LINE DRAWING ALGORITHMS

- Digital Differential Analyzer (DDA)
- Bresenham's line algorithm

Author: prof. Yevhenii Borodavka

Digital Differential Analyzer

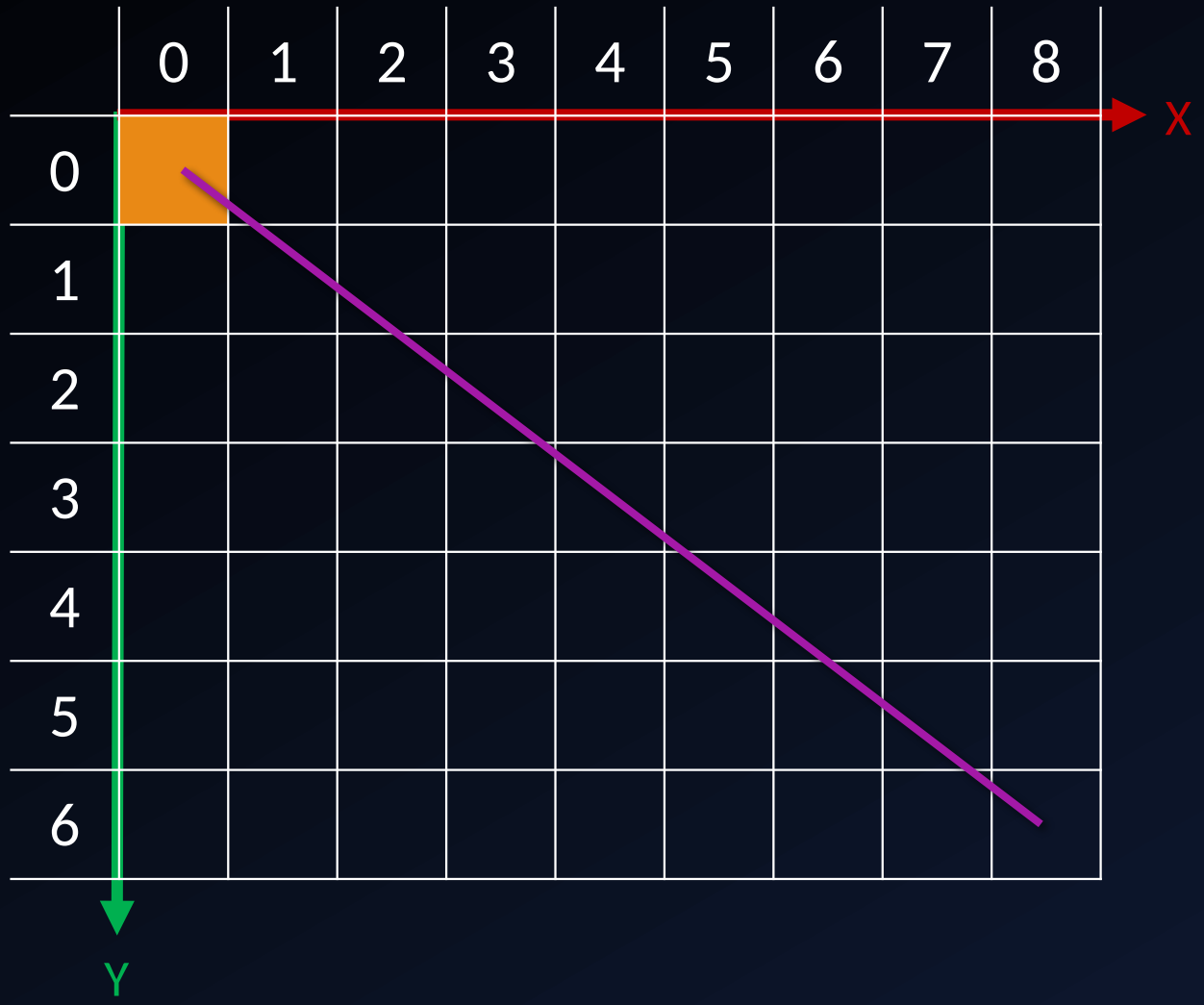
DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics to generate a line segment between two specified endpoints. It is a simple and efficient algorithm that plots the line by using the incremental difference between the x-coordinates and y-coordinates of the two endpoints.

The steps involved in the DDA line generation algorithm are:

1. Input the two endpoints of the line segment, (x_1,y_1) and (x_2,y_2).
2. Calculate the difference between the x-coordinates and y-coordinates of the endpoints as **dx** and **dy** respectively.
3. Calculate the maximum between **dx** and **dy** and set it as a **step**.
4. Calculate the increments: $\mathbf{X_i=dx/step}$ and $\mathbf{Y_i=dy/step}$.
5. Set the initial point of the line as (x_1,y_1).
6. Loop through the **step**, incrementing x by $\mathbf{X_i}$ and y by $\mathbf{Y_i}$.
7. Plot the pixel at the calculated (x,y) coordinate.
8. Repeat steps 6 and 7 until the endpoint (x_2,y_2) is reached.

