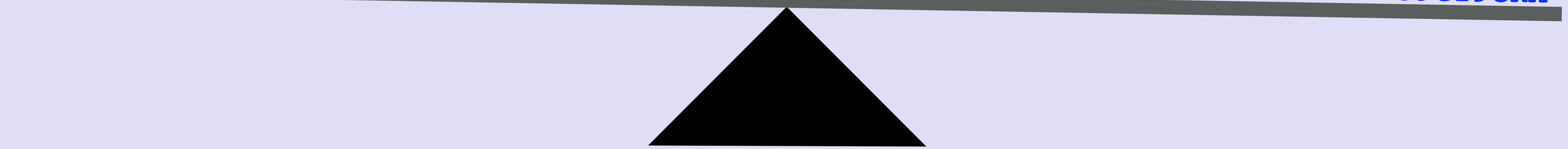
IOTA TUTORIAL 16

normalizedBundleHash

NBBKCKPFECRKCDIBKSTHZYZKSXFEUPTIJRK9FECFKPTTSWTLUWGIFS9AHSDT9LASABRD9KDVFJ9GT9CKA



INTRO

- In this video I will explain how the normalizedBundleHash is created and what it purpose is.

NORMALIZED BUNDLEHASH

- How the normalizedBundleHash is created, see:
  https://github.com/iotaledger/iota.lib.js/blob/v0.4.7/lib/crypto/bundle/bundle.js
  `Bundle.prototype.normalizedBundle = function(bundleHash)`

- The normalizedBundleHash is created by using the bundleHash.
  If you do not know what the bundleHash is, watch IOTA tutorial 15.

- The normalizedBundleHash is used to create or validate a signature.
  This will be explained in detail in IOTA tutorial 17.

- In this tutorial we assume the bundleHash (=81 trytes) is already created, for example:
  bundleHash = **`NBBKCKPFECRKCDIBKSTHZYZKSXFEUPTIJRK9FECFKPTTSWTLUWGIFS9AHSDT9LASABRD9KDVFJ9GT9CKA`**

NORMALIZED BUNDLEHASH

- Divide the bundleHash in 3 parts, each part has 27 trytes:

```
bundleHash = NBBKCKPFECRKCDIBKSTHZYZKSXF|EUPTIJRK9FECFKPTTSWTLUWGIFS|9AHSDT9LASABRD9KDVFJ9GT9CKA
part0: NBBKCKPFECRKCDIBKSTHZYZKSXF
part1: EUPTIJRK9FECFKPTTSWTLUWGIFS
part2: 9AHSDT9LASABRD9KDVFJ9GT9CKA
```

- For each part convert each tryte to its decimal value.

```
part0: N=-13,B=2,B=2,K=11,..
part1: E=5,U=-6,P=-11,T=-7,..
part2: 9=0,A=1,H=8,S=-8,..
```

- For each part calculate the sum.

```
part0: sum = -13 + 2 + 2 + 11,.. = 45
part1: sum = 5 + -6 + -11 + -7,.. = 5
part2: sum = 0 + 1 + 8 + -8,.. = 42
```

IOTA TRYTE ALPHABET

Trits	Dec	Tryte		Trits	Dec	Tryte
0, 0, 0	0	9				
1, 0, 0	1	A		-1,-1,-1	-13	N
-1, 1, 0	2	B		0,-1,-1	-12	O
0, 1, 0	3	C		1,-1,-1	-11	P
1, 1, 0	4	D		-1, 0,-1	-10	Q
-1,-1, 1	5	E		0, 0,-1	-9	R
0,-1, 1	6	F		1, 0,-1	-8	S
1,-1, 1	7	G		-1, 1,-1	-7	T
-1, 0, 1	8	H		0, 1,-1	-6	U
0, 0, 1	9	I		1, 1,-1	-5	V
1, 0, 1	10	J		-1,-1, 0	-4	W
-1, 1, 1	11	K		0,-1, 0	-3	X
0, 1, 1	12	L		1,-1, 0	-2	Y
1, 1, 1	13	M		-1, 0, 0	-1	Z

NORMALIZED BUNDLEHASH

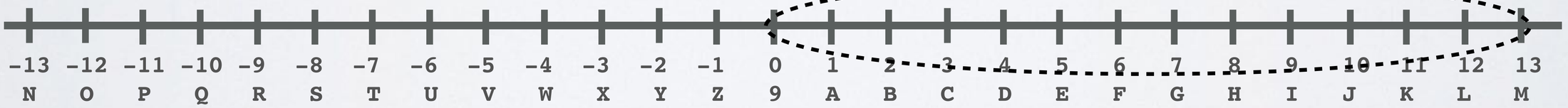
- If the sum >=0 the part has more "weight" on the right side.

- For each part the goal is to redistribute the "weight" evenly.

- This is done by taking the first tryte of the part and **reduce** its tryte value by 1 and calculate the sum.
  Check if the sum < 0, if not keep reducing the tryte value.
  If the tryte value is -13, take the next tryte and do the same as before.
  Stop the process if the sum < 0.

```
First tryte is Y, decimal value = -2
Sum   =  5, 4, 3, 2, 1, 0, -1
Value = -2,-3,-4,-5,-6,-7, -8
NormalizedbundleHash part0 = SHRLM..
```

```
part0: YHRLM.., sum = 5
```



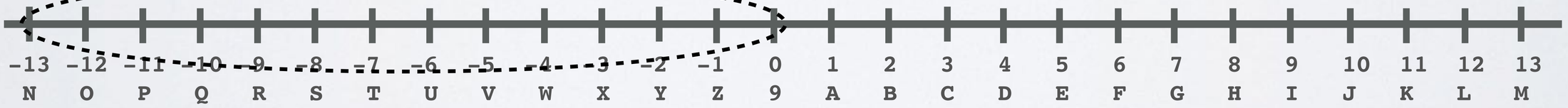
NORMALIZED BUNDLEHASH

- If the sum < 0 the part has more "weight" on the left side.

