



# 早速問題紹介：Merkle-Hellman-knapsack 暗号

$n$  : ビット数

$$w = (w_0, w_1, \dots, w_n)$$

$$\beta_i = rw_i \mod q$$

$a_i$  = 平文の  $i$  ビット目が 1 なら 1, 0 なら 0

$$c = \sum a_i \beta_i$$

$\beta$  と  $c$  が与えられるので、そこから

$a$  を予測する  $\Rightarrow$  平文を解読することができる

$a$  を総当たりすると  $2^n$  の計算量

## SVPに向けて

このように式変形をすると定数倍と足し算だけ（一次結合）の形にすることができる。

$$c = \sum a_i \beta_i$$

$$a_0 \beta_0 + a_1 \beta_1 - c = 0$$

一次結合の和が小さい時、格子の SVP という問題に落とせがち。

## SVPで解くことができるかもしれない問題

$$c = \sum a_i \beta_i$$

$$a_0 \beta_0 + a_1 \beta_1 - c = 0$$

$$x_0 \beta_0 + x_1 \beta_1 + x_2 c$$

「格子」を使って、↑を小さくする  $x_0, x_1, x_2$  を求める。

小さくしようと頑張ると自然に 0 になる。

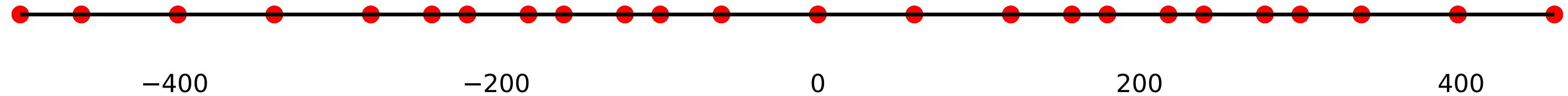
# SVPに向けて

$$x_0\beta_0 + x_1\beta_1 + x_2c$$

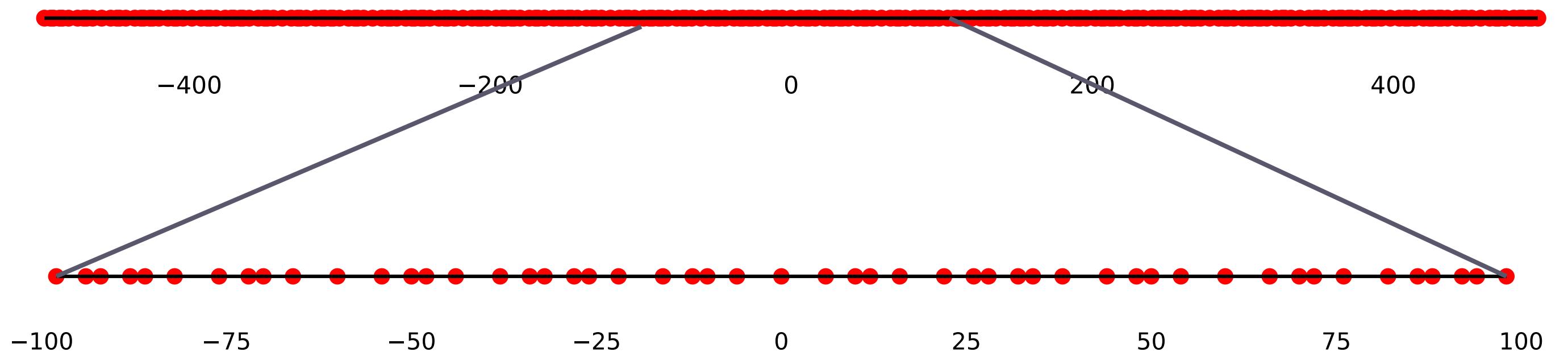
$$\beta_0 = 338, \beta_1 = 398, c = 398$$

# SVPに向けて

$x_0, x_1, x_2$  が -5~5 の点をプロット



$x_0, x_1, x_2$  が -50~50 の点をプロット



より広い範囲でプロットすると、より小さな値が見つかりやすくなる

## 自明に小さくできる例

$$x_0\beta_0 + x_1\beta_1 + x_2c$$

$$\beta_0 = 338, \beta_1 = 398, c = 398$$

$$x_0 = c = 398,$$

$$x_1 = 0,$$

$$c = -\beta_0 = -338$$

$$c\beta_0 - \beta_0c = 0$$

ただ小さくするだけなら実は簡単。

## 自明に小さくできる例

$$x_0\beta_0 + x_1\beta_1 + x_2c$$

$$\beta_0 = 338, \beta_1 = 398, c = 398$$

$$x_0 = c = 398,$$

$$x_1 = 0,$$

$$c = -\beta_0 = -338$$

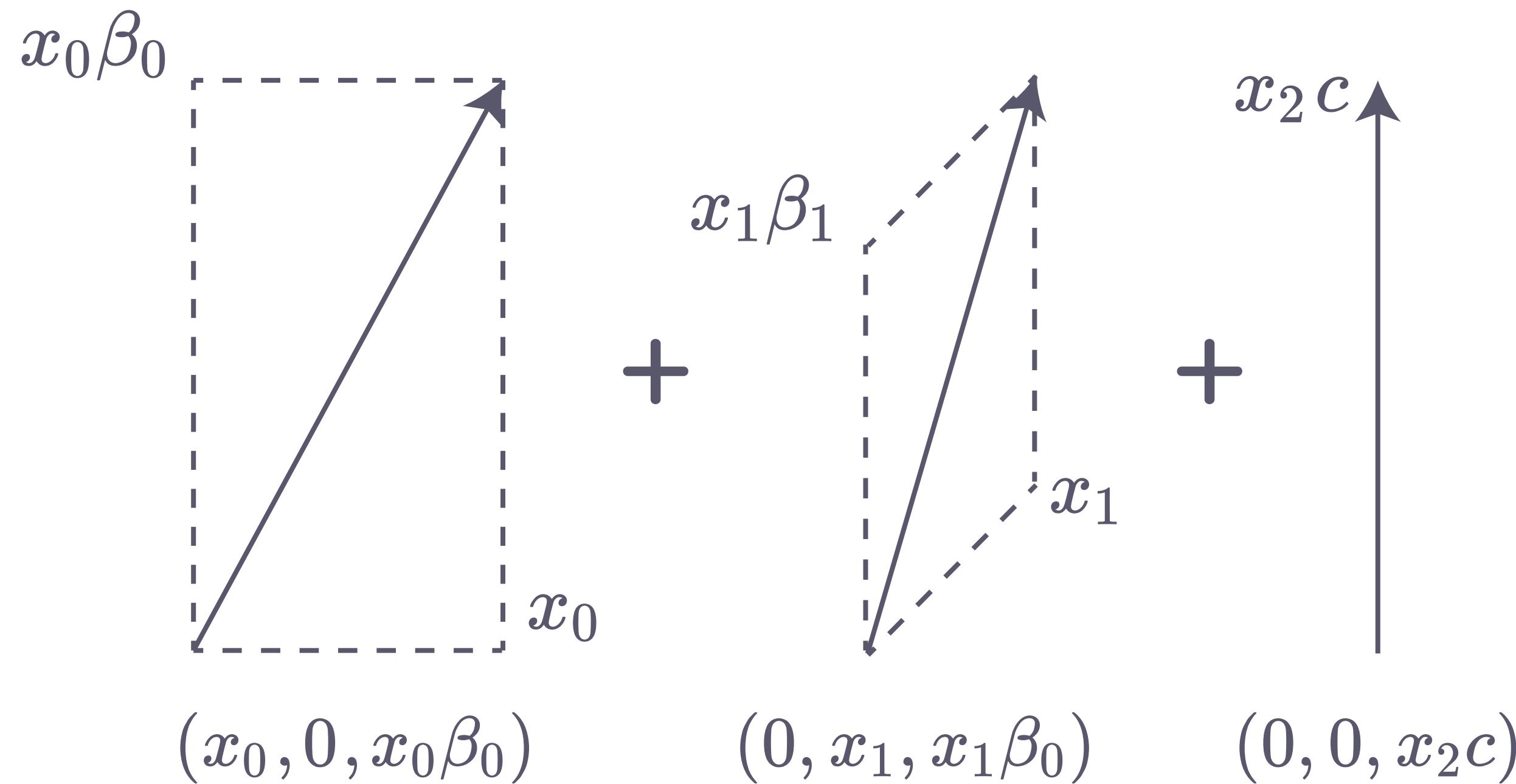
$$c\beta_0 - \beta_0c = 0$$

$x_0, x_1$  も、↑ の式の値も全部小さくする

# SVPに向けて

$(x_0, 0, x_0\beta_0),$   
 $(0, x_1, x_1\beta_1),$   
 $(0, 0, x_2c)$

の3本のベクトルの和を最小化する  $x_0, x_1, x_2$  を  
見つけようとする。



## SVPに向けて

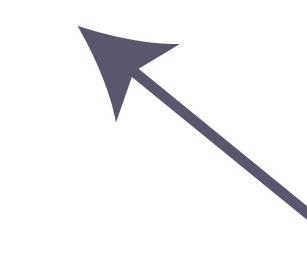
$$x_0\beta_0 + x_1\beta_1 + x_2c$$

$$\beta_0 = 338, \beta_1 = 398, c = 398$$

  $x_0=0, x_1=1, x_2=-1$  とすると *Win!*  
 $(0,1,0)$

$x_0=398, x_1=0, x_2=-338$  とすると

$(398,0,0)$



ベクトルの和を最小化するので、  
こっちは優先順位が低い。

## SVPに向けて

$(x_0, 0, x_0\beta_0),$

$(0, x_1, x_1\beta_1),$  の和は

$(0, 0, x_2c)$

$$(x_0, x_1, x_2) \begin{pmatrix} 1 & 0 & \beta_0 \\ 0 & 1 & \beta_1 \\ 0 & 0 & c \end{pmatrix}$$