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



$dx=(8-0)=8$

$dy=(6-0)=6$

$step=max(dx,dy)=max(8,6)=8$

$X_i=dx/step=8/8=1$

$Y_i=dy/step=6/8=3/4$

LOOP #0

$X=x_1=0$

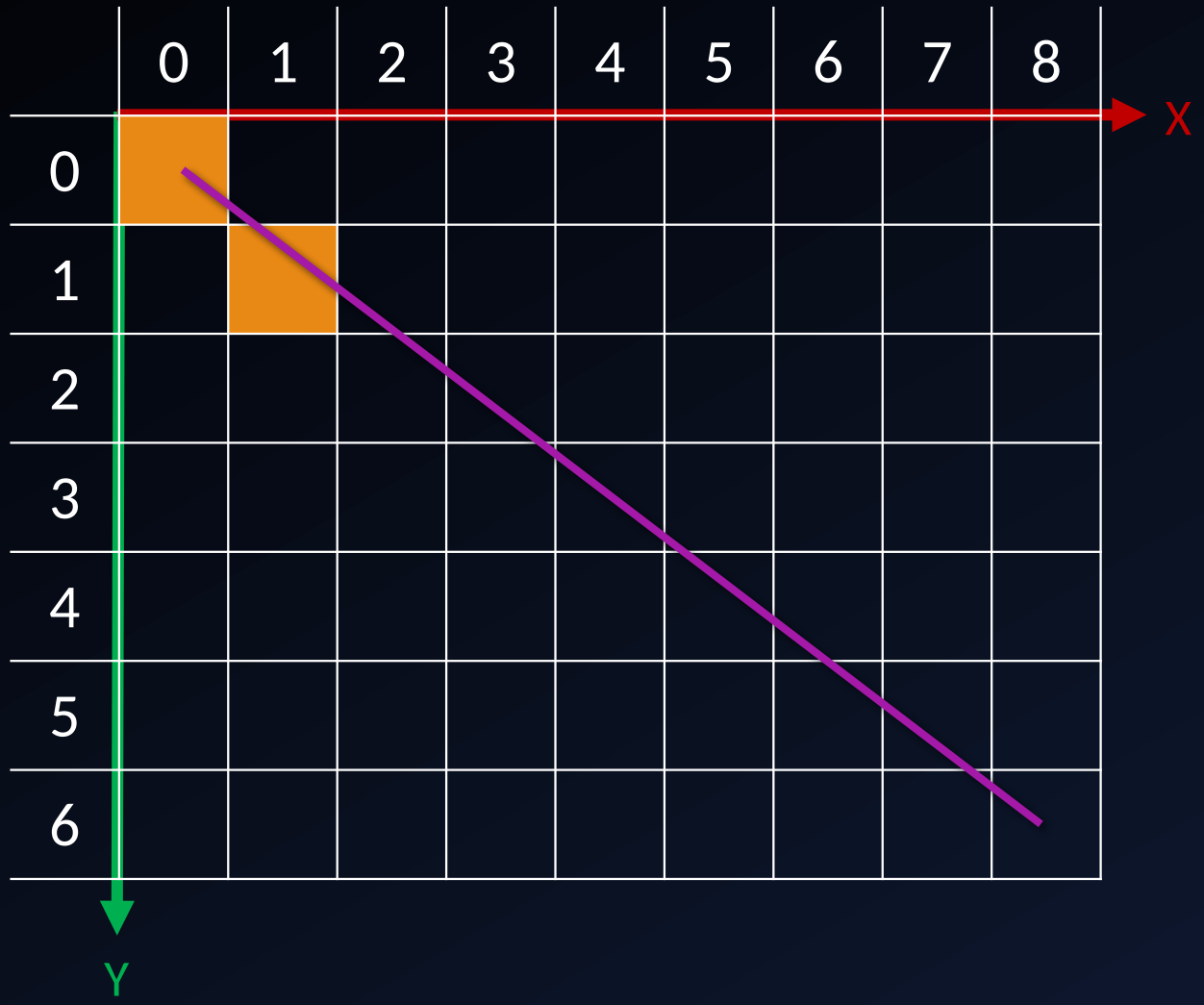
$Y=y_1=0$

$x=round(X)=round(0)=0$

$y=round(Y)=round(0)=0$

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



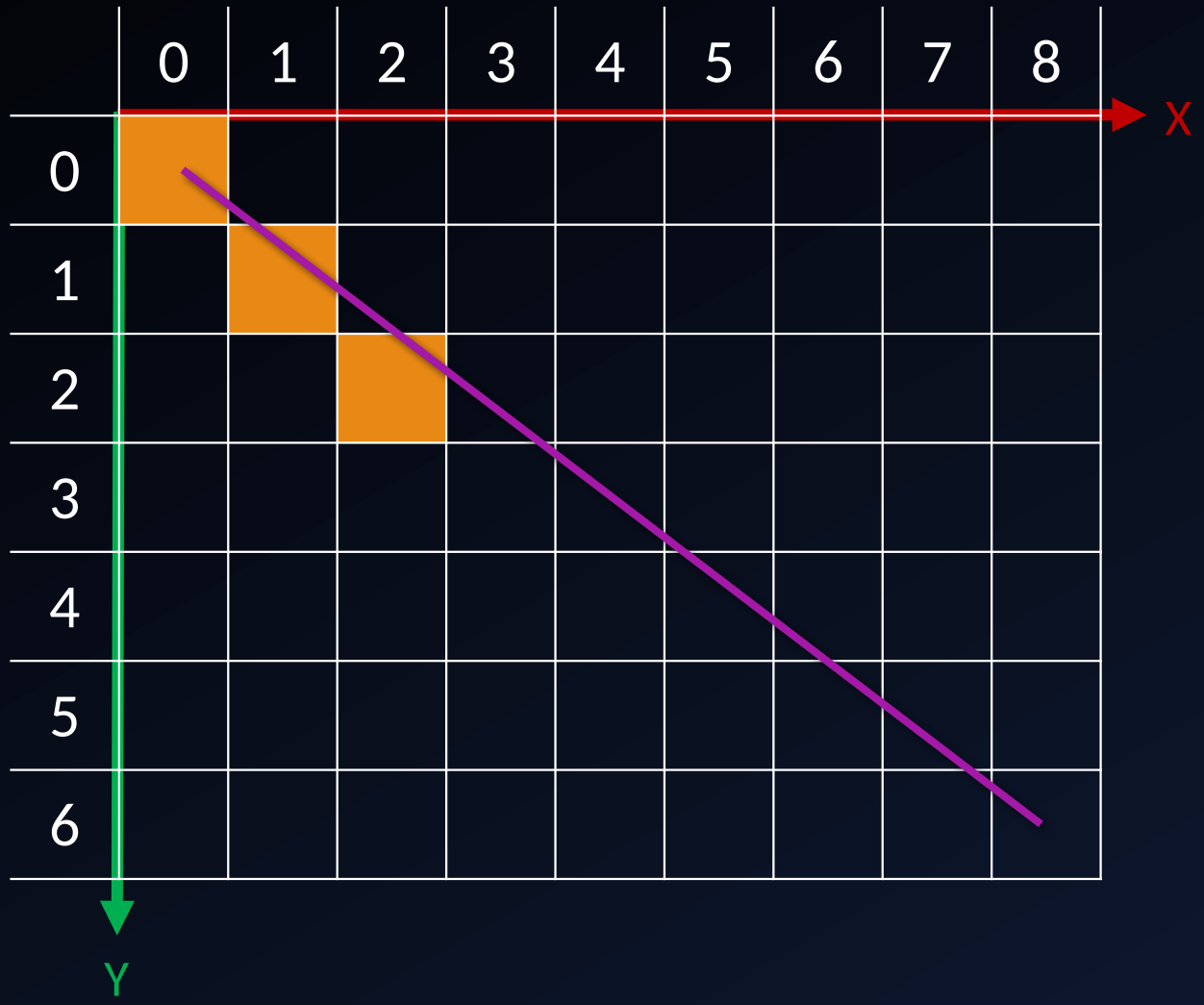
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #1

X=X+X_i=(0+1)=1
Y=Y+Y_i=(0+0.75)=0.75
x=round(X)=round(1)=1
y=round(Y)=round(0.75)=1

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



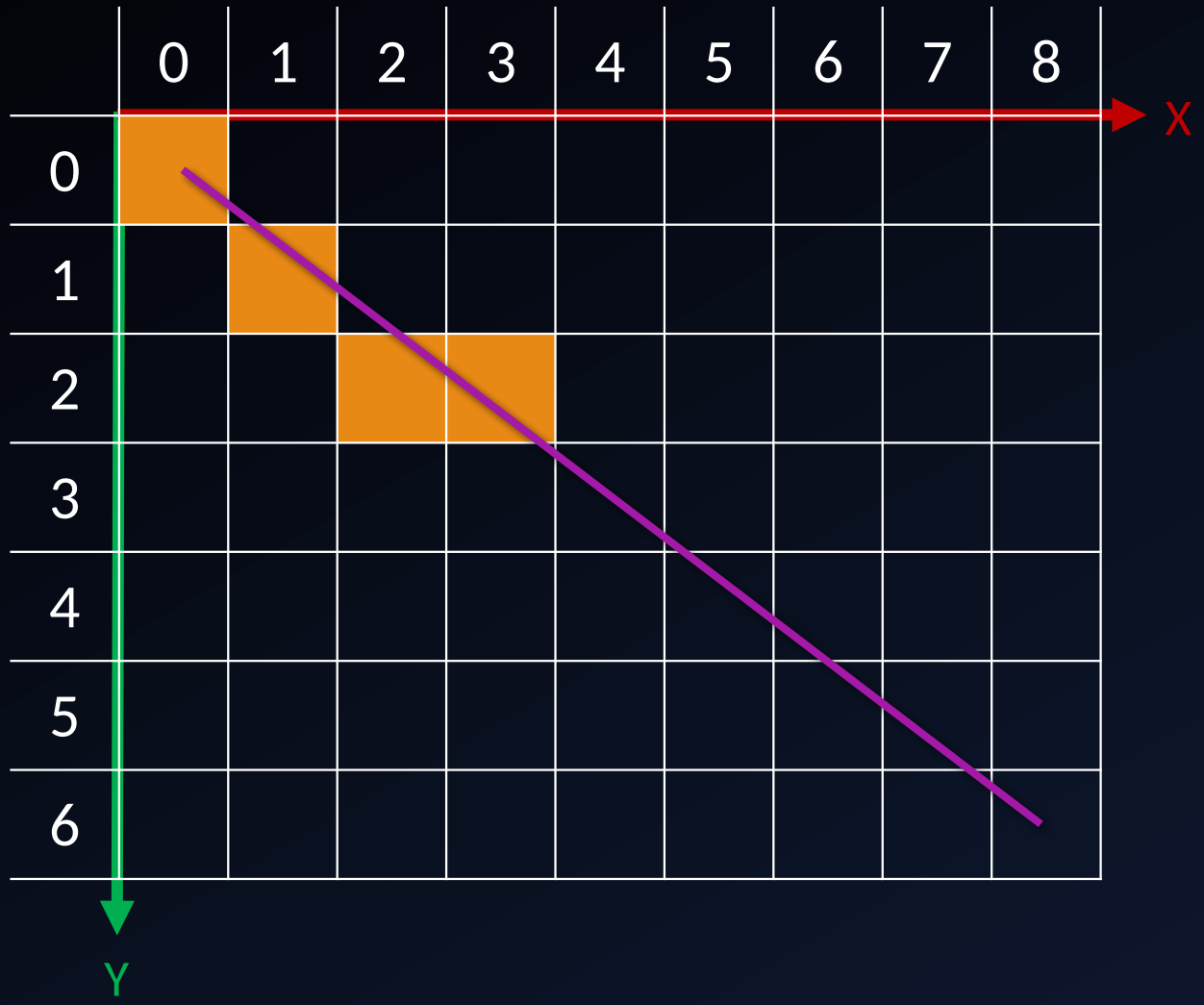
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #2

$X=X+X_i=(1+1)=2$
$Y=Y+Y_i=(0.75+0.75)=1.5$
x=round(X)=round(2)=2
y=round(Y)=round(1.5)=2

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



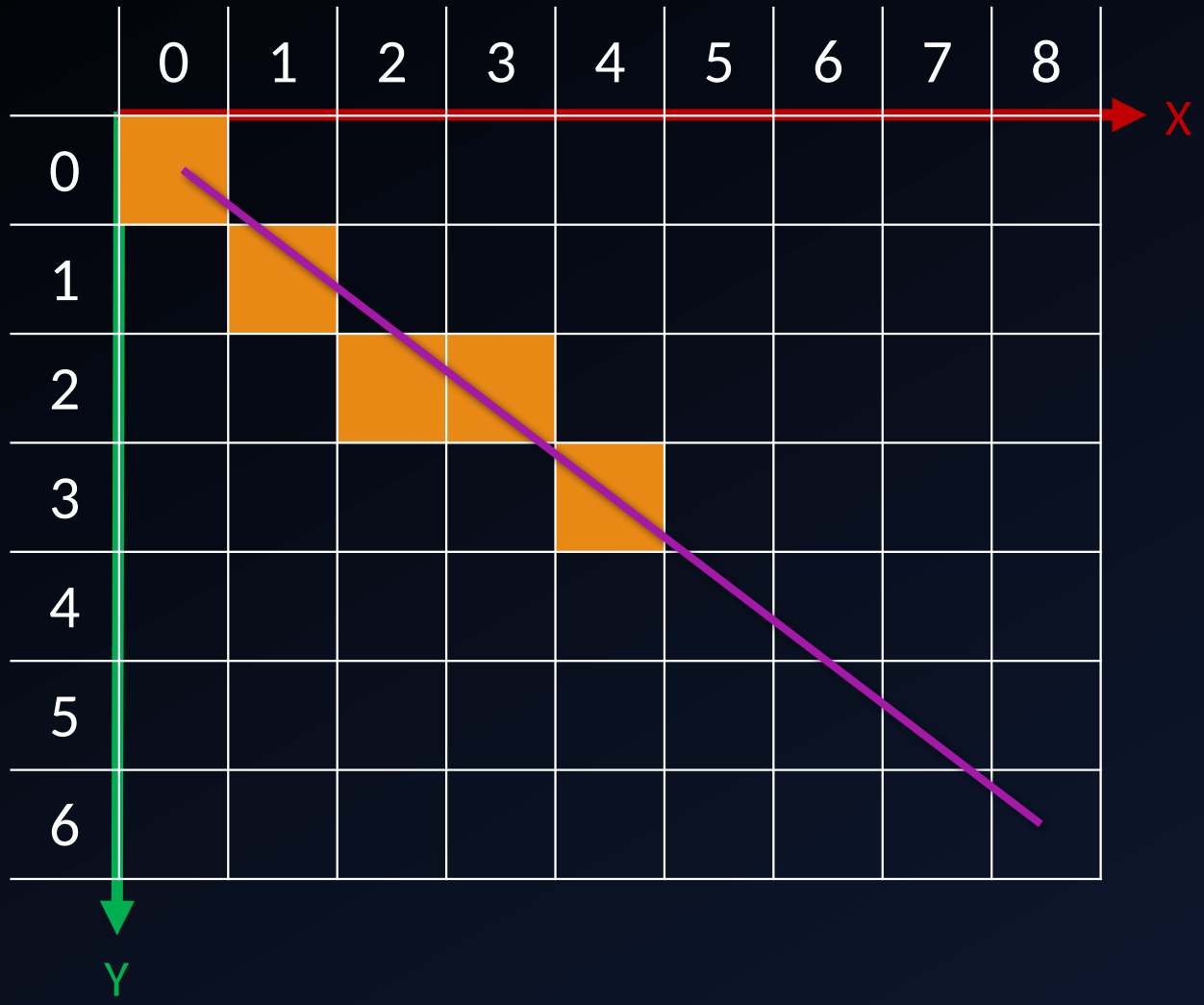
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #3

