11	0.27	0.26	0.24	0.23	0.31	0.39
		SuperLU	1.48	0.80	0.77	0.53	0.50	0.38	0.29	0.26	0.21
		SuperLU+IR	2.75	1.24	1.07	0.78	0.71	0.57	0.44	0.42	0.35
MIXTANK	ND	MUMPS	0.67	0.26	0.18	0.14	0.14	0.13	0.16	0.18	0.21
		SuperLU	1.47	0.73	0.68	0.45	0.43	0.31	0.23	0.21	0.17
TWOTONE	MC64	MUMPS	—	1.03	0.81	0.84	0.86	0.85	0.92	0.96	1.05
	+AMD	SuperLU	—	3.49	3.88	2.69	2.61	1.58	1.23	1.03	0.86
		SuperLU+IR	—	6.66	5.65	7.44	3.42	2.73	1.59	1.41	1.17

"—" Indicates not enough memory.

Source: Amestoy, Duff, L'Excellent, & Li (2001). http://mumps.enseeiht.fr/doc.html

Table XIV. Memory Used during Factorization (in Megabytes, Per Processor)

Matrix	Ordering	Solver	Number of processors					
			4		16		64	
			Avg.	Max.	Avg.	Max.	Avg.	Max.
BBMAT	AMD	MUMPS	147	176	52	65	32	40
		SuperLU	113	114	50	51	33	34
	ND	MUMPS	114	118	44	53	28	35
		SuperLU	124	128	60	61	43	44
ECL32	AMD	MUMPS	190	212	55	64	32	41
		SuperLU	113	115	42	44	24	25
	ND	MUMPS	132	139	39	44	25	28
		SuperLU	79	81	33	34	21	22
FIDAPM11	AMD	MUMPS	65	67	25	30	16	19
		SuperLU	38	39	16	16	10	10
INVEXTR1	ND	MUMPS	65	85	23	28	17	22
		SuperLU	47	48	22	22	15	16
LHR71C	MC64 +AMD	MUMPS	54	48	22	25	16	20
		SuperLU	49	51	27	29	21	21
MIXTANK	ND	MUMPS	84	87	29	31	19	21
		SuperLU	55	56	23	23	14	15
RMA10	AMD	MUMPS	39	42	17	25	11	21
		SuperLU	32	33	15	16	10	11
TWOTONE	MC64 +AMD	MUMPS	167	180	57	67	42	60
		SuperLU	66	80	35	41	24	24
WANG4	AMD	MUMPS	69	82	22	23	15	20
		SuperLU	33	34	14	14	8	9

Source: Amestoy, Duff, L'Excellent, & Li (2001). http://mumps.enseeiht.fr/doc.html

Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

Algorithm	Serial	PRAM	Memory	# procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2 ($N^{7/3}$)	N	$N^{3/2}$ ($N^{5/3}$)	N ($N^{4/3}$)
Jacobi	N^2 ($N^{5/3}$)	N ($N^{2/3}$)	N	N
Explicit inverse	N^2	$\log N$	N^2	N^2
Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	$N \log N$ ($N^{4/3}$)	N
Conj. grad.	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2(1/3)} \log N$	N	N
RB SOR	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2}$ ($N^{1/3}$)	N	N
FFT	**$N \log N$**	**$\log N$**	**N**	**N**
Multigrid	N	$\log^2 N$	N	N
Lower bound	**N**	**$\log N$**	**N**	

PRAM = idealized parallel model with zero communication cost.

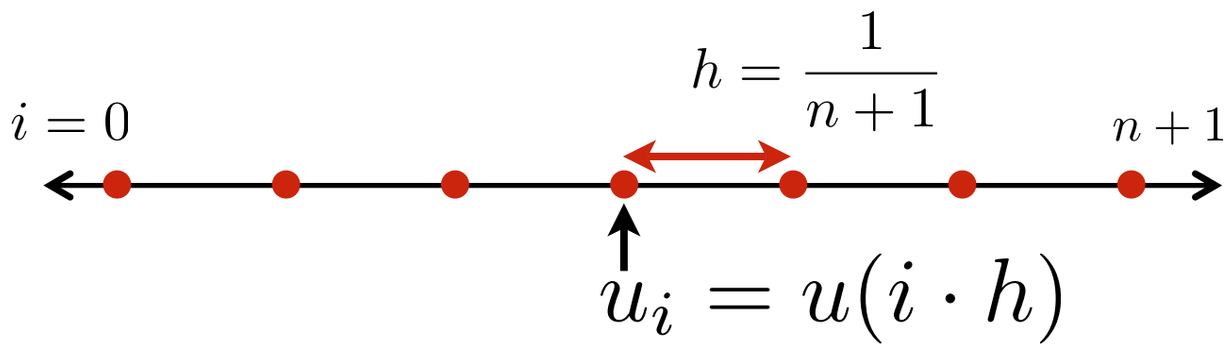
Source: Demmel (1997)

Review: Poisson's equation

Discretizing 1-D Poisson

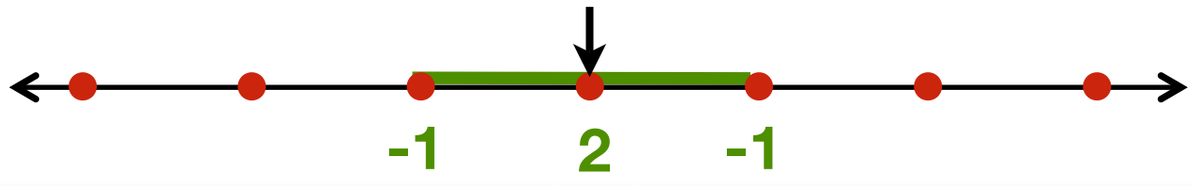
$$-\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1, \quad u(0) = u(1) = 0$$

Discretize:



Approximate: $$-\frac{d^2u(x)}{dx^2}\Big|_{x=x_i} \approx \frac{2u_i - u_{i-1} - u_{i+1}}{h^2}$$

"Stencil":



Express stencil in matrix notation

Approximation:

$$\approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} \quad = \quad f_i \triangleq f(ih)$$

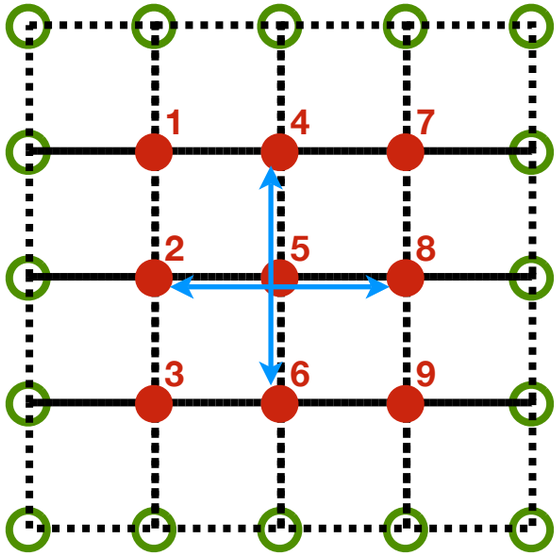
$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \cdots & & \\ & & & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} \quad = \quad h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix}$$

$$\Downarrow$$

$$T_n \cdot u \quad = \quad h^2 f$$

Discretizing 2-D Poisson

Graph and stencil



$$
\left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \\ 4 \\ 5 \\ 6 \\ \\ 7 \\ 8 \\ 9 \end{array}
$$

Serial time complexity for RB SOR, CG, and sparse LU

In 2-D: $O(n^3)$

Naïve (dense) : $O(n^6)$

Optimal : $O(n^2)$

Exploiting structure to obtain fast algorithms for 2-D Poisson

- **Dense LU**: Assume no structure ⇒ O(n^6)
- **Sparse LU**: Sparsity ⇒ O(n^3), need extra memory
- **CG**: Symmetric positive definite ⇒ O(n^3), a little extra memory
- **RB SOR**: Fixed sparsity pattern ⇒ O(n^3), no extra memory
- Today — **FFT**: Exploit numerical structure ⇒ **O(n^2 log n)**

Numerical properties of the Poisson stencil

Express stencil in matrix notation

Approximation:

$$\approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} \quad = \quad f_i \triangleq f(ih)$$

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \cdots & & \\ & & & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} \quad = \quad h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix}$$

$$\Downarrow$$

$$T_n \cdot u \quad = \quad h^2 f$$

1-D Poisson: Eigendecomposition of T_n

- *Lemma*: Eigenvalues and eigenvectors of T_n are

$$T_n z_j = \lambda_j z_j \quad \Longrightarrow \quad T_n = Z \Lambda Z^T$$

$$Z Z^T = I$$

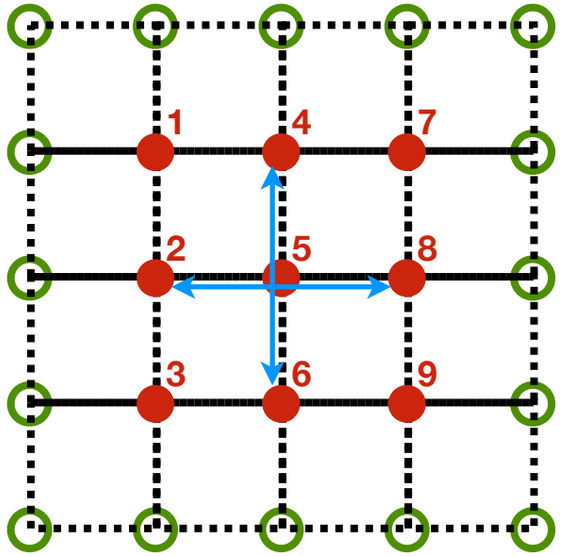
$$\lambda_j = 2\left(1 - \cos\frac{\pi j}{n+1}\right)$$

$$z_j(k) = \sqrt{\frac{2}{n+1}} \sin\frac{\pi j k}{n+1}$$

- *Exercise*: **Verify**. Hint: $\sin a + \sin b = 2 \sin\frac{a+b}{2} \cos\frac{a-b}{2}$

Eigendecomposition for discretized 2-D Poisson stencil matrix?