で書き換えることができる。

# SVPに向けて

この行列を与えて、 $x_0, x_1, x_2$  が整数限定で、

最小のベクトルを見つける問題：Shortest Vector Problem

$$(x_0, x_1, x_2) \begin{pmatrix} 1 & 0 & \beta_0 \\ 0 & 1 & \beta_1 \\ 0 & 0 & c \end{pmatrix}$$

で書き換えることができる。

# Shortest Vector を見つける方法

sagemath を使えば一瞬

```
beta = [338, 398]
cipher = 398

mat = [
[1,0,338],
[0,1,398],
[0,0,-(338+398)],
]

mat = matrix(ZZ, mat).LLL()
print(mat)
```

```
[*'-') < sage a.sage
[ 1  1  0]
[ 12 -12 16]
[ 7  -6 -22]
```

LLL() というやつを呼び出せば  
求めてくれる。

# Shortest Vectorを見つける方法

sagemath を使えば一瞬

$$x_0\beta_0 + x_1\beta_1 + x_2c$$

cは2つの和

$x_0=1 \quad x_1=1 \quad \text{計算結果} = 0$

```
beta = [338, 398]
cipher = 398

mat = [
[1,0,338],
[0,1,398],
[0,0,-(338+398)],
]

mat = matrix(ZZ, mat).LLL()
print(mat)
```

/home/kurenai/ctf\_le
(\*'-'< sage a.sage
[ 1 1 0]
[ 12 -12 16]
[ 7 -6 -22]