$X=X+X_i=(2+1)=3$
$Y=Y+Y_i=(1.5+0.75)=2.25$
x=round(X)=round(3)=3
y=round(Y)=round(2.25)=2

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



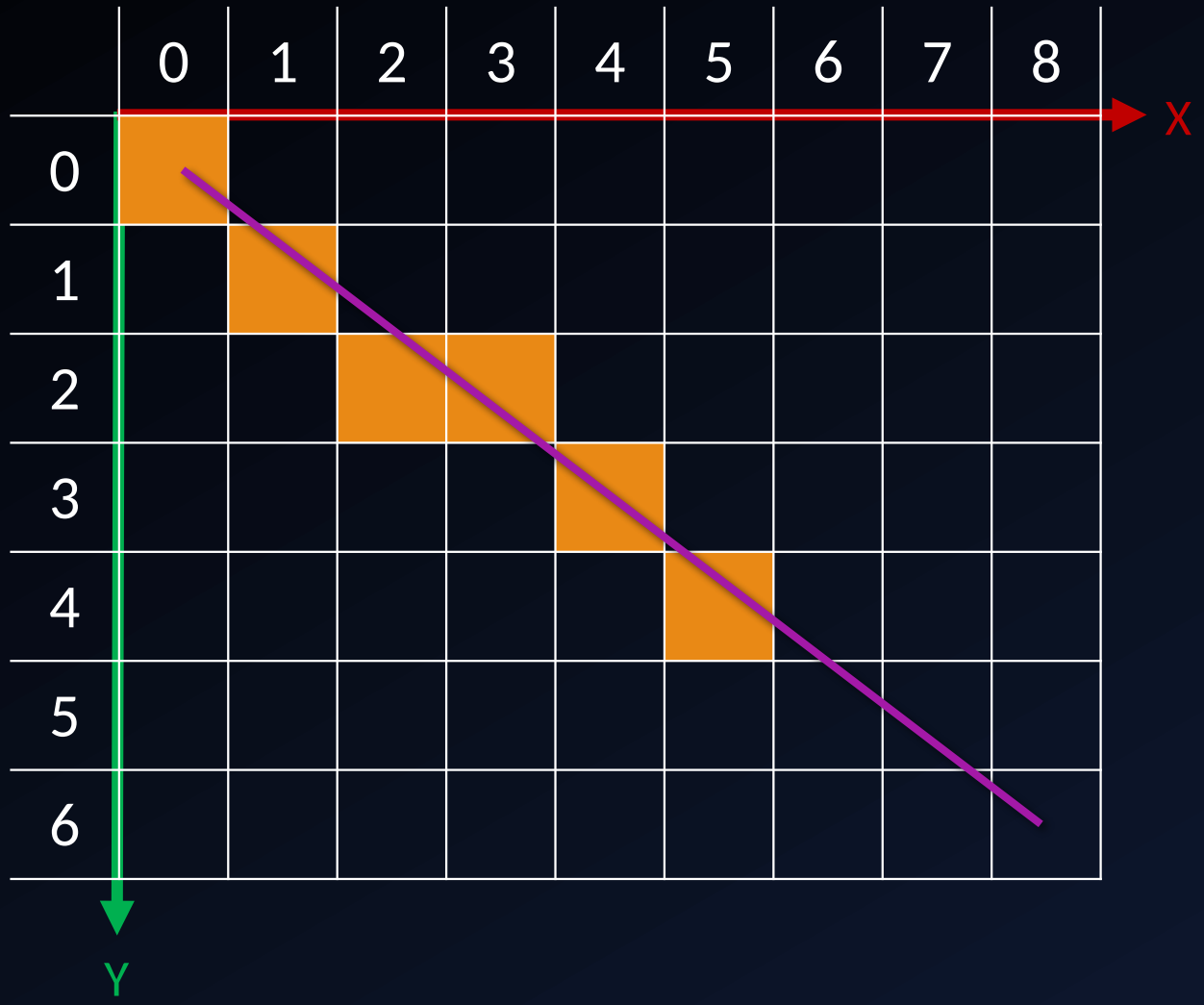
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #4

$X=X+X_i=(3+1)=4$
$Y=Y+Y_i=(2.25+0.75)=3$
x=round(X)=round(4)=4
y=round(Y)=round(3)=3

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



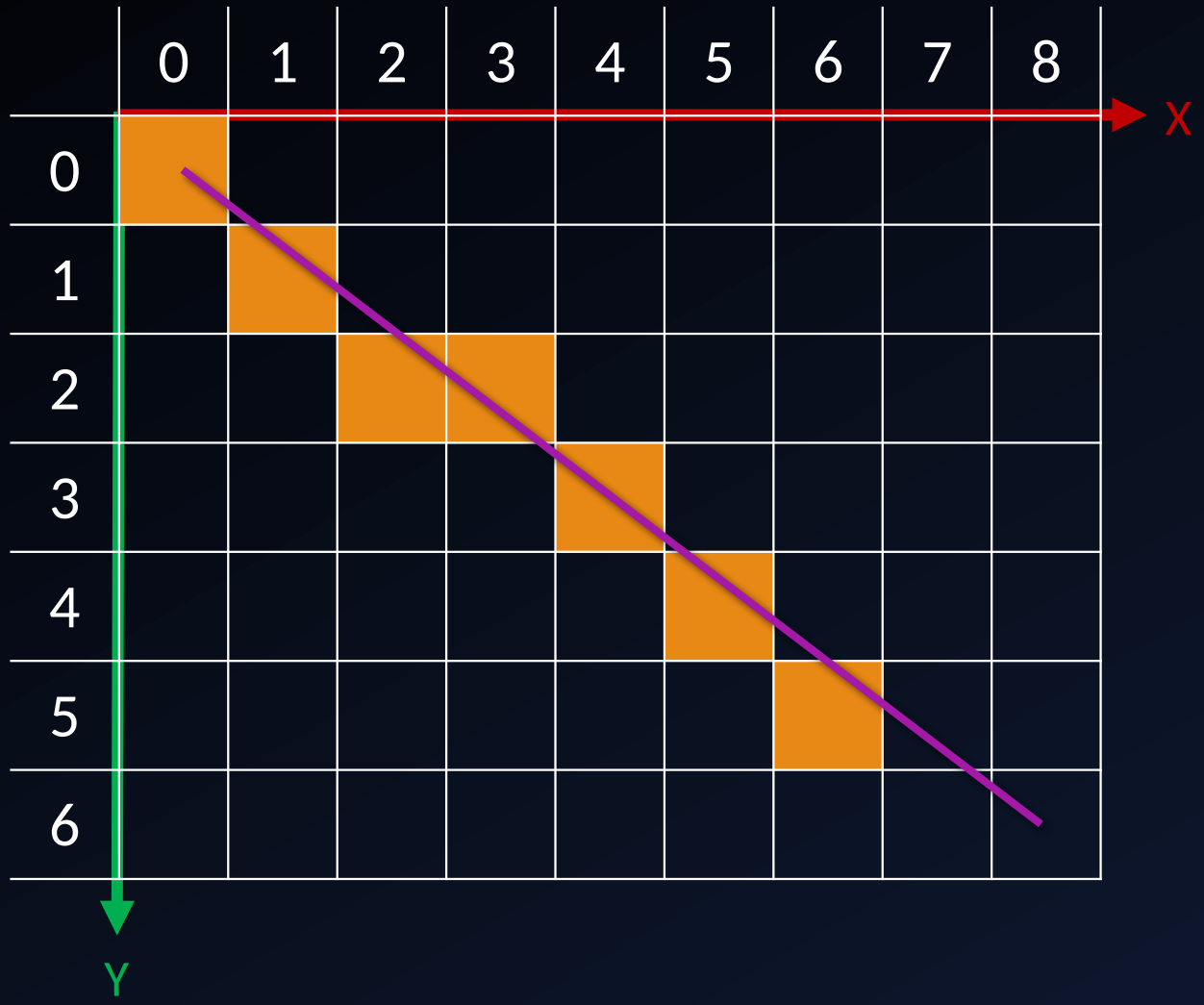
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #5

