2015ゲームクリエイター特訓
プログラムコースI
講義資料B　反射ベクトル

2015/06/06 Yuji YASUHARA

この部分を解説

```
void OnCollisionEnter(Collision collision) {
      if (collision.contacts.Length > 0) {/* hit for wall or paddle */
         /* calculate the reflecting vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
   }
```

ボールが
何かに接触したら
跳ね返ってほしい

ボールは

速度ベクトルを

持っている

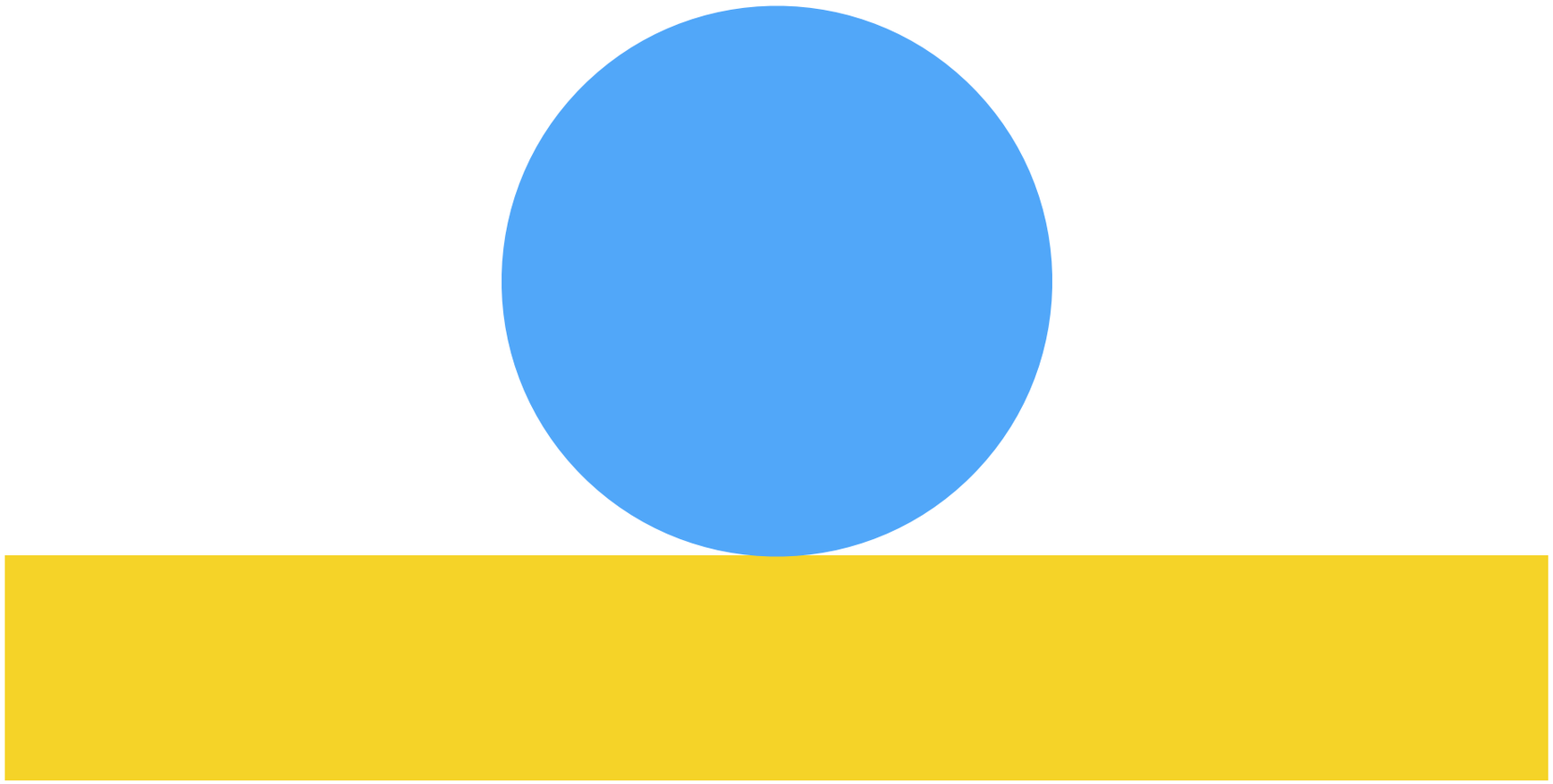
速度ベクトルとは：

どのぐらい速いかと

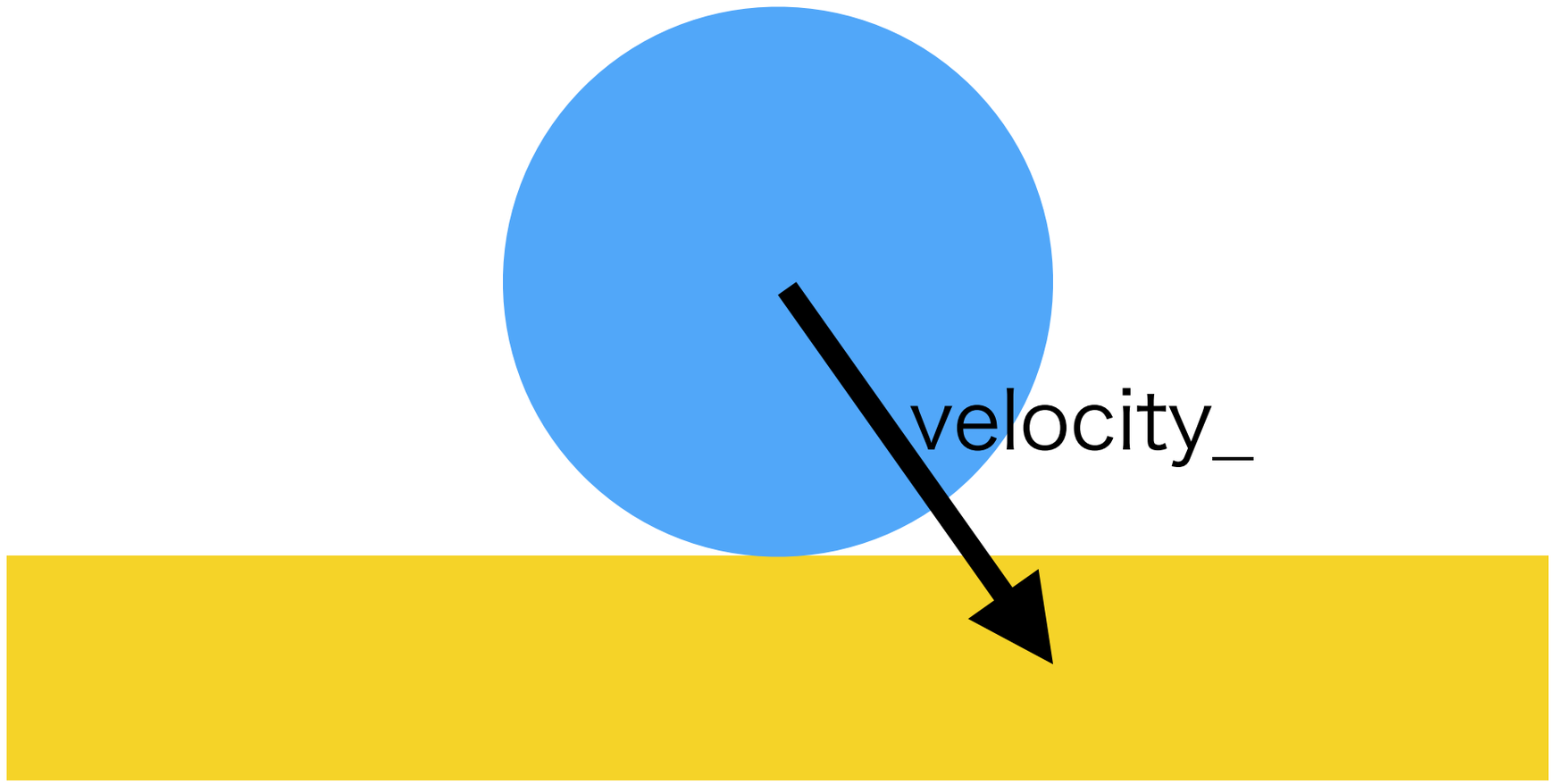
動いている方向を表す

接触したときに
速度ベクトルを
適切に変更する！

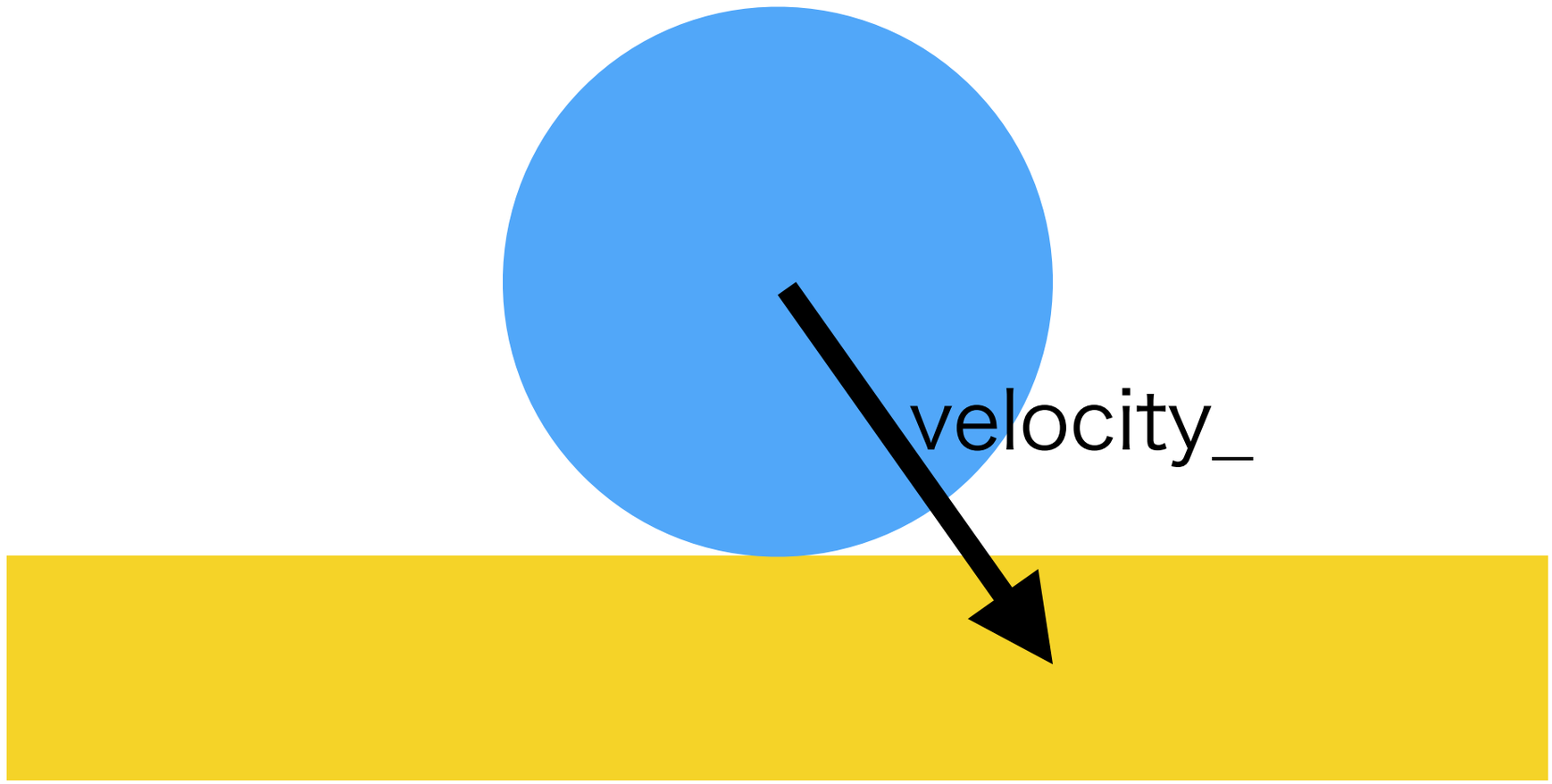
ぶつかった状態が
こうだとする



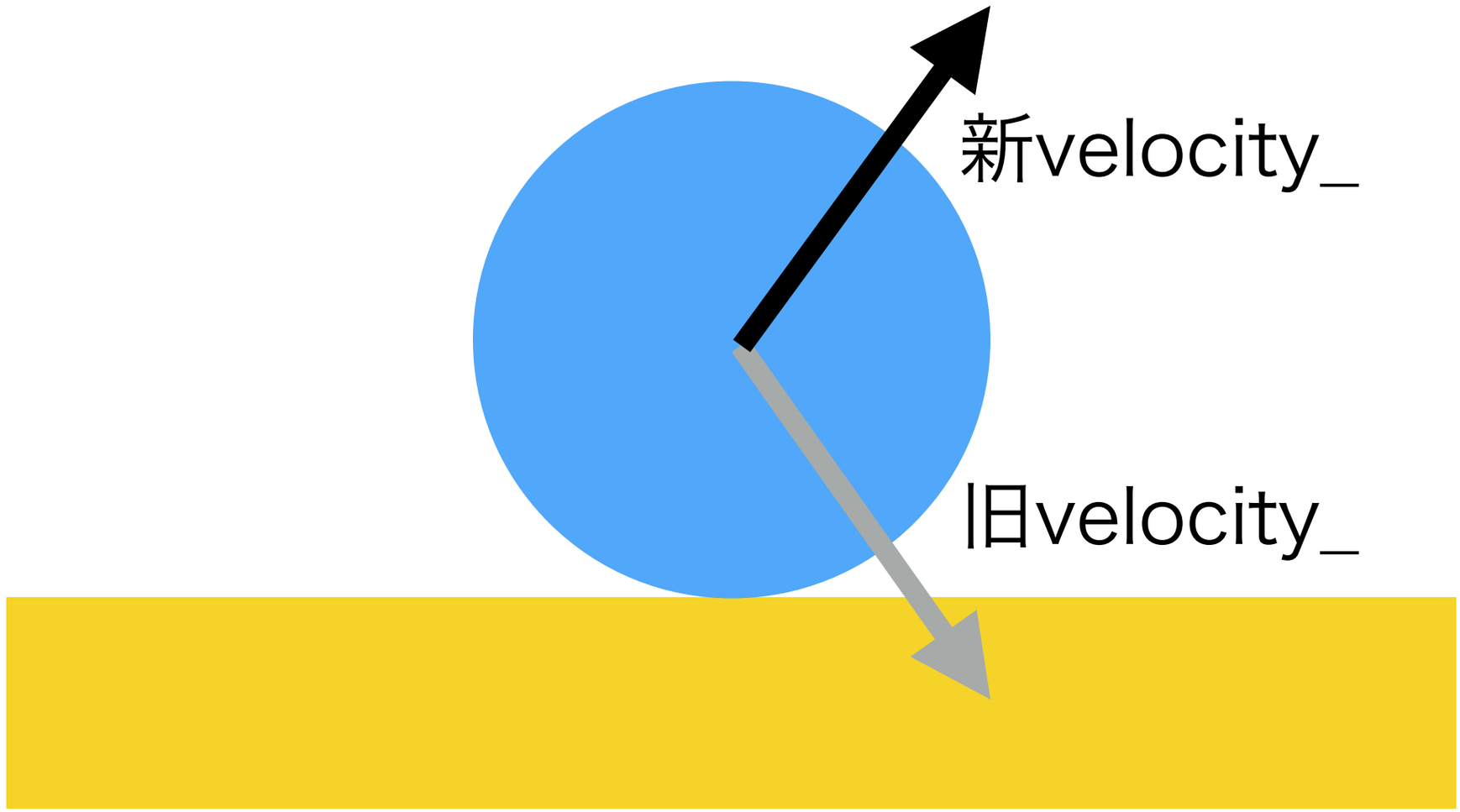
velocity_ は
ボールの速度ベクトル



速度ベクトルの方向に
進んできたので



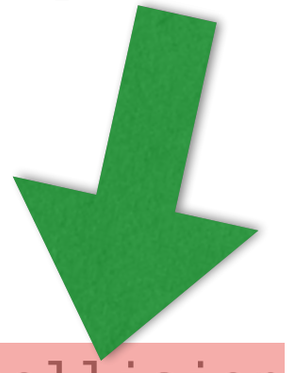
velocity_ を
こう変更できれば勝利



OnCollisionEnter
は接触時に呼ばれる

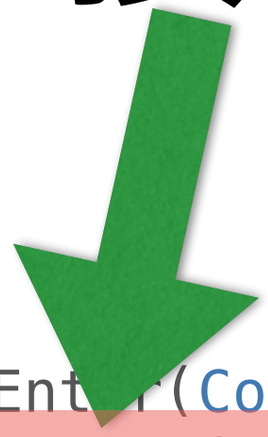
```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
      /* calculate the reflecting vector */
      Vector3 p = collision.contacts[0].point;
      Vector3 n = transform.position - p;
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

これが相手の情報



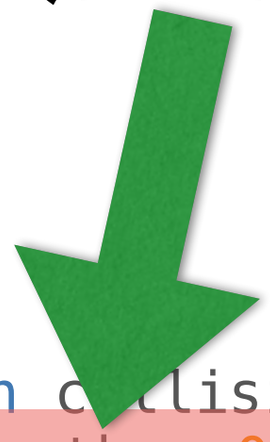
```
void OnCollisionEnter(Collision collision) {
      if (collision.contacts.Length > 0) {/* hit for wall or paddle */
         /* calculate the reflecting vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
   }
```

接触点の集合



```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
      /* calculate the reflecting vector */
      Vector3 p = collision.contacts[0].point;
      Vector3 n = transform.position - p;
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

ひとつ以上あるので



```
void OnCollisionEnter(Collision collision) {
      if (collision.contacts.Length > 0) {/* hit for wall or paddle */
         /* calculate the reflecting vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
   }
```

最初の接触点だけを扱う（手抜き）

```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) { /* hit for wall or paddle */
      /* calculate the reflecting vector */
      Vector3 p = collision.contacts[0].point;
      Vector3 n = transform.position - p;
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

手抜き：
同時に複数の
接触点がある
場合を考慮していない

接触点が
p に格納される