次元を増やしてみた問題があるので一緒に解いて  
見ましょう

## 問題 2

$$80x \mod 128$$



の計算結果が与えられます。

ただし、値に 2 程度の誤差に入る可能性  
があります。

## CVPに向けて

$$80x \mod 128$$

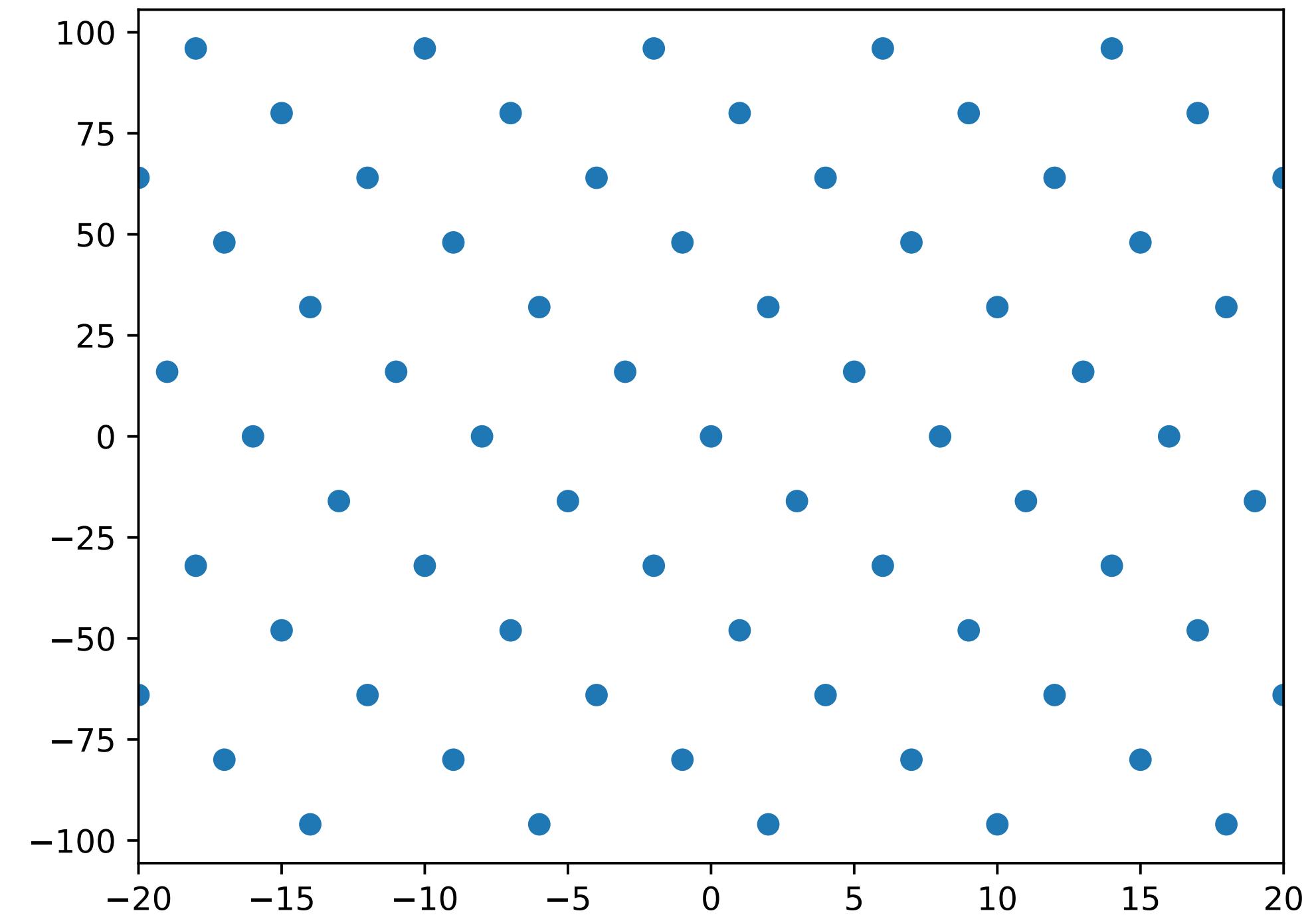


$$80x + 128m$$

計算結果が 16(誤差なし) の場合、  
格子でどうやって求めるか考えてみよう

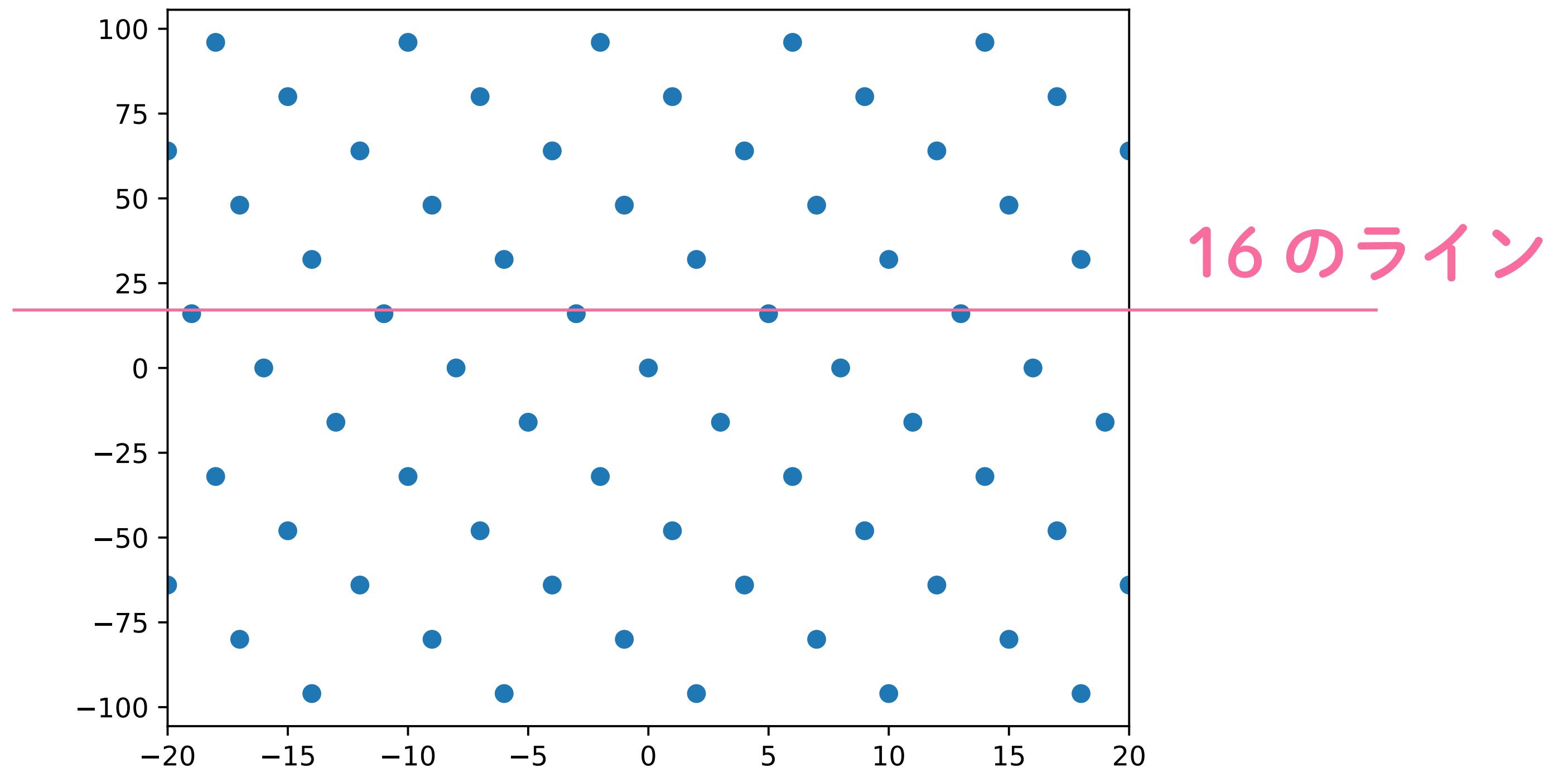
# CVPに向けて

$80x + 128m$  グラフにプロットしてみるとこんな感じ



# CVPに向けて

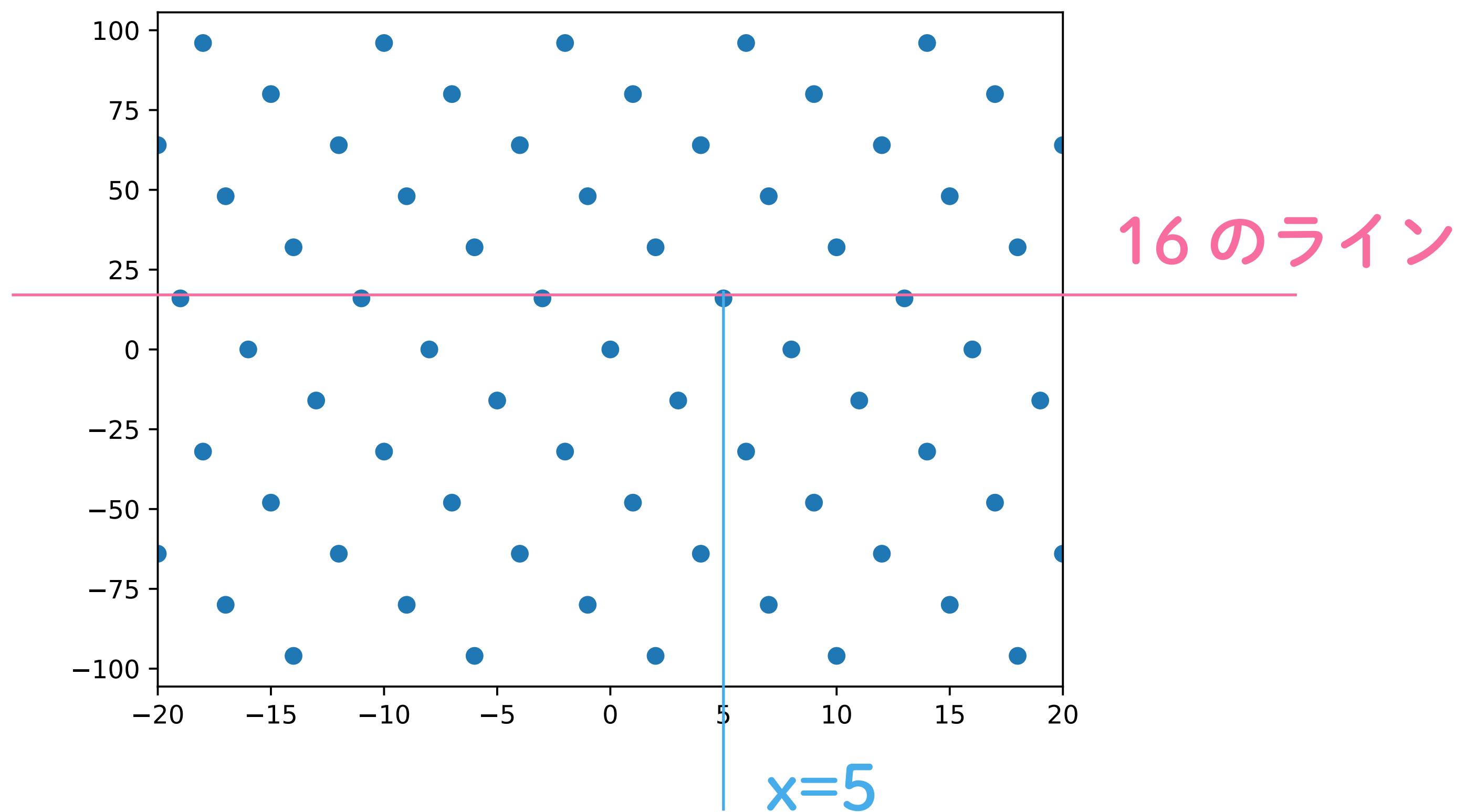
$80x + 128m$  が 16 ということは？



# CVPに向けて

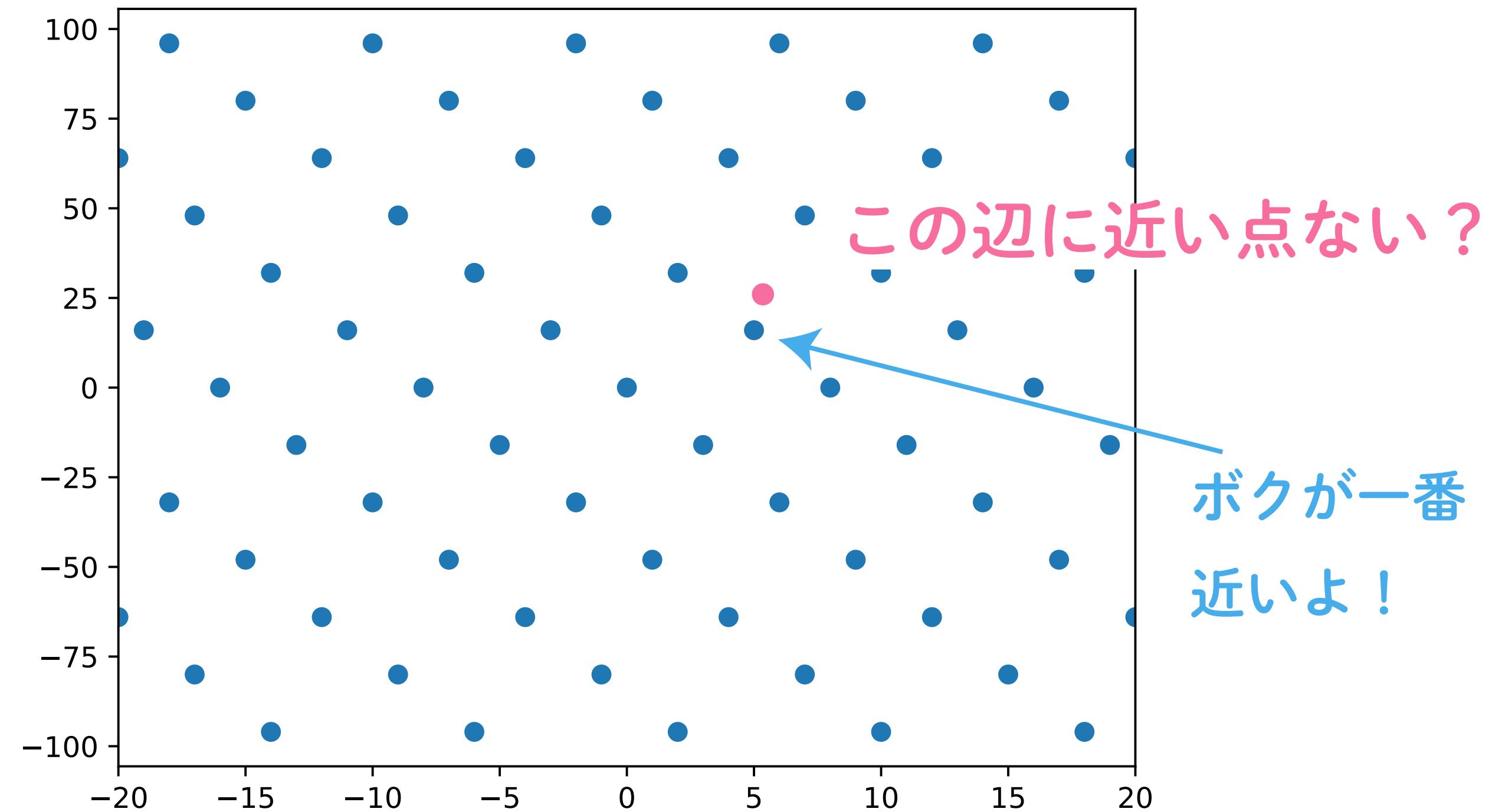
$80x + 128m$  が 16 ということは？

$$x=5$$



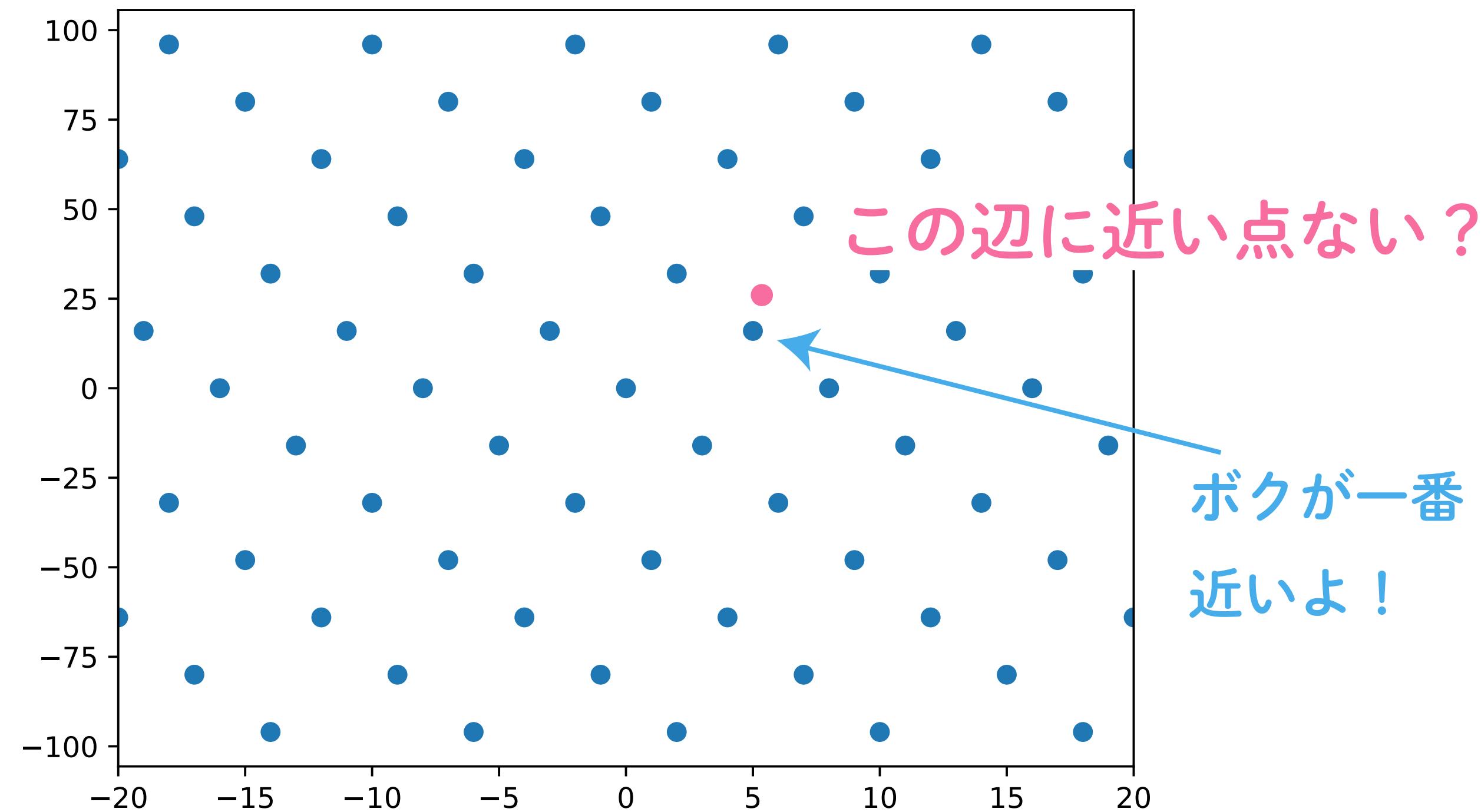