$X=X+X_i=(4+1)=5$
$Y=Y+Y_i=(3+0.75)=3.75$
x=round(X)=round(5)=5
y=round(Y)=round(3.75)=4

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



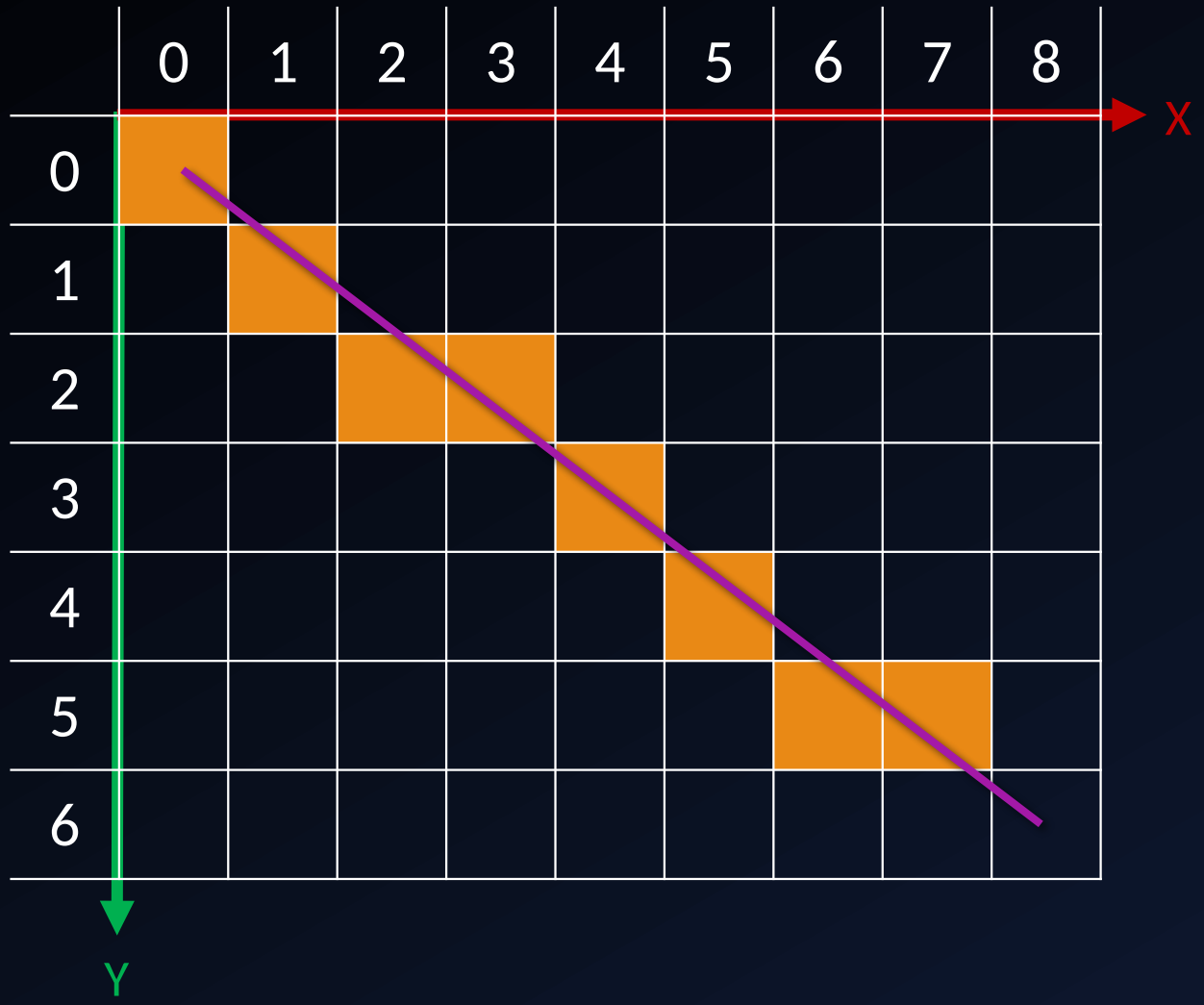
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #6

$X=X+X_i=(5+1)=6$
$Y=Y+Y_i=(3.75+0.75)=4.5$
x=round(X)=round(6)=6
y=round(Y)=round(4.5)=5

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



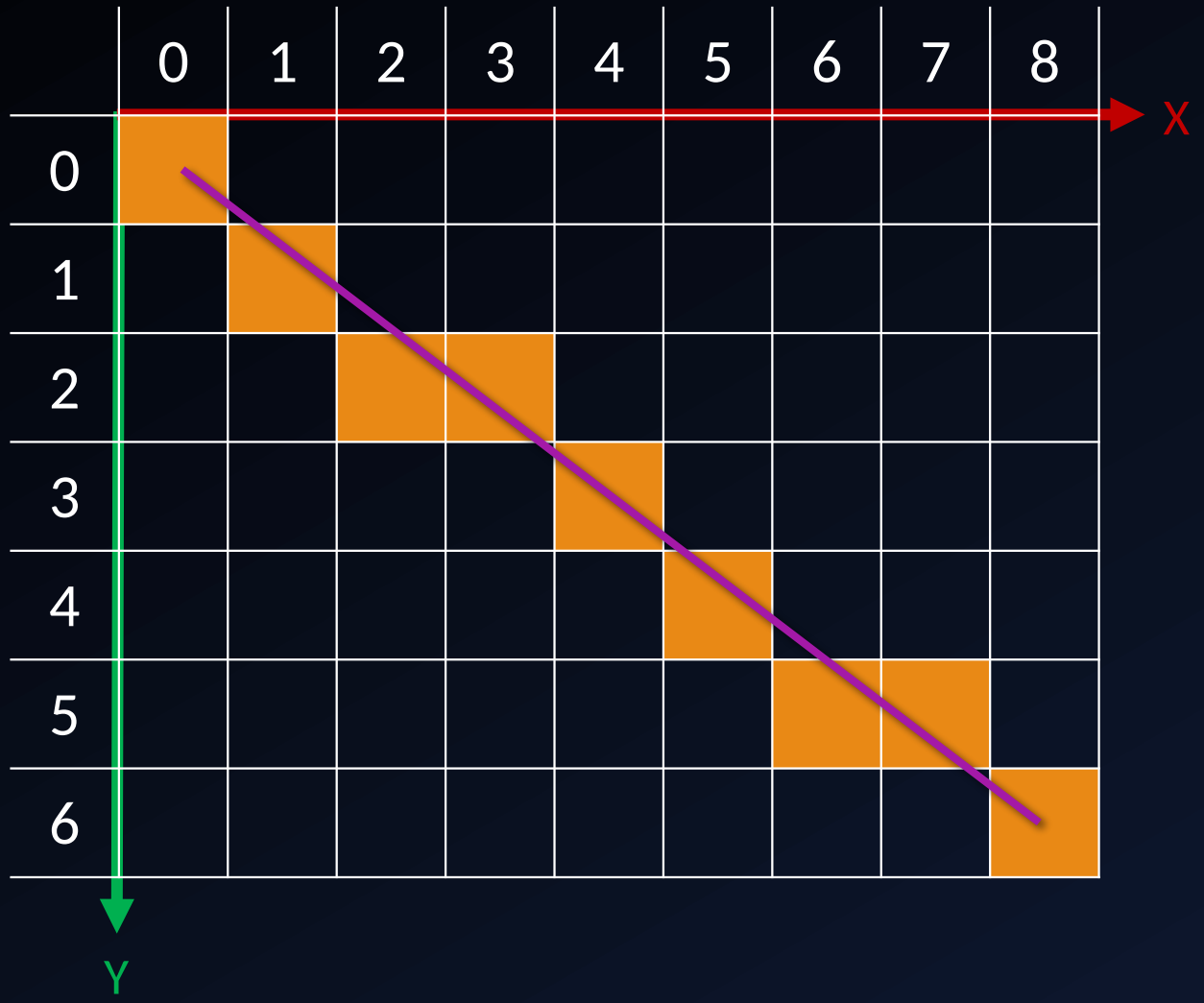
dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #7

X=X+X_i=(6+1)=7
Y=Y+Y_i=(4.5+0.75)=5.25
x=round(X)=round(7)=7
y=round(Y)=round(5.25)=5

Digital Differential Analyzer

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



dx=(8-0)=8
dy=(6-0)=6
step=max(dx,dy)=max(8,6)=8
X_i=dx/step=8/8=1
Y_i=dy/step=6/8=0.75

LOOP #8

$X=X+X_i=(7+1)=8$
$Y=Y+Y_i=(5.25+0.75)=6$
x=round(X)=round(8)=8
y=round(Y)=round(6)=6

BRESENHAM'S LINE ALGORITHM

Bresenham's algorithm — a fundamental method in Computer Graphics — is a clever way of approximating a continuous straight line with discrete pixels, ensuring that the line appears straight and smooth on a pixel-based display. Was invented by Jack Bresenham in 1962 and published in 1965.

Bresenham's algorithm has been extended to produce circles, ellipses, cubic and quadratic Bezier curves, as well as native anti-aliased versions of those.

The algorithm uses the slope (**k**) of the original line along with a value called the "decision parameter“ (**dp**) in tandem to help us choose the pixels that create the most precise straight-line approximation between these two points.

