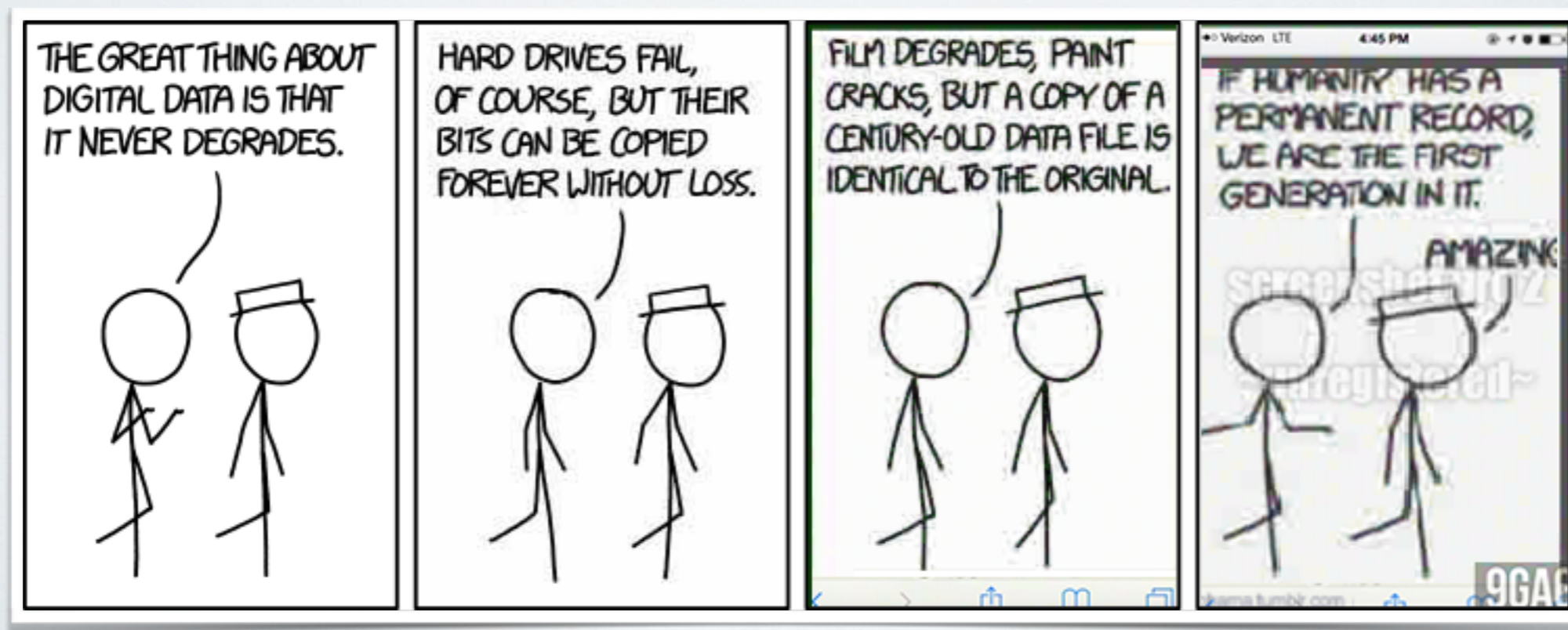


ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 10
27 апреля 2020 г.



СЖАТИЕ (УПАКОВКА) ИНФОРМАЦИИ

- Принцип: сжимаем битовую цепочку за счет специфического кодирования.
- Обычные данные: T_1 = время доступа к данным.
- Сжатые данные:
 T_2 = время доступа к сжатым данным + время на распаковку.
- Часто $T_2 < T_1$!
- Чем меньше размер, тем больше скорость доступа.
- Иногда места действительно мало.



СЖАТИЕ БЕЗ ПОТЕРЬ

Возможно точное (бит-в-бит) восстановление исходной битовой цепочки.



СЖАТИЕ С ПОТЕРЯМИ

«А, все равно не видно (не слышно)...»

JPEG, MPEG и т.д.

ПРОГРАММА НА СЕГОДНЯ

Сжатие без потерь:

