An Embarrassingly Simple 2^n-Time Algorithm for SVP—And How We Hope to Improve It

Divesh Aggarwal
Noah Stephens-Davidowitz

Game Plan

- Basics of sieving
- Embarrassingly simple algorithm: Sieving by averages
- Hope to simplify the proof of correctness
- Hope to make it faster

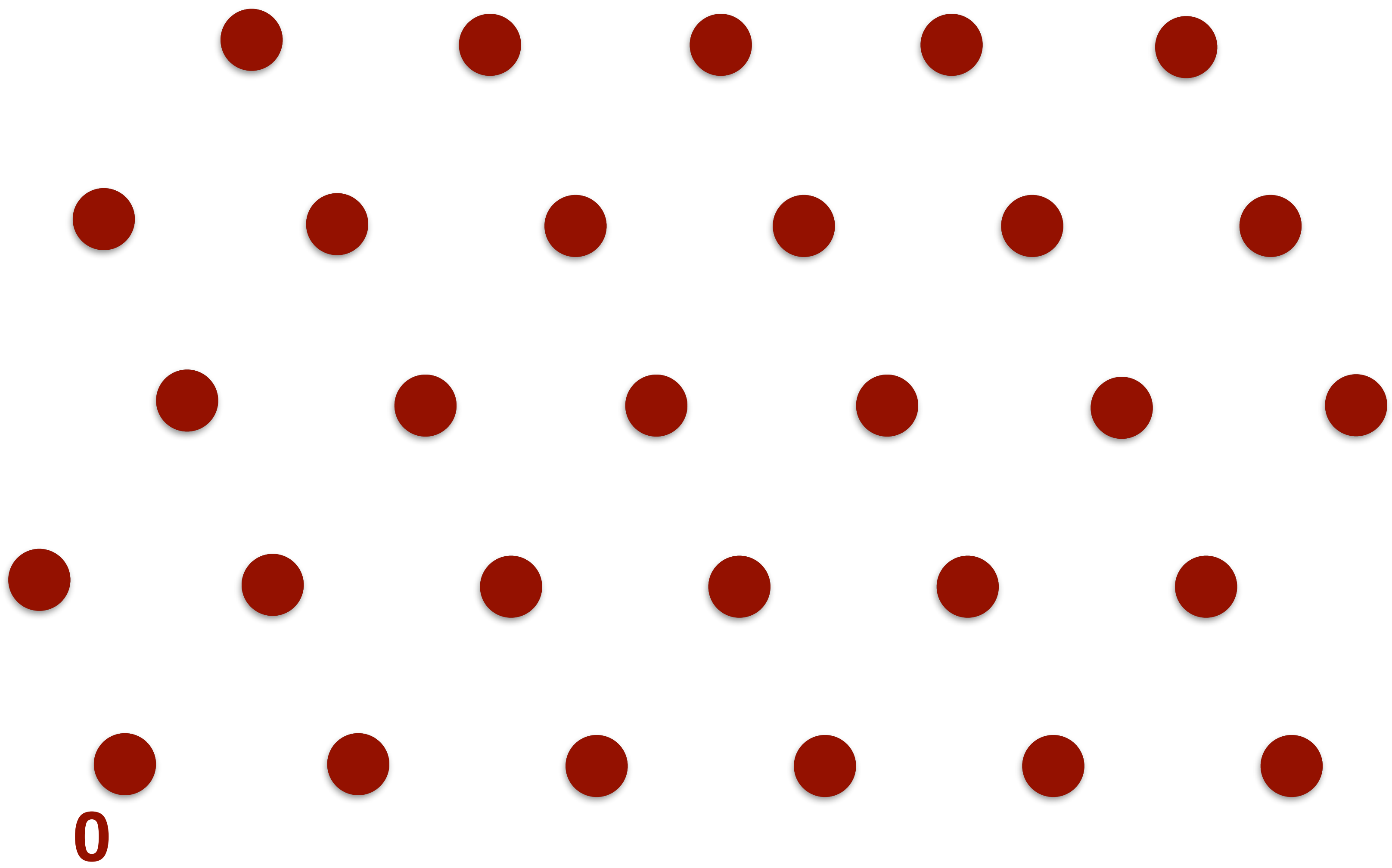
Lattices

Lattices

- $\mathcal{L}$ is a discrete set of vectors in $\mathbb{R}^n$

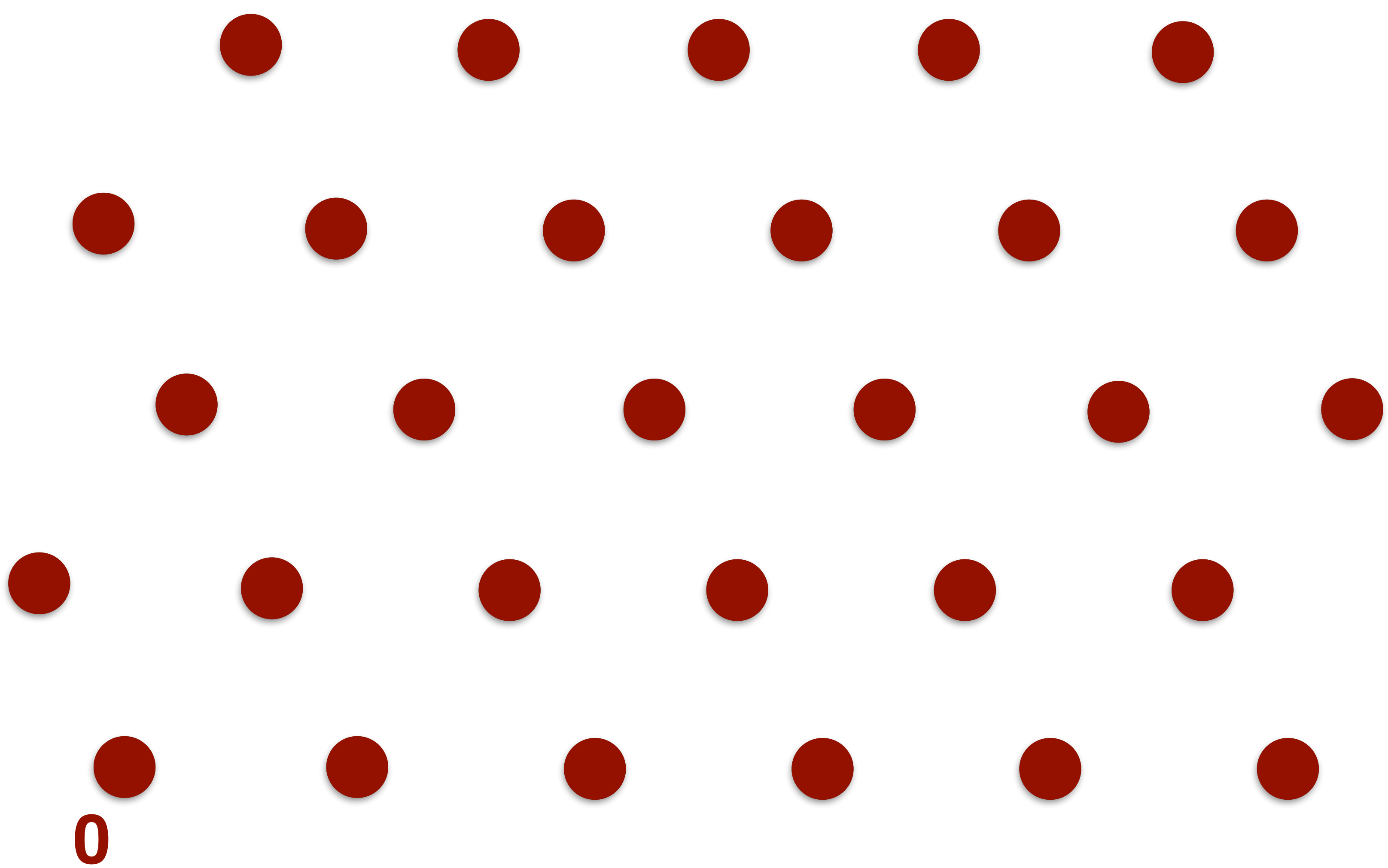
Lattices

- $\mathcal{L}$ is a discrete set of vectors in $\mathbb{R}^n$



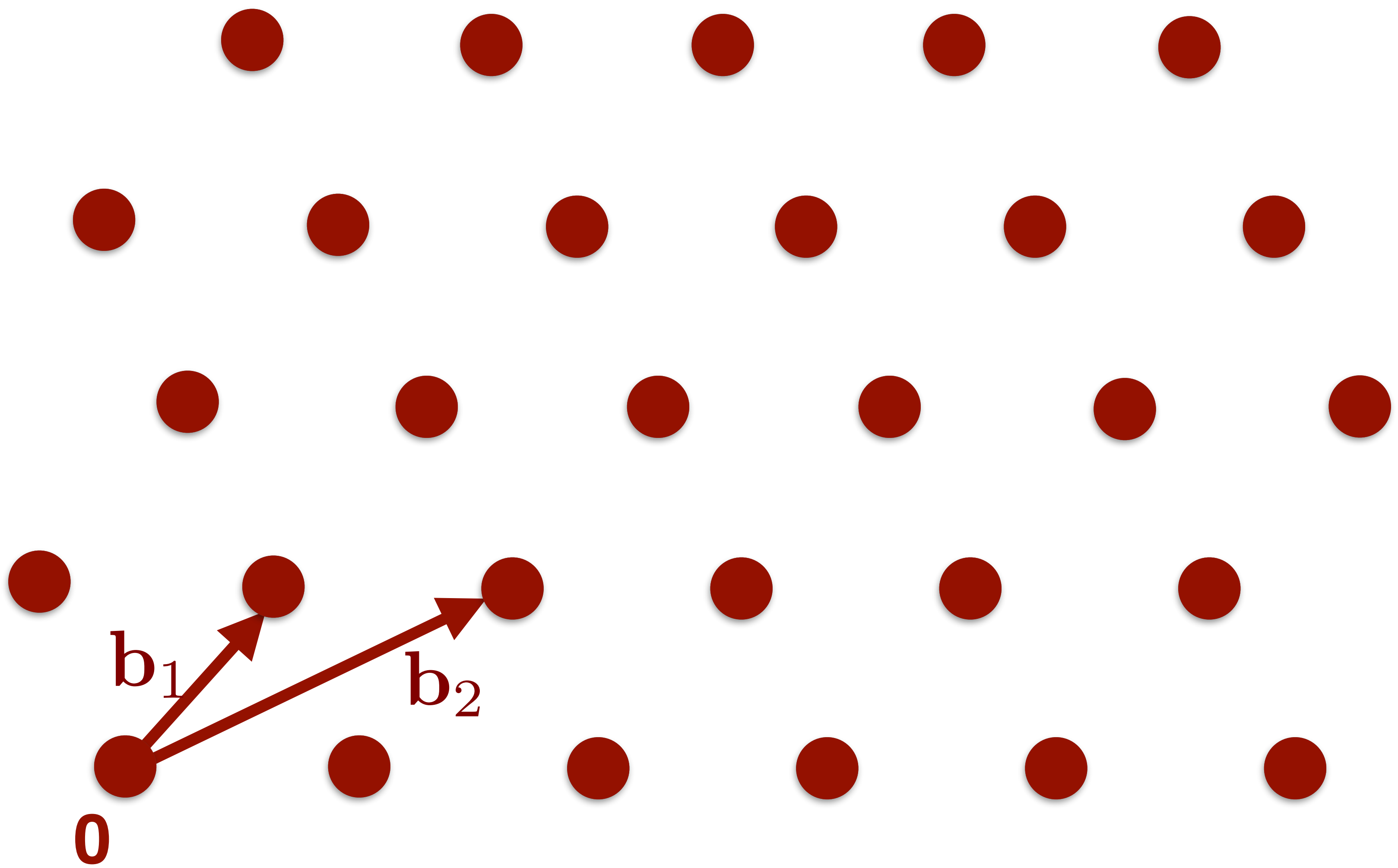
Lattices

- $\mathcal{L}$ is a discrete set of vectors in $\mathbb{R}^n$
- Specified by a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$, linearly independent vectors



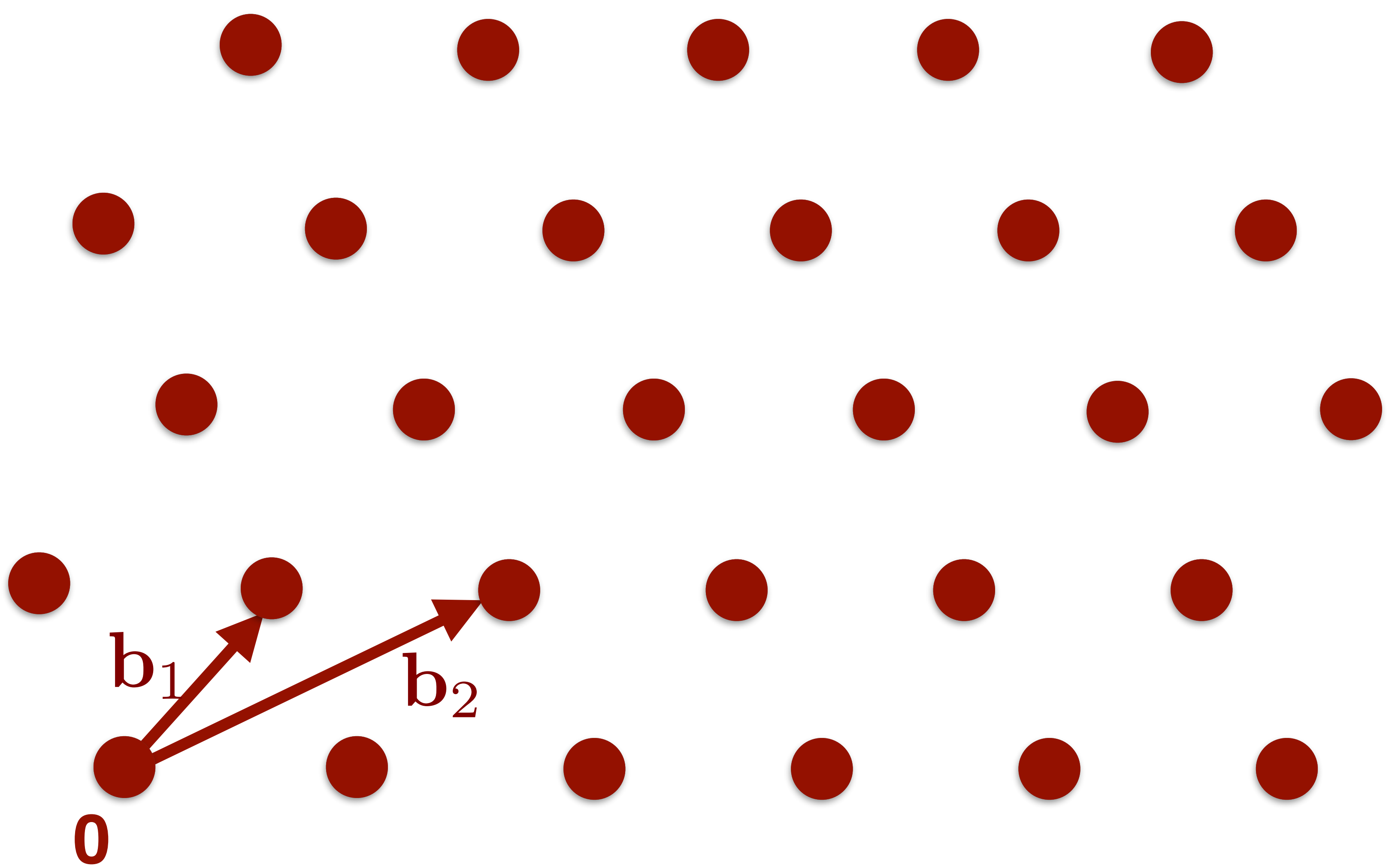
Lattices

- $\mathcal{L}$ is a discrete set of vectors in $\mathbb{R}^n$
- Specified by a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$, linearly independent vectors

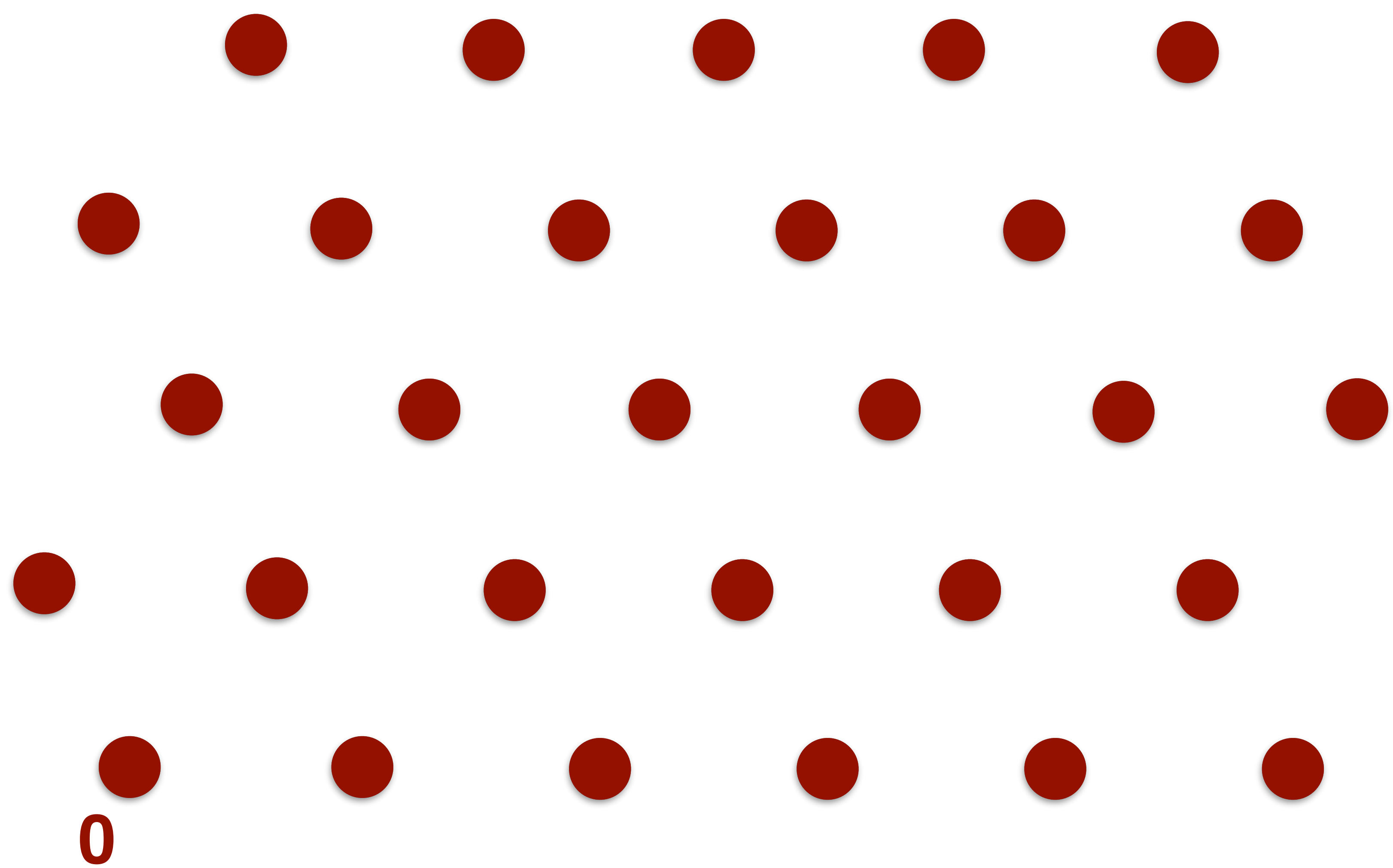


Lattices

- $\mathcal{L}$ is a discrete set of vectors in $\mathbb{R}^n$
- Specified by a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$, linearly independent vectors
- $\mathcal{L} = \{a_1\mathbf{b}_1 + \cdots + a_n\mathbf{b}_n \mid a_i \in \mathbb{Z}\}$

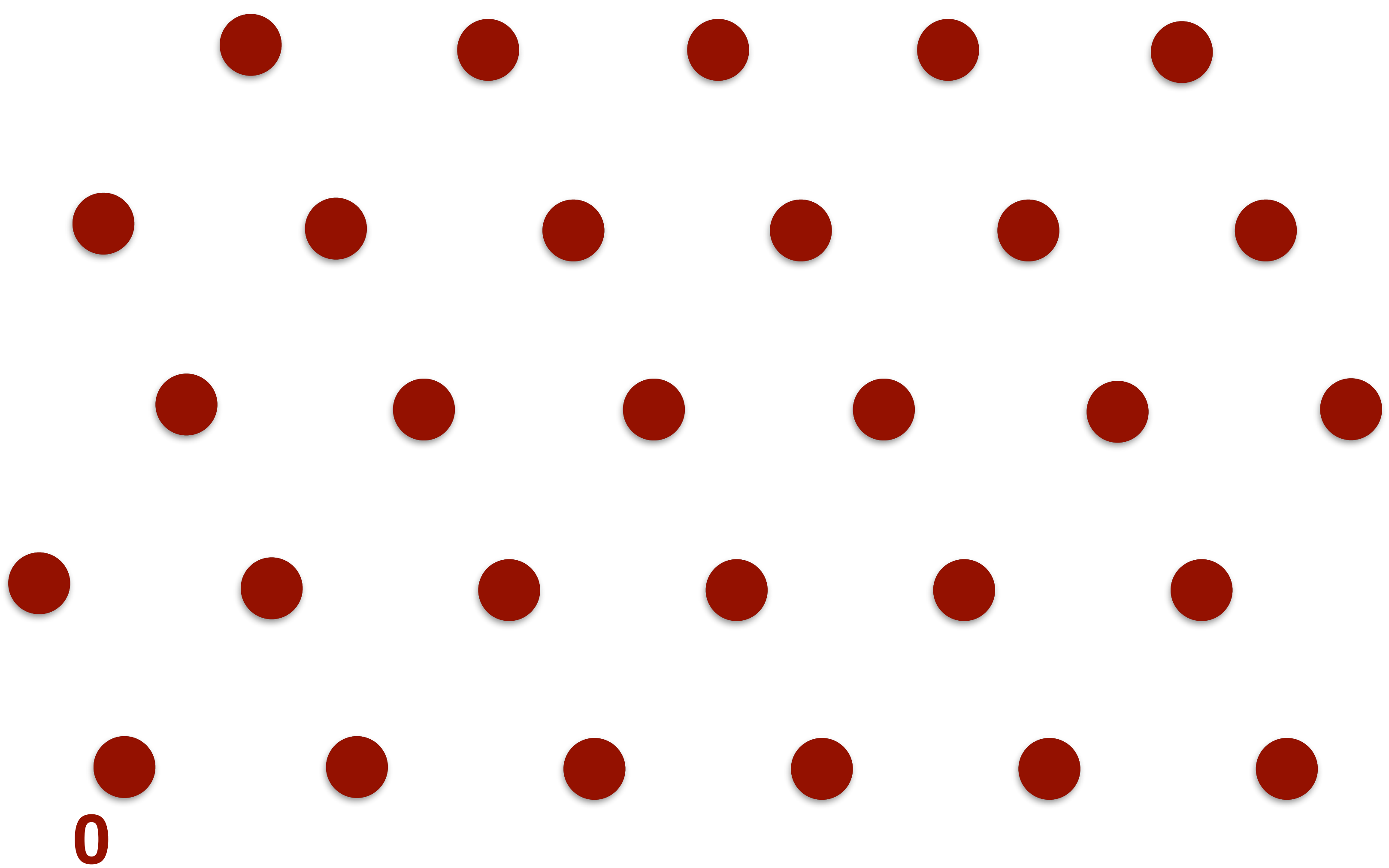


Shortest Vector Problem



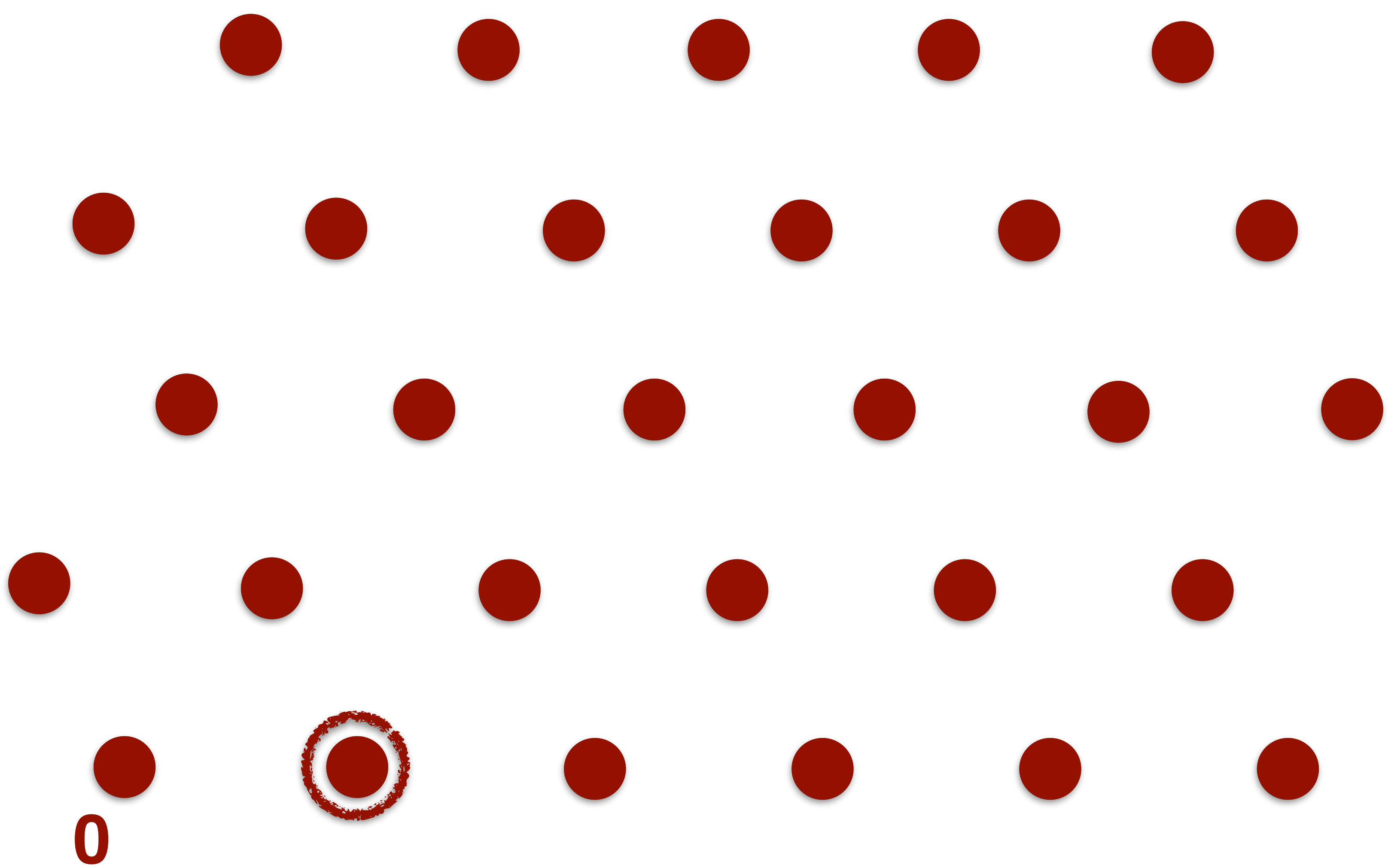
Shortest Vector Problem

- $\mathrm{SVP}(\mathcal{L}) =$ shortest non-zero $\mathbf{y} \in \mathcal{L}$



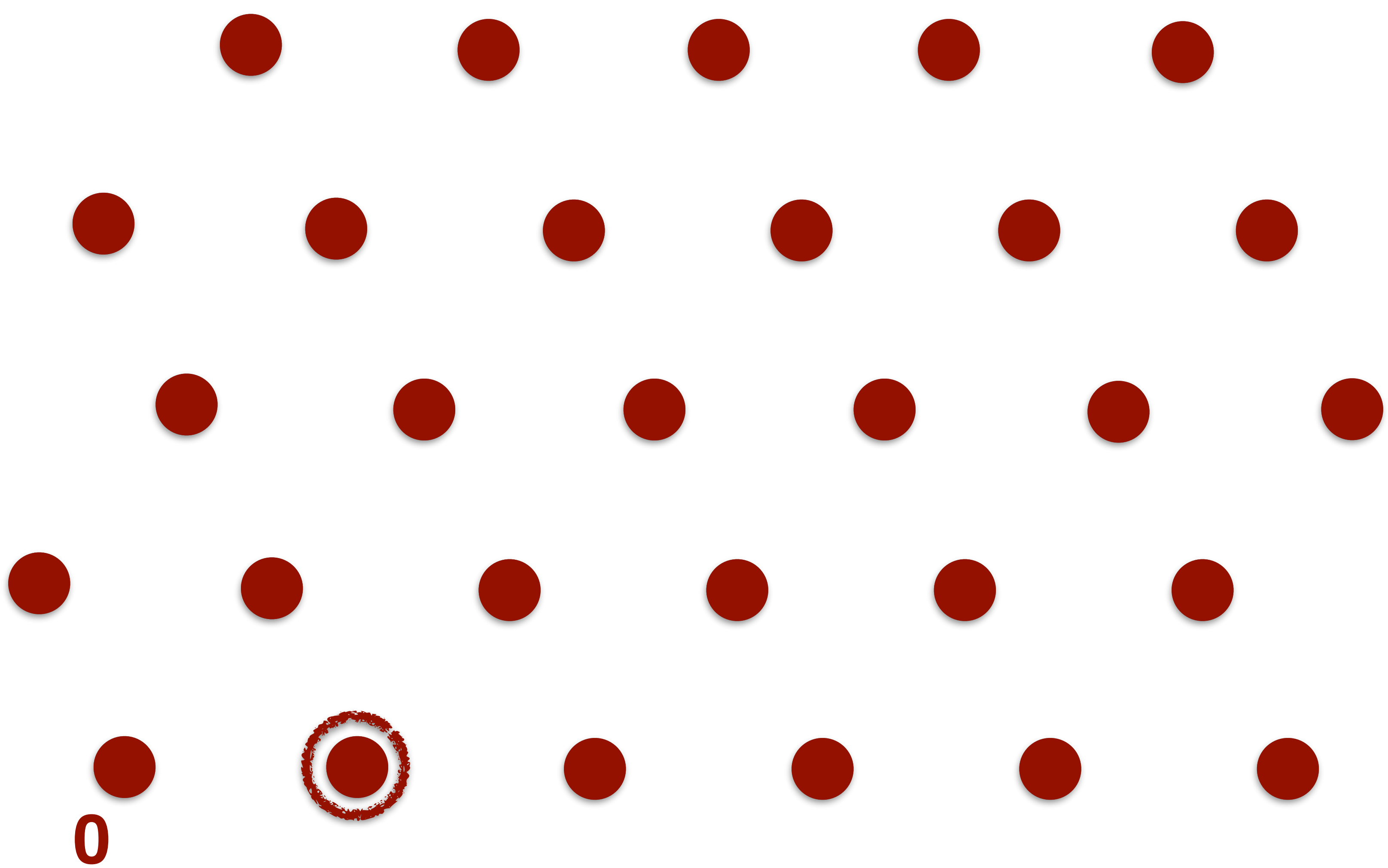
Shortest Vector Problem

- $\mathrm{SVP}(\mathcal{L}) =$ shortest non-zero $\mathbf{y} \in \mathcal{L}$



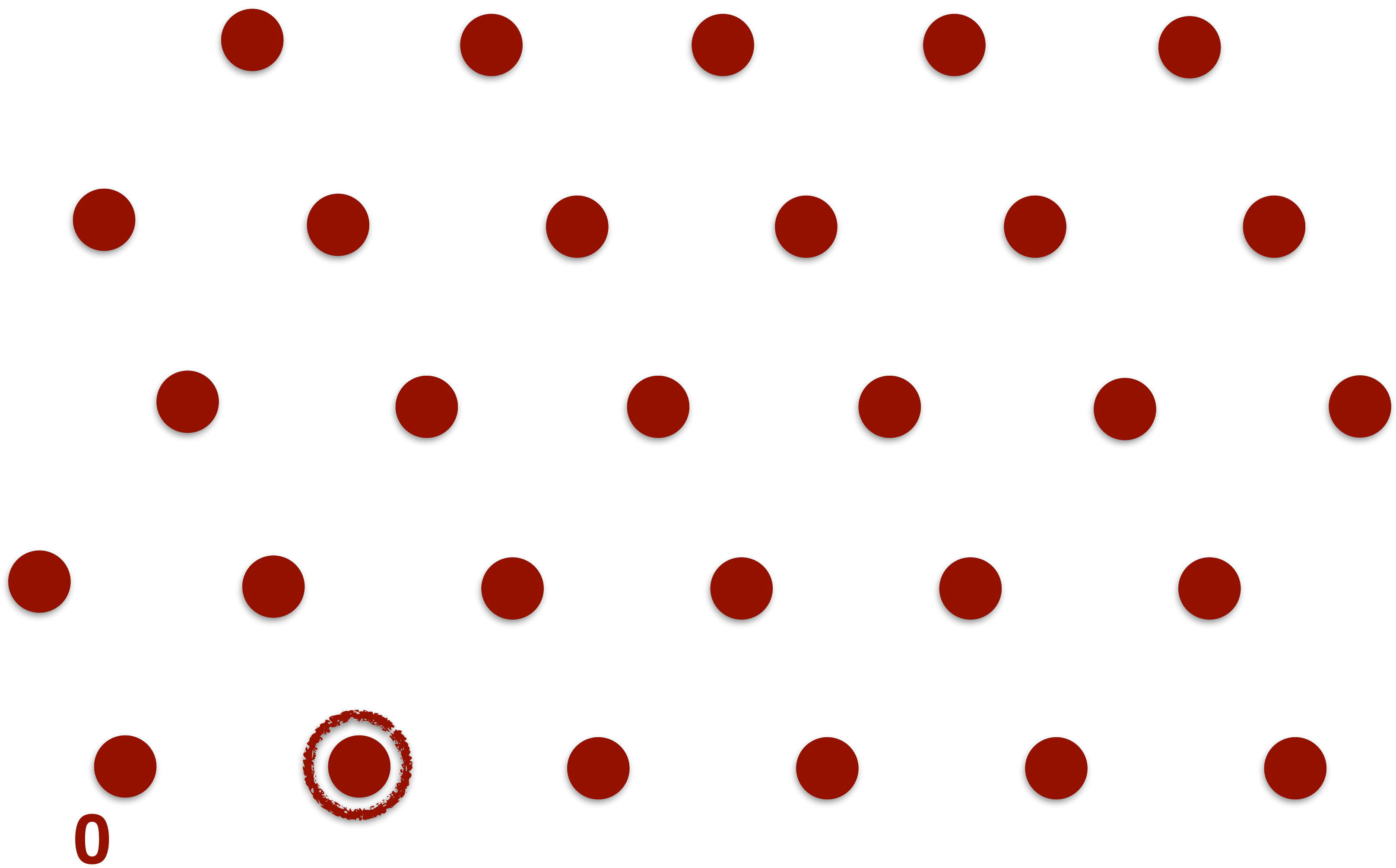
Shortest Vector Problem

- $\mathrm{SVP}(\mathcal{L}) =$ shortest non-zero $\mathbf{y} \in \mathcal{L}$
- NP-hard (even to approximate).



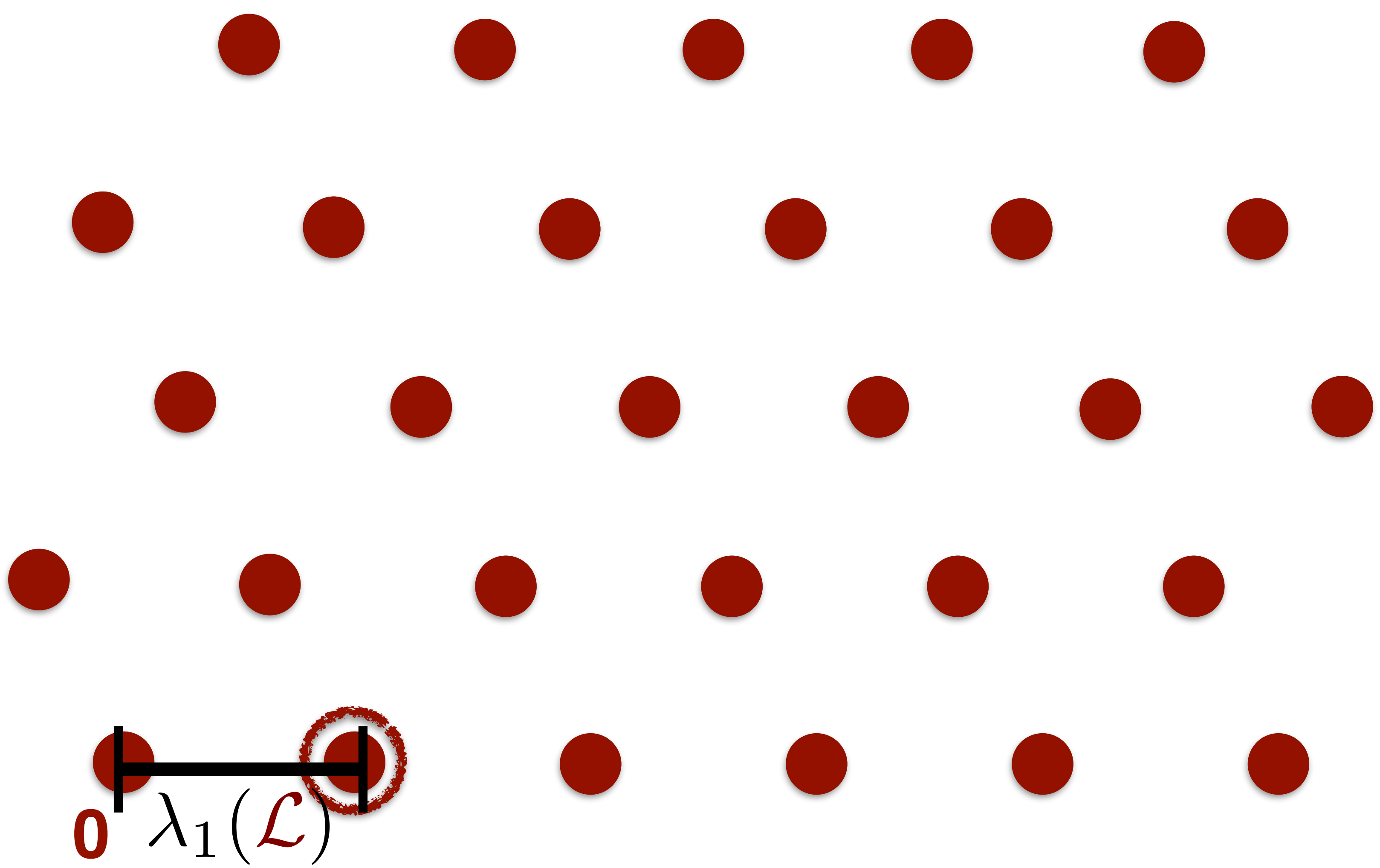
Shortest Vector Problem

- $\mathsf{SVP}(\mathcal{L}) =$ shortest non-zero $\mathbf{y} \in \mathcal{L}$
- NP-hard (even to approximate).
- $\lambda_1(\mathcal{L}) = ||\mathsf{SVP}(\mathcal{L})||$

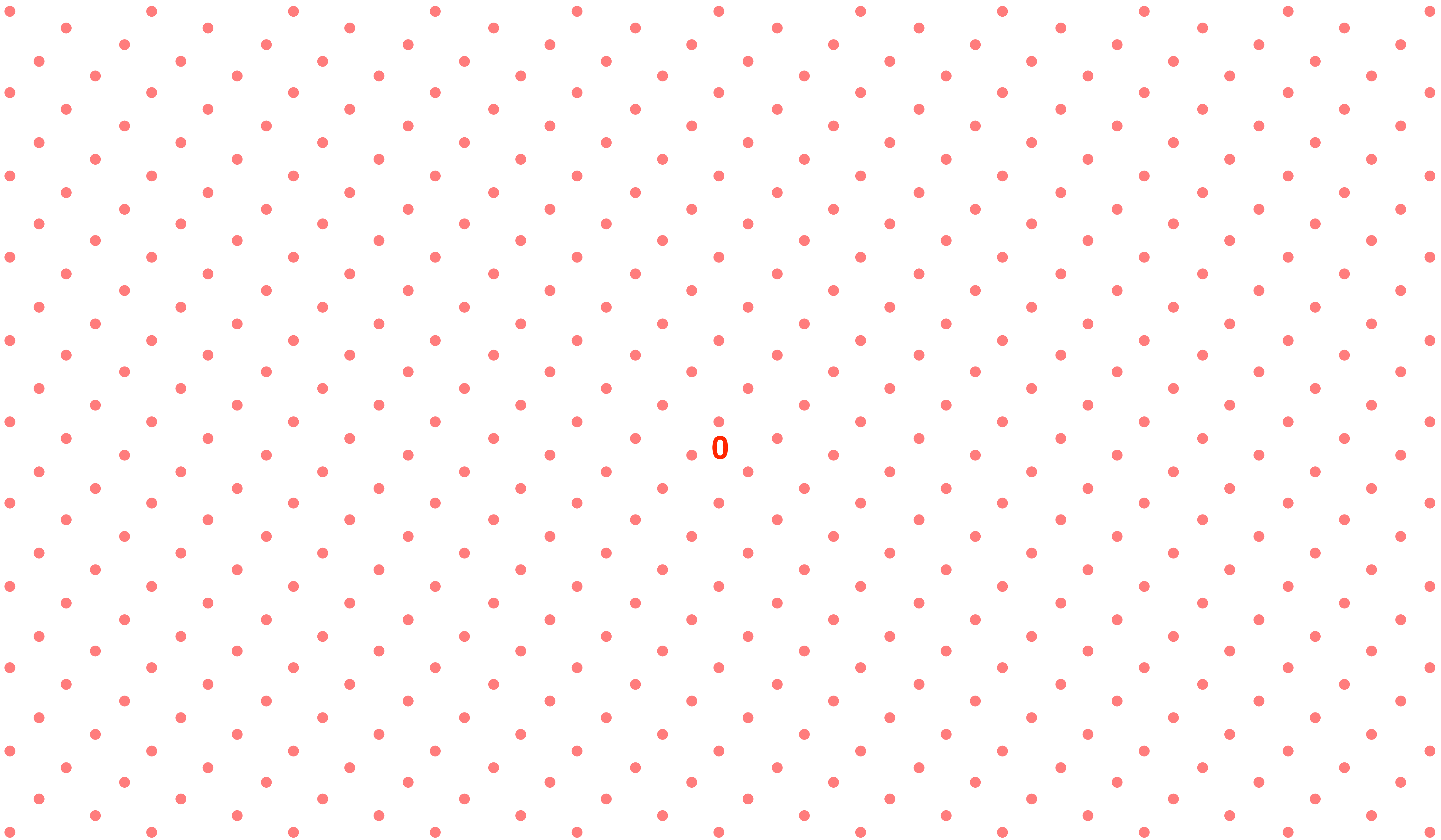


Shortest Vector Problem

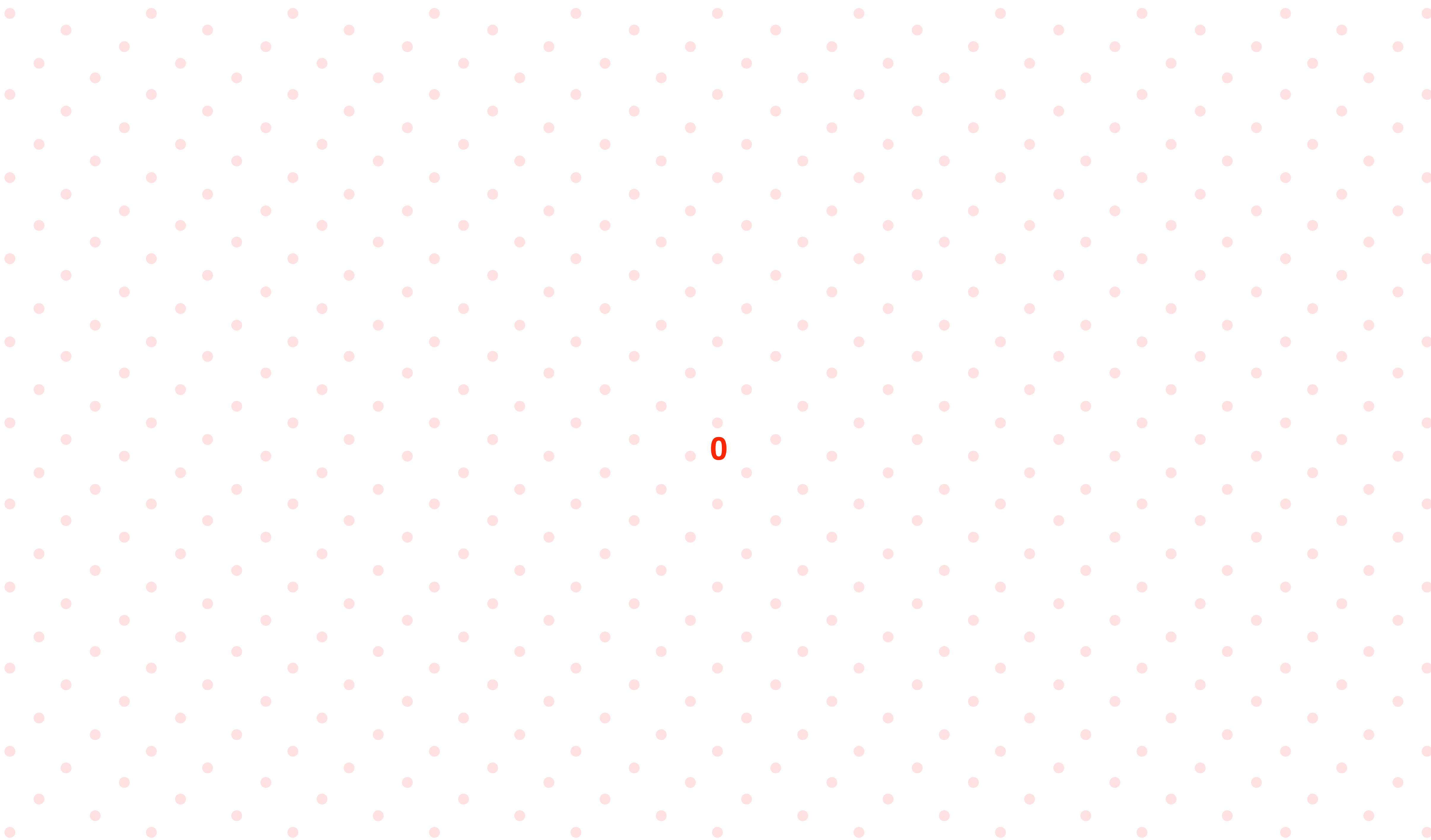
- $\mathsf{SVP}(\mathcal{L}) =$ shortest non-zero $\mathbf{y} \in \mathcal{L}$
- NP-hard (even to approximate).
- $\lambda_1(\mathcal{L}) = ||\mathsf{SVP}(\mathcal{L})||$