The "decision parameter" guides the algorithm's incremental decisions about whether to move horizontally or diagonally and which pixel to color next.

As the algorithm progresses, we'll continuously update the decision parameter based on whether it was positive, negative, or zero in the previous iteration. This constant refinement helps us minimize the deviation from the original line.

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.

$dx=(8-0)=8$

$dy=(6-0)=6$

$k=dy/dx=6/8=3/4$

$dp_0=-1/2$

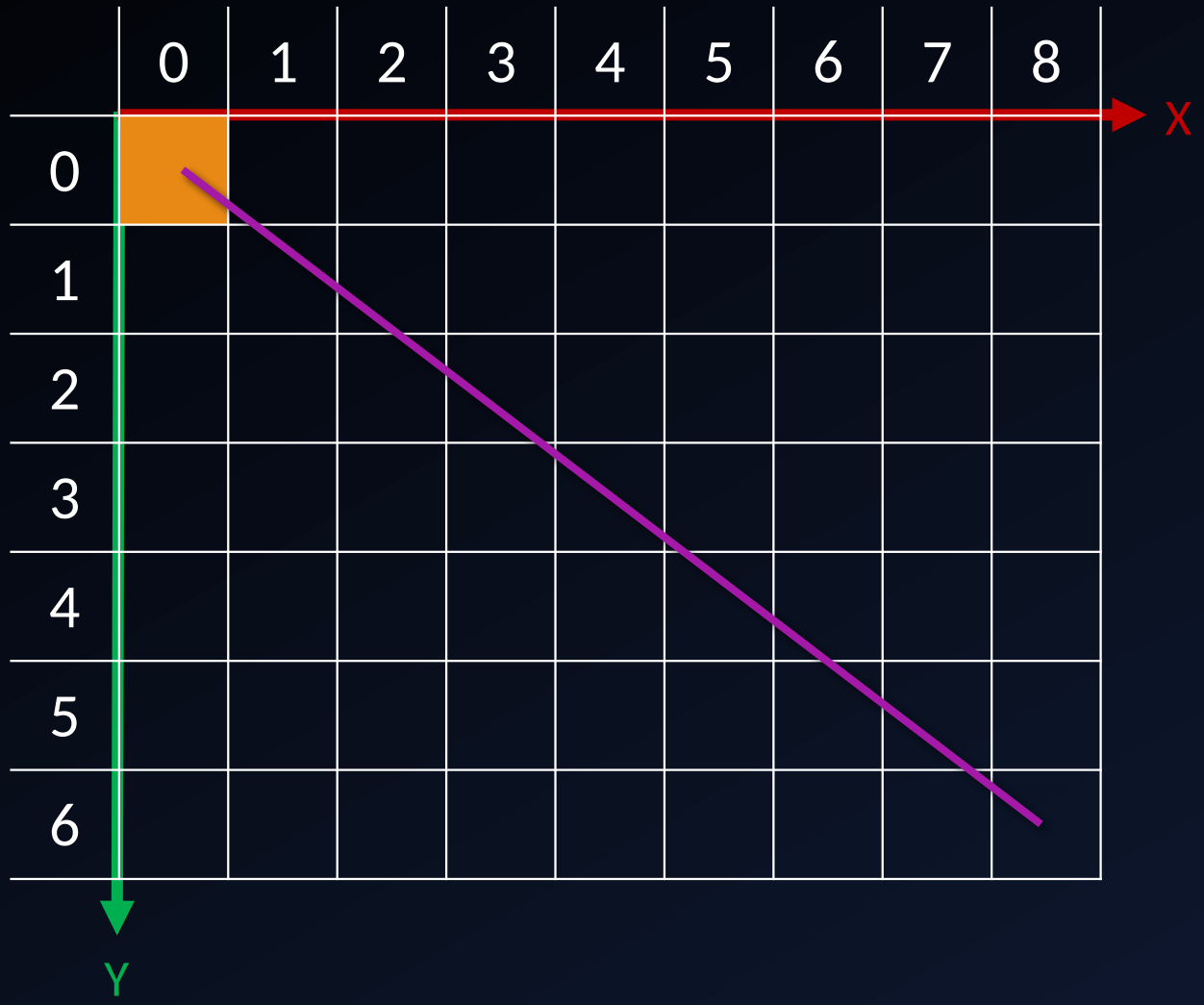
$if(dp_i>=0)\ then\ dp_i=dp_i-1$

$dp_i=dp_{i-1}-2dx$

As you see, the calculations need floating-point arithmetic. So Bresenham multiplies all variables by 2dx and gets only integer arithmetic.

