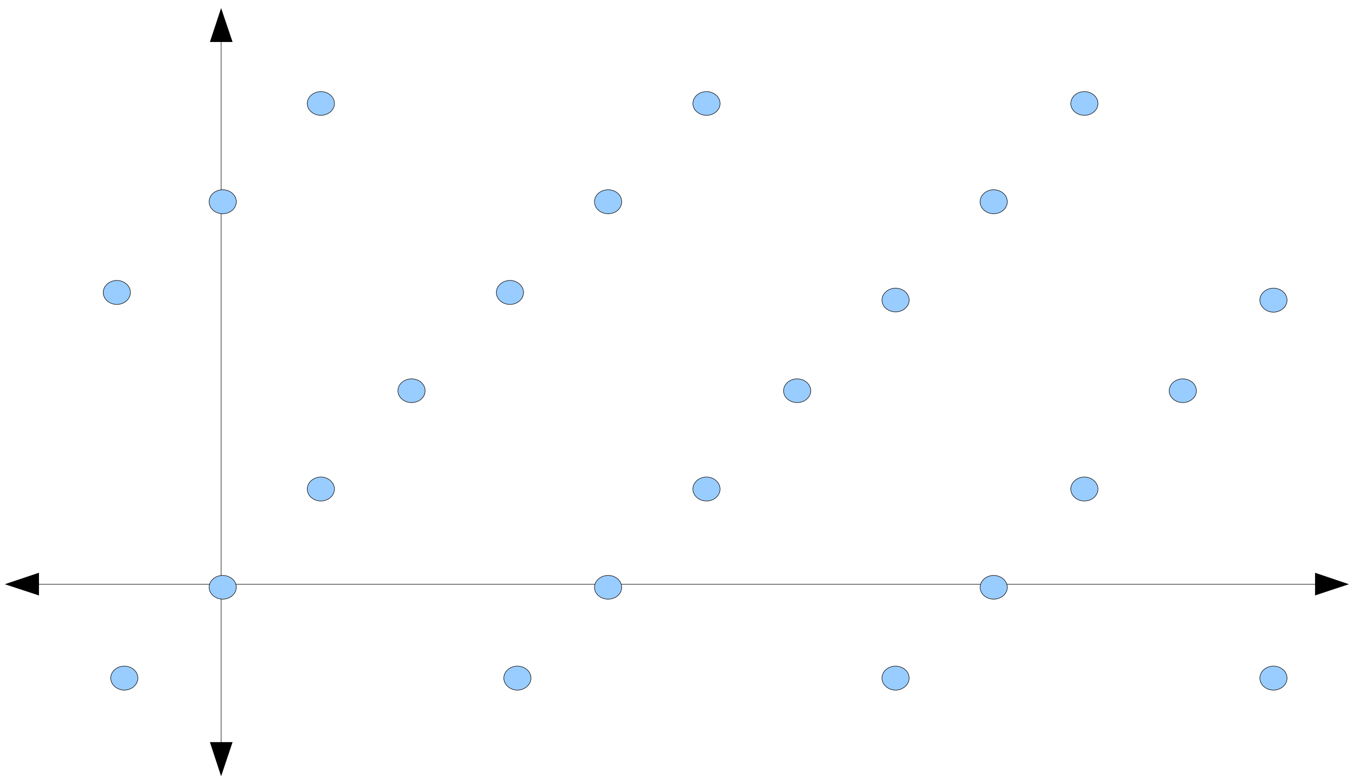
# On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem

Vadim Lyubashevsky
Daniele Micciancio

To appear at Crypto 2009

# Lattices

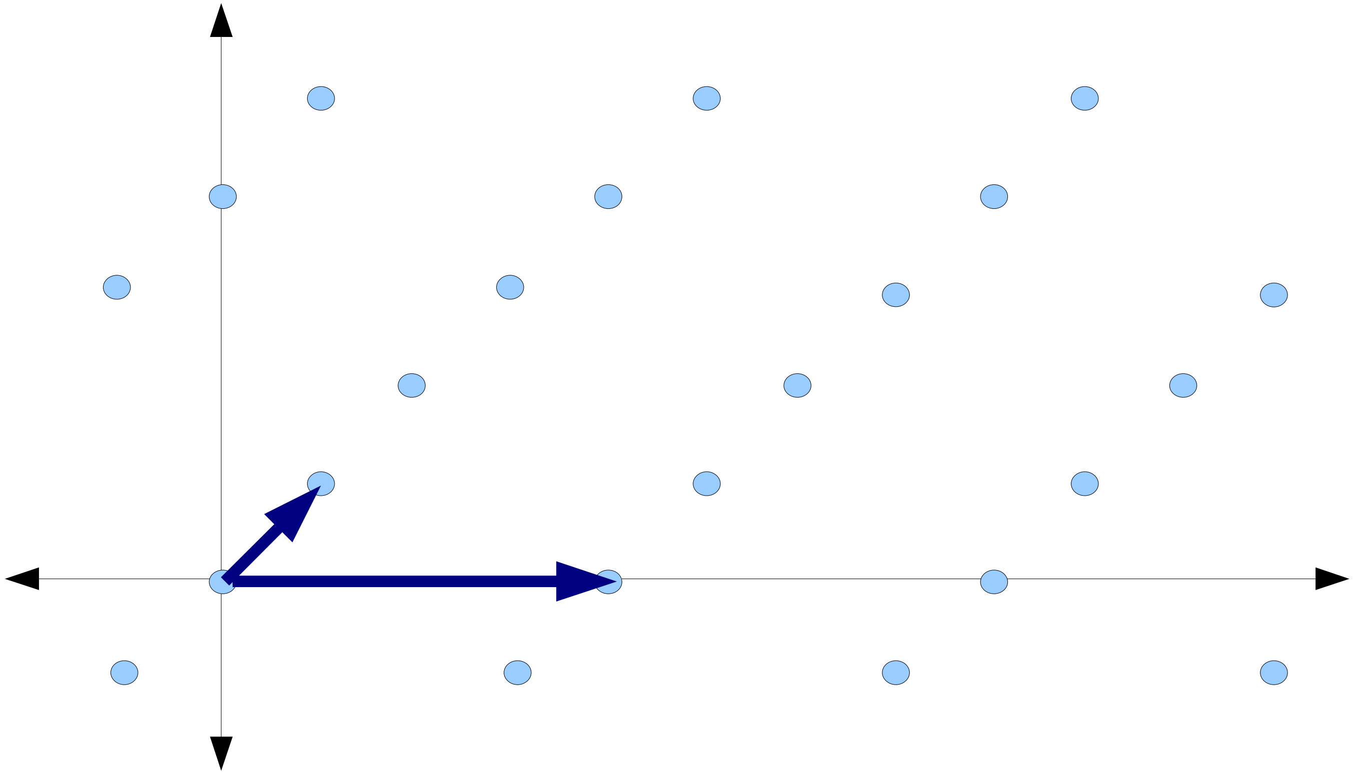


Lattice: A discrete subgroup of $R^n$

Group elements are vectors.

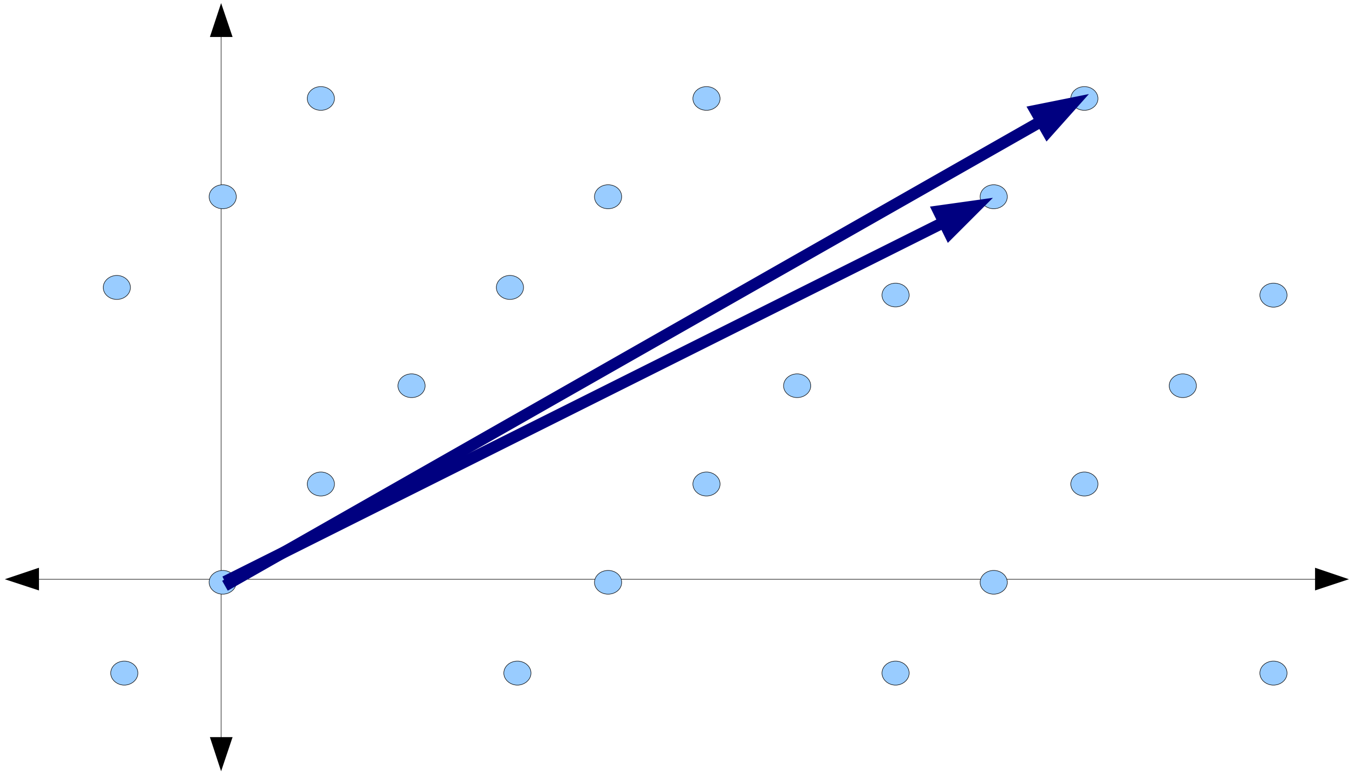
Group operation is the usual vector addition.

# Lattices



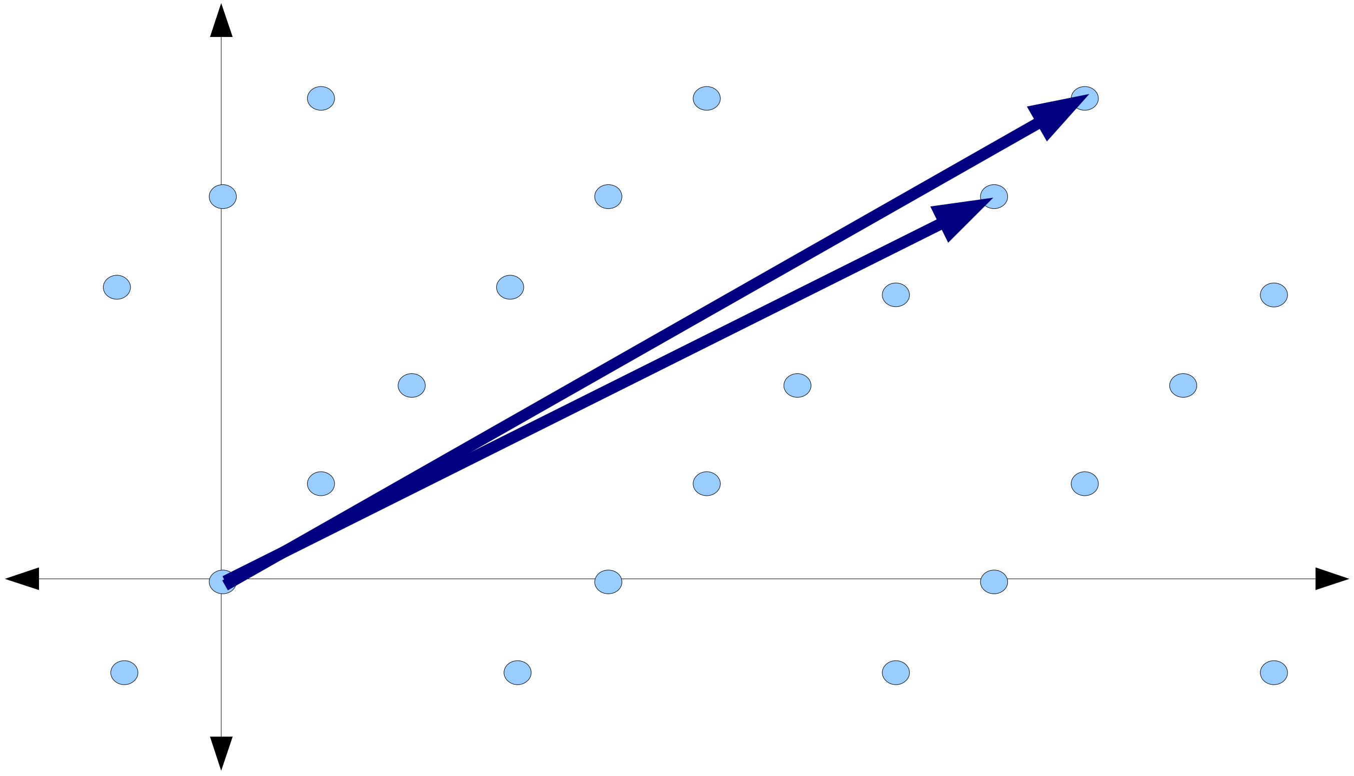
Basis: A set of linearly independent vectors that generate the lattice.

# Lattices



Basis: A set of linearly independent vectors that generate the lattice.