# CVP が得意な問題

CVP ではちょっとずれてる点から近い点を見つける問題！



# CVP が得意な問題

誤差から近い値を求める問題→CVP に応用できがち



## 格子の貼り方

今回は二次元なので、ベクトルが二本必要

$$80x + 128m$$

だから

$$x * (?, 80) + m * (?, 128)$$

の形にする。

$$\begin{pmatrix} ? & 80 \\ ? & 128 \end{pmatrix}$$

## 格子の貼り方

$$\begin{pmatrix} ? & 80 \\ ? & 128 \end{pmatrix}$$

格子を貼るために行列式が 0 以外でなければならぬので、? に適当な値を入れておく。

$$\begin{pmatrix} 1 & 80 \\ 0 & 128 \end{pmatrix}$$

## CVP の適応に向けて

$80x + 128m$  が 17 付近にある。 $x$  はなにか？

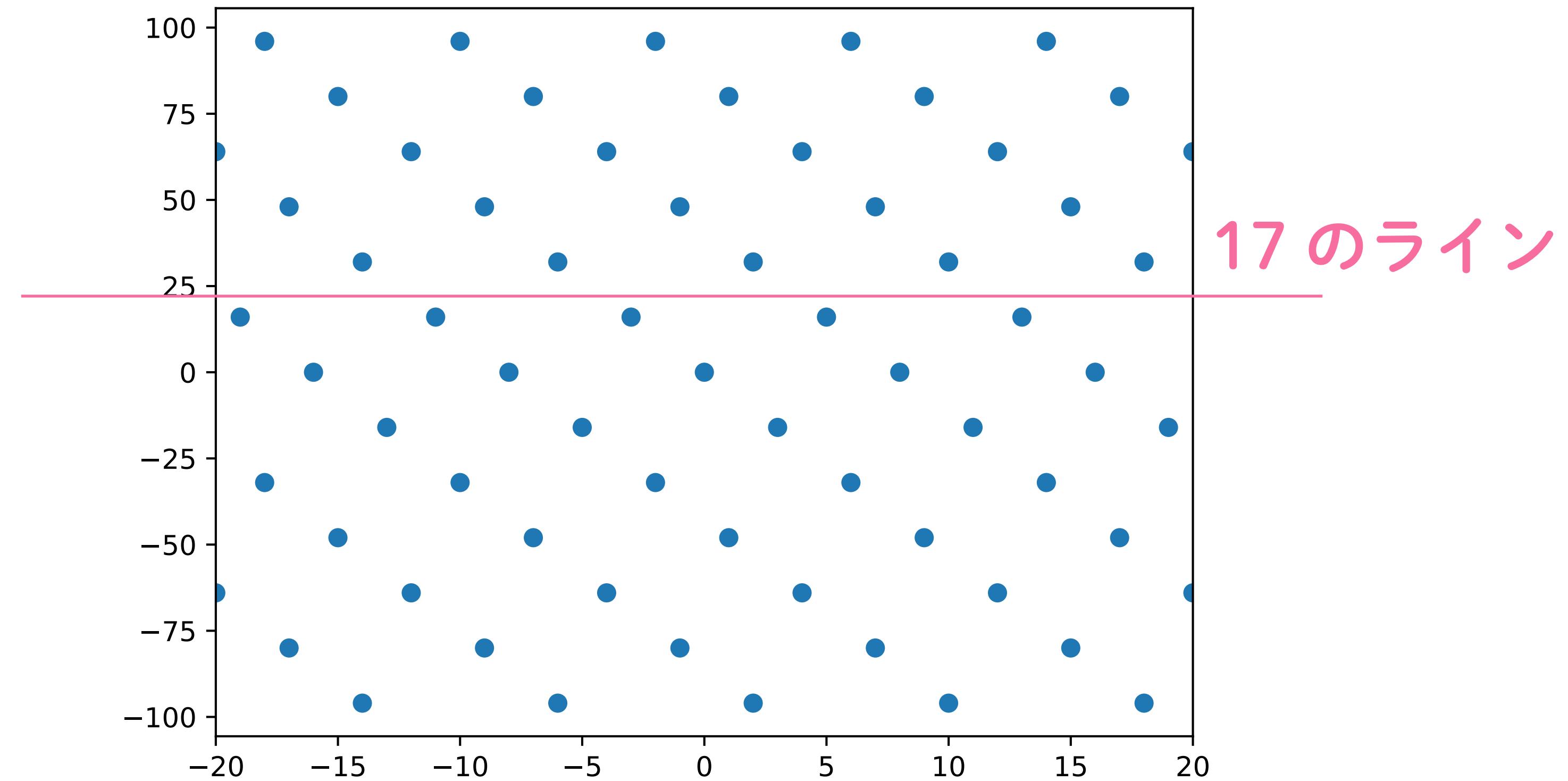
$$\begin{pmatrix} 1 & 80 \\ 0 & 128 \end{pmatrix}$$