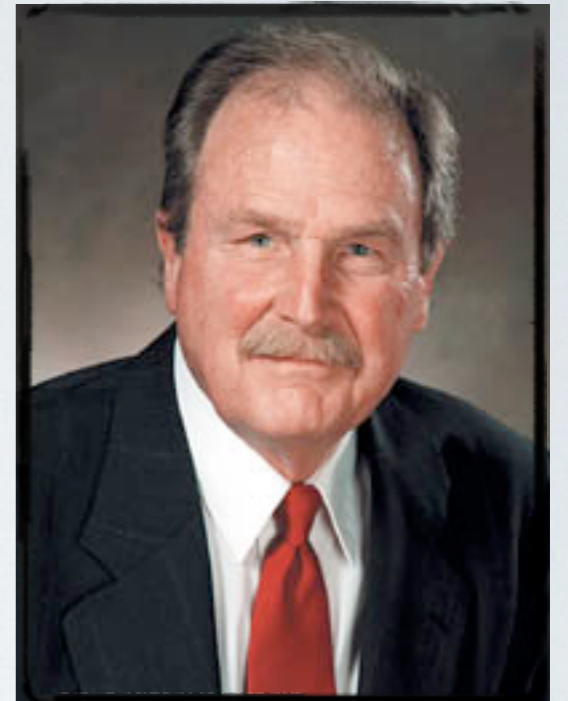
- Алгоритм RLE.
- Алгоритм Хаффмана.
- Алгоритм LZ77.
- Алгоритм LZW.

АЛГОРИТМ RLE

- Run-Length Encoding (сжатие повторяющихся цепочек). РСХ, ILBM.
- Ч/б изображение: ...**ББББББББББЧБББББЧЧБББББЧЧЧЧЧЧЧ**...
- Упаковка: **9Б 1Ч 5Б 2Ч 5Б 9Ч**.
- Общий случай: выбирается редко используемый байт-префикс (**P**).
 - **XXXX** → **P 4 X**.
 - **X** → **X**.
 - **PPPP** → **P 4 P**.
 - **P** → **P 1 P**.

АЛГОРИТМ ХАФФМАНА

- В обычном файле все символы кодируются 8-битовыми цепочками, вне зависимости от частоты их появления.
- **Идея:** кодировать более часто встречающиеся символы более короткими цепочками.
- **Префиксные коды:** ни одна кодовая слово не является префиксом любого другого.
- Первый шаг: подсчет частоты вхождения символов в исходном файле.



ПРИМЕР КОДИРОВАНИЯ ПО ХАФФМАНУ

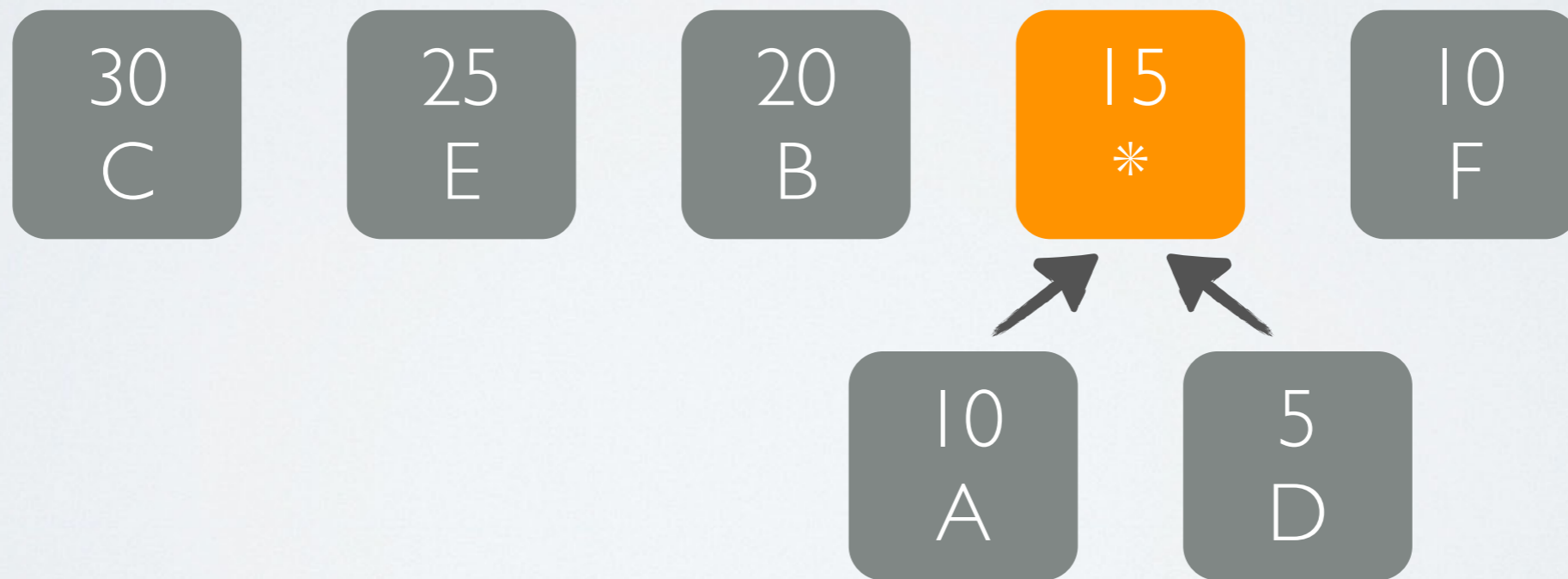
СИМВОЛ	A	B	C	D	E	F
Частота вхождений	10	20	30	5	25	10