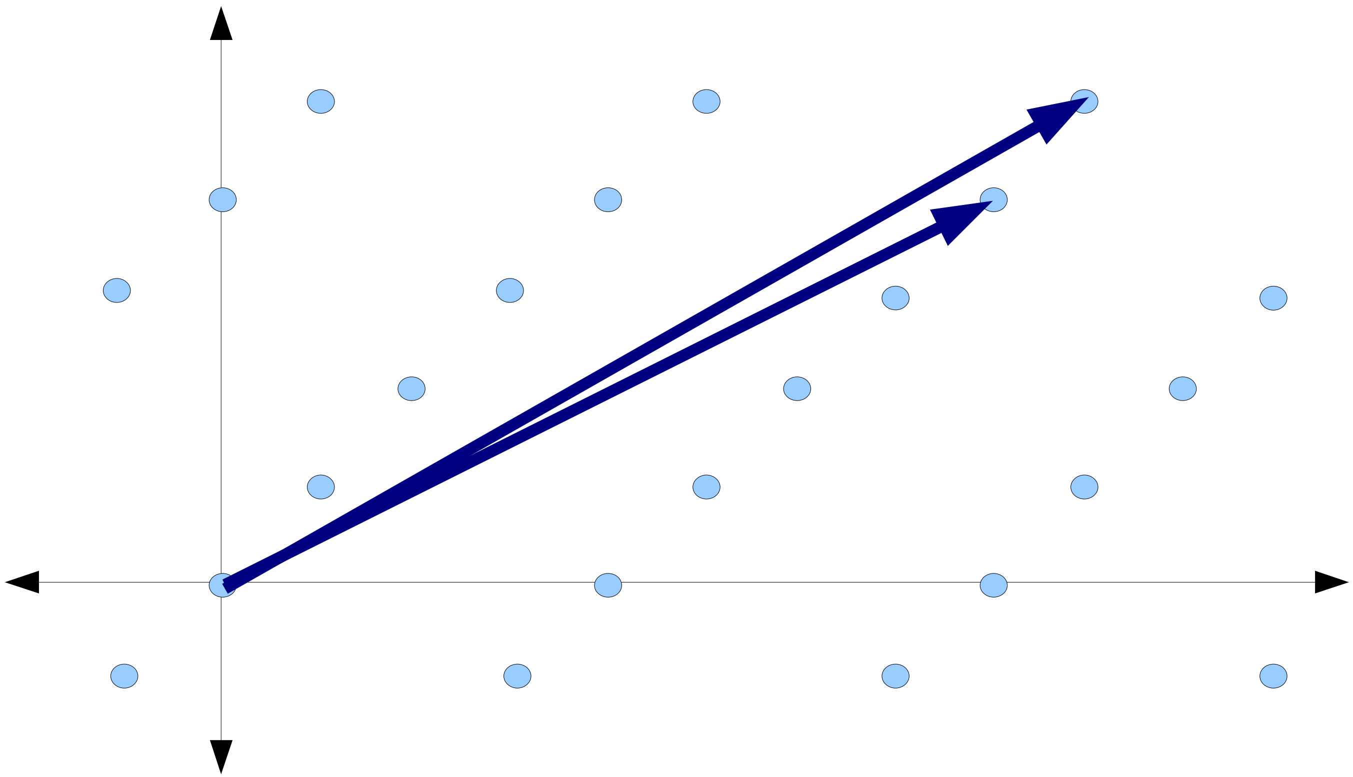
# Shortest Vector Problem



SVP(B): Given a lattice basis B, find the shortest vector

$SVP_g$(B): Given a lattice basis B, find a vector that is no more than g times longer than the shortest vector

# Minimum Distance Problem (Decision Version of SVP)



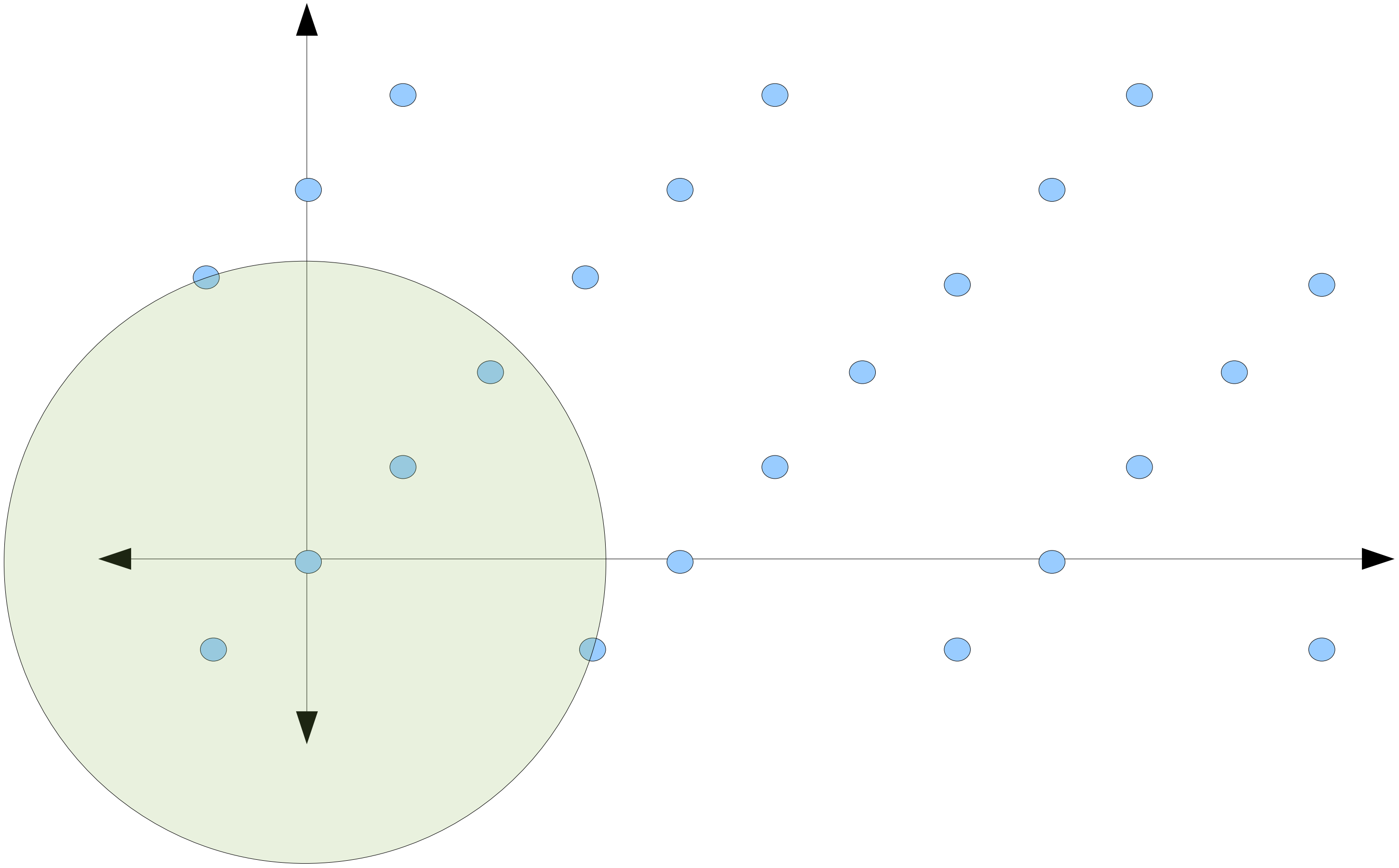
GapSVP(B,d): Given a lattice basis B, is the shortest vector at most d?

$\text{GapSVP}_g$(B,d): Given a lattice basis B, answer YES if shortest vector at most d. Answer NO if shortest vector greater than gd.

# Hardness of SVP and GapSVP

- $SVP_g$ and $GapSVP_g$ are NP-hard for any constant g [Kho '04]
- $SVP_g$ and $GapSVP_g$ can be solved for $g=2^{O(n\log\log n/\log n)}$ [LLL '82, AKS '01]
- SVP and GapSVP can be solved exactly in time $2^{O(n)}$ [AKS '01]
- $GapSVP_g < SVP_g$, but $SVP_g < GapSVP_g$ not known to exist. Big open question!

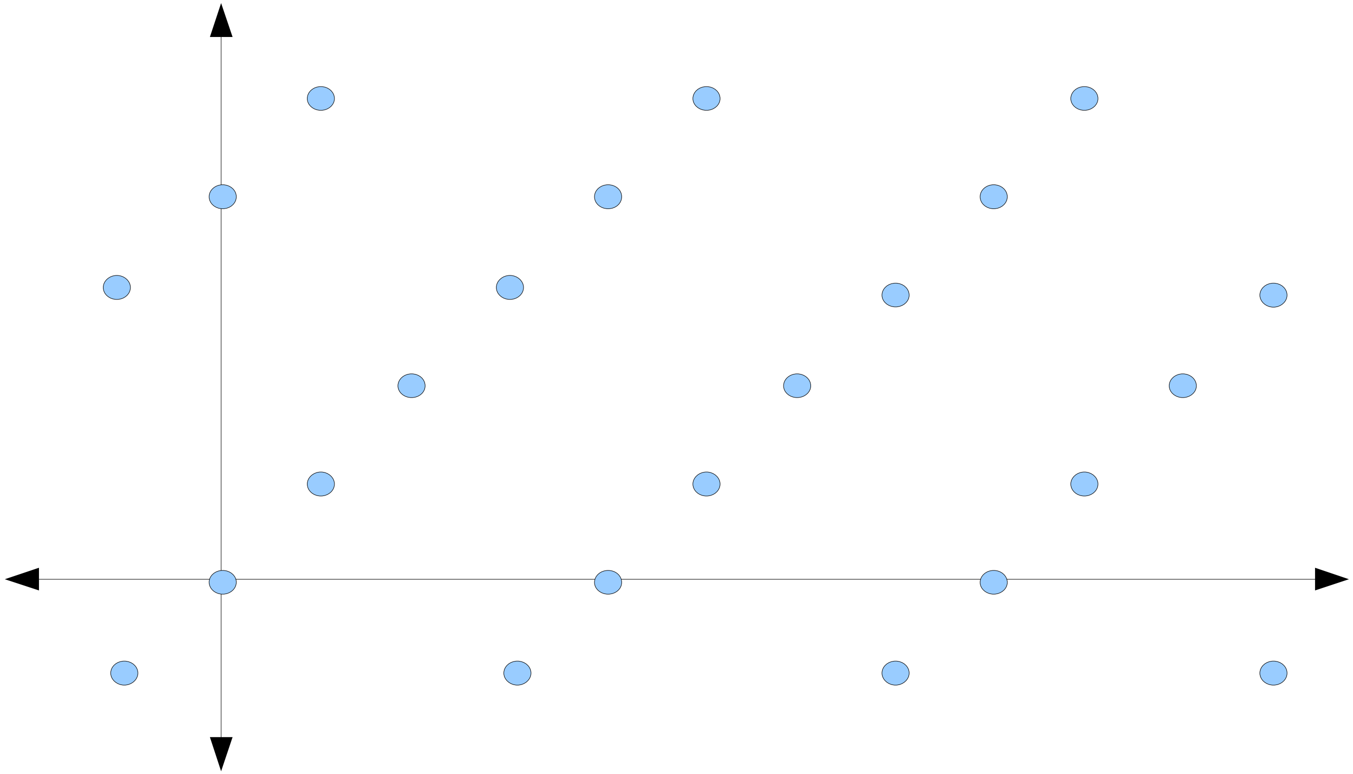
# Shortest Independent Vector Problem



r: smallest number such that ball of radius r contains n linearly independent lattice vectors

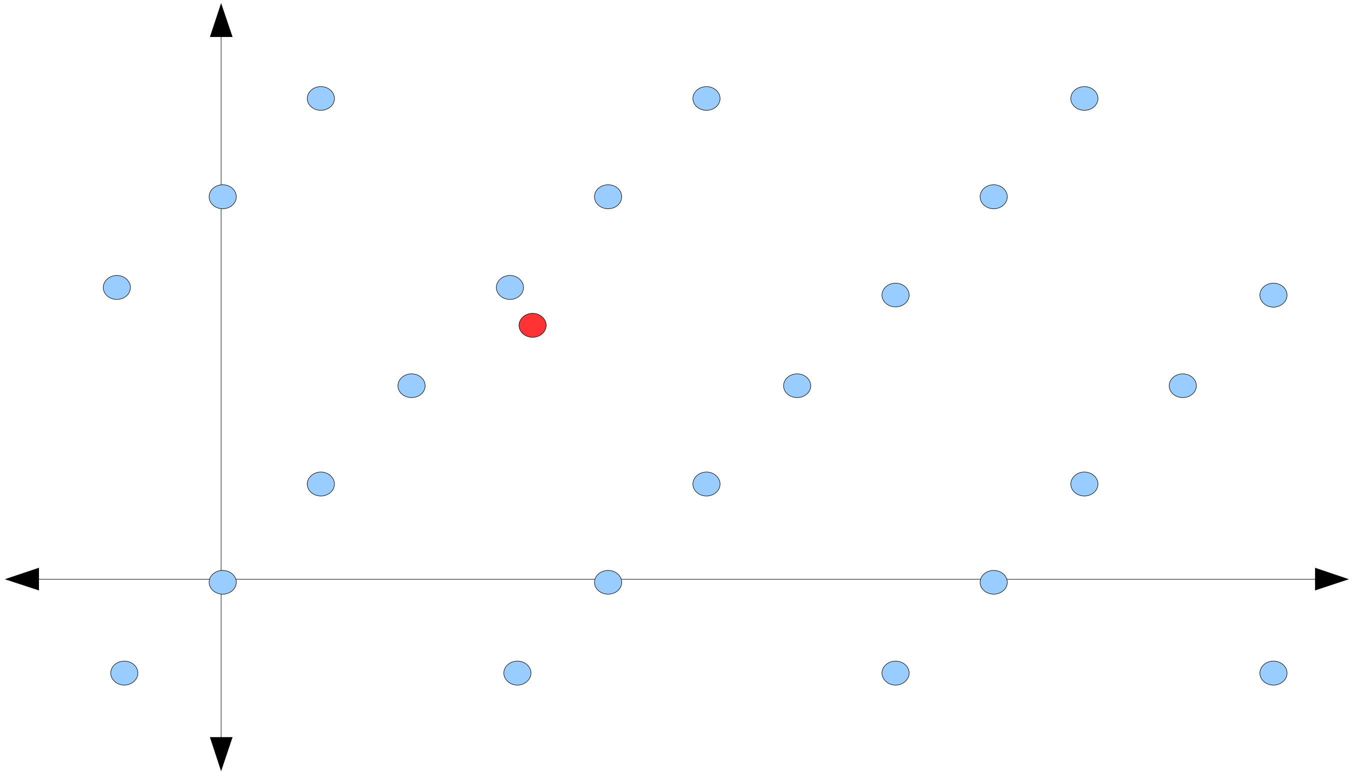
$SIVP_g(B)$: Given a lattice basis B of dimension n, find n linearly independent vectors of length at most gr

# Unique Shortest Vector Problem



$uSVP_g(B)$: Given a lattice basis B where the shortest vector is g times smaller than the second shortest linearly independent vector, find the shortest vector

# Bounded Distance Decoding



Have a lattice with minimum distance d (don't necessarily know d)

$BDD_g(B,t)$: Given a lattice basis B and a target t such that

dist(B,t)<gd, find the nearest lattice vector to t

# Why are Lattices Interesting?

(In Cryptography)

- Ajtai ('96) showed that solving *“average” instances* of lattice problems implies solving *all instances* of lattice problems
- Possible to base cryptography on worst-case instances of problems

# Lattice-Based Primitives

## Minicrypt

- One-way functions [Ajt '96]
- Collision-resistant hash functions [Ajt '96,MR '07]
- Identification schemes [MV '03,Lyu '08, KTX '08]
- Signature schemes [LM '08, GPV '08]

All Based on SIVP and GapSVP

## Public-Key Cryptosystems

- [AD '97] (uSVP)
- [Reg '03] (uSVP)