Graph and stencil



$$T_{n\times n} = \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array}\right)$$

2-D Poisson

$$
\begin{aligned}
f(x,y) &= -\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial^2 u(x,y)}{\partial y^2} \\
&\Downarrow \\
h^2 f_{ij} &= 4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}
\end{aligned}
$$

2-D Poisson

$$
\begin{aligned}
f(x,y) &= -\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial^2 u(x,y)}{\partial y^2} \\
&\Downarrow \\
h^2 f_{ij} &= 4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} \\
&\Downarrow \\
F &= (f_{ij}) \\
U &= (u_{ij}) \\
h^2 F &= T_n \cdot U + U \cdot T_n
\end{aligned}
$$

2-D Poisson: Eigendecomposition

$$
\begin{aligned}
h^2 F &= T_n \cdot U + U \cdot T_n \\
&\Downarrow \\
X &\equiv z_i z_j^T \\
T_n \cdot X + X \cdot T_n &= T_n \cdot \left(z_i z_j^T\right) + \left(z_i z_j^T\right) \cdot T_n \\
&= \lambda_i z_i z_j^T + \left(z_i z_j^T\right) \lambda_j \\
&= (\lambda_i + \lambda_j) \cdot X \\
&\Downarrow \\
\therefore \quad z_i z_j^T &= \text{“eigenvector”} \\
\lambda_i + \lambda_j &= \text{eigenvalue}
\end{aligned}
$$

Relating T_n and T_{nxn}: "vec(X)" and Kronecker products

- Let ***vec(X)*** = stacks columns of matrix *X* into a single vector

$$X \equiv \begin{pmatrix} x_1 \cdots x_n \end{pmatrix}$$

$$\mathrm{vec}(X) \equiv \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- **Kronecker product**

$$A \otimes B \equiv \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{pmatrix}$$

Facts about "vec(*X*)" and "⊗"

$$
\begin{aligned}
\mathrm{vec}(A+B) &= \mathrm{vec}(A)+\mathrm{vec}(B) \\
\mathrm{vec}(A\cdot X) &= (I\otimes A)\cdot \mathrm{vec}(X) \\
\mathrm{vec}(X\cdot B) &= (B\otimes I)\cdot \mathrm{vec}(X)
\end{aligned}
$$

$$
\begin{aligned}
A\otimes(B\otimes C) &= (A\otimes B)\otimes C \\
(A\cdot C)\otimes(B\cdot D) &= (A\otimes B)\cdot(C\otimes D) \\
(A\otimes B)^T &= A^T\otimes B^T \\
(A\otimes B)^{-1} &= A^{-1}\otimes B^{-1}
\end{aligned}
$$

Note: In MATLAB, `kron(A,B)` computes *A*⊗*B*

Relating T_n and T_{nxn}

$$
\begin{aligned}
h^2 F &= T_n \cdot U + U \cdot T_n \\
&\Downarrow \\
h^2 \mathrm{vec}(F) &= ((I \otimes T_n) + (T_n \otimes I)) \cdot \mathrm{vec}(U) \\
&\Downarrow \\
T_{n \times n} &\equiv (I \otimes T_n) + (T_n \otimes I) \\
&= (Z \otimes Z) \cdot (I \otimes \Lambda + \Lambda \otimes I) \cdot (Z \otimes Z)^T
\end{aligned}
$$

Extending to higher dimensions

$$
\begin{aligned}
T_n &= Z\Lambda Z^T \\
&\Downarrow \\
T_{n\times n} &= (I \otimes T_n) + (T_n \otimes I) \\
&= (Z \otimes Z) \cdot (I \otimes \Lambda + \Lambda \otimes I) \\
&\Downarrow \\
T_{n\times n\times n} &= (I \otimes I \otimes T_n) + (I \otimes T_n \otimes I) + (T_n \otimes I \otimes I) \\
&= (Z \otimes Z \otimes Z) \cdot (I \otimes I \otimes \Lambda + ...) \cdot (Z \otimes Z \otimes Z)^T
\end{aligned}
$$

Using the eigendecomposition to solve 1-D Poisson

$$
\begin{aligned}
u &= T_n^{-1} \cdot h^2 f \\
&= (Z \Lambda Z^T)^{-1} h^2 f \\
&= h^2 Z \Lambda^{-1} Z^T f
\end{aligned}
$$

Need a fast $Z^T * f$, $Z * x$

Using the eigendecomposition to solve 2-D Poisson

$$\mathrm{vec}(U) = h^2 \cdot (Z \otimes Z)(I \otimes \Lambda + \Lambda \otimes I)^{-1}(Z \otimes Z)^T \cdot \mathrm{vec}(F)$$

$$\Downarrow$$

1. $F' = Z^T F Z$

2. $u'_{jk} = \dfrac{h^2 f'_{jk}}{\lambda_j + \lambda_k} \qquad \forall j, k$

3. $U = Z U' Z^T$

Need a fast $Z^T * f$, $Z * x$

Administrivia

Two joint classes with CS 8803 SC

- **Tues 2/19**: Floating-point issues in parallel computing by me
- **Tues 2/26**: GPGPUs by Prof. Hyesoon Kim
- **Both classes meet in Klaus 1116E**

Administrative stuff

- Front-end login node: **ccil.cc.gatech.edu** (CoC Unix account)
  - We “own” **warp43—warp56**
  - Some docs (**MPI**): http://www-static.cc.gatech.edu/projects/ihpcl/mpi.html
  - **Sign-up** for mailing list: https://mailman.cc.gatech.edu/mailman/listinfo/ihpc-lab

Homework 1: Parallel conjugate gradients

- Implement a parallel solver for Ax = b (serial C version provided)
  - Evaluate on three matrices: 27-pt stencil, and two application matrices
  - "Simplified:" No preconditioning
  - **Bonus**: Reorder, precondition
- Performance models to understand scalability of your implementation
  - Make measurements
  - Build predictive models
- Collaboration encouraged: Compare programming models or platforms

Fast Fourier transform

Discrete Fourier transform (DFT)

- Recall "Z"

$$z_{jk} = \sqrt{\frac{2}{n+1}} \sin \frac{\pi(j+1)(k+1)}{n+1}$$

Note:
j, k 0-based

- Multiplying by "Z" is almost the (m-point) DFT:

$$\begin{aligned} y &= \Phi^{(m)} x \\ \Phi^{(m)} &\equiv \left(\omega^{jk}\right)_{0 \le j,k < m} \\ \omega &= e^{\frac{-2\pi i}{m}} = \cos \frac{2\pi}{m} - i \cdot \sin \frac{2\pi}{m} \qquad i = \sqrt{-1} \end{aligned}$$

DFT and its inverse

$$
\begin{aligned}
\Phi &\equiv \left(\omega^{jk}\right)_{0\le j,k<m} \\
y &= \Phi \cdot x \\
&\Downarrow \\
y_j &= \sum_{k=0}^{m-1} \omega^{jk} x_k \\
&\Downarrow \\
\Phi^{-1} &= \frac{1}{m}\Phi^* \\
x_k &= \frac{1}{m}\sum_{j=0}^{m-1} \omega^{-jk} y_j
\end{aligned}
$$

DFT as polynomial evaluation

- Assume m = power of 2

$$
\begin{aligned}
y_j &= \sum_{k=0}^{m-1} \omega^{jk} x_k = \sum_{k=0}^{m-1} x_k \cdot \left(\omega^j\right)^k \\
&\equiv X\left(\omega^j\right)
\end{aligned}
$$

$$
\begin{aligned}
X(w) &\equiv \sum_{k=0}^{m-1} x_k \cdot w^k \\
&= x_0 + x_2 w^2 + x_4 w^4 + \cdots && \Leftarrow \textbf{Even terms} \\
&\quad + w \cdot \left[x_1 + x_3 w^2 + x_5 w^4 + \cdots\right] && \Leftarrow \textbf{Odd terms} \\
&= X_{\text{even}}(w^2) + w \cdot X_{\text{odd}}(w^2)
\end{aligned}
$$

A fast algorithm

$$y_j = \left(\Phi^{(m)} x\right)_j \quad = \quad X_{\text{even}}(\omega^{2j}) + \omega^j \cdot X_{\text{odd}}(\omega^{2j})$$

- X_{even} & X_{odd} have degree $m/2-1$ [assume $m = 2^s$]
- Evaluate at $(\varpi^j)^2$ for $0 <= j <= m-1$
  - Actually just $m/2$ points, since:

$$\begin{aligned}\left(\omega^{j+\frac{m}{2}}\right)^2 &= \left(\omega^j \cdot \omega^{\frac{m}{2}}\right) = \omega^{2j} \cdot \omega^m \\ &= \left(\omega^j\right)^2, \quad \text{for } 0 \le j < \frac{m}{2}\end{aligned}$$

  - FFT on m points reduced to 2 FFTs on $m/2$ points ⇒ **Divide and conquer**

Fast Fourier transform algorithm

$\mathrm{FFT}(x, \omega, m)$

if $m == 1$ **then**

return x_0

else

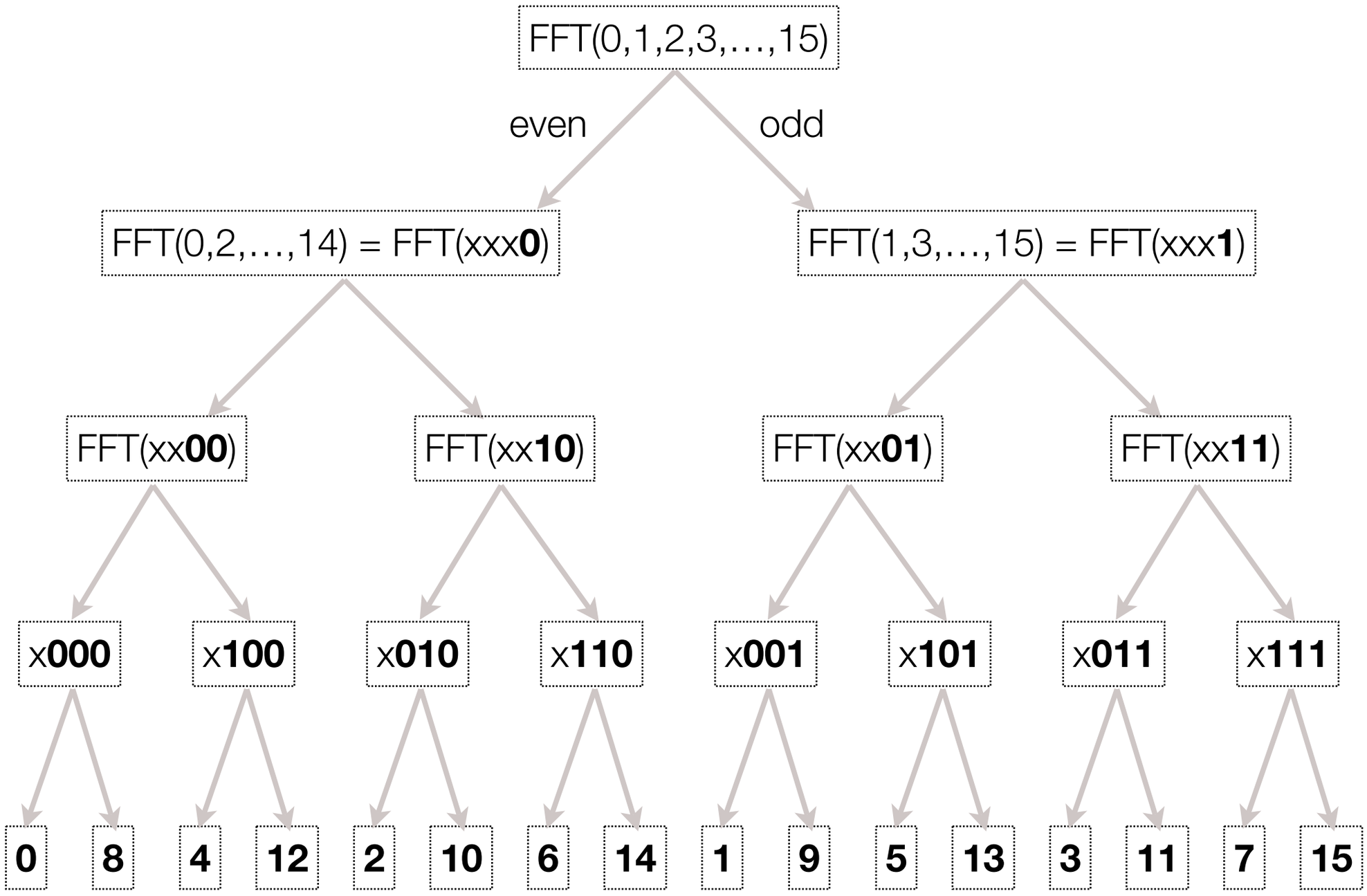
$x_{\text{even}} \leftarrow \mathrm{FFT}(x_{\text{even}}, \omega^2, \frac{m}{2})$