Сортируем по числу вхождений:



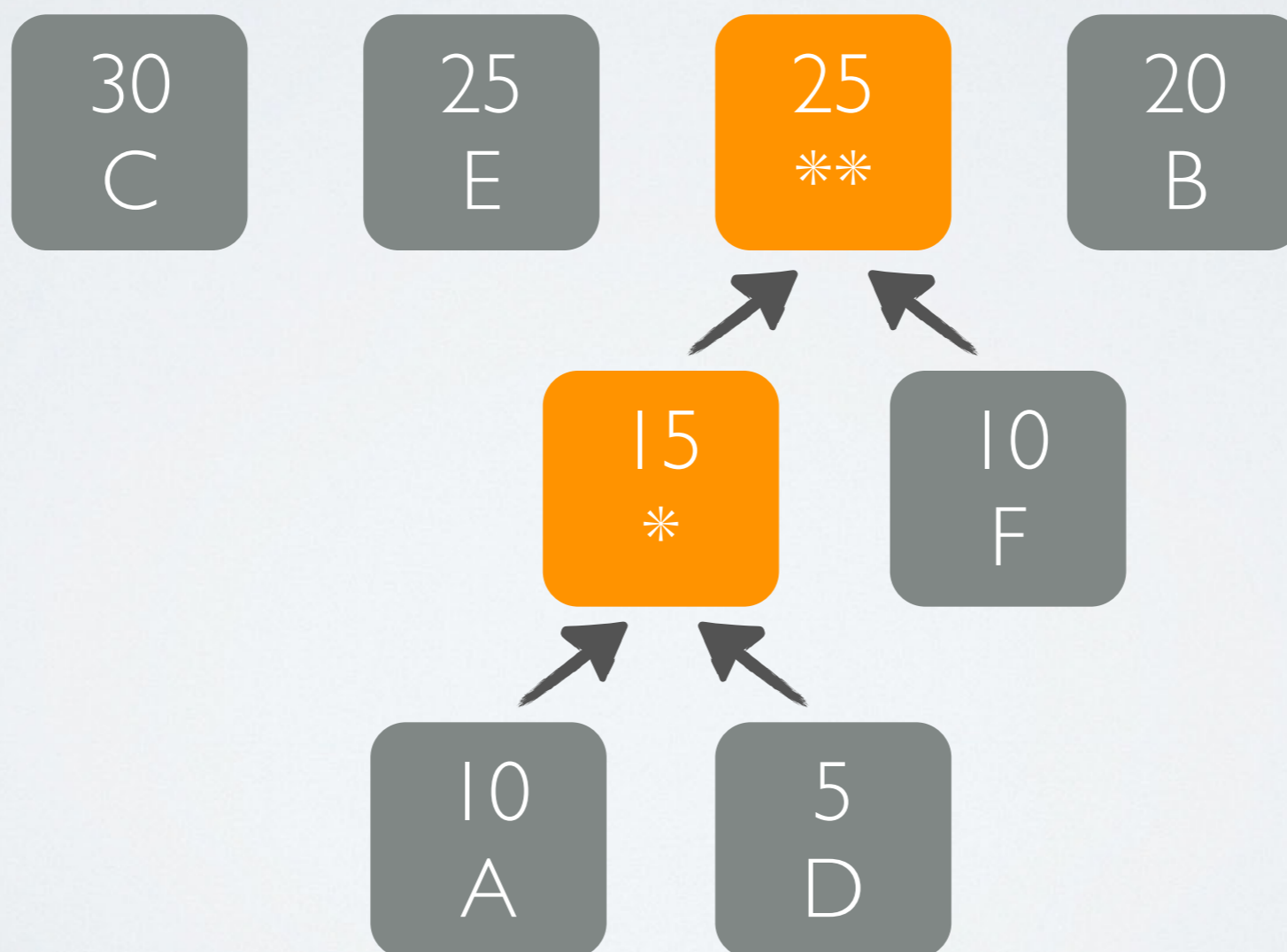
ОБЪЕДИНЕНИЕ (ШАГ 1)

Два символа с минимальной частотой (A и D) формируют «суммарный» узел (*) с частотой 15.

