Using SVM to learn loop invariant

Li Jiaying
ISTD, SUTD
2015-Sep-25

About me

- Li Jiaying
- 3rd year Ph.D. student, ISTD
- supervisor: Sun Jun
- Research interest: Program Analysis
- Current research topic: loop invariant

Outline

Outline

- Why?

Outline

- Why?
- What?

Outline

- Why?
- What?
- How?

Why?

Why?

- Software security is quite a crucial issue.

Why?

- Software security is quite a crucial issue.
- Ensure code is correct

Why?

- Software security is quite a crucial issue.
- Ensure code is correct
- Testing shows presence of defects, not their absences

Why?

- Software security is quite a crucial issue.
- Ensure code is correct
- Testing shows presence of defects, not their absences
- Verification: Prove some properties are true

Why?

- Software security is quite a crucial issue.
- Ensure code is correct
- Testing shows presence of defects, not their absences
- Verification: Prove some properties are true
  - for all nodes n: n.next.previous == n
  - x > 0

Hoare triple

- { P1 } code { P2 }
- Given P1 holds, after code execution, P2 holds.

- {x>0} x++ {x>1}
- {x<0} x++ {x<0}

Code examples

Code examples

```
assert (x >= 0);
z = x + 1;
assert(z > 0);
```

Code examples

```
assert (x >= 0);
z = x + 1;
assert(z > 0);
```

```
assert (x >= 0);
if (x != 0)
    z = x;
else
    z = x + 1;
assert(z > 0);
```

Code examples

```
assert (x >= 0);
z = x + 1;
assert(z > 0);
```

```
assert (x >= 0);
if (x != 0)
    z = x;
else
    z = x + 1;
assert(z > 0);
```

{x>= 0} /\ {x!= 0}

=> z = x > 0

\/

{x>= 0} /\ {x=0} =>

z = x + 1 > 0

Code examples

```
assert (x >= 0);
z = x + 1;
assert(z > 0);
```

```
assert (x >= 0);
if (x != 0)
    z = x;
else
    z = x + 1;
assert(z > 0);
```

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

{x>= 0} /\ {x!= 0}

=> z = x > 0

\/

{x>= 0} /\ {x=0} =>

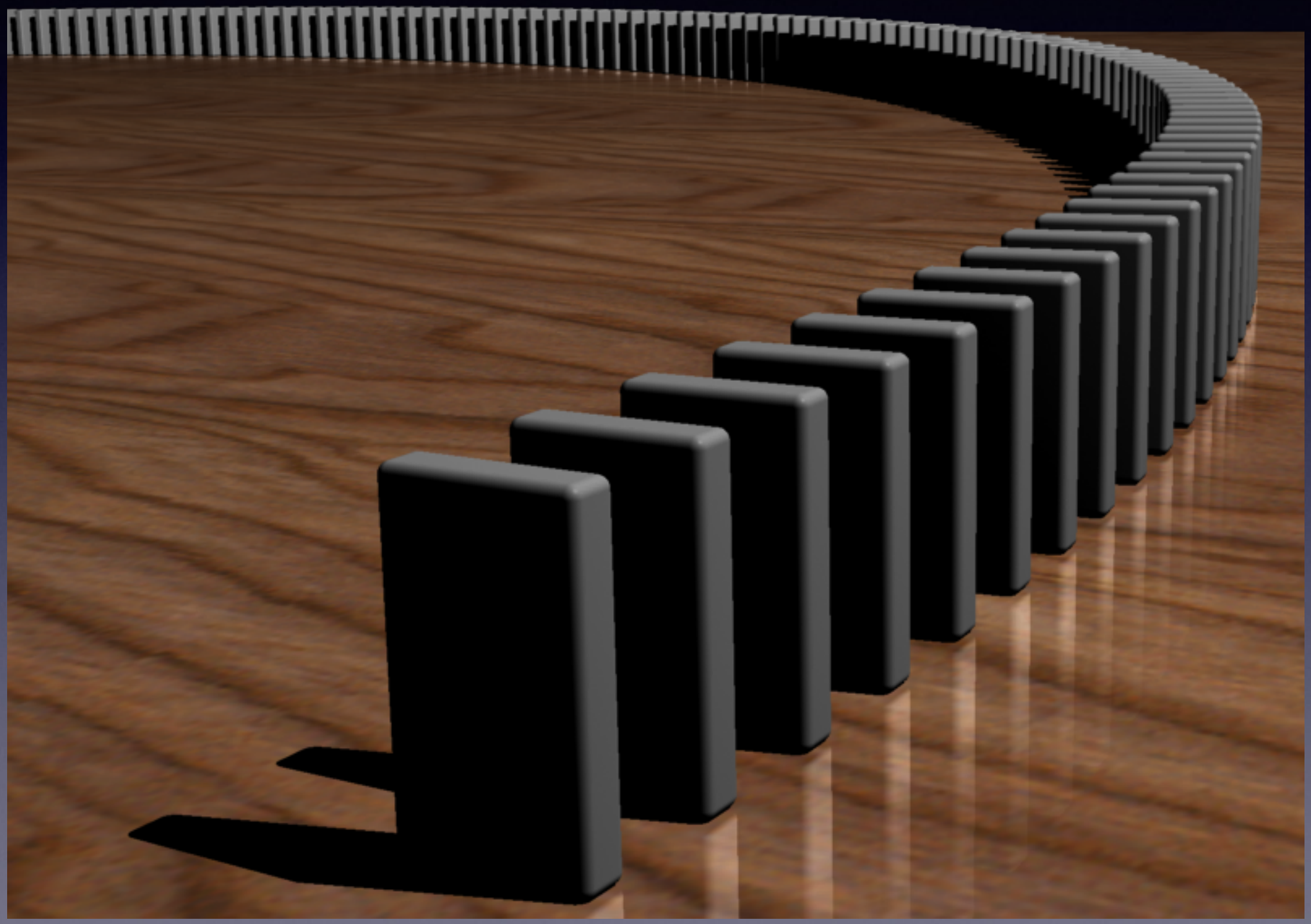
z = x + 1 > 0

Proof

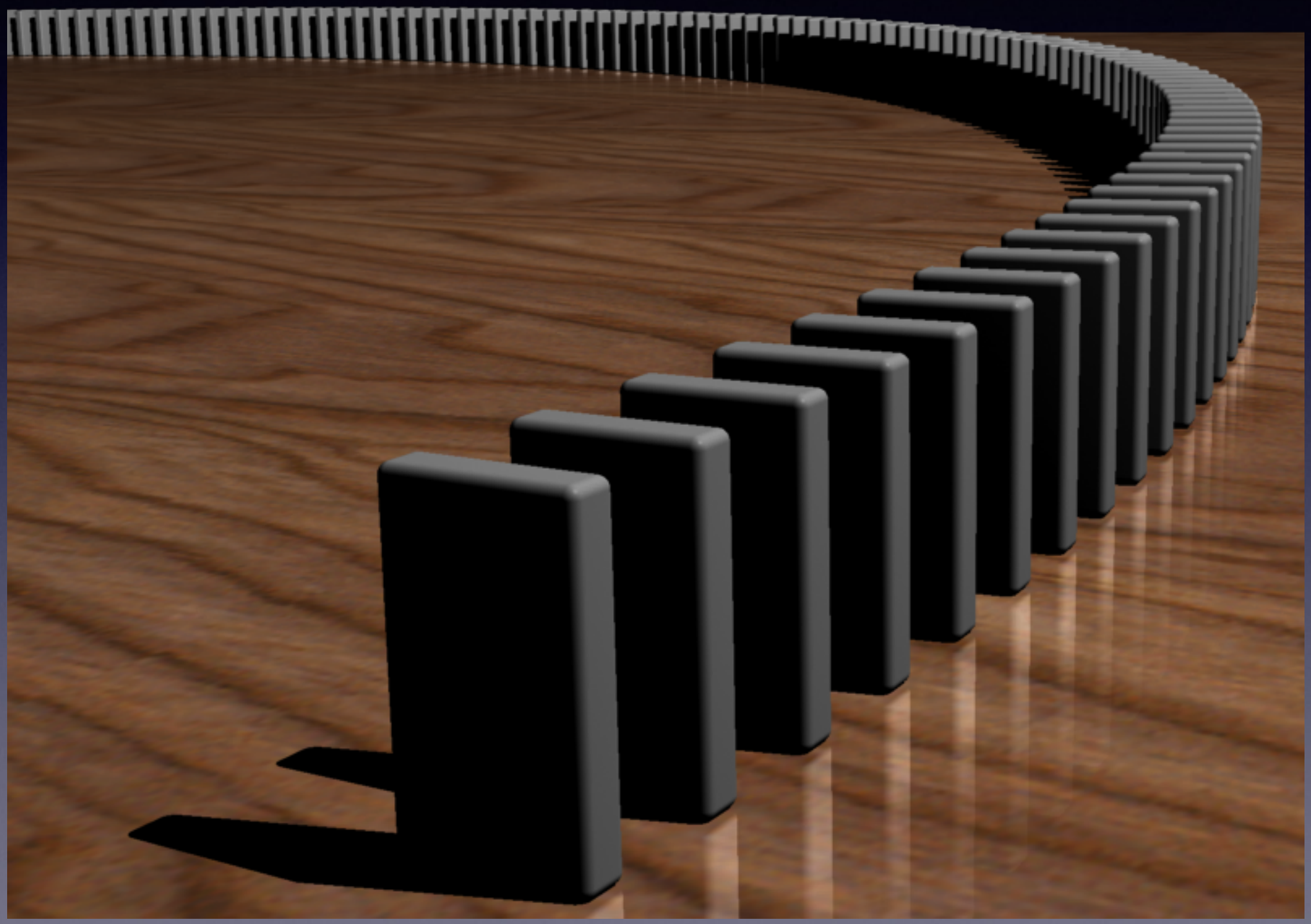
```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

z + i >= 0

Proof: induction



If I push down the first domino,
will the last one fall down eventually?



Proof: induction

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

z + i >= 0

Proof: induction

- Before loop: {z+i >= 0}

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

z + i >= 0

Proof: induction

- Before loop: {z+i >= 0}
- {z+i >= 0} loop_body {z+i >= 0}

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

z + i >= 0

Proof: induction

- Before loop: {z+i >= 0}
- {z+i >= 0} loop_body {z+i >= 0}
- => After loop: {z+i >= 0}

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

z + i >= 0

Proof: induction

- Before loop: {z+i >= 0}
- {z+i >= 0} loop_body {z+i >= 0}
- => After loop: {z+i >= 0}

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

{z+i >= 0} ∧
{i < 0}
=> z > 0

z + i >= 0

Loop invariant

z + i >= 0

Loop invariant

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

Loop invariant

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv
```