$x_{\text{odd}} \leftarrow \mathrm{FFT}(x_{\text{odd}}, \omega^2, \frac{m}{2})$

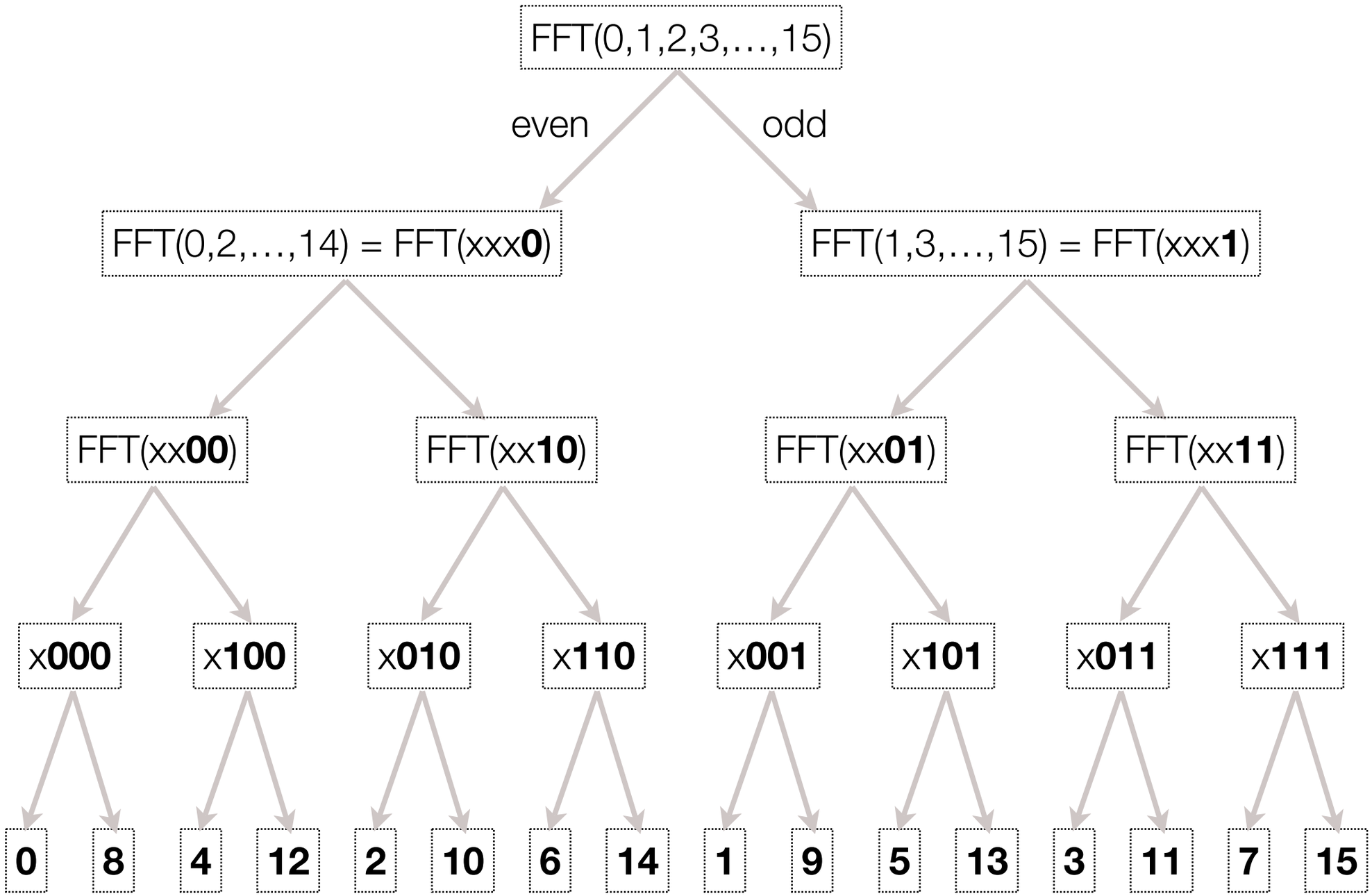
$w \leftarrow \left[w^0, w^1, \ldots, \omega^{\frac{m}{2}-1}\right]$ $\Longleftarrow$ precomputed

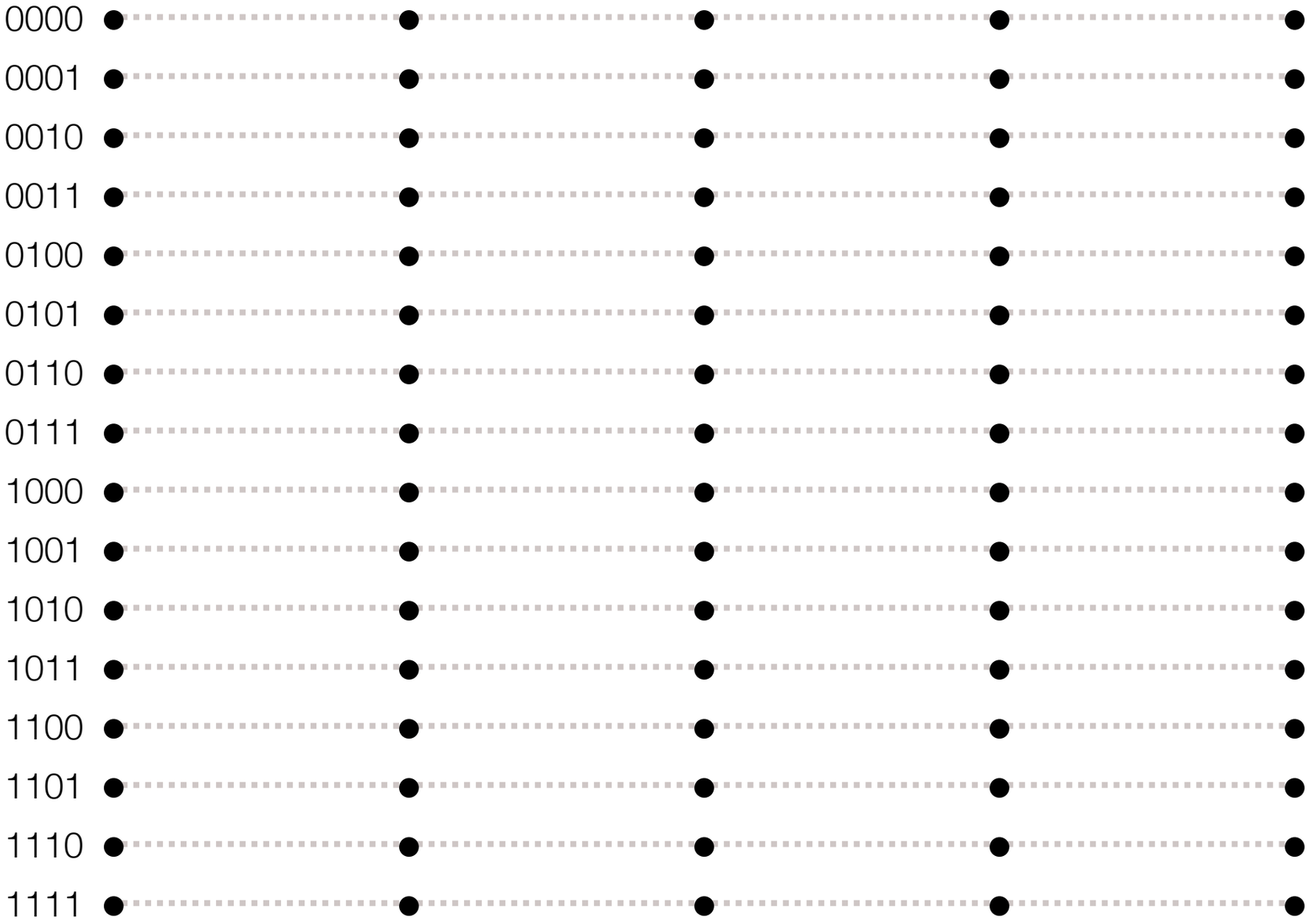
return $\left[x_{\text{even}} + (w. * x_{\text{odd}}),\ x_{\text{even}} - (w. * x_{\text{odd}})\right]$