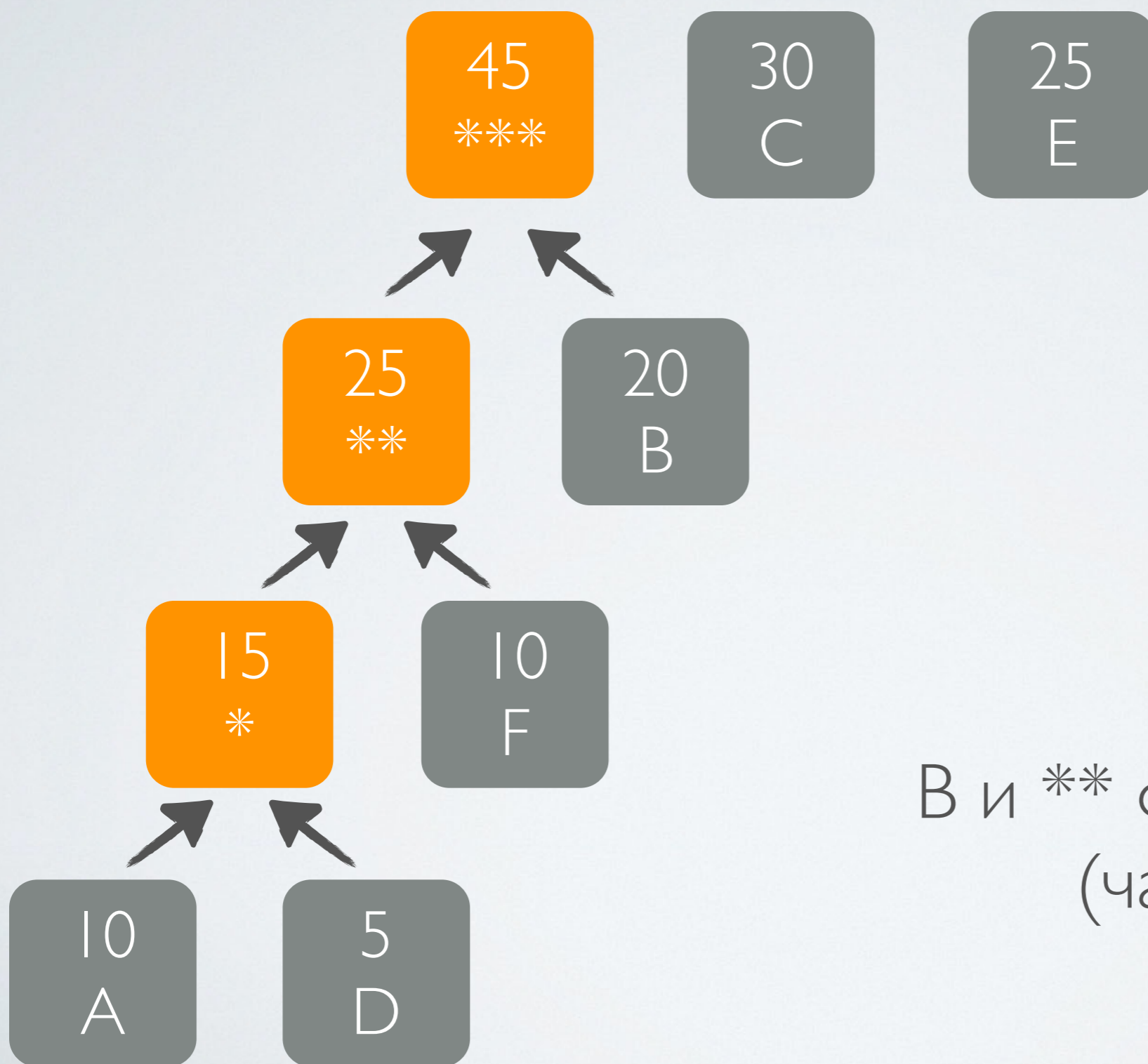


ОБЪЕДИНЕНИЕ (ШАГ 2)

Узлы F и * формируют узел ** (частота 25).