(?, 17) に近い点を教えて！

…？ あれ？  $x$  の値がわからないぞ…？

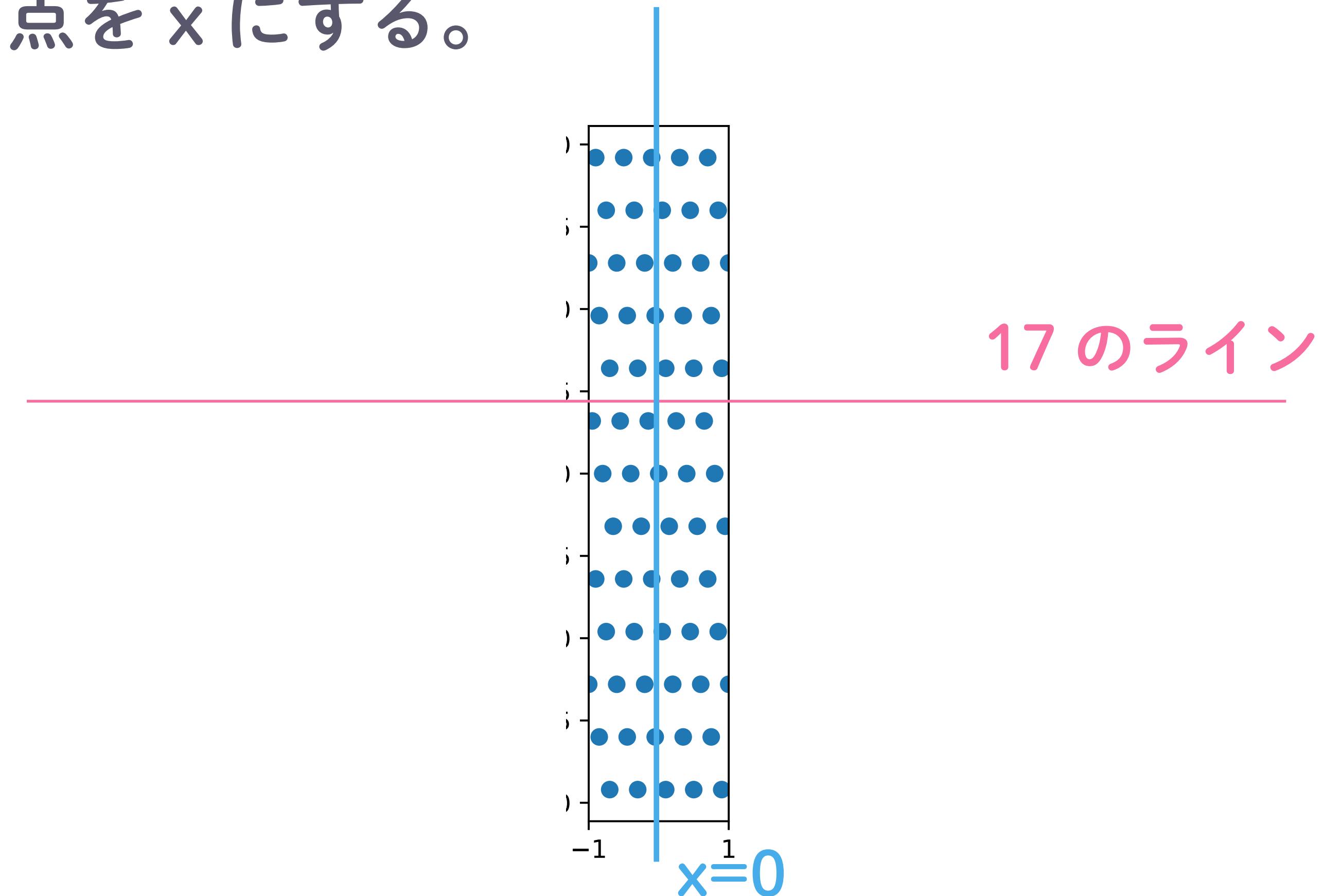
# CVP の適応に向けて

横に線は引けるけど、縦に線が引けなくて困った…



# 比重を変える

そういうときは横軸と縦軸の比重を変更して、  
適当な点を×にする。

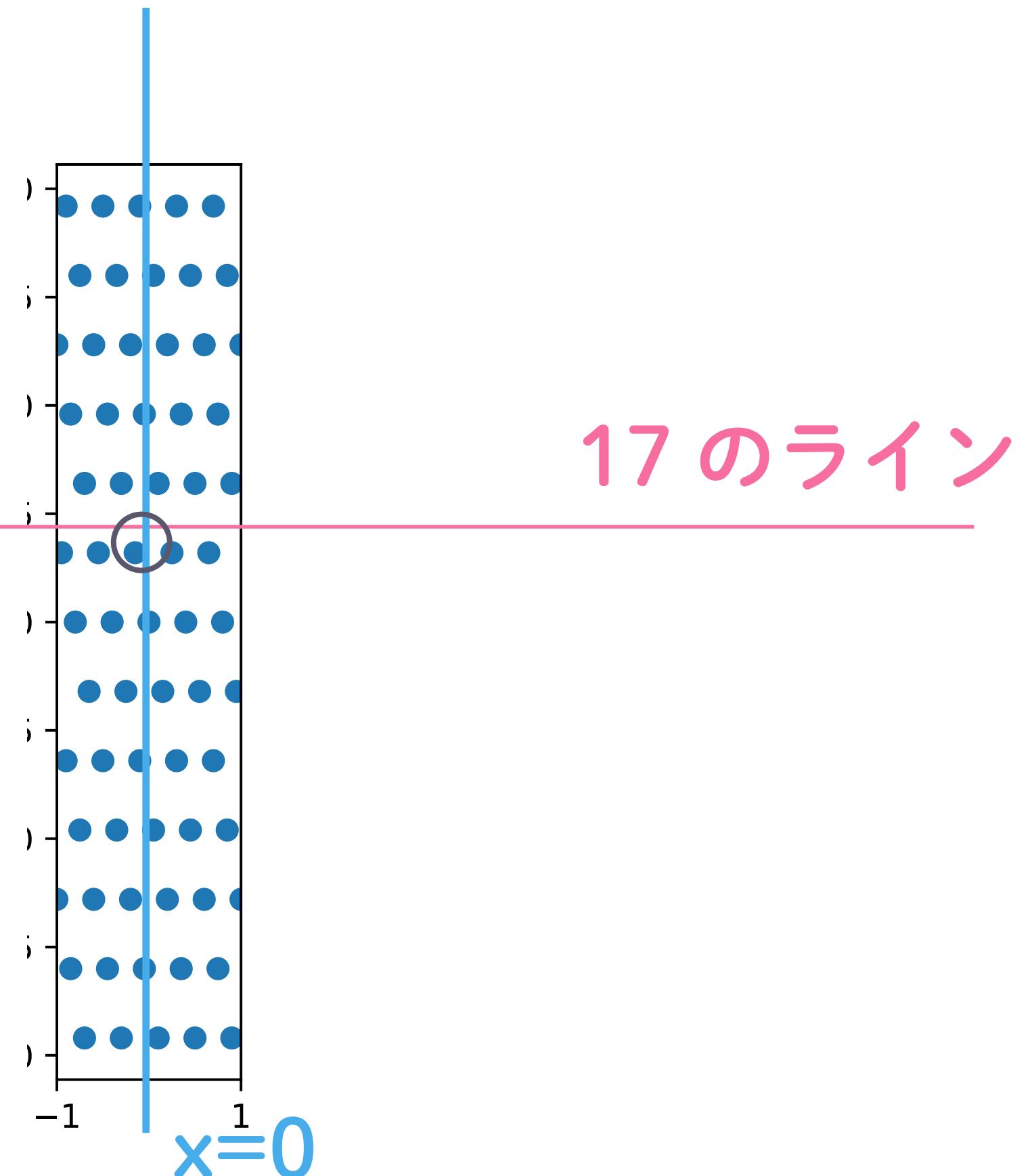


横軸を圧縮する

## 比重を変える

そういうときは横軸と縦軸の比重を変更して、  
適当な点を  $\times$  にする。

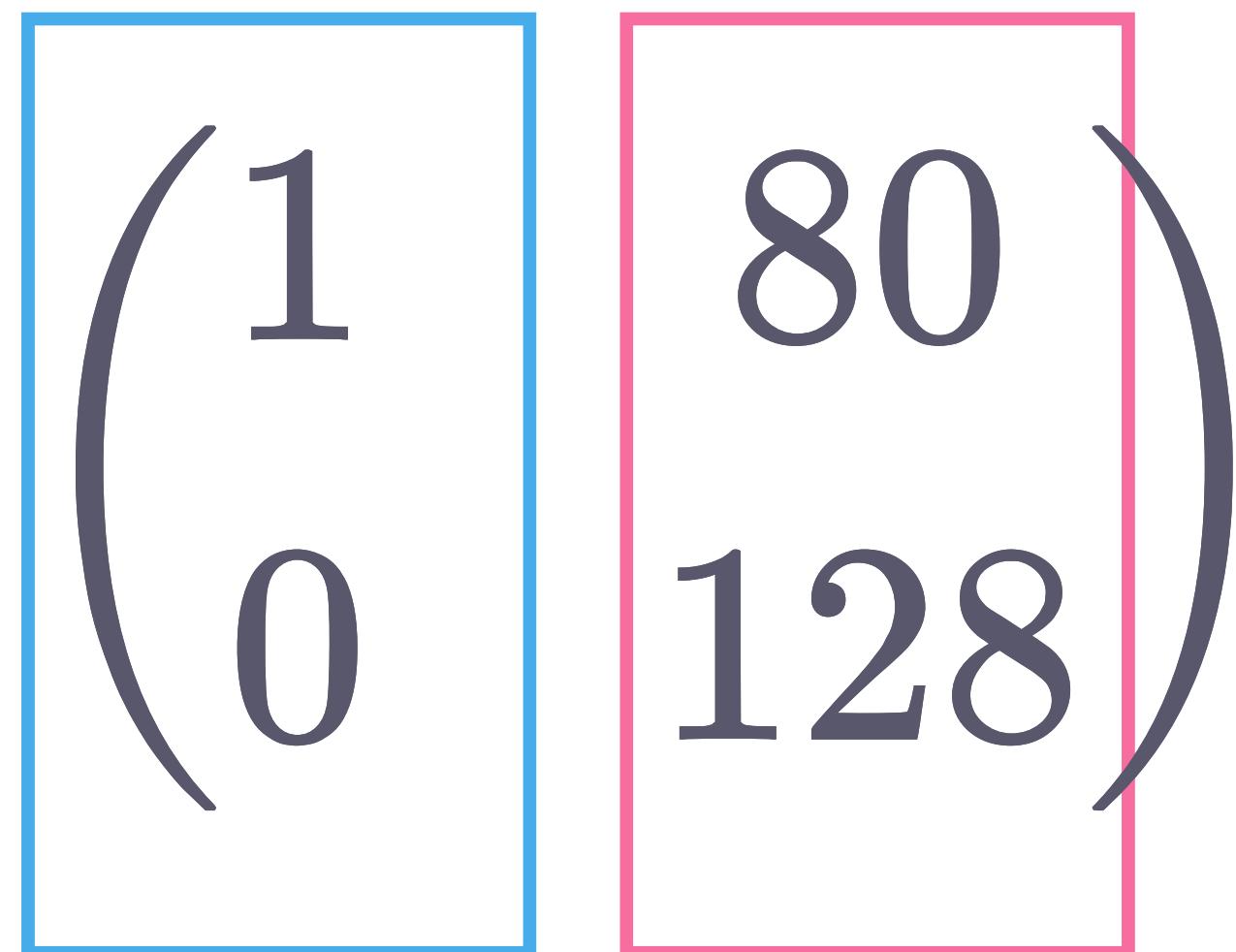
$\times$  も誤差になるので  
自然に 16 の点  
が見つかる。



横軸を圧縮する

# 比重を変える

$x$  の比重  $y$  の比重



# 比重を変える

x の比重

を下げる ↓ y の比重を上げる ↑

$$\left( \begin{array}{c} 1/r \\ 0 \end{array} \right)$$

$$\left( \begin{array}{c} 80 \times s \\ 128 \times s \end{array} \right)$$