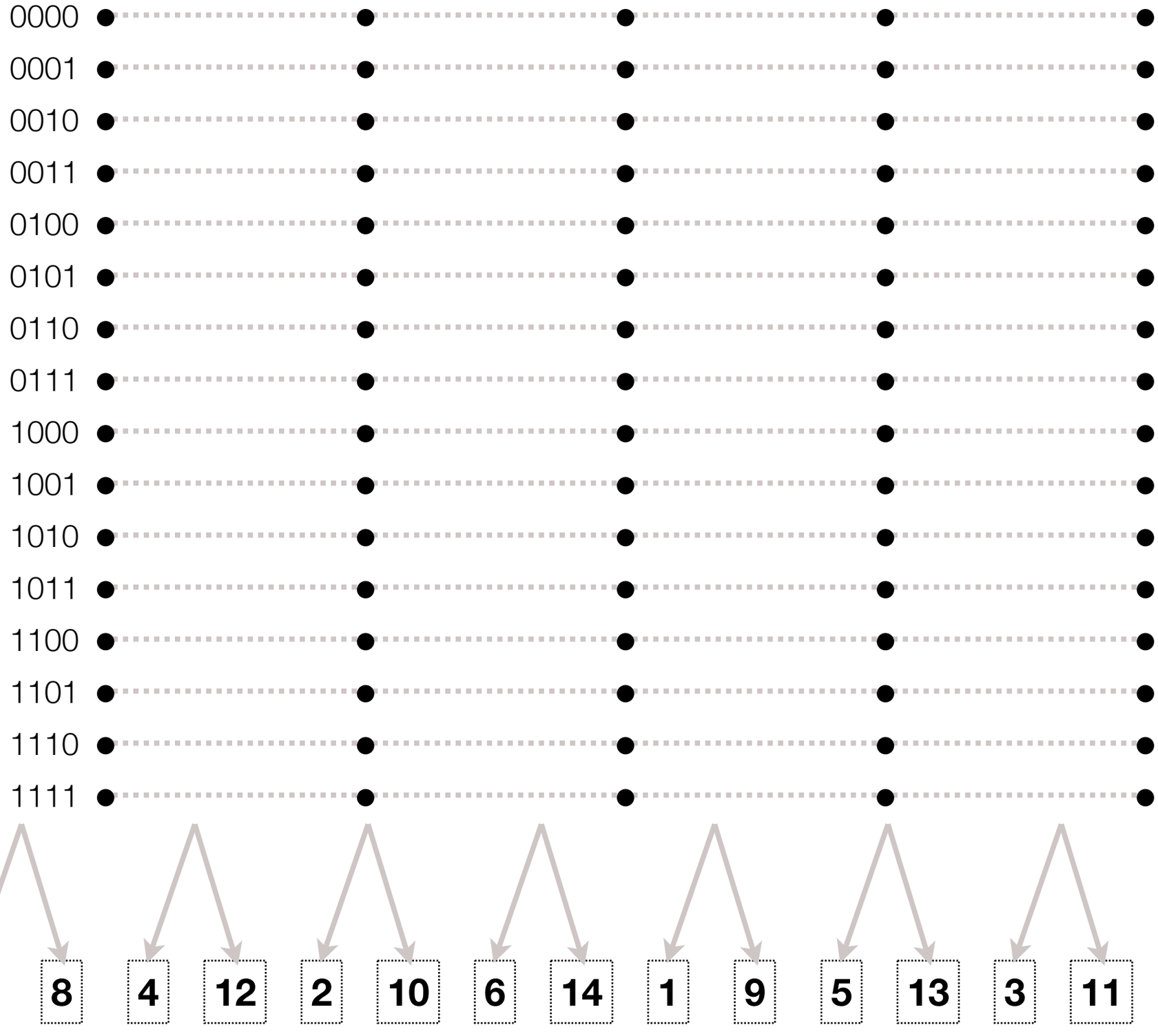
Call-tree

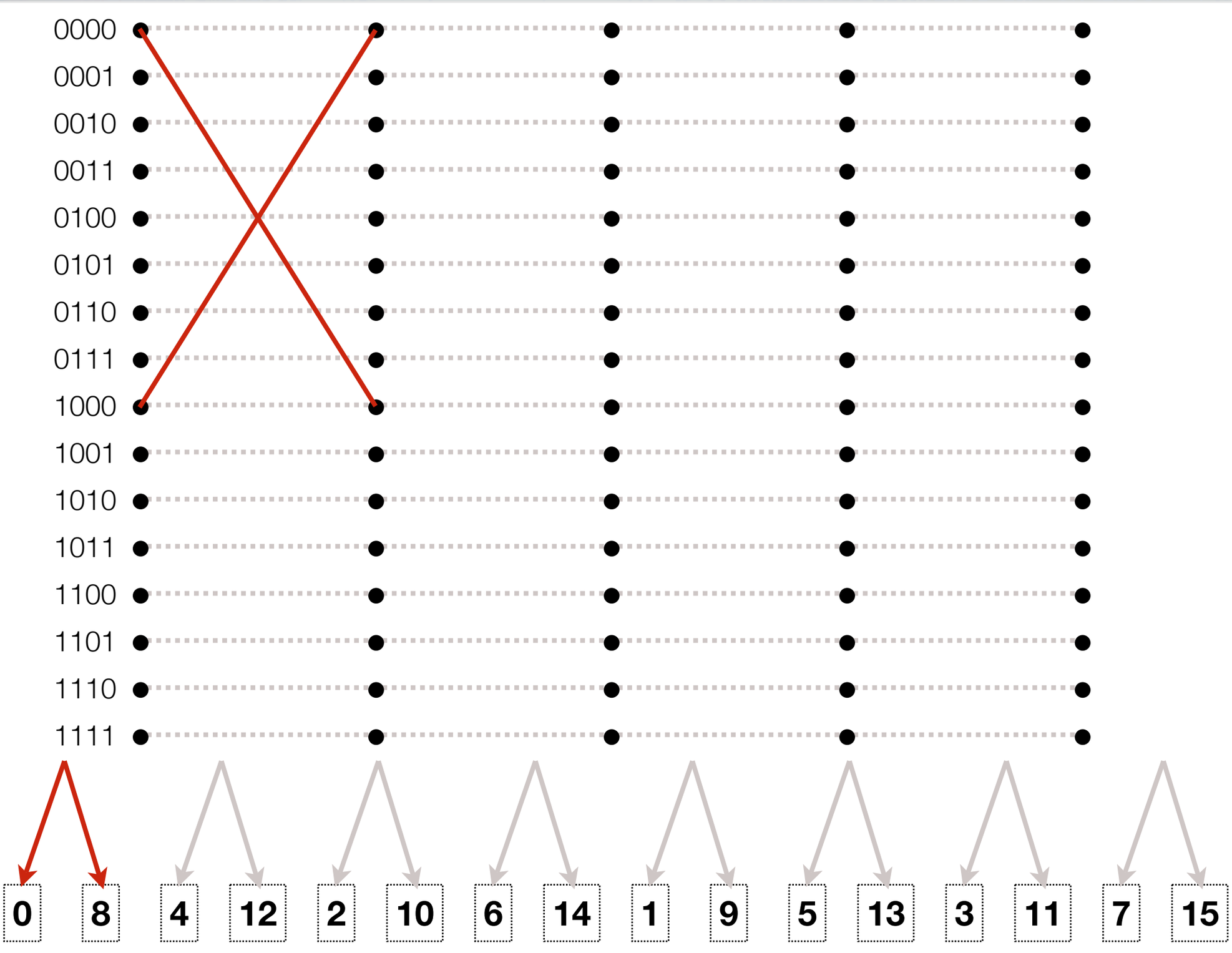

FFT(0,1,2,3,…,15)
even
odd
FFT(0,2,…,14) = FFT(xxx**0**)
FFT(1,3,…,15) = FFT(xxx**1**)
FFT(xx**00**)
FFT(xx**10**)
FFT(xx**01**)
FFT(xx**11**)
x**000**
x**100**
x**010**
x**110**
x**001**
x**101**
x**011**
x**111**
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