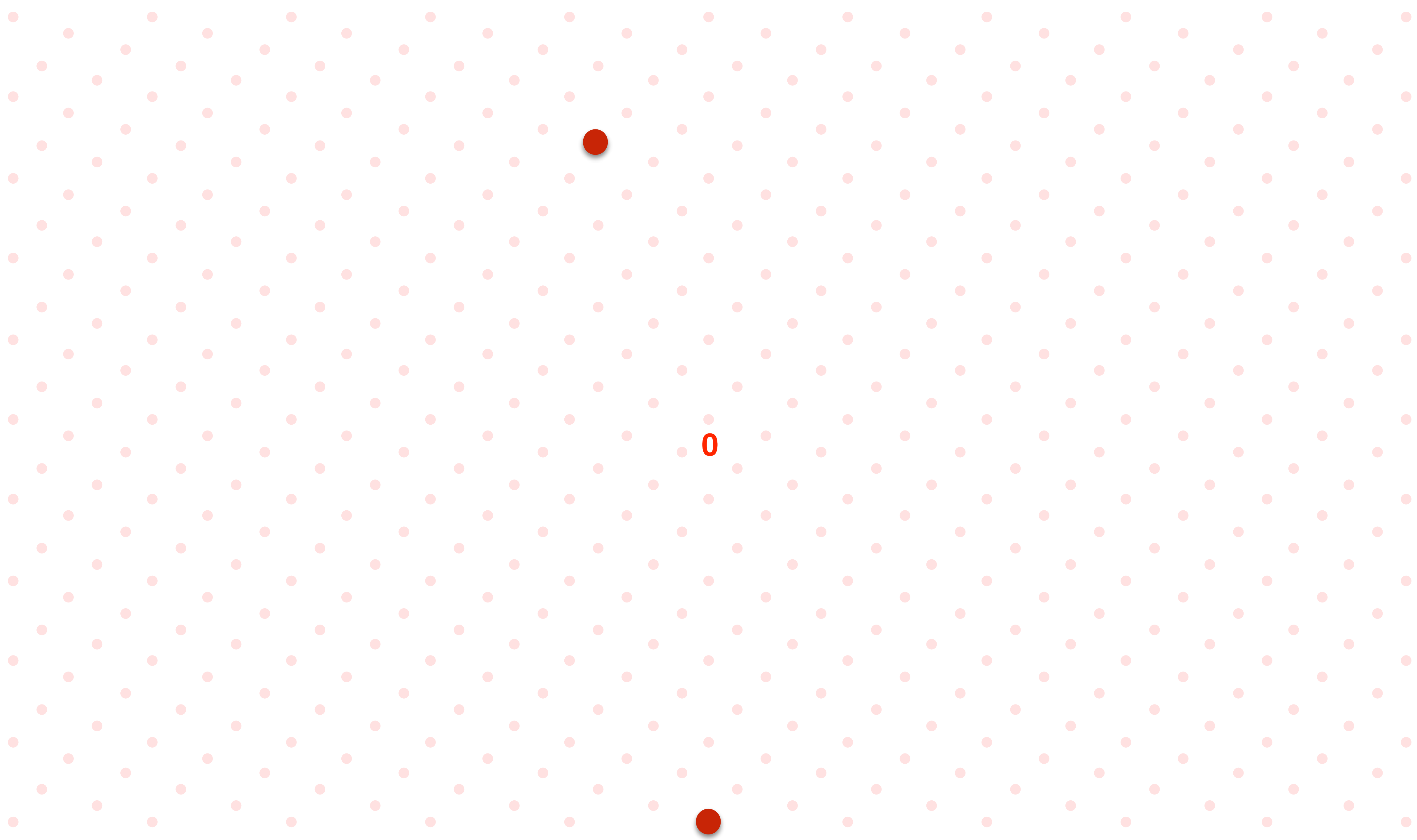
Sieving



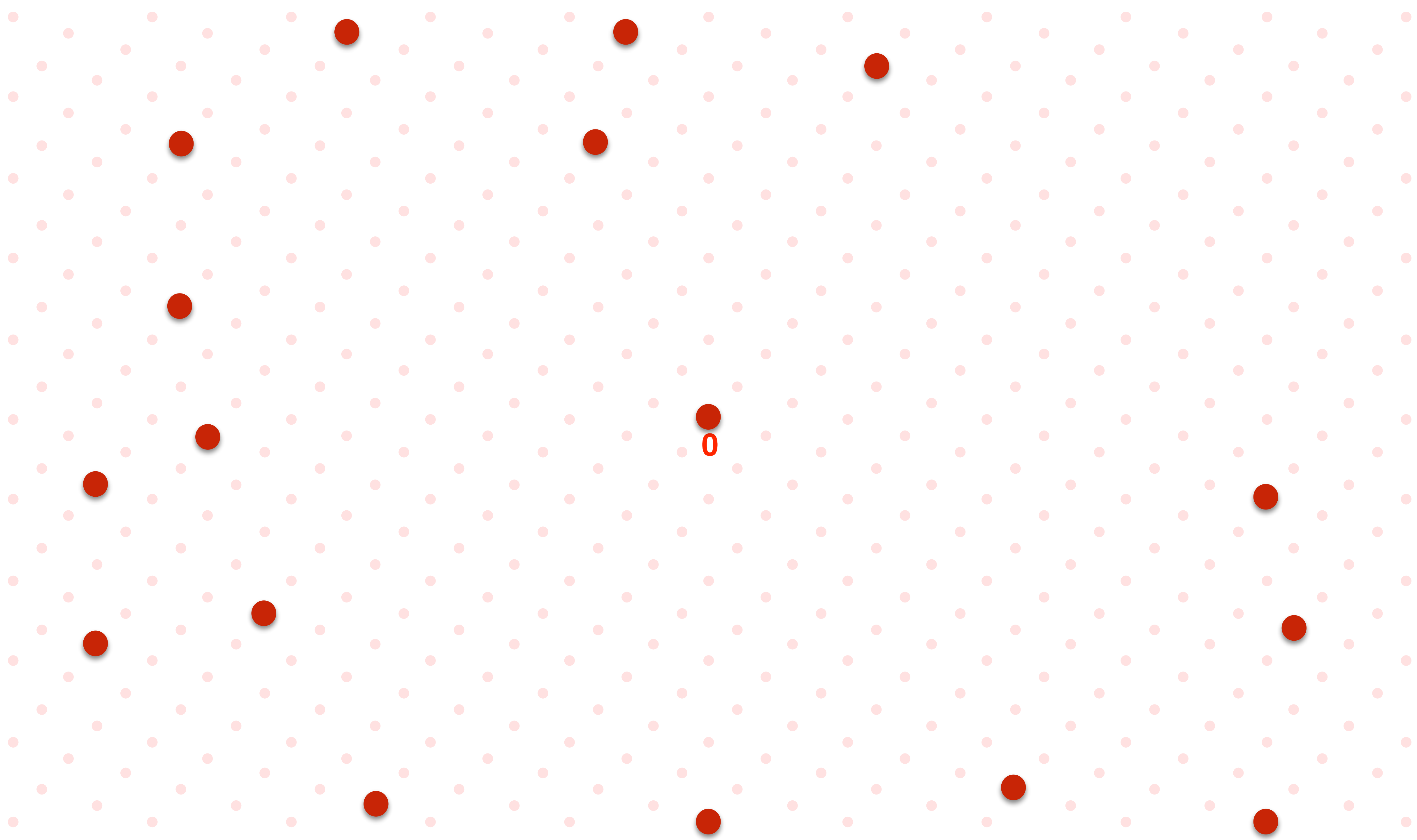
Sieving



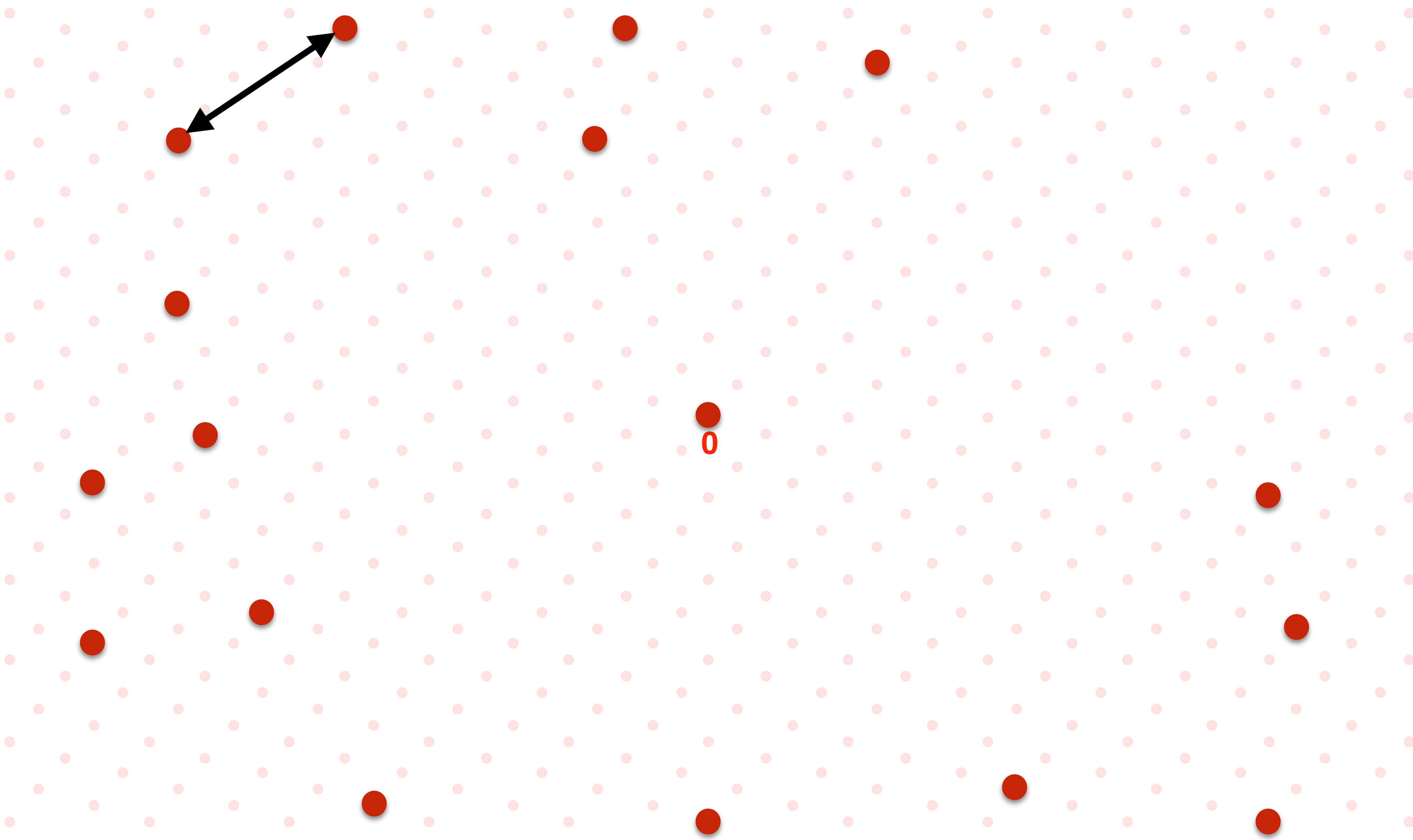
Sieving



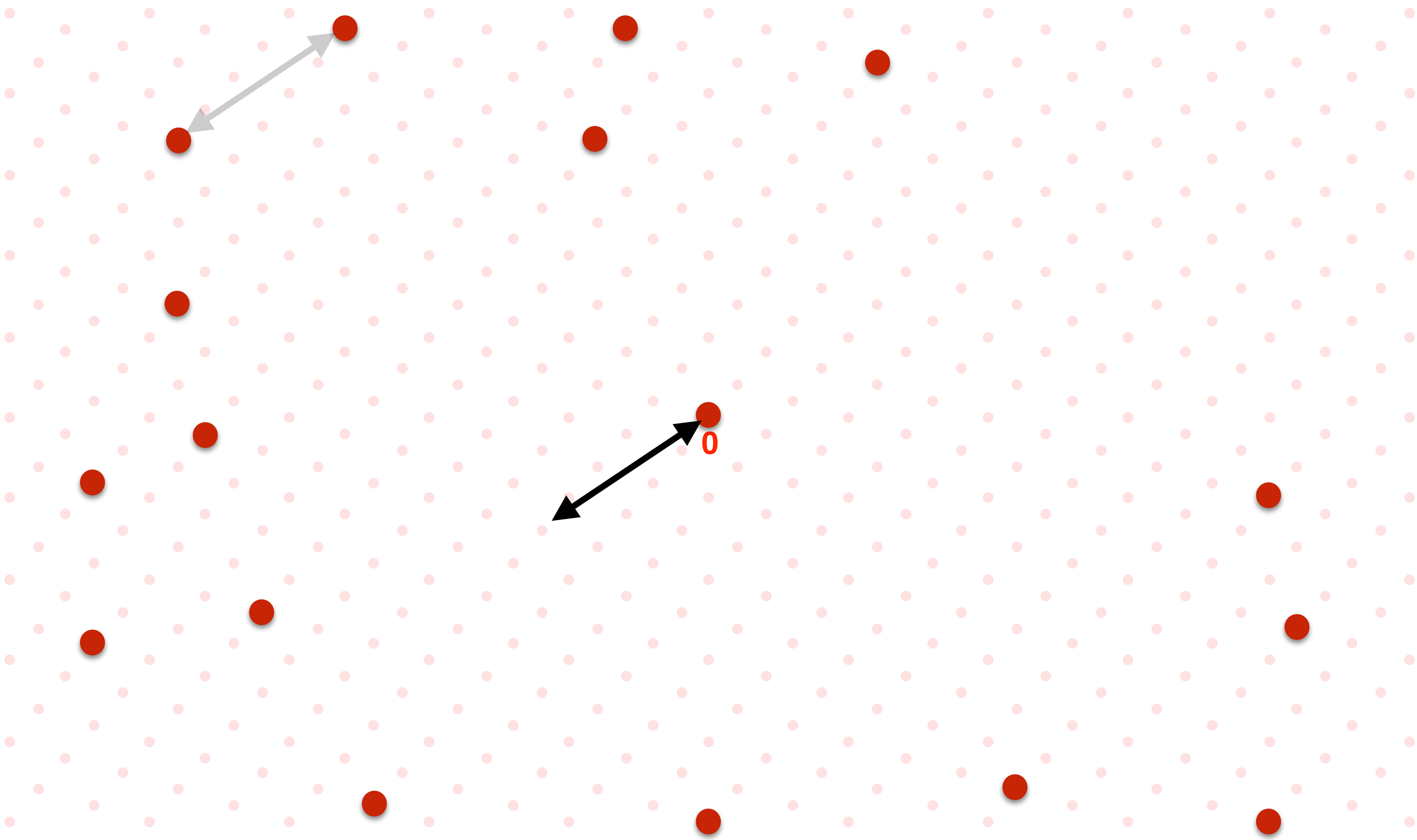
Sieving



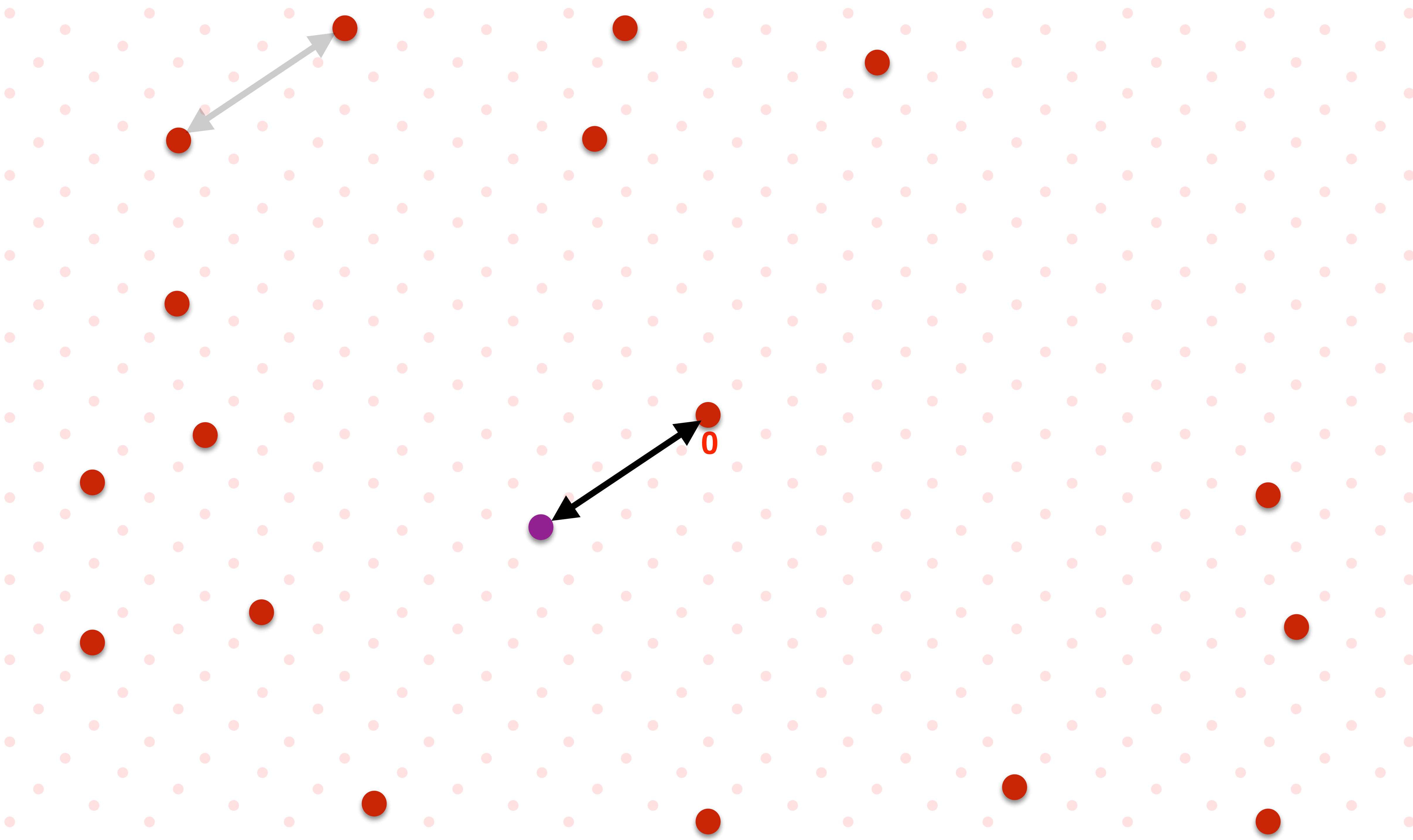
Sieving



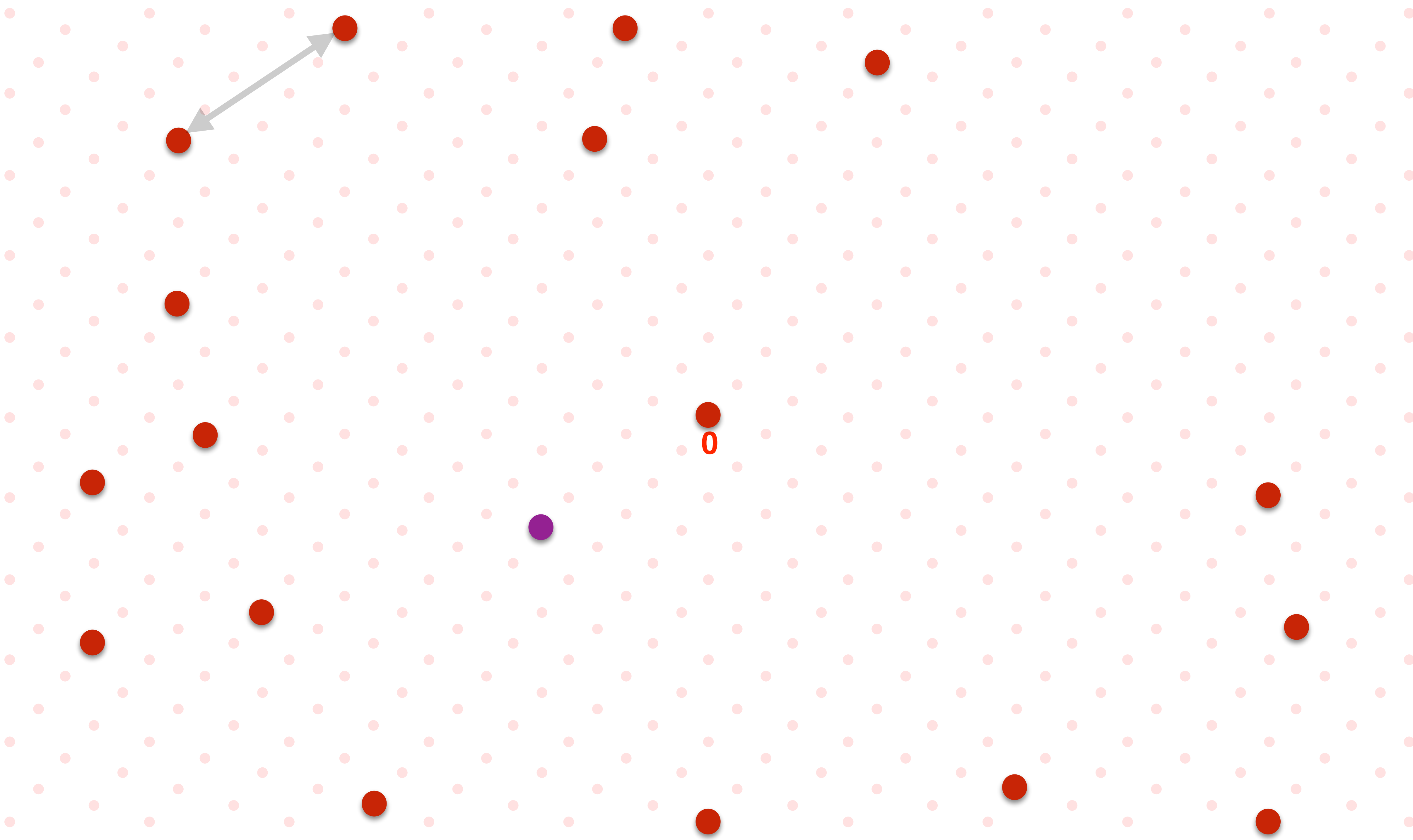
Sieving



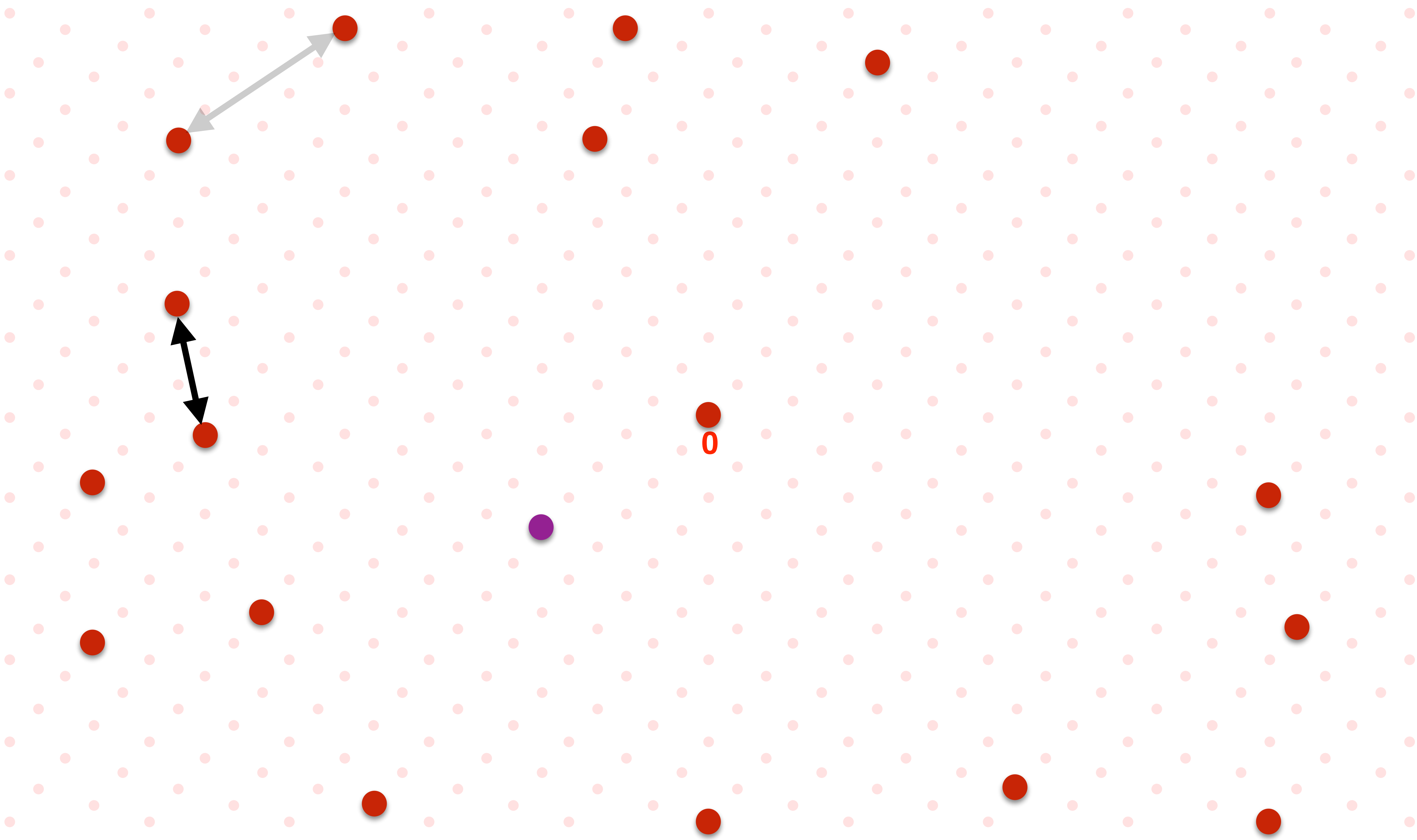
Sieving



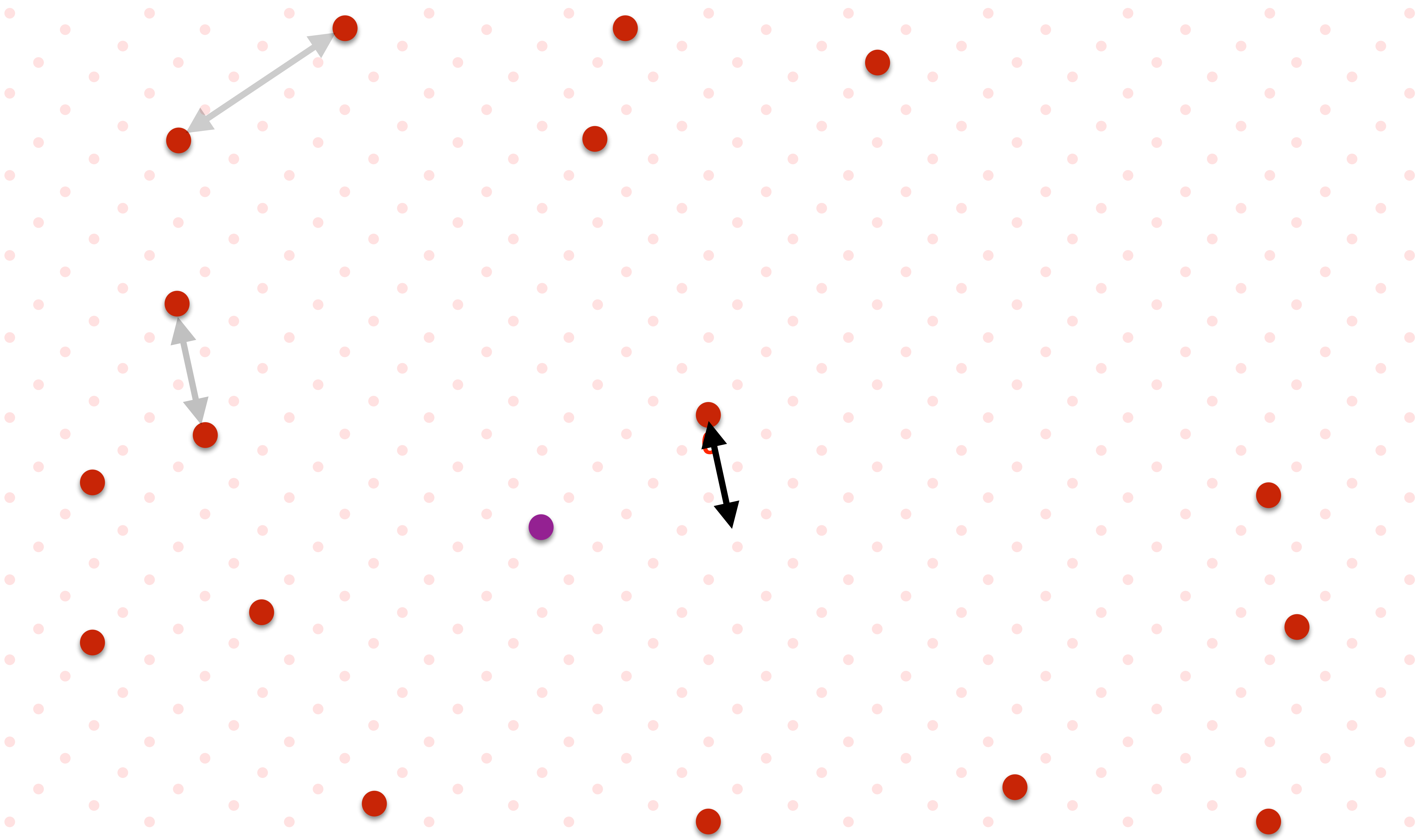
Sieving



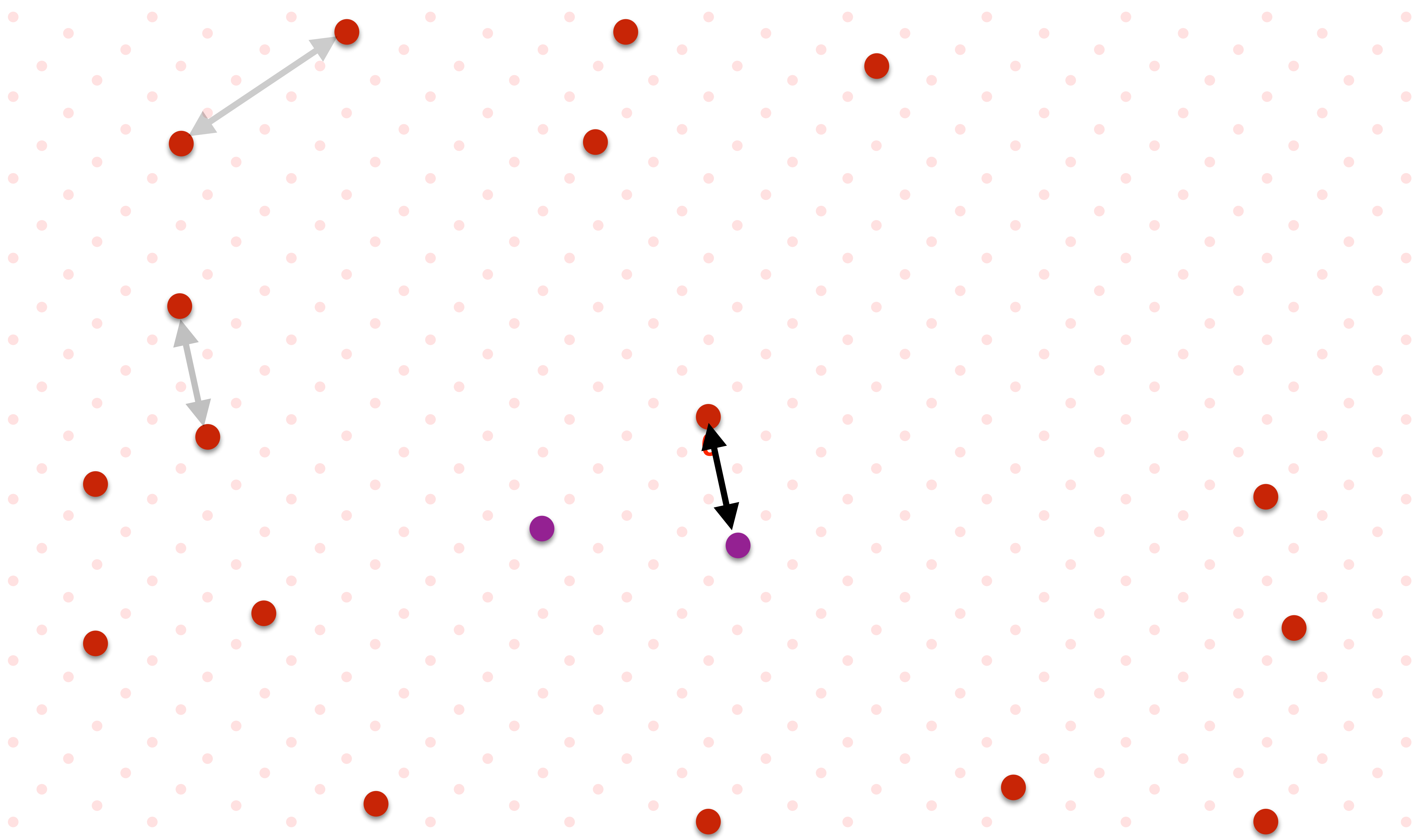
Sieving



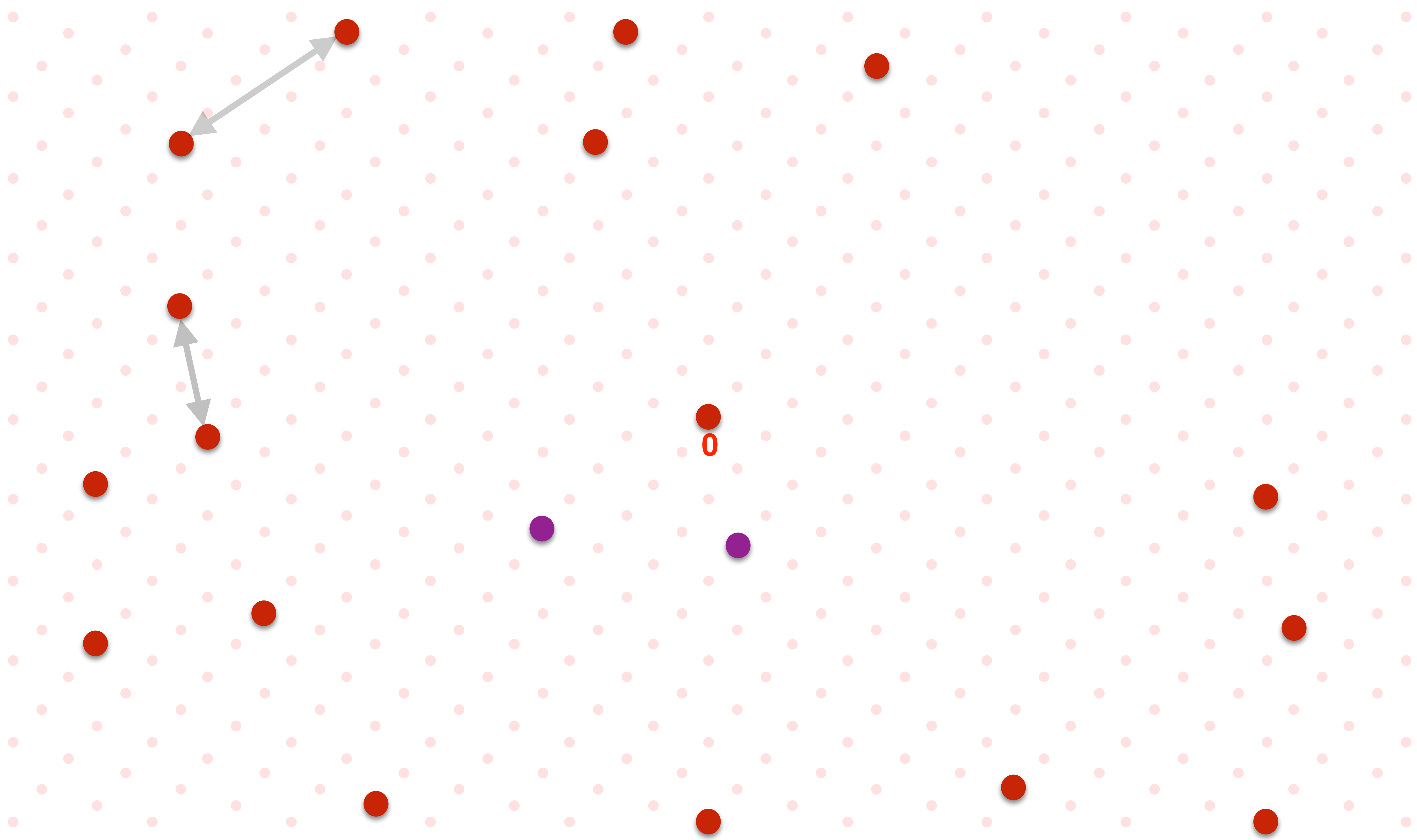
Sieving



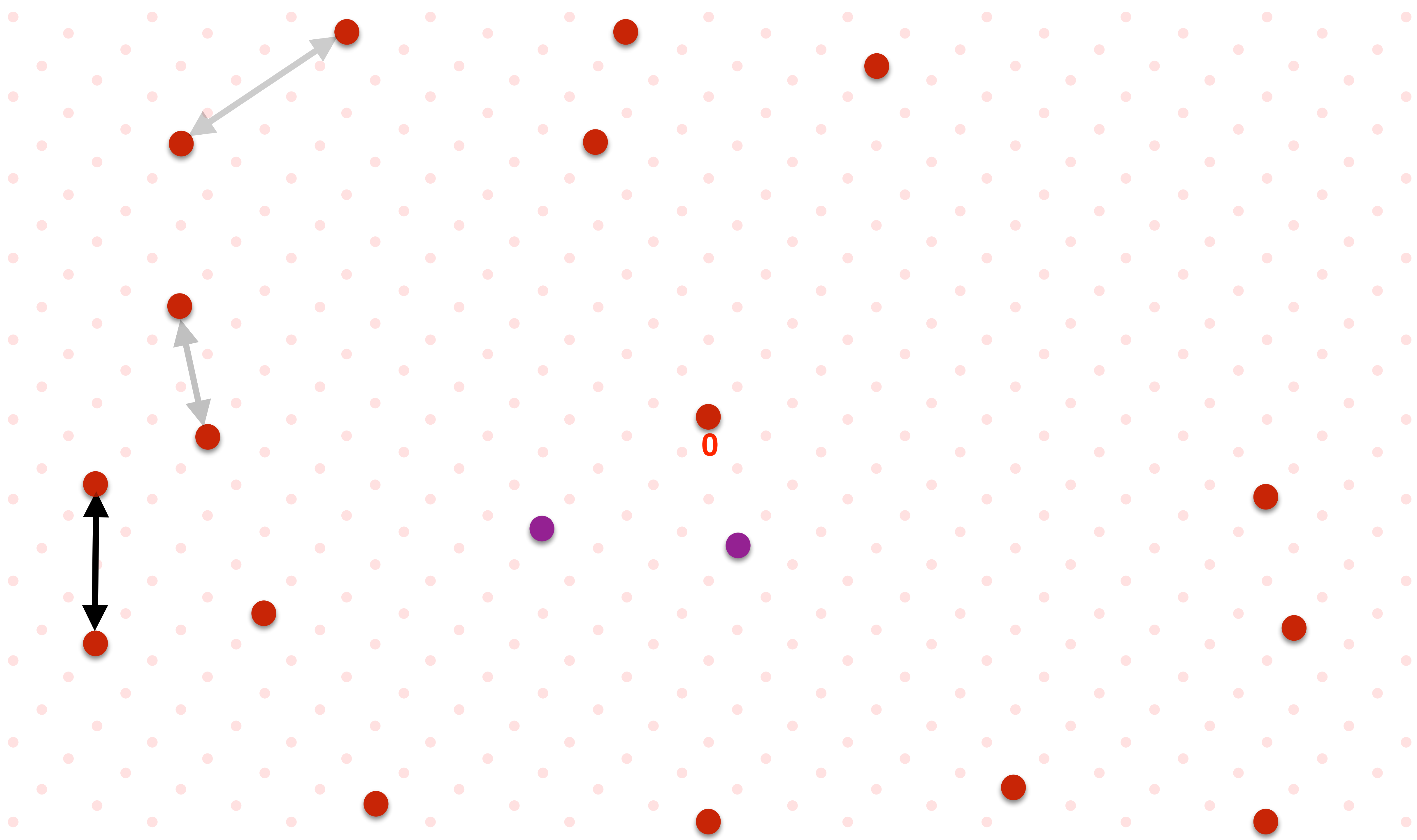
Sieving



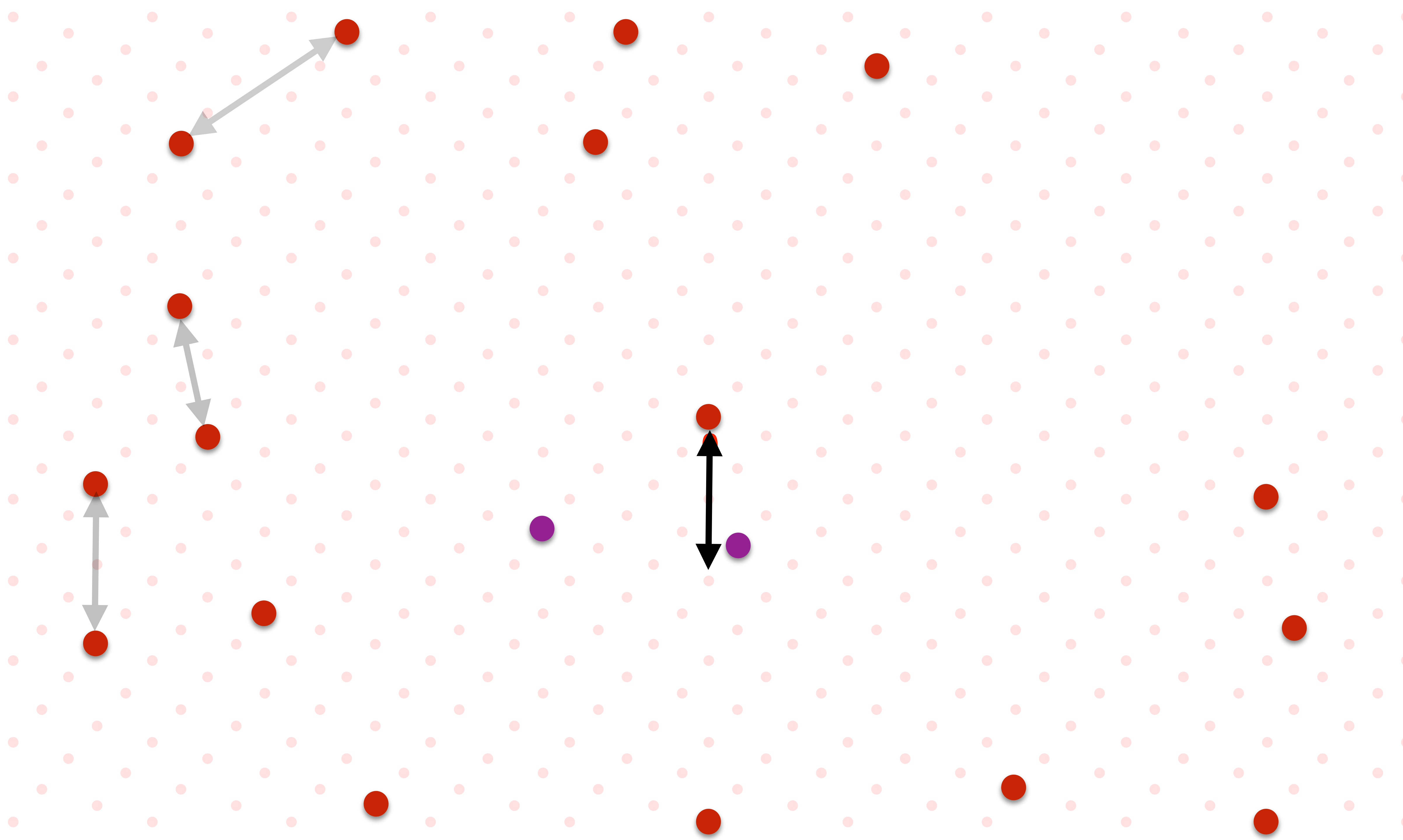
Sieving



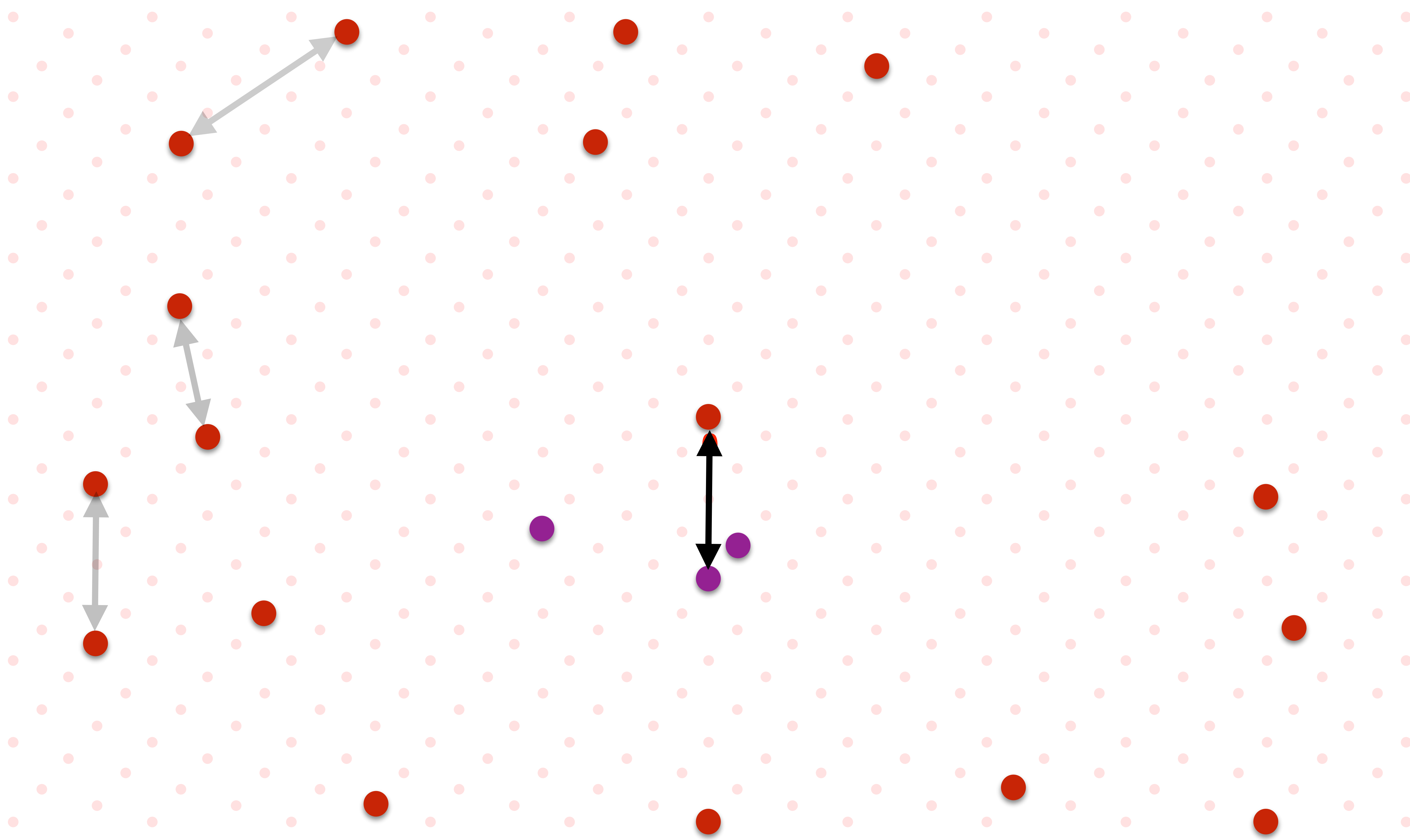