Loop invariant

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv
```

Question: True?

Strong loop invariant

```
assert (P1);     // Precondition: P1     P1=>Inv
while (B) {      // loop condition: B    Inv /\ B --s--> Inv
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2    Inv /\ !B => P2
```

Strong loop invariant

```
assert (P1);     // Precondition: P1    P1=>Inv
while (B) {      // loop condition: B   Inv /\ B --s--> Inv
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2    Inv /\ !B => P2
```

```
assert (x >= 0);
i = x, z = 0;
while (i >= 0) {
    z = z + 1;
    i = i - 1;
}
assert (z > 0);
```

- Question: {i+z >= 0}?

Finding strong loop invariant

Finding strong loop invariant

- An old topic

Finding strong loop invariant

- An old topic
- Static approach
  - Infer loop invariant based on source code without executing program
- Dynamic approach
  - Infer loop invariant based on testing results

Our approach

- Using SVM to learn loop invariant

Another example…

```
assert (x + y > 0);
while (x >= 0) {
    x--;
    y++;
}
assert (y > 0);
```

Instrumentation

- Another example…

```
assert (x + y > 0);
while (x >= 0) {
    x--;
    y++;
}
assert (y > 0);
```

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

All possible outputs

P1 \ P2	Pass	Fail
Pass		
Fail		

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Sample-invariant table

P1 \ P2	Pass	Fail
Pass	Inv???	Inv???
Fail	Inv???	Inv???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

P1=>Inv

Inv /\ B --s--> Inv

Inv /\ !B => P2

P1 \ P2	Pass	Fail
Pass	???	???
Fail	???	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

P1=>Inv

Inv /\ B --s--> Inv

Inv /\ !B => P2

P1 \ P2	Pass	Fail
Pass	+	+
Fail	???	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1    P1=>Inv
while (B) {      // loop condition: B   Inv /\ B --s--> Inv
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2   Inv /\ !B => P2
```

P1 \ P2	Pass	Fail
Pass	+	+
Fail	???	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1    P1=>Inv
while (B) {      // loop condition: B   Inv /\ B --s--> Inv
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2    Inv /\ !B => P2
```

P1 \ P2	Pass	Fail
Pass	+	counter example
Fail	???	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv




 Inv /\ !B => P2
```

P1 \ P2	Pass	Fail
Pass	+	counter example
Fail	???	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv




 Inv /\ !B => P2
```

P1 \ P2	Pass	Fail
Pass	+	counter example
Fail	unknown	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

```
assert (P1);     // Precondition: P1
while (B) {      // loop condition: B
    S1;          // loop body: Si
    S2;
    S3;
}
assert (P2);     // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv

 Inv /\ !B => P2
```

P1 \ P2	Pass	Fail
Pass	+	counter example
Fail	unknown	???

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Sample-invariant table

P1 \ P2	Pass	Fail
Pass	+	counter example
Fail	unknown	-

```
print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

1\2	P	F
P	+	x
F	?	-

Collecting data

- random generate test input
  - (1,0) (2,0) (-1,0) (-2,0)

```
x = rand();
y = rand();

print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Labelling data

1\2	P	F
P	+	x
F	?	-

- random generate test input
  - (1,0) (2,0) (-1,0) (-2,0)
- label these test cases according to sample-invariant table
  - + (1,0) (2,0)
  - - (-1,0) (-2,0)

```
x = rand();
y = rand();