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



$dx=(8-0)=8$

$dy=(6-0)=6$

$k=(dy/dx)*2dx=2dy=12$

$dp_0=(-1/2)*2dx=-dx=-8$

$if(dp_i>=0)\ then\ dp_i=dp_i-2dx$

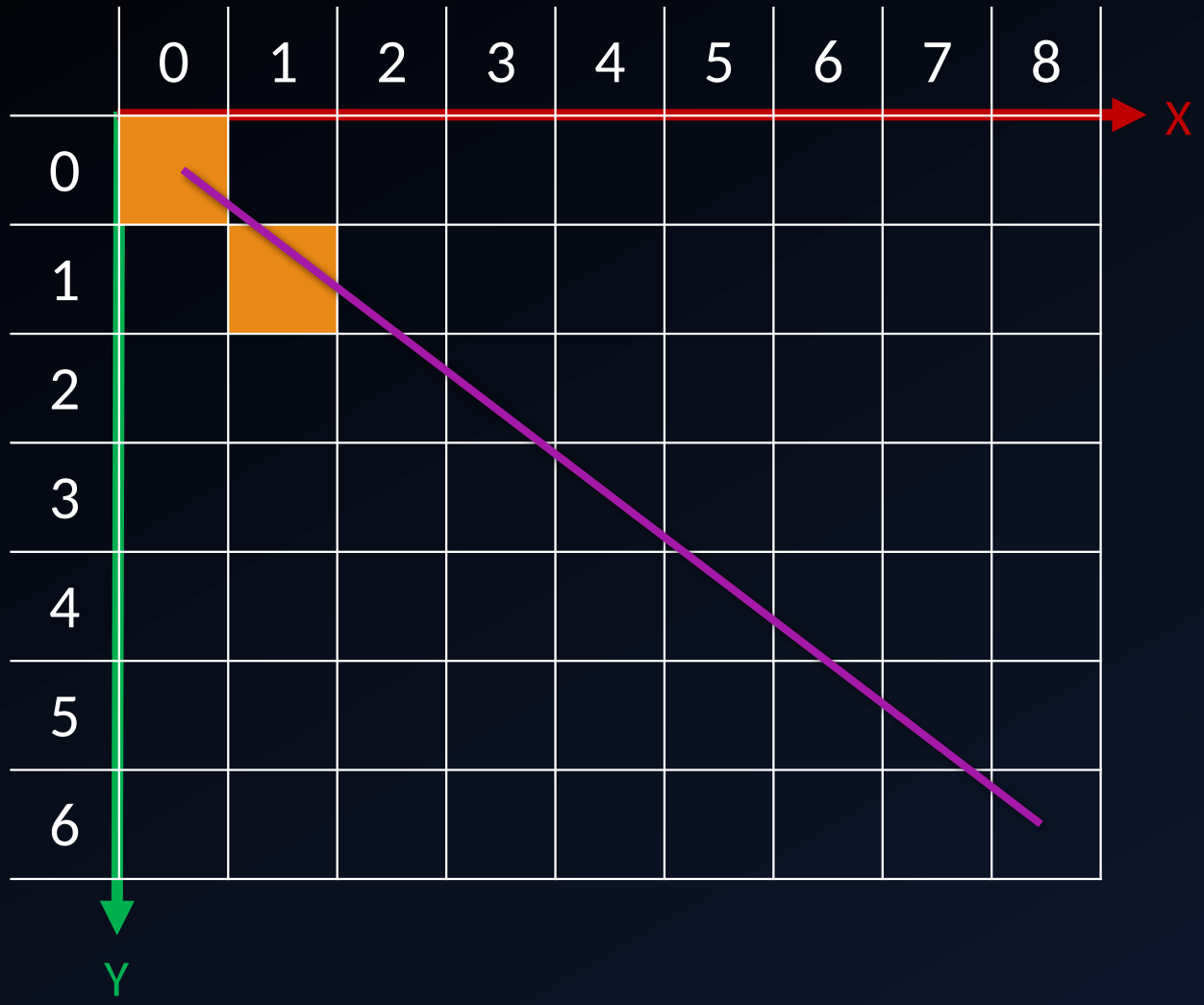
LOOP #0

$X=x_1=0$

$Y=y_1=0$

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



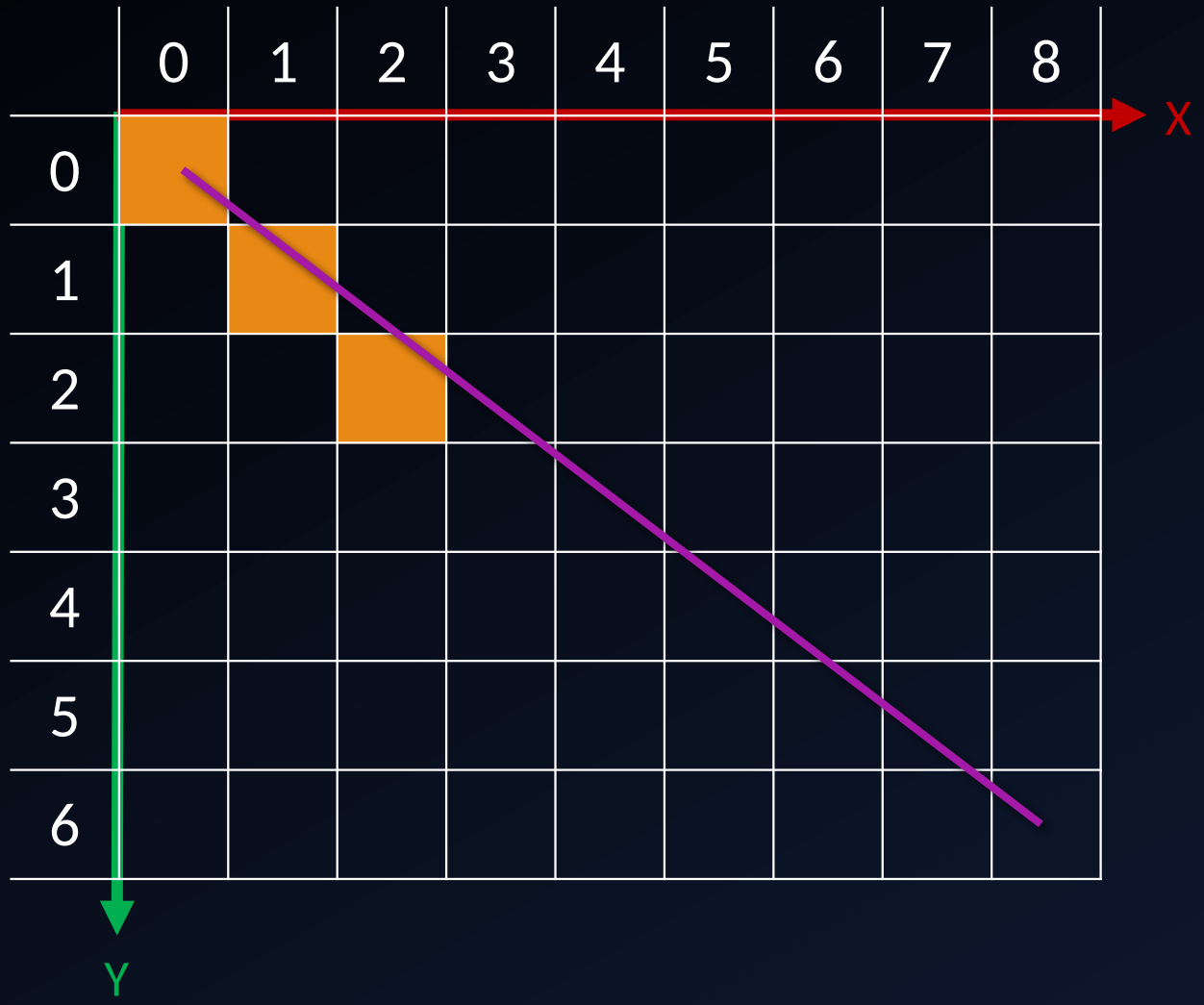
$dx=(8-0)=8$
$dy=(6-0)=6$
$k=(dy/dx)*2dx=2dy=12$
$dp_0=(-1/2)*2dx=-dx=-8$
$if(dp_i>=0)\ then\ dp_i=dp_i-2dx$

LOOP #1

$dp_1=dp_0+k=(-8+12)=4$
$dp_1>=0\ then$
$X=(X+1)=(0+1)=1$
$Y=(Y+1)=(0+1)=1$
$dp_1=dp_1-2dx=(4-2*8)=-12$

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



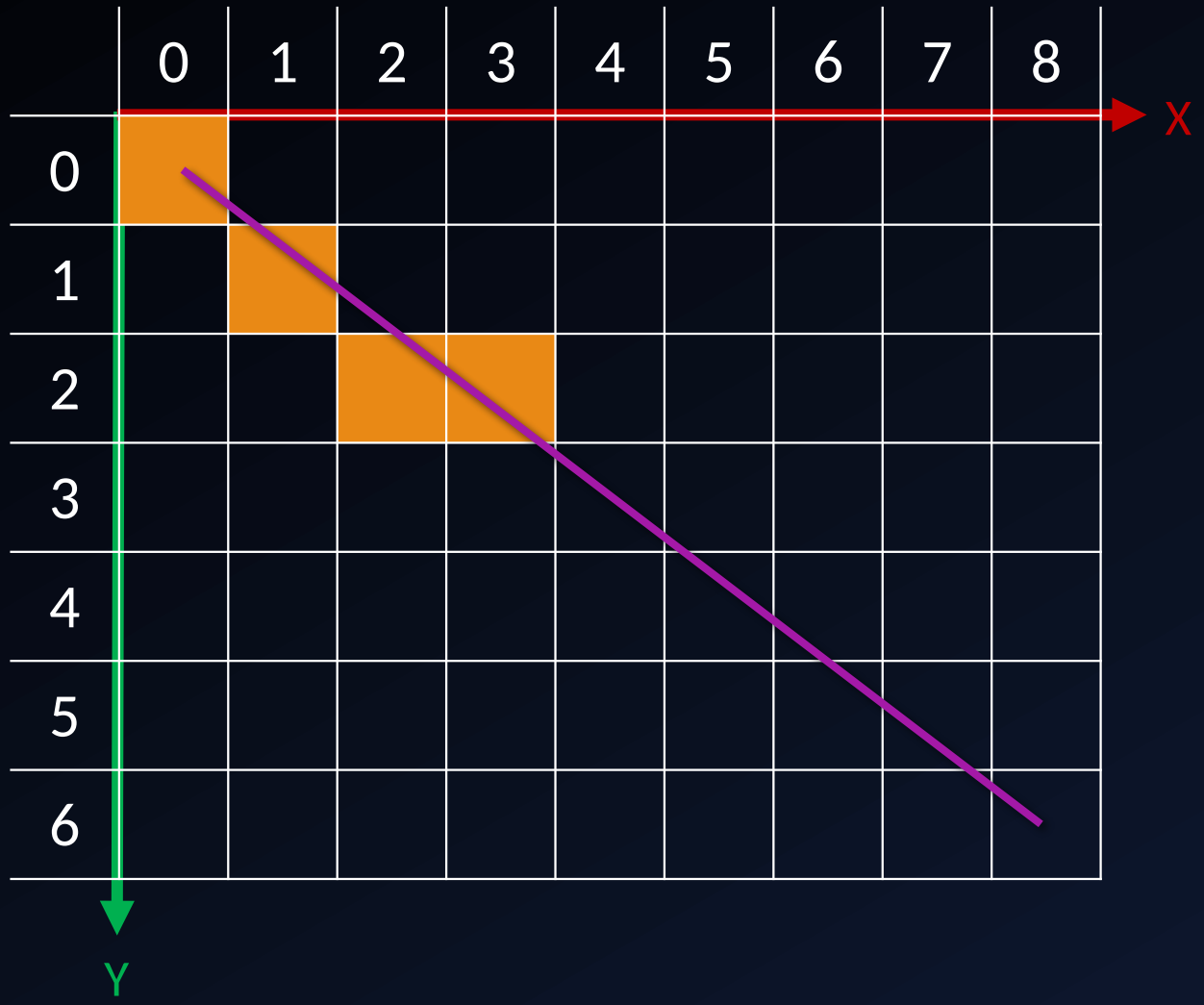
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #2

dp_2=dp_1+k=(-12+12)=0
dp_2>=0 then
X=(X+1)=(1+1)=2
Y=(Y+1)=(1+1)=2
dp_2=dp_2-2dx=(0-2*8)=-16

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



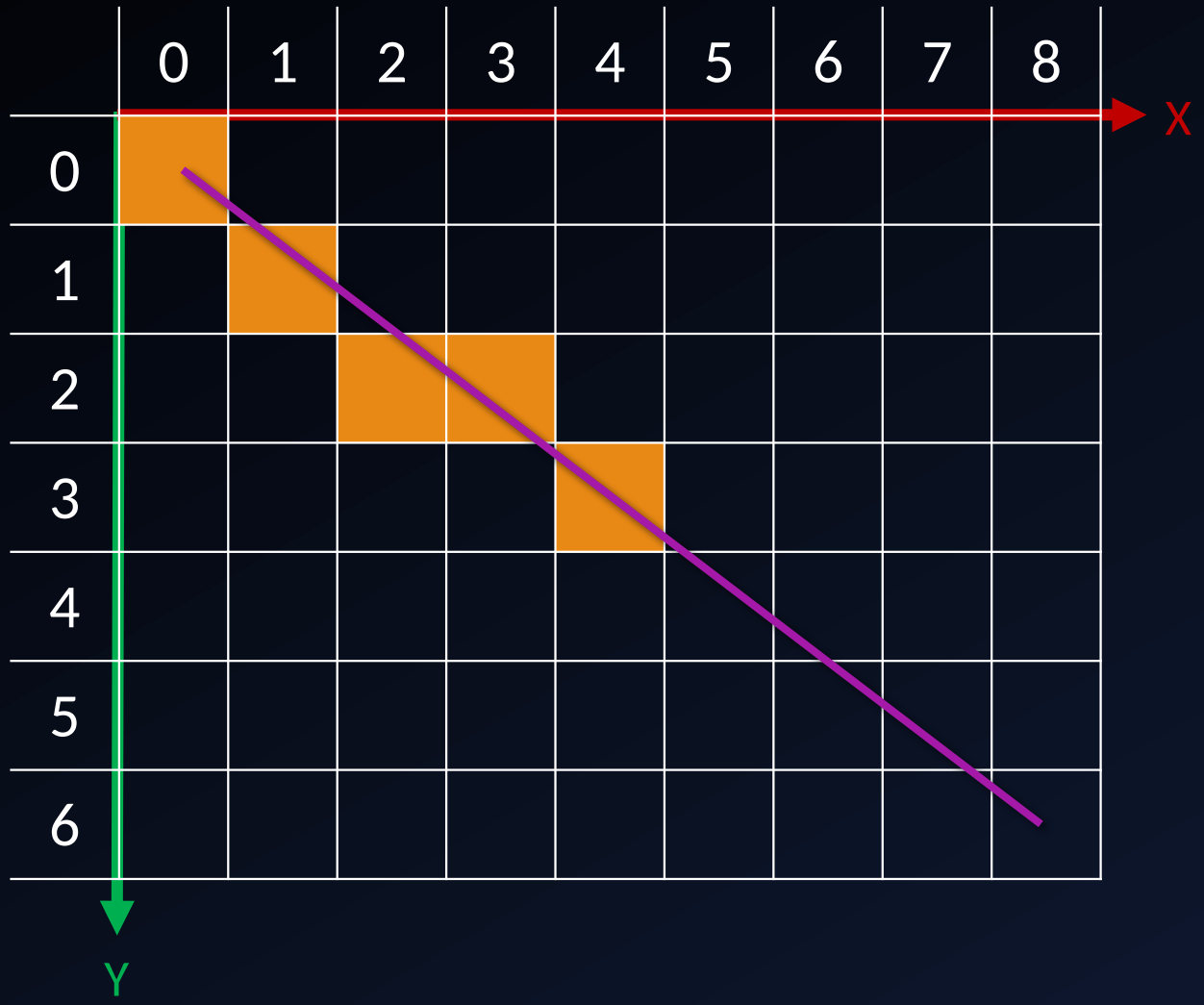
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #3