```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
        /* calculate the reflecting vector */
        Vector3 p = collision.contacts[0].point;
        Vector3 n = transform.position - p;
        n.Normalize();
        float v = -2 * Vector3.Dot(velocity_, n);
        var r = n * v;
        velocity_ += r; /* change the velocity vector */
    }
}
```

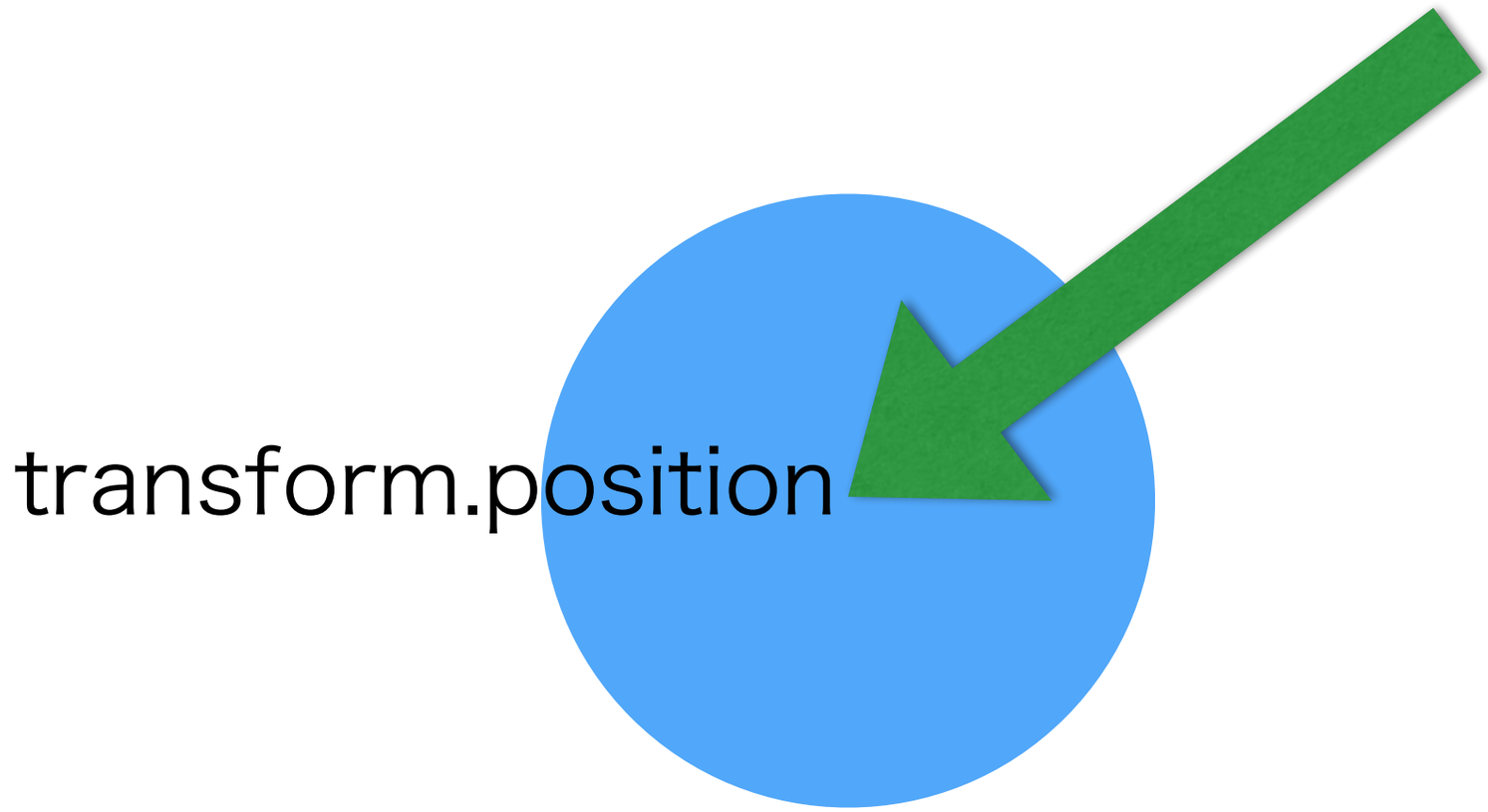
これはボールの位置