print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Finding a classifier

1\2	P	F
P	+	x
F	?	-

1\2	P	F
P	+	x
F	?	-

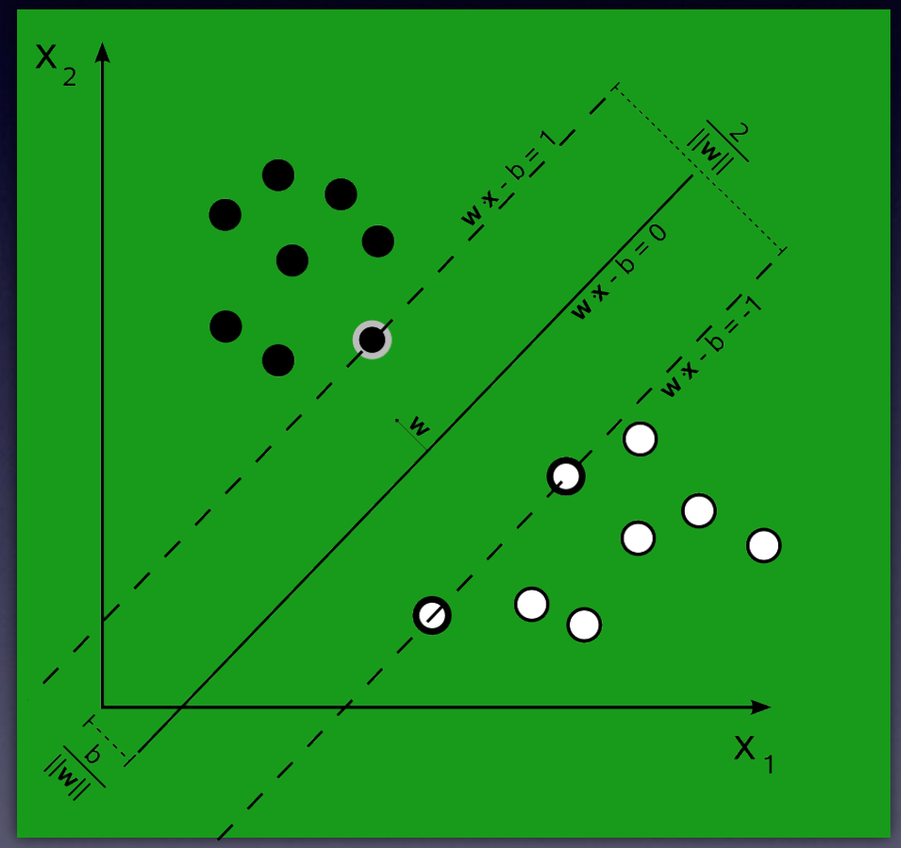
Finding a classifier

- A typical Machine Learning problem

1\2	P	F
P	+	x
F	?	-

Finding a classifier

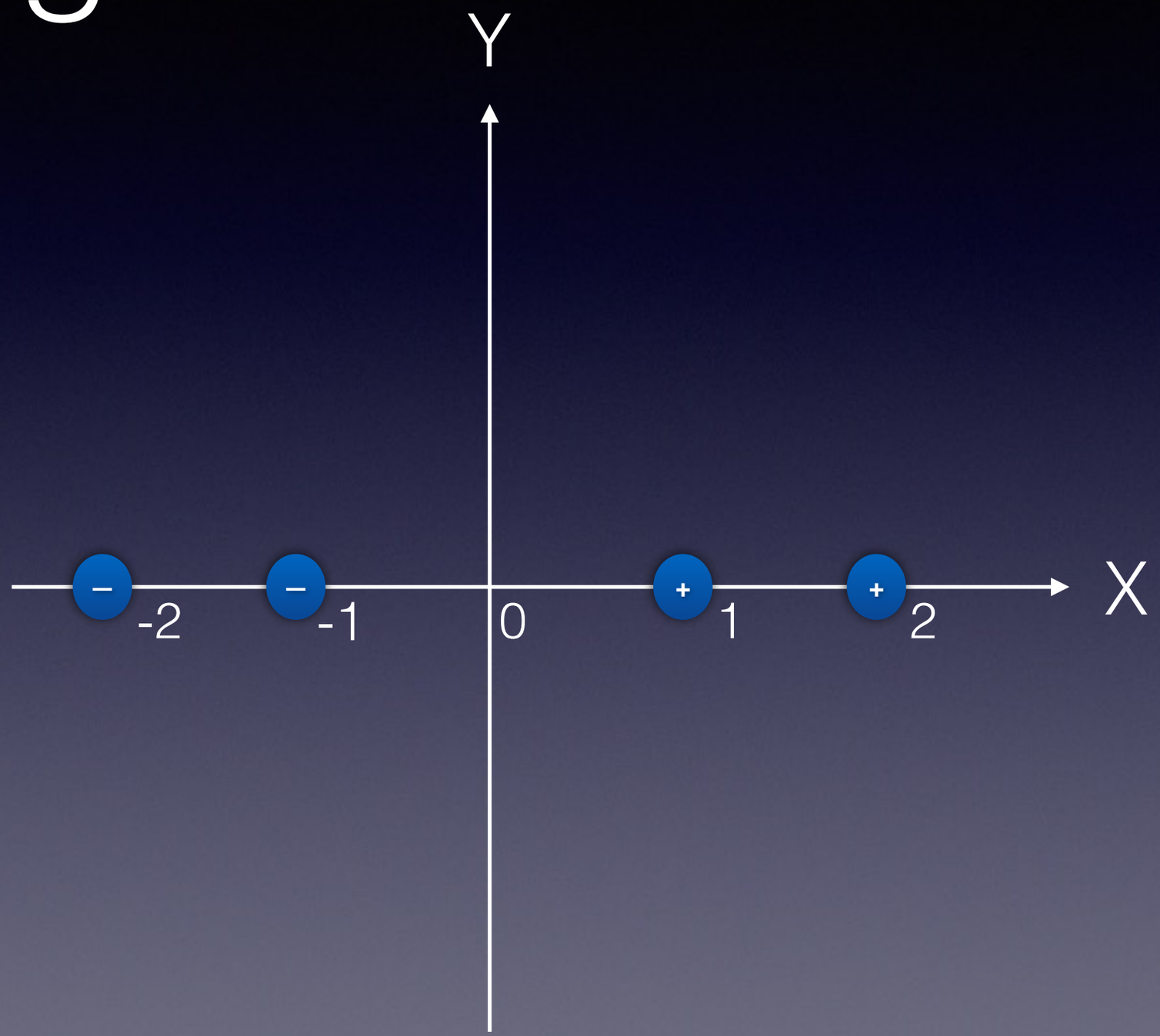
- A typical Machine Learning problem
- SVM (Support Vector Machine)
  - a separator which has maximum distance with nearest samples



1\2	P	F
P	+	x
F	?	-

Getting candidate

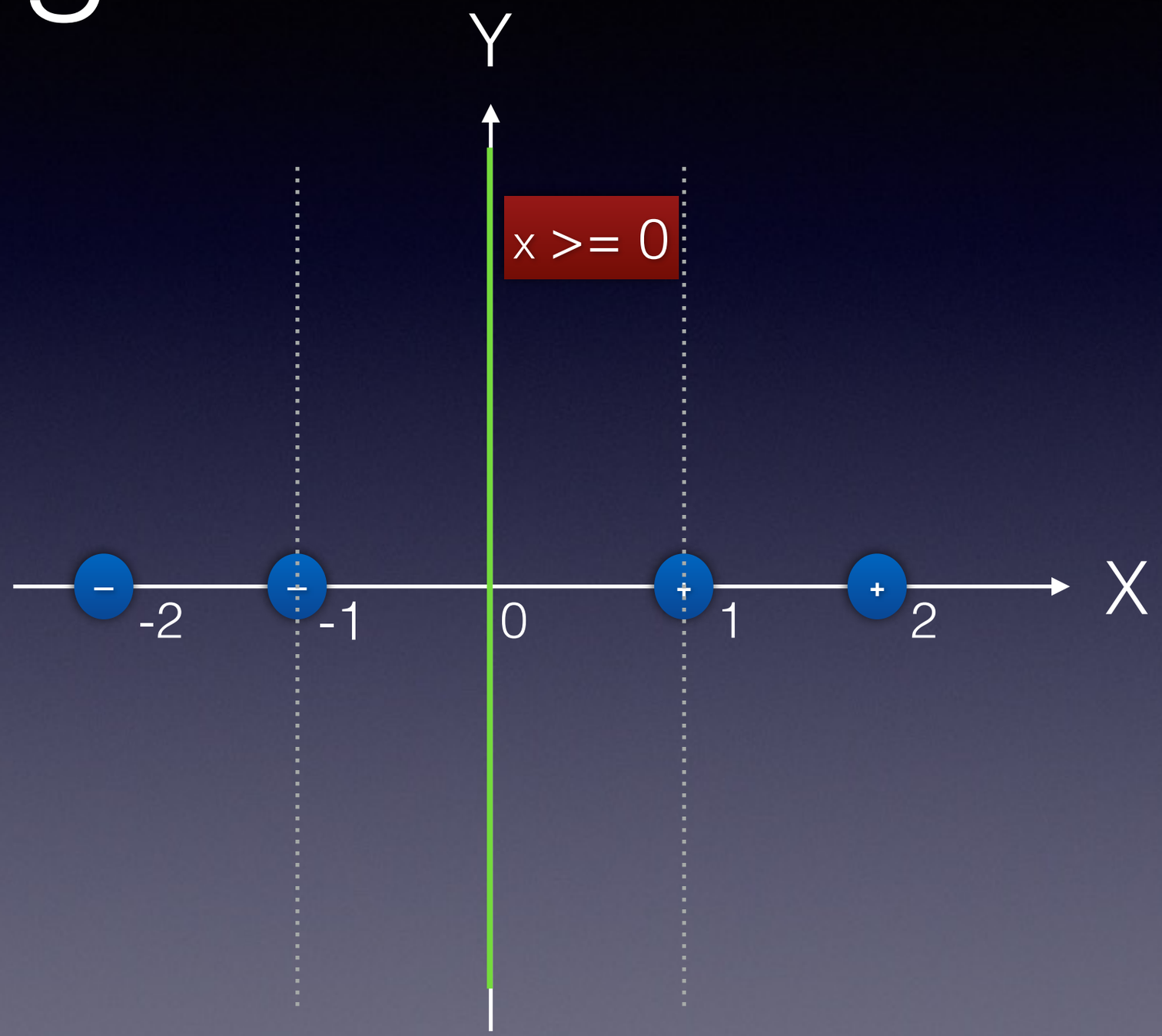
- + (1,0) (2,0)
- - (-1,0) (-2,0)