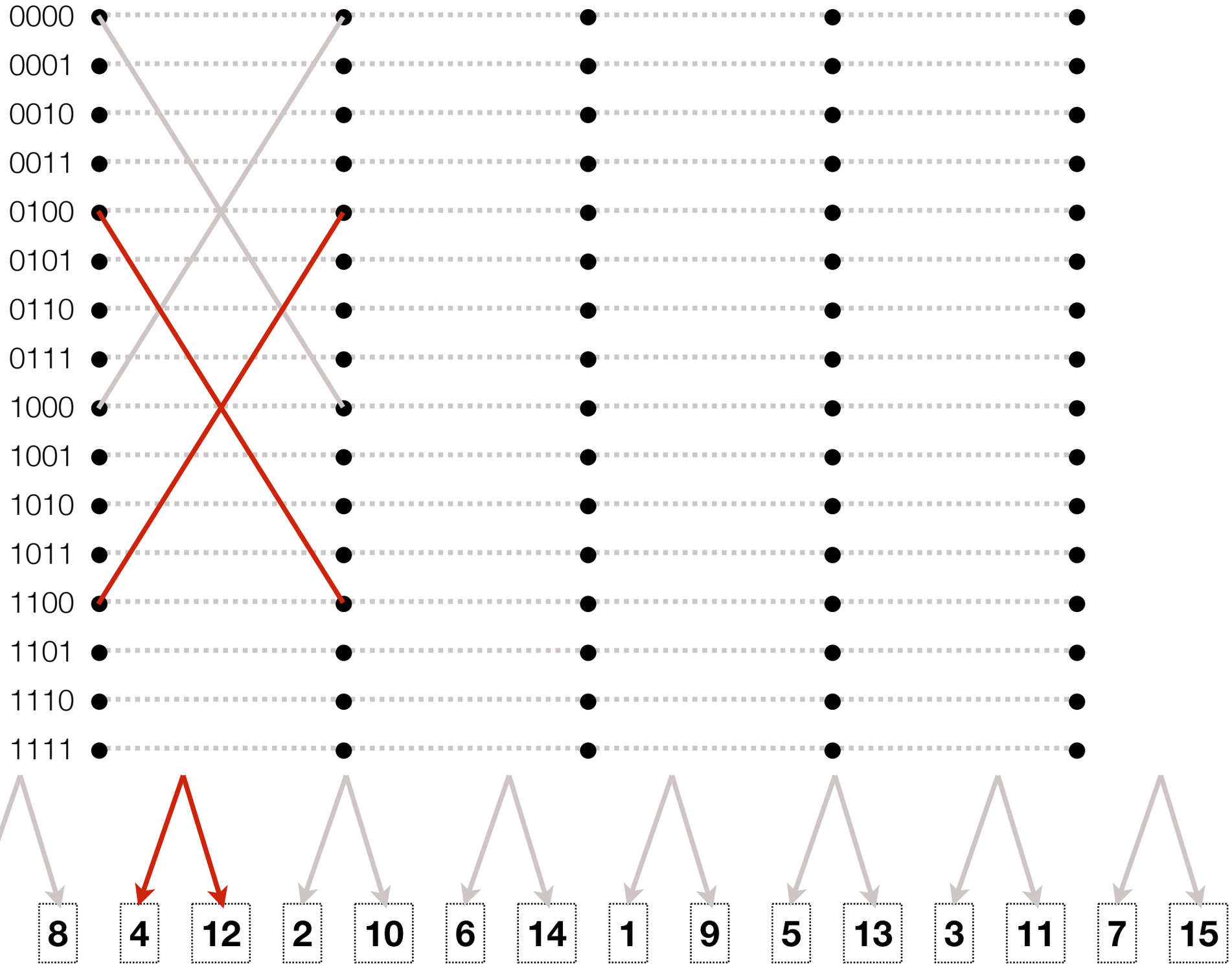
Call-tree

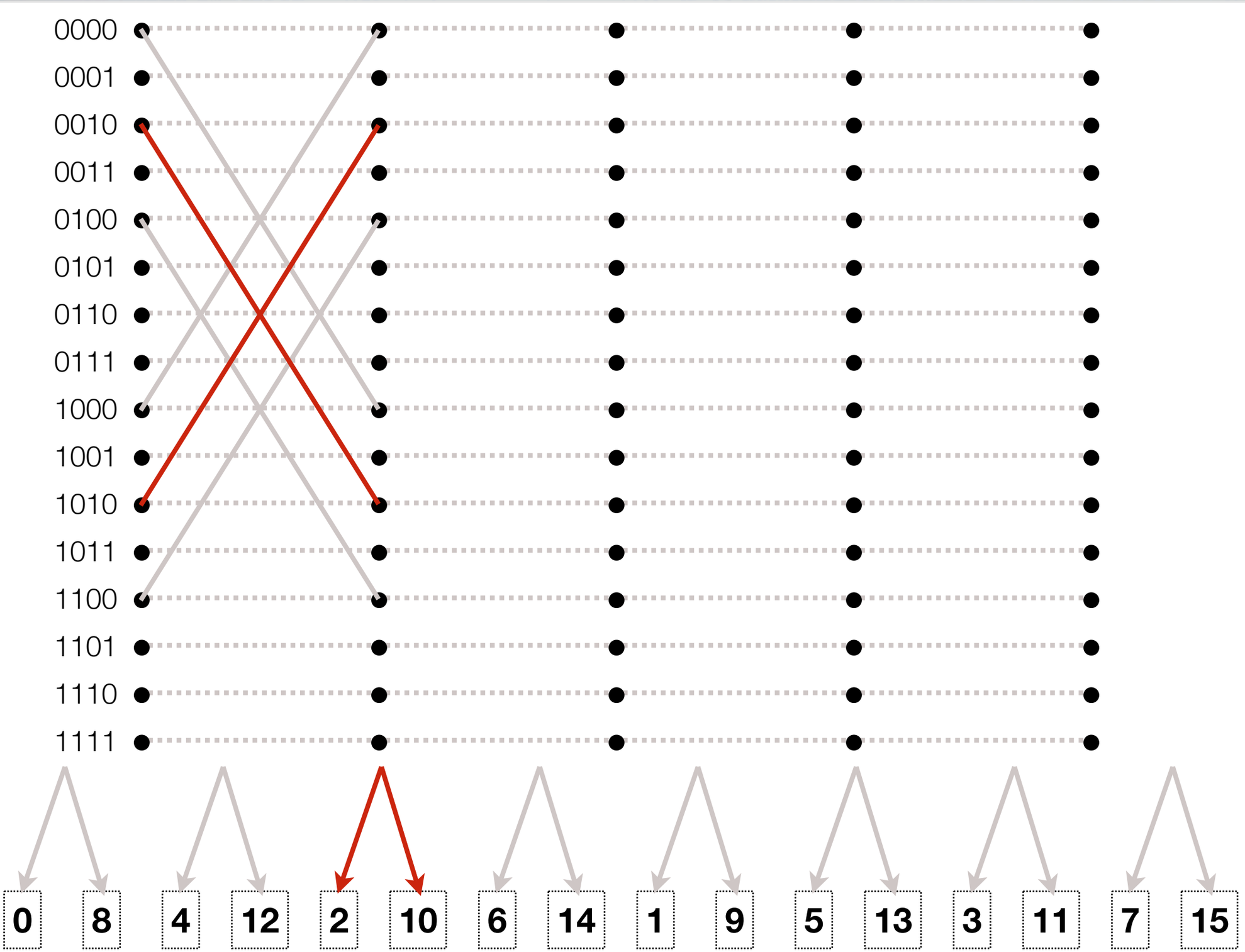

FFT(0,1,2,3,…,15)
even
odd
FFT(0,2,…,14) = FFT(xxx**0**)
FFT(1,3,…,15) = FFT(xxx**1**)
FFT(xx**00**)
FFT(xx**10**)
FFT(xx**01**)
FFT(xx**11**)
x**000**
x**100**
x**010**
x**110**
x**001**
x**101**
x**011**
x**111**
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