Sieving



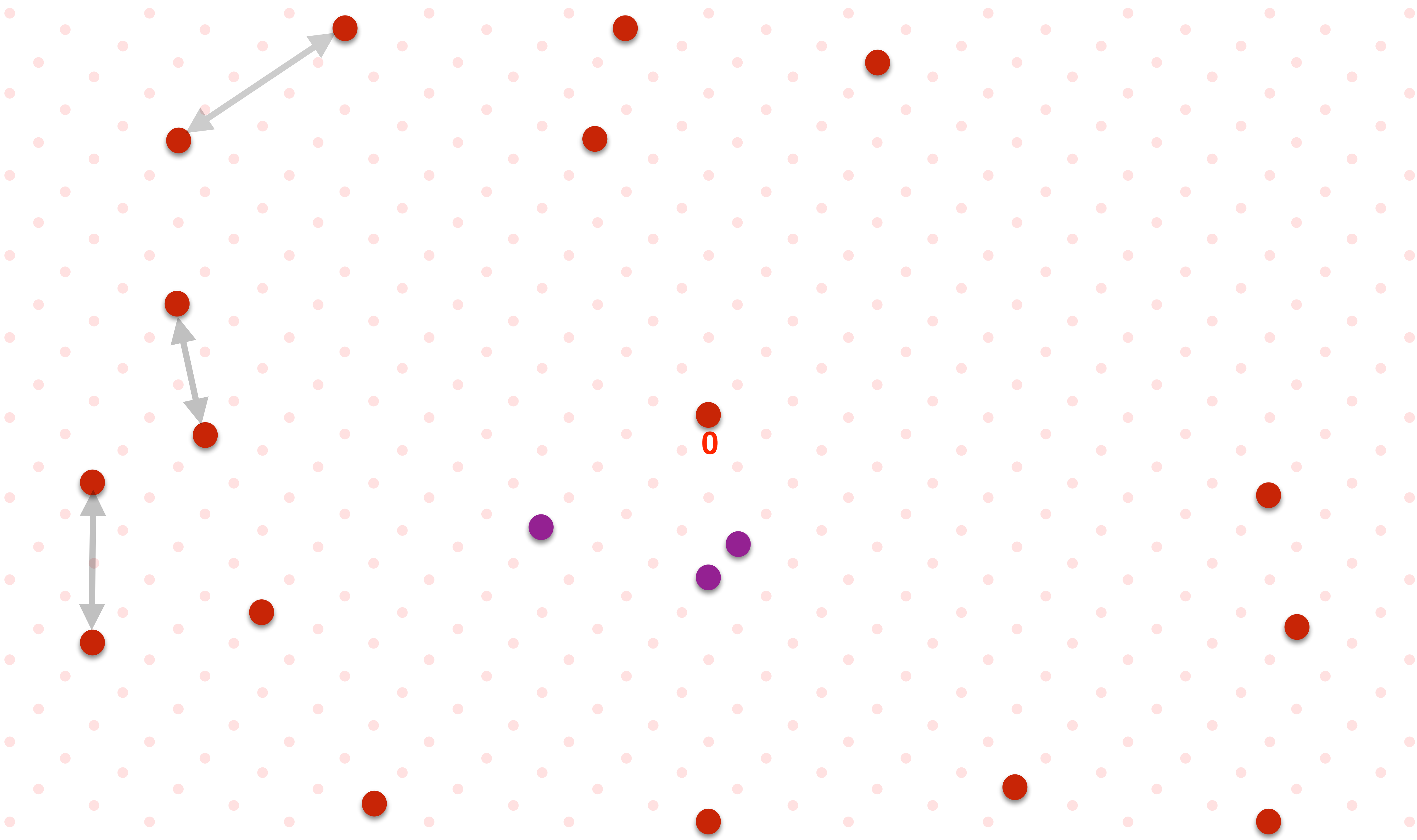
Sieving



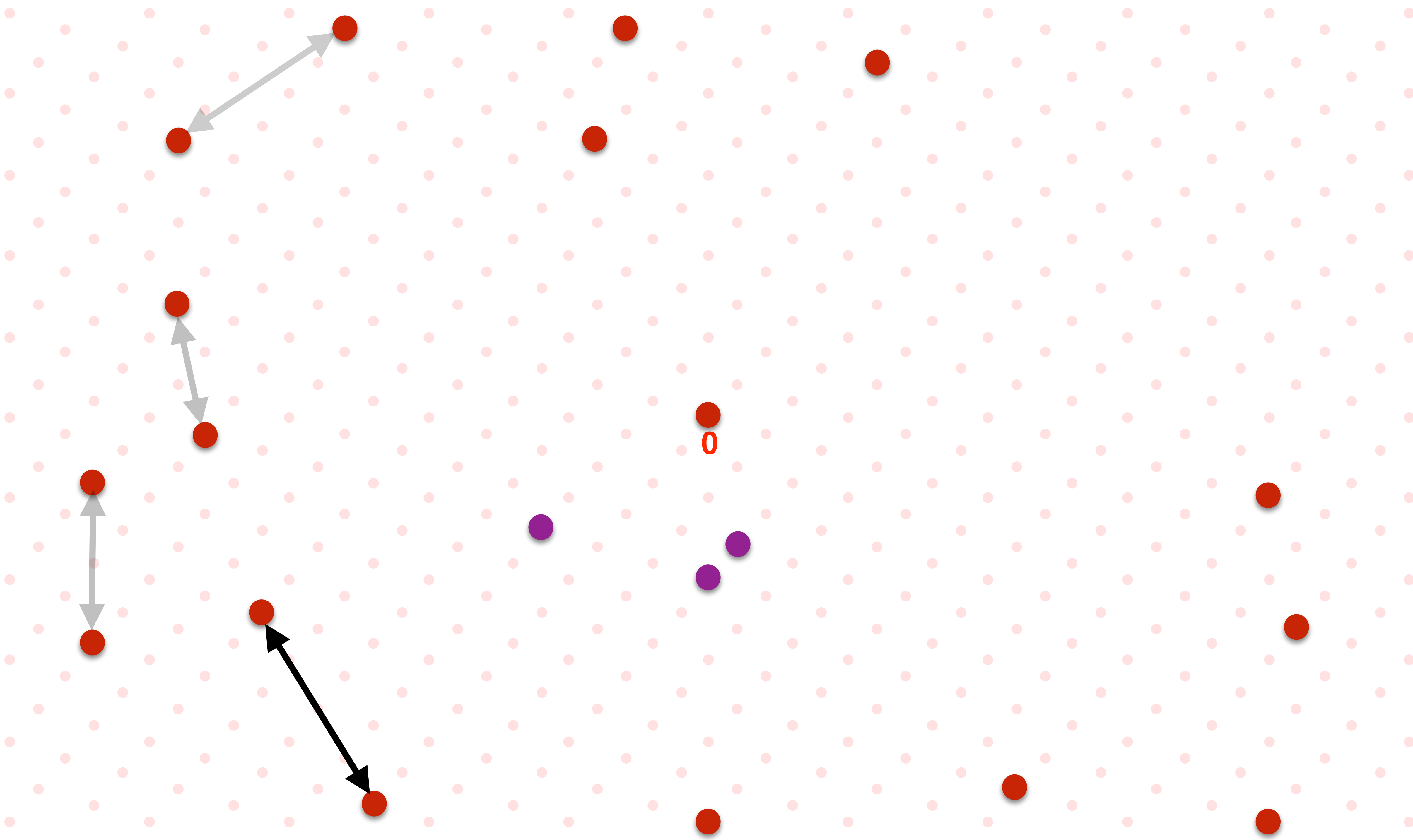
Sieving



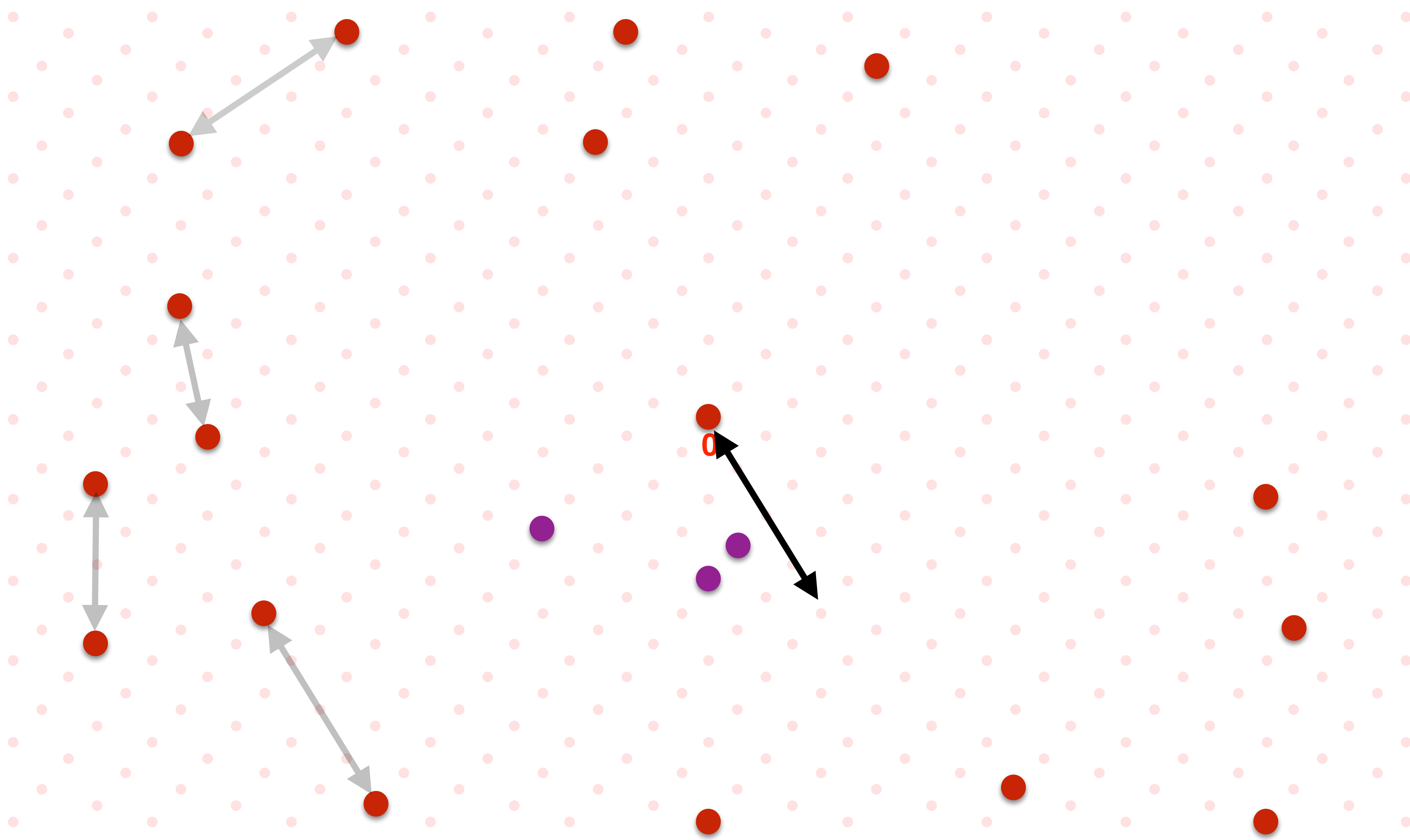
Sieving



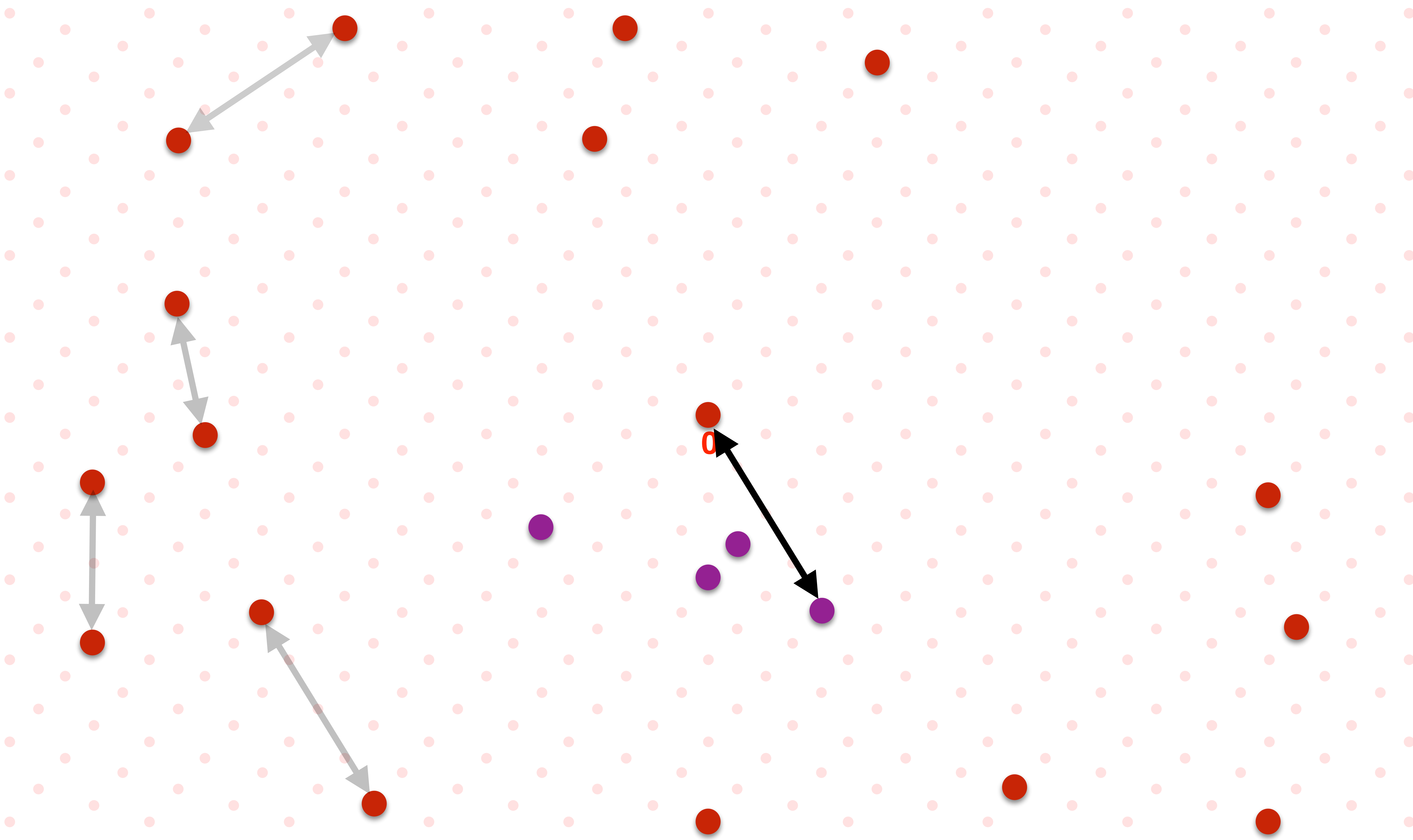
Sieving



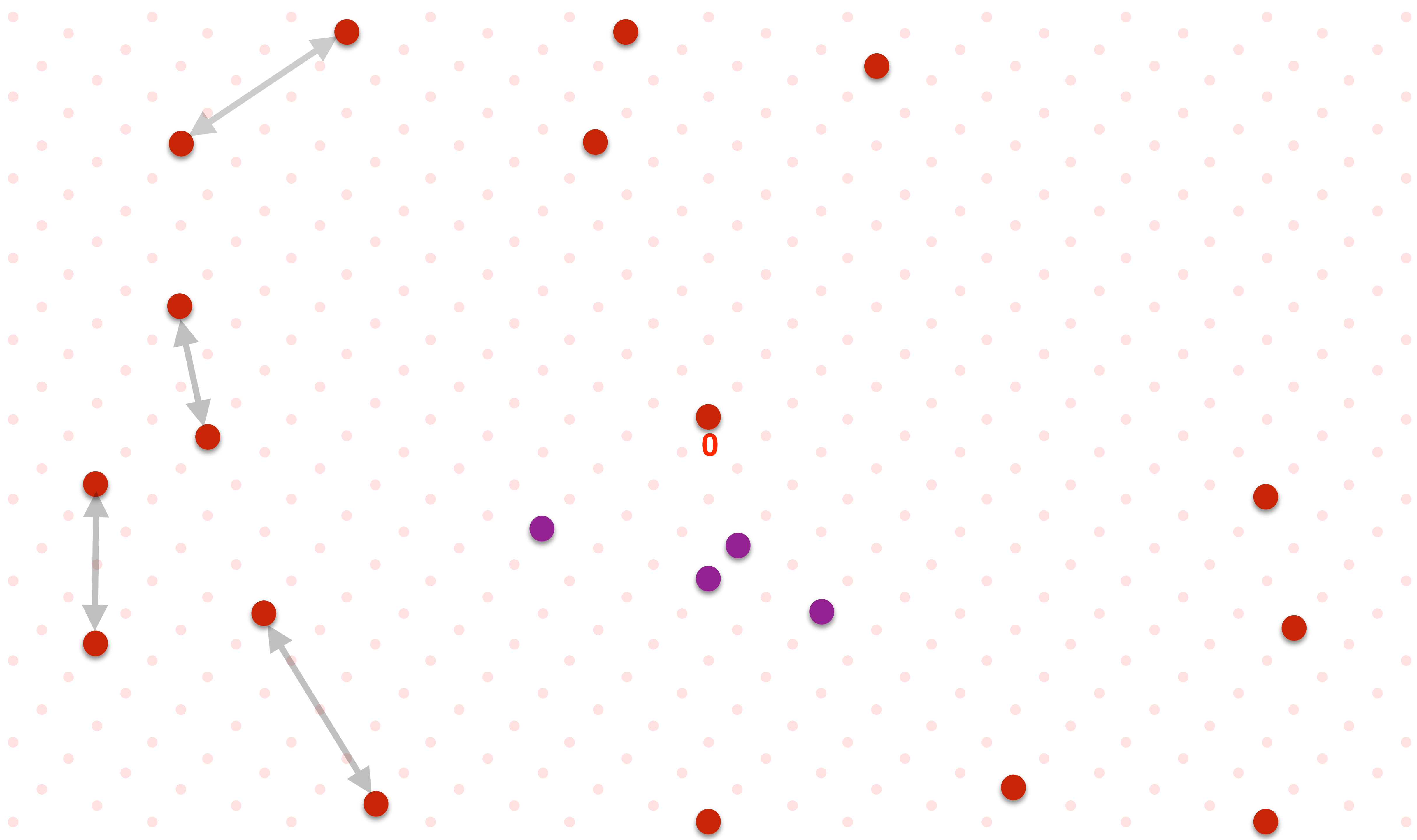
Sieving



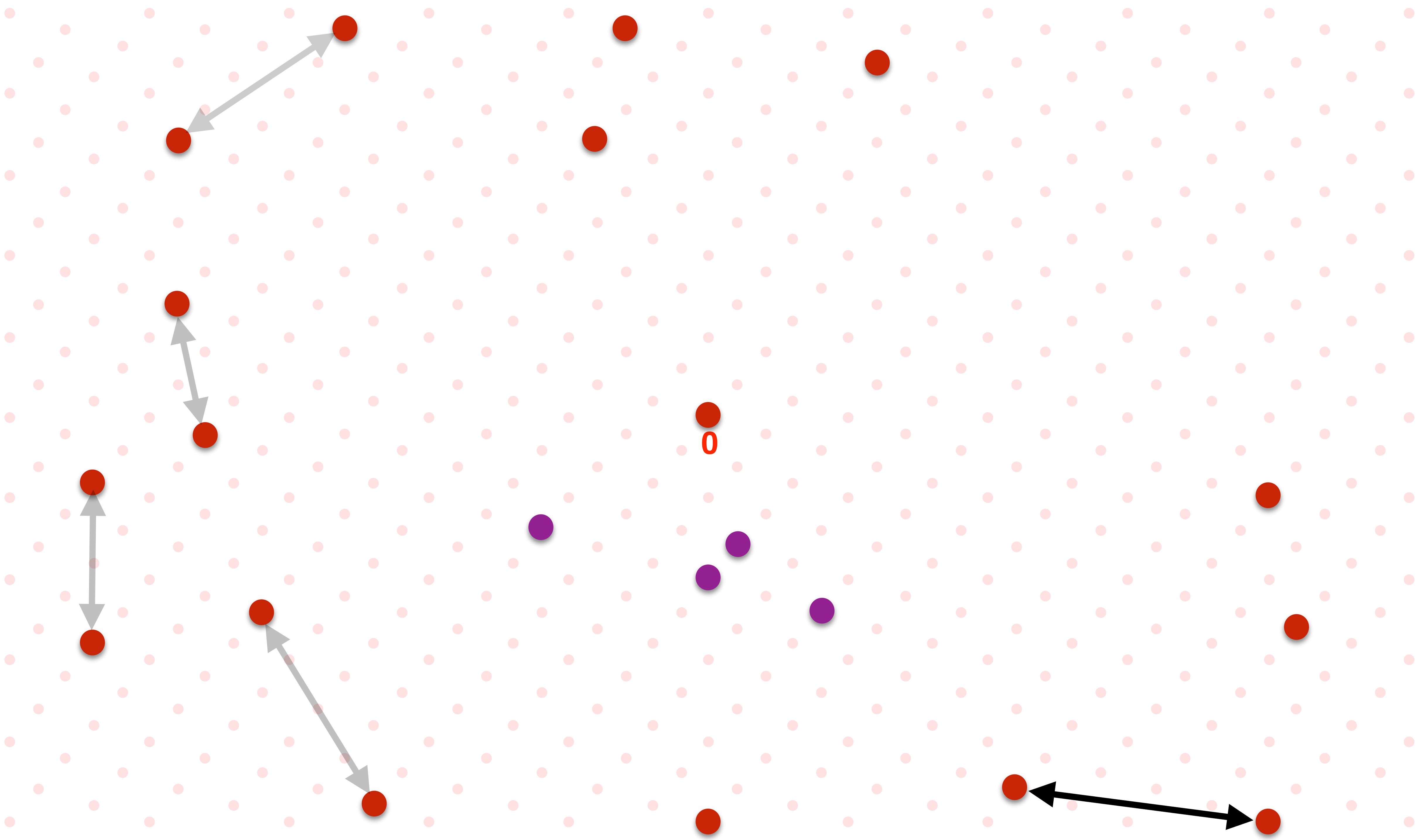
Sieving



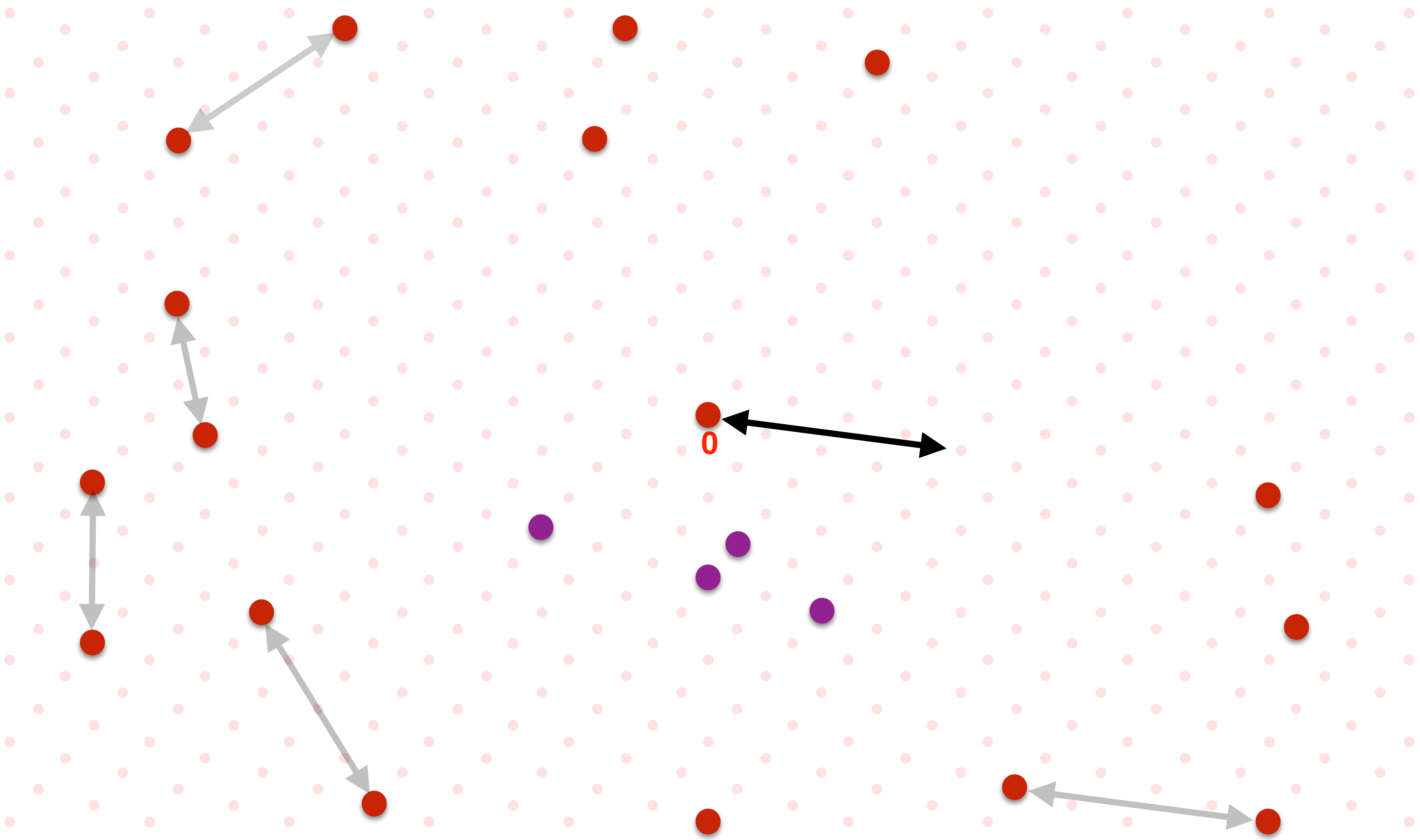
Sieving



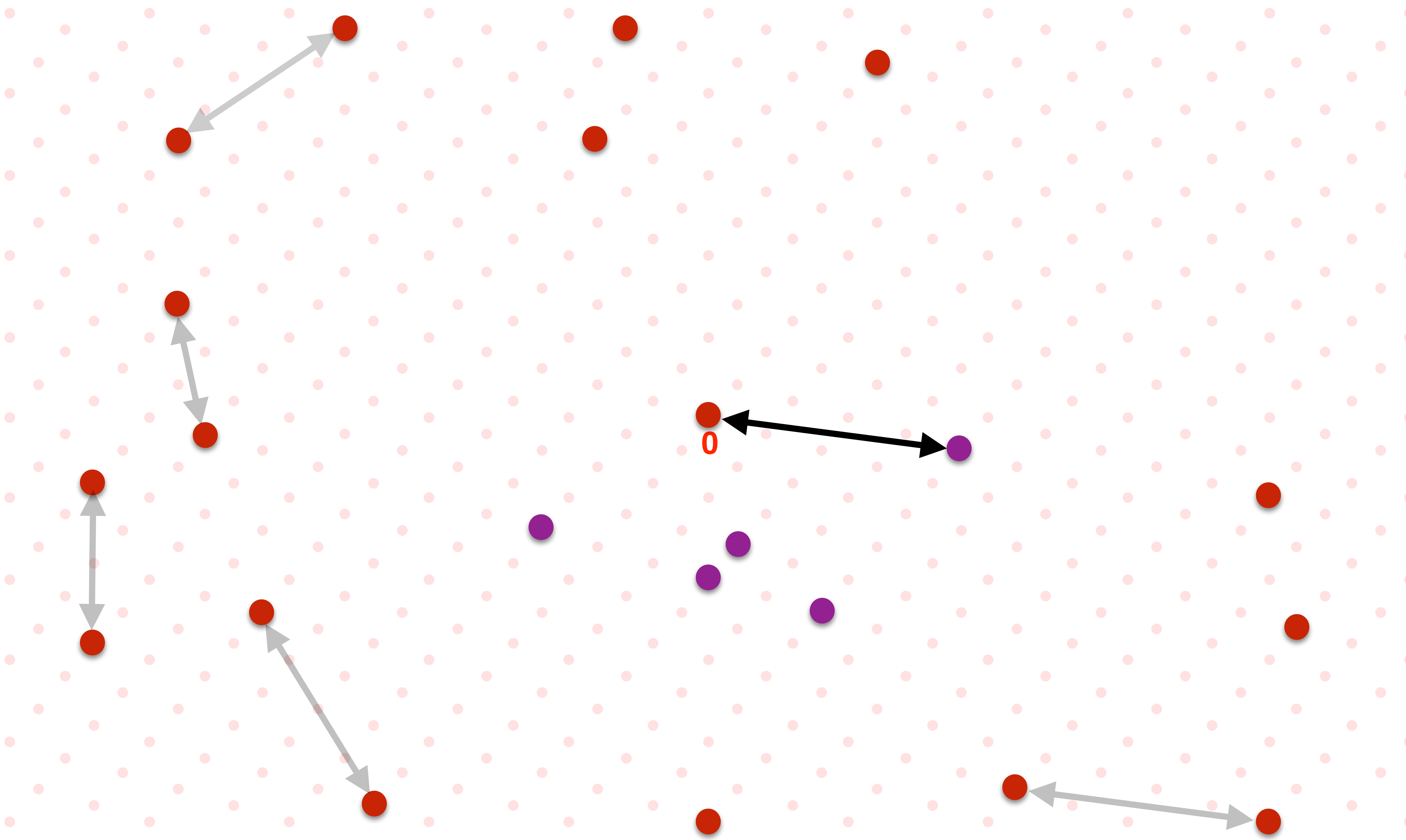
Sieving



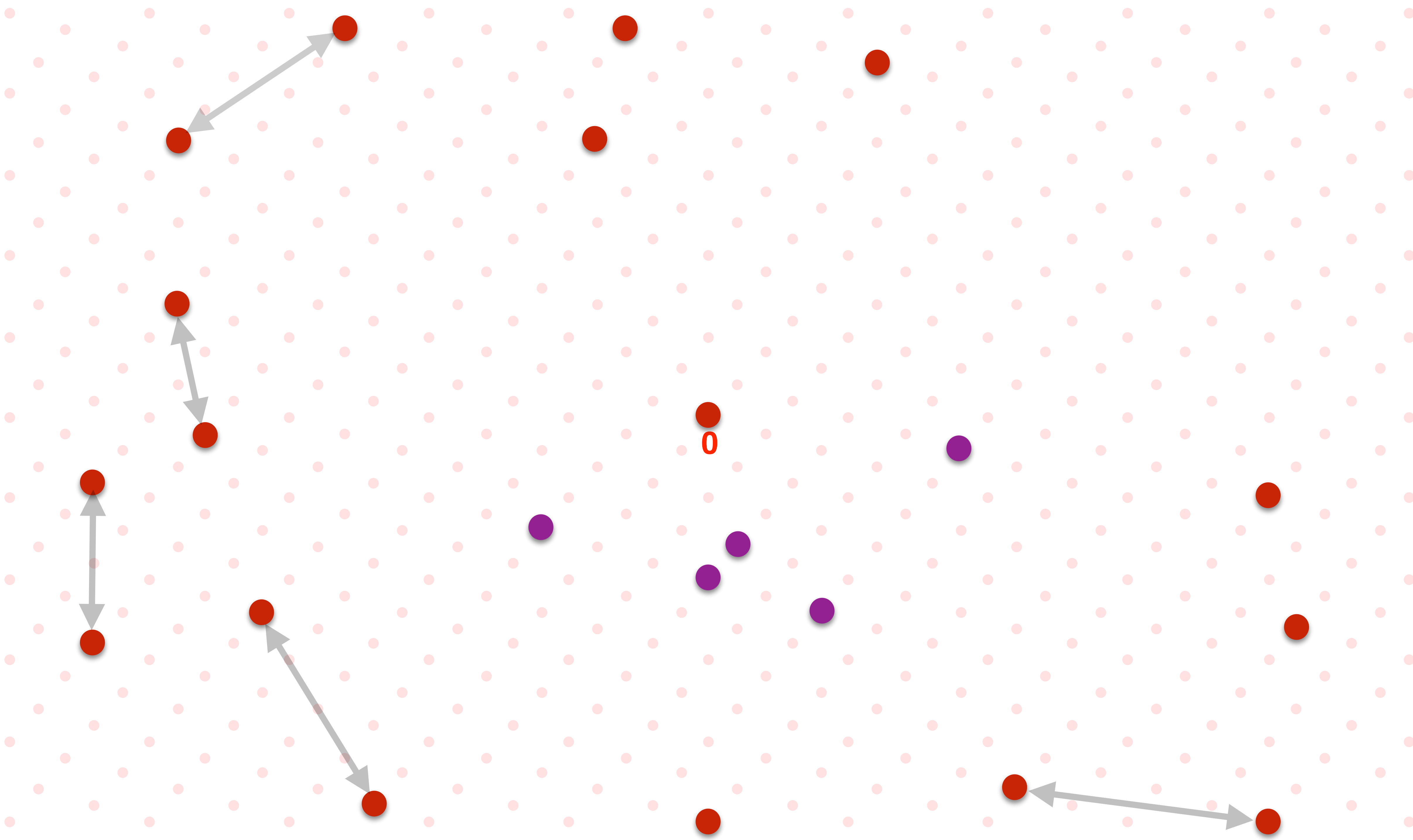
Sieving



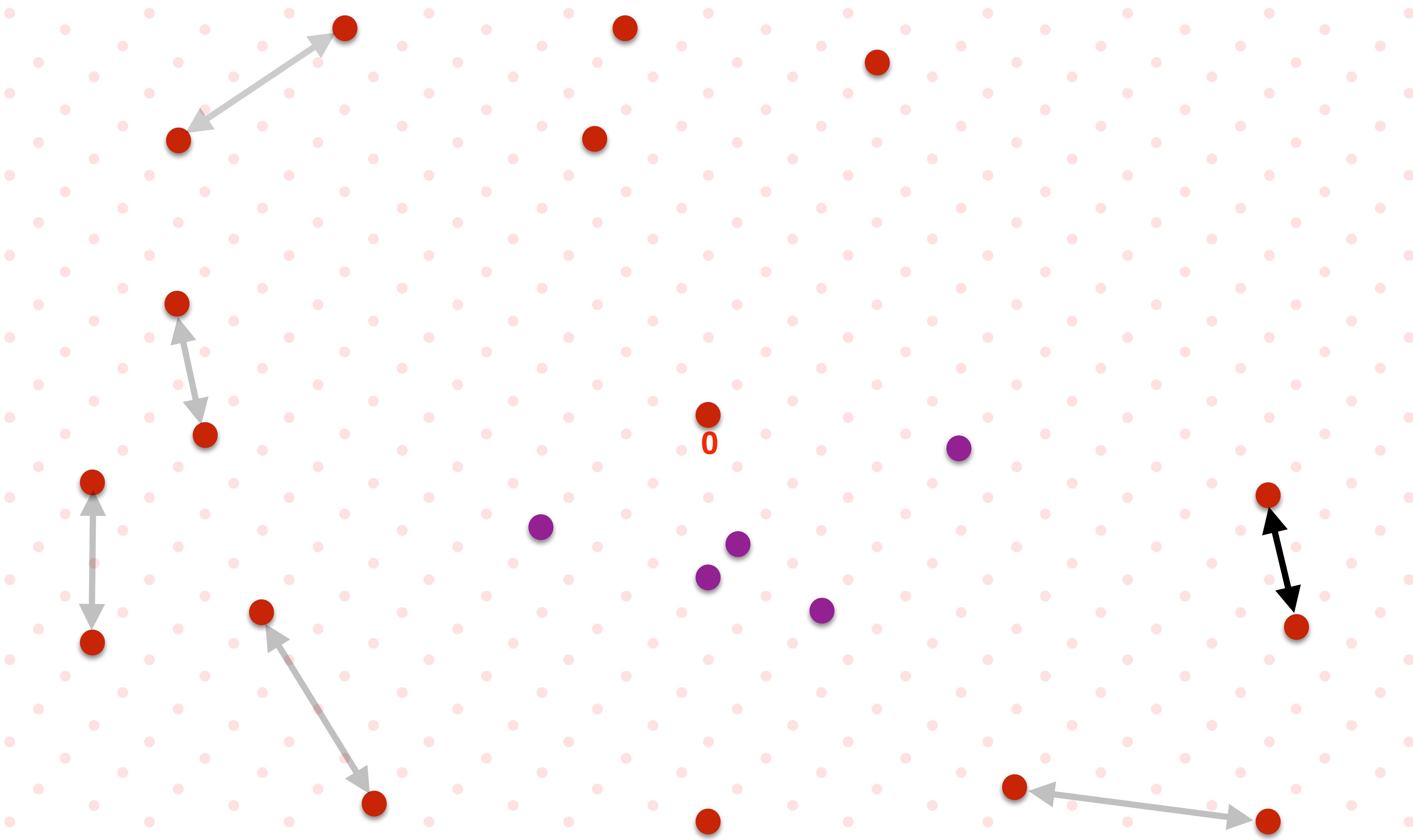
Sieving



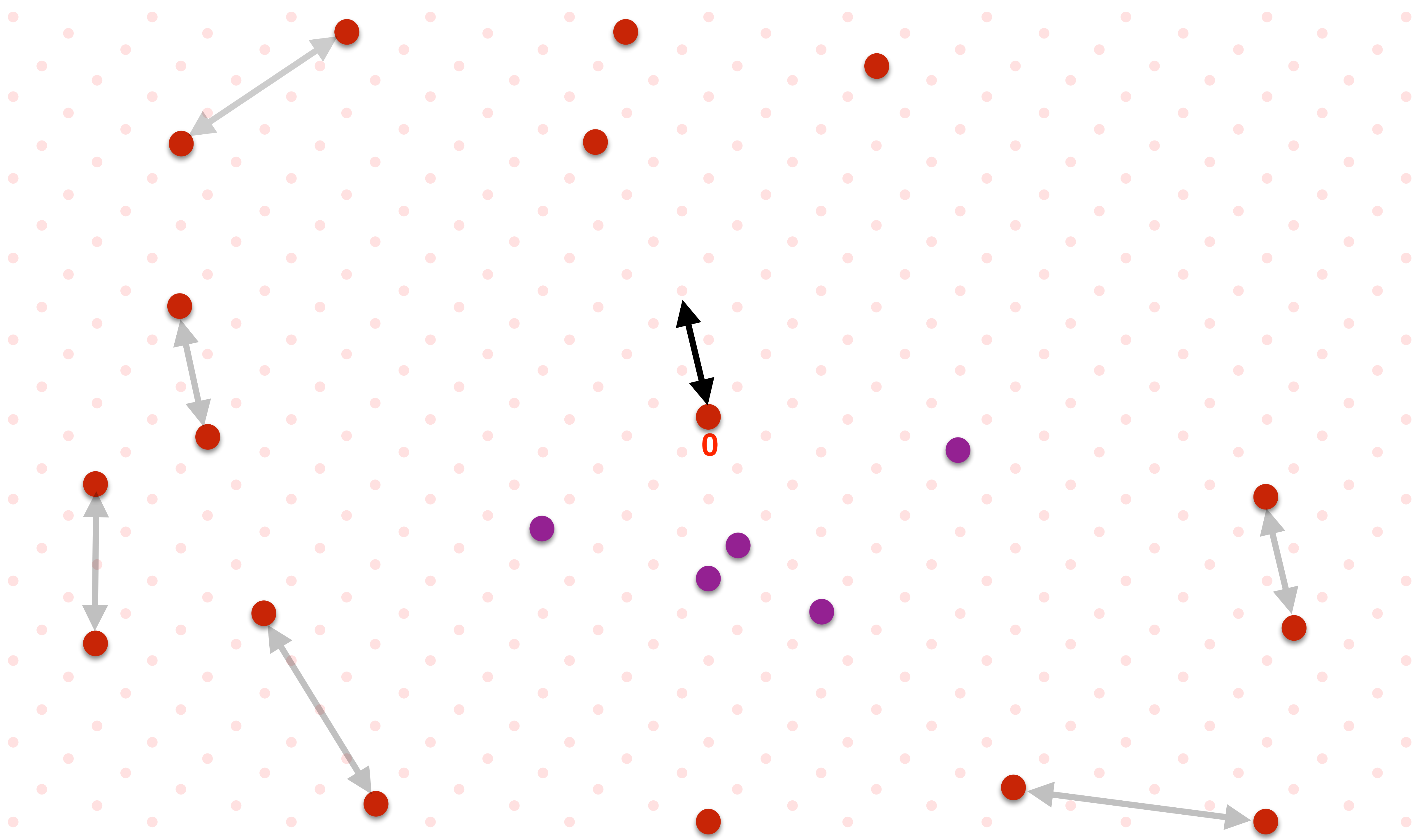
Sieving



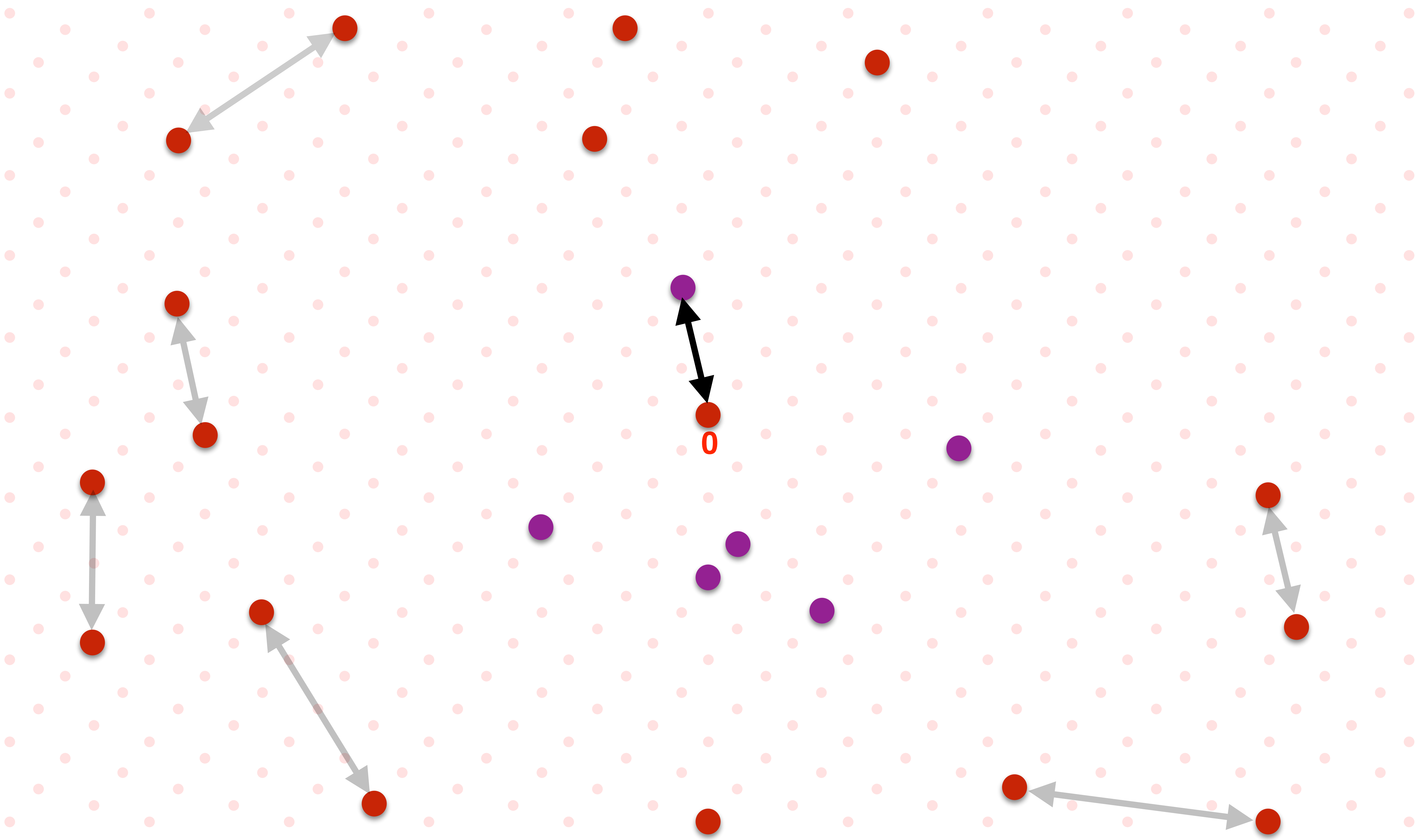
Sieving