Getting candidate

1\2	P	F
P	+	x
F	?	-

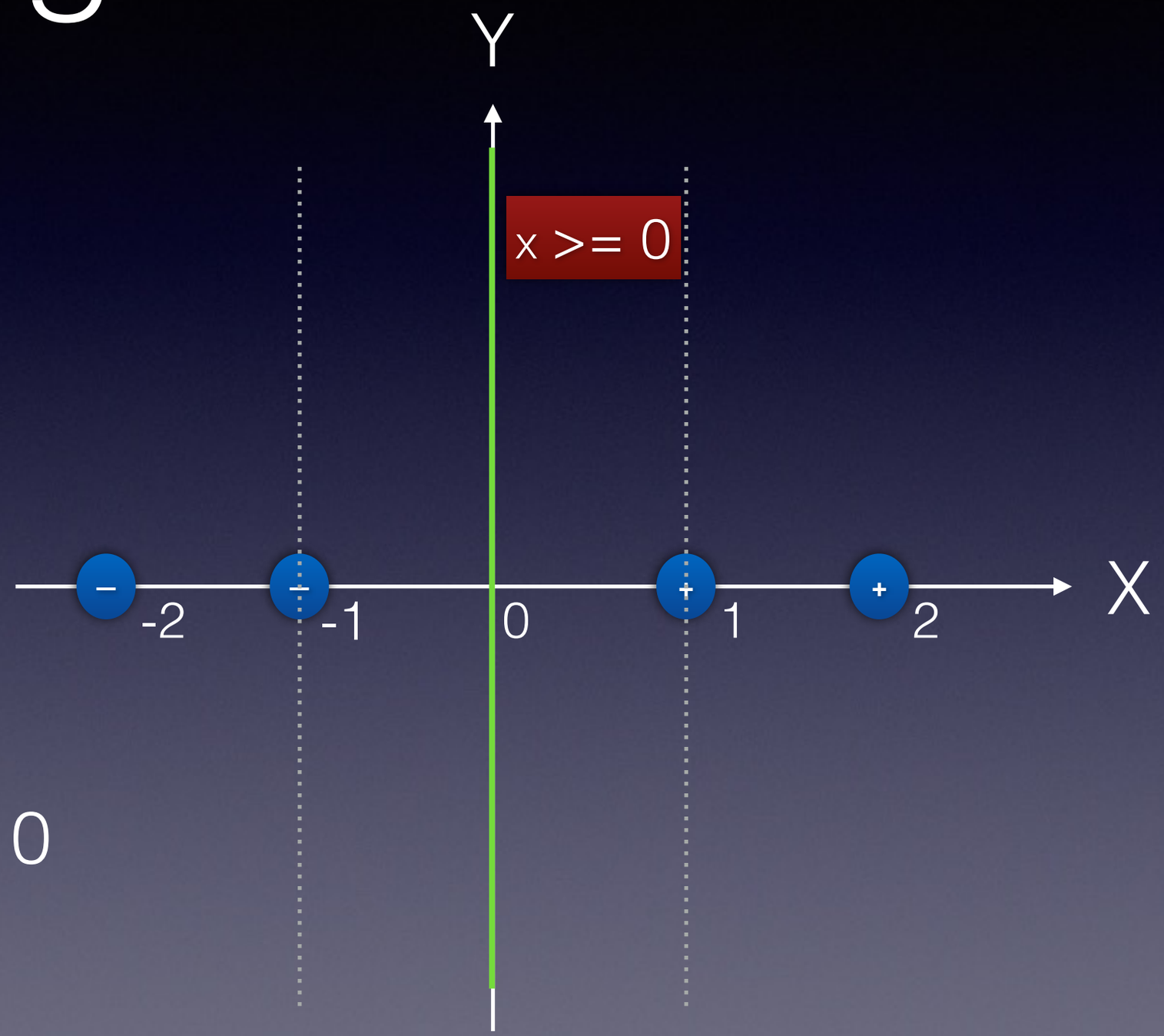
- + (1,0) (2,0)
- - (-1,0) (-2,0)



1\2	P	F
P	+	x
F	?	-

Getting candidate

- + (1,0) (2,0)
- - (-1,0) (-2,0)



- candidate: x >= 0

1\2	P	F
P	+	x
F	?	-

Refining candidate

- candidate: x >= 0

```
x = rand();
y = rand();

print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Refining candidate

1\2	P	F
P	+	x
F	?	-

- candidate: x >= 0
- from candidate to test cases
  - (0,-1) (0,0) (0,1)

```
x = rand();
y = rand();

print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

Refining candidate

1\2	P	F
P	+	x
F	?	-

- candidate: x >= 0
- from candidate to test cases
- label these test cases
  - - (0,-1)
  - ? (0,0)
  - + (0,1)

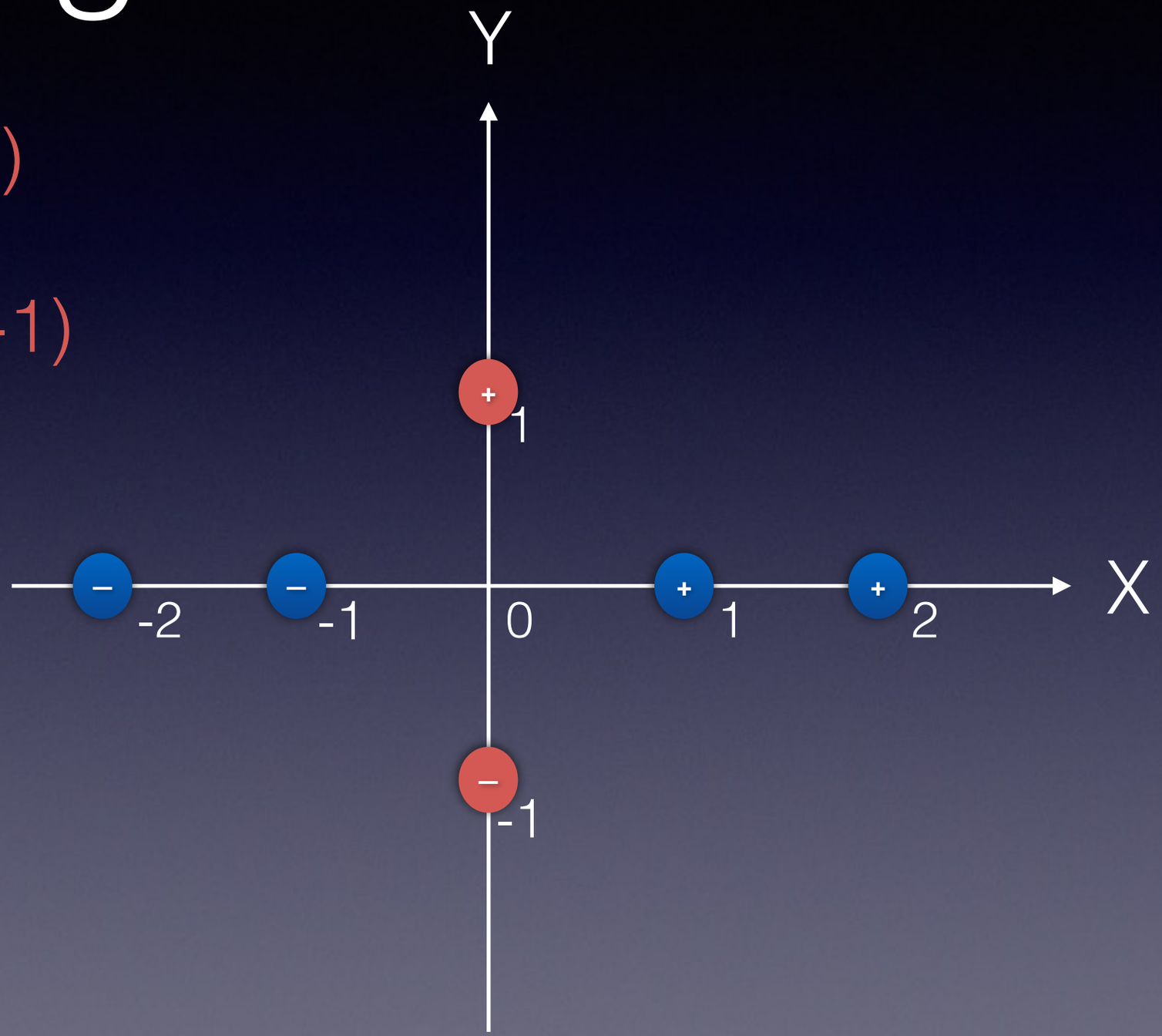
```
x = rand();
y = rand();

print x, y
if (x + y > 0)
    print 'pass P1'
while (x >= 0) {
    x--;
    y++;
}
if  (y > 0)
    print 'pass P2'
```