dp_3=dp_2+k=(-16+12)=-4
dp_3<0 then
X=(X+1)=(2+1)=3
Y=(Y+0)=(2+0)=2
dp_3=dp_3

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



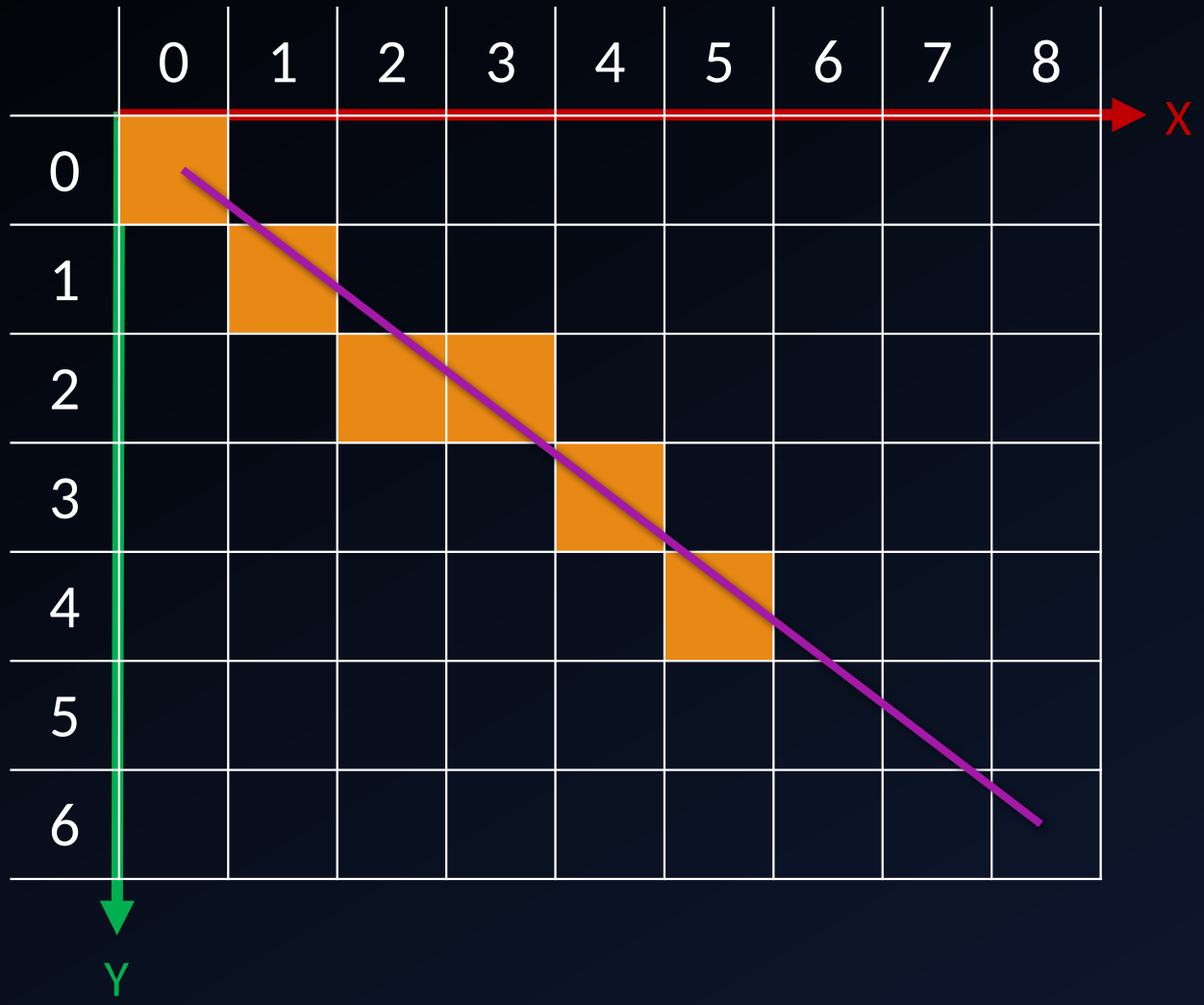
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #4

dp_4=dp_3+k=(-4+12)=8
dp_4>=0 then
X=(X+1)=(3+1)=4
Y=(Y+1)=(2+1)=3
dp_4=dp_4-2dx=(8-2*8)=-8

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



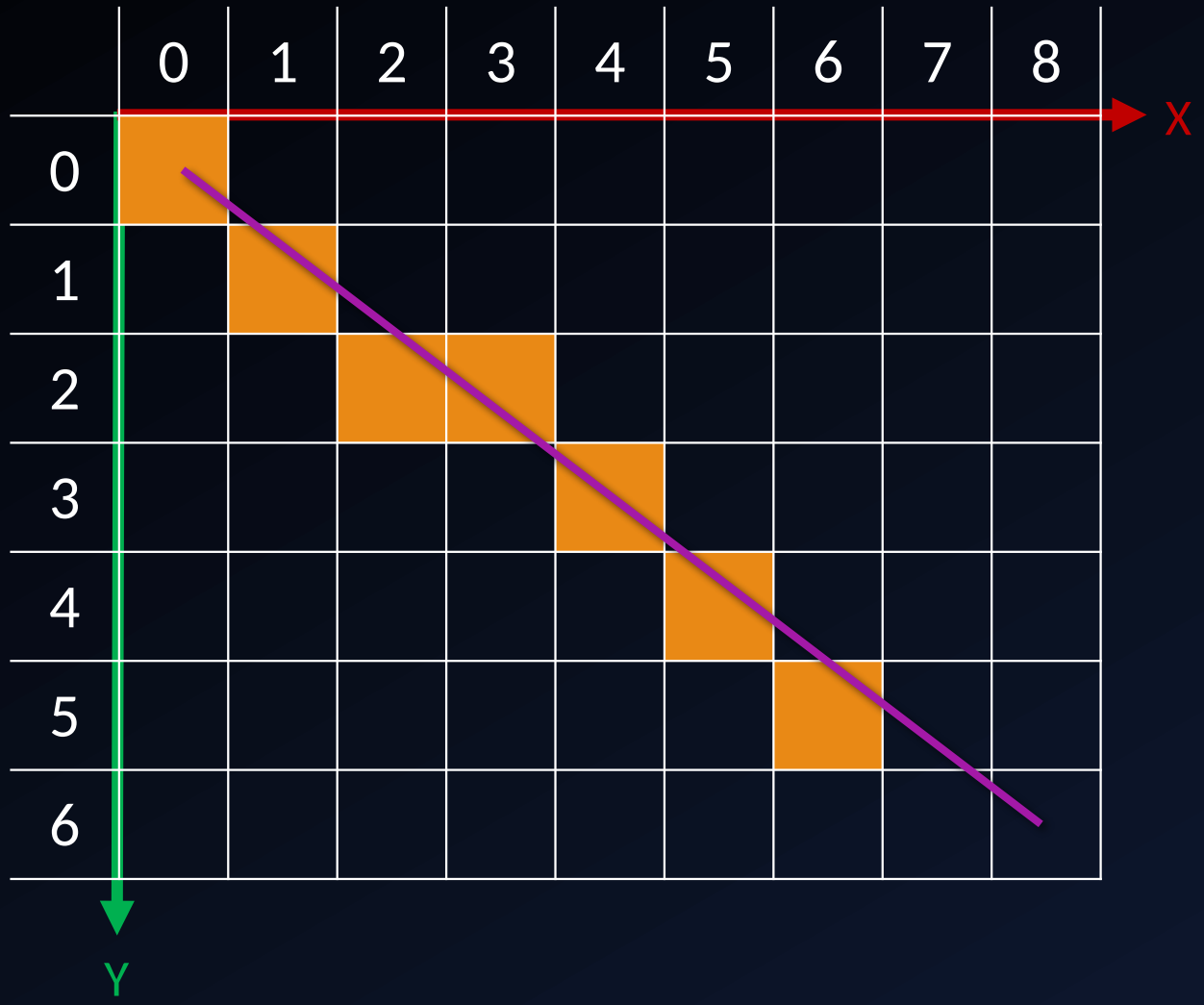
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #5

dp_5=dp_4+k=(-8+12)=4
dp_5>=0 then
X=(X+1)=(4+1)=5
Y=(Y+1)=(3+1)=4
dp_5=dp_5-2dx=(4-2*8)=-12