## 実装方法

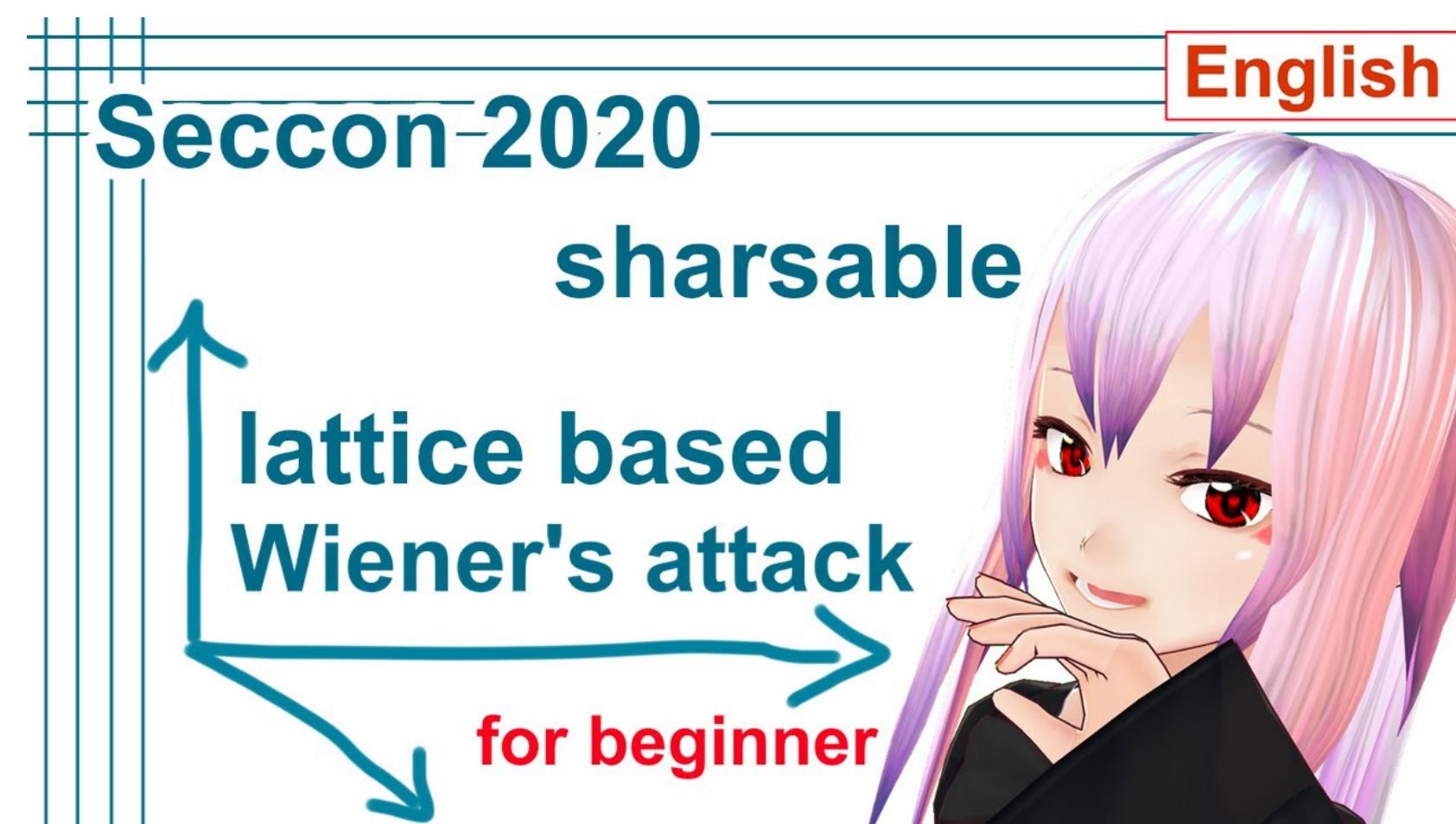
とてもかんたん。

LLL を実装して、babai' s Algorithm というものを適応するだけ。

```
mat = [
[1/8, 80],
[0, 128]
]
mat = Matrix(QQ, mat)
mat = mat.LLL()
target_vector = [0, 129]
print(babai_cvp(mat, target_vector))
```

## 比重を変えるときの目安

- ・乱暴に × の座標を小さくしそぎるのは良くない。
- ・適切な設定方法がある。
- ・↓ の動画参照



[English] How to use lattice in CTF? - SECCON 2020 sharsable writeup by Kurenaif

# LLL の問題の解き方

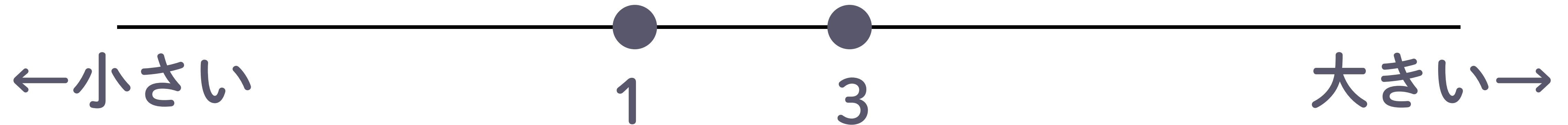
- ・いかにうまく格子を構築するかにかかる
- ・たくさん問題を見て感覚を養ってほしい
- ・詳しい話は今日の話から逸脱するので省略
- ・「LLL CTF」で検索！