1\2	P	F
P	+	x
F	?	-

Refining candidate

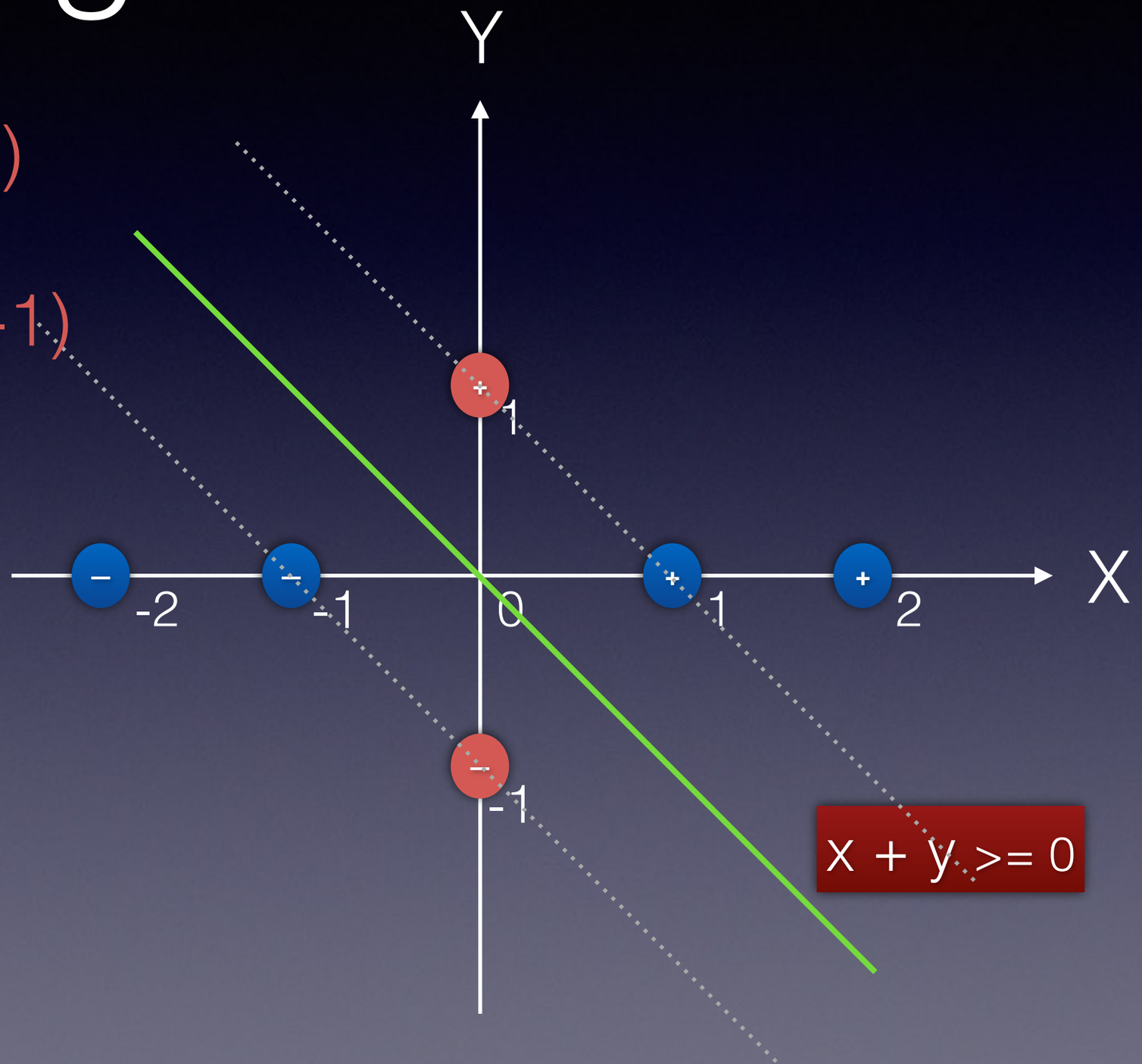
- \+ (1,0) (2,0) (0,1)
- \- (-1,0) (-2,0) (0,-1)
- ~~? (0,0)~~



1\2	P	F
P	+	x
F	?	-

Refining candidate

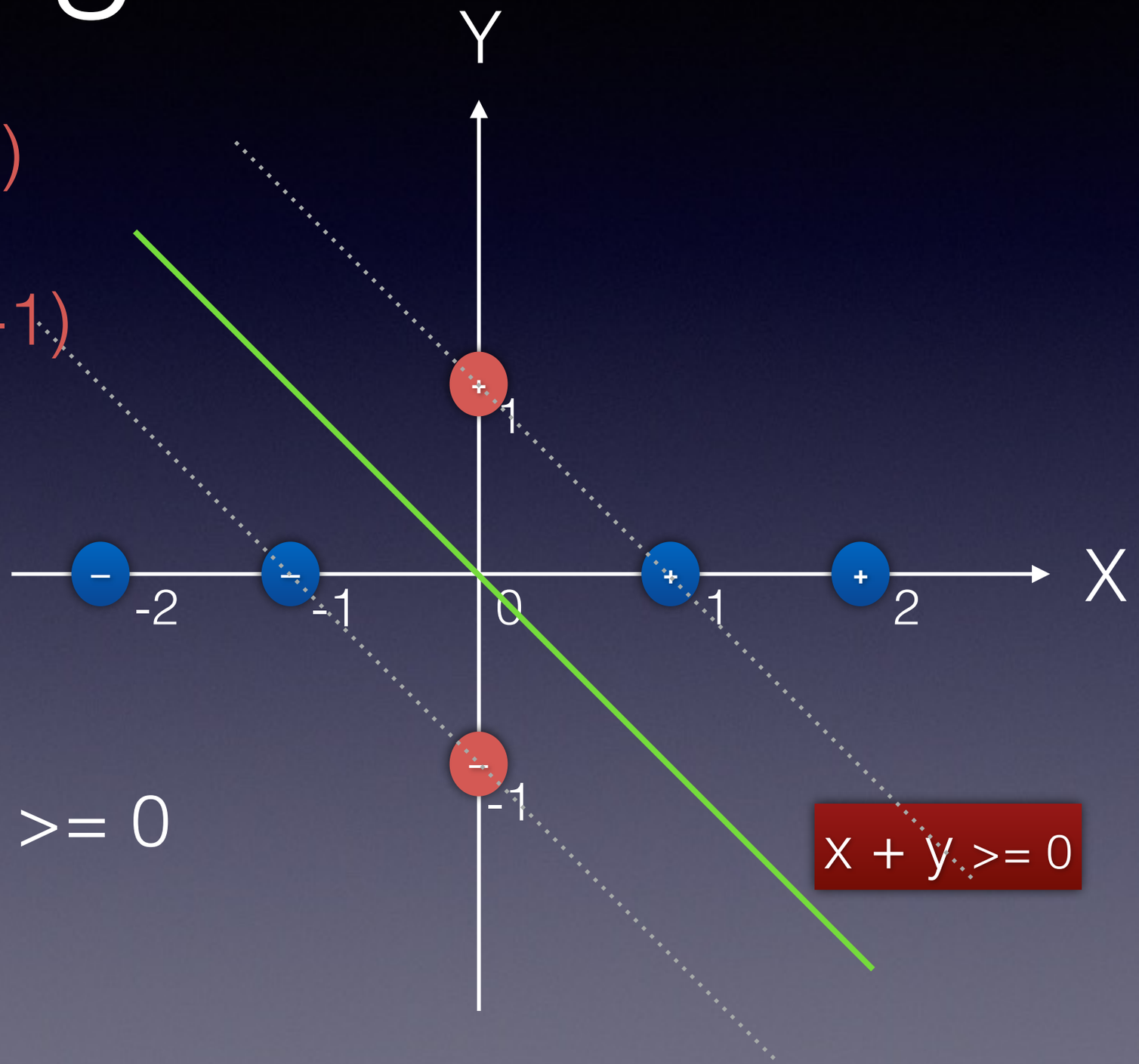
- + (1,0) (2,0) (0,1)
- - (-1,0) (-2,0) (0,-1)
- ~~? (0,0)~~



1\2	P	F
P	+	x
F	?	-

Refining candidate

- + (1,0) (2,0) (0,1)
- - (-1,0) (-2,0) (0,-1)
- ~~? (0,0)~~
- candidate: x + y >= 0



1\2	P	F
P	+	x
F	?	-

Refining candidate

- Repeat refining process until convergence
- .....
- loop invariant candidate: x + y >= 0

Checking candidate

```
assert (x + y > 0);          // Precondition: P1    P1=>Inv
while (x >= 0) {             // loop condition: B   Inv /\ B --s--> Inv
    x--;                     // loop body: Si
    y++;
}
assert (y > 0);              // Postcondition: P2    Inv /\ !B => P2
```

Checking candidate

```
assert (x + y > 0);          // Precondition: P1     P1=>Inv
while (x >= 0) {             // loop condition: B    Inv /\ B --s--> Inv
    x--;                     // loop body: Si
    y++;
}
assert (y > 0);              // Postcondition: P2    Inv /\ !B => P2
```

- {x+y>0} => {x+y>=0}

Checking candidate

```
assert (x + y > 0);          // Precondition: P1     P1=>Inv
while (x >= 0) {             // loop condition: B    Inv /\ B --s--> Inv
    x--;                     // loop body: Si
    y++;
}
assert (y > 0);              // Postcondition: P2    Inv /\ !B => P2
```

- {x+y>0} => {x+y>=0}
- {x+y>=0} /\ {x>=0} =>{newX+newY>=0}

Checking invariant strength

```
assert (x + y > 0);          // Precondition: P1     P1=>Inv
while (x >= 0) {             // loop condition: B    Inv /\ B --s--> Inv
    x--;                     // loop body: Si
    y++;
}
assert (y > 0);              // Postcondition: P2    Inv /\ !B => P2
```

- {x+y>0} => {x+y>=0}
- {x+y>=0} ∧ {x>=0} =>{newX+newY>=0}
- {x+y>=0} ∧ {x<0} => {y>0}

What we have done

What we have done

- We have found loop invariant candidate

What we have done

- We have found loop invariant candidate
- and we have proved it is really loop invariant

What we have done

- We have found loop invariant candidate
- and we have proved it is really loop invariant
- and also strong enough to prove postcondition

So.....

- We finish proving.

Thank you

- Welcome to visit and join our group.