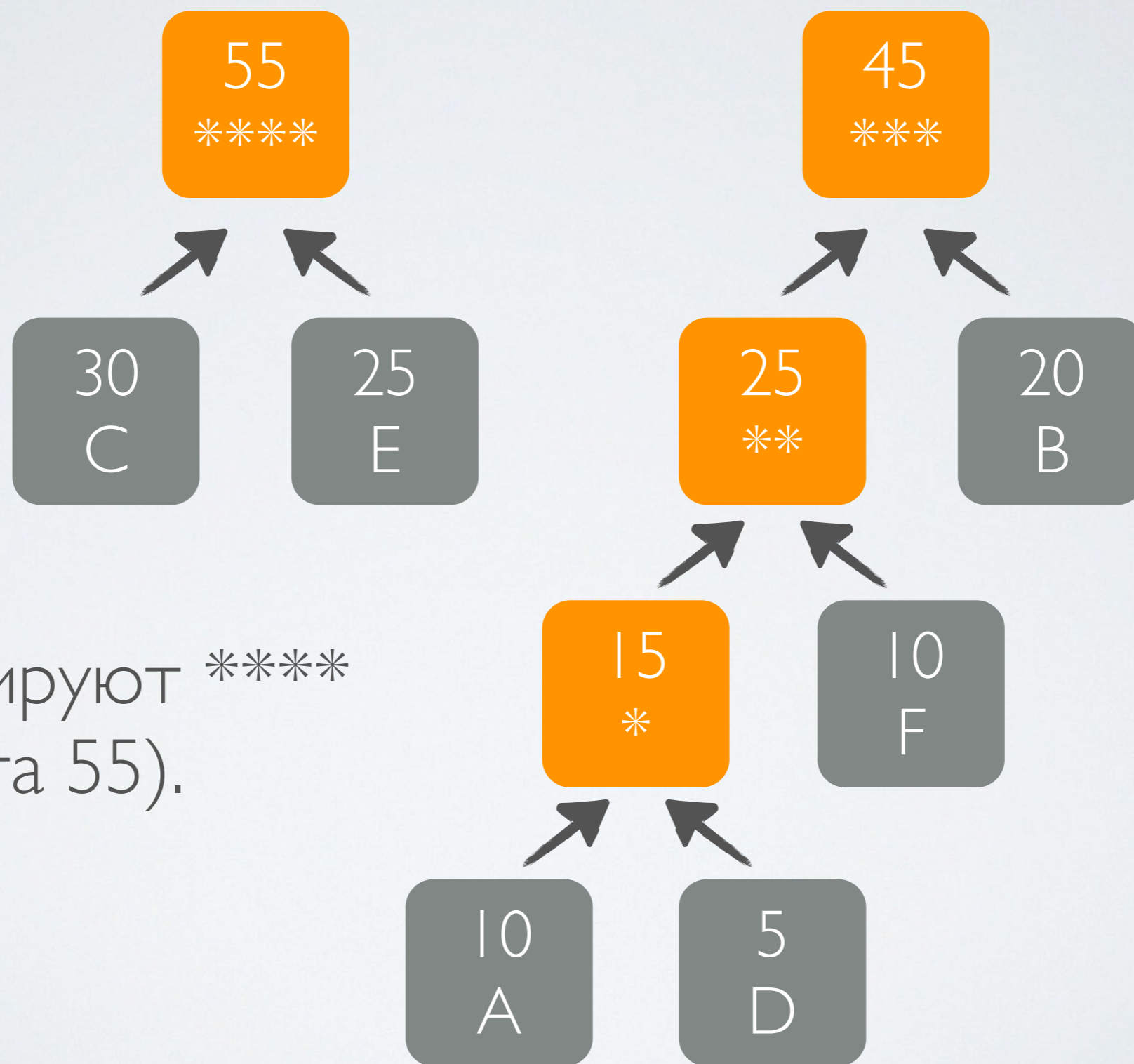


ОБЪЕДИНЕНИЕ (ШАГ 3)