## 小話

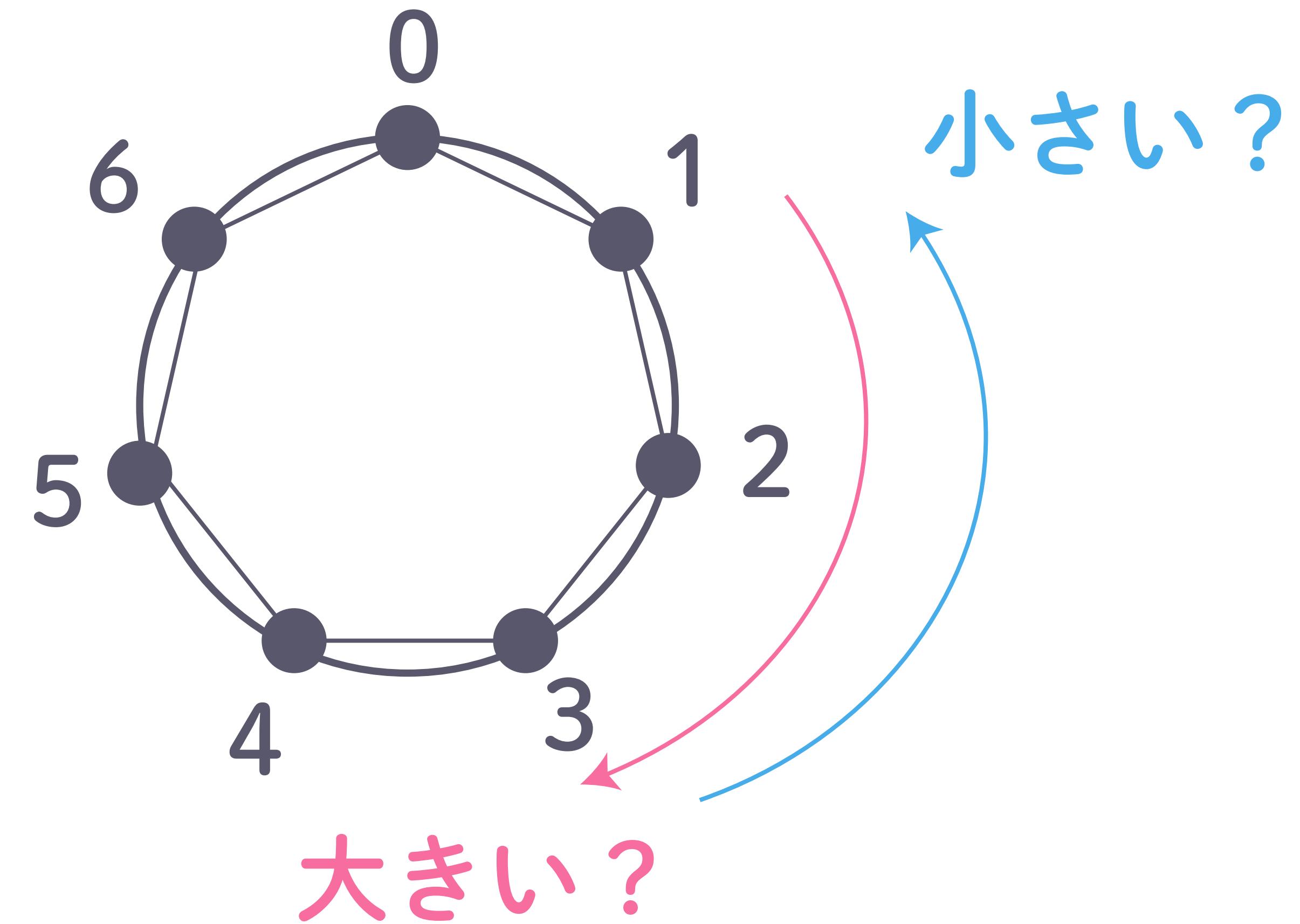
- ・直感的には誤差のある情報をたくさん集めると真の値が見えてくる気がする
- ・今から話すのは  
mod と誤差は恐ろしく相性が悪いという話