# Lattice-Based Primitives

## Minicrypt

- One-way functions [Ajt '96]
- Collision-resistant hash functions [Ajt '96,MR '07]
- Identification schemes [MV '03,Lyu '08, KTX '08]
- Signature schemes [LM '08, GPV '08]

All Based on SIVP and GapSVP

## Public-Key Cryptosystems

- [AD '97] (uSVP)
- [Reg '03] (uSVP)
- [Reg '05] (SIVP and GapSVP under quantum reductions)
- [Pei '09] (GapSVP)

# Cryptosystem Hardness Assumptions

| | uSVP | BDD | GapSVP | SIVP (quantum) |
|---|---|---|---|---|
| Ajtai-Dwork '97 | $O(n^2)$ | | | |
| Regev '03 | $O(n^{1.5})$ | | | |
| Regev '05 | | | | $O(n^{1.5})$ |
| Peikert '09 | | $O(n^{1.5})$ | $O(n^2)$ | $O(n^{2.5})$ |

# Cryptosystem Hardness Assumptions

| | uSVP | BDD | GapSVP | SIVP (quantum) |
|---|---|---|---|---|
| Ajtai-Dwork '97 | $O(n^{2})$ | $O(n^{2})$ | $O(n^{2.5})$ | $O(n^{3})$ |
| Regev '03 | $O(n^{1.5})$ | $O(n^{1.5})$ | $O(n^{2})$ | $O(n^{2.5})$ |
| Regev '05 | - | - | - | $O(n^{1.5})$ |
| Peikert '09 | $O(n^{1.5})$ | $O(n^{1.5})$ | $O(n^{2})$ | $O(n^{2.5})$ |

Implications of our results

# Lattice-Based Primitives

## Minicrypt

- One-way functions [Ajt '96]
- Collision-resistant hash functions [Ajt '96,MR '07]
- Identification schemes [MV '03,Lyu '08, KTX '08]
- Signature schemes [LM '08, GPV '08]

All Based on GapSVP and SIVP

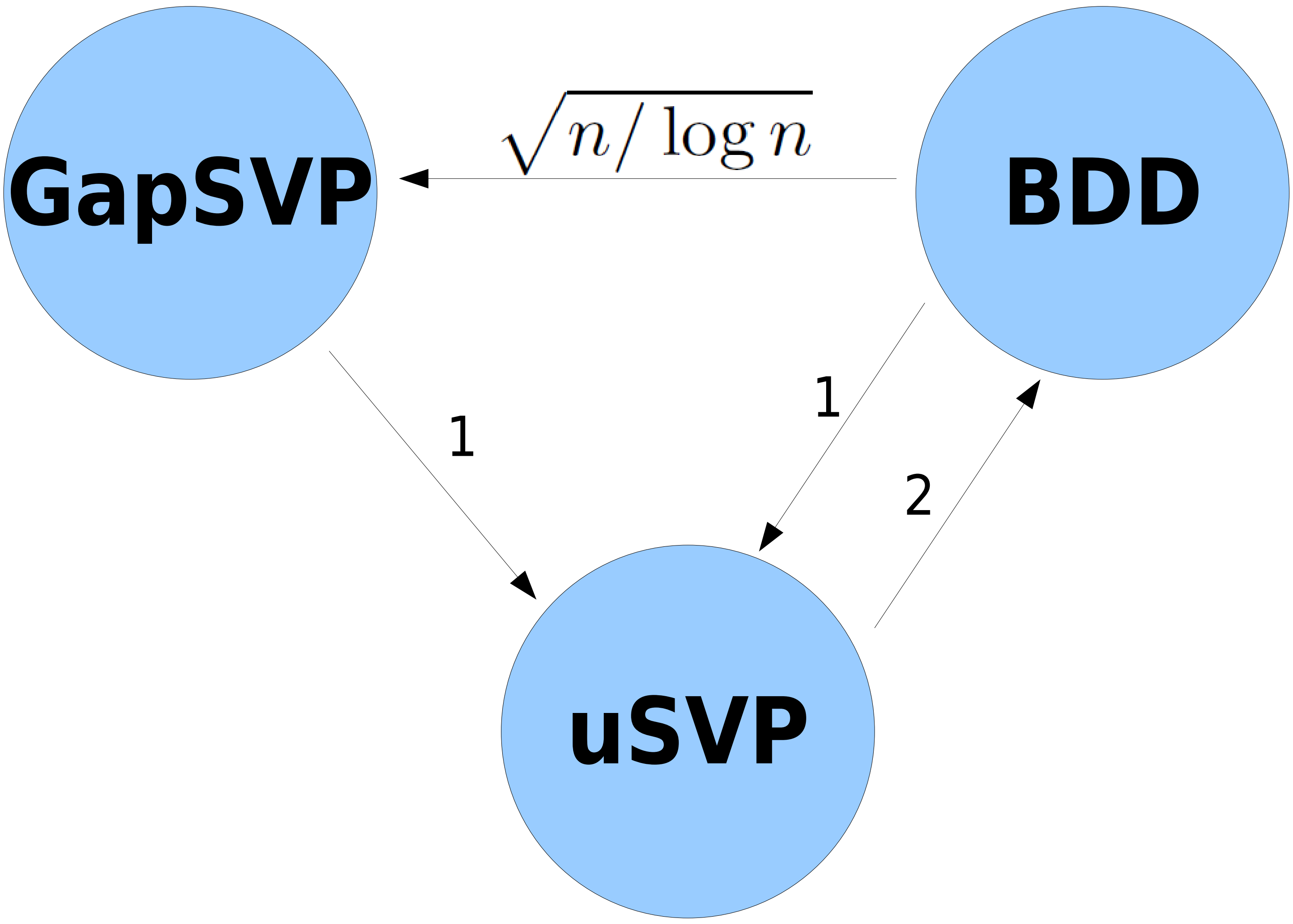
## Public-Key Cryptosystems

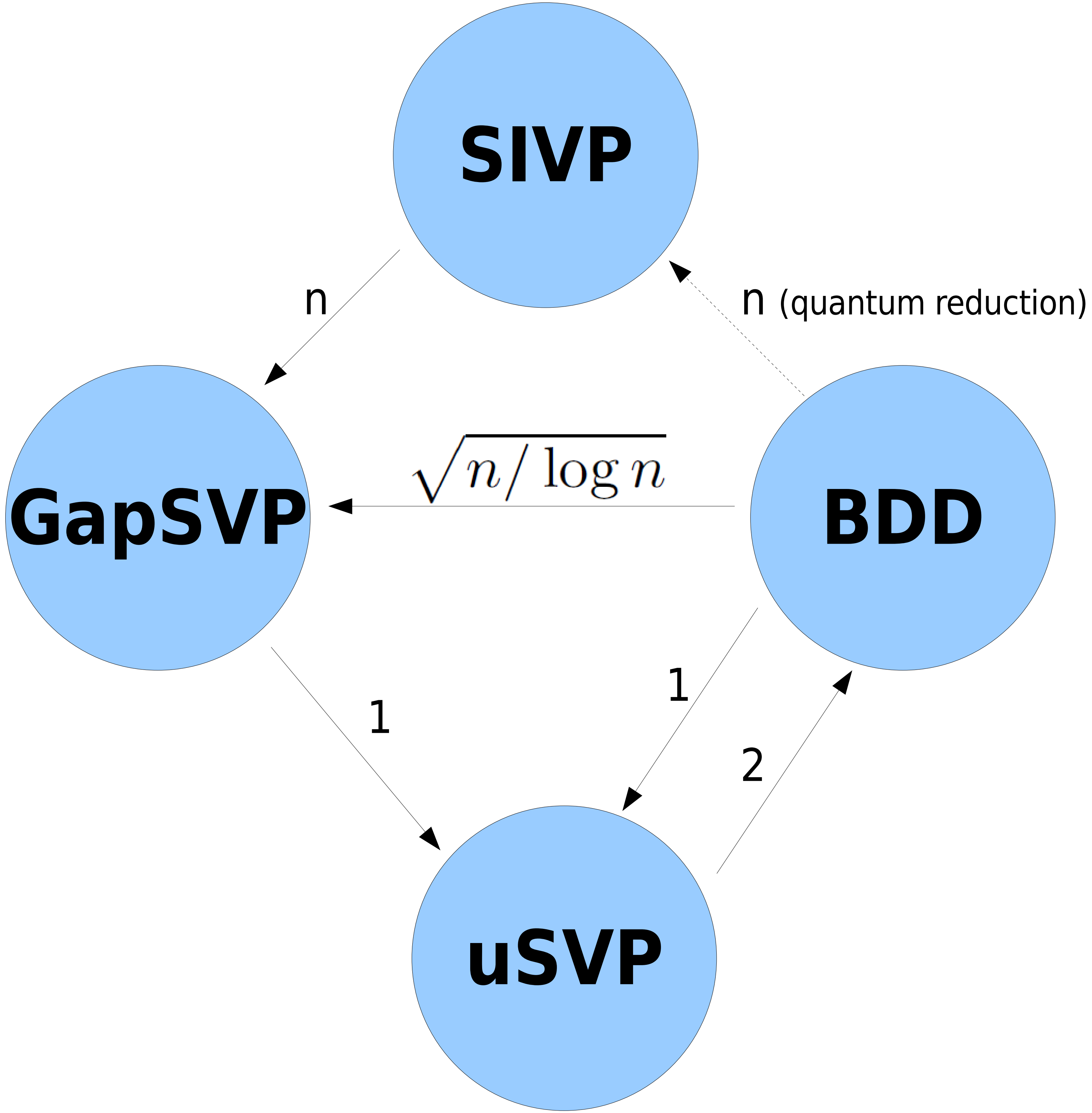
- [AD '97] (uSVP)
- [Reg '03] (uSVP)
- [Reg '05] (SIVP and GapSVP under quantum reductions)
- [Pei '09] (GapSVP)