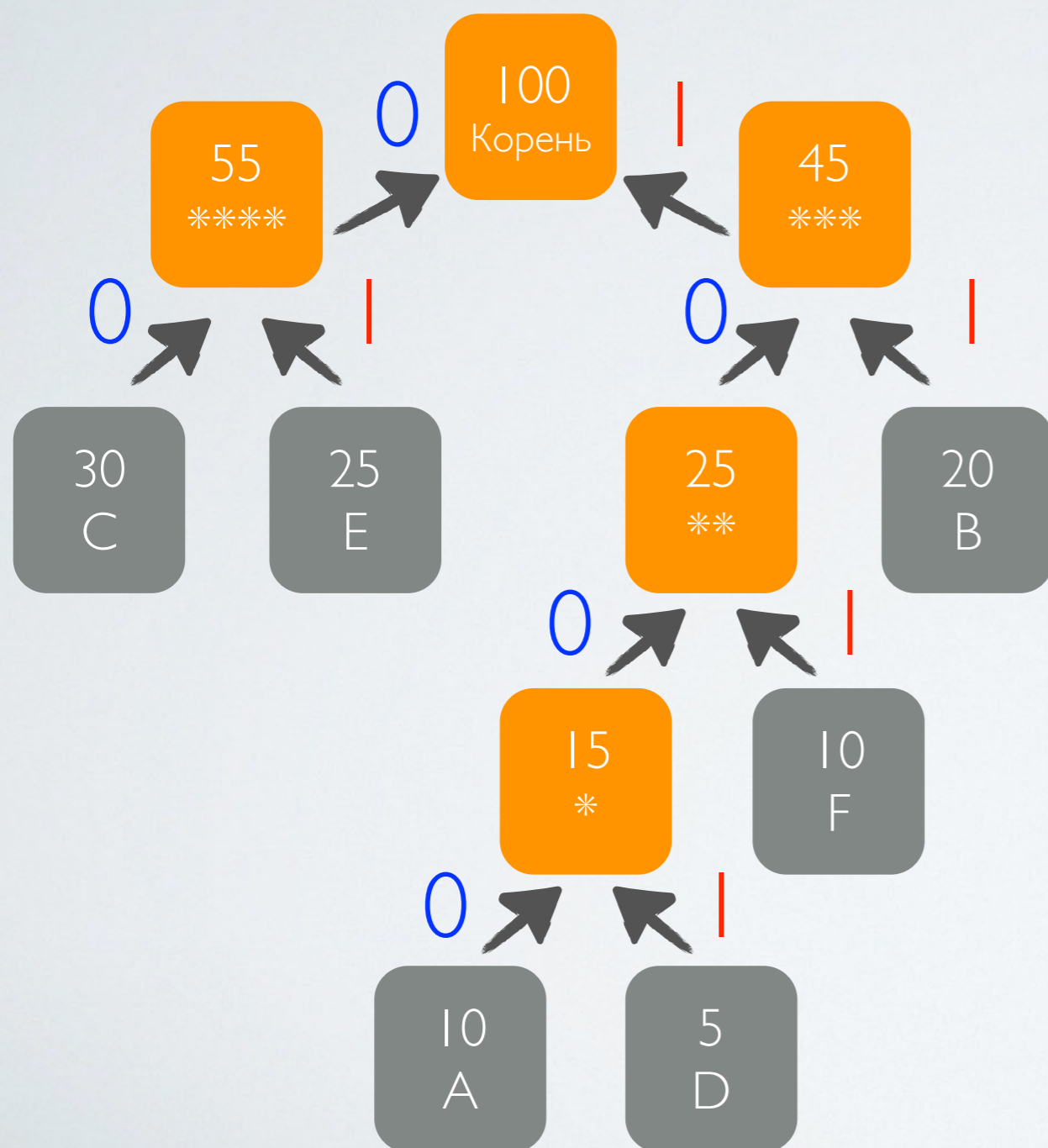
В и ** формируют ***
(частота 45).

ОБЪЕДИНЕНИЕ (ШАГ 4)



С и Е формируют *****
(частота 55).

ИТОГОВОЕ ДЕРЕВО



СИМВОЛ	Частота	К. слово
C	30	00
E	25	01
B	20	11
F	10	101
A	10	1000
D	5	1001

КАНОНИЧЕСКИЙ КОД ХАФФМАНА

- Получившуюся таблицу кодов сортируем по парному ключу (длина цепочки, ASCII код символа)
- Первый символ: цепочка «все нули».
- Последующие символы: прибавляем 1 и дополняем нулями справа до нужной длины.

A = 1000
B = 11
C = 00
D = 1001
E = 01
F = 101



B = 11
C = 00
E = 01
F = 101
A = 1000
D = 1001



B = 00
C = 01
E = 10
F = 110
A = 1110
D = 1111

РАСПАКОВКА

- Тривиальна при известном дереве (префиксные коды расшифровываются однозначно).
- Канонический код Хаффмана: дерево восстанавливается из таблицы соответствий «символ – длина кодового слова» при известной максимальной длине кода.
- Для упаковки массива длин размером 256 можно воспользоваться RLE.

НА СКОЛЬКО СЖАЛОСЬ?

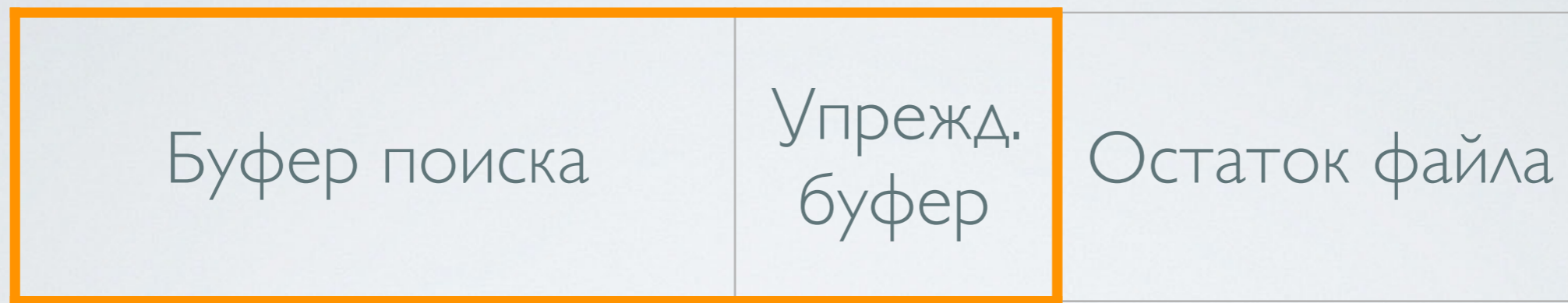
- Было: $100 \text{ байт} \times 8 \text{ бит} = 800 \text{ бит}$.
- Стало: $(30+25+20) \times 2 + 10 \times 3 + (10+5) \times 4 = 240 \text{ бит}$.
- Набор длин кодов: $6 \text{ букв} \times 2 \text{ бита} + 16 \text{ (RLE)} + 8 \text{ (макс. длина)} = 36 \text{ бит}$.
- Итого: $(240+36) / 800 = 34,5 \%$.