Thank you

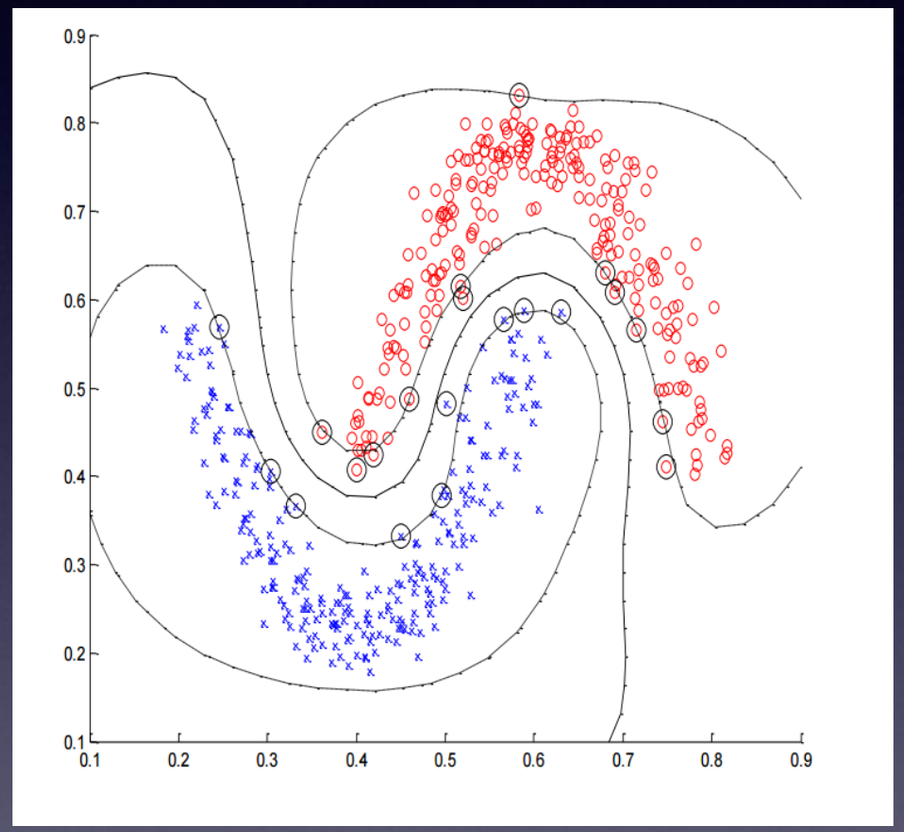
- Welcome to visit and join our group.
- Questions?

Thank you

1\2	P	F
P	+	x
F	?	-

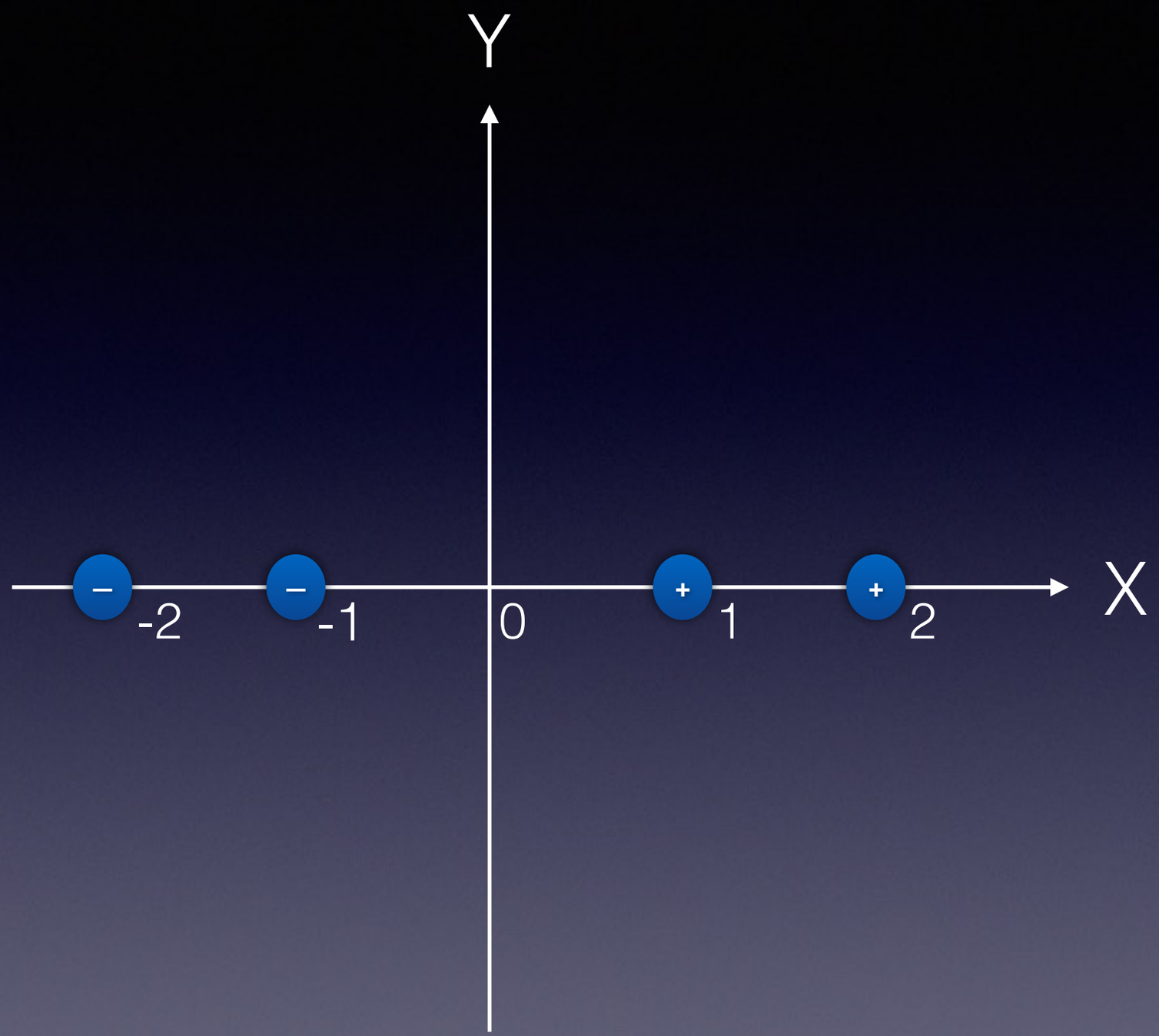
Finding a classifier

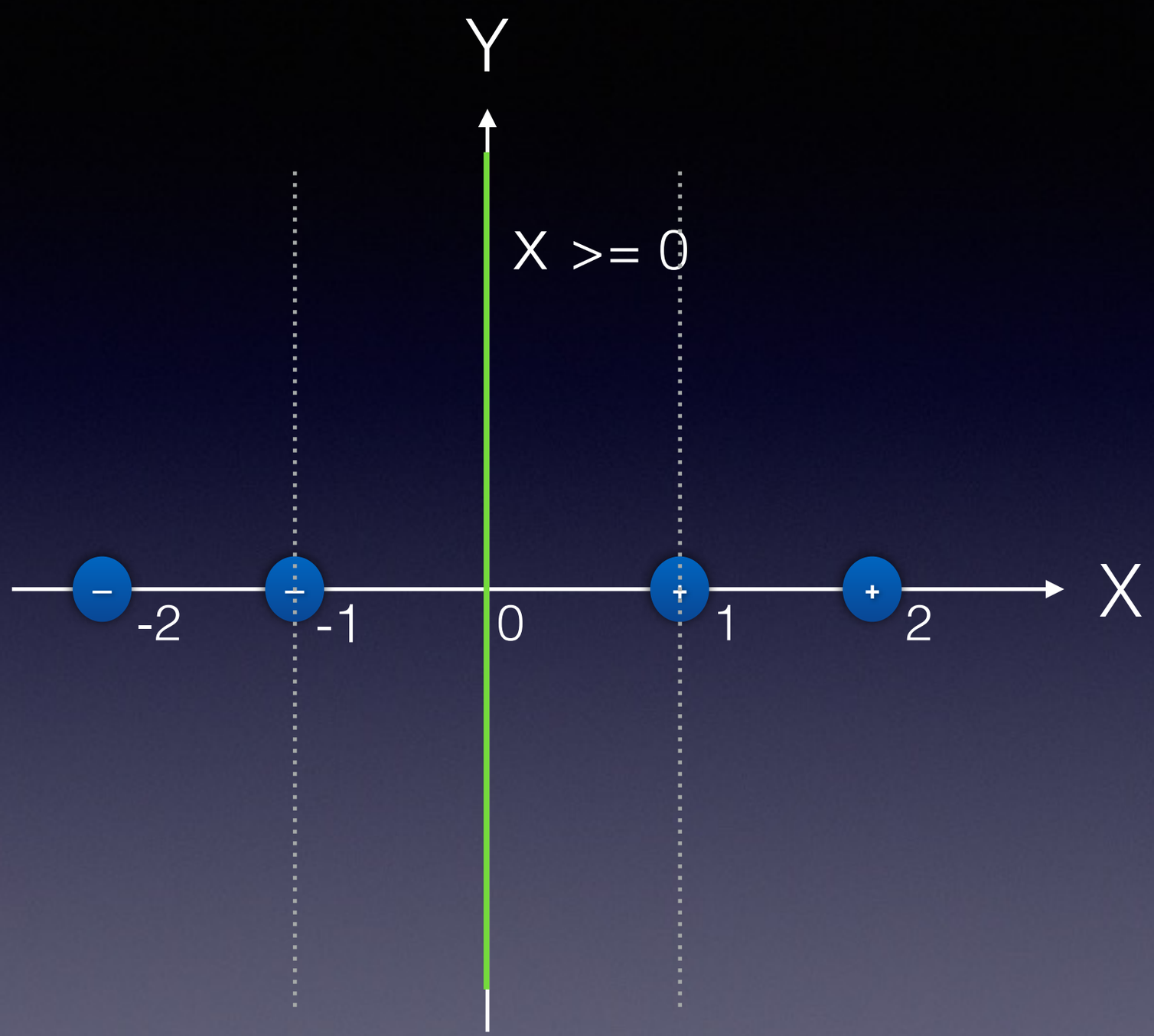
- A typical Machine Learning problem
- SVM (Support Vector Machine)
  - a separator which has maximum distance with nearest samples
  - Kernel method to handle non-linear case