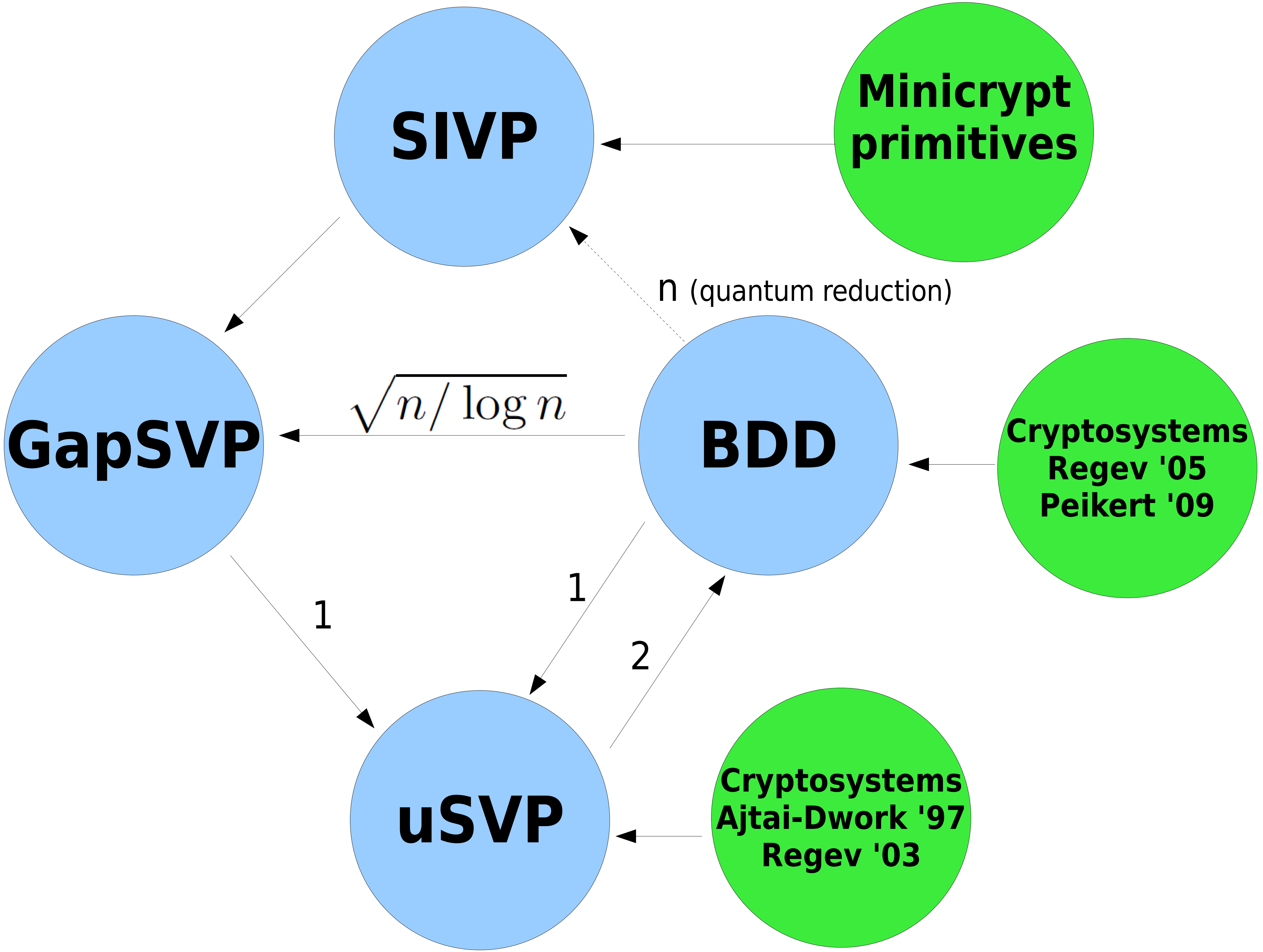
All Based on GapSVP and quantum SIVP

**Major Open Problem:**
**Construct cryptosystems based on SIVP**

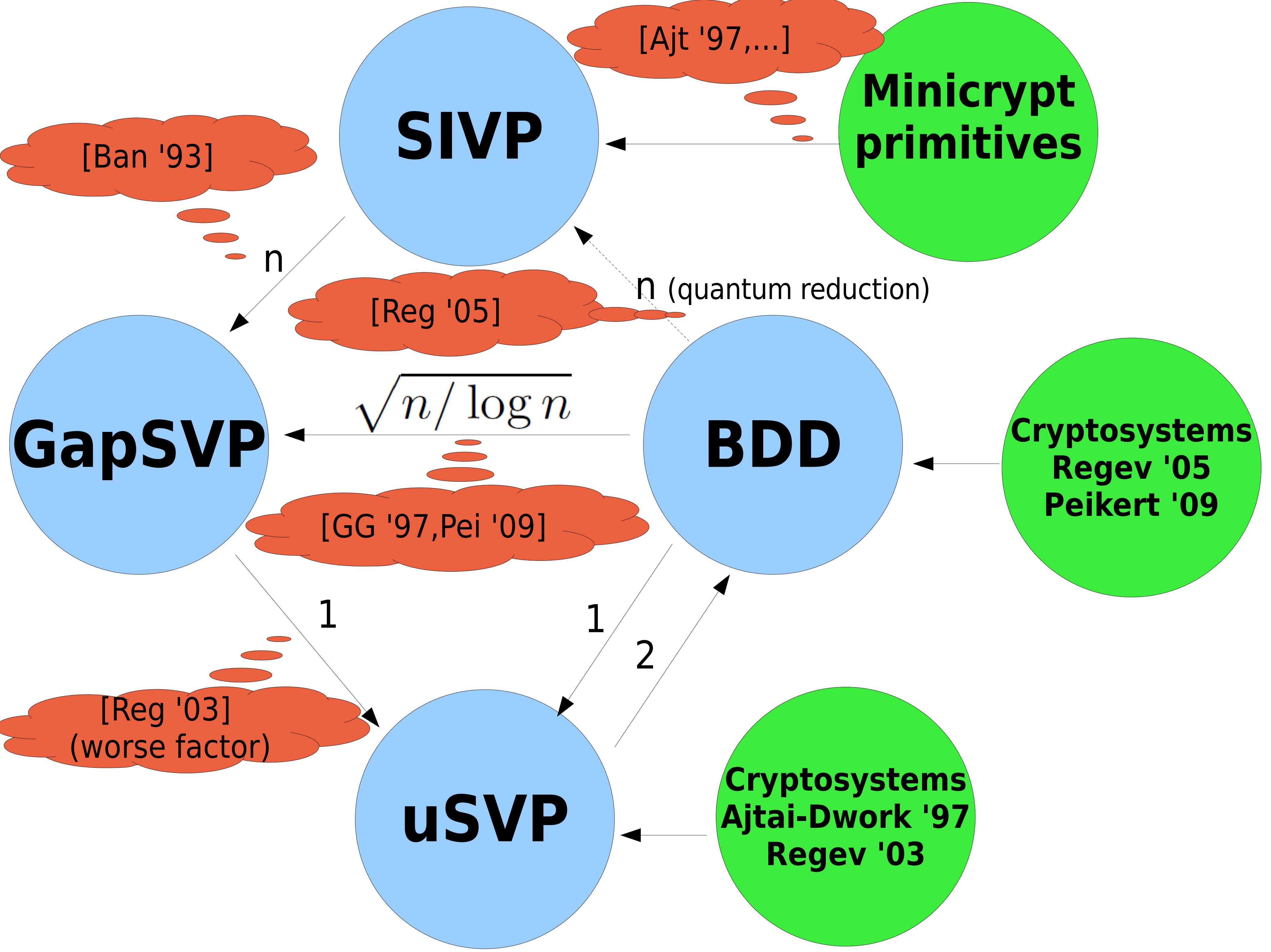
# Reductions

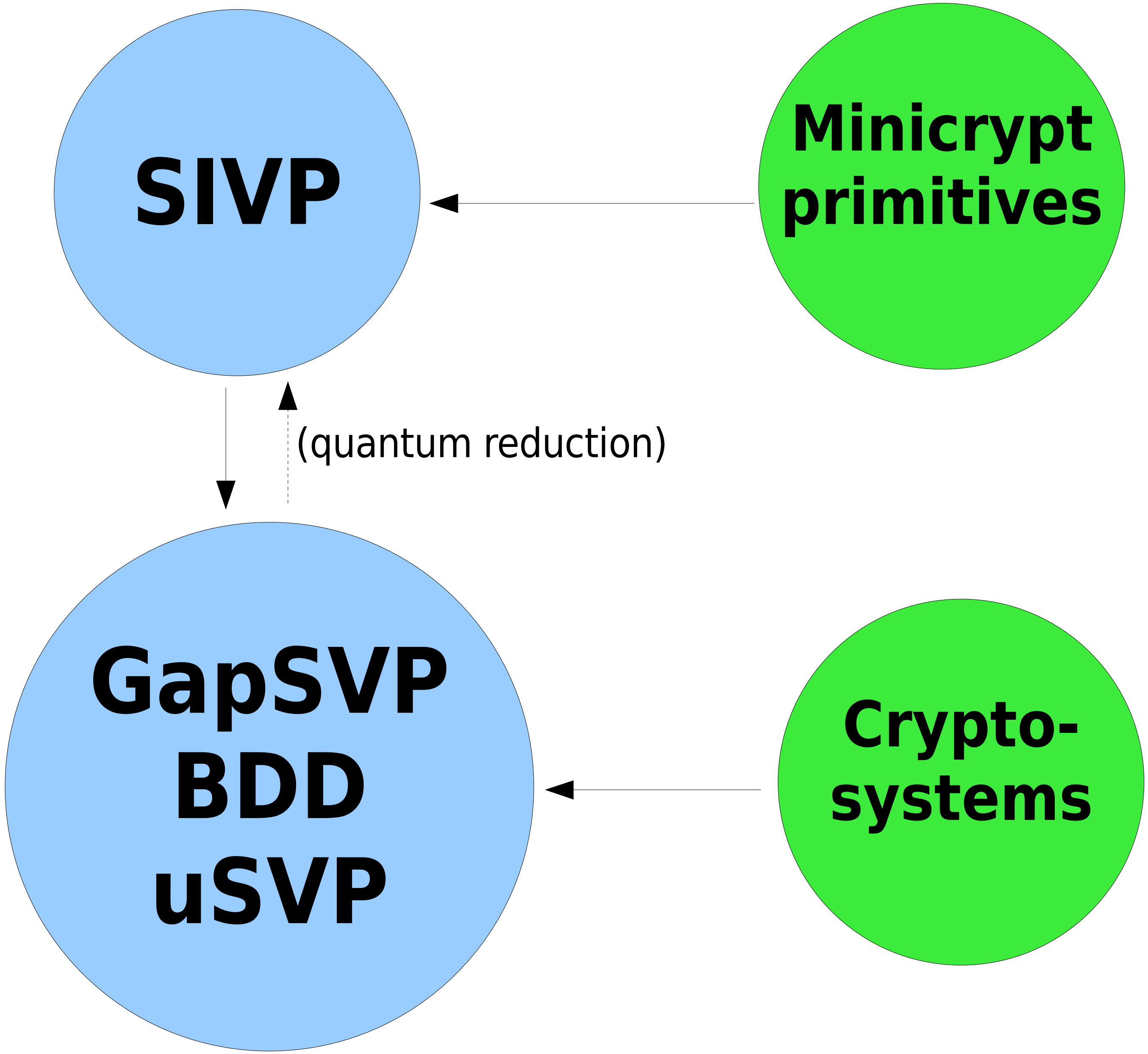



SIVP
n
n (quantum reduction)
GapSVP
$\sqrt{n/\log n}$
BDD
1
1
2
uSVP



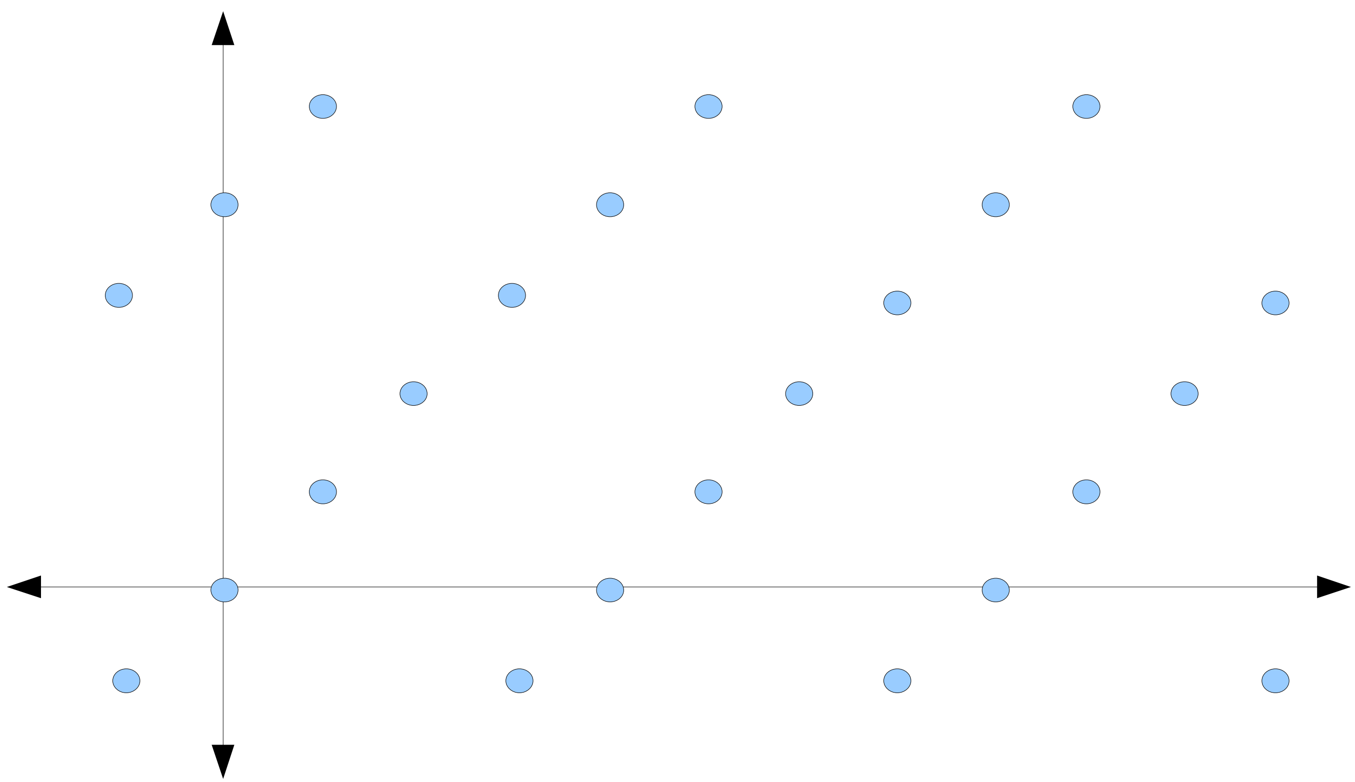
SIVP
Minicrypt primitives
n (quantum reduction)
GapSVP
$\sqrt{n/\log n}$
BDD
Cryptosystems Regev '05 Peikert '09
1
1
2
uSVP
Cryptosystems Ajtai-Dwork '97 Regev '03


[Ajt '97,...]
Minicrypt primitives
SIVP
[Ban '93]
n
[Reg '05]
n (quantum reduction)
GapSVP
$\sqrt{n/\log n}$
BDD
Cryptosystems
Regev '05
Peikert '09
[GG '97,Pei '09]
1
1
2
[Reg '03]
(worse factor)
uSVP
Cryptosystems
Ajtai-Dwork '97
Regev '03


SIVP
Minicrypt primitives
(quantum reduction)
GapSVP
BDD
uSVP
Crypto-systems

# Proof Sketch (GapSVP<BDD)

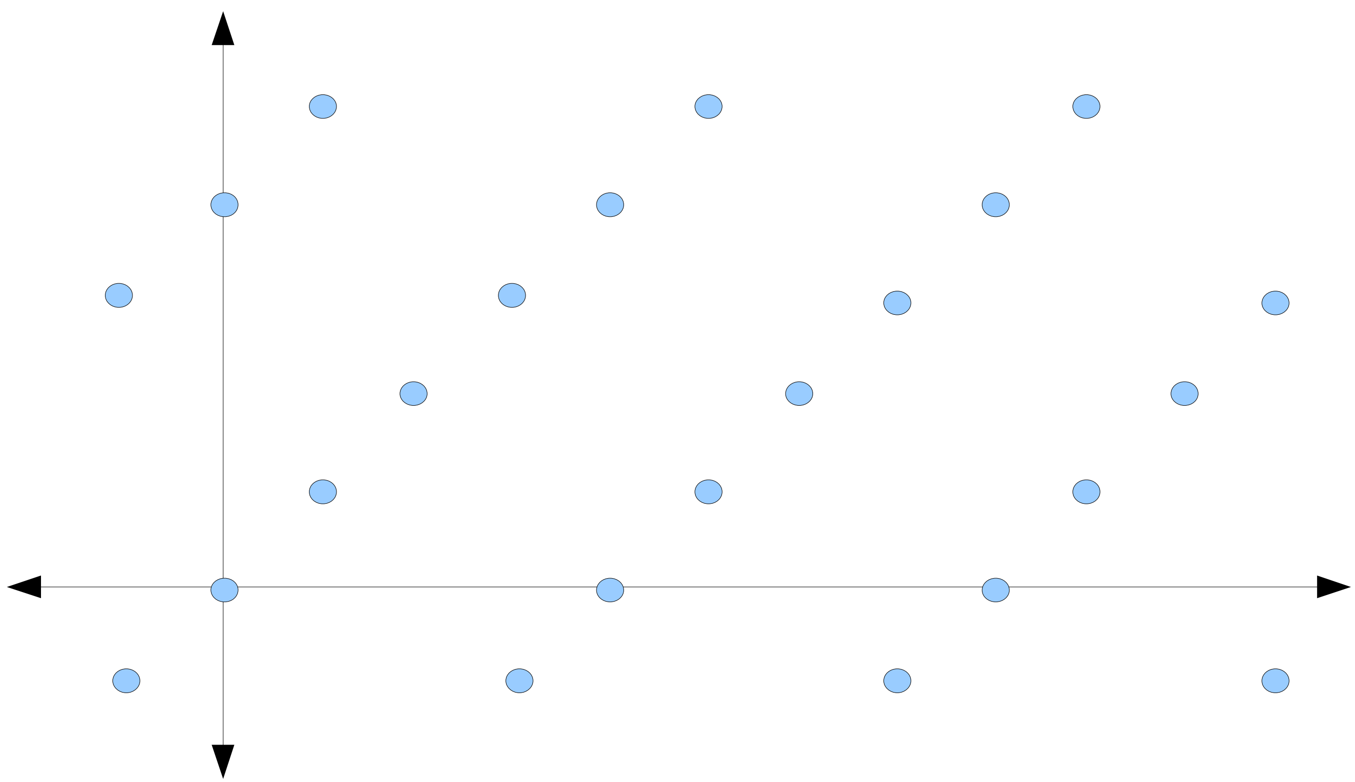
Recall the Goldreich-Goldwasser proof that GapSVP$_{\sqrt{n/\log n}}$ is in coAM