ПРОГРАММА НА СЕГОДНЯ

- ✓ Алгоритм RLE.
- ✓ Алгоритм Хаффмана.
- Алгоритм LZ77.
- Алгоритм LZW.

АЛГОРИТМ LZ77

Окно



Текущий указатель

- В окне ищется максимальная последовательность, соответствующая содержимому упреждающего буфера.
- В выходной поток пишется тройка (смещение, длина, след. символ в буфере).
- Окно сдвигается на длину найденной последовательности + 1.

ПРИМЕР LZ77

Исходные данные: «abracadabra».

Красным цветом отмечен буфер поиска.

Синим цветом отмечена совпадающая последовательность.

Окно	Позиция	Длина	Символ
abracadabra	0	0	a
a bracadabra	0	0	b
ab racadabra	0	0	r
abra cadabra	3	1	c
abrac adabra	2	1	d
abracad abra	7	4	нет

Код: (0, 0, a), (0, 0, b), (0, 0, r), (3, 1, c), (2, 1, d), (7, 4, -)

ПРИМЕР LZ77

Исходные данные: «abababcab».

Красным цветом отмечен буфер поиска.

Синим цветом отмечена совпадающая последовательность.

Окно	Позиция	Длина	Символ
abababcab	0	0	a
a bababcab	0	0	b
a babab cab	2	4	c
a babab cab	3	2	нет

Код: (0, 0, a), (0, 0, b), (2, 4, c), (3, 2, -)

АЛГОРИТМ LZW

- Заводим таблицу строк (например, размером 4096). Первые 256 кодов соответствуют символам ASCII.
- Алгоритм.
 - Ищем с текущей позиции строку **s** максимальной длины, существующую в таблице.
 - Выводим в файл индекс **s** в массиве (код).
 - Если файл кончился, то выход, иначе добавляем в таблицу строку **s+c** (**c** – следующий символ) и устанавливаем текущую позицию на символ **c**.

ПРИМЕР LZW

X A B C X A B X A B B X A B B X A B D

s	c	Выходной поток	Новые коды
X	A	X	256 = XA
A	B	A	257 = AB
B	C	B	258 = BC
C	X	C	259 = CX
XA	B	256	260 = XAB
B	X	B	261 = BX
XAB	B	260	262 = XABB
BX	A	261	263 = BXA
AB	B	257	264 = ABB
BXA	B	263	265 = BXAB
B	D	B	266 = BD
D	Конец файла	D	нет

LZW РАСПАКОВКА

- Прочитать **Код₀** из файла и вывести **Таблица[Код₀]**.
- Пока входной поток не пуст:
 - читаем **Код₁** из файла;
 - выводим **Таблица[Код₁]**;
 - добавляем **Таблица[Код₀] + ПервыйСимвол(Таблица[Код₁])**;
 - заменяем **Код₀ ← Код₁**.