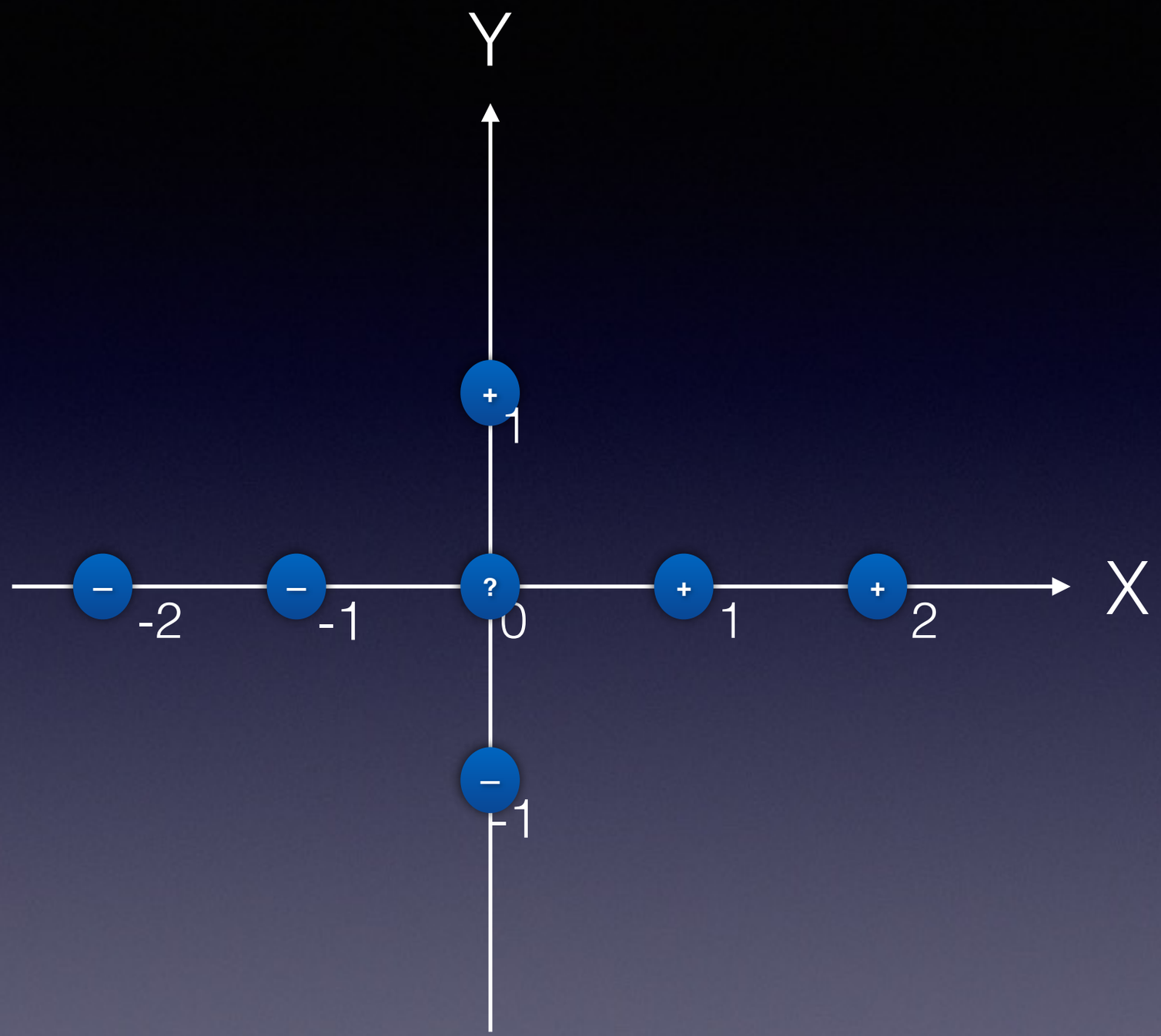


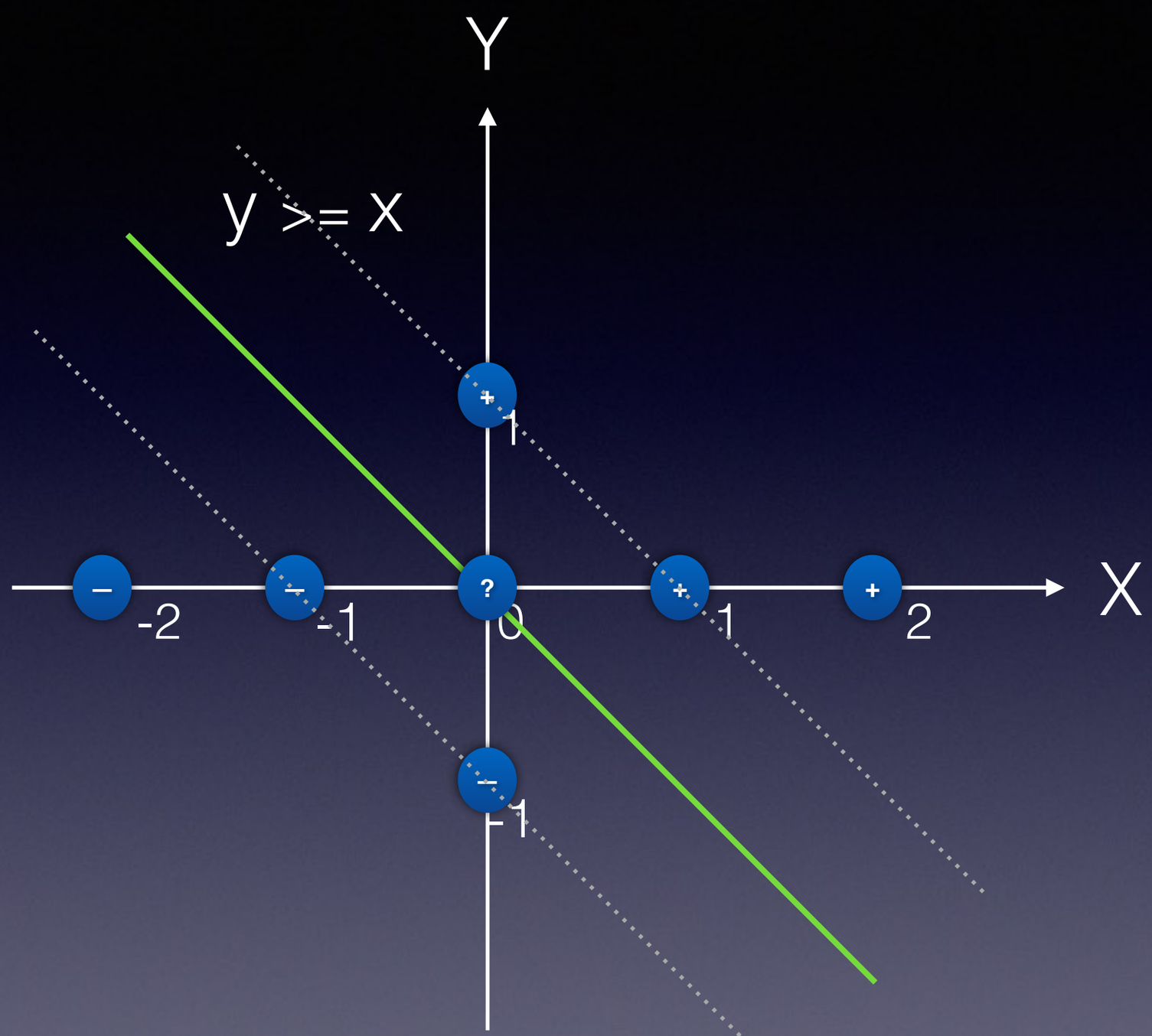
Using SVM to learn loop invariant

- 1. Instrument program.
- 2. Run program to gather data.
- 3. Use SVM to classify, get invariant candidate.
- 4. Check convergence.
- 5. Check loop invariant validation.
- 6. Check the strength of loop invariant.