# Proof Sketch (GapSVP<BDD)

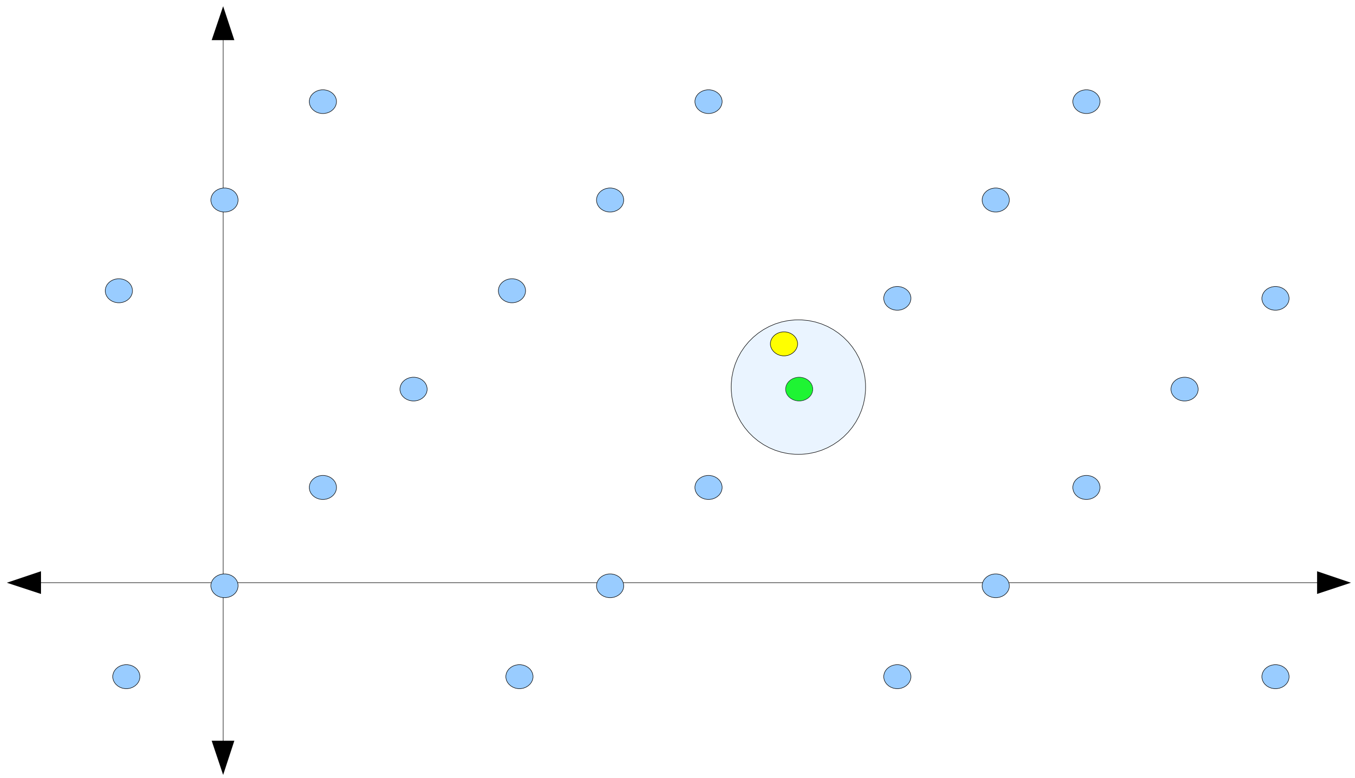
Recall the Goldreich-Goldwasser proof that GapSVP$_{\sqrt{n/\log n}}$ is in coAM

Suppose the minimum distance is d



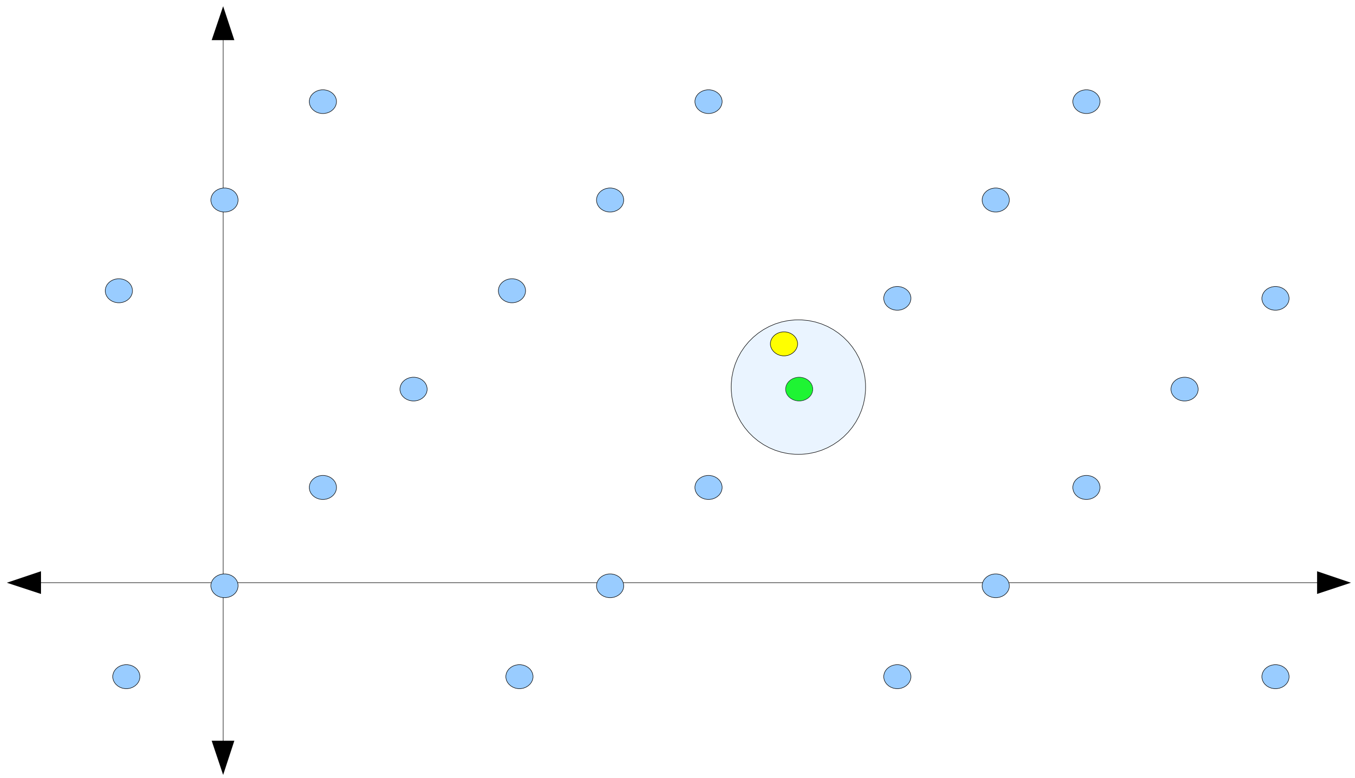
# Proof Sketch (GapSVP<BDD)

1. Verifier picks a random lattice point and a random point within distance d/2 of the lattice point. Sends the random point to the prover.



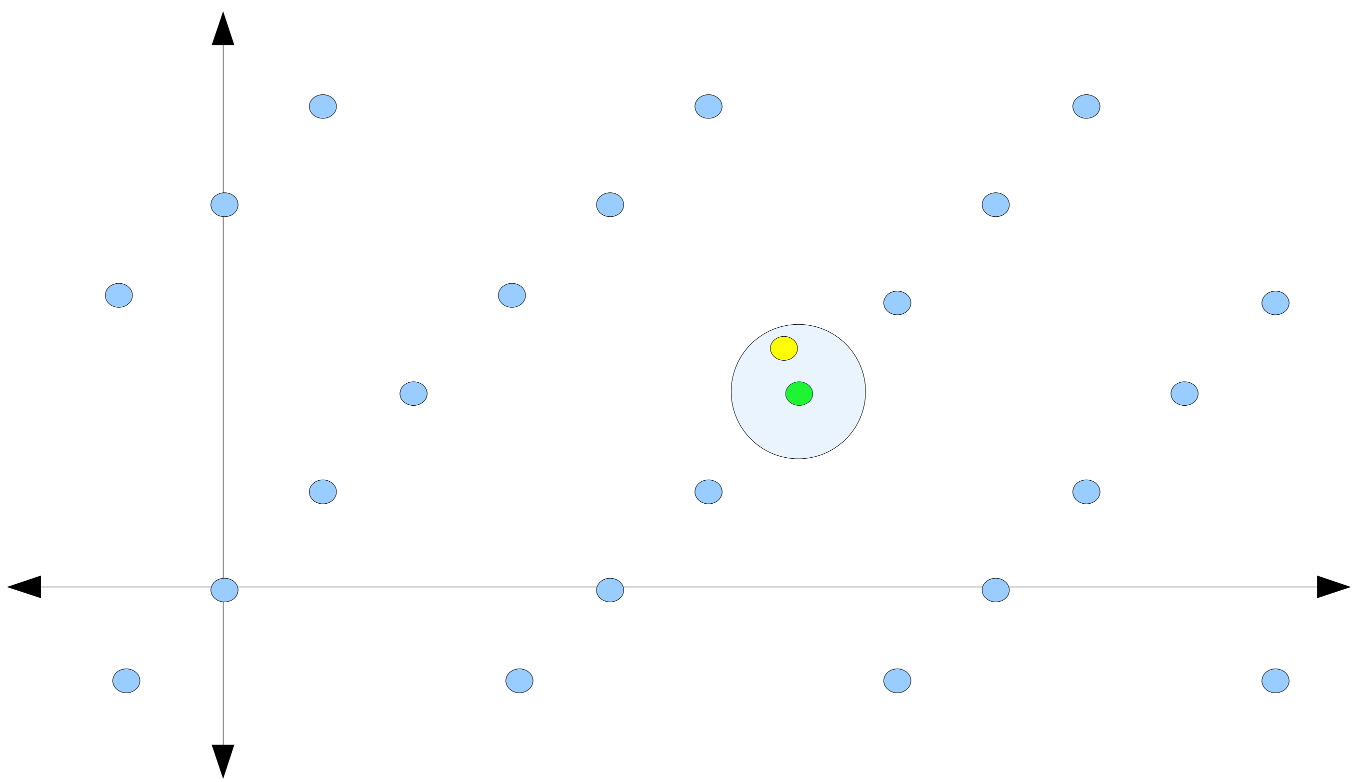
# Proof Sketch (GapSVP<BDD)

1. Verifier picks a random lattice point and a random point within distance d/2 of the lattice point. Sends the random point to the prover.
2. Prover finds the closest lattice point and sends it to the verifier



# Proof Sketch (GapSVP<BDD)

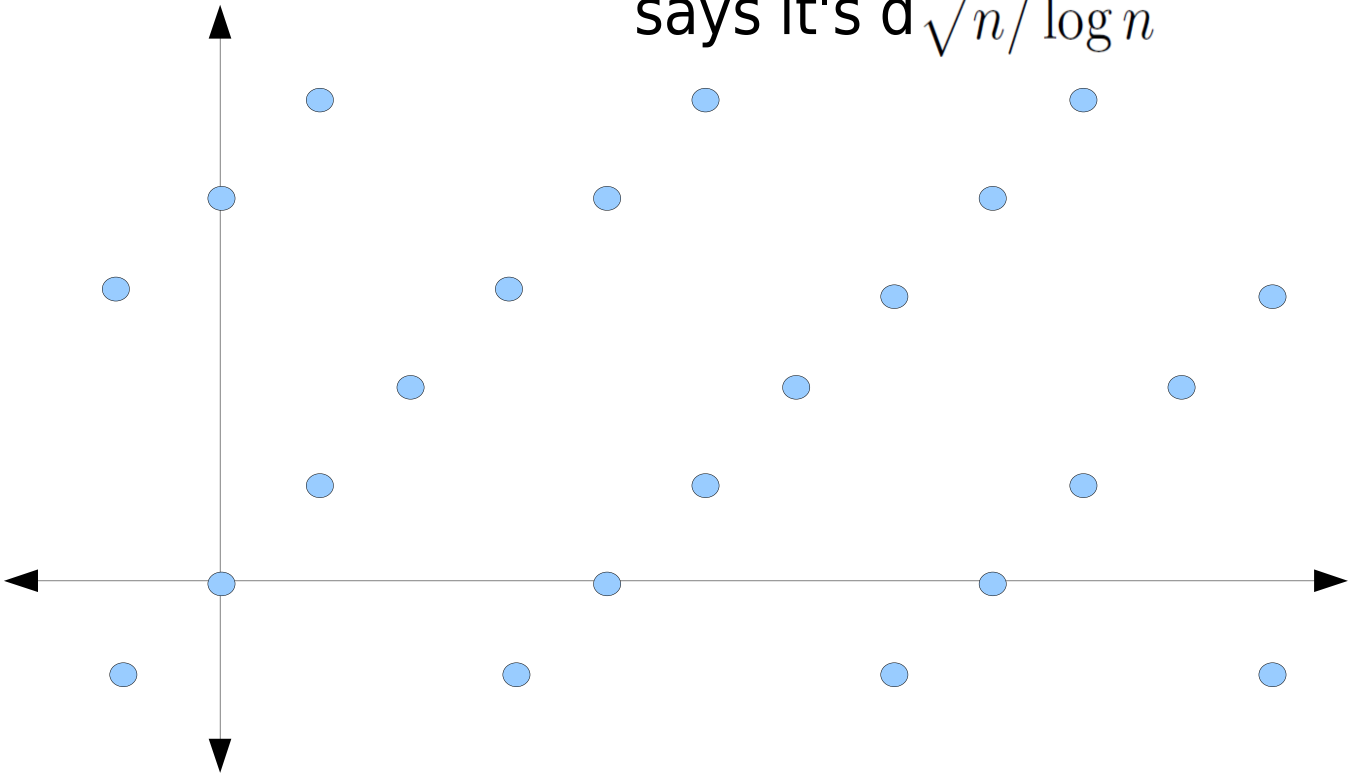
1. Verifier picks a random lattice point and a random point within distance d/2 of the lattice point. Sends the random point to the prover.
2. Prover finds the closest lattice point and sends it to the verifier
3. Verifier accepts iff he got back his own lattice point