ЕСТЬ ПРОБЛЕМА С РАСПАКОВКОЙ

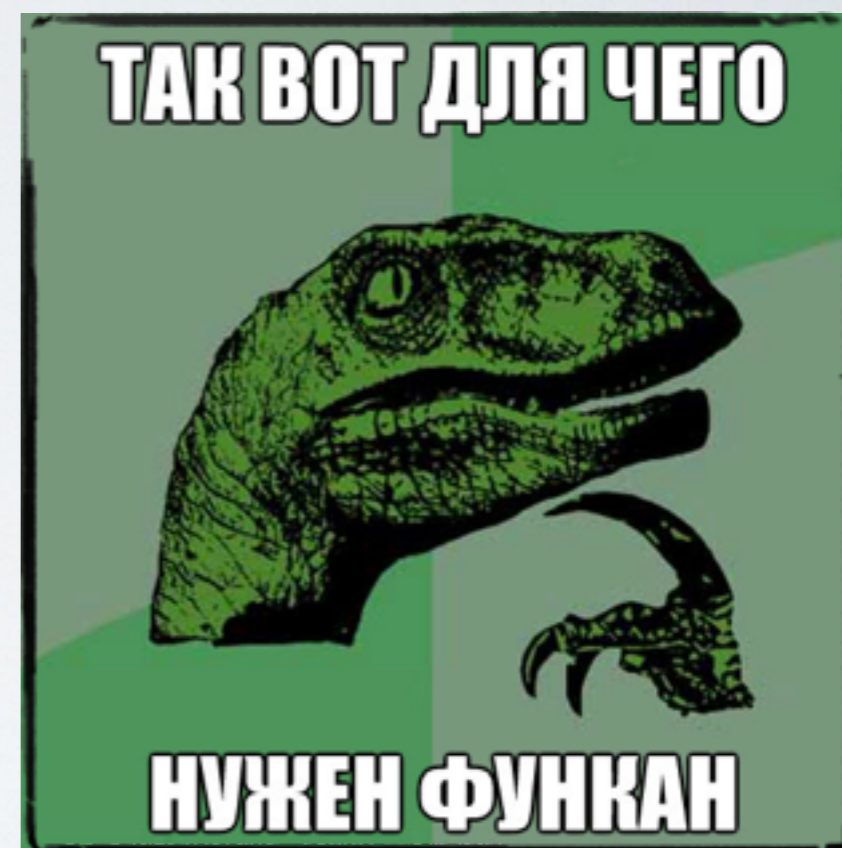
- Пусть строка XYZ есть в таблице (код 333). Пусть далее:
.....XYZXYZXYZ.
- $s=XYZ$, $c=X$, выводим 333, добавляем код 400=XYZX.
- На следующем шаге выводим код 400 (XYZX).
- Распаковщик увидит код 400 раньше, чем он определен.
- Решение: если считан неизвестный код, он будет соответствовать цепочке **Код₀** + ПервыйСимвол(**Код₀**).

СЖАТИЕ С ПОТЕРЯМИ

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ix\omega} dx$$

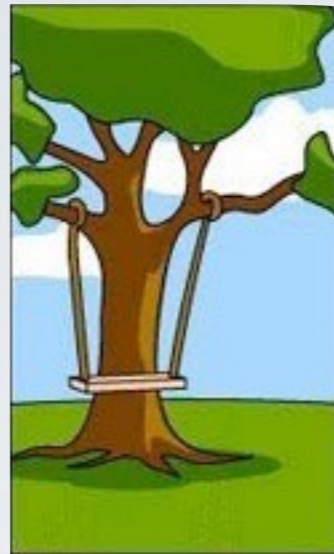
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}$$

$$y[n] = (x * g)[n] = \sum_{k=-\infty}^{\infty} x[k]g[n - k]$$





Как объяснил клиент
чего он хочет



Как понял клиента
начальник проекта



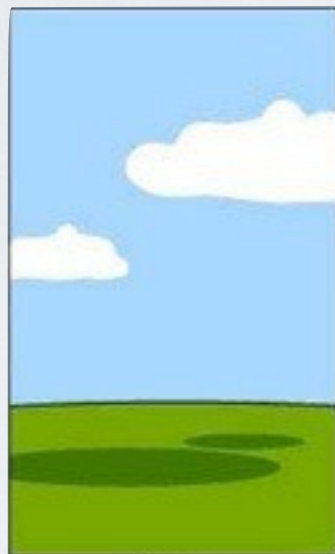
Как описал проект
аналитик



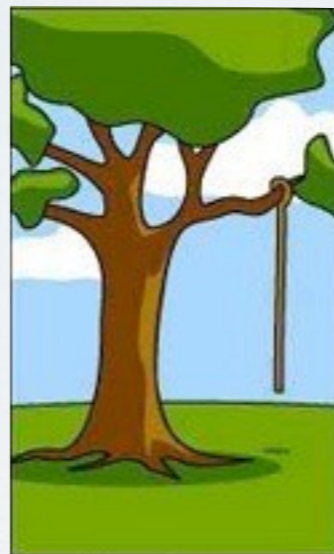
Как написал
программист



Как представил проект
бизнес-консультант



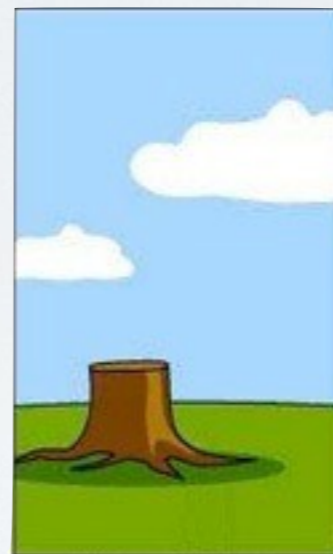
Как
задокументировали
проект



Какие фичи
удалось внедрить



Как заплатил
клиент



Как работала
техническая
поддержка



Что было нужно
клиенту

КОНЕЦ ДЕСЯТОЙ ЛЕКЦИИ