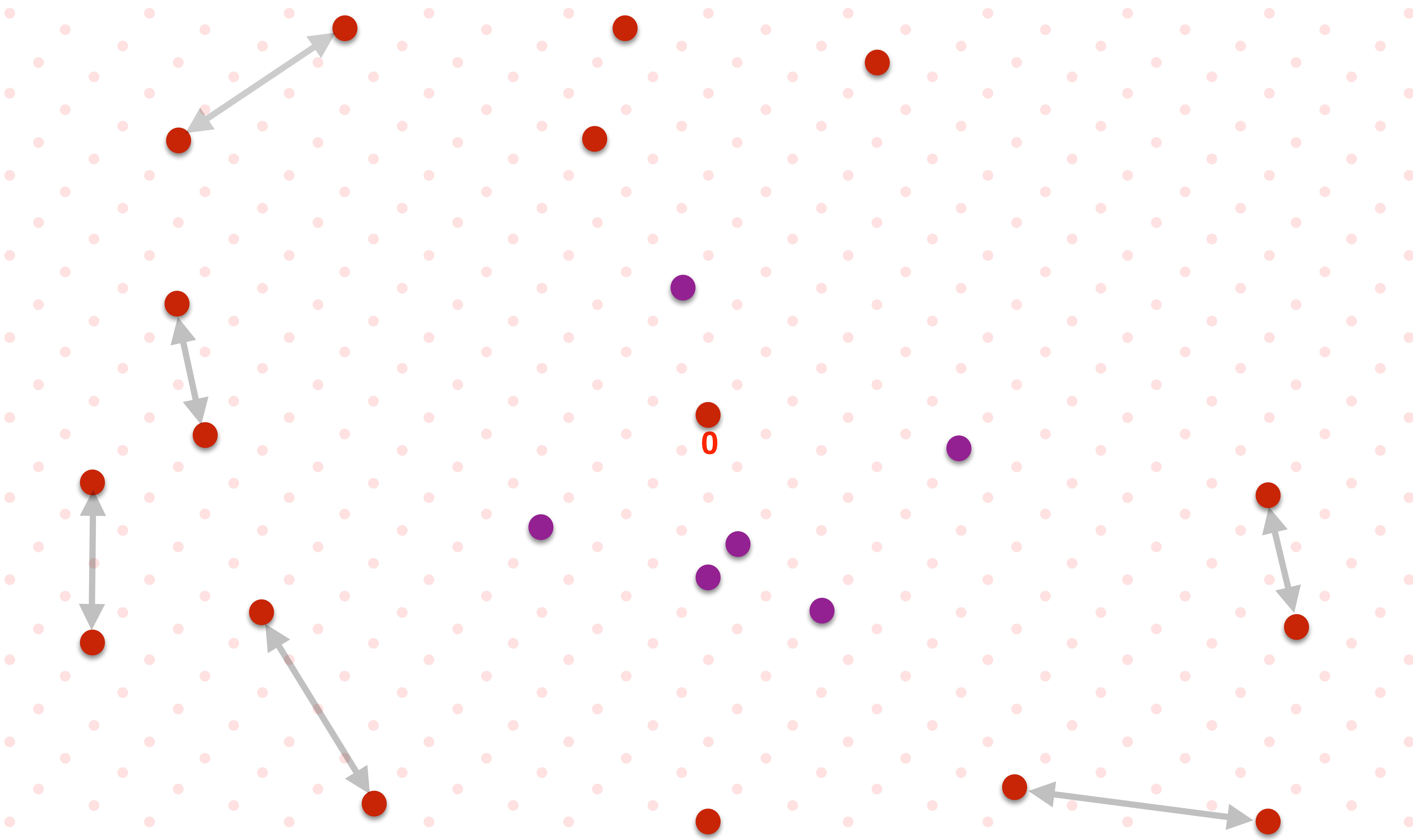
Sieving



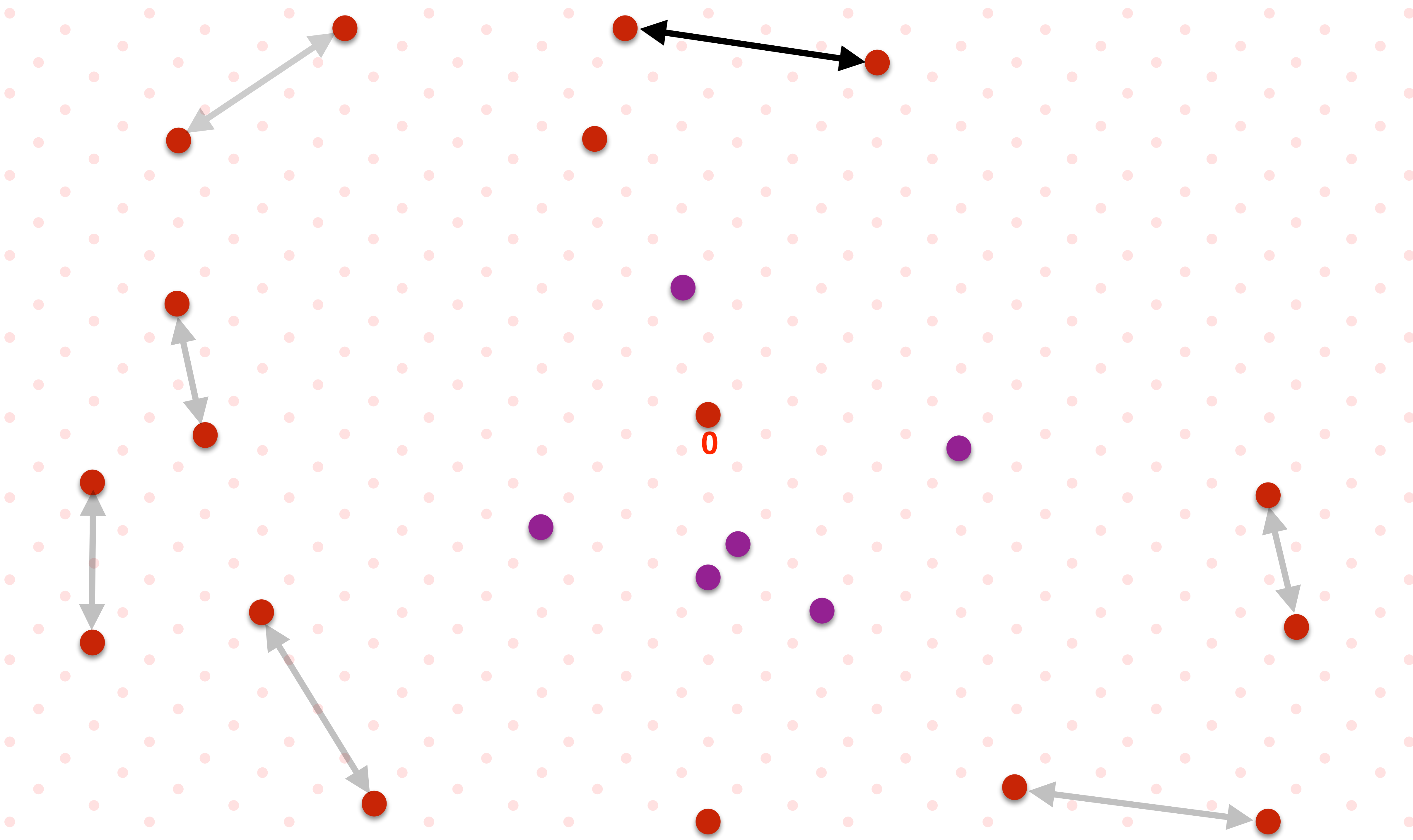
Sieving



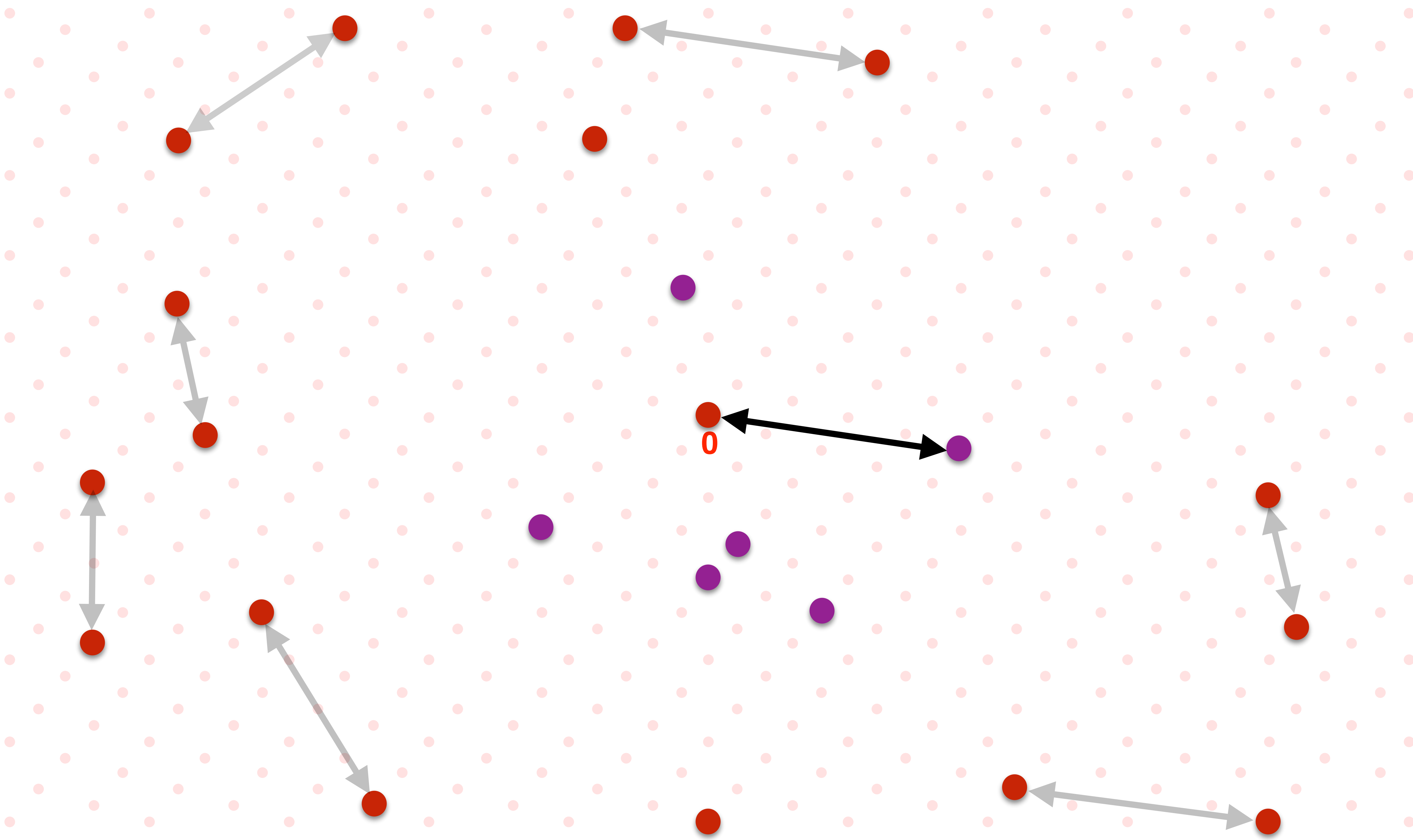
Sieving



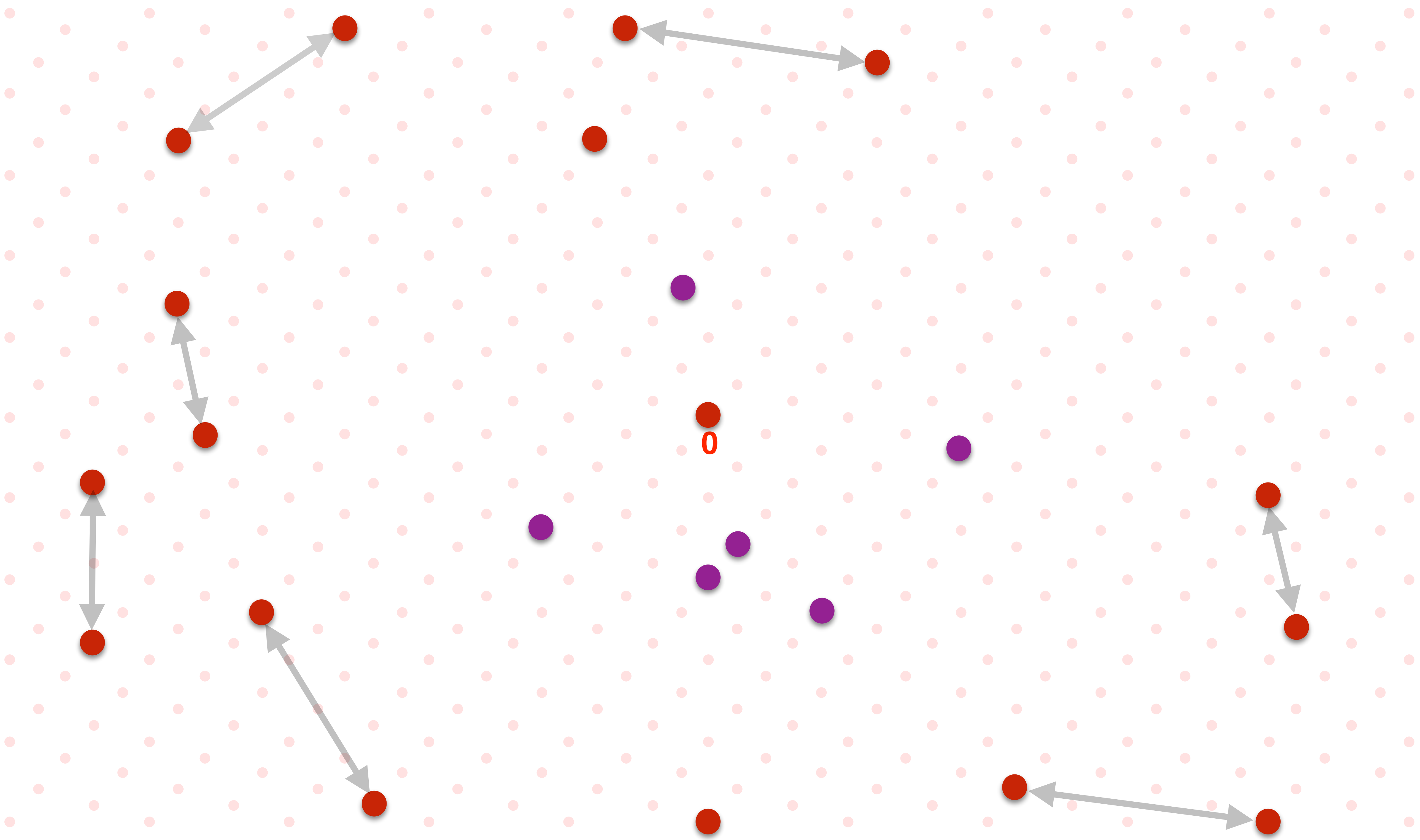
Sieving



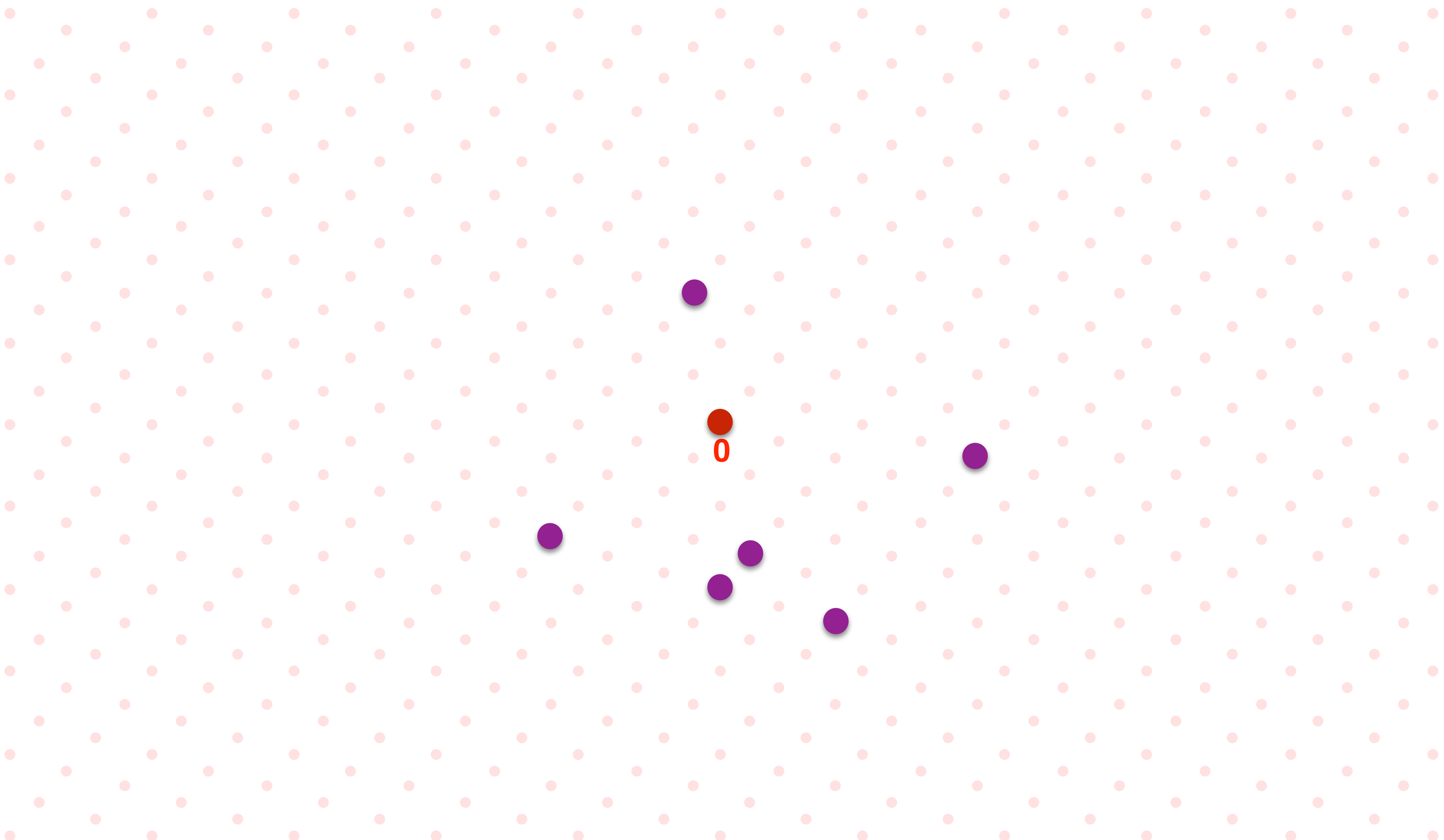
Sieving



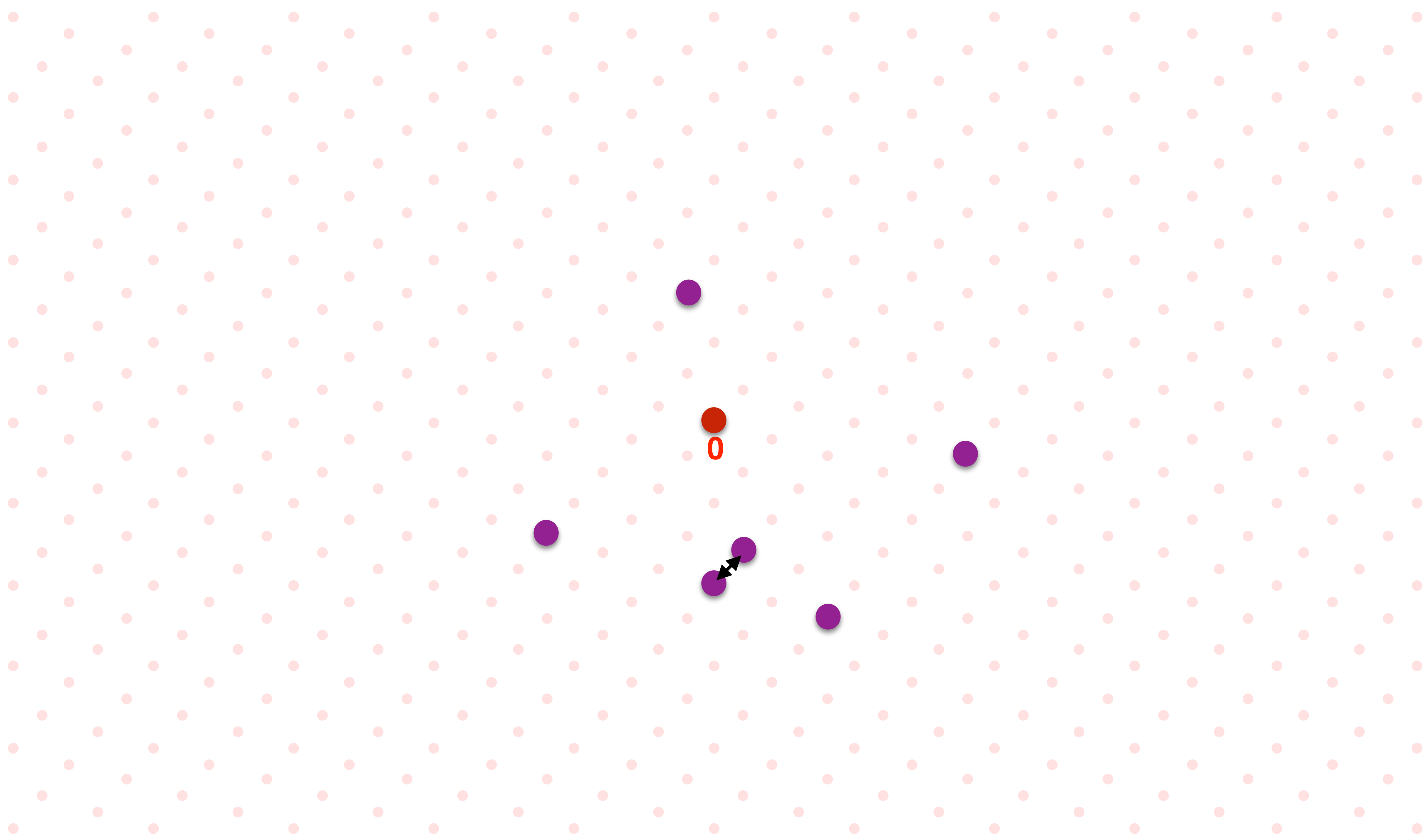
Sieving



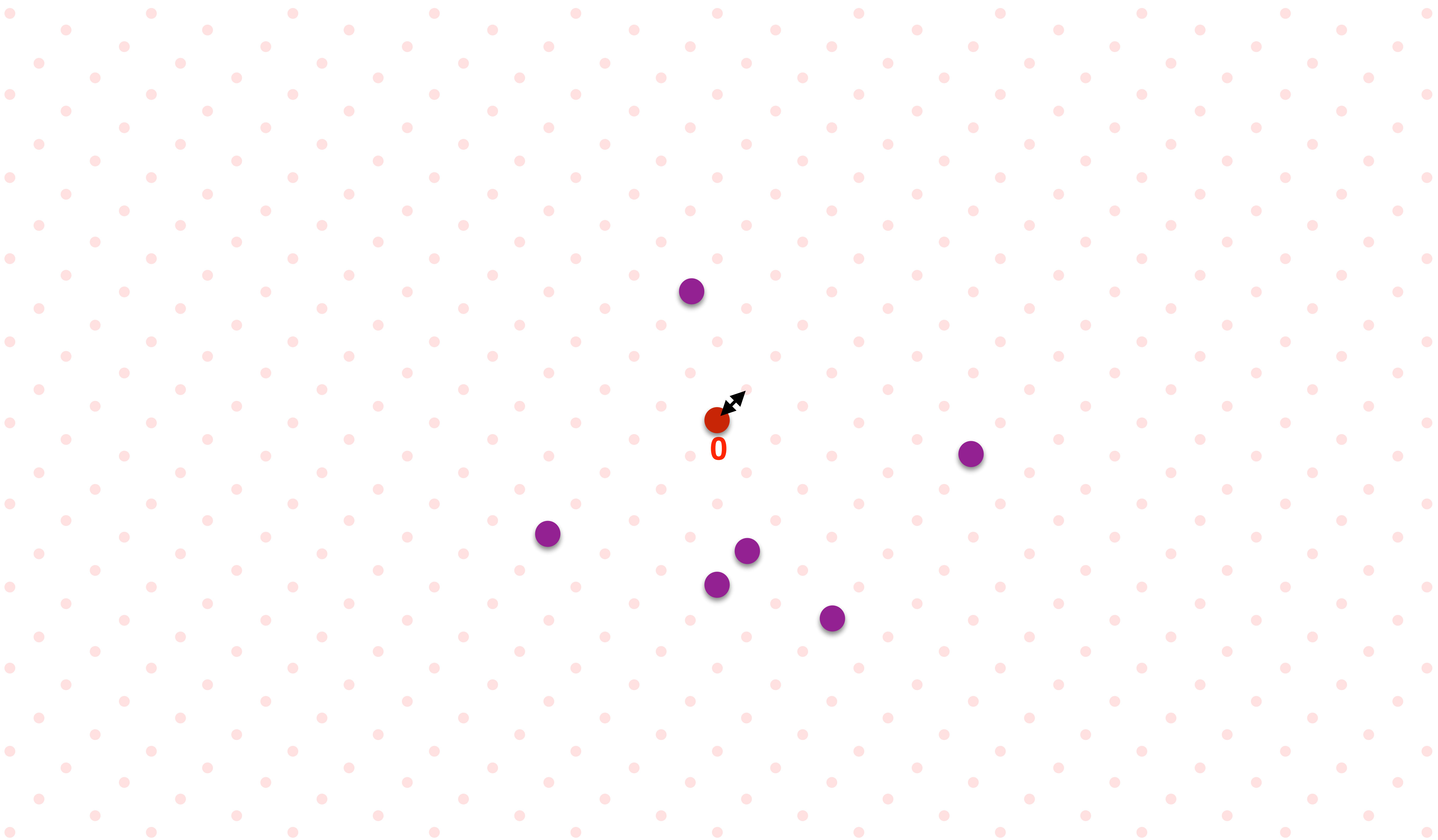
Sieving



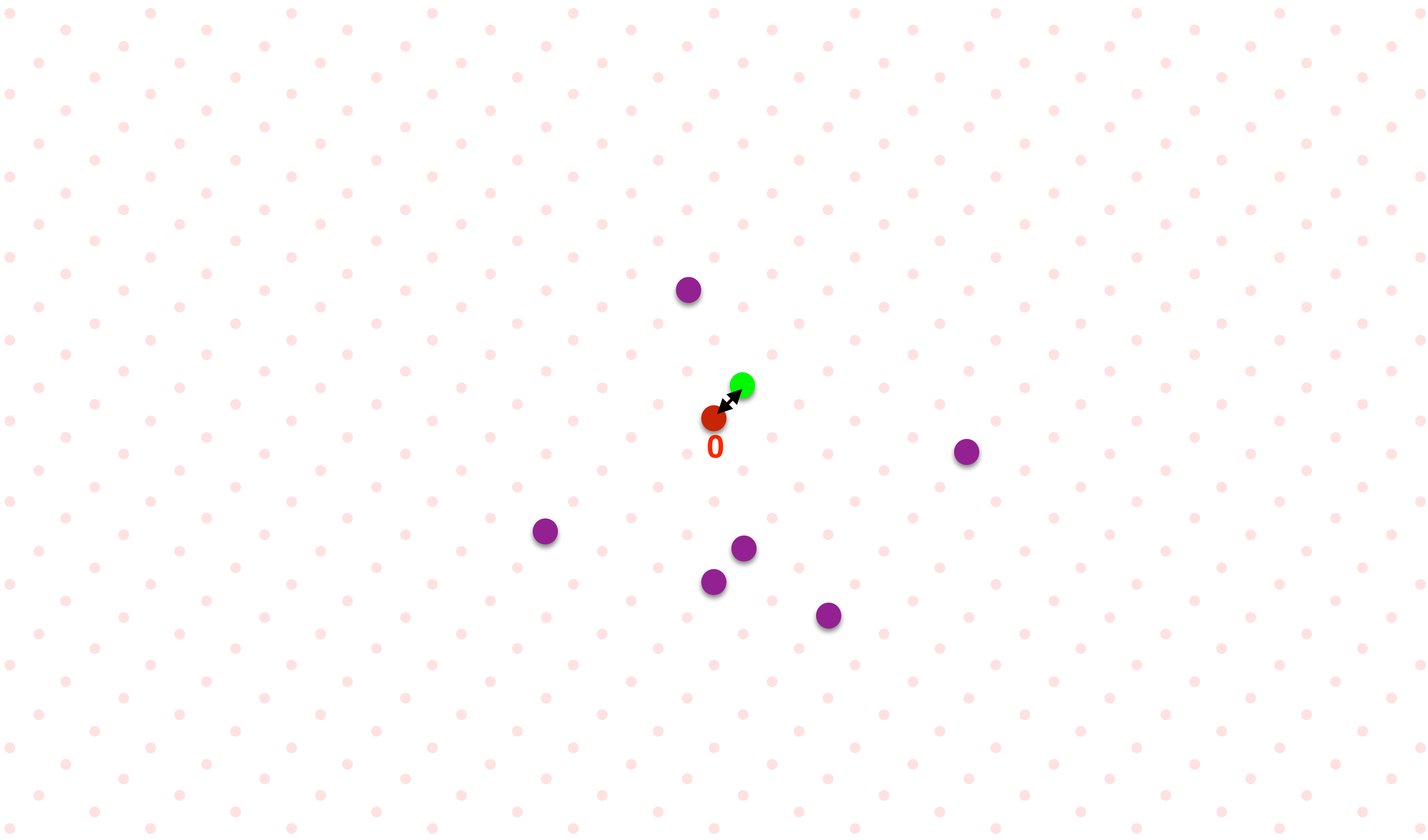
Sieving



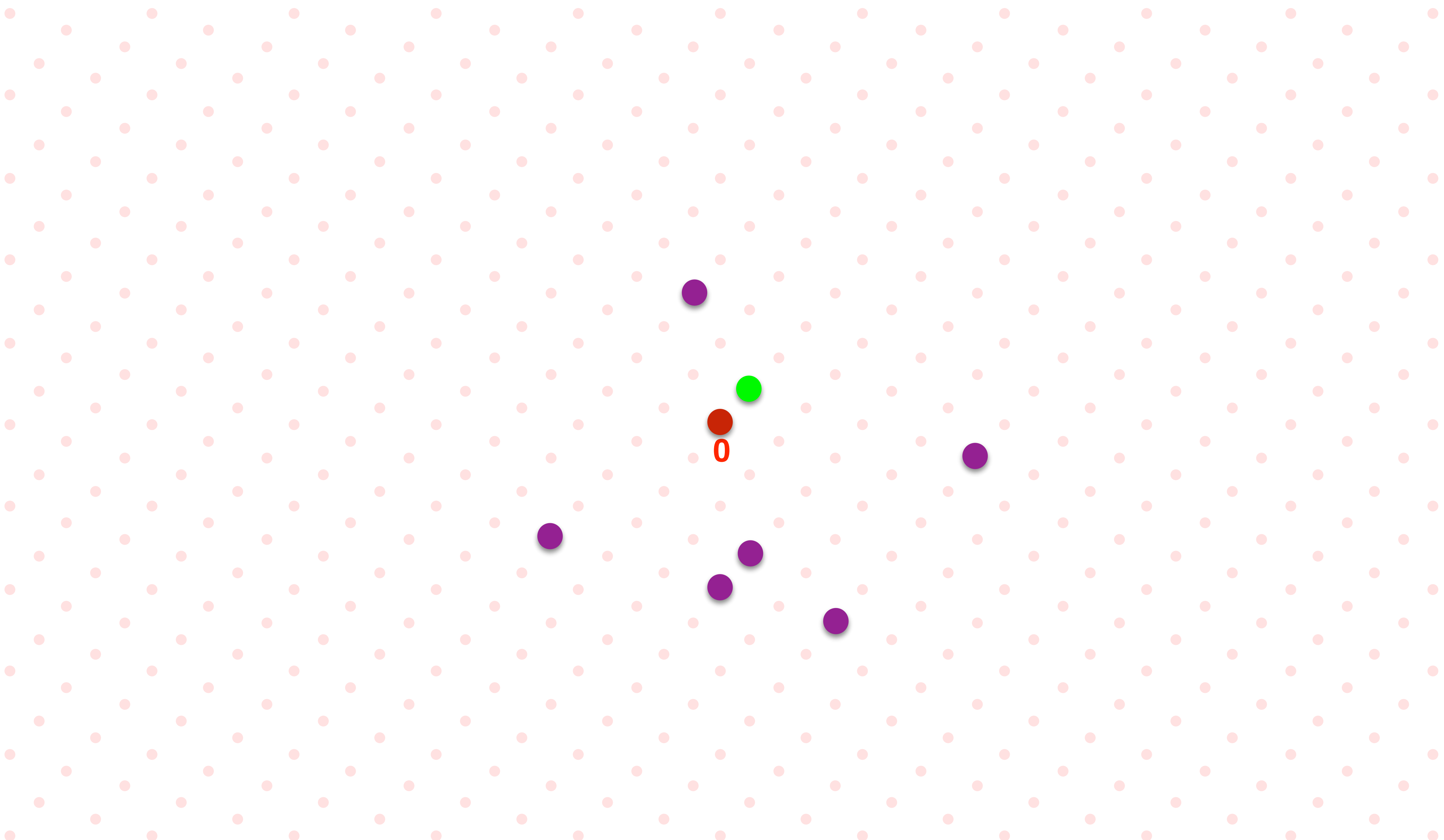
Sieving



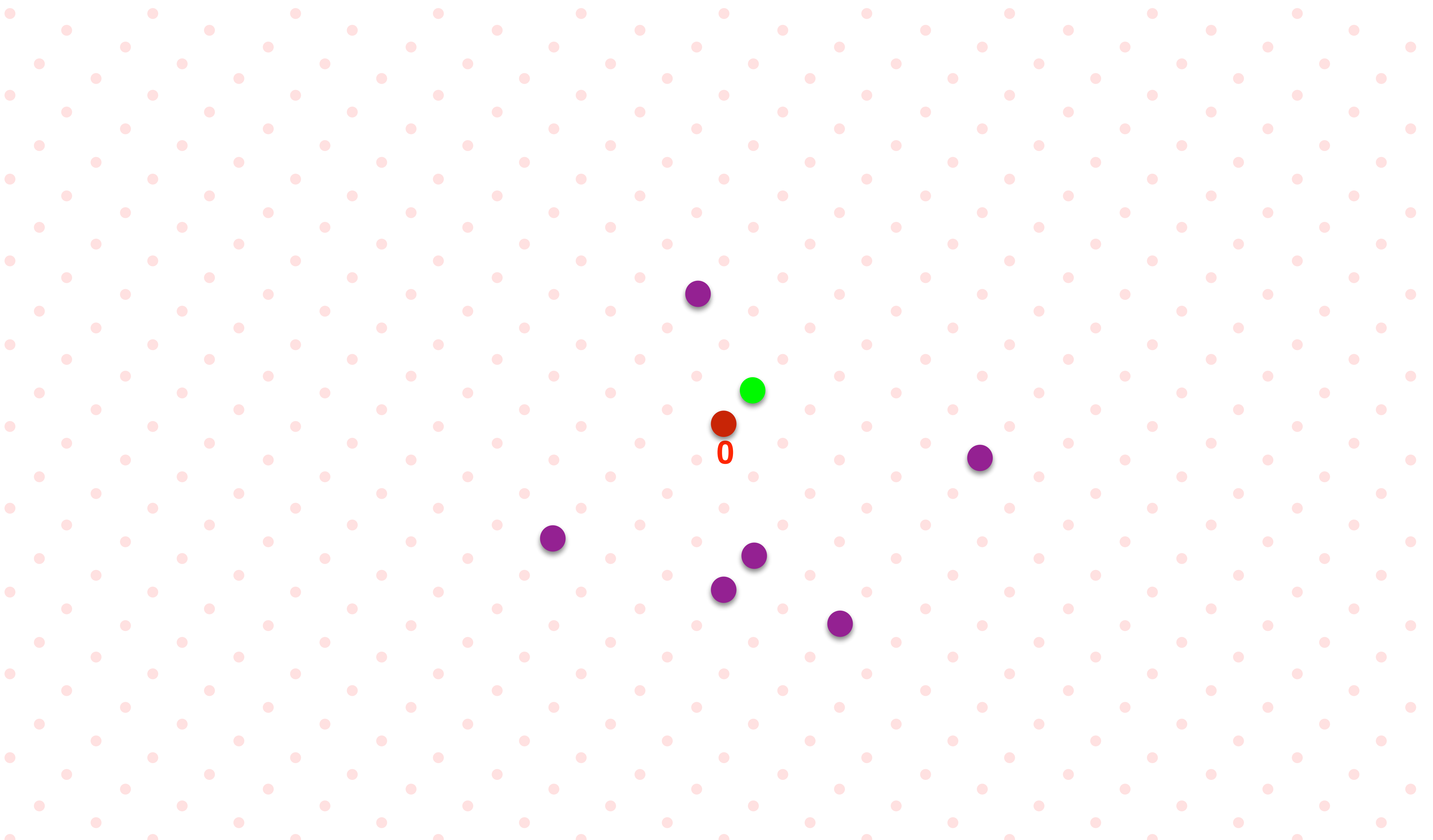
Sieving



Sieving



Sieving



Sieving

Sieving

Pros

Sieving

Pros

- Very natural technique.
- Clearly “makes progress” at each step.
- Can be made to work.