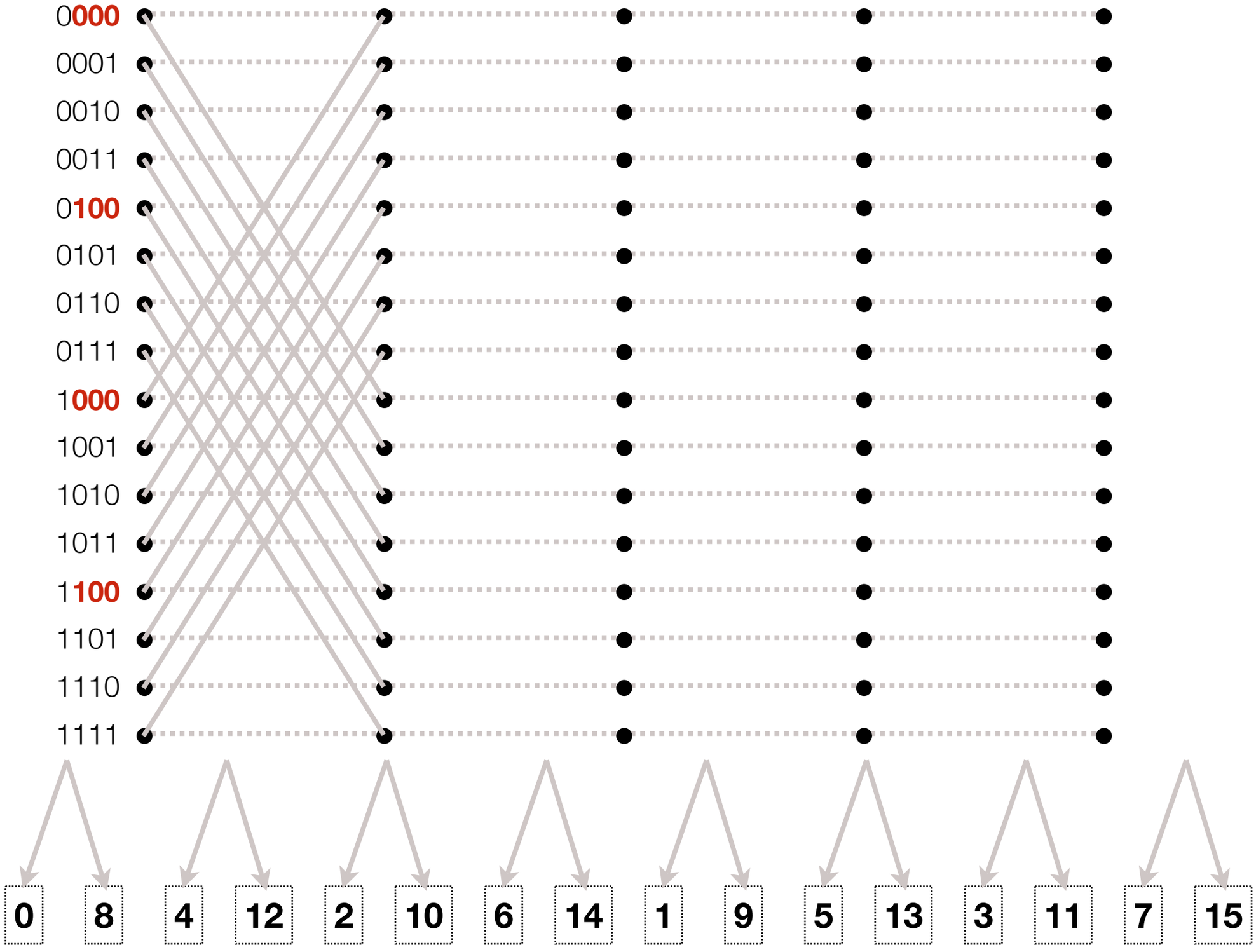

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

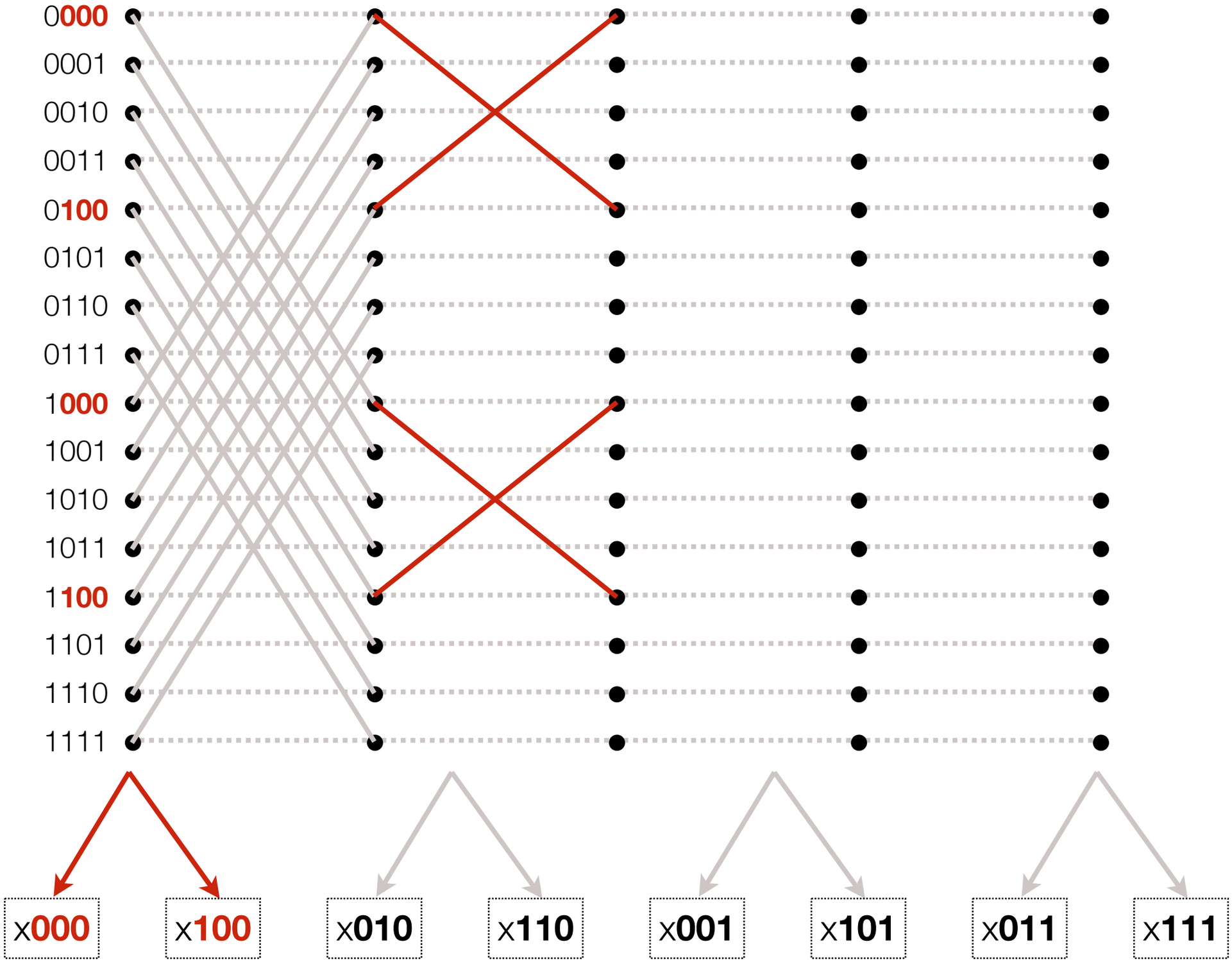

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

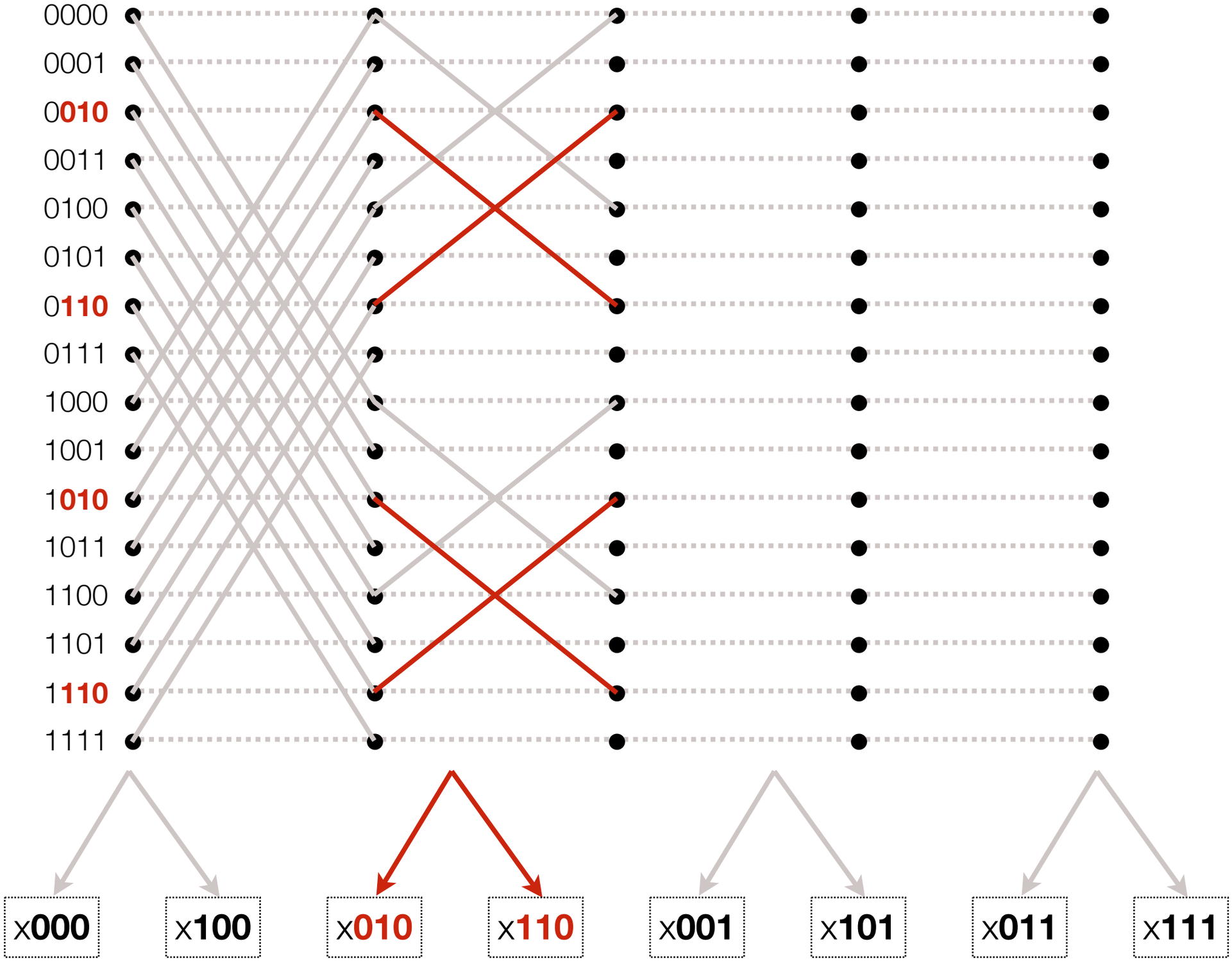

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

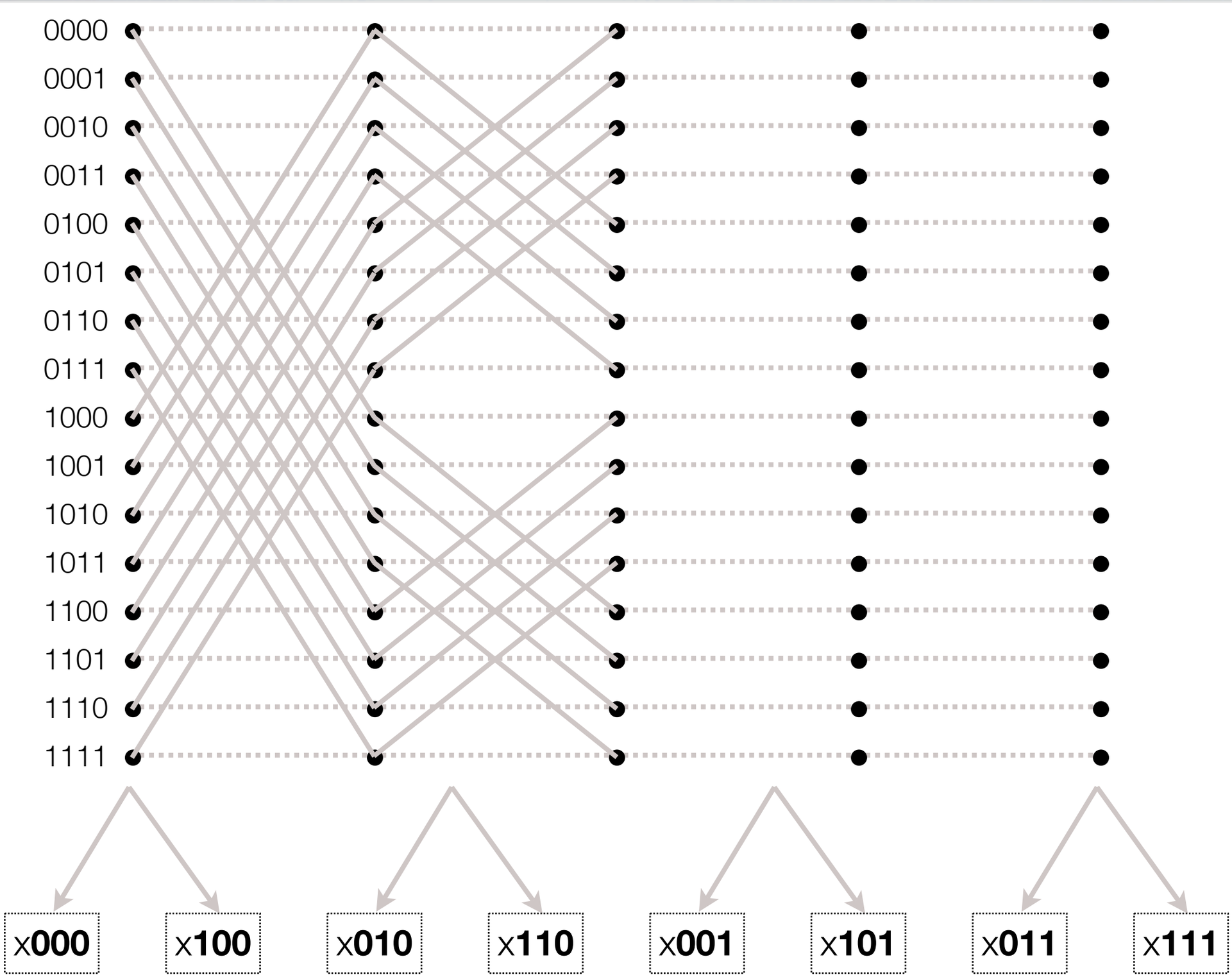

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

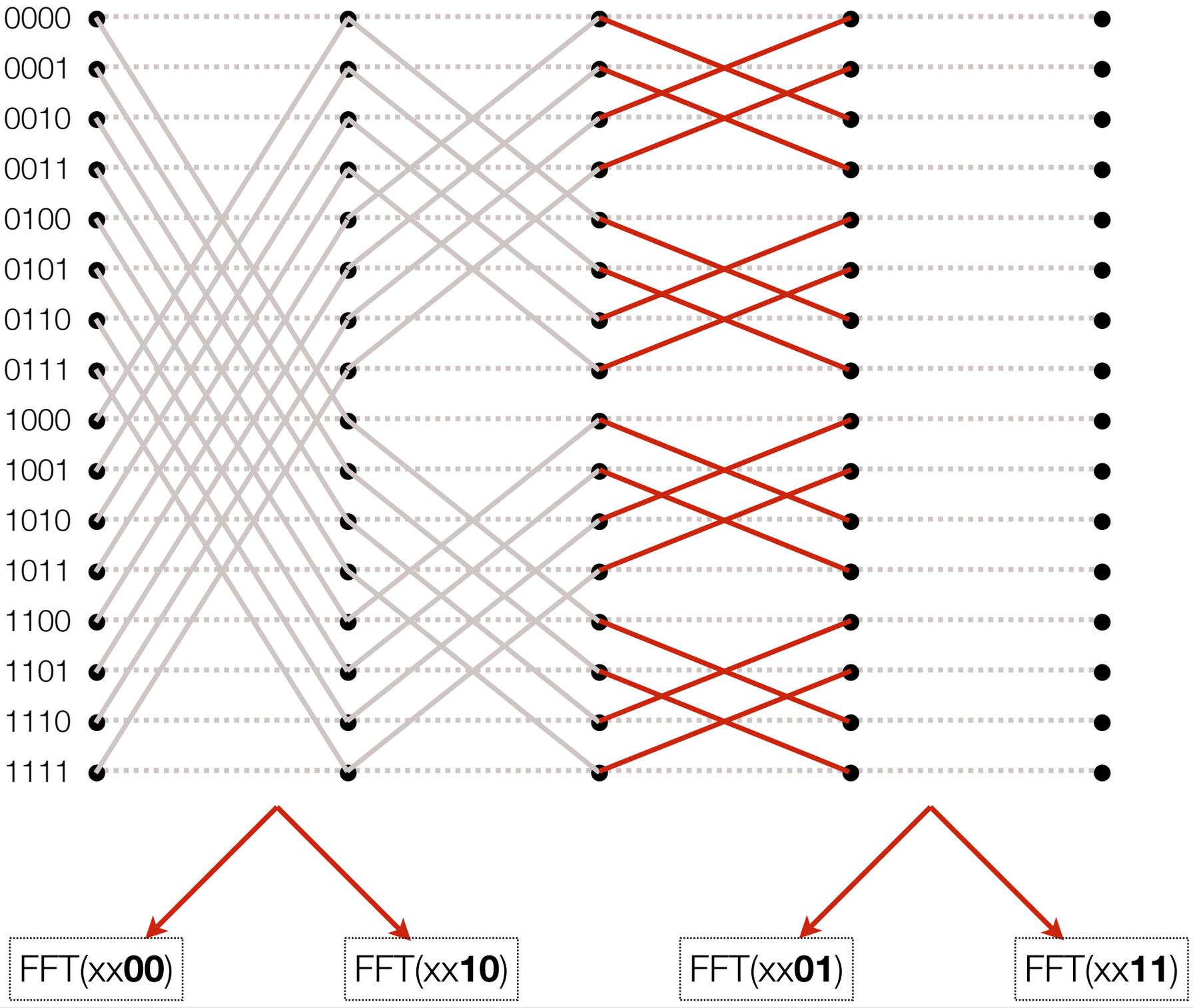

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