```
void OnCollisionEnter(Collisio collision) {
      if (collision.contacts  th > 0) {/* hit for wall or paddle */
         /* calculate the ref ng vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
   }
```

接触点との差

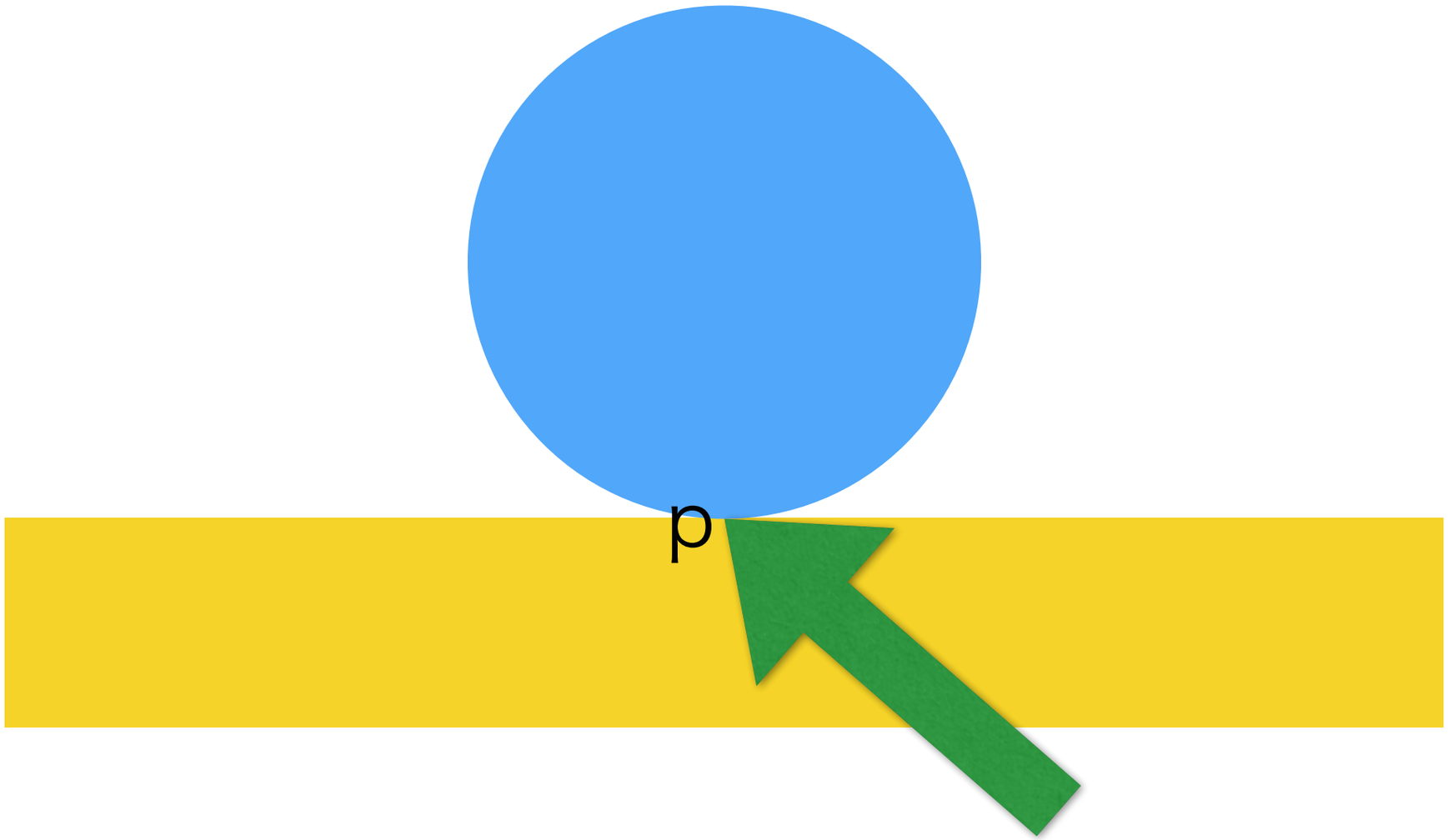
```
void OnCollisionEnter(Collisio collision) {
      if (collision.contacts  th > 0) {/* hit for wall or paddle */