## 数直線の大小

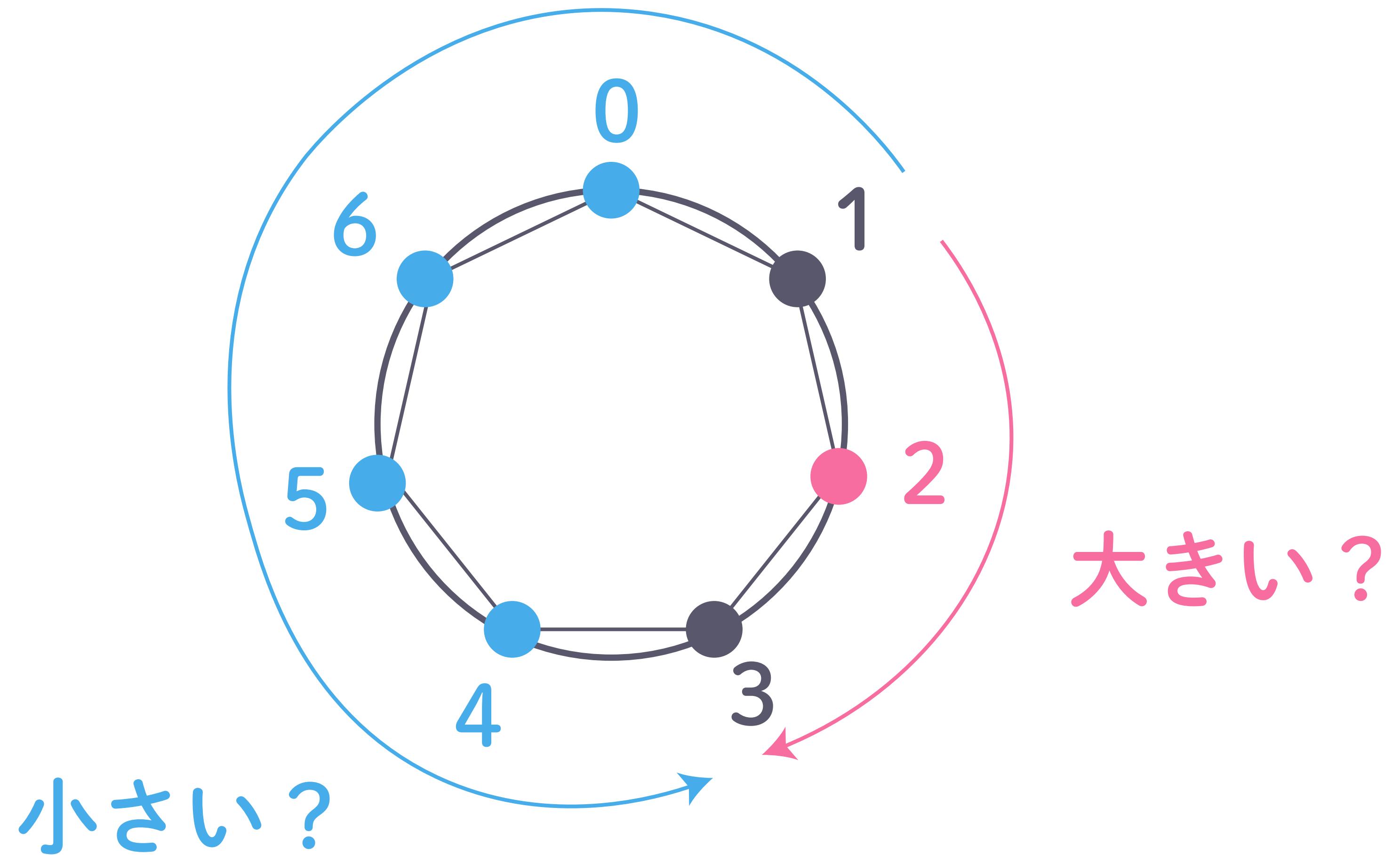
馴染みのある数直線：3は1より大きい。



3 と 1 はどちらが大きい？

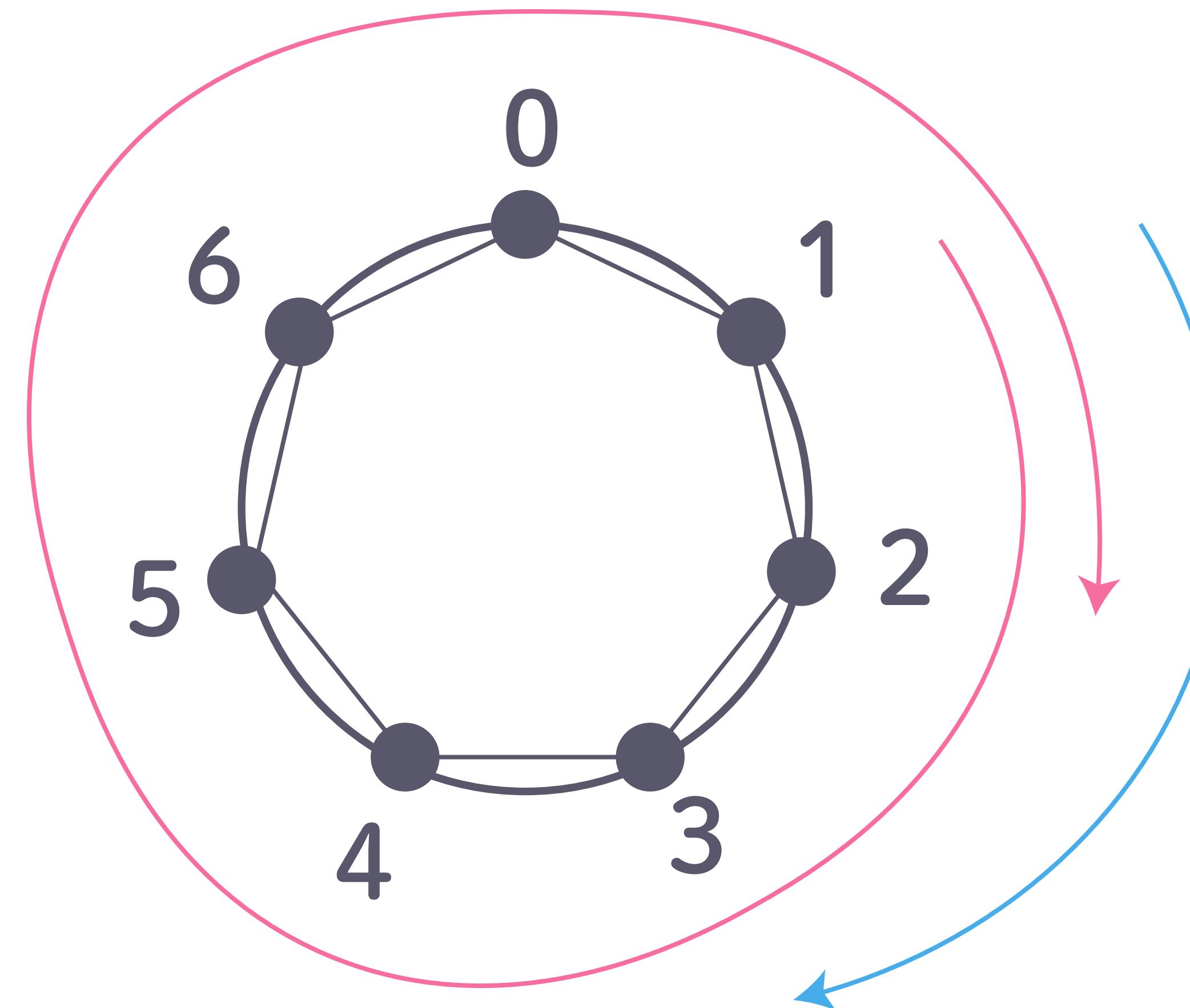


3 は 1 より大きいかもしれないし、小さいかもしれない



# mod の世界

$1 \rightarrow 2$  と  $1 \rightarrow 3$  はどちらが大きい？



- ・「大きいか小さいか」が存在しない世界
- ・「小さい」の定義が難しいので、mod をとった値の「誤差」はとても扱いが難しい。

# mod の世界から格子の世界へ

大小がない



大小がある

円の世界

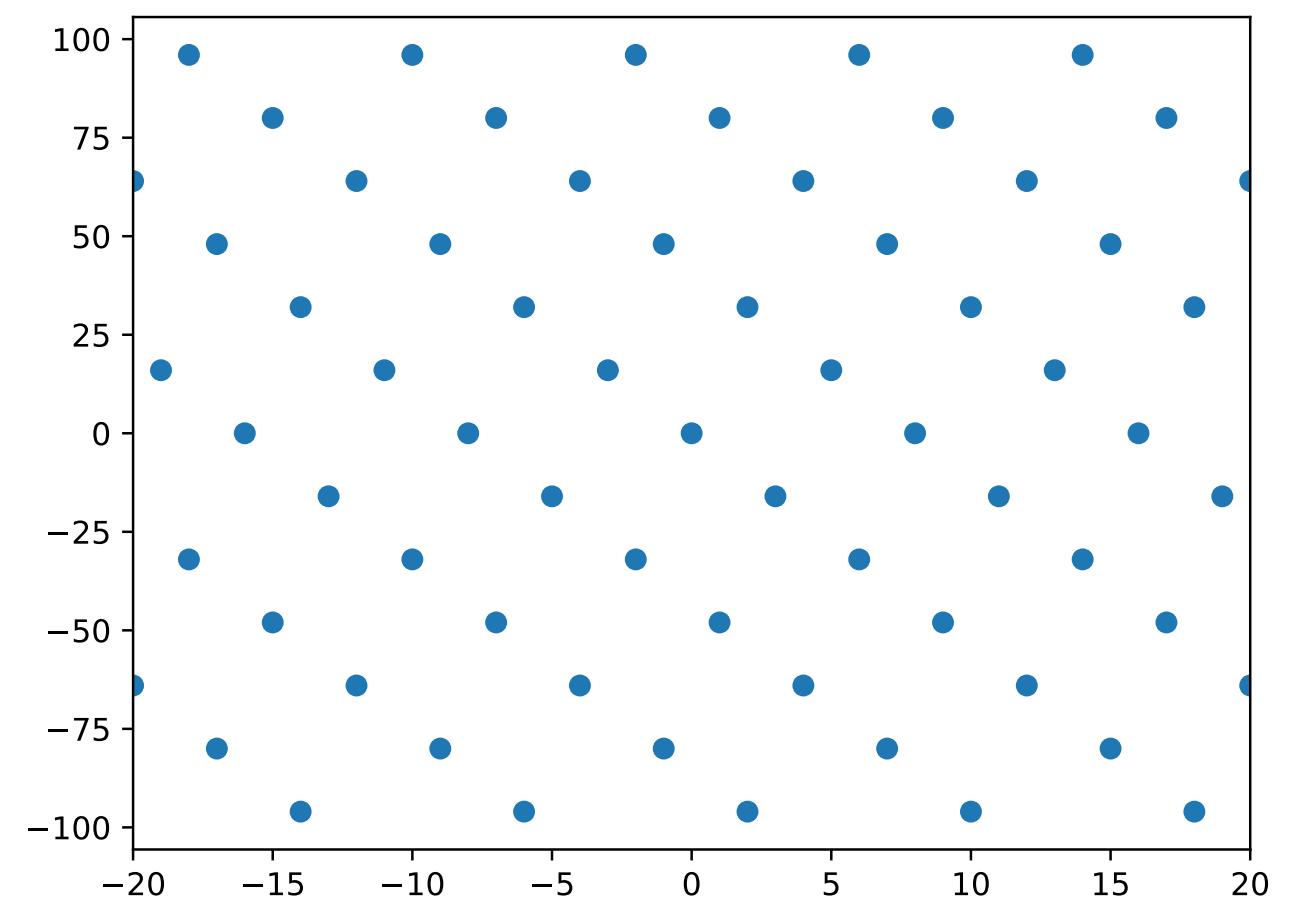
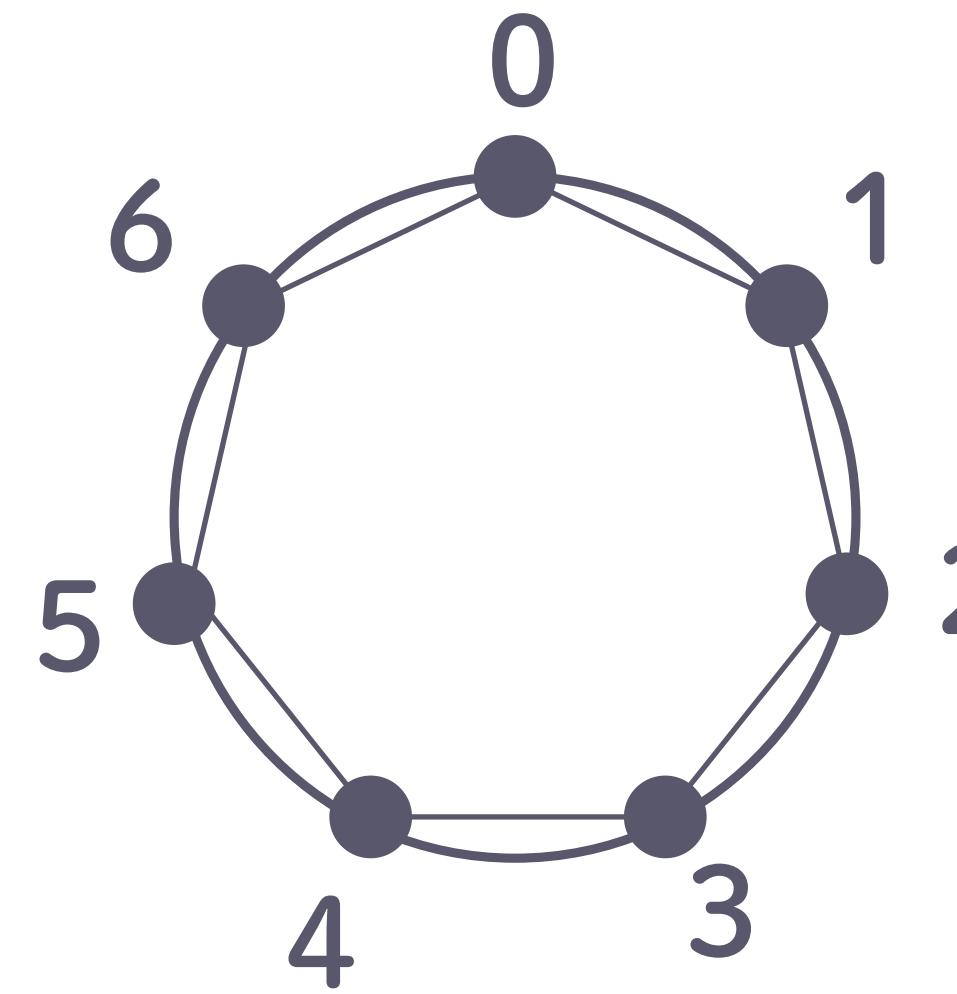


数直線の世界

格子と mod は

恐ろしく相性が良い

$$80x \mod 128 \rightarrow 80x + 128m$$



## 今日のまとめ

- LLL を使った問題の解き方 (SVP, CVP)
- 比重を変えて問題を工夫して解く方法
- mod と格子の話
- 格子の雰囲気を掴んでもくれたら嬉しい。もう少し色々問題を解いて感覺を掴んだら理論を勉強してほしい。