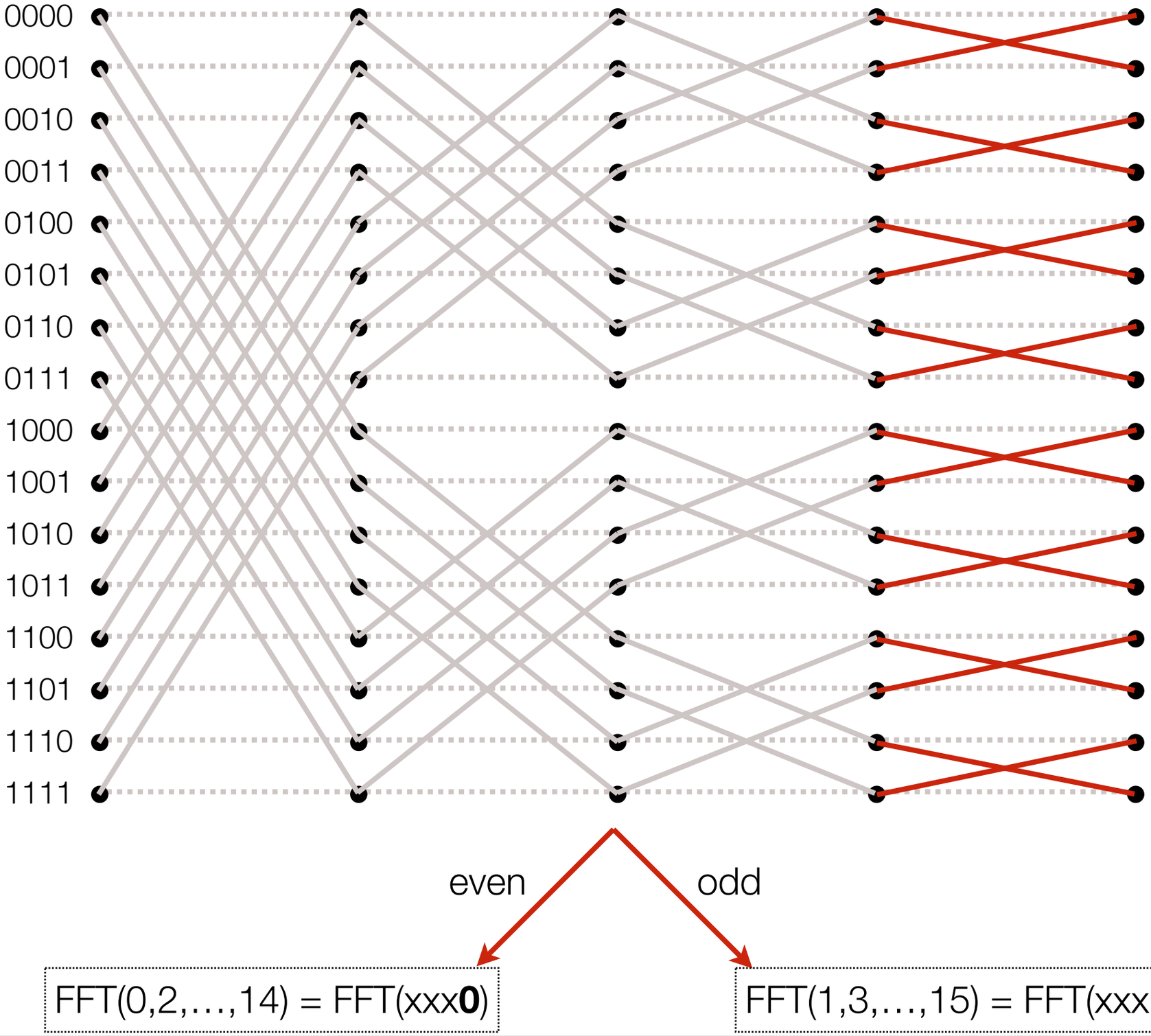

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
8
4
12
2
10
6
14
1
9
5
13
3
11
7
15

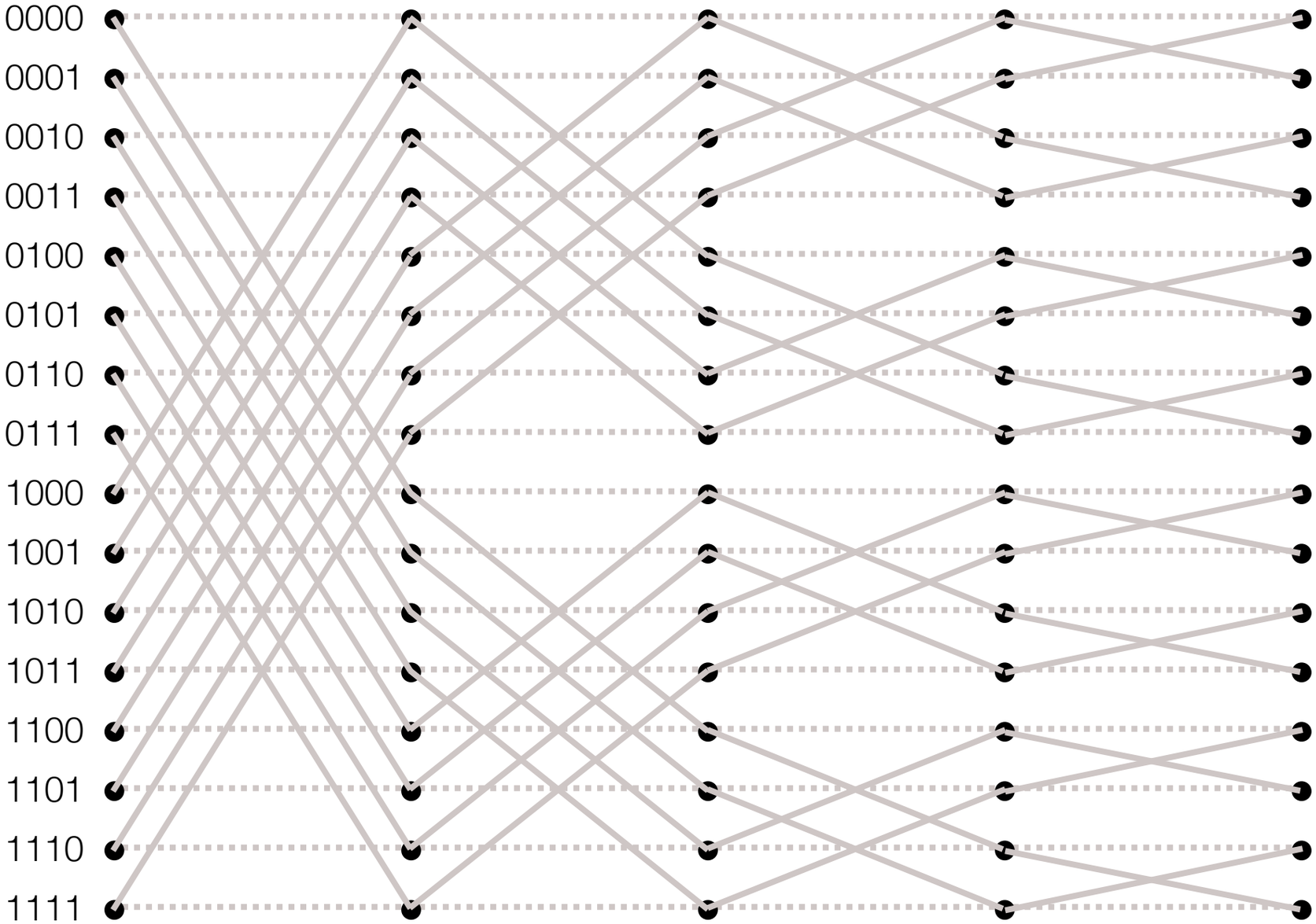

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
x000
x100
x010
x110
x001
x101
x011
x111


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
x000
x100
x010
x110
x001
x101
x011
x111

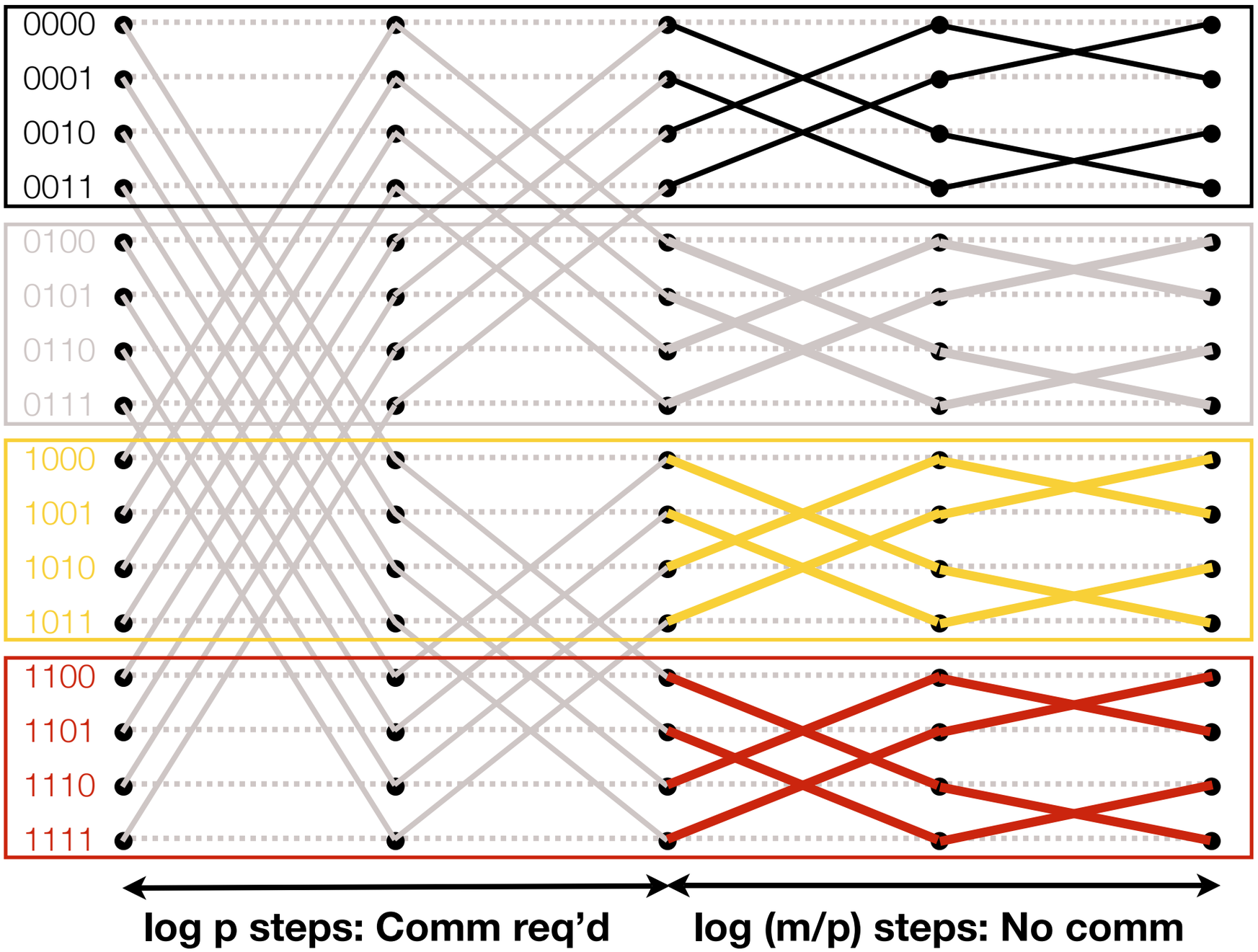

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
x000
x100
x010
x110
x001
x101
x011
x111