Sieving

Pros

- Very natural technique.
- Clearly “makes progress” at each step.
- Can be made to work.

Cons

Sieving

Pros

- Very natural technique.
- Clearly “makes progress” at each step.
- Can be made to work.

Cons

- Hard to analyze.
- What is the distribution of the vectors at each step?
- How common are collisions?

Sieving Algorithms

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]
Perturbation	$2^{2.5n}$	[NV08, PS09, MV10, ...]

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]
Perturbation	$2^{2.5n}$	[NV08, PS09, MV10, …]
Heuristic (no proof of correctness)	$(3/2)^{n/2+o(n)} \approx 2^{0.3n}$	[NV08, Laa15, BDGL15, BLS16, …]

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]
Perturbation	$2^{2.5n}$	[NV08, PS09, MV10, …]
Heuristic (no proof of correctness)	$(3/2)^{n/2+o(n)} \approx 2^{0.3n}$	[NV08, Laa15, BDGL15, BLS16, …]
Sieving by Averages (Discrete Gaussian)	$2^{n+o(n)}$	[ADRS15, AS17]

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]
Perturbation	$2^{2.5n}$	[NV08, PS09, MV10, …]
Heuristic (no proof of correctness)	$(3/2)^{n/2+o(n)} \approx 2^{0.3n}$	[NV08, Laa15, BDGL15, BLS16, …]
Sieving by Averages (Discrete Gaussian)	$2^{n+o(n)}$	[ADRS15, AS17]
Sieving by Averages (Discrete Gaussian)	$2^{n/2+o(n)}$	[ADRS15] (only approx. decision SVP)

Sieving Algorithms

Perturbation	$2^{O(n)}$	[AKS01]
Perturbation	$2^{2.5n}$	[NV08, PS09, MV10, …]
Heuristic (no proof of correctness)	$(3/2)^{n/2+o(n)} \approx 2^{0.3n}$	[NV08, Laa15, BDGL15, BLS16, …]
Sieving by Averages (Discrete Gaussian)	$2^{n+o(n)}$	[ADRS15, AS17]
Sieving by Averages (Discrete Gaussian)	$2^{n/2+o(n)}$	[ADRS15] (only approx. decision SVP)
????	Fast!	[FuturePeople18]

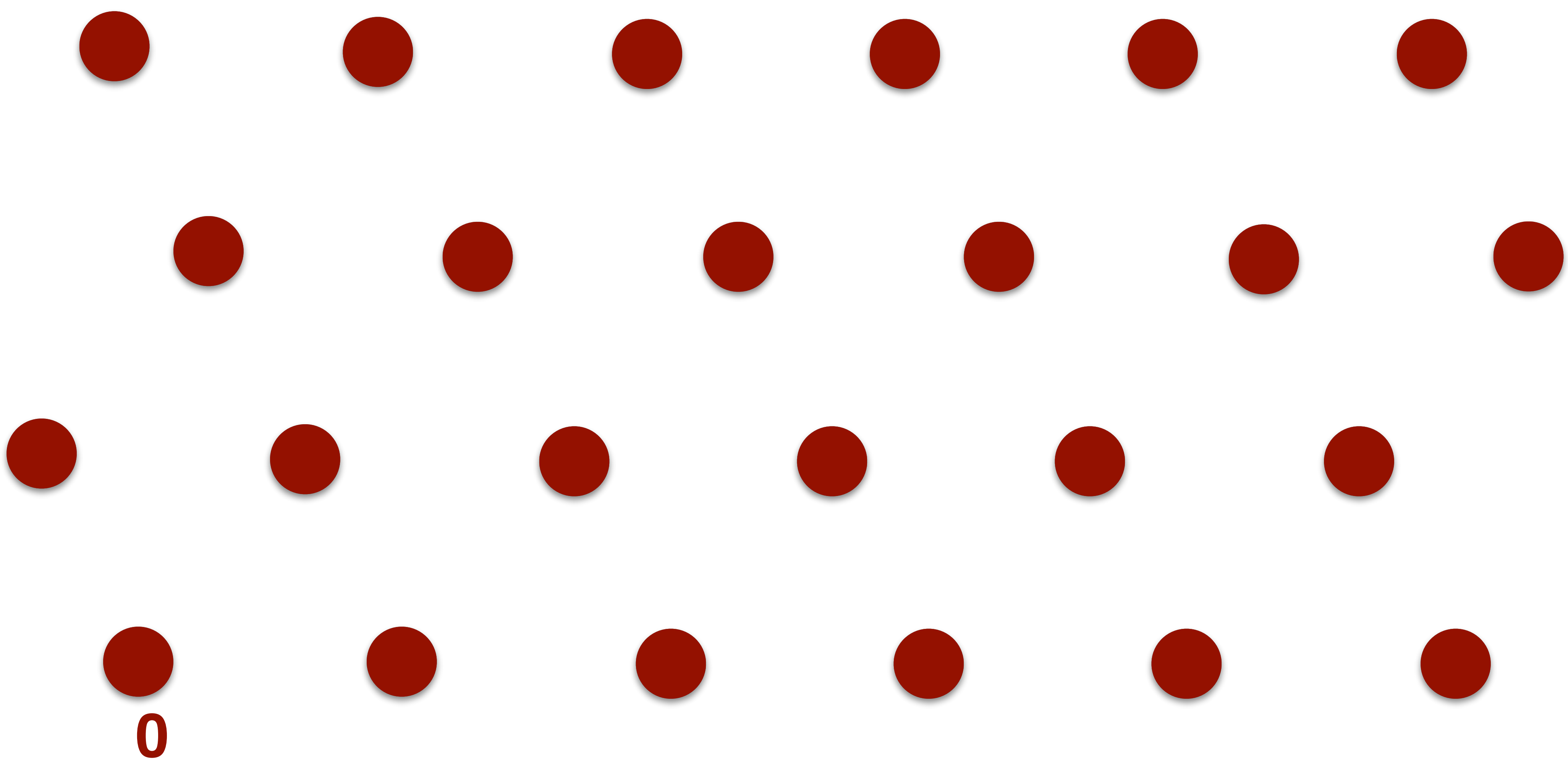
Sieving by Averages

Sieving by Averages

Suppose that, instead of taking the *difference* $\mathbf{x} - \mathbf{y}$ of pairs of lattice vectors, we take the *average* $\frac{\mathbf{x} + \mathbf{y}}{2}$.

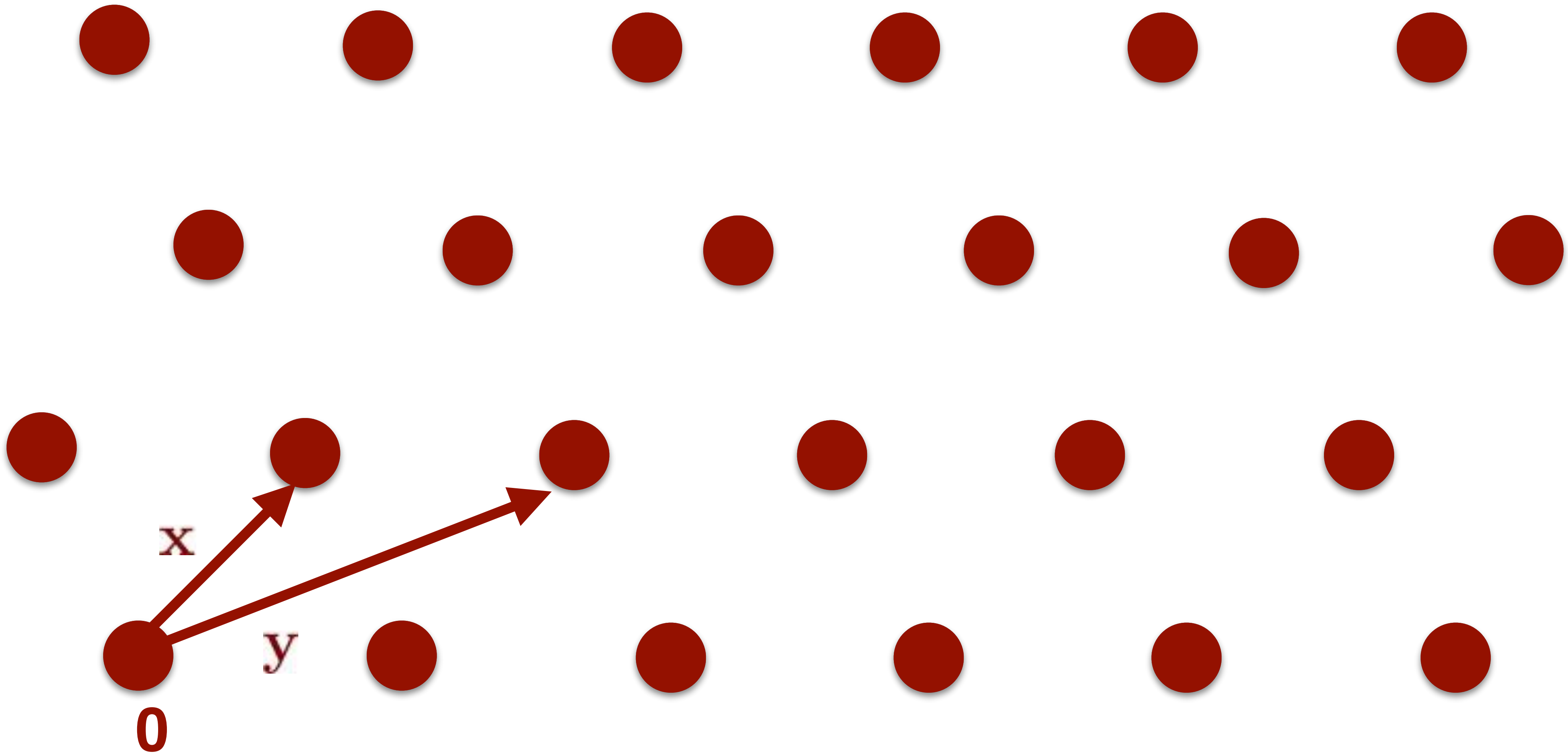
Sieving by Averages

Suppose that, instead of taking the *difference* $\mathbf{x} - \mathbf{y}$ of pairs of lattice vectors, we take the *average* $\frac{\mathbf{x} + \mathbf{y}}{2}$.



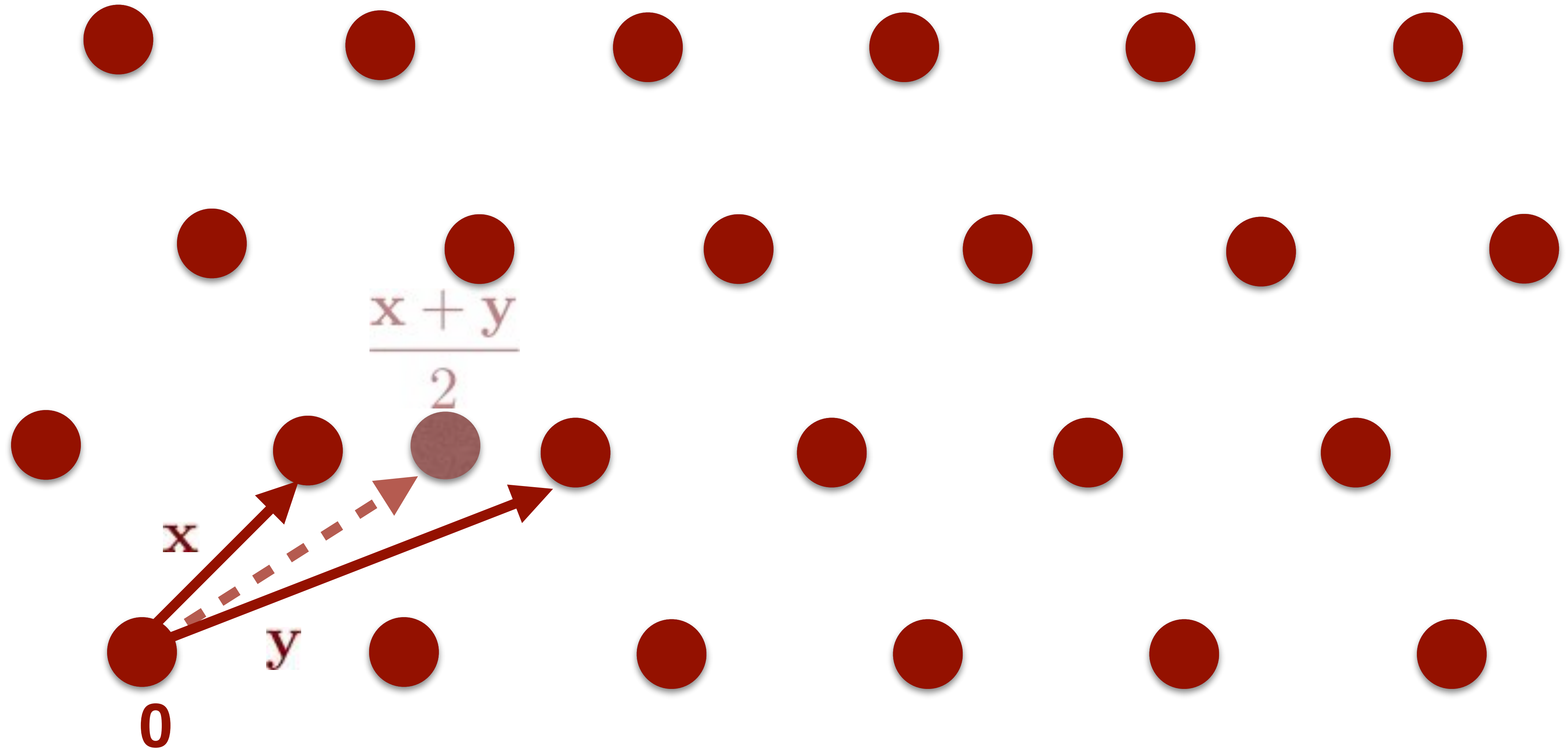
Sieving by Averages

Suppose that, instead of taking the *difference* $\mathbf{x} - \mathbf{y}$ of pairs of lattice vectors, we take the *average* $\frac{\mathbf{x}+\mathbf{y}}{2}$.



Sieving by Averages

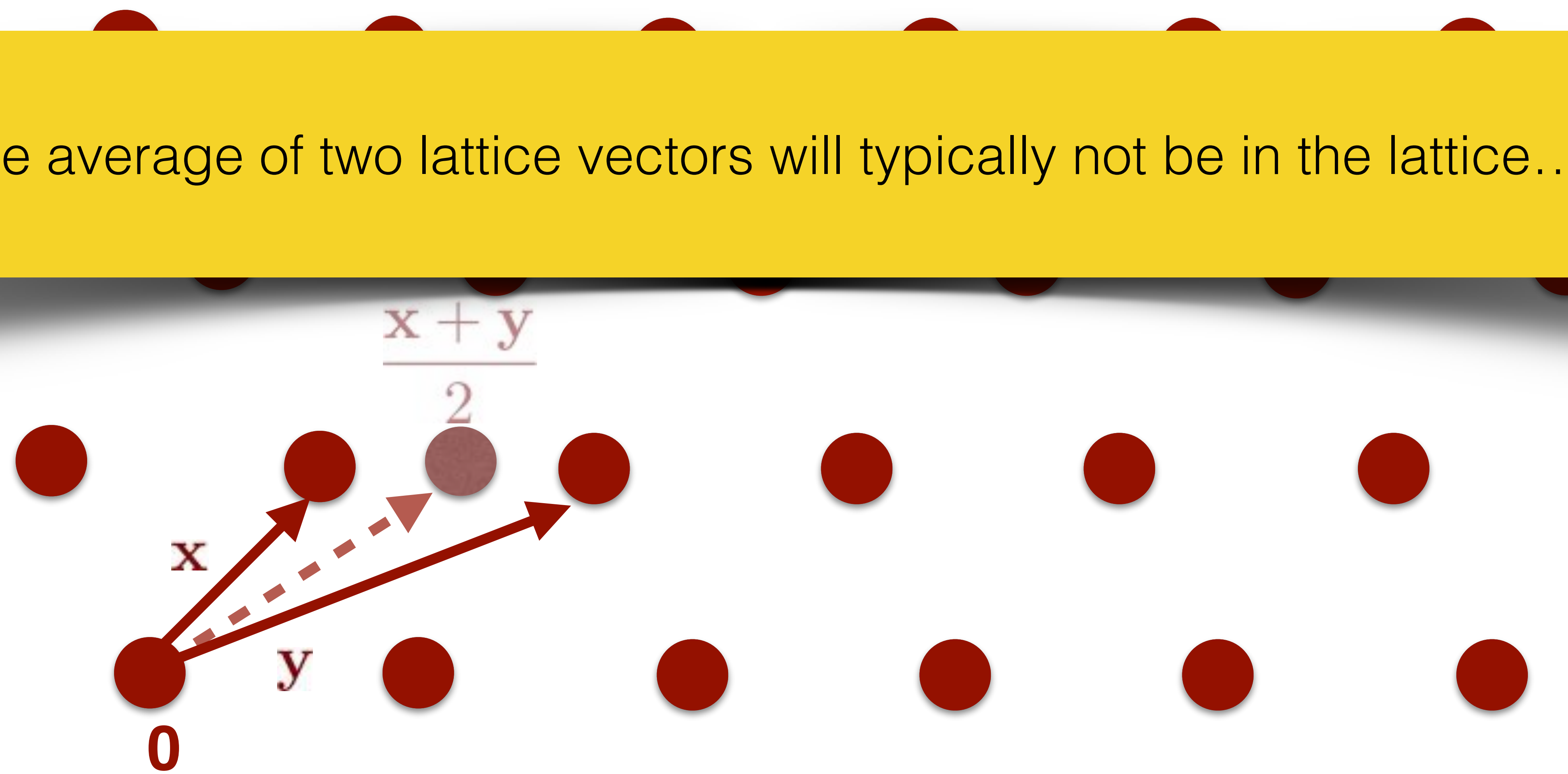
Suppose that, instead of taking the *difference* $\mathbf{x} - \mathbf{y}$ of pairs of lattice vectors, we take the *average* $\frac{\mathbf{x}+\mathbf{y}}{2}$.



Sieving by Averages

Suppose that, instead of taking the *difference* $\mathbf{x} - \mathbf{y}$ of pairs of lattice vectors, we take the *average* $\frac{\mathbf{x}+\mathbf{y}}{2}$.

The average of two lattice vectors will typically not be in the lattice…



Sieving by Averages

Sieving by Averages

When do we have $\dfrac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

Sieving by Averages

When do we have $\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n$$

Sieving by Averages

When do we have $\dfrac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n \qquad \mathbf{y}_2 = a_{2,1}\mathbf{b}_1 + \cdots + a_{2,n}\mathbf{b}_n$$

Sieving by Averages

When do we have $\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n \qquad \mathbf{y}_2 = a_{2,1}\mathbf{b}_1 + \cdots + a_{2,n}\mathbf{b}_n$$

$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} = \frac{a_{1,1} + a_{2,1}}{2} \cdot \mathbf{b}_1 + \cdots + \frac{a_{1,n} + a_{2,n}}{2} \cdot \mathbf{b}_n$$

Sieving by Averages

When do we have $\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n \qquad \mathbf{y}_2 = a_{2,1}\mathbf{b}_1 + \cdots + a_{2,n}\mathbf{b}_n$$

$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} = \frac{a_{1,1} + a_{2,1}}{2} \cdot \mathbf{b}_1 + \cdots + \frac{a_{1,n} + a_{2,n}}{2} \cdot \mathbf{b}_n$$

$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L} \iff a_{1,i} \equiv a_{2,i} \mod 2$$

Sieving by Averages

When do we have $\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n \qquad \mathbf{y}_2 = a_{2,1}\mathbf{b}_1 + \cdots + a_{2,n}\mathbf{b}_n$$

$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} = \frac{a_{1,1} + a_{2,1}}{2} \cdot \mathbf{b}_1 + \cdots + \frac{a_{1,n} + a_{2,n}}{2} \cdot \mathbf{b}_n$$

$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L} \iff a_{1,i} \equiv a_{2,i} \mod 2$$

$$\iff \mathbf{y}_1 \equiv \mathbf{y}_2 \mod 2\mathcal{L}$$

Sieving by Averages

When do we have $\dfrac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$?

$$\mathbf{y}_1 = a_{1,1}\mathbf{b}_1 + \cdots + a_{1,n}\mathbf{b}_n \qquad \mathbf{y}_2 = a_{2,1}\mathbf{b}_1 + \cdots + a_{2,n}\mathbf{b}_n$$

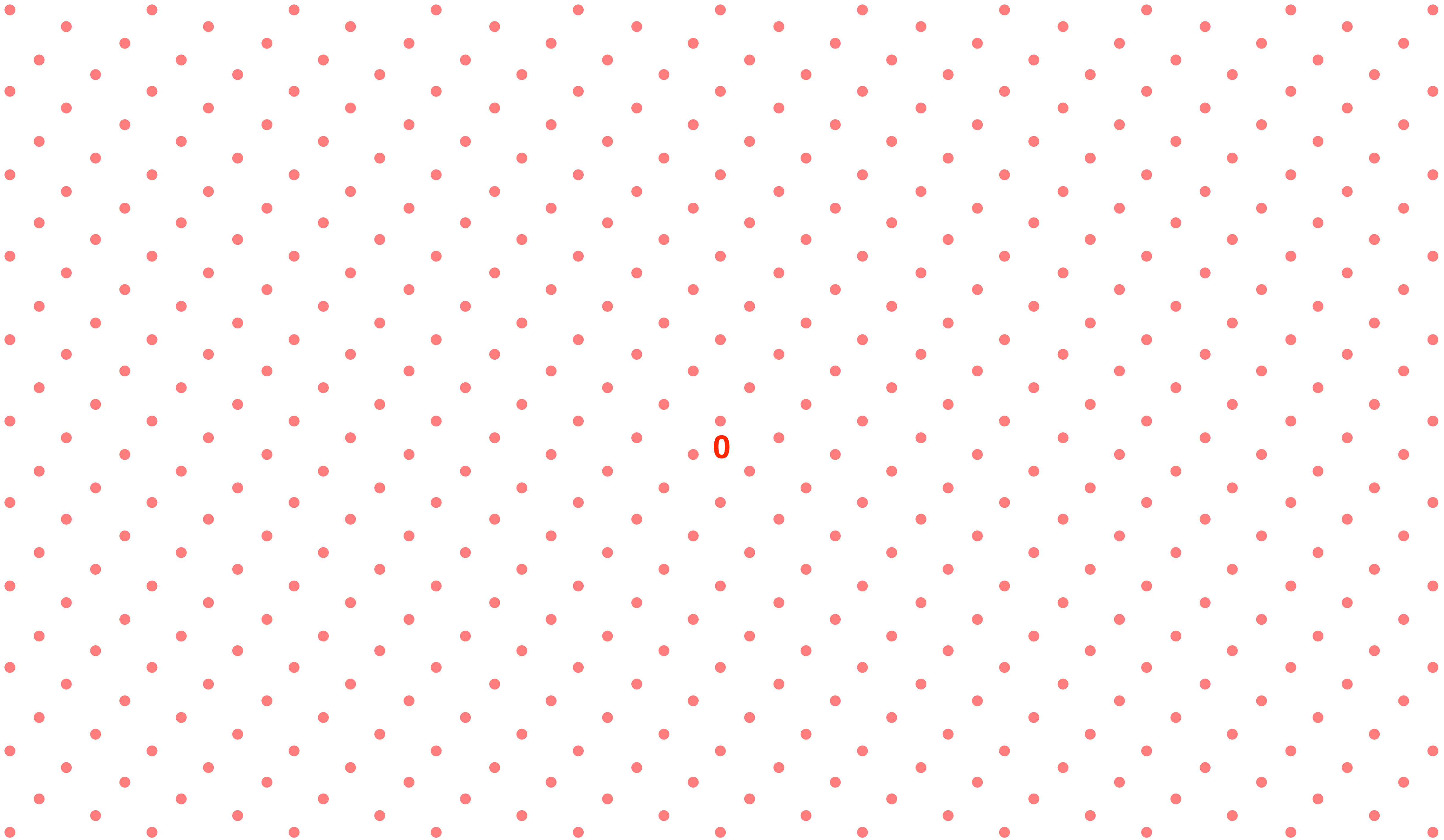
We have $\dfrac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L}$ if and only if $\mathbf{y}_1, \mathbf{y}_2$ are in the same ***coset*** *of $2\mathcal{L}$.*

(Note that there are 2^n cosets.)

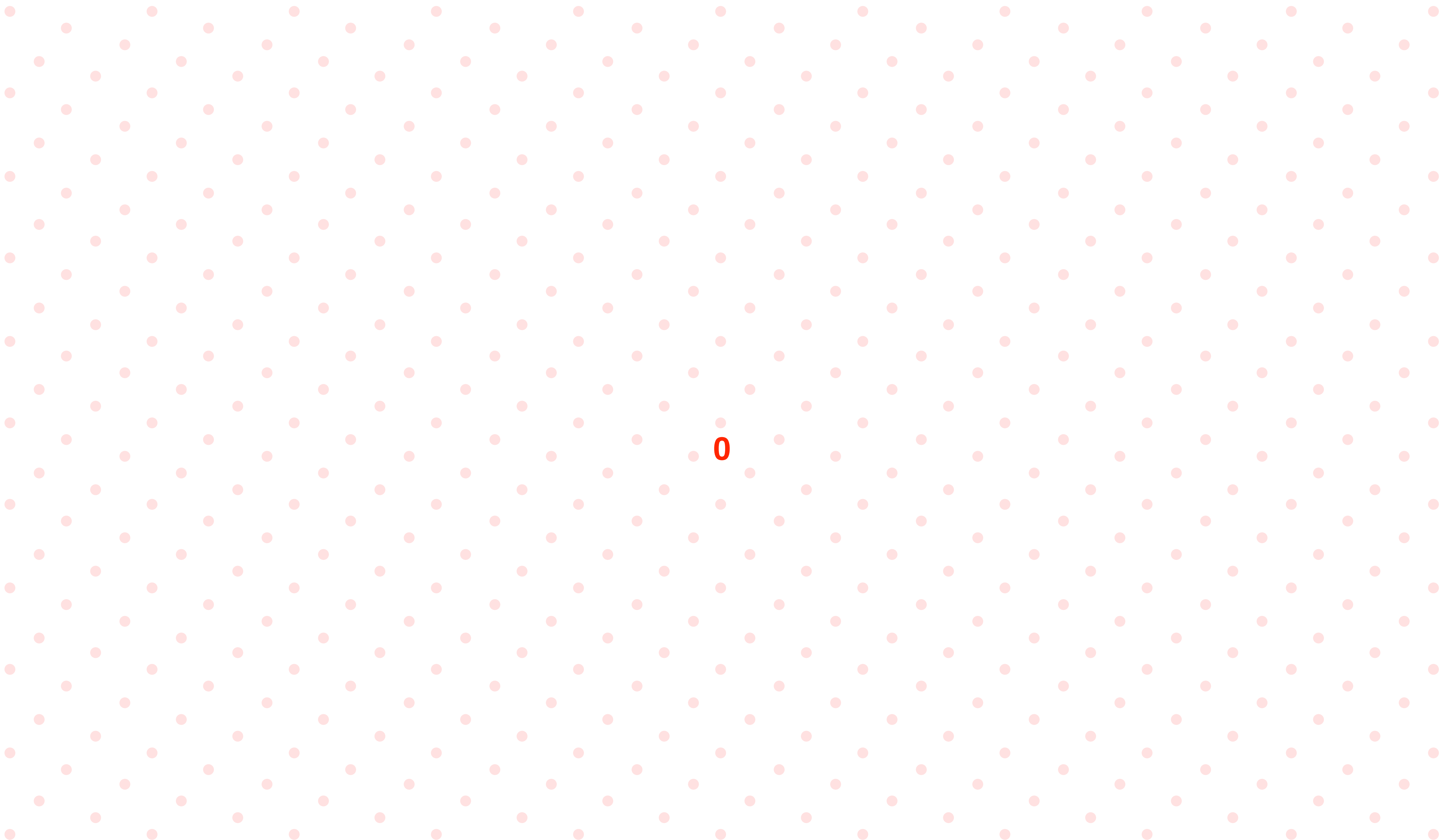
$$\frac{\mathbf{y}_1 + \mathbf{y}_2}{2} \in \mathcal{L} \iff a_{1,i} \equiv a_{2,i} \mod 2$$