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



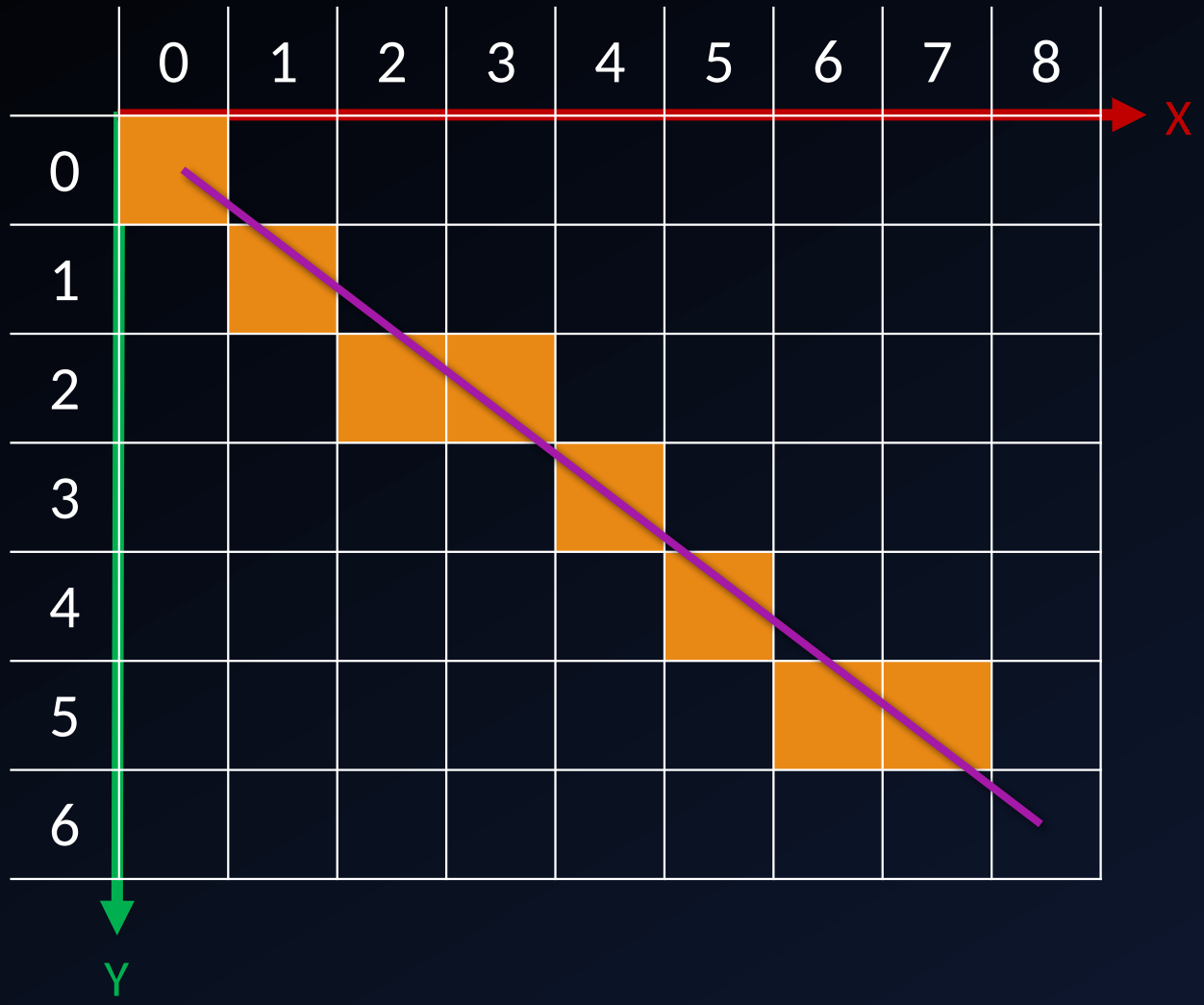
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #6

dp_6=dp_5+k=(-12+12)=0
dp_6>=0 then
X=(X+1)=(5+1)=6
Y=(Y+1)=(4+1)=5
dp_6=dp_6-2dx=(0-2*8)=-16

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



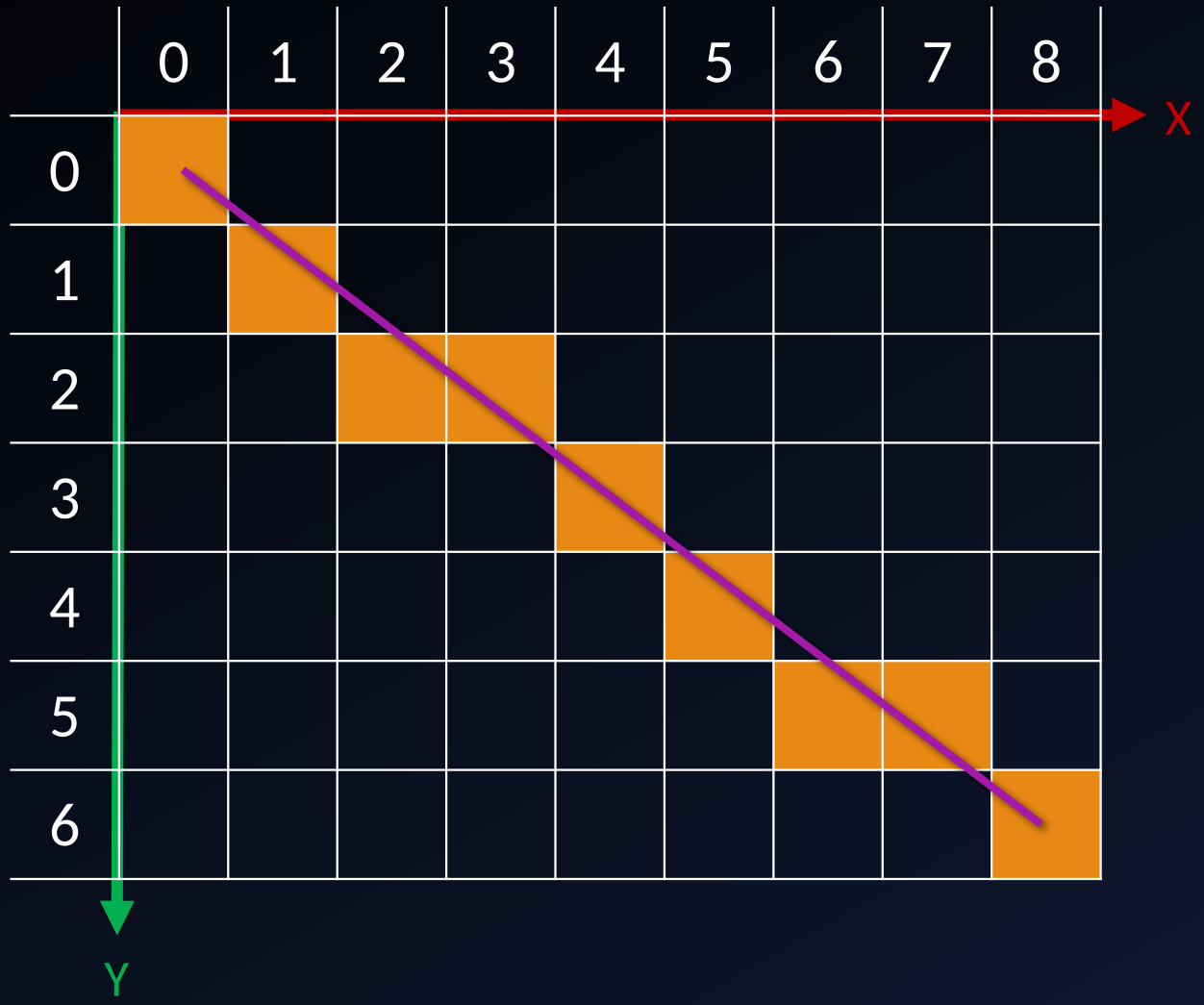
dx=(8-0)=8
dy=(6-0)=6
k=(dy/dx)*2dx=2dy=12
dp_0=(-1/2)*2dx=-dx=-8
if(dp_i>=0) then dp_i=dp_i-2dx

LOOP #7

dp_7=dp_6+k=(-16+12)=-4
dp_7<0 then
X=(X+1)=(6+1)=7
Y=(Y+0)=(5+0)=5
dp_7=dp_7

BRESENHAM'S LINE ALGORITHM

An example. We have a line (0,0) - (8,6). Need to find pixels for the line rendering.



$dx=(8-0)=8$

$dy=(6-0)=6$

$k=(dy/dx)*2dx=2dy=12$

$dp_0=(-1/2)*2dx=-dx=-8$

$if(dp_i>=0)\ then\ dp_i=dp_i-2dx$

LOOP #8

$dp_8=dp_7+k=(-4+12)=8$

$dp_8>=0\ then$

$X=(X+1)=(7+1)=8$

$Y=(Y+1)=(5+1)=6$

$dp_8=dp_8-2dx=(8-2*8)=-8$

Thank you!