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
FFT(xx**00**)
FFT(xx**10**)
FFT(xx**01**)
FFT(xx**11**)

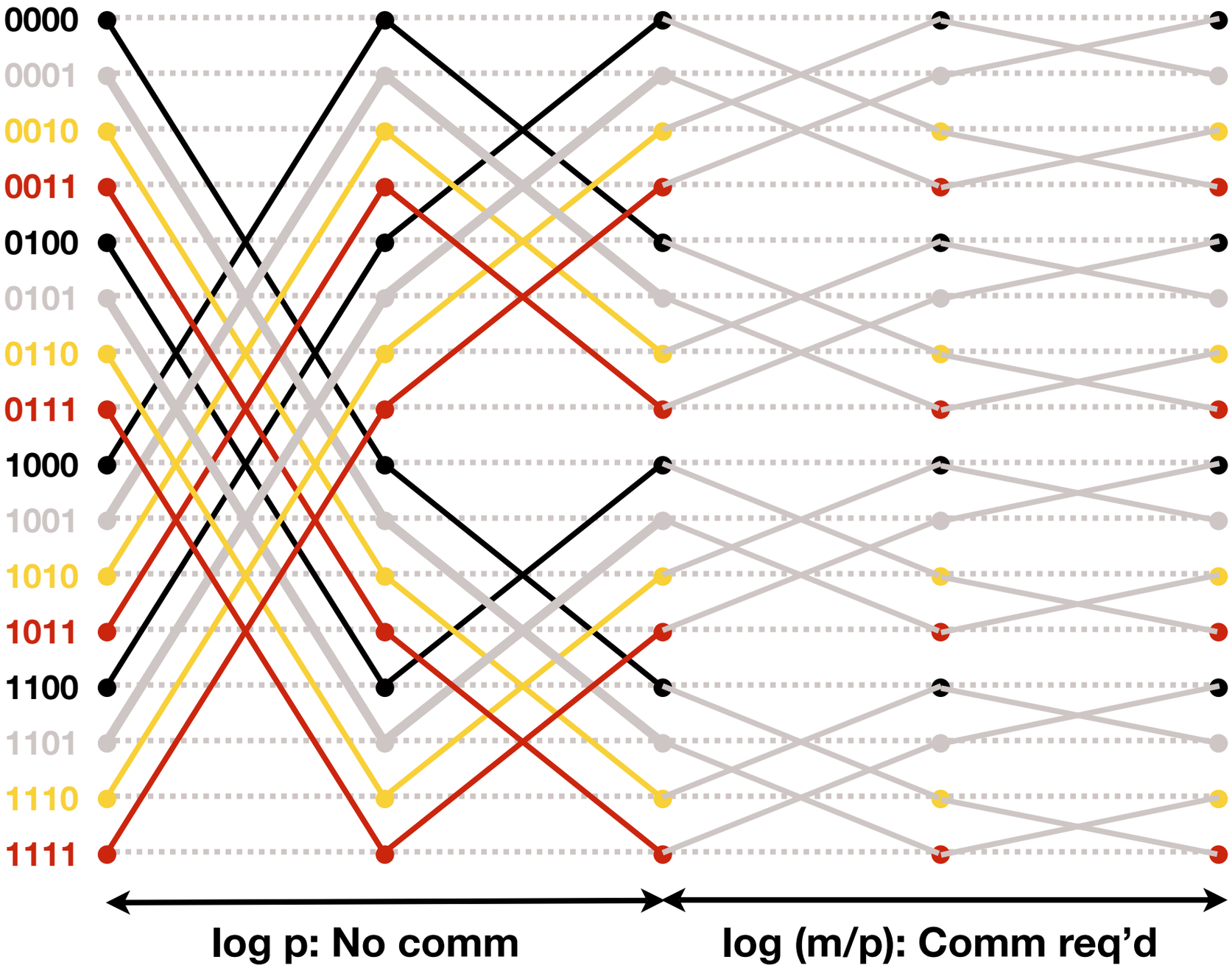

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
even
odd
FFT(0,2,…,14) = FFT(xxx**0**)
FFT(1,3,…,15) = FFT(xxx**1**)



PRAM complexity?



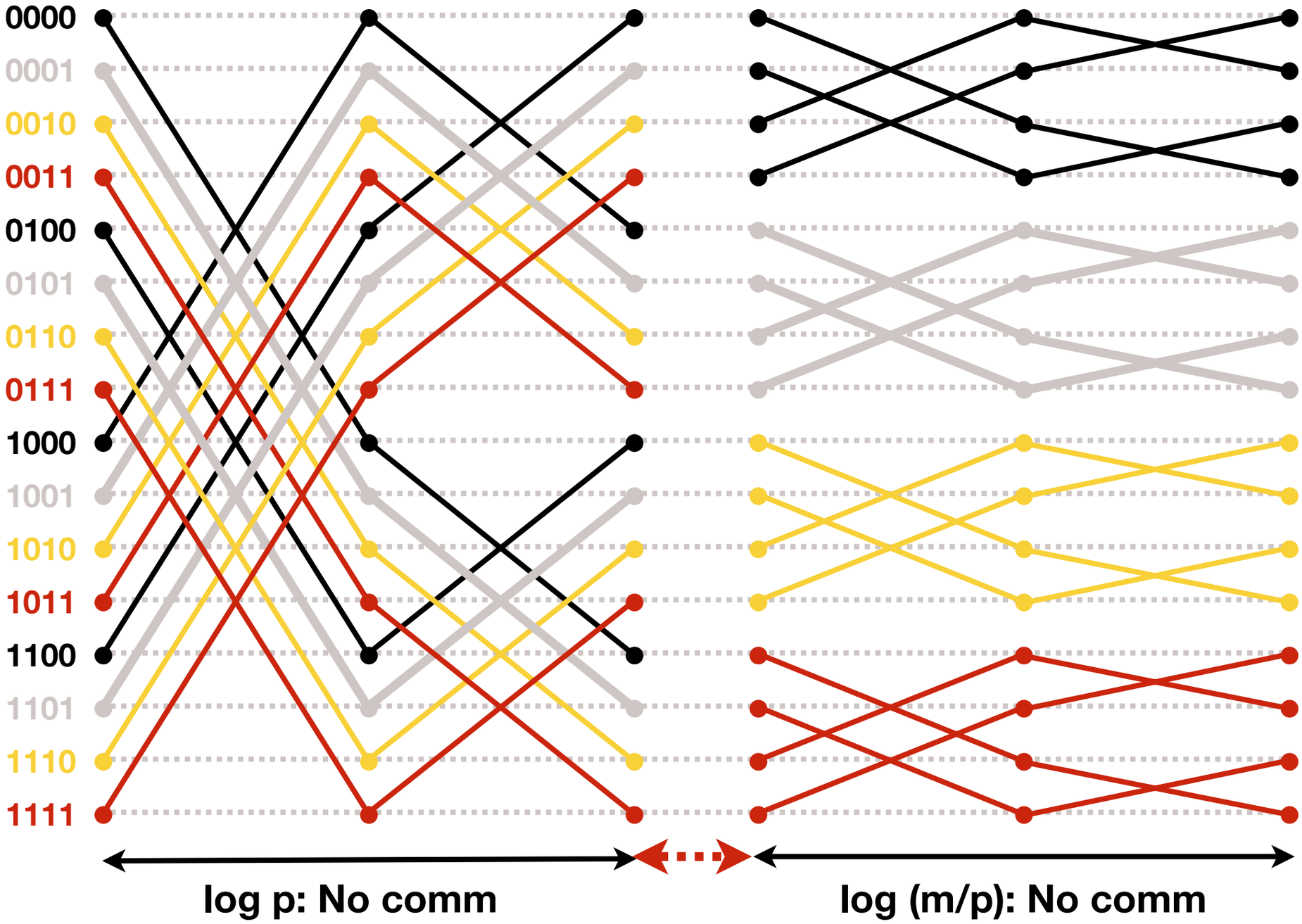
Block Layout
(p=4)



Cyclic Layout
(p=4)

Parallel complexity (block or cyclic)

$$
\begin{aligned}
T_p &= \frac{2m \log m}{p} + \left(\alpha + \frac{1}{\beta}\frac{m}{p}\right) \log p \\
&= \frac{2m \log m}{p} + \alpha \log p + \frac{1}{\beta}\frac{m \log p}{p}
\end{aligned}
$$



FFT with transpose (p=4)

“Transpose” communication step

Cyclic

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Block

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

$$T_p^{(\text{block/cyclic})} = \frac{2m\log m}{p} + \alpha\log p + \frac{1}{\beta}\frac{m}{p}\log p$$

$$T_p^{(\text{transpose})} = \frac{2m\log m}{p} + \alpha(p-1) + \frac{1}{\beta}\frac{m}{p}\frac{p-1}{p}$$

Higher-dimensional FFTs

- *d*-dimensional FFT: Apply 1-D FFT in all dimensions
- 2-D FFT
  - 2-D blocked layout: Apply parallel 1-D for each row and column
  - Block row layout: serial 1-D on rows, parallel 1-D on columns
  - Block row layout: serial 1-D on rows, transpose, serial 1-D again

- **Software**: FFTW (fftw.org)

Future fast Fourier transform?

- Paper by Edelman, McCorquodale, Toledo in SIAM J. Sci. Comput. (1999)
- Suppose input/output vectors req'd to be in block layout
- Standard algorithm: **Transpose** (block to cyclic), local, **transpose**, local
- New algorithms: Approximate Φ matrix and use only 1 communication
  - [1]: SVD-based (rank-k approximation of Φ matrix): Trade flops for less comm.
  - [2]: Approximate Φ^*x using fast multipole method

“In conclusion…”