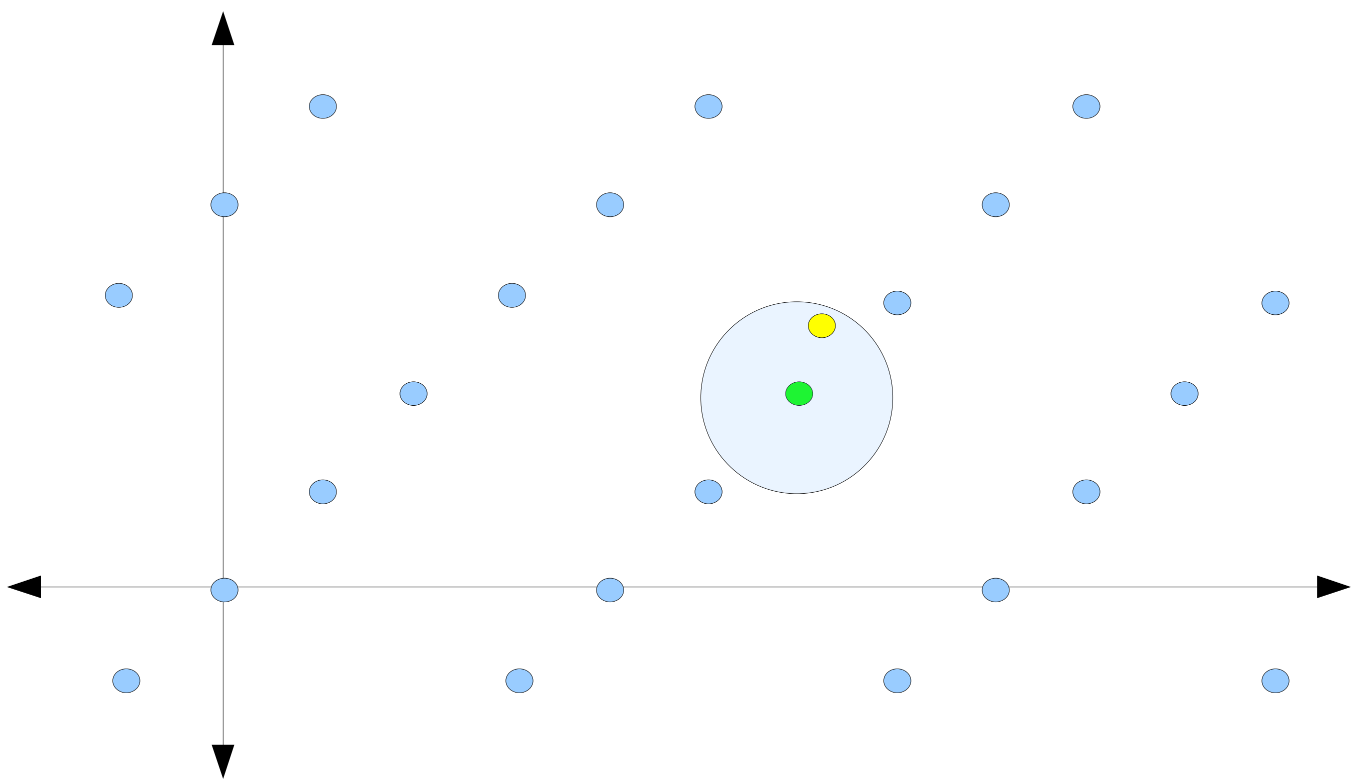
# Proof Sketch (GapSVP<BDD)

Recall the Goldreich-Goldwasser proof that GapSVP$_{\sqrt{n/\log n}}$ is in coAM

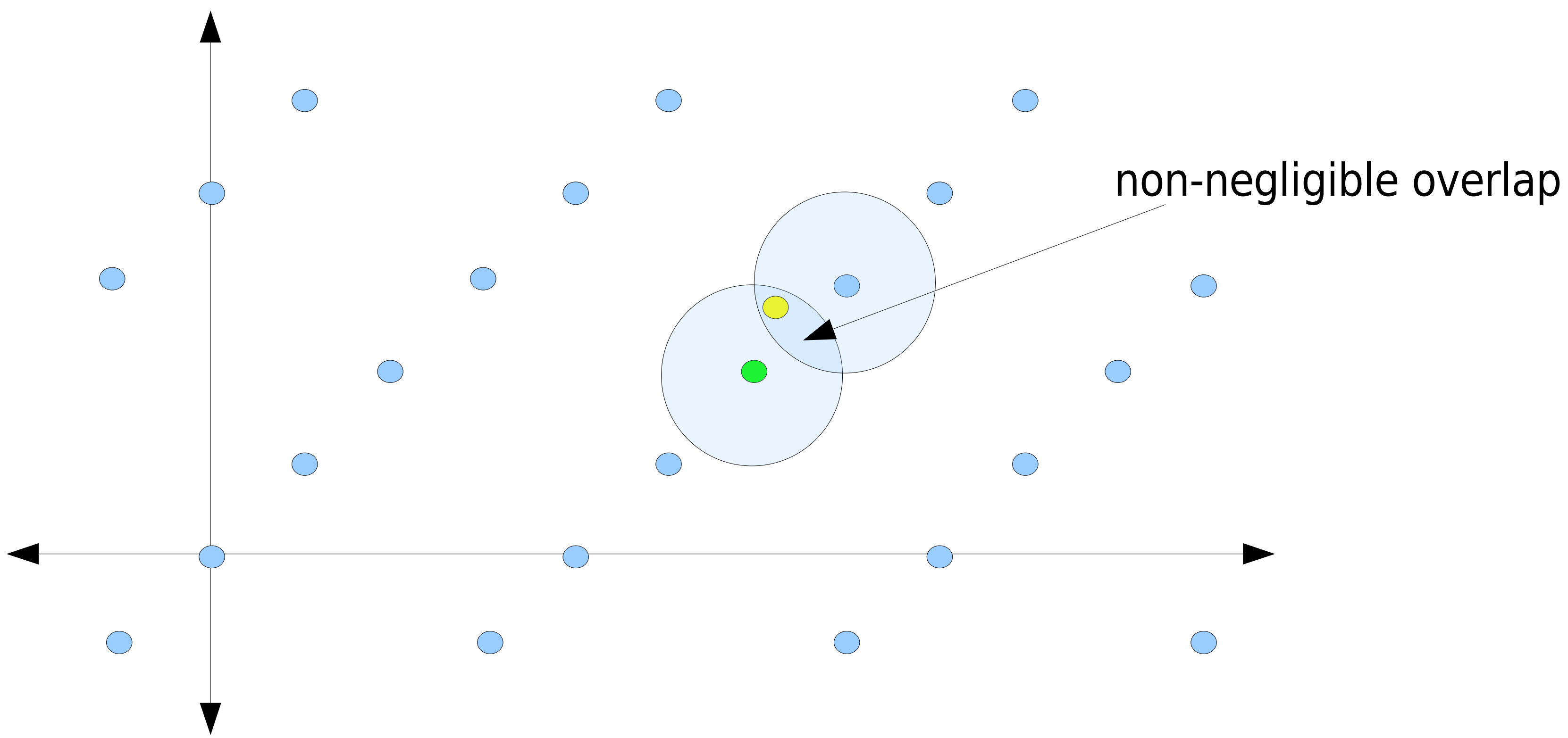
Suppose the minimum distance is d, but the prover cheats and says it's d$\sqrt{n/\log n}$



# Proof Sketch (GapSVP<BDD)

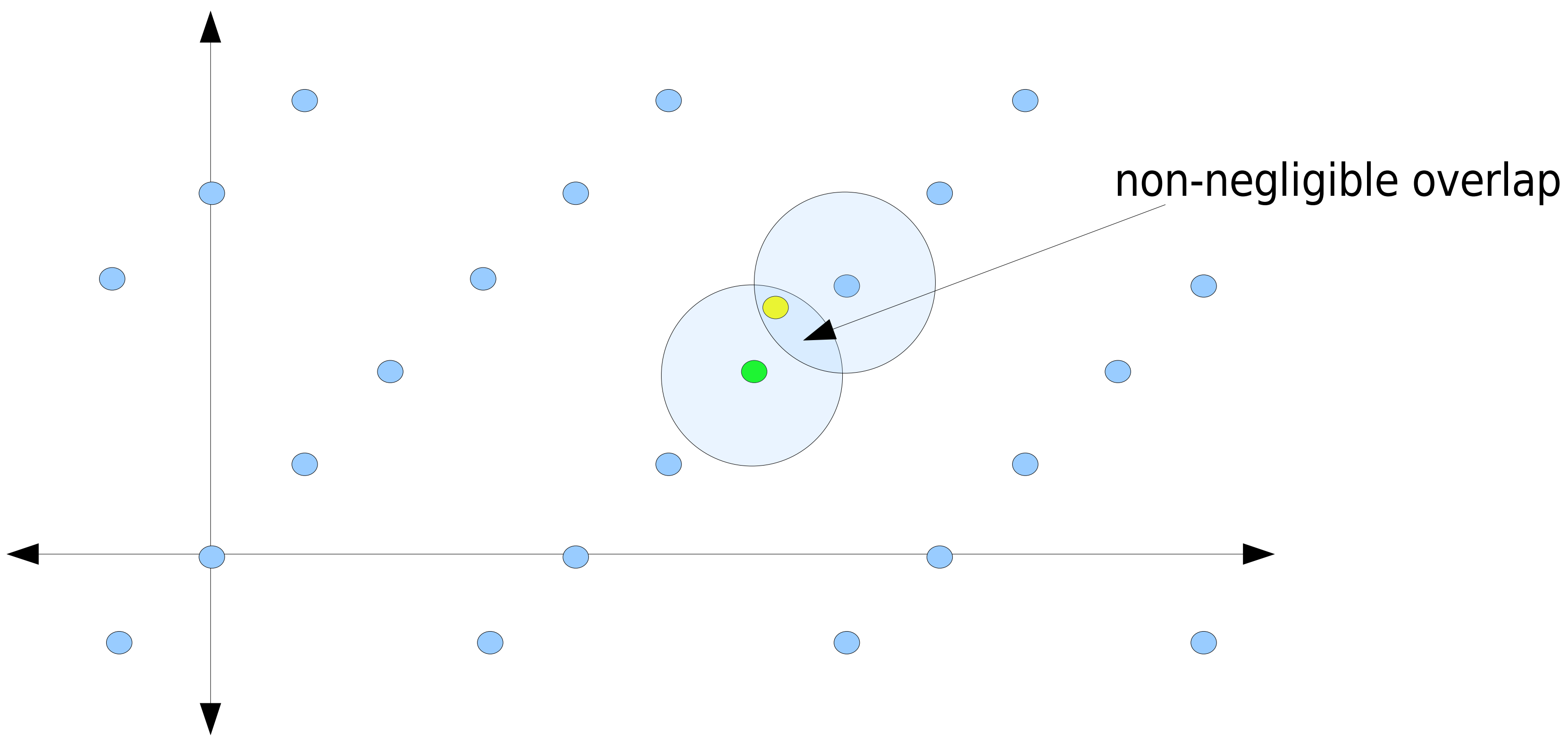


# Proof Sketch (GapSVP<BDD)



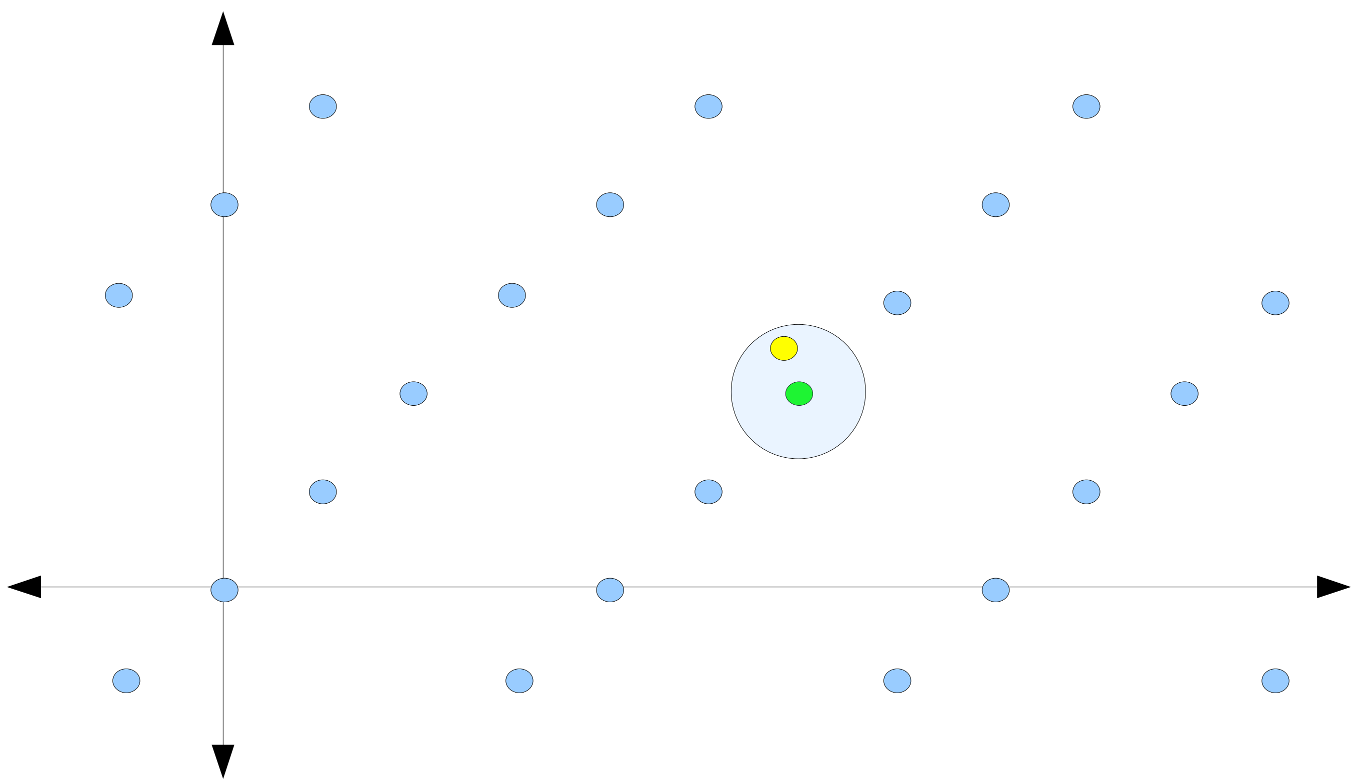
# Proof Sketch (GapSVP<BDD)

Prover will make mistakes a non-negligible fraction of the time



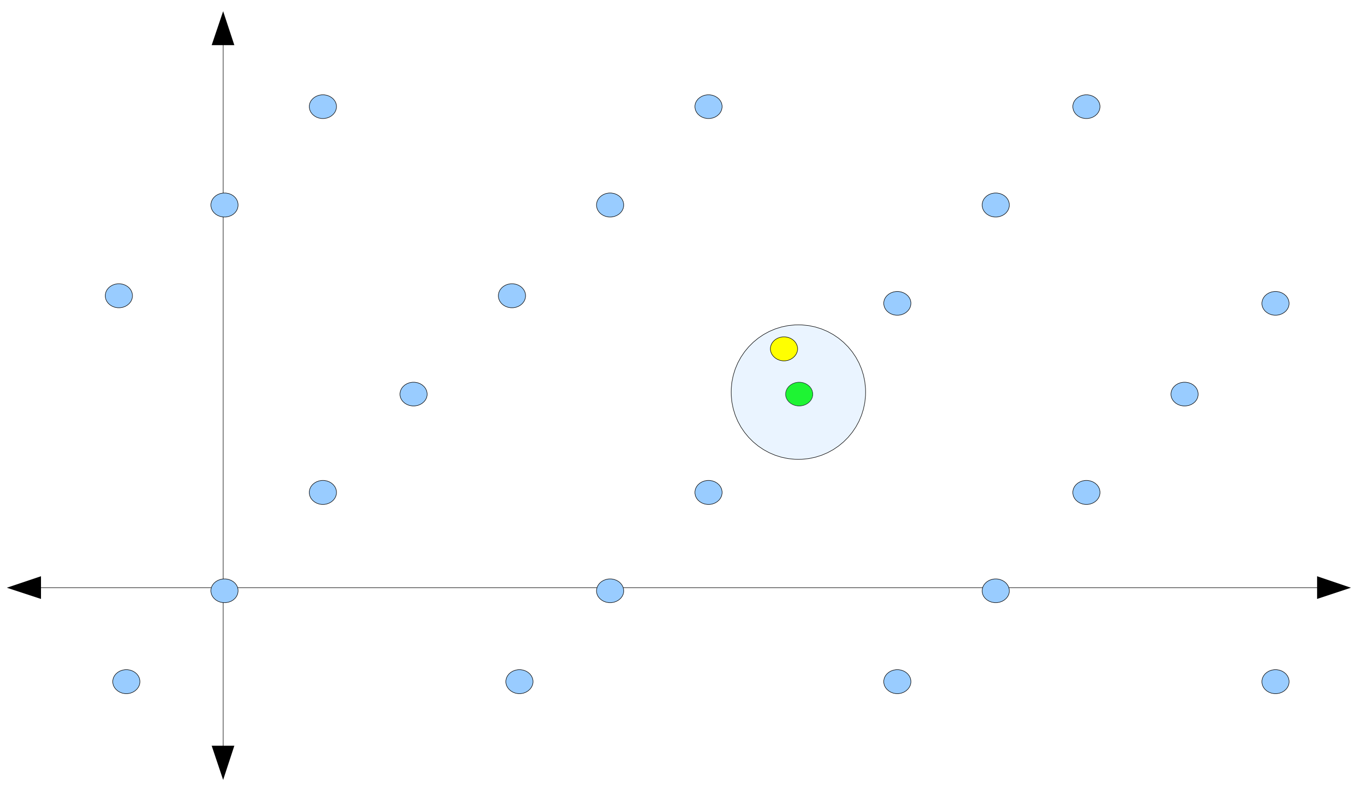
# Proof Sketch (GapSVP<BDD)

How powerful must the prover be to always return the correct point?