         /* calculate the ref  ng vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
}
```

ボールの位置は中心
これが transform.position



接触位置はここ

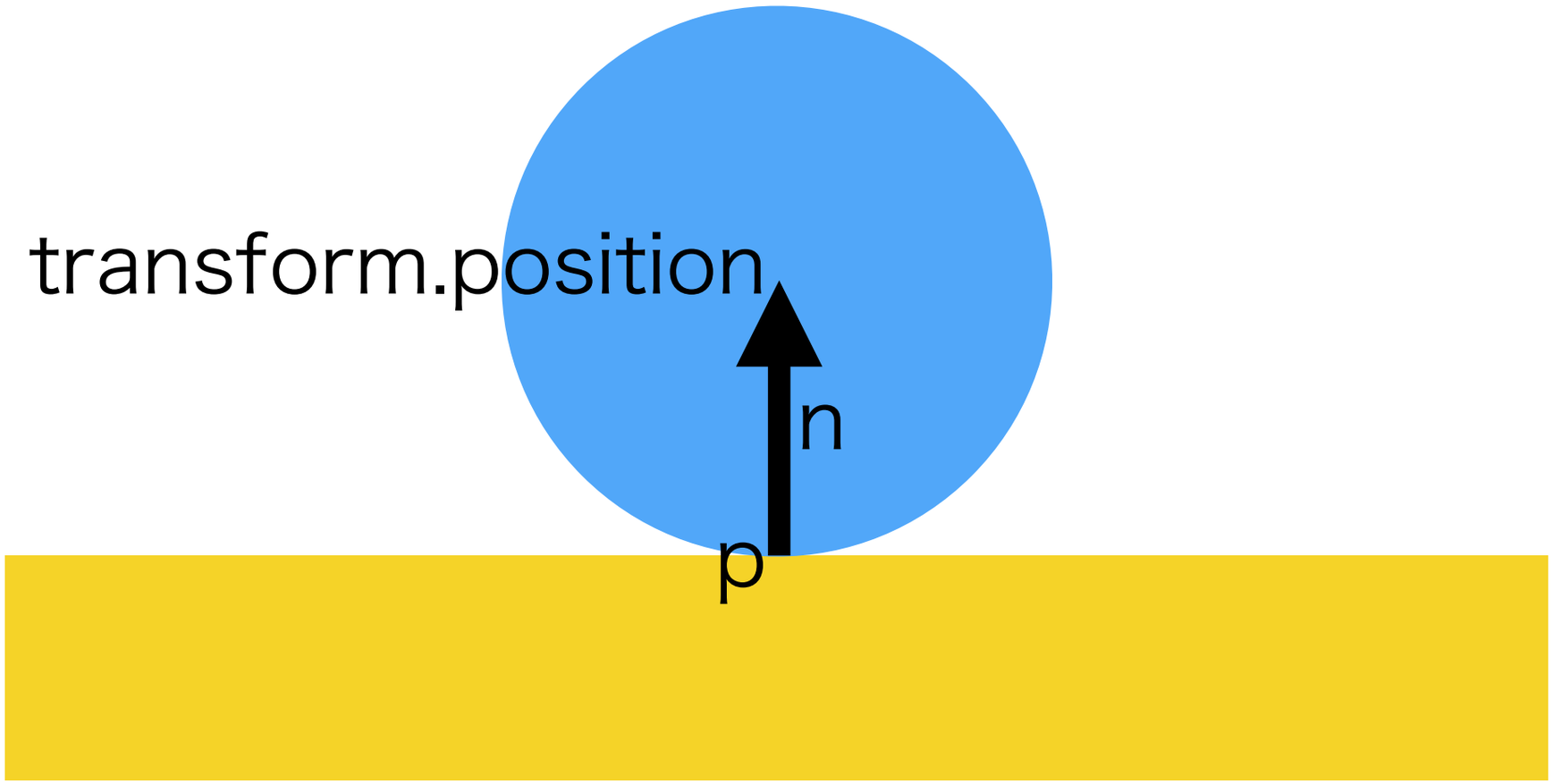
これが p



なのでこの n は

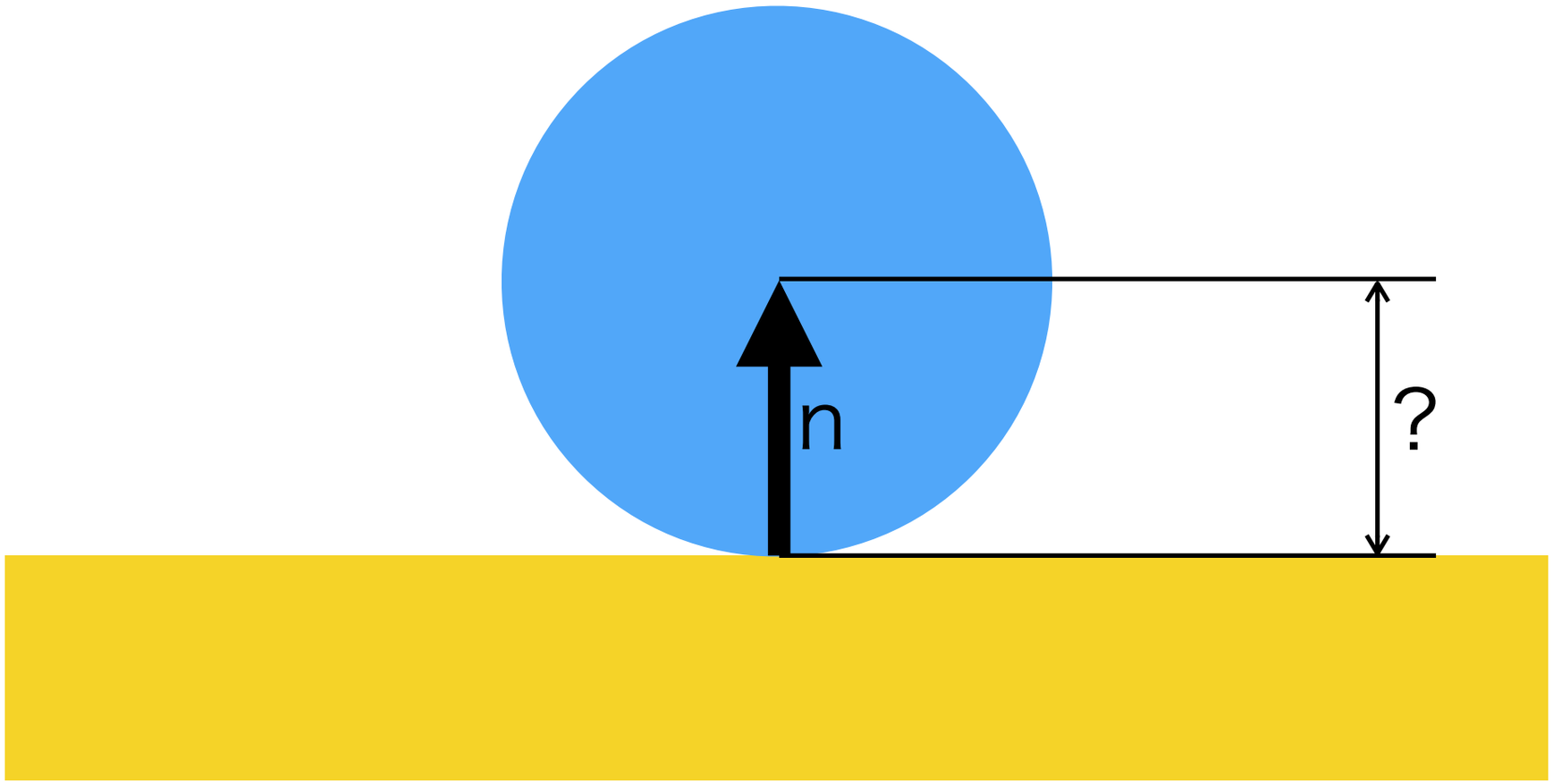
```
void OnCollisionEnter(Collisio collision) {
      if (collision.contacts  th > 0) {/* hit for wall or paddle */