$$\iff \mathbf{y}_1 \equiv \mathbf{y}_2 \mod 2\mathcal{L}$$

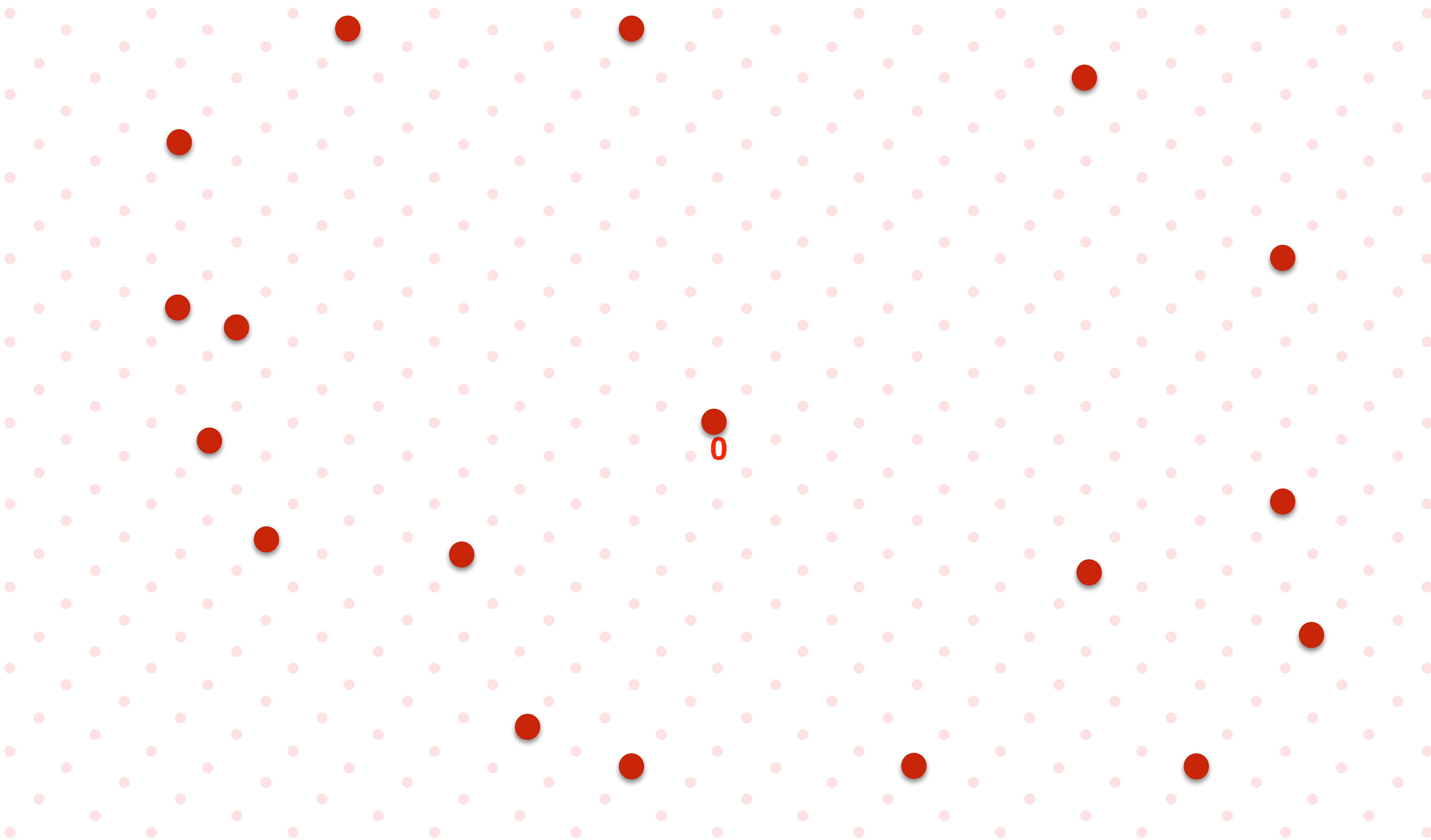
Sieving by Averages



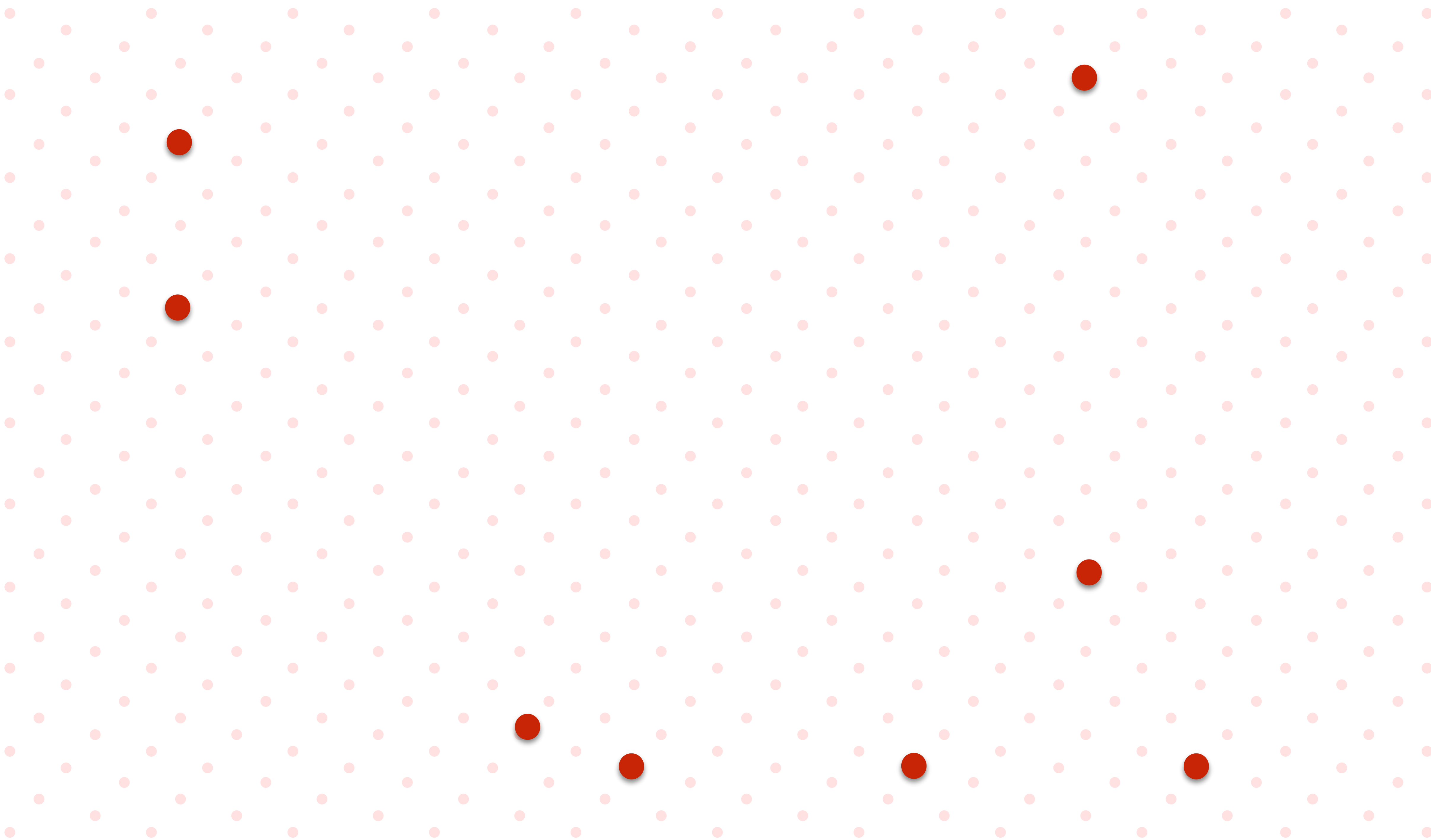
Sieving by Averages



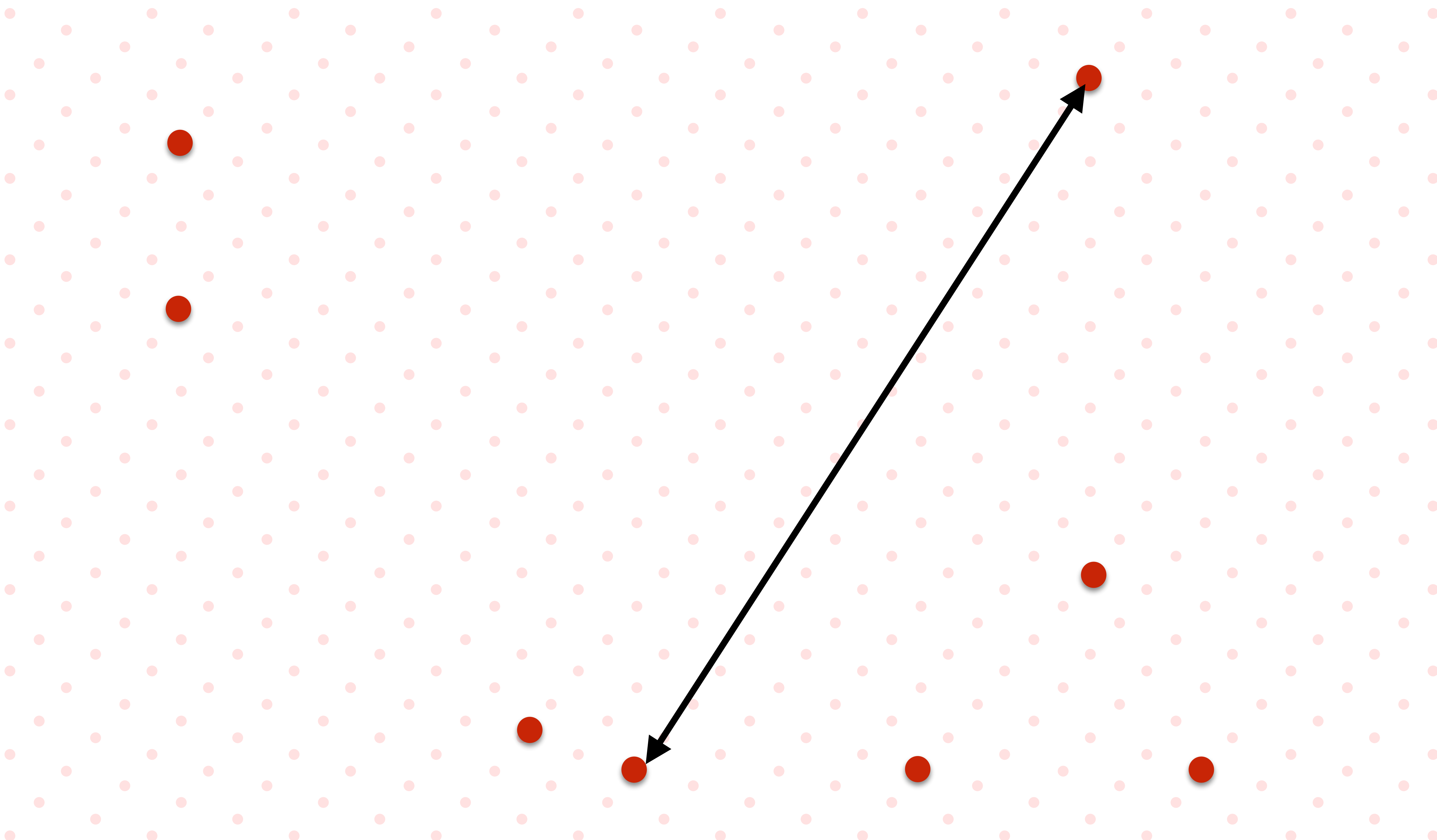
Sieving by Averages



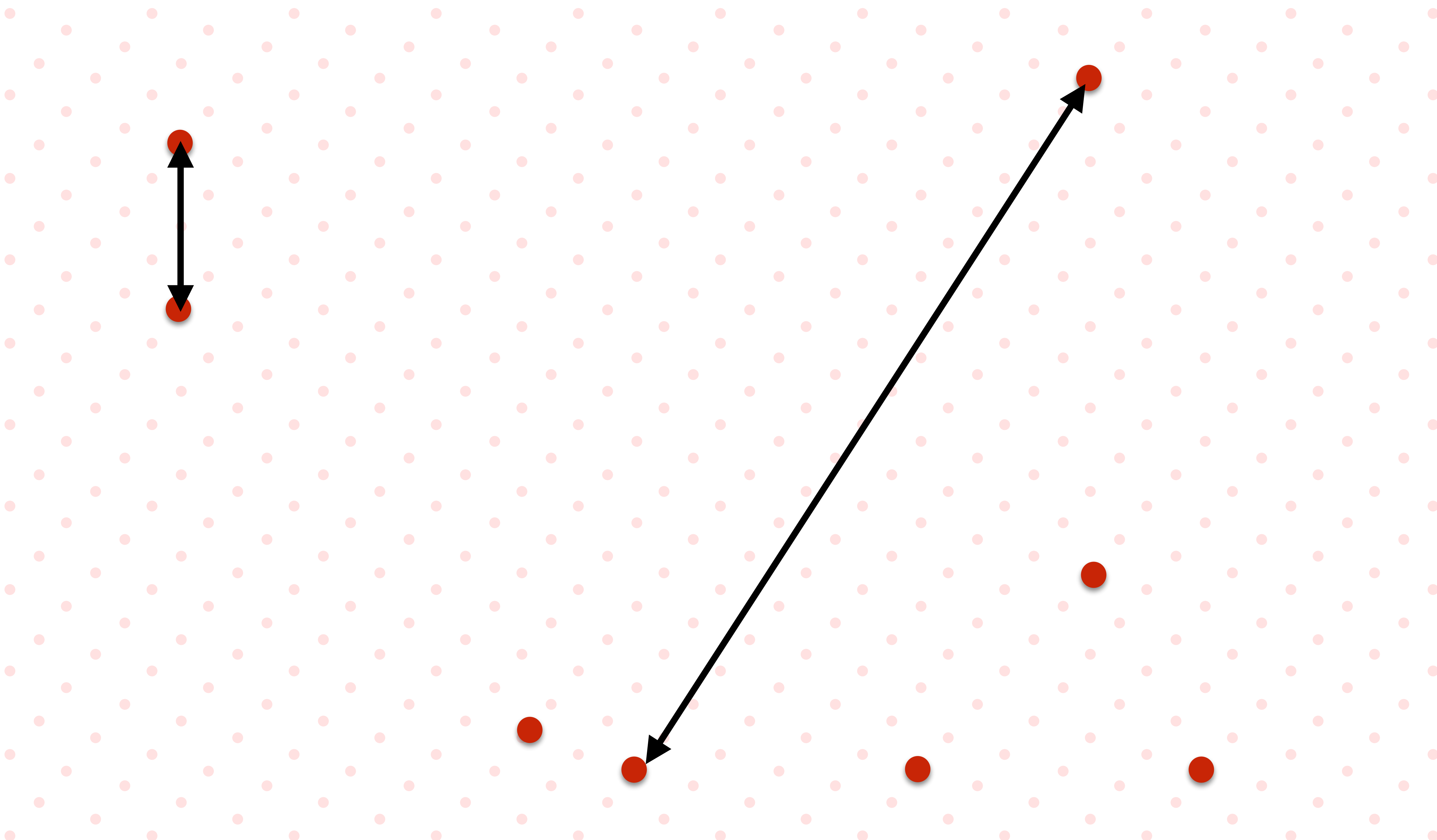
Sieving by Averages



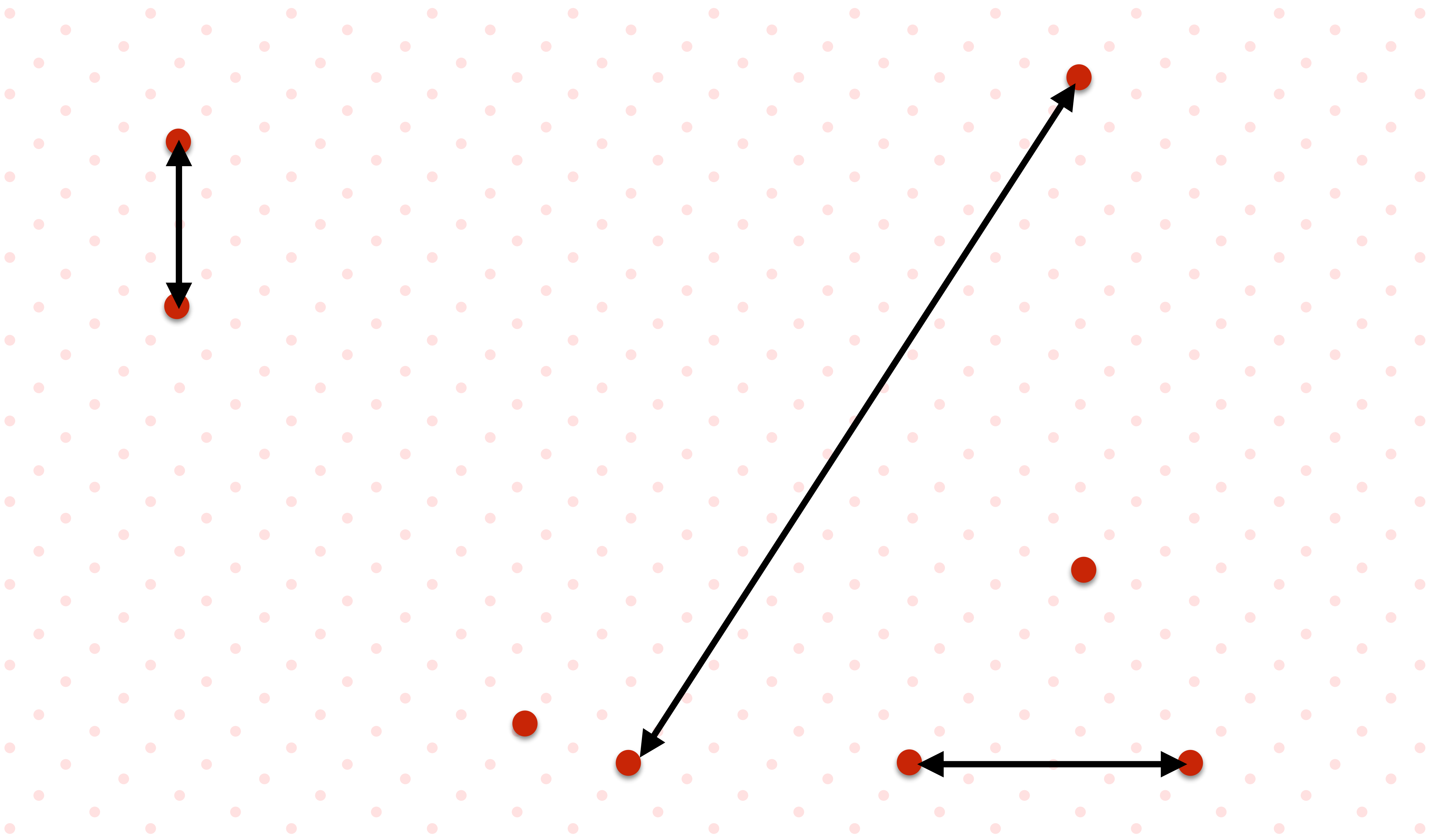
Sieving by Averages



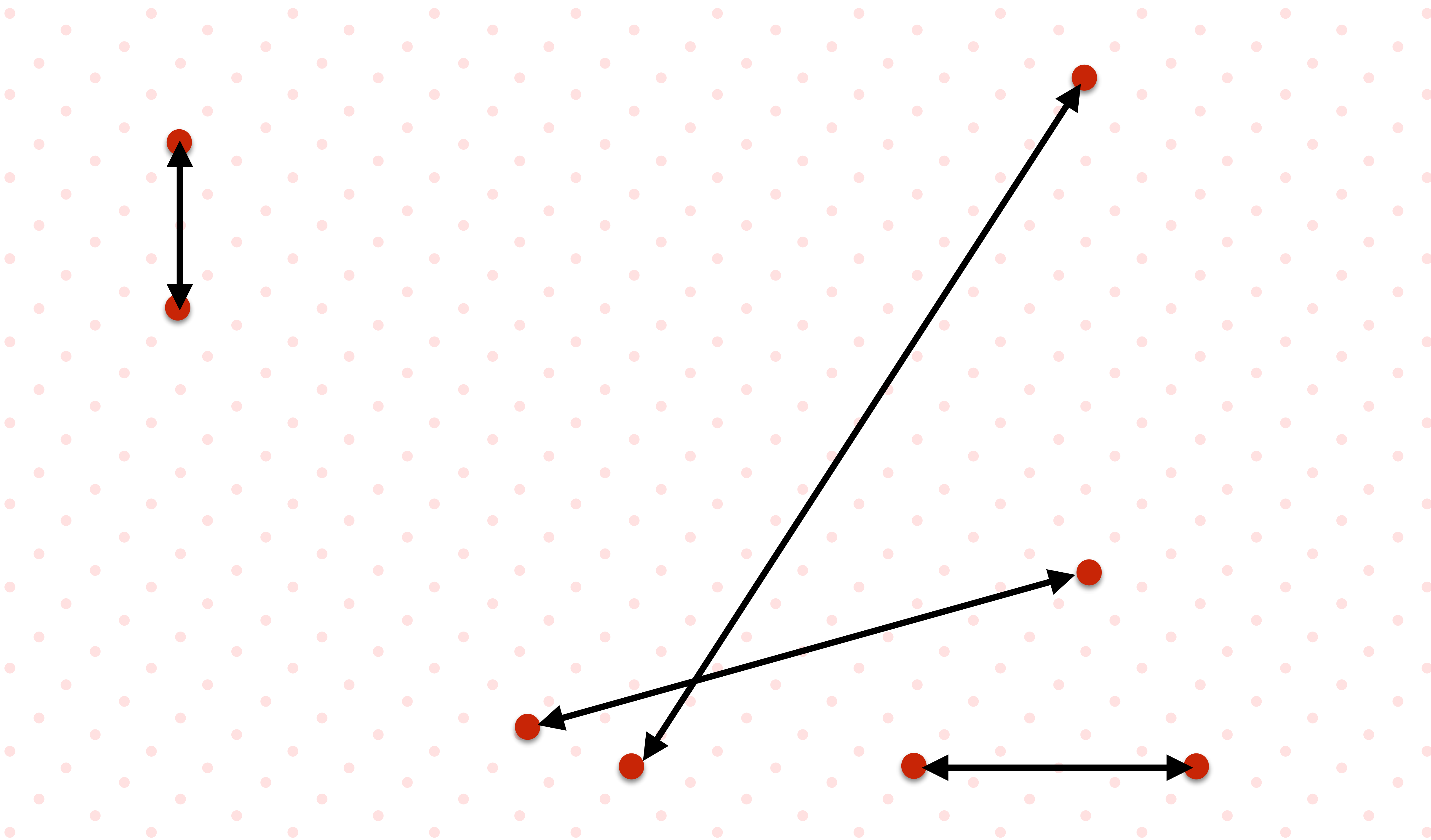
Sieving by Averages



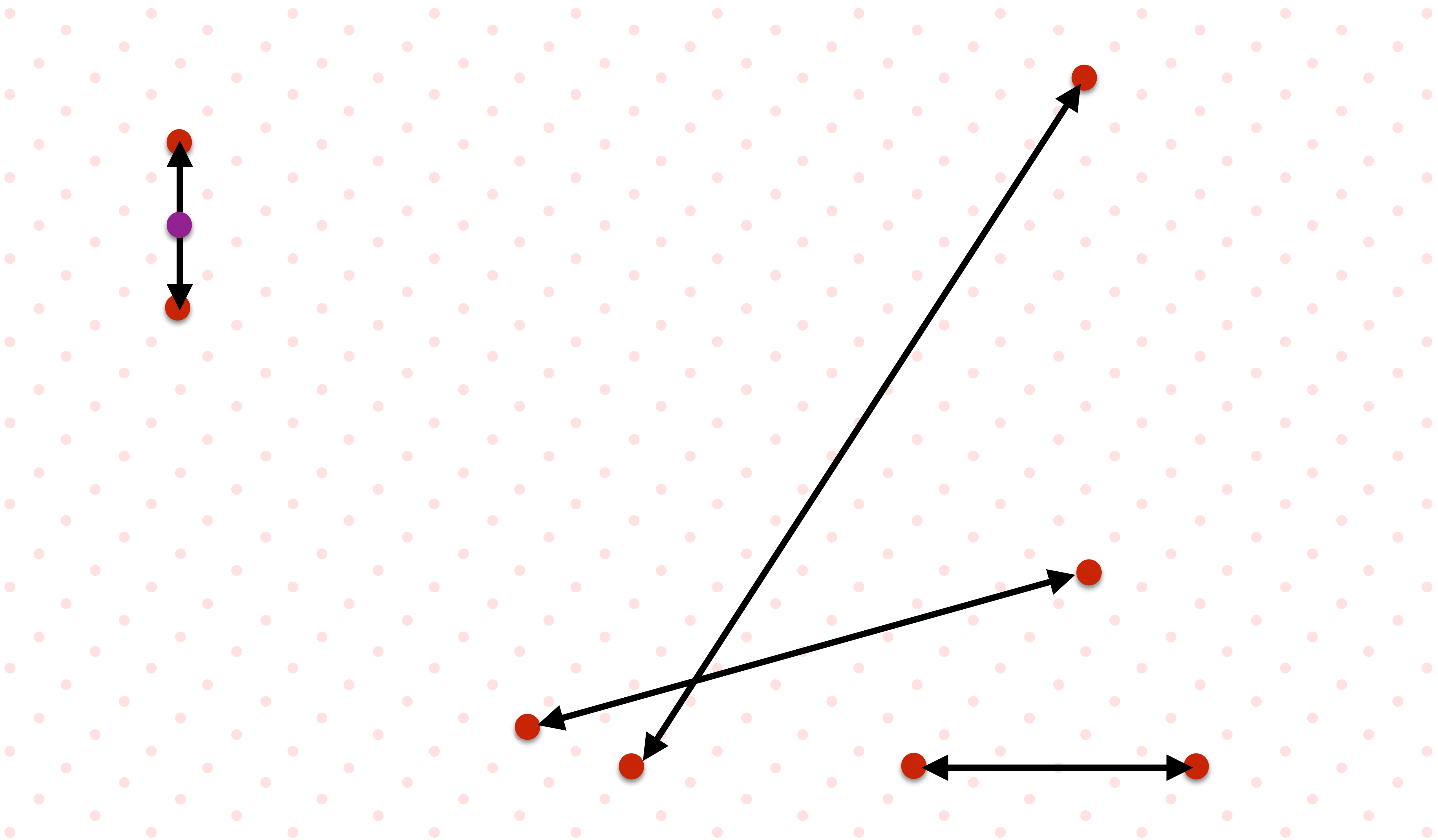
Sieving by Averages



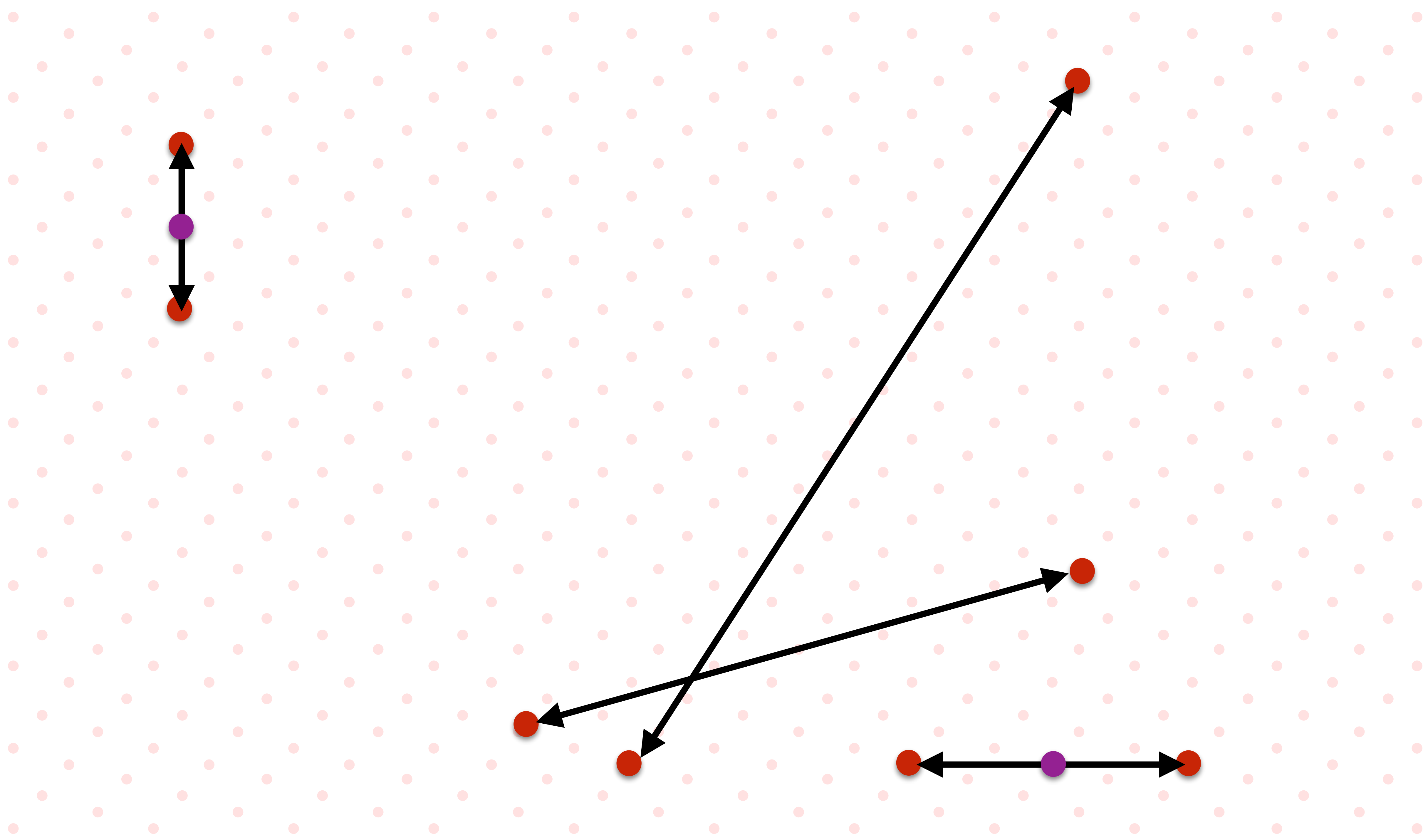
Sieving by Averages



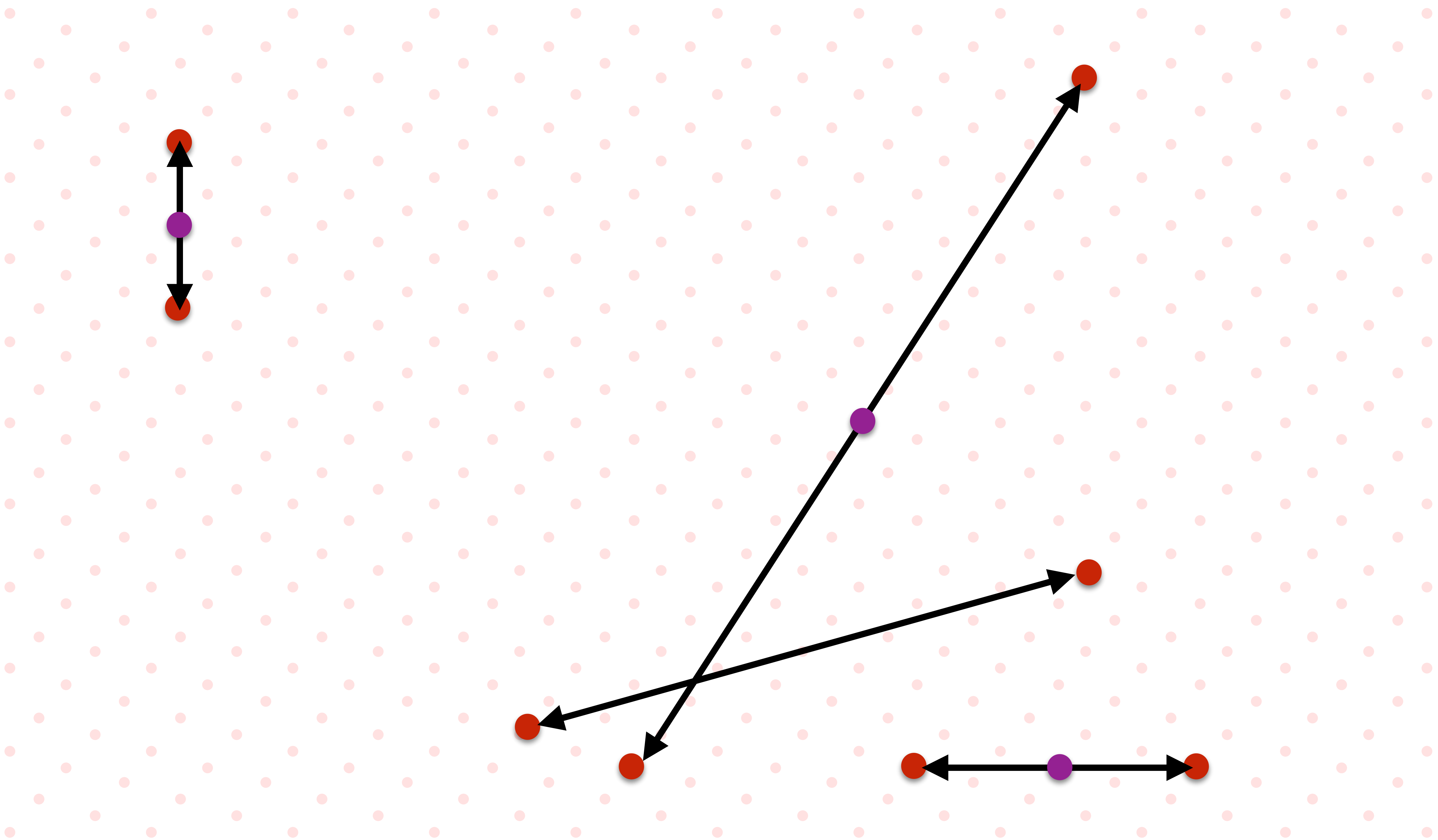
Sieving by Averages



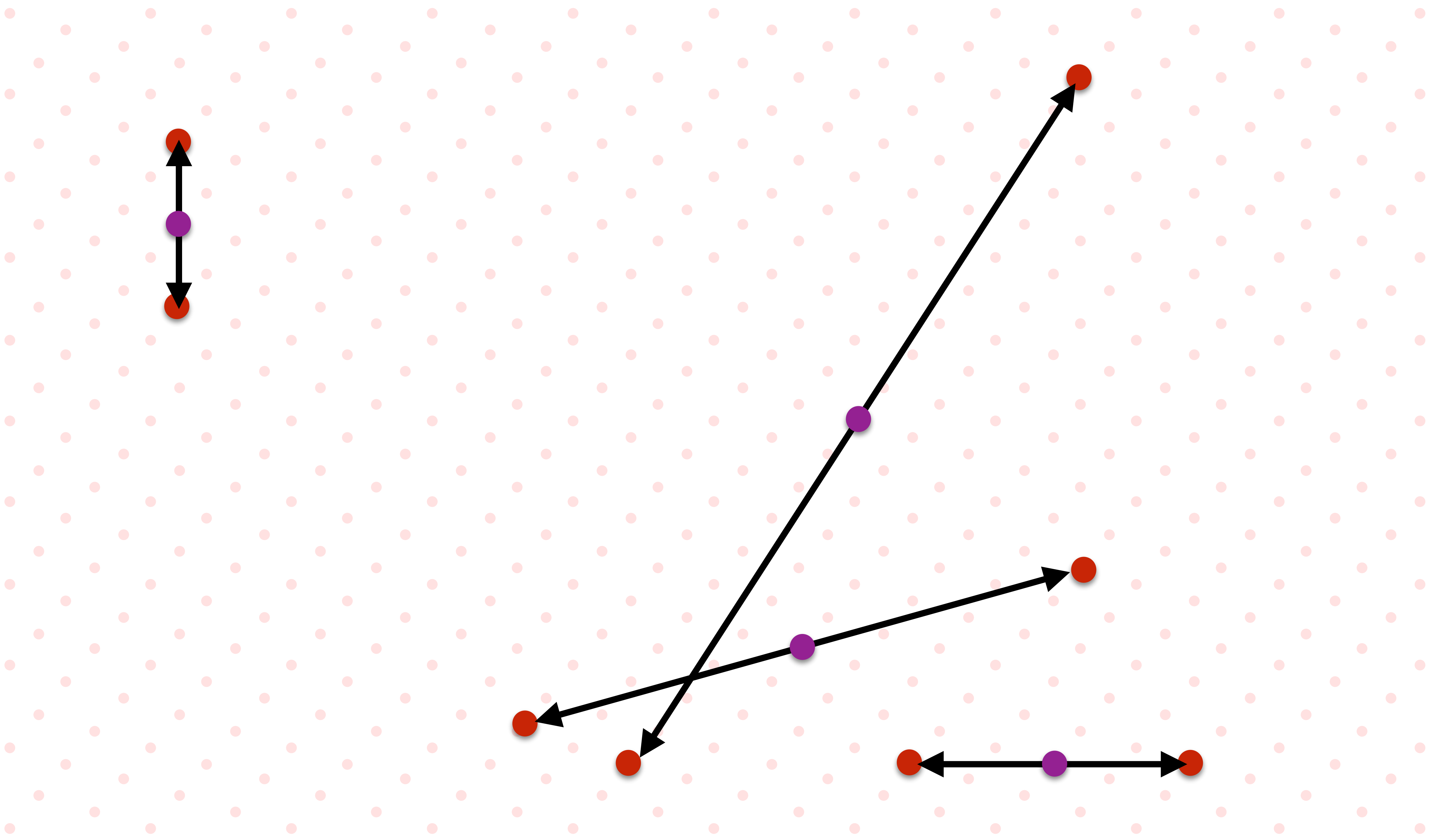
Sieving by Averages



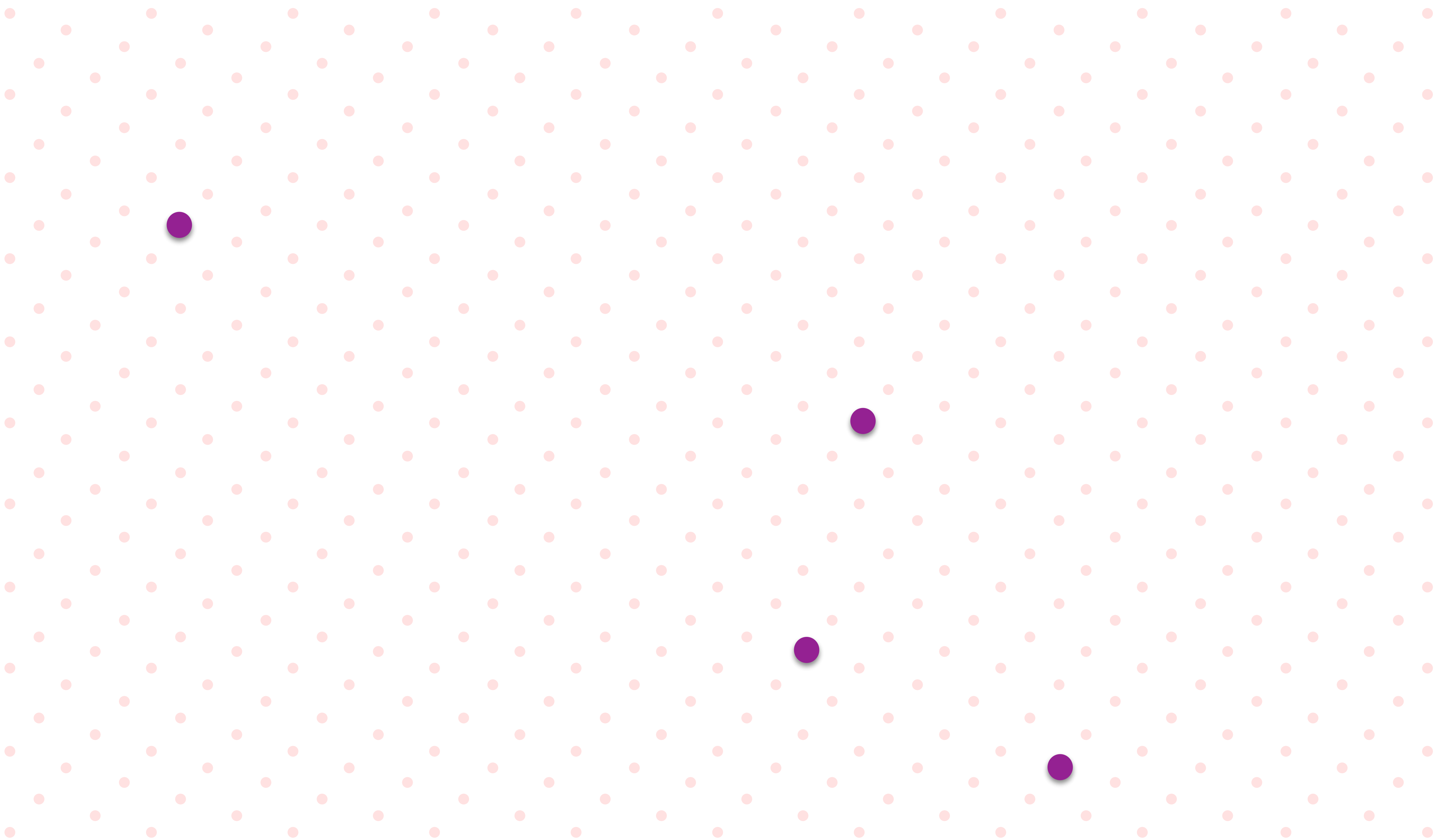
Sieving by Averages



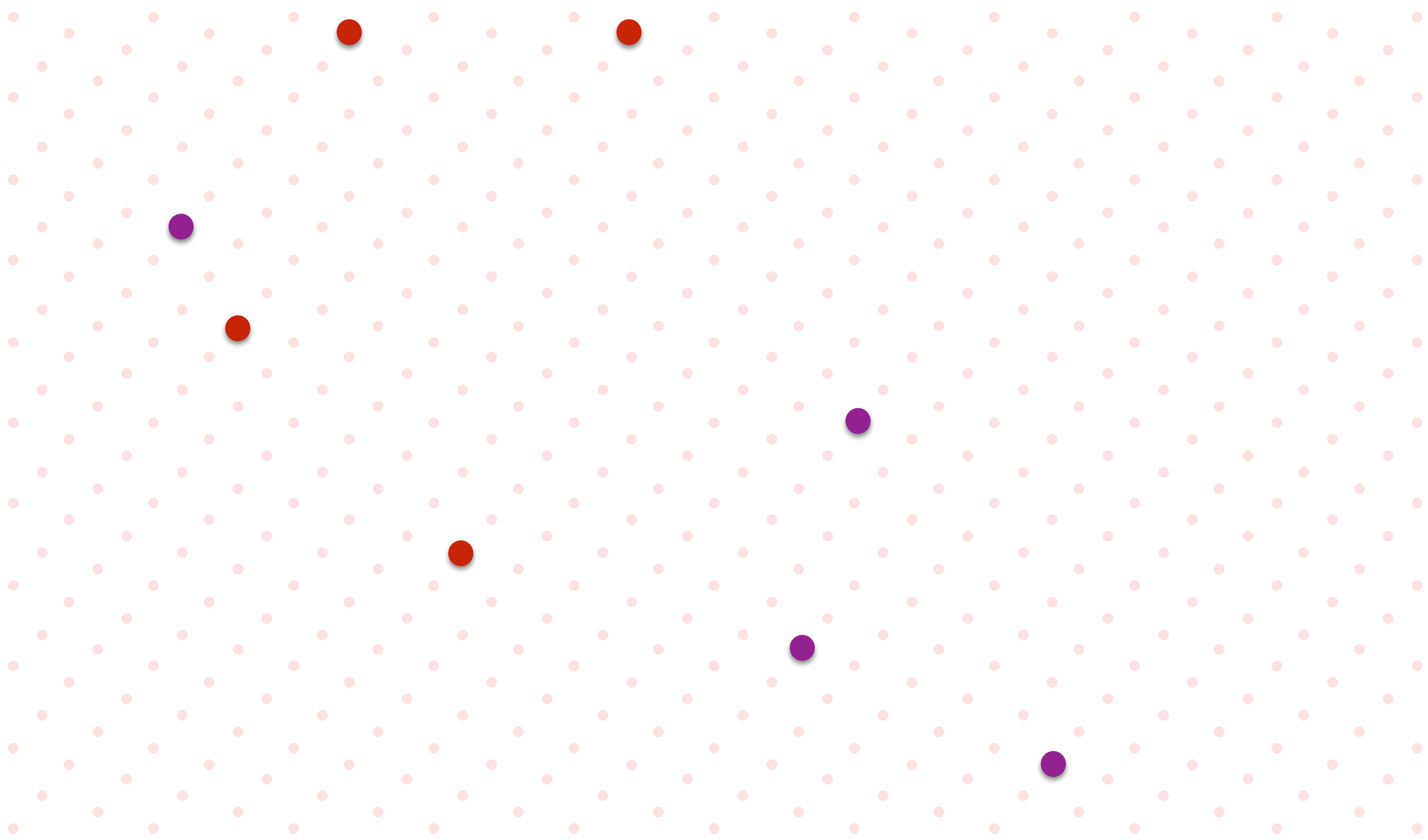
Sieving by Averages



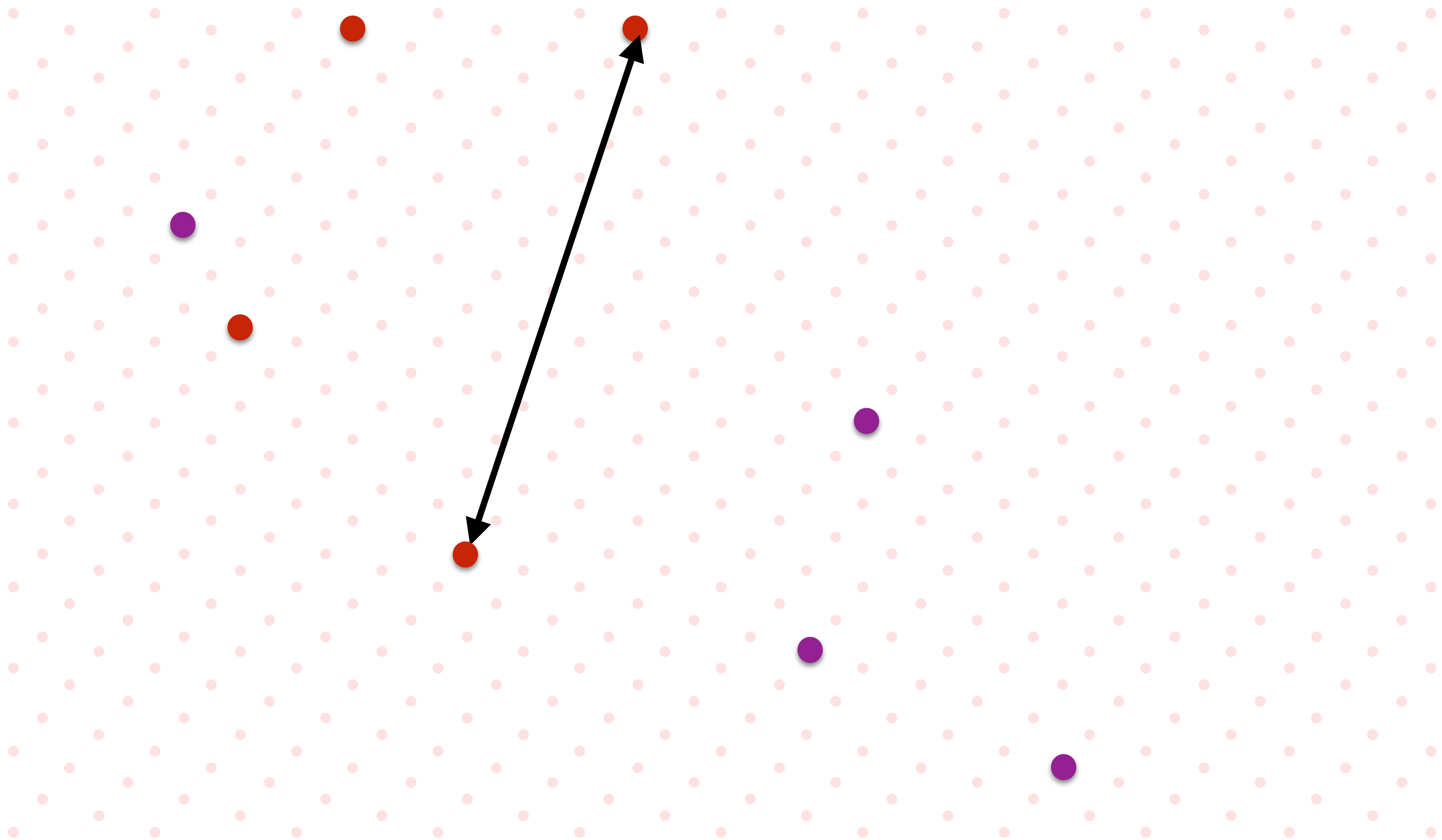
Sieving by Averages



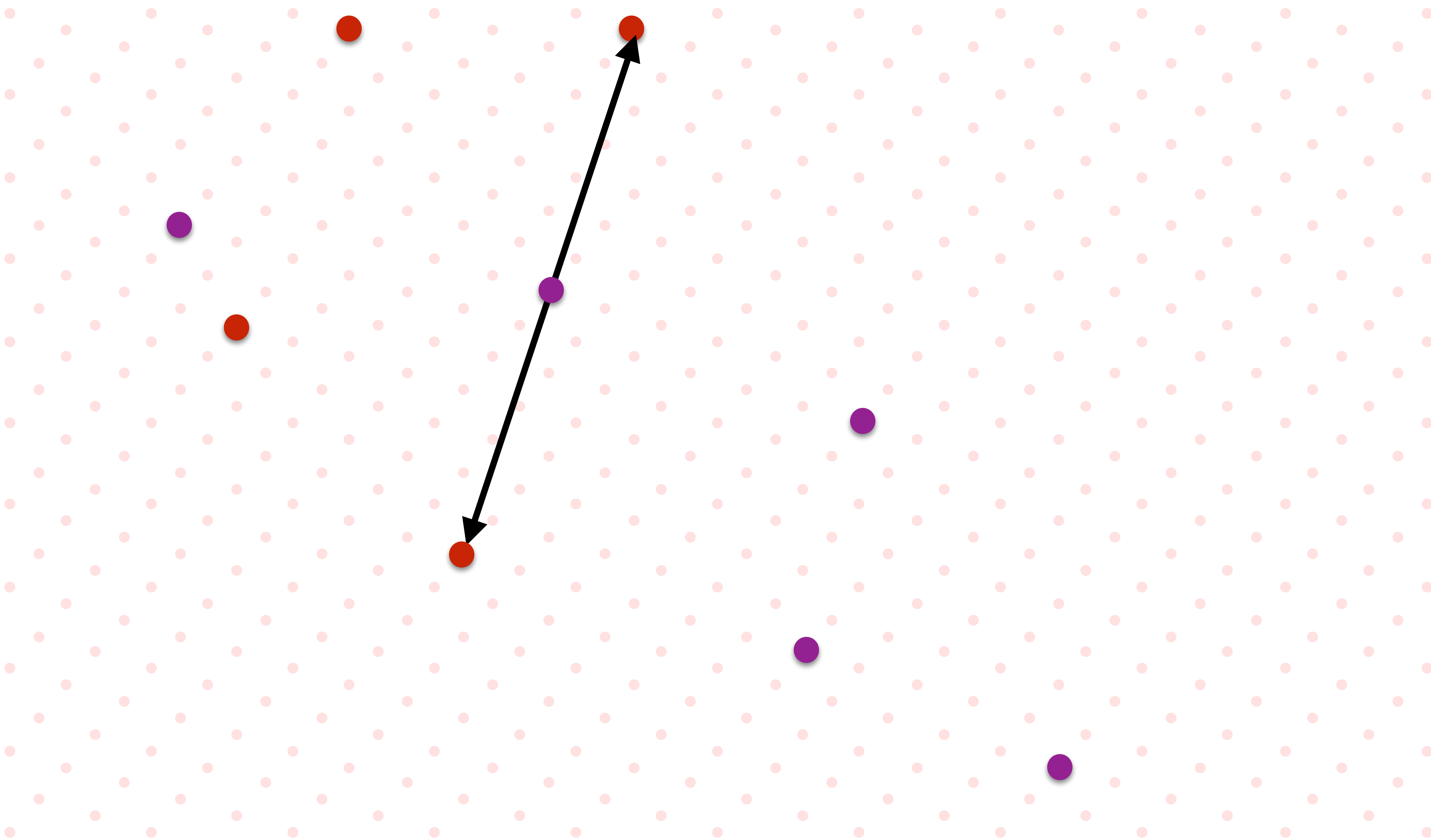
Sieving by Averages



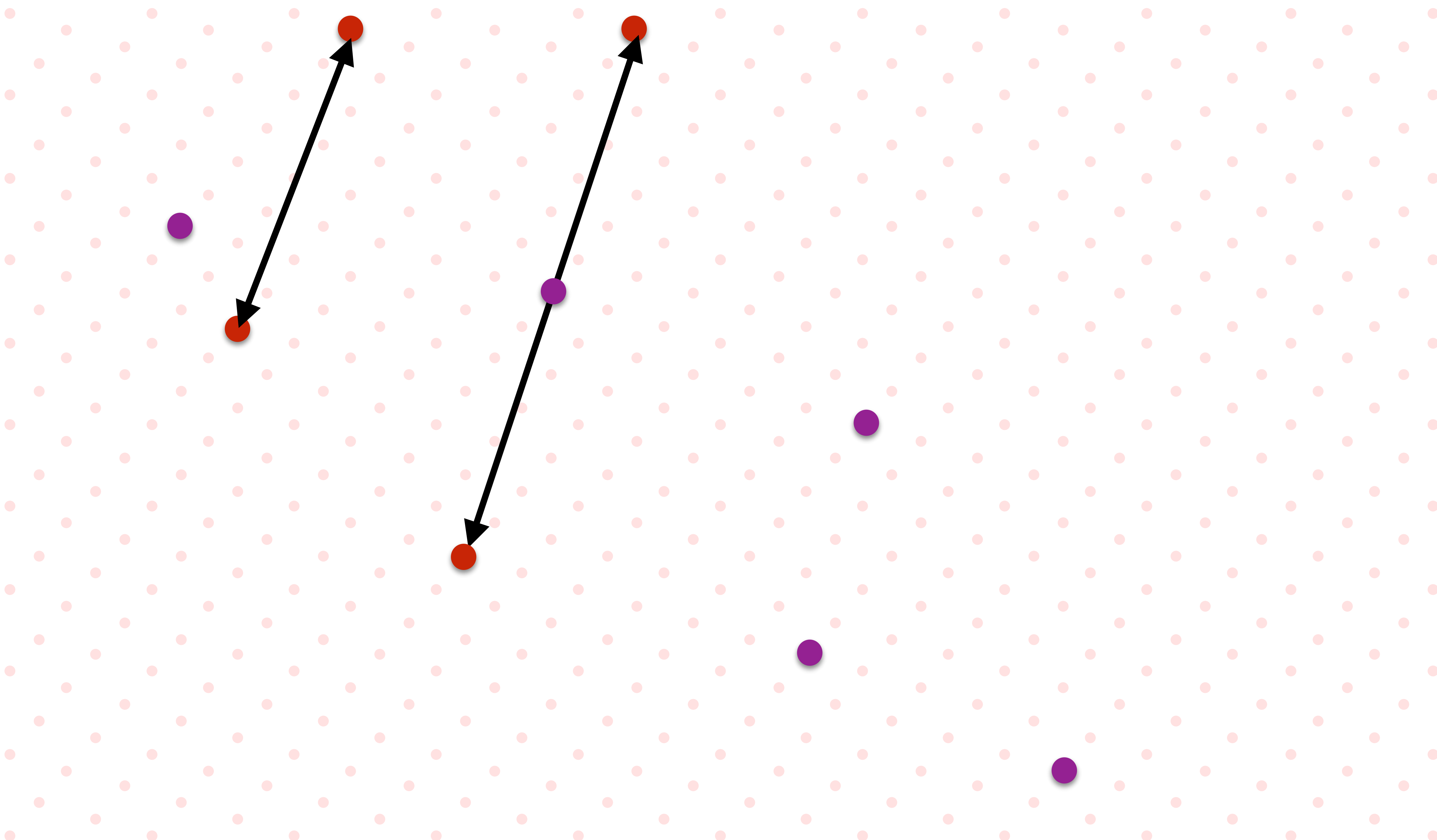
Sieving by Averages



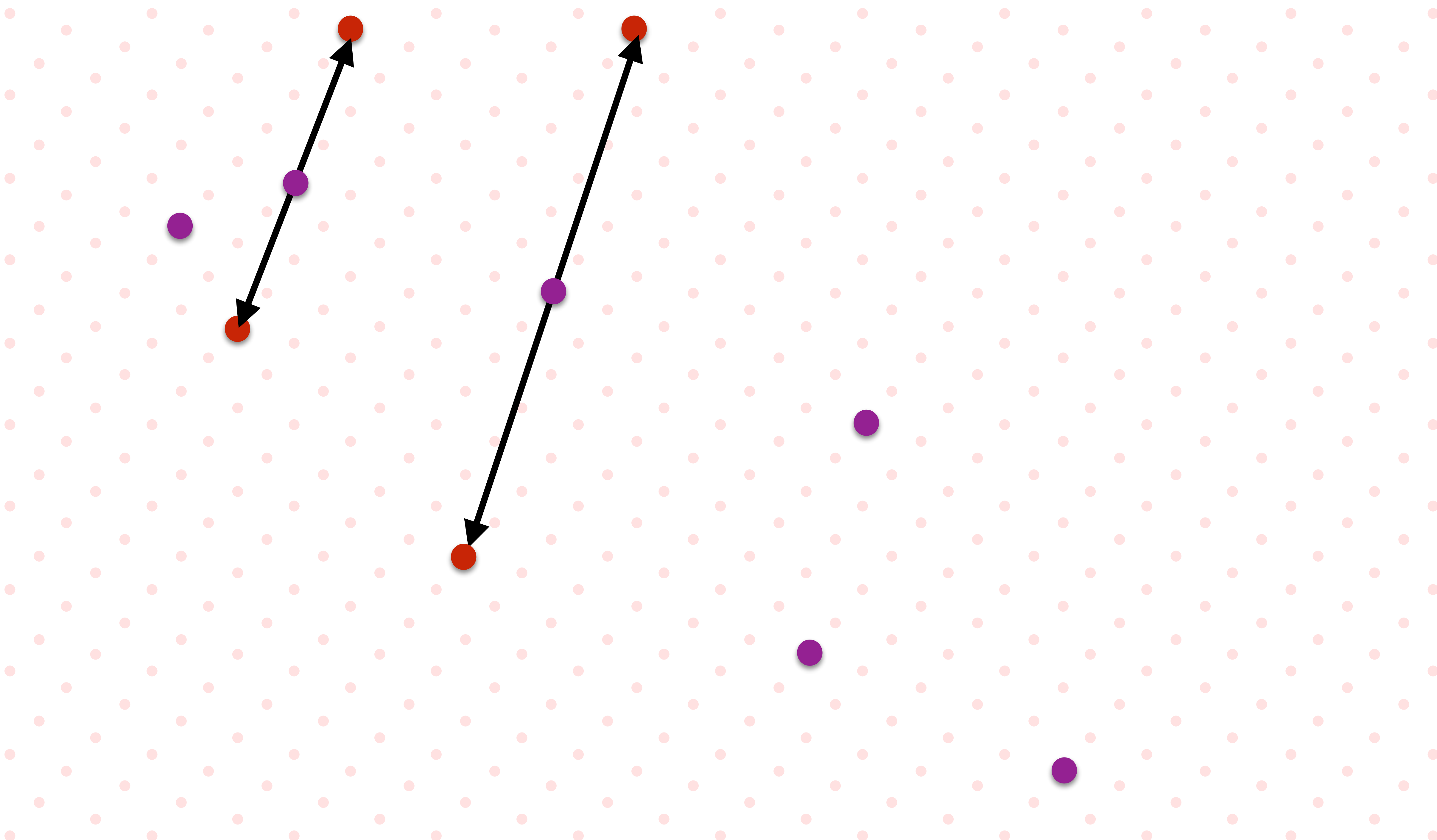
Sieving by Averages



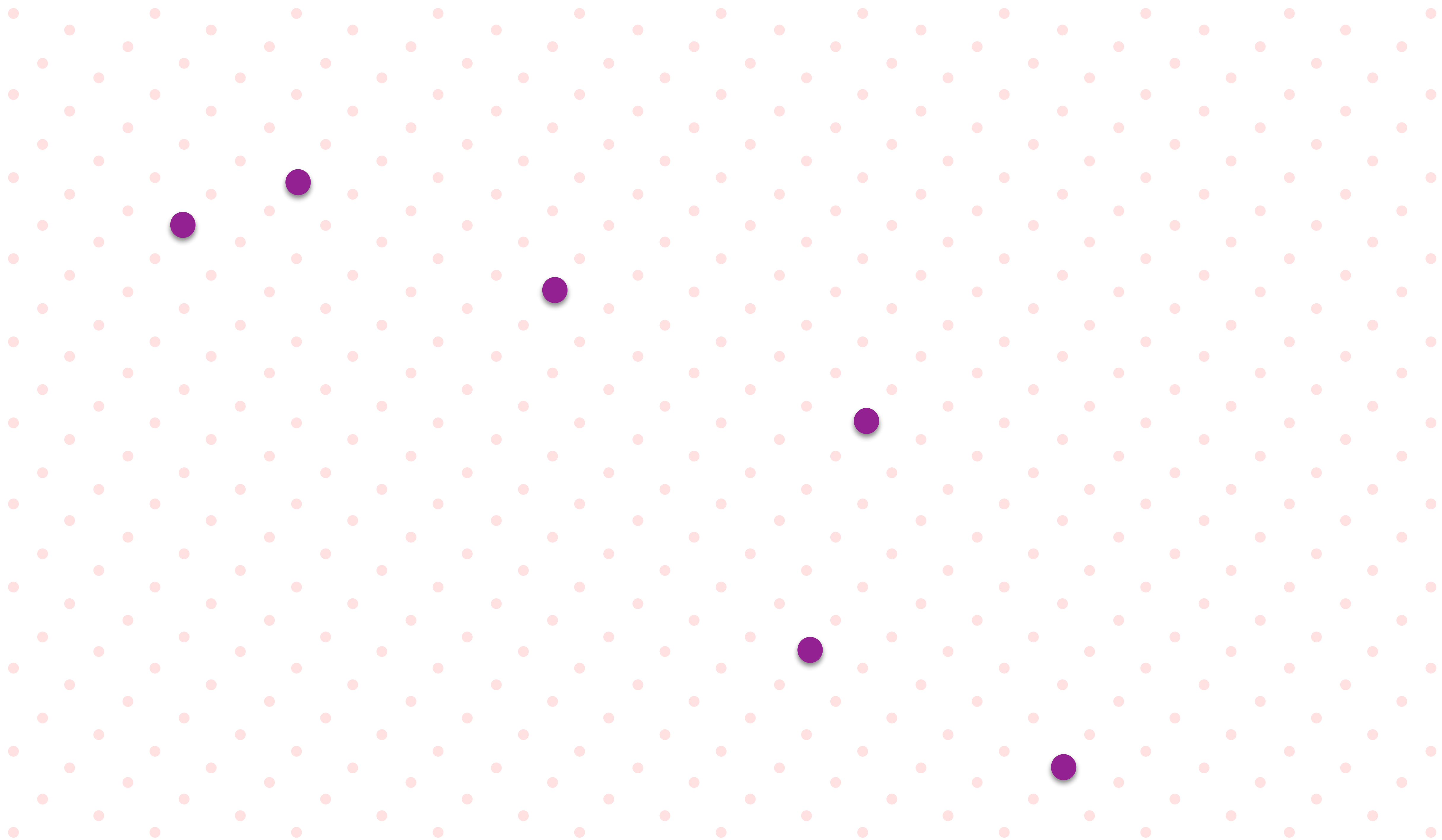
Sieving by Averages



Sieving by Averages



Sieving by Averages



Sieving by Averages

1. Start with many vectors sampled from some nice distribution.
2. For each coset of $2\mathcal{L}$, group the vectors within the coset into disjoint pairs (randomly).
3. Take the average of each pair.
4. Repeat this procedure on the averages.

Sieving by Averages

Cons

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

- Ignoring the geometry makes the distribution of vectors easier to analyze.

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

- Ignoring the geometry makes the distribution of vectors easier to analyze.
  - In [ADRS15], we combined this with careful rejection sampling to solve SVP.

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

- Ignoring the geometry makes the distribution of vectors easier to analyze.
    - In [ADRS15], we combined this with careful rejection sampling to solve SVP.
- It actually just works! [AS17]

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

- Ignoring the geometry makes the distribution of vectors easier to analyze.
    - In [ADRS15], we combined this with careful rejection sampling to solve SVP.
- It actually just works! [AS17]