Y
X
–
–
+
+
-2
-1
0
1
2


Y
X >= 0
X
–
+
+
+
-2
-1
0
1
2


Y
+
1
–
–
?
+
+
-2
-1
0
1
2
X
–
-1

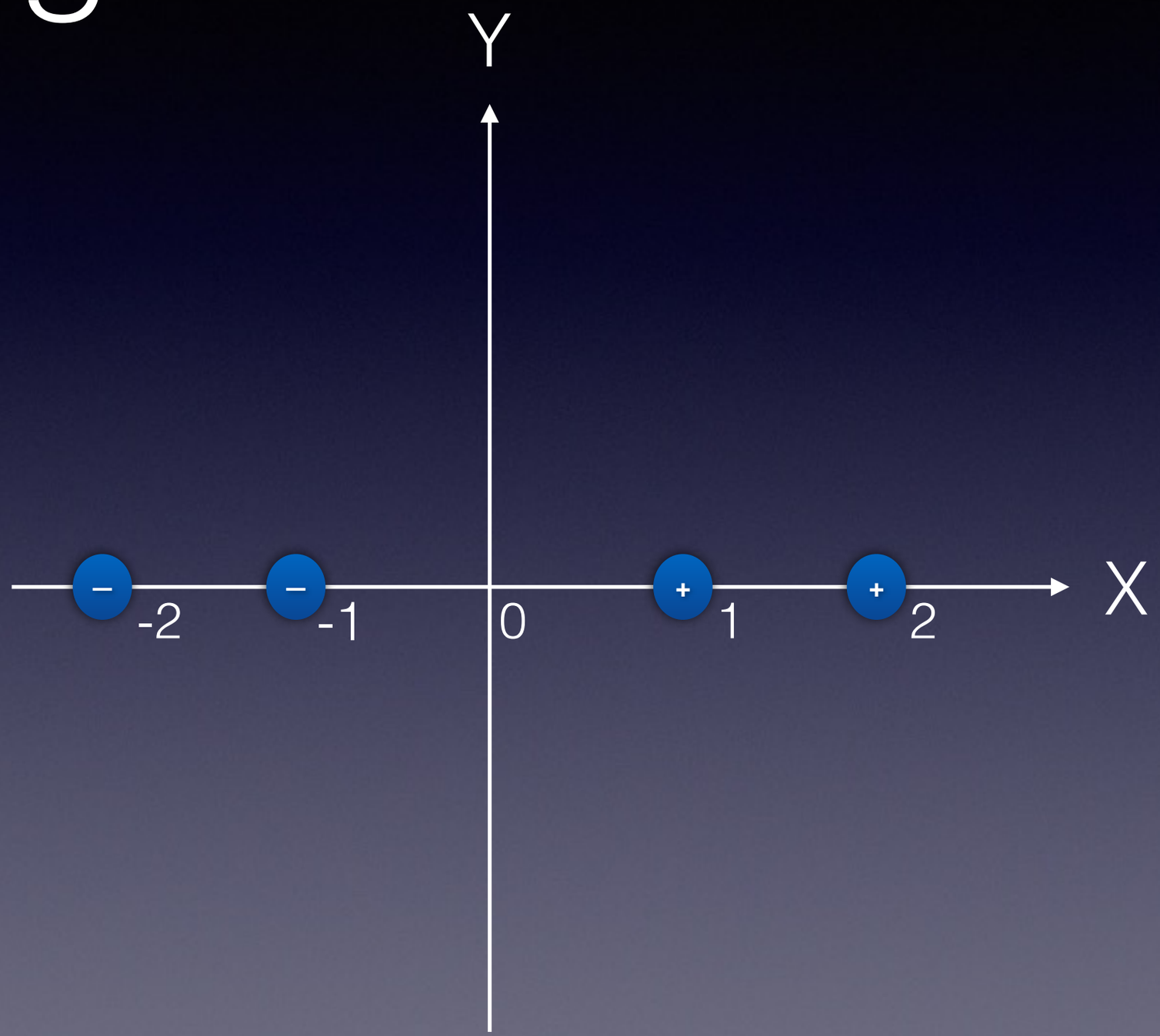

Y
y >= x
+
1
–
-2
–
-1
?
0
+
1
+
2
X
–
-1

So……….

1\2	P	F
P	+	x
F	?	-

Getting candidate

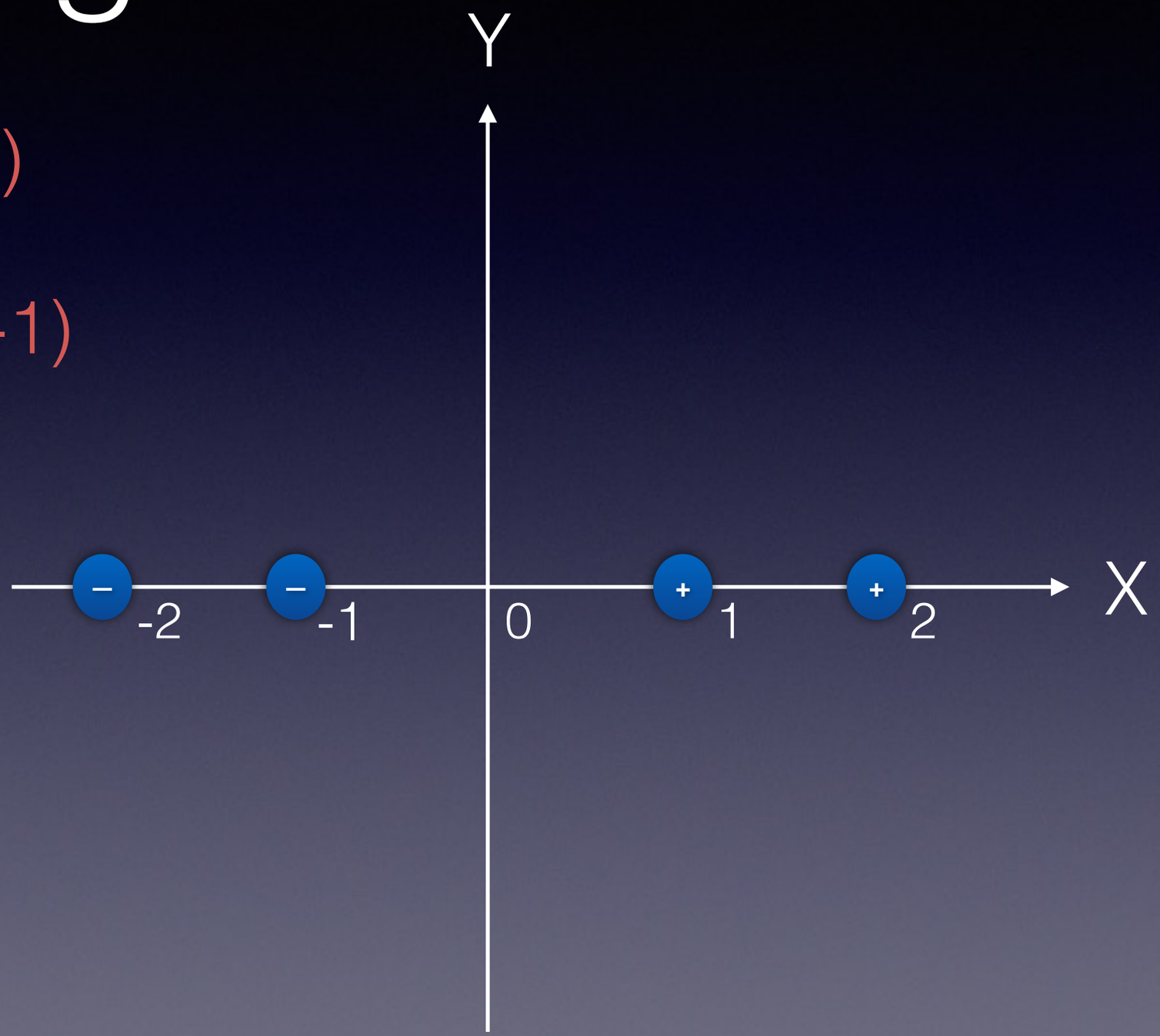
- + (1,0) (2,0)
- - (-1,0) (-2,0)



1\2	P	F
P	+	x
F	?	-

Refining candidate

- + (1,0) (2,0) (0,1)
- - (-1,0) (-2,0) (0,-1)
- ~~? (0,0)~~



Checking candidate

```
assert (x + y > 0);
while (x >= 0) {
    x--;
    y++;
}
assert (y > 0);
```

Checking candidate

```
assert (x + y > 0);          // Precondition: P1
while (x >= 0) {             // loop condition: B
    x--;                     // loop body: Si
    y++;
}
assert (y > 0);              // Postcondition: P2
```

```
P1=>Inv
Inv /\ B --s--> Inv
```