         /* calculate the ref ng vector */
         Vector3 p = collision.contacts[0].point;
         Vector3 n = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
}
```

このベクトルになる
向きはこう



このベクトルの長さは？

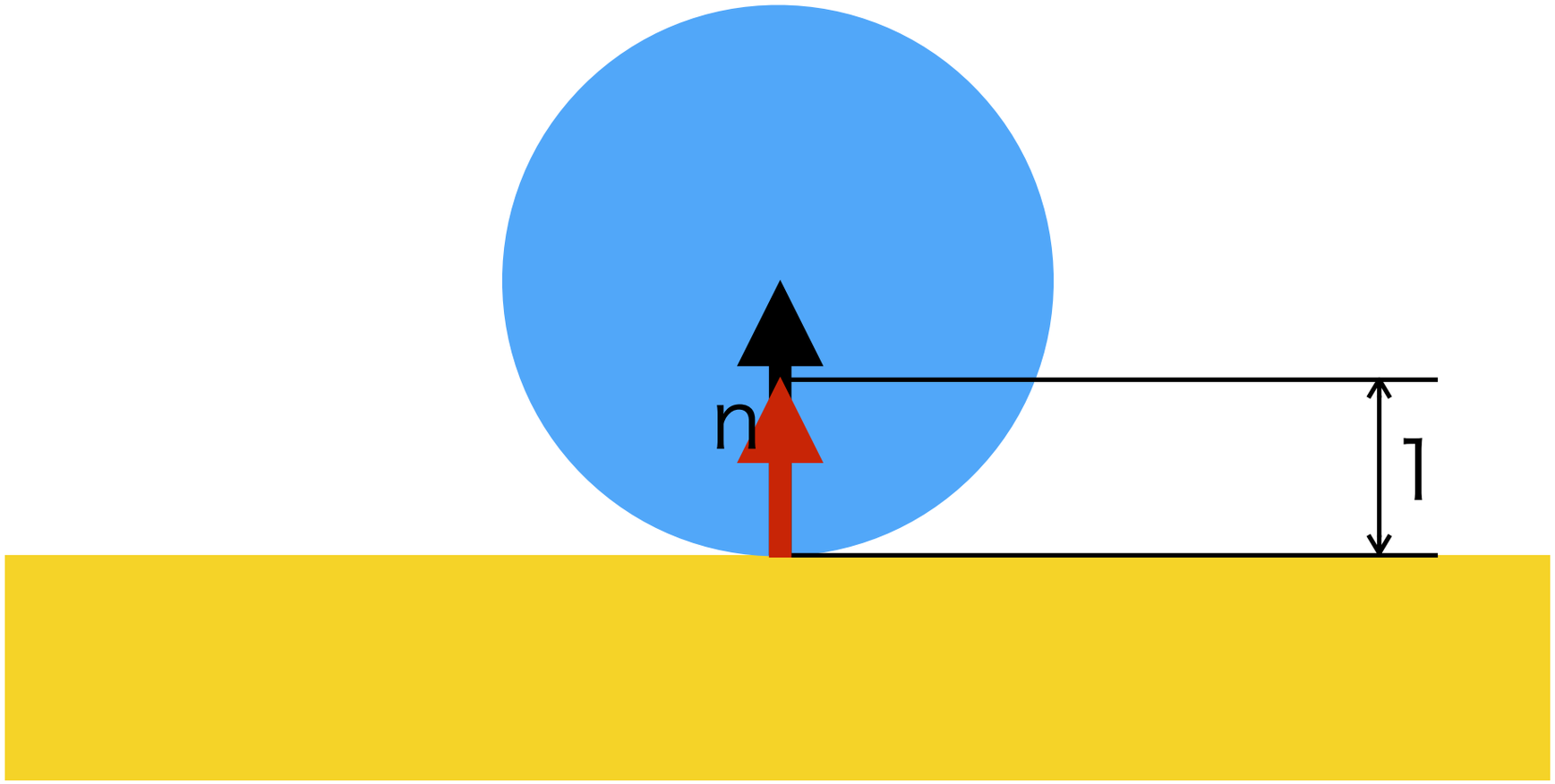
1ではない



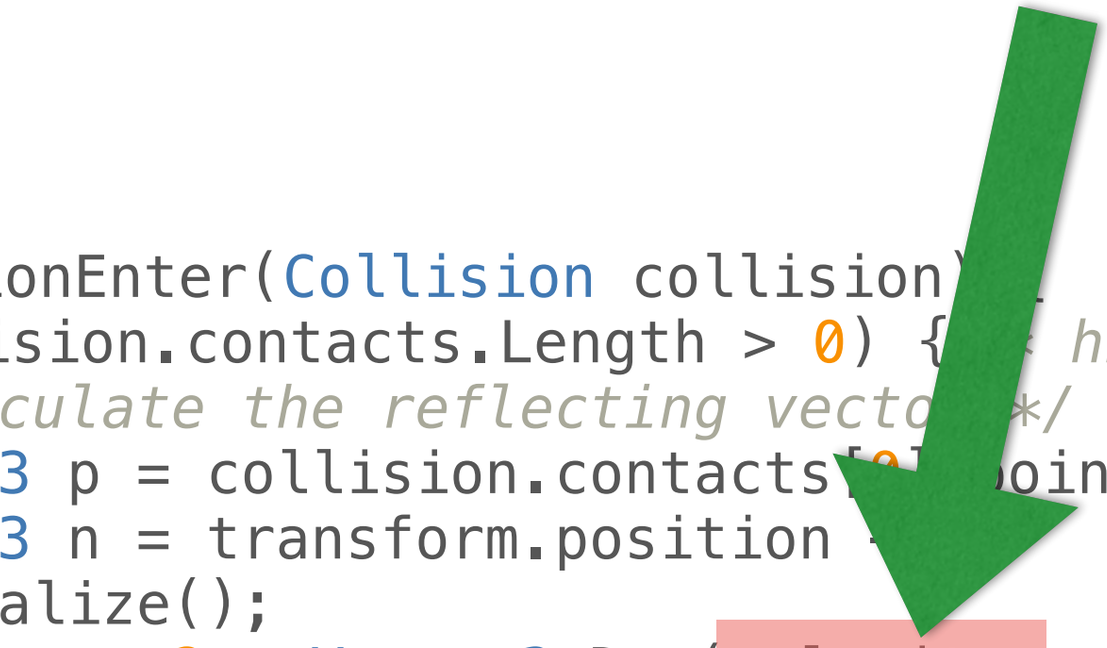
Normalize()
は正規化する処理