    - No rejection sampling or perturbation needed!

Sieving by Averages

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Pros

- Ignoring the geometry makes the distribution of vectors easier to analyze.
    - In [ADRS15], we combined this with careful rejection sampling to solve SVP.
- It actually just works! [AS17]
    - No rejection sampling or perturbation needed!
    - Yields the fastest known algorithm for SVP!

Seriously…

Seriously…

The fastest known SVP algorithm* [AS17]:

Seriously…

The fastest known SVP algorithm* [AS17]:

** with a known proof of correctness.*

Seriously…

The fastest known SVP algorithm* [AS17]:

1. Start with $2^{n+o(n)}$ not-too-short lattice vectors (sampled from the discrete Gaussian).

** with a known proof of correctness.*

Seriously…

The fastest known SVP algorithm* [AS17]:

1. Start with $2^{n+o(n)}$ not-too-short lattice vectors (sampled from the discrete Gaussian).
2. Do this “sieving by averages” thing.

** with a known proof of correctness.*

Seriously…

The fastest known SVP algorithm* [AS17]:

1. Start with $2^{n+o(n)}$ not-too-short lattice vectors (sampled from the discrete Gaussian).
2. Do this “sieving by averages” thing.
3. Output the shortest non-zero vector that you see.

** with a known proof of correctness.*

Why Does This Work?

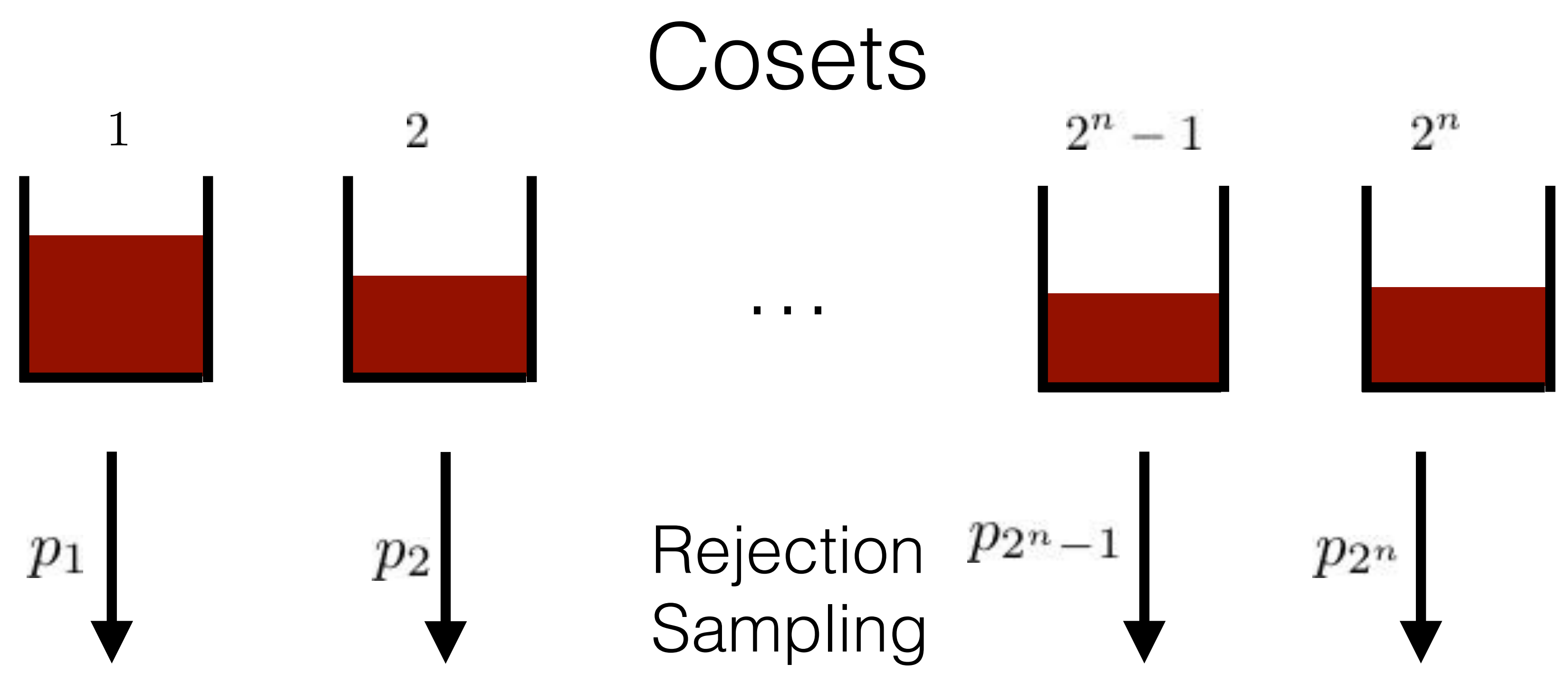
Cosets

Why Does This Work?

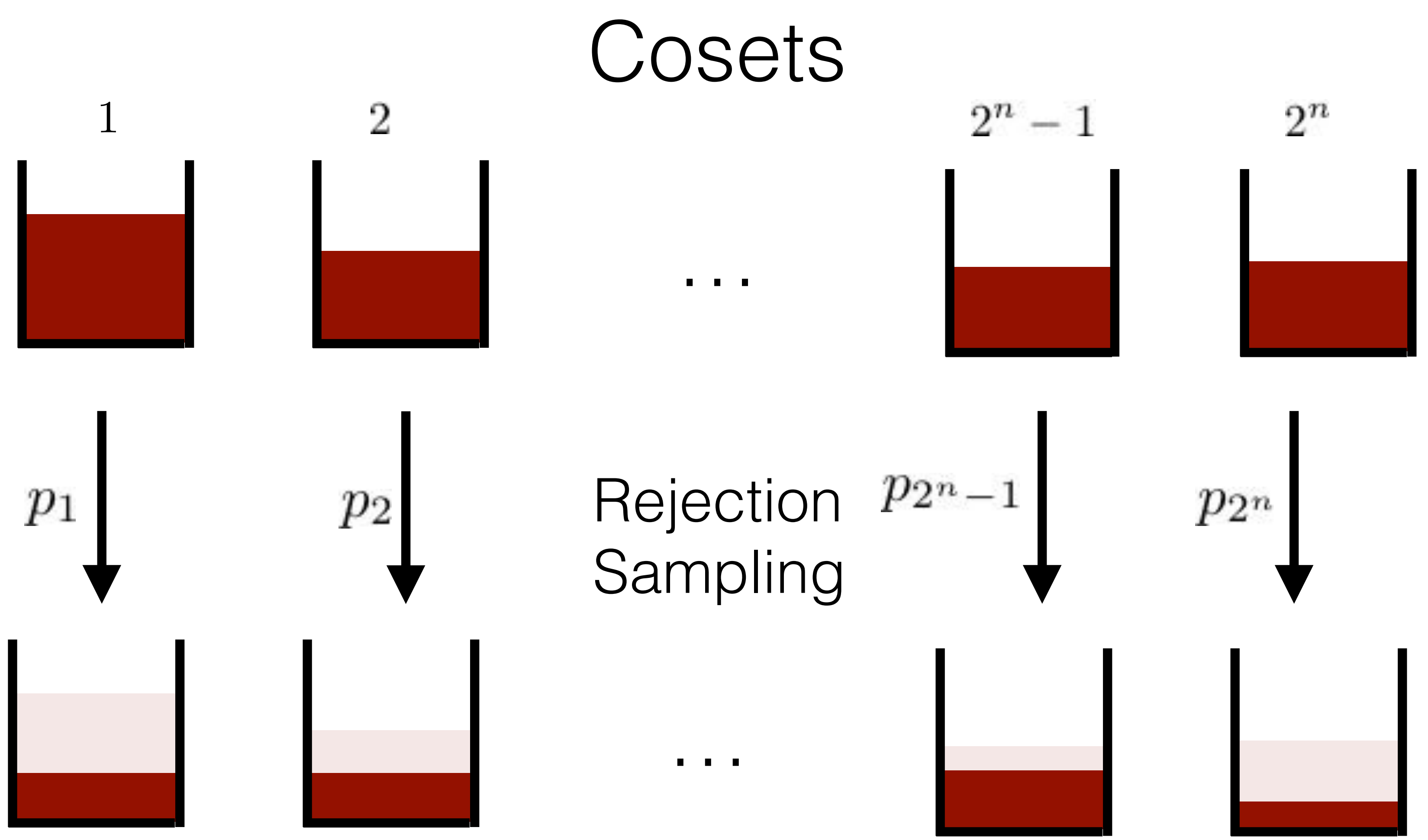
Cosets

Rejection
Sampling

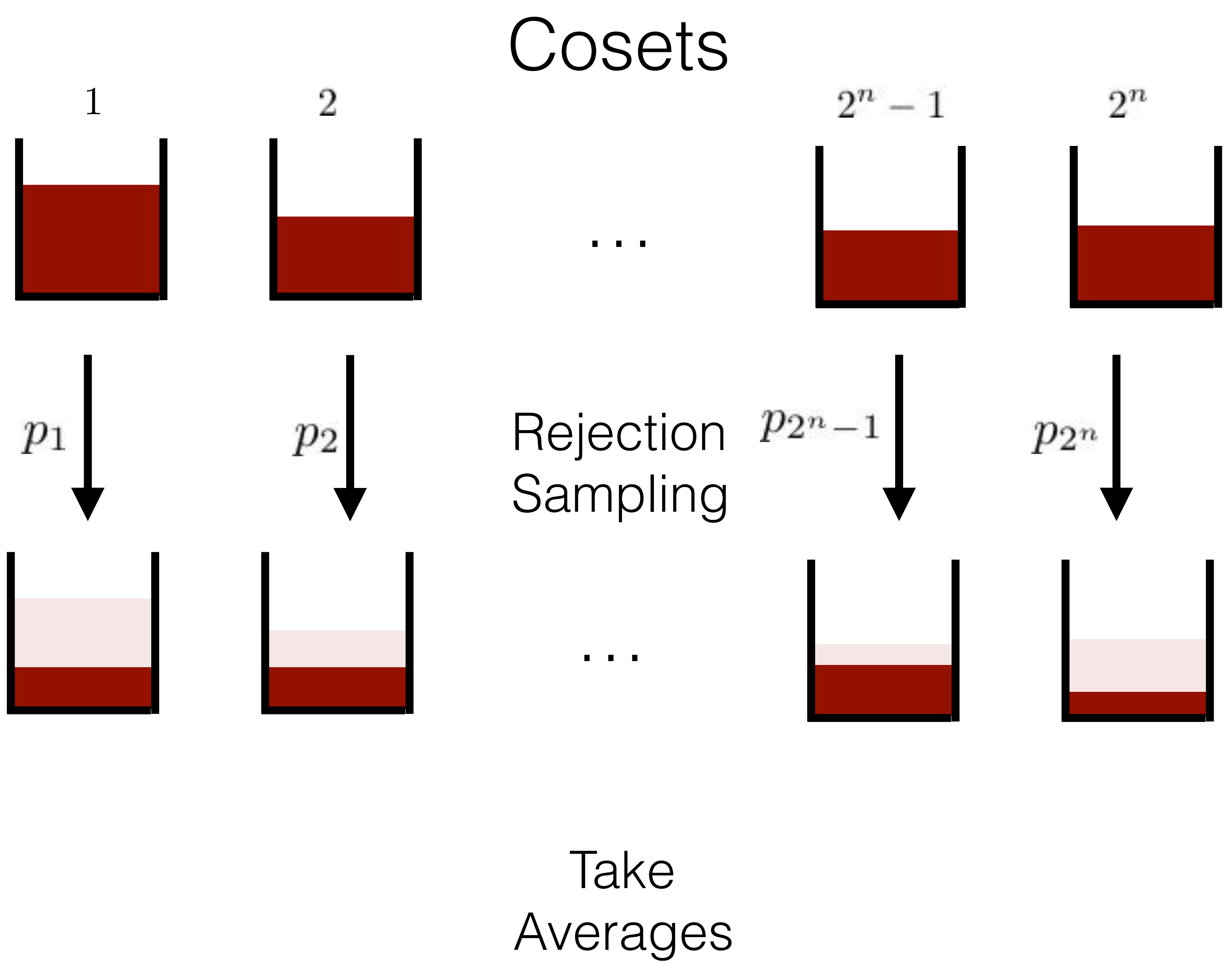
Why Does This Work?



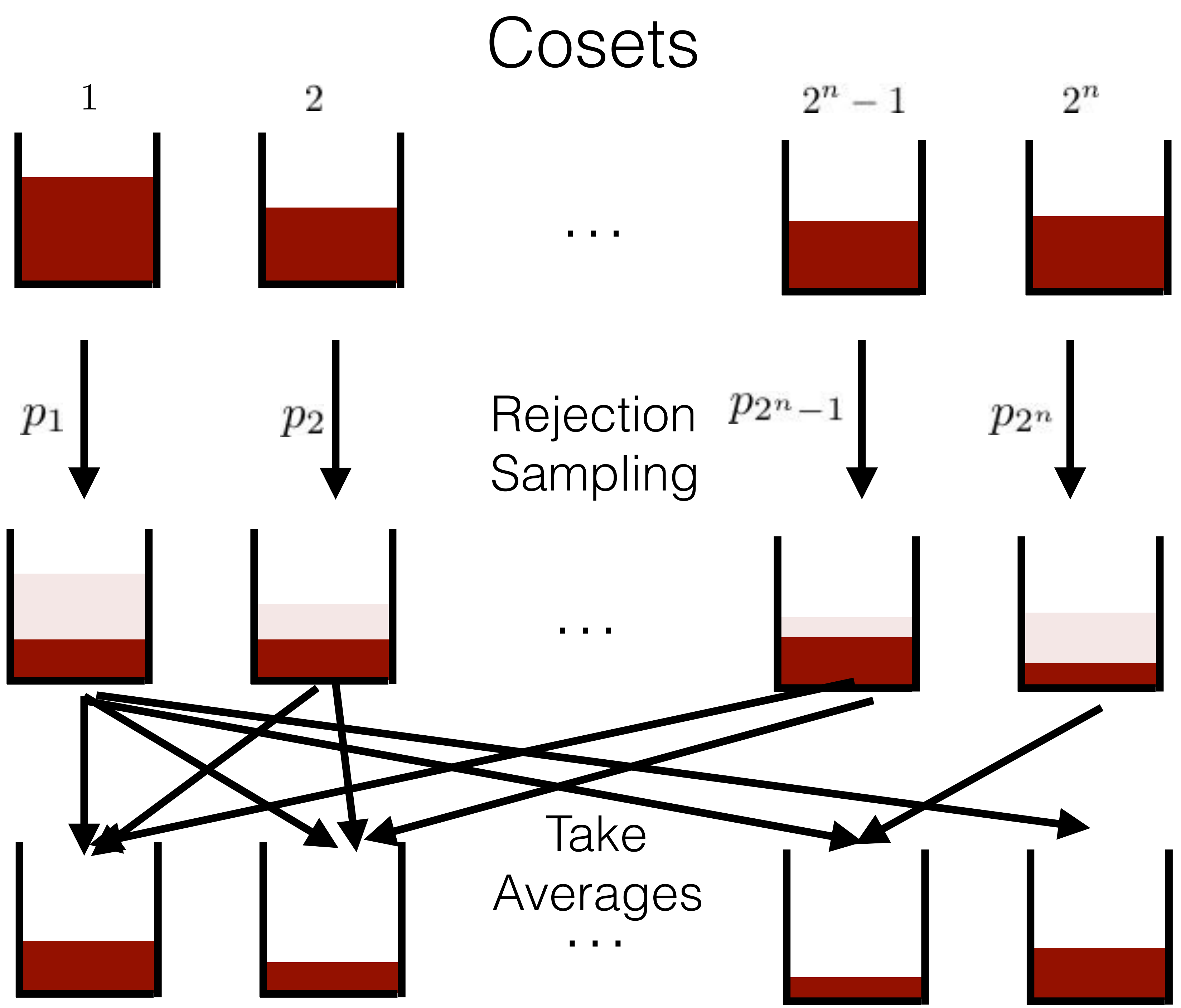
Why Does This Work?



Why Does This Work?

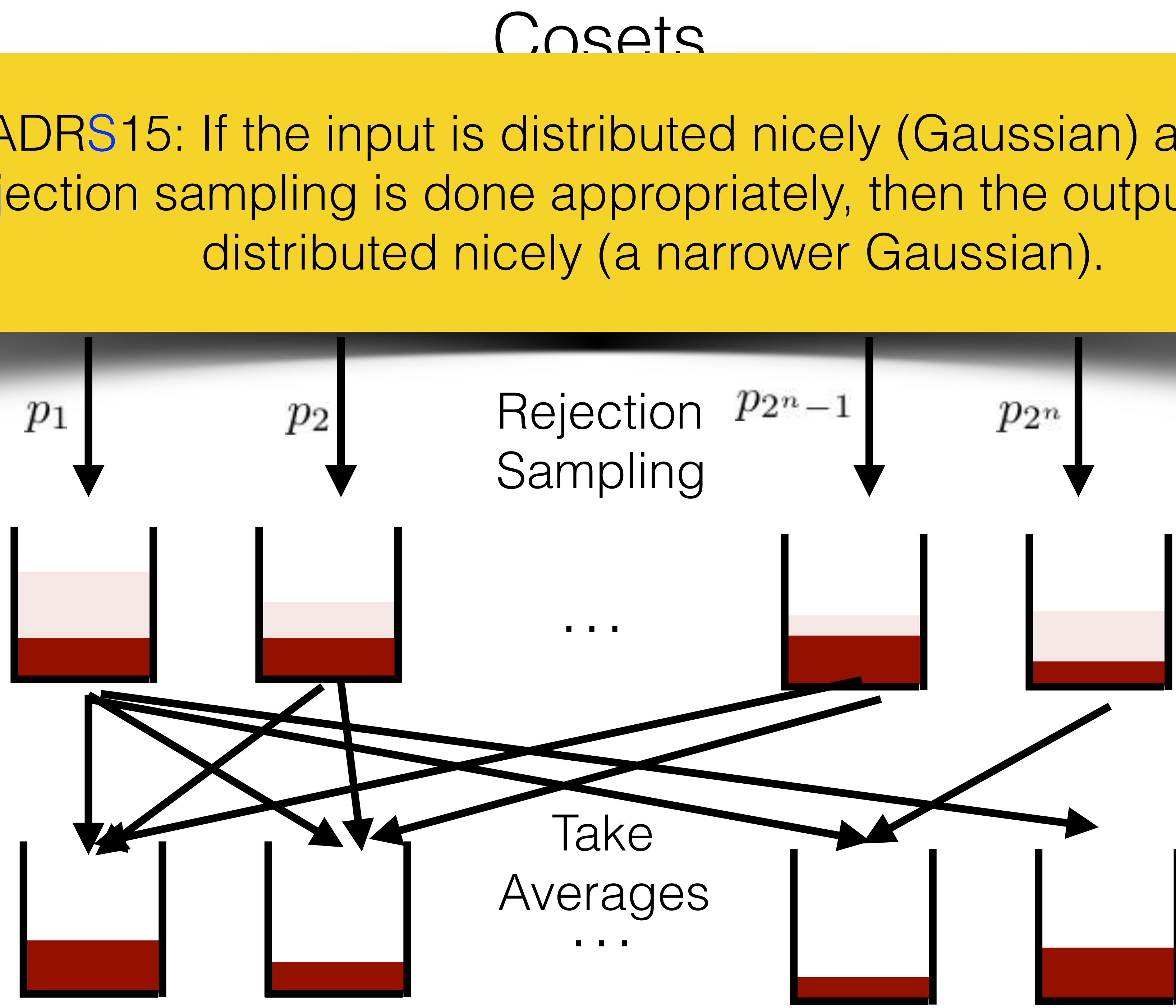


Why Does This Work?



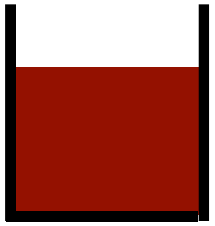
Why Does This Work?

ADRS15: If the input is distributed nicely (Gaussian) and the rejection sampling is done appropriately, then the output will be distributed nicely (a narrower Gaussian).

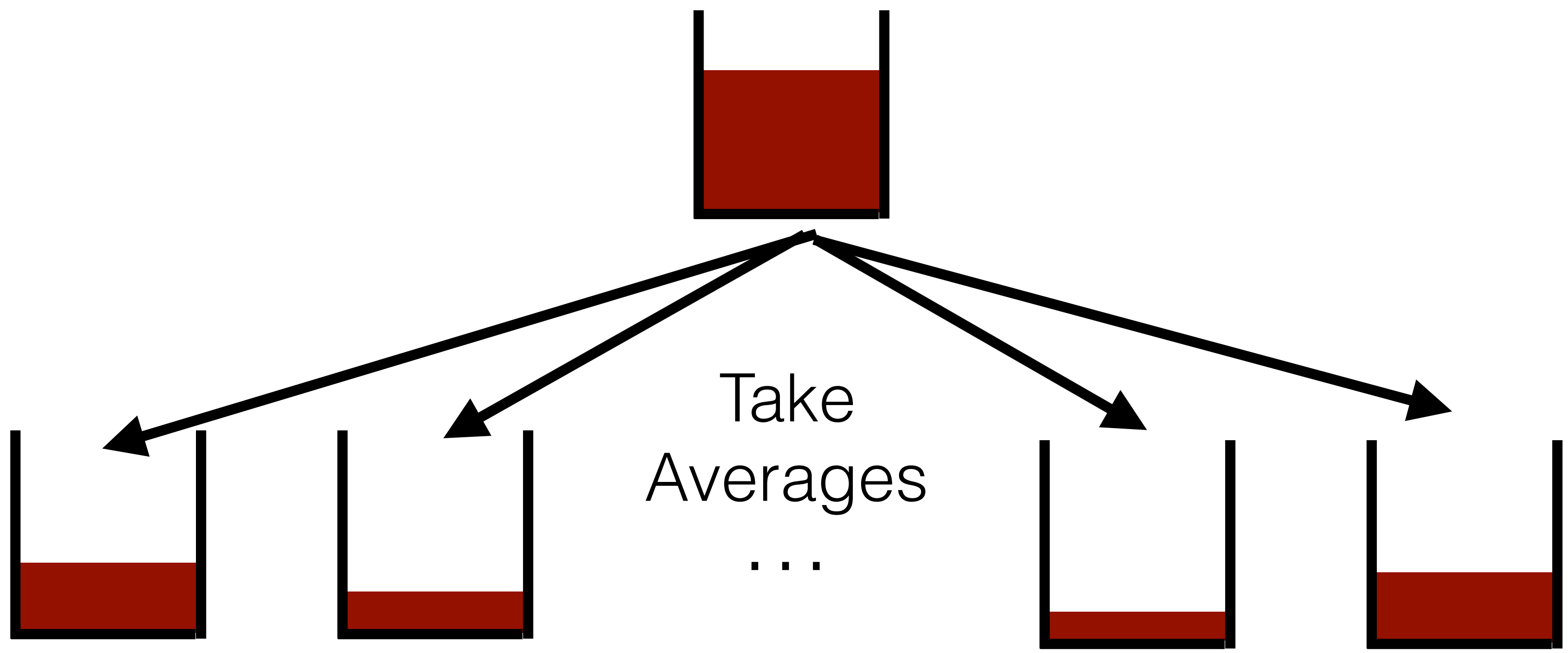


Why Does This Work?

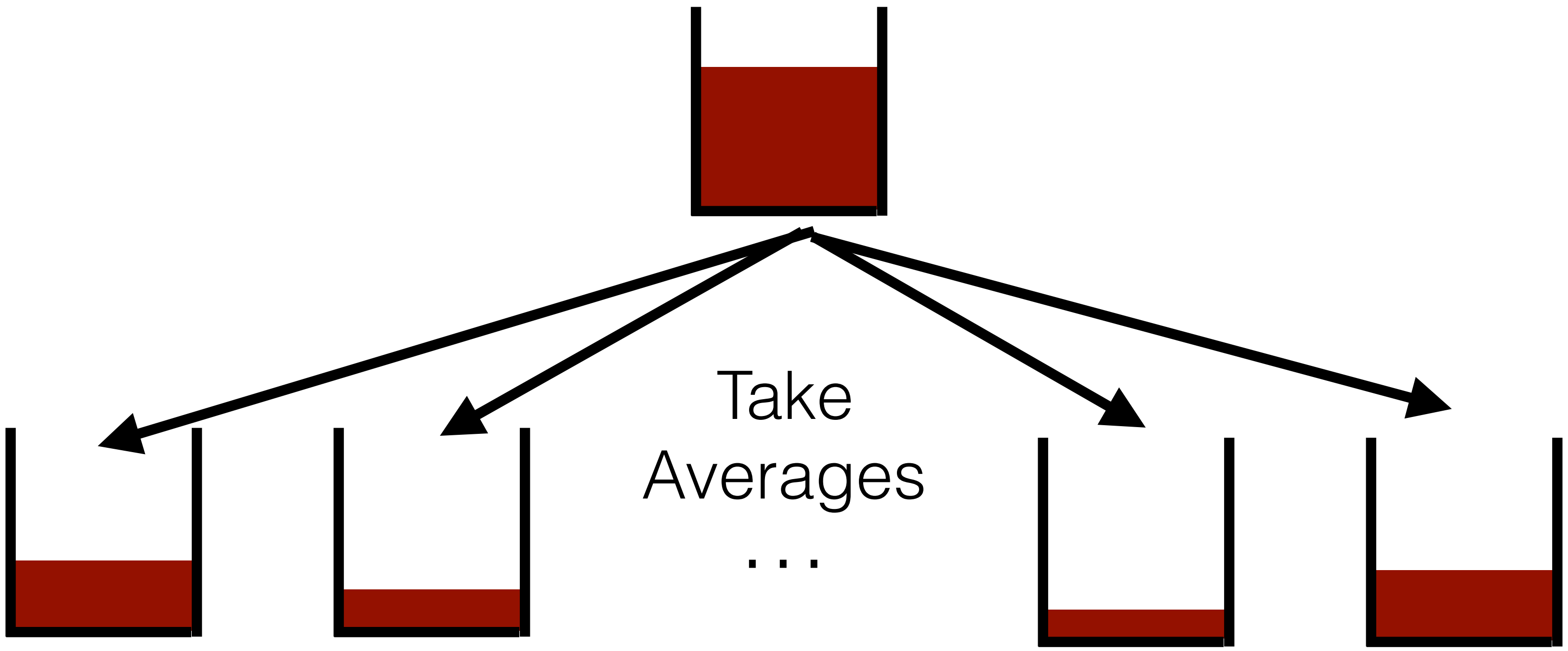
Why Does This Work?



Why Does This Work?



Why Does This Work?

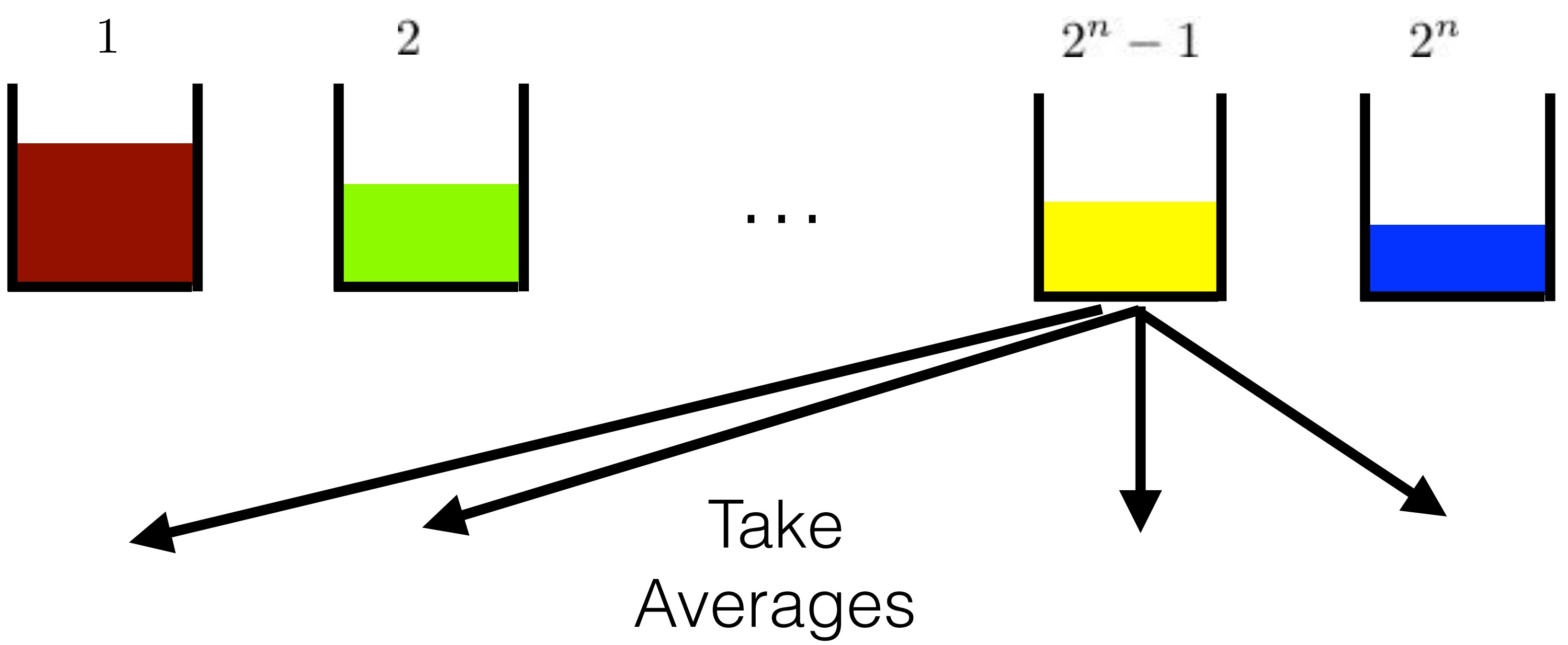


AS17: If the input is distributed nicely (Gaussian) *within* each coset, then the averages will be distributed nicely (a narrower Gaussian) *within* each coset.

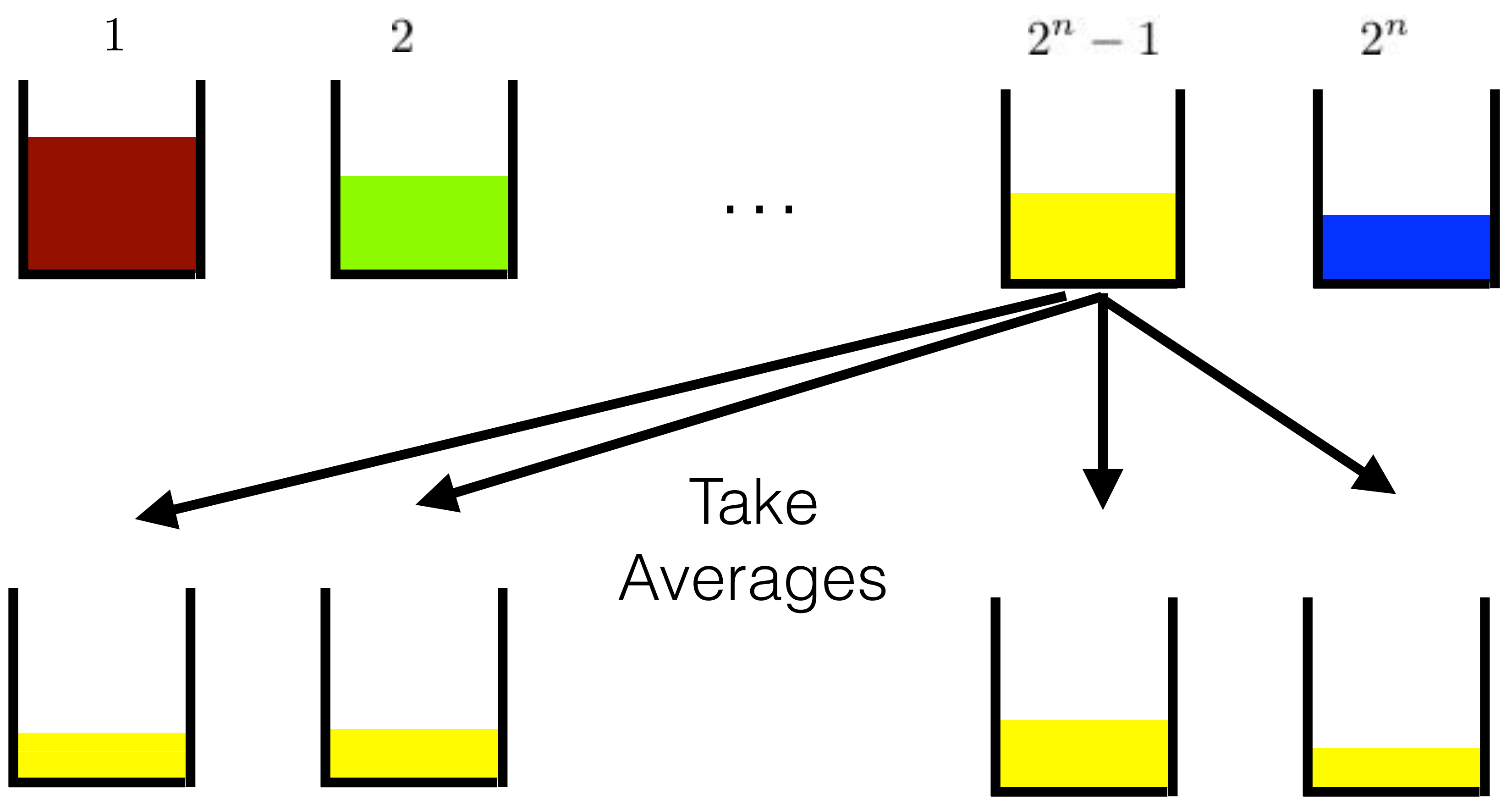
Why Does This Work?

Why Does This Work?