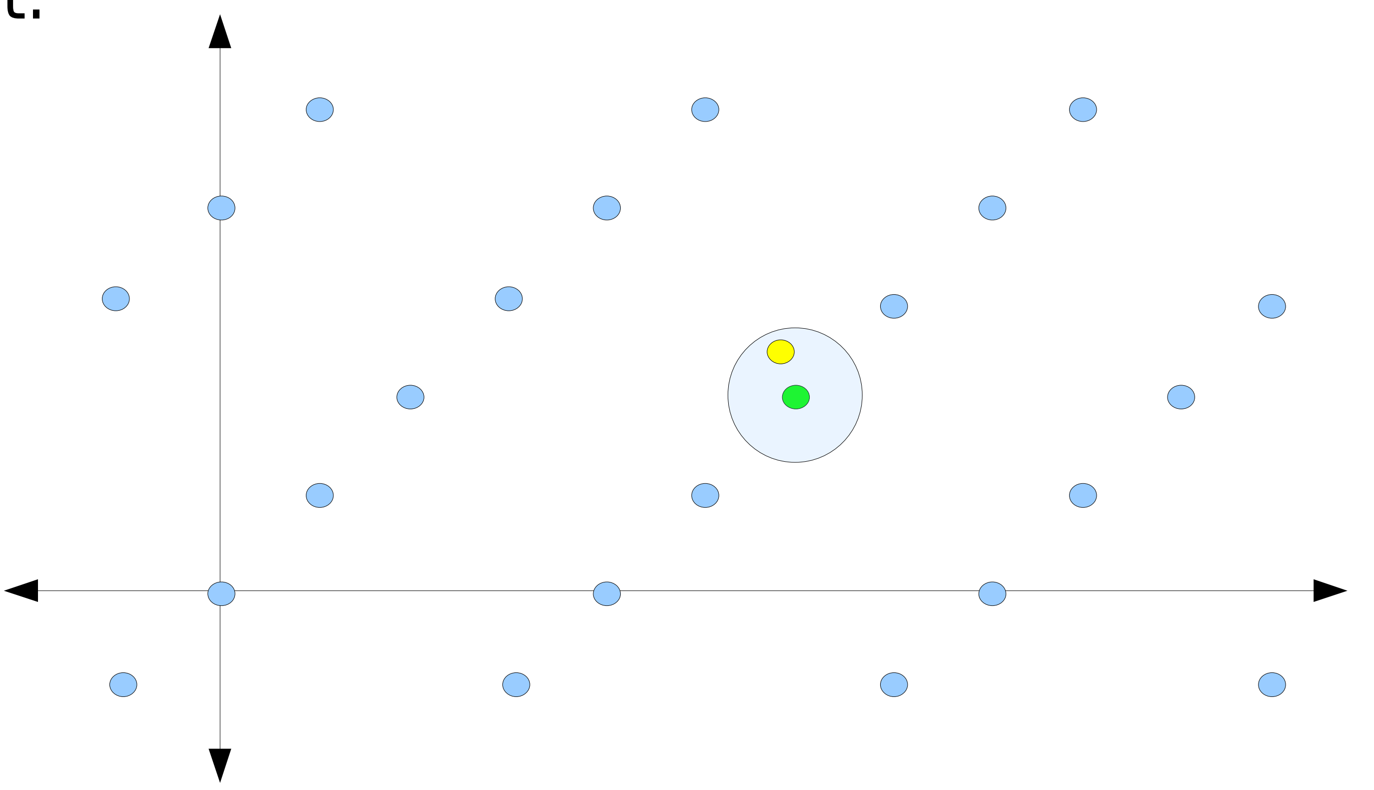
# Proof Sketch (GapSVP<BDD)

Prover just needs to be a BDD oracle [Peikert '09]

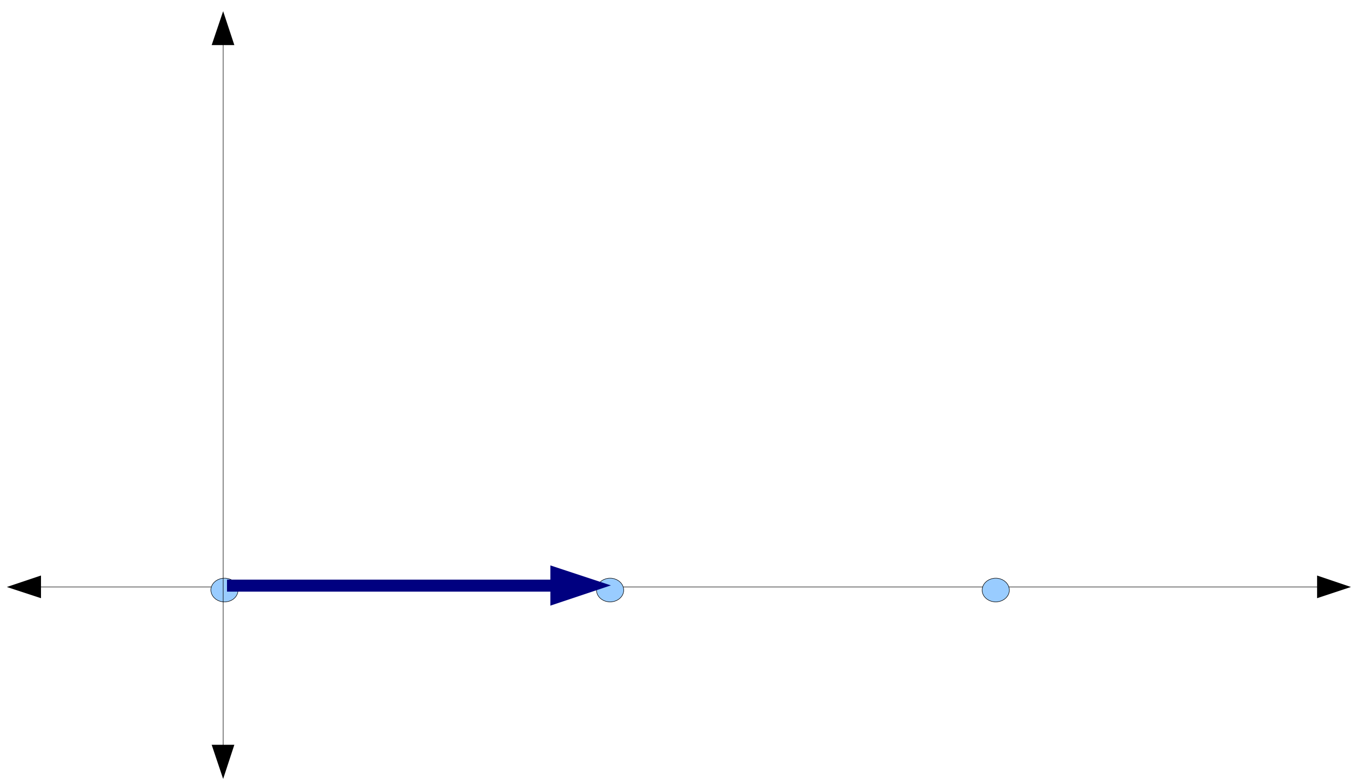


# Proof Sketch (GapSVP<BDD)

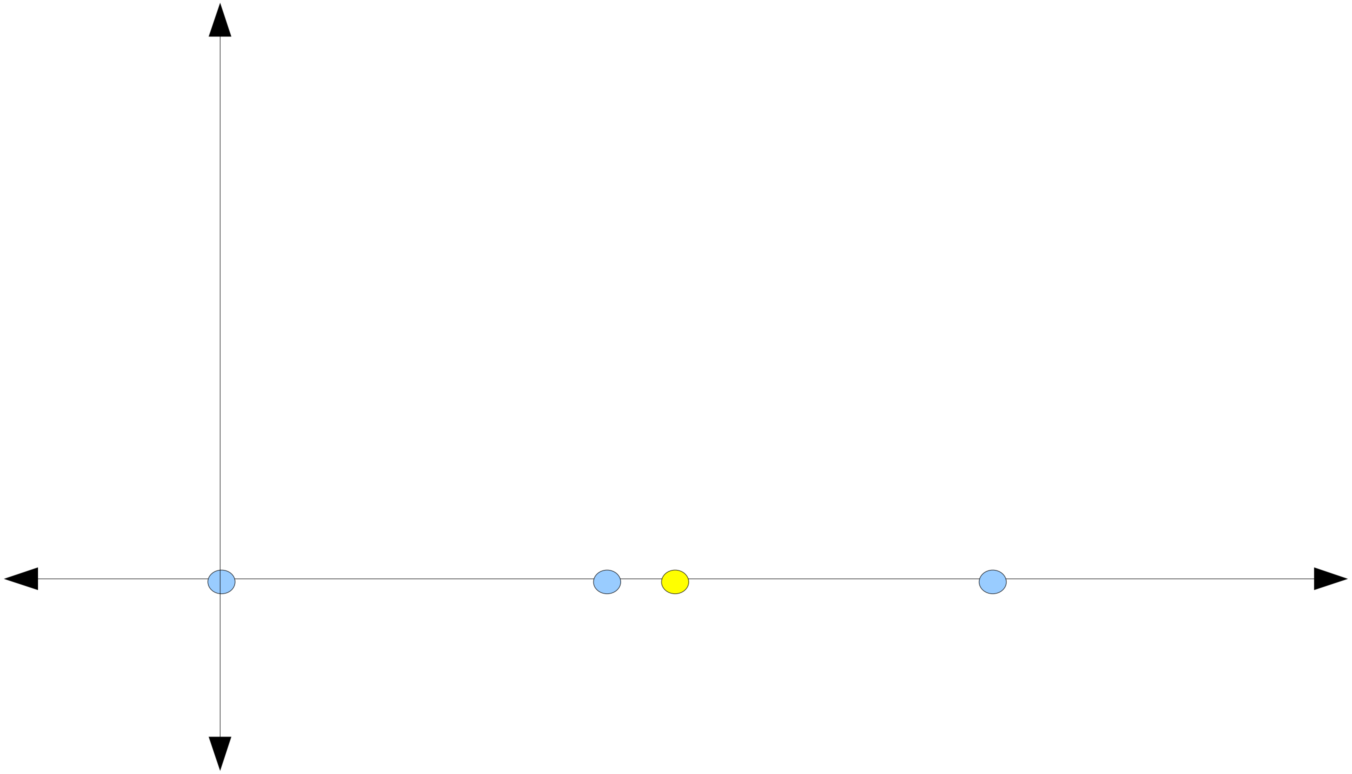
So, if you have a BDD oracle, just play the game by yourself. Create a random lattice point, add noise, use BDD oracle, and see if you always get back your lattice point.



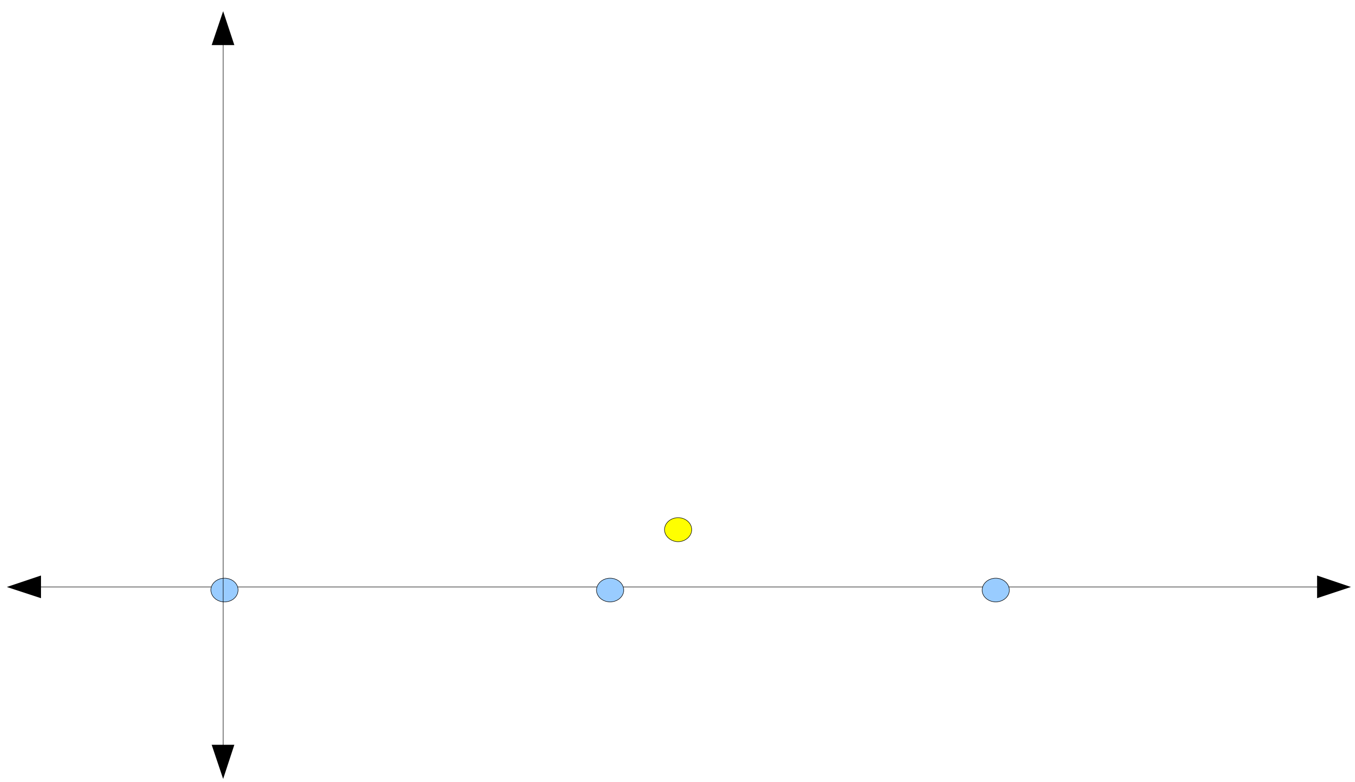
# Proof Sketch (BDD < uSVP)