```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
        /* calculate the reflecting vector */
        Vector3 p = collision.contacts[0].point;
        Vector3 n = transform.position - p;
        n.Normalize();
        float v = -2 * Vector3.Dot(velocity_, n);
        var r = n * v;
        velocity_ += r; /* change the velocity vector */
    }
}
```

正規化すると
方向は同じで長さが1
になる（重要）



速度ベクトル

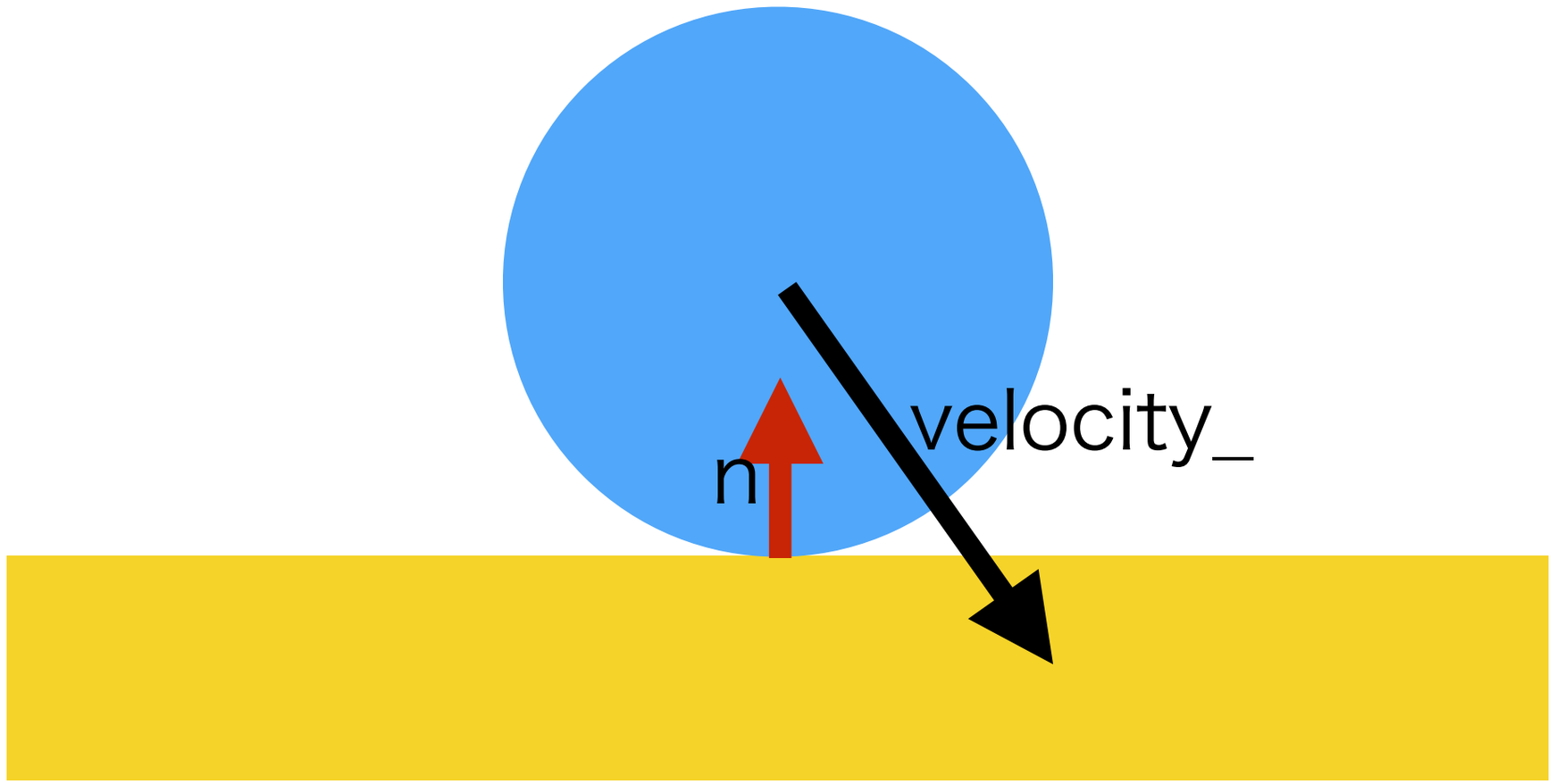