- Again redistribute the "weight" evenly.

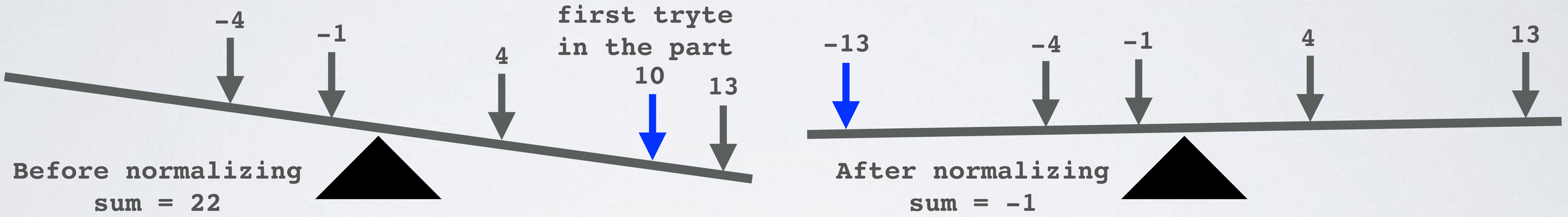
- This is done by taking the first tryte of the part and **increase** its tryte value by 1 and calculate the sum.
  Check if the sum >=0, if not keep increasing the tryte value.
  If the tryte value is +13, take the next tryte and do the same as before.
  Stop the process if the sum >=0.

```
First tryte is K, decimal value = 11
Second tryte is S, decimal value = -8
Sum   = -3,-2,-1,-1, 0
Value = 11,12,13,-8,-7
NormalizedbundleHash part1 = MTION..
```



NORMALIZED BUNDLEHASH

- You can think of normalizing the bundleHash as balancing a seesaw, by manipulating its "weight" (=trytes) to reach a more equilibrium state.



NORMALIZED BUNDLEHASH

- If all 3 parts of the bundleHash are normalized, the normalizedBundleHash is created. It has the same number of trytes as the bundleHash (81 trytes).

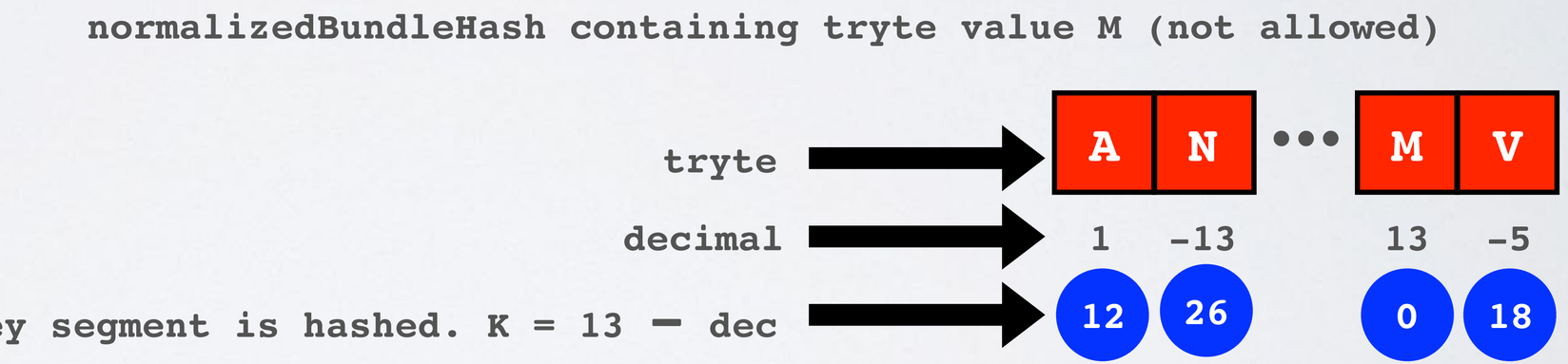
- A normalizedBundleHash will not have any tryte value M.

- During the creation of the bundleHash, the normalizedBundleHash was also calculated to check if it contained tryte value M. If the normalizedBundleHash had value M, the obsoleteTag in the first transactionObject of the bundle was updated and the bundleHash was recalculated. This process was repeated until the normalizedBundleHash did not contained tryte value M.

NORMALIZED BUNDLEHASH

- Question:
  Why not having tryte value M in the normalizedBundleHash?

- Answer:
  The normalizedBundleHash is used to create and validate the signature.
  During the creation of the signature (=signatureFragments), tryte value M will reveal a part of the private key.



- Note: If the answer is confusing skip this slide, watch IOTA tutorial 17 and watch this part of the video again.

NORMALIZED BUNDLEHASH

- To recap:
  - The bundle transactionObject addresses, values, obsoleteTags, timestamps, currentIndexes and lastIndexes are used to calculate the bundelHash.
  - This bundelHash is added for each transactionObject in the bundle.
  - To create or validate a signature the normalizedBundleHash is needed.
  - The normalizedBundleHash is created by extracting the bundleHash from the transactionObject and normalizing the bundleHash.
  - The normalizedBundleHash contains no tryte value M and the "weights" of the trytes are evenly distributed.