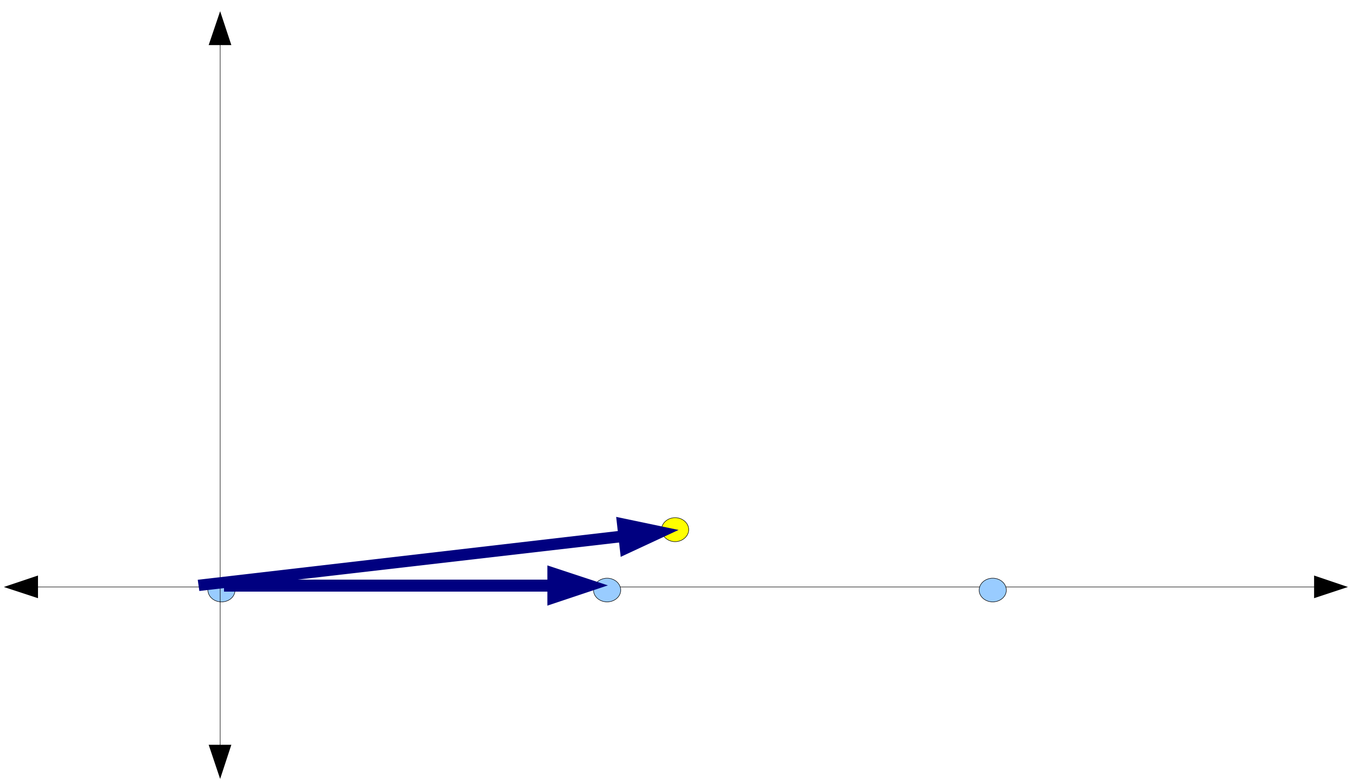
# Proof Sketch (BDD < uSVP)



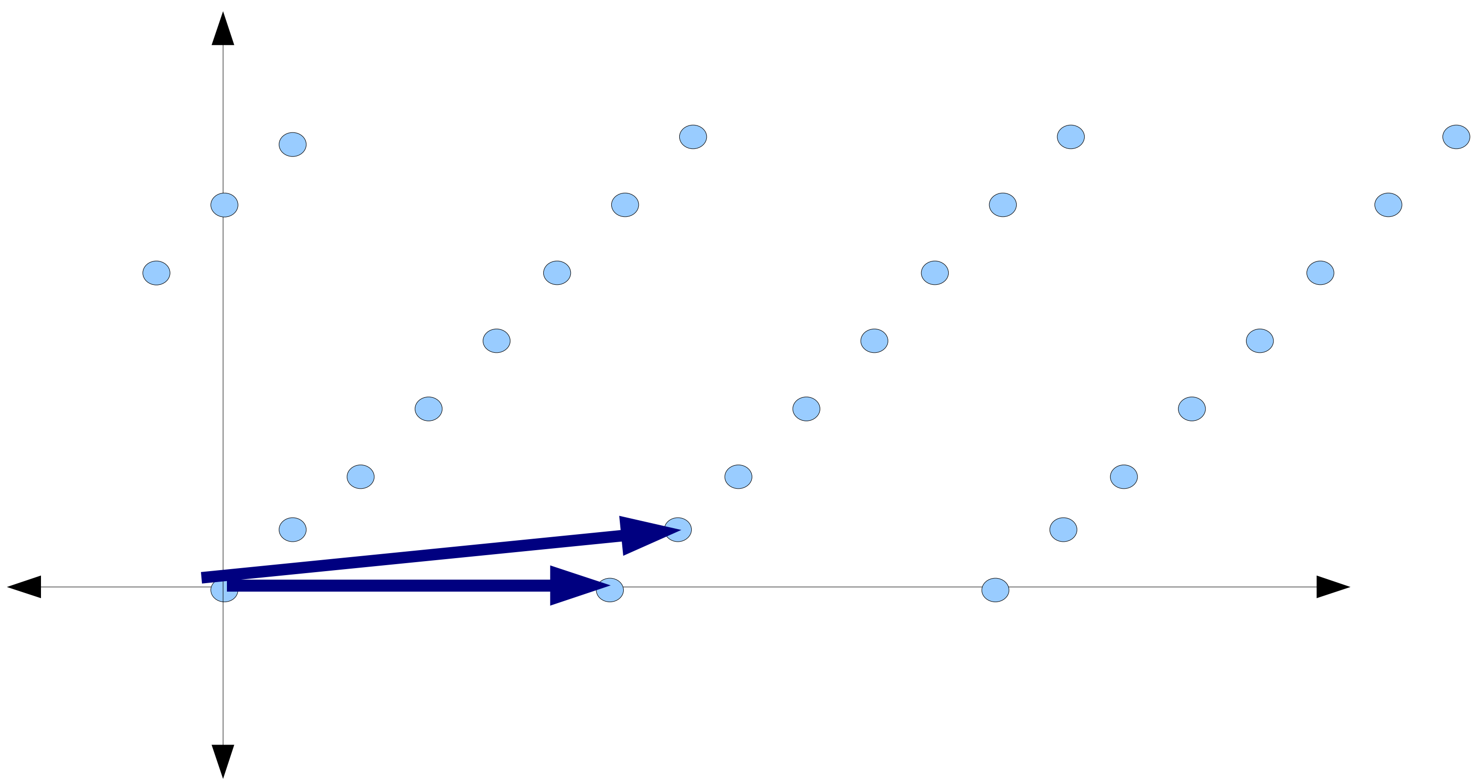
# Proof Sketch (BDD < uSVP)



# Proof Sketch (BDD < uSVP)

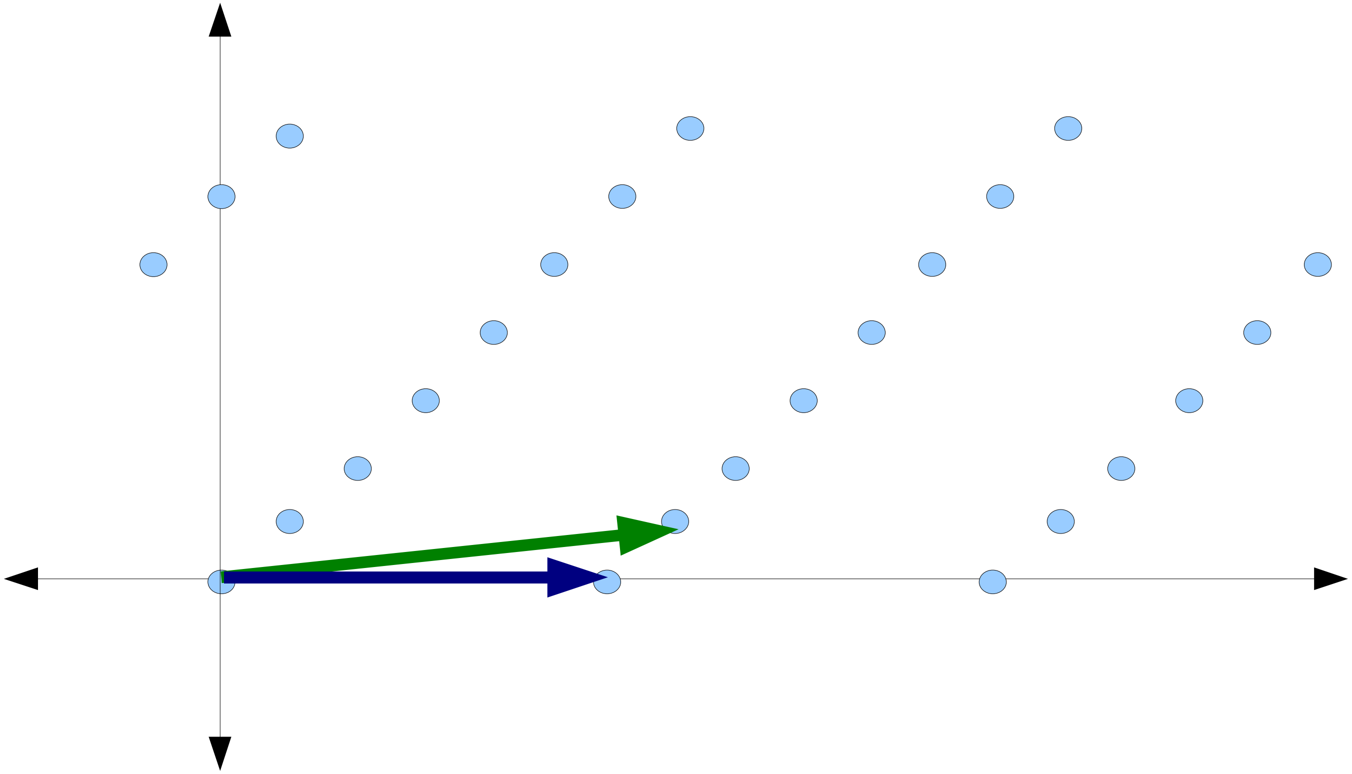


# Proof Sketch (BDD < uSVP)



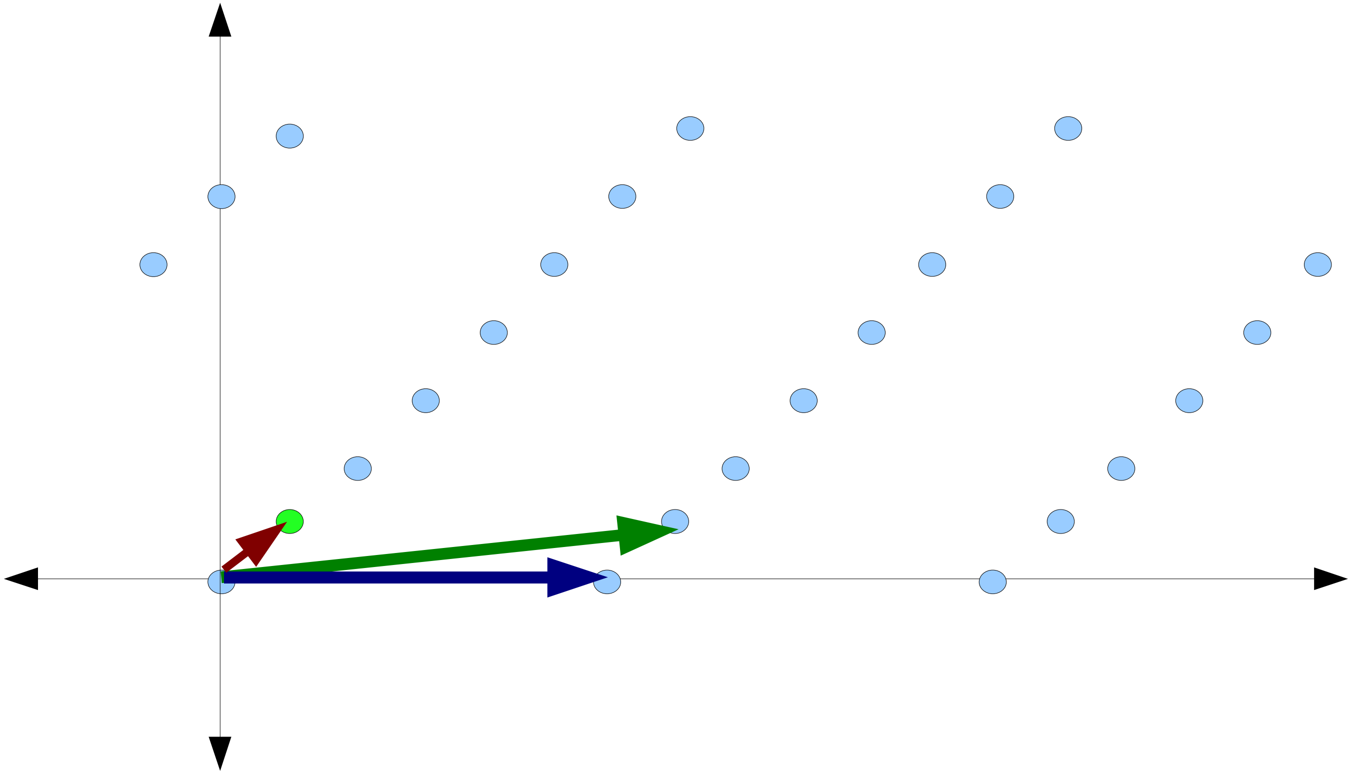
# Proof Sketch (BDD < uSVP)

New basis vector used exactly once in constructing the unique shortest vector



# Proof Sketch (BDD < uSVP)

New basis vector used exactly once in constructing the unique shortest vector



# Proof Sketch (BDD < uSVP)

New basis vector used exactly once in constructing the unique shortest vector

Subtracting unique shortest vector from new basis vector gives the closest point to the target.