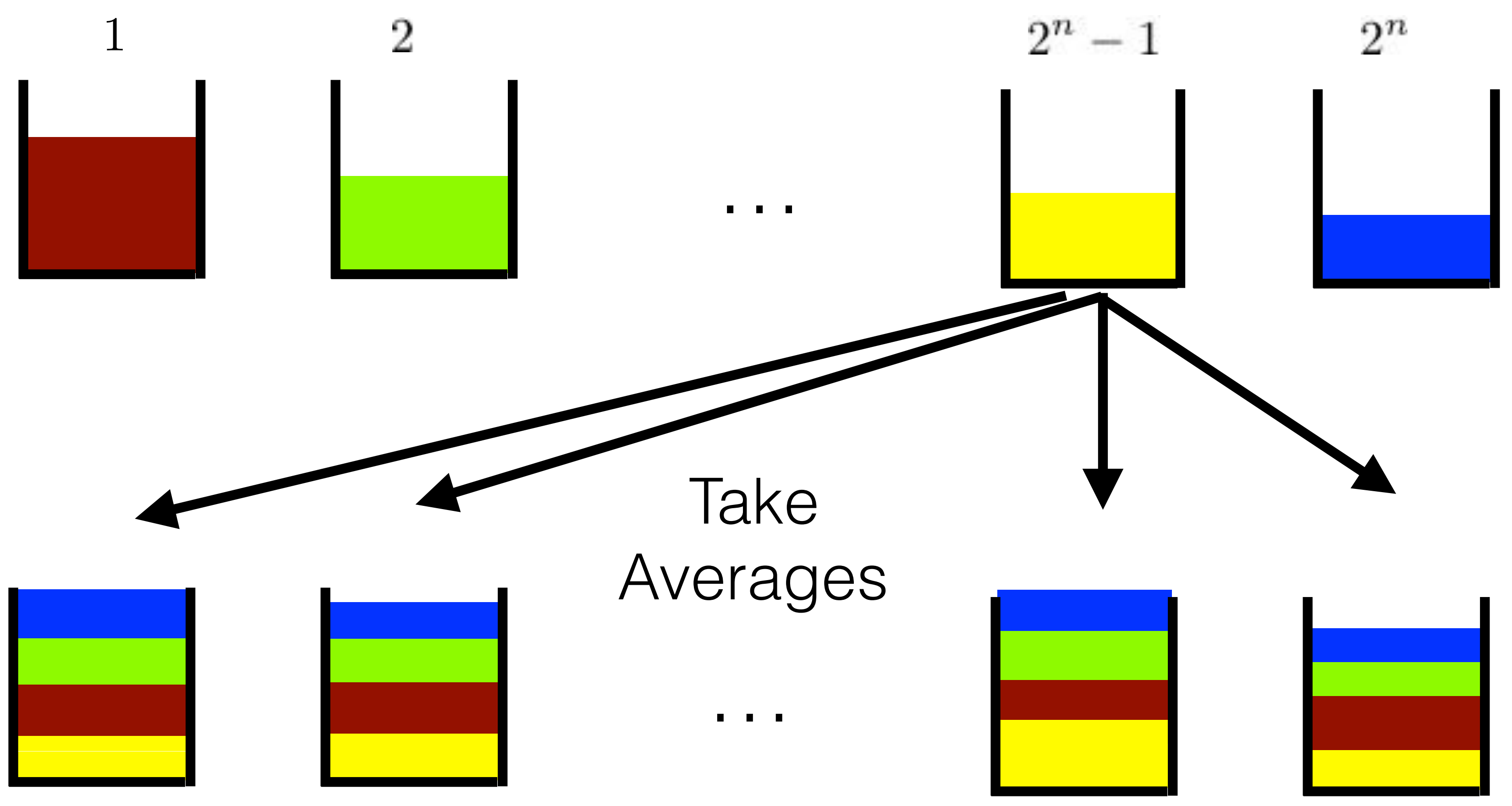
Why Does This Work?



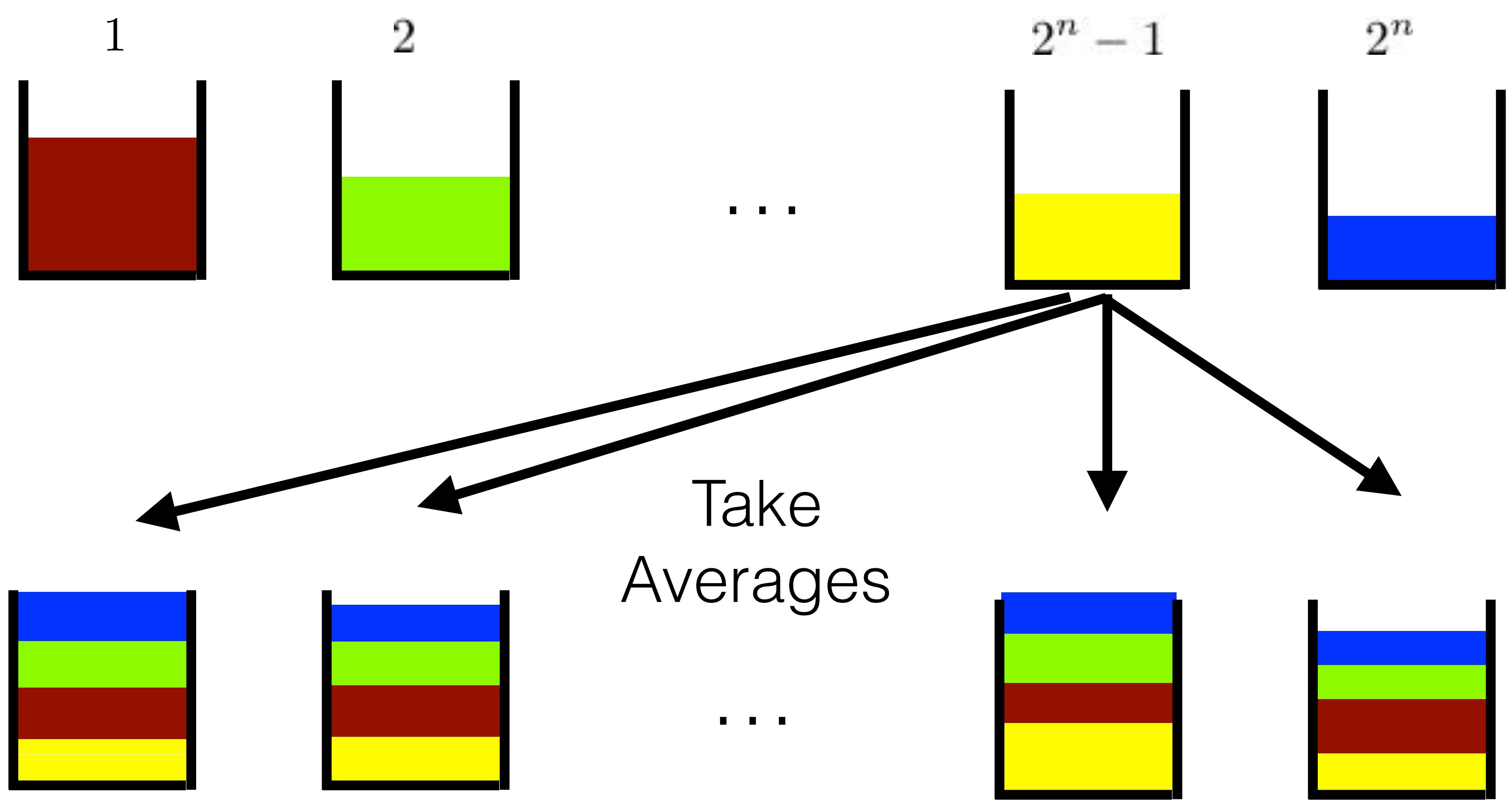
Why Does This Work?



Why Does This Work?

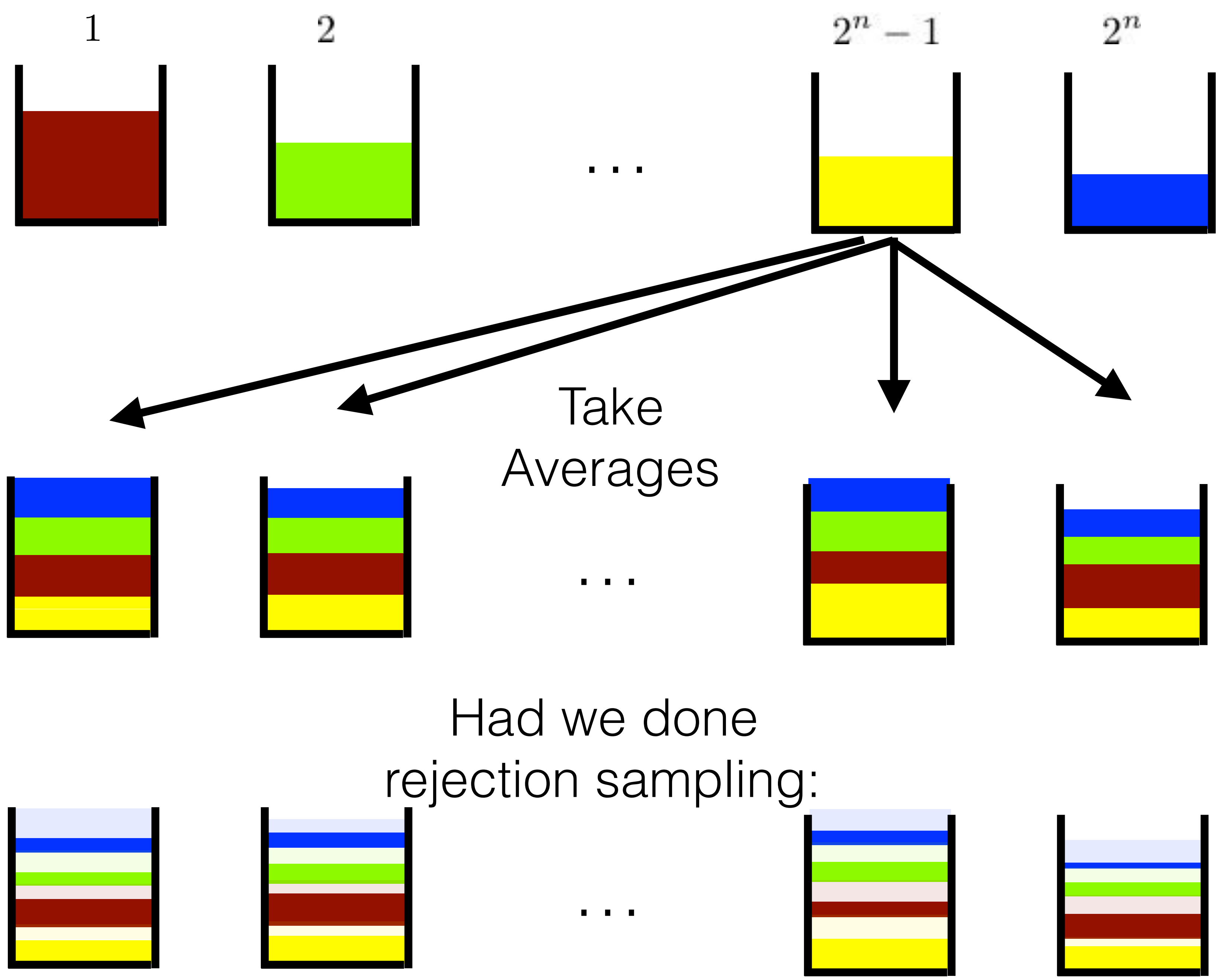


Why Does This Work?



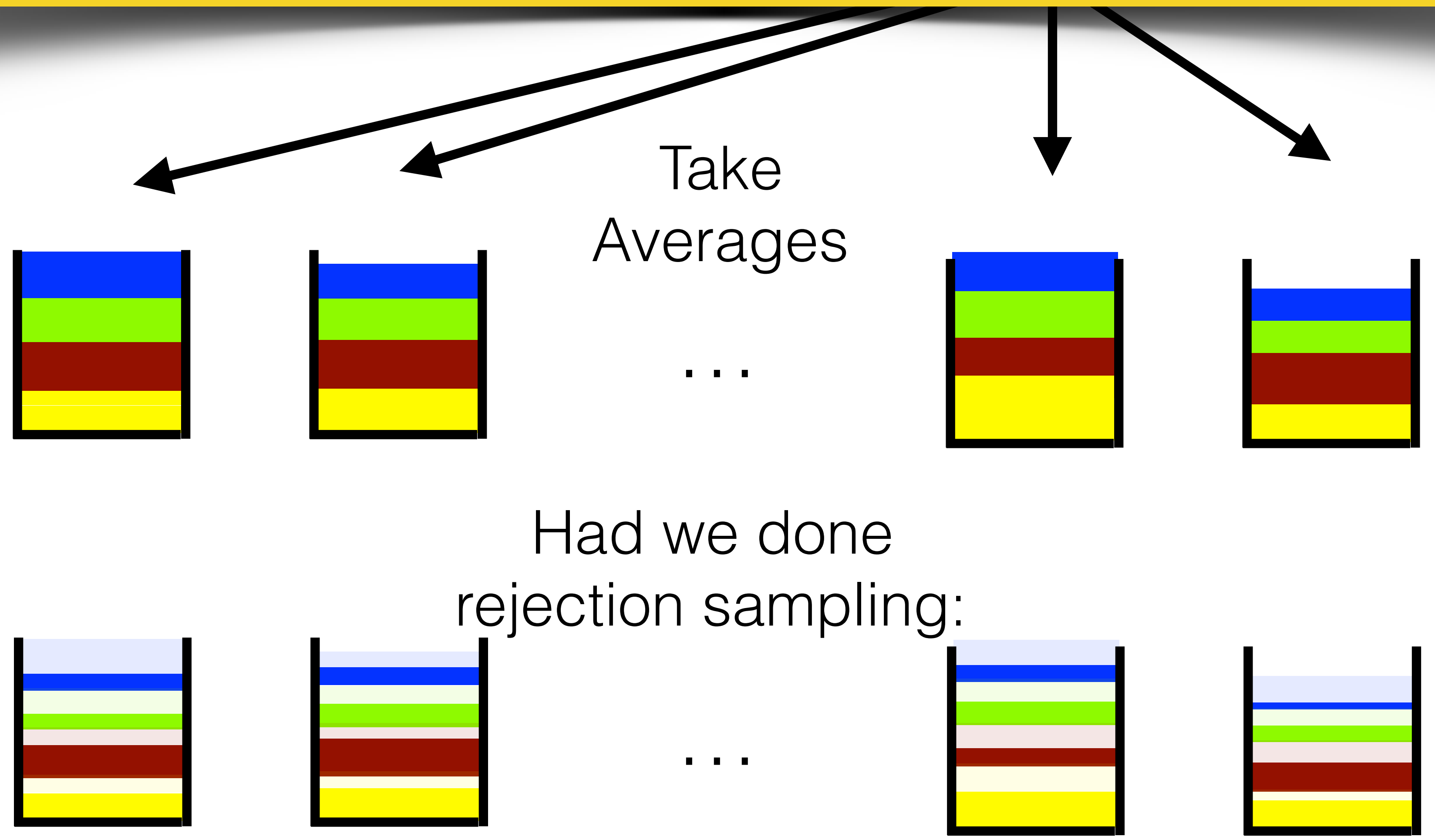
Had we done
rejection sampling:

Why Does This Work?



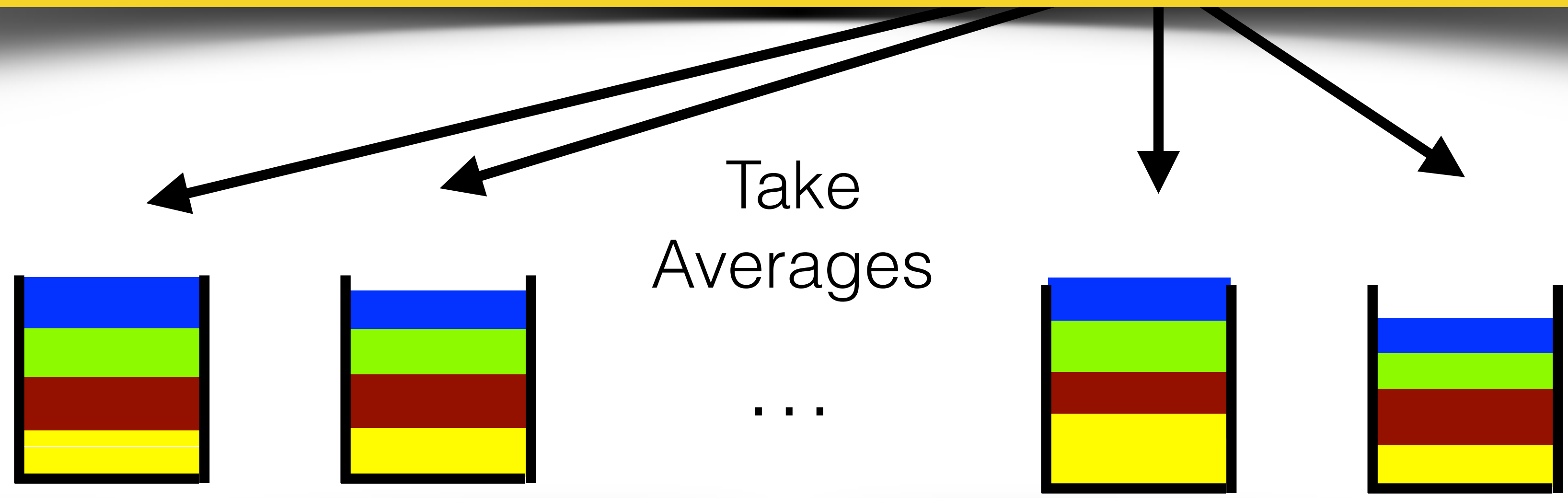
Why Does This Work?

Rejection sampling only reweights the cosets in the output distribution. If we don't do rejection sampling, we just get more vectors from each coset.

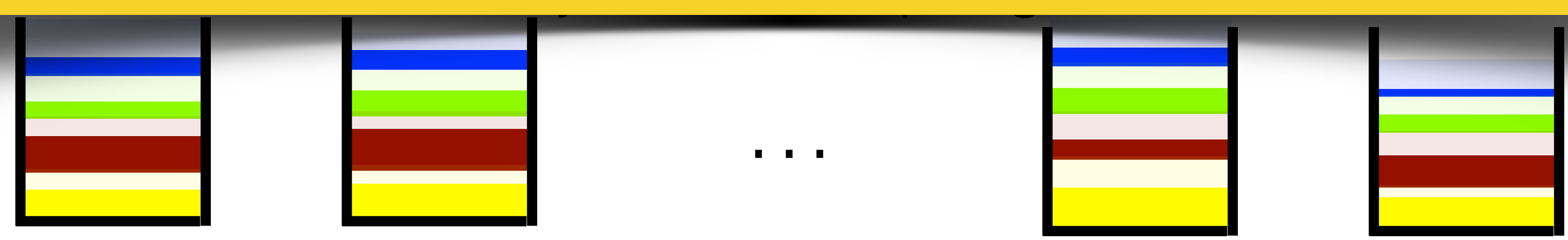


Why Does This Work?

Rejection sampling only reweights the cosets in the output distribution. If we don't do rejection sampling, we just get more vectors from each coset.



The rejection sampling procedure from ADRS15 was pointless!



Why Does This Work?

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Moral explanation: Because $\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]$ for symmetric distributions.

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Moral explanation: Because $\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]$ for symmetric distributions.

$\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2]$

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Moral explanation: Because $\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]$ for symmetric distributions.

$$\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2 + 2\langle \mathbf{x}, \mathbf{y} \rangle + \|\mathbf{y}\|^2]$$

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Moral explanation: Because $\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]$ for symmetric distributions.

$$\begin{aligned}\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] &= \mathbb{E}[\|\mathbf{x}\|^2 + 2\langle \mathbf{x}, \mathbf{y}\rangle + \|\mathbf{y}\|^2] \\ &= \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2] + 2\mathbb{E}[\langle \mathbf{x}, \mathbf{y}\rangle]\end{aligned}$$

Why Does This Work?

Formal proof: The probability of seeing any vector when we run this simple algorithm is always greater than the probability of seeing it when we run the ADRS15 rejection sampling algorithm.

Moral explanation: Because $\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]$ for symmetric distributions.

$$\begin{aligned}\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] &= \mathbb{E}[\|\mathbf{x}\|^2 + 2\langle \mathbf{x}, \mathbf{y}\rangle + \|\mathbf{y}\|^2] \\ &= \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2] + 2\mathbb{E}[\langle \mathbf{x}, \mathbf{y}\rangle] \\ &= \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]\end{aligned}$$

Why Does This Work?

$$\boxed{\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}$$

Why Does This Work?

$$\boxed{\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}$$

$$\mathbb{E}\left[\left\|\frac{\mathbf{x}+\mathbf{y}}{2}\right\|^2\right] = \frac{\mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}{4}$$

Why Does This Work?

$$\boxed{\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}$$

$$\mathbb{E}\left[\left\|\frac{\mathbf{x}+\mathbf{y}}{2}\right\|^2\right] = \frac{\mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}{4}$$

$$= \frac{\mathbb{E}[\|\mathbf{x}\|^2]}{2}$$

(If $\mathbf{x}$ and $\mathbf{y}$ have the same marginal distribution.)

Why Does This Work?

$$\boxed{\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}$$

$$\mathbb{E}\left[\left\|\frac{\mathbf{x}+\mathbf{y}}{2}\right\|^2\right] = \frac{\mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}{4}$$

$$= \frac{\mathbb{E}[\|\mathbf{x}\|^2]}{2}$$

(If $\mathbf{x}$ and $\mathbf{y}$ have the same marginal distribution.)

In our case, $\mathbf{x}$ comes from some symmetric distribution, and $\mathbf{y}$ comes from the same distribution, conditioned on $\mathbf{x} \equiv \mathbf{y} \bmod 2\mathcal{L}$

Why Does This Work?

$$\boxed{\mathbb{E}[\|\mathbf{x}+\mathbf{y}\|^2] = \mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}$$

$$\mathbb{E}\left[\left\|\frac{\mathbf{x}+\mathbf{y}}{2}\right\|^2\right] = \frac{\mathbb{E}[\|\mathbf{x}\|^2] + \mathbb{E}[\|\mathbf{y}\|^2]}{4}$$

$$= \frac{\mathbb{E}[\|\mathbf{x}\|^2]}{2}$$

(If $\mathbf{x}$ and $\mathbf{y}$ have the same marginal distribution.)

In our case, $\mathbf{x}$ comes from some symmetric distribution, and $\mathbf{y}$ comes from the same distribution, conditioned on $\mathbf{x} \equiv \mathbf{y} \bmod 2\mathcal{L}$

The expected squared norm of our vectors drops by a factor of two at every step!

Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.

Why Does This Work?

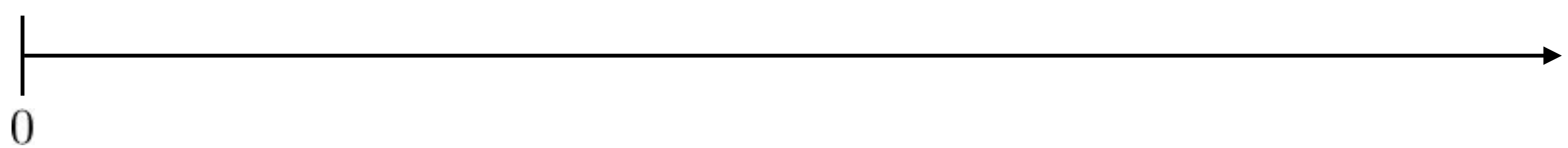
The expected squared norm of our vectors drops by a factor of two at every step.

- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.

Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

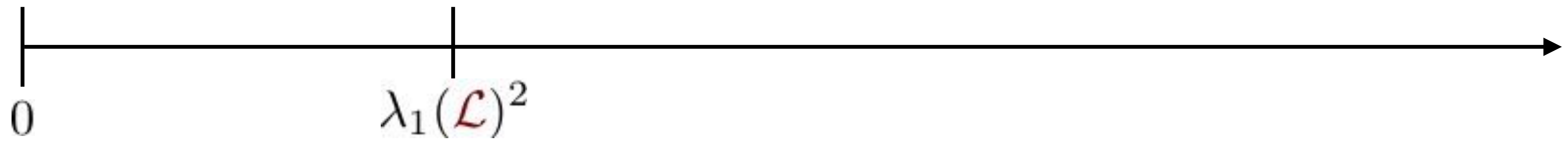
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- “Sieve by averages” $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

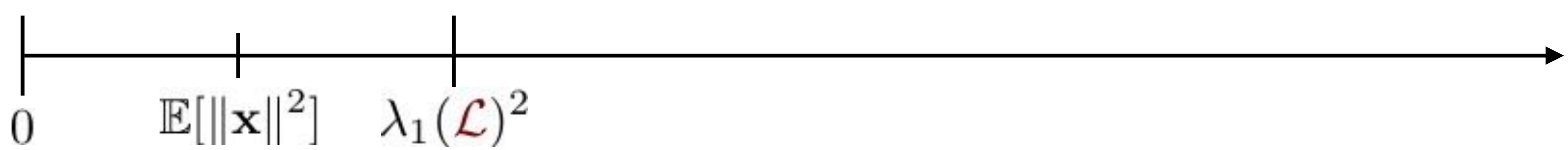
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- “Sieve by averages” $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

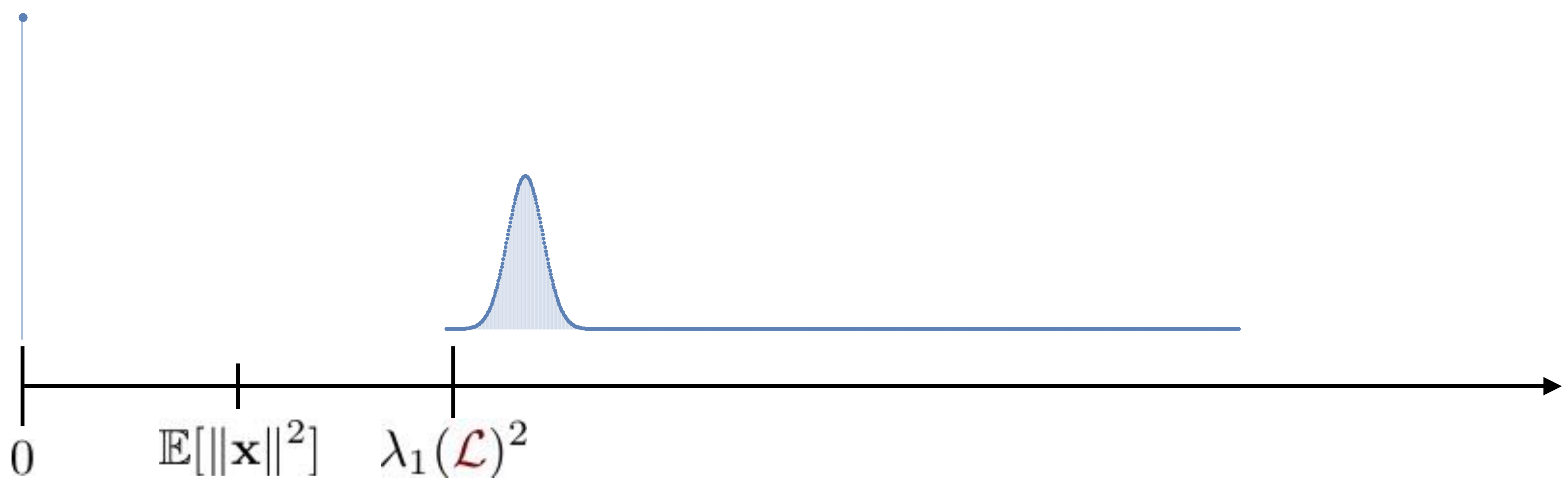
The expected squared norm of our vectors drops by a factor of two at every step.

- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.

Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

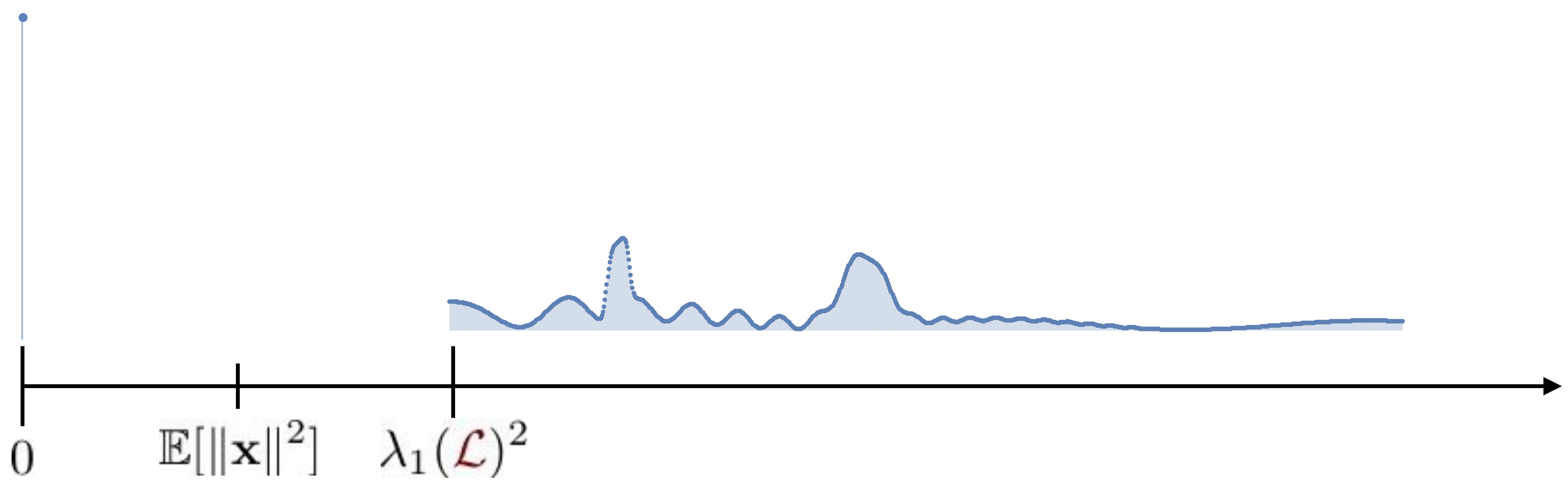
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

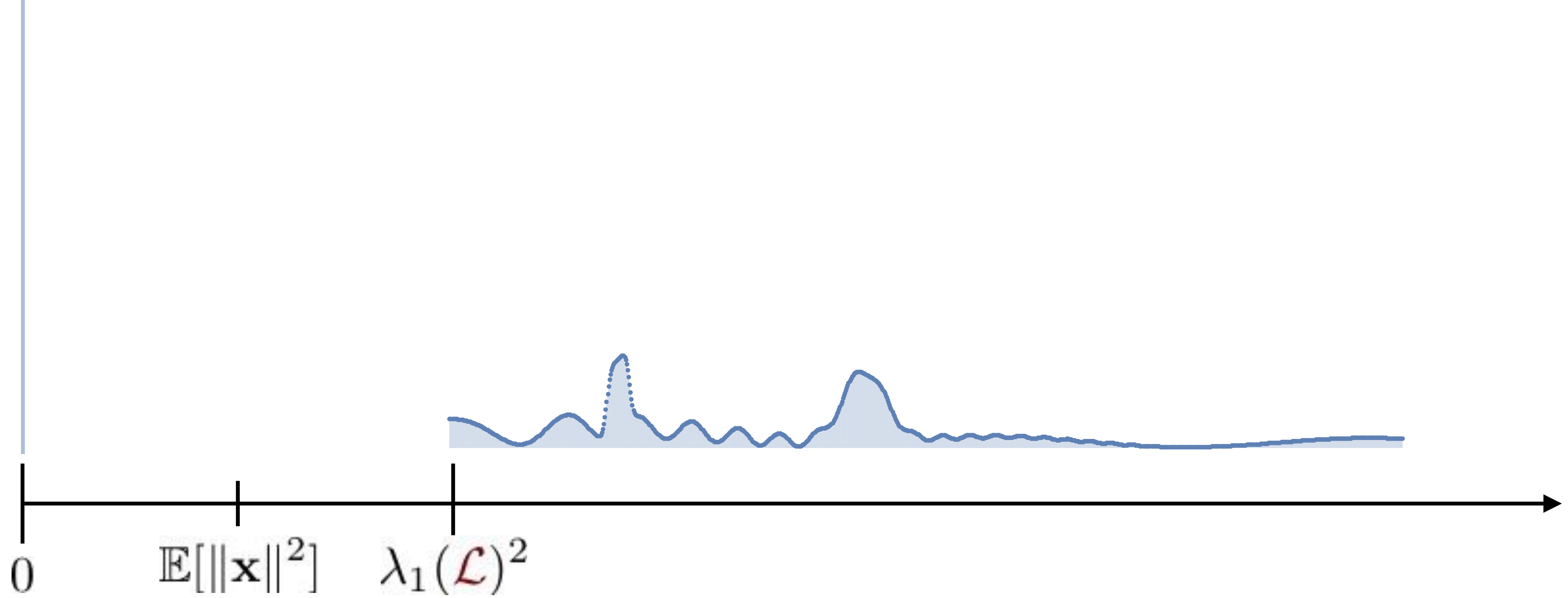
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

The expected squared norm of our vectors drops by a factor of two at every step.

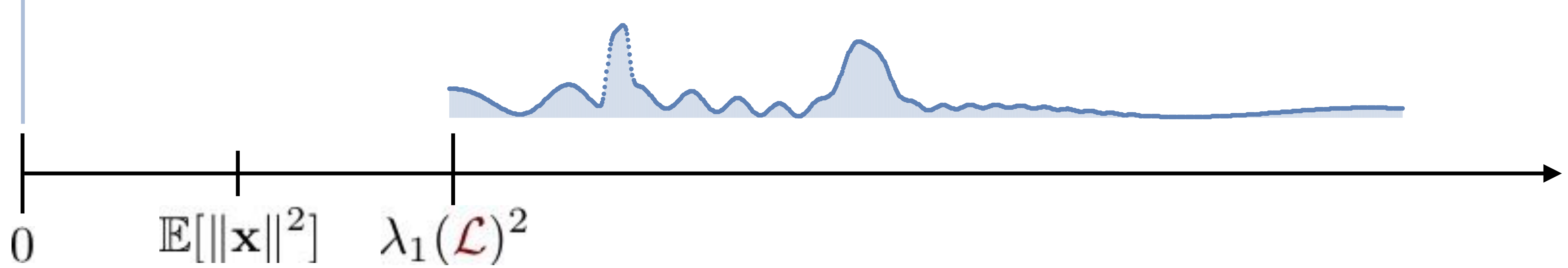
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

If we could get a reasonable bound on the probability of seeing the zero vector, then we could show that this algorithm solves (approximate) SVP.

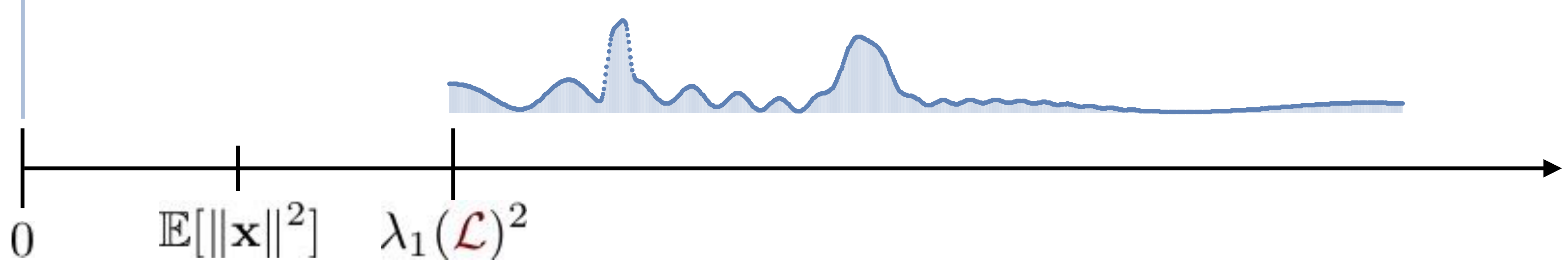
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

~~If we could get a reasonable bound on the probability of seeing the zero vector, then we could show that this algorithm solves (approximate) SVP.~~

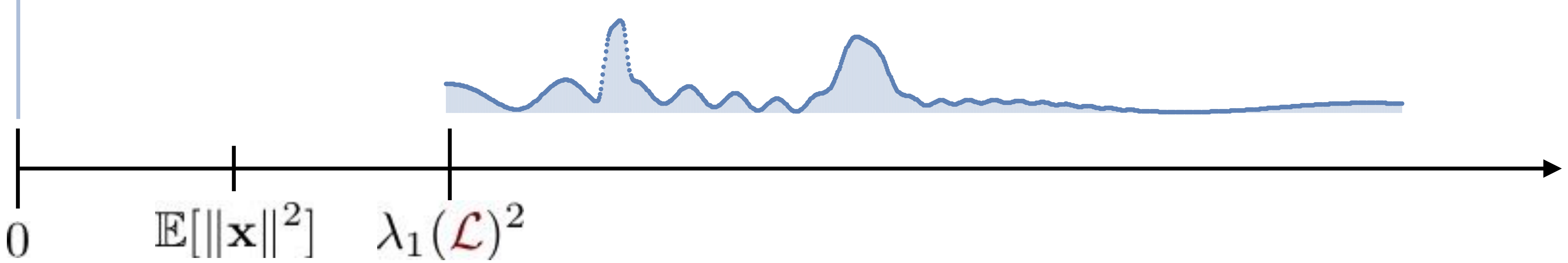
- Start with vectors with $\mathbb{E}[\|\mathbf{x}\|^2] = 2^\ell \cdot \lambda_1(\mathcal{L})^2$.
- "Sieve by averages" $\ell + 1$ times to get samples with $\mathbb{E}[\|\mathbf{x}\|^2] = \lambda_1(\mathcal{L})^2/2$.



Why Does This Work?

~~If we could get a reasonable bound on the probability of seeing the zero vector, then we could show that this algorithm solves (approximate) SVP.~~

If we could show “anything non-trivial” about this distribution, then we could (probably) show that this algorithm solves (approximate) SVP.



Why We Care

Why We Care

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

Why We Care

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

A bit of a lie.

Why We Care

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

A bit of a lie.

Sieving by Averages (Discrete Gaussian)	$2^{n/2+o(n)}$	[ADRS15] (only decision SVP)

Why We Care

Cons

- Completely ignores geometry of the lattice!
- Not even clear that the vectors tend to get shorter…
- Seemingly can't do better than $2^{n+o(n)}$ time and space.

A bit of a lie.

Sieving by Averages (Discrete Gaussian)	$2^{n/2+o(n)}$	[ADRS15] (only decision SVP)

We know how to "remove the rejection sampling procedure" from the main [ADRS15] algorithm (the $2^{n+o(n)}$-time algorithm for SVP).

If we could do this for the $2^{n/2+o(n)}$-time algorithm, then it would provably solve SVP (at least approximately)!

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of two at every step.

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

$$\mathcal{L} \xrightarrow{\bmod\ 2\mathcal{L}'} \mathcal{L}'$$

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of two at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}'} \mathcal{L}' \xrightarrow{\bmod \mathcal{L}} \mathcal{L}/2$$

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}'} \mathcal{L}' \xrightarrow{\bmod \mathcal{L}} \mathcal{L}/2$$

The expected squared norm will drop by a factor of four, but the output vectors will be in $\mathcal{L}/2$.

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at every step.

Greedy

$$\mathcal{L} \subseteq \mathcal{L}'$$

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}'} \mathcal{L}' \xrightarrow{\bmod \mathcal{L}} \mathcal{L}/2$$

The expected squared norm will drop by a factor of four, but the output vectors will be in $\mathcal{L}/2$.

If we double the vectors to put them in $\mathcal{L}$, the final expected squared norm will be unchanged…

Being Less Greedy (and making no progress…)

The expected squared norm of our vectors drops by a factor of $\boxed{\text{two}}$ at

The running time will be roughly $|\mathcal{L}/2\mathcal{L}'| + |\mathcal{L}'/\mathcal{L}|$.

$$2\mathcal{L} \subseteq 2\mathcal{L}' \subseteq \mathcal{L}$$

$$\mathcal{L} \xrightarrow{\bmod\ 2\mathcal{L}'} \mathcal{L}' \xrightarrow{\bmod\ \mathcal{L}} \mathcal{L}/2$$

The expected squared norm will drop by a factor of four, but the output vectors will be in $\mathcal{L}/2$.

If we double the vectors to put them in $\mathcal{L}$, the final expected squared norm will be unchanged…

A “Tower” of Lattices

A “Tower” of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

A “Tower” of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod\ 2\mathcal{L}_1} \mathcal{L}_1$$

A "Tower" of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2$$

A “Tower” of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\bmod 2\mathcal{L}_\ell} \mathcal{L}_\ell$$

A "Tower" of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\text{mod } 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\text{mod } 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\text{mod } 2\mathcal{L}_\ell} \mathcal{L}_\ell \xrightarrow{\text{mod } \mathcal{L}/2^{k-1}} \mathcal{L}/2^k$$

A "Tower" of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\bmod 2\mathcal{L}_\ell} \mathcal{L}_\ell \xrightarrow{\bmod \mathcal{L}/2^{k-1}} \mathcal{L}/2^k$$

The expected squared norm will drop by a factor of 2^ℓ.

A “Tower” of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\bmod 2\mathcal{L}_\ell} \mathcal{L}_\ell \xrightarrow{\bmod \mathcal{L}/2^{k-1}} \mathcal{L}/2^k$$

The expected squared norm will drop by a factor of 2^ℓ.

If we multiply the output vectors by 2^k to put them in $\mathcal{L}$, the final expected squared norm will drop by $2^{\ell-2k}$.

A “Tower” of Lattices

$$\mathcal{L} \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \cdots \subseteq \mathcal{L}_\ell \subseteq \mathcal{L}/2^k$$

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\bmod 2\mathcal{L}_\ell} \mathcal{L}_\ell \xrightarrow{\bmod \mathcal{L}/2^{k-1}} \mathcal{L}/2^k$$

The expected squared norm will drop by a factor of 2^ℓ.

If we multiply the output vectors by 2^k to put them in $\mathcal{L}$, the final expected squared norm will drop by $2^{\ell-2k}$.

The running time will be roughly $\sum_i |\mathcal{L}_i/2\mathcal{L}_{i+1}|$.

A “Tower” of Lattices

We can take $|\mathcal{L}_i/2\mathcal{L}_{i+1}| = 2^{n/2+o(n)}$ to get an algorithm with this running time.

$$\mathcal{L} \xrightarrow{\bmod 2\mathcal{L}_1} \mathcal{L}_1 \xrightarrow{\bmod 2\mathcal{L}_2} \mathcal{L}_2 \longrightarrow \cdots \xrightarrow{\bmod 2\mathcal{L}_\ell} \mathcal{L}_\ell \xrightarrow{\bmod \mathcal{L}/2^{k-1}} \mathcal{L}/2^k$$

The expected squared norm will drop by a factor of 2^ℓ.

If we multiply the output vectors by 2^k to put them in $\mathcal{L}$, the final expected squared norm will drop by $2^{\ell-2k}$.

The running time will be roughly $\sum_i |\mathcal{L}_i/2\mathcal{L}_{i+1}|$.

What We Know About This Faster Algorithm

What We Know About This Faster Algorithm

- There exists a rejection sampling procedure like in [ADRS15] that provably yields a correct algorithm [RS15].

What We Know About This Faster Algorithm

- There exists a rejection sampling procedure like in [ADRS15] that provably yields a correct algorithm [RS15].
    - Don't know how to implement it when the vectors get short.

What We Know About This Faster Algorithm

- There exists a rejection sampling procedure like in [ADRS15] that provably yields a correct algorithm [RS15].
  - Don't know how to implement it when the vectors get short.

- The analysis that we used in [AS17] to "remove the rejection sampling" does not seem to work for this faster algorithm.

What We Know About This Faster Algorithm

- There exists a rejection sampling procedure like in [ADRS15] that provably yields a correct algorithm [RS15].
  - Don't know how to implement it when the vectors get short.

- The analysis that we used in [AS17] to "remove the rejection sampling" does not seem to work for this faster algorithm.

- The "expected squared norm analysis" still works.

What We Know About This Faster Algorithm

- There exists a rejection sampling procedure like in [ADRS15] that provably yields a correct algorithm [RS15].
  - Don't know how to implement it when the vectors get short.

- The analysis that we used in [AS17] to "remove the rejection sampling" does not seem to work for this faster algorithm.

- The "expected squared norm analysis" still works.
  - "Anything non-trivial about distribution" $\Longrightarrow$ faster SVP algorithm!

Parting Thoughts

Parting Thoughts

- Are there other interesting sieving procedures?

Parting Thoughts

- Are there other interesting sieving procedures?
- Can we close the gap between search and decision (approximate) SVP?

Parting Thoughts

- Are there other interesting sieving procedures?
- Can we close the gap between search and decision (approximate) SVP?
  - Can someone please tell me something about this distribution?!

Parting Thoughts

- Are there other interesting sieving procedures?
- Can we close the gap between search and decision (approximate) SVP?
  - Can someone please tell me something about this distribution?!
- Can we close the gap between provable and heuristic algorithms?

Parting Thoughts

- Are there other interesting sieving procedures?
- Can we close the gap between search and decision (approximate) SVP?
  - Can someone please tell me something about this distribution?!
- Can we close the gap between provable and heuristic algorithms?
- Can we close the gap between asymptotically faster algorithms and enumeration?

Thanks!