```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {  /* hit for wall or paddle */
      /* calculate the reflecting vecto  */
      Vector3 p = collision.contacts[0].  oint;
      Vector3 n = transform.position -
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

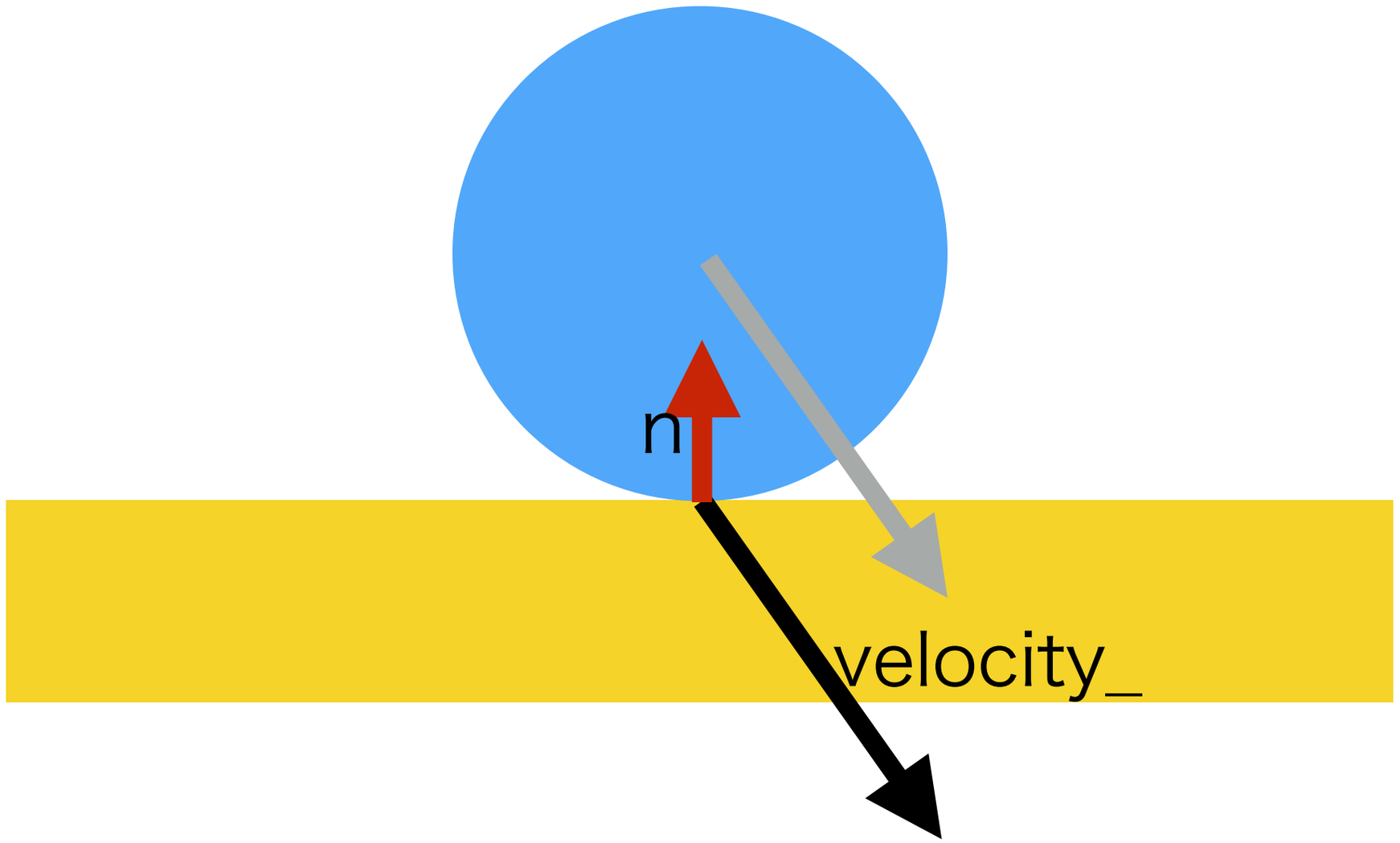
さっきの n と内積を取る

```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) { /* hit for wall or paddle */
      /* calculate the reflecting vector */
      Vector3 p = collision.contacts[0].point;
      Vector3 n = transform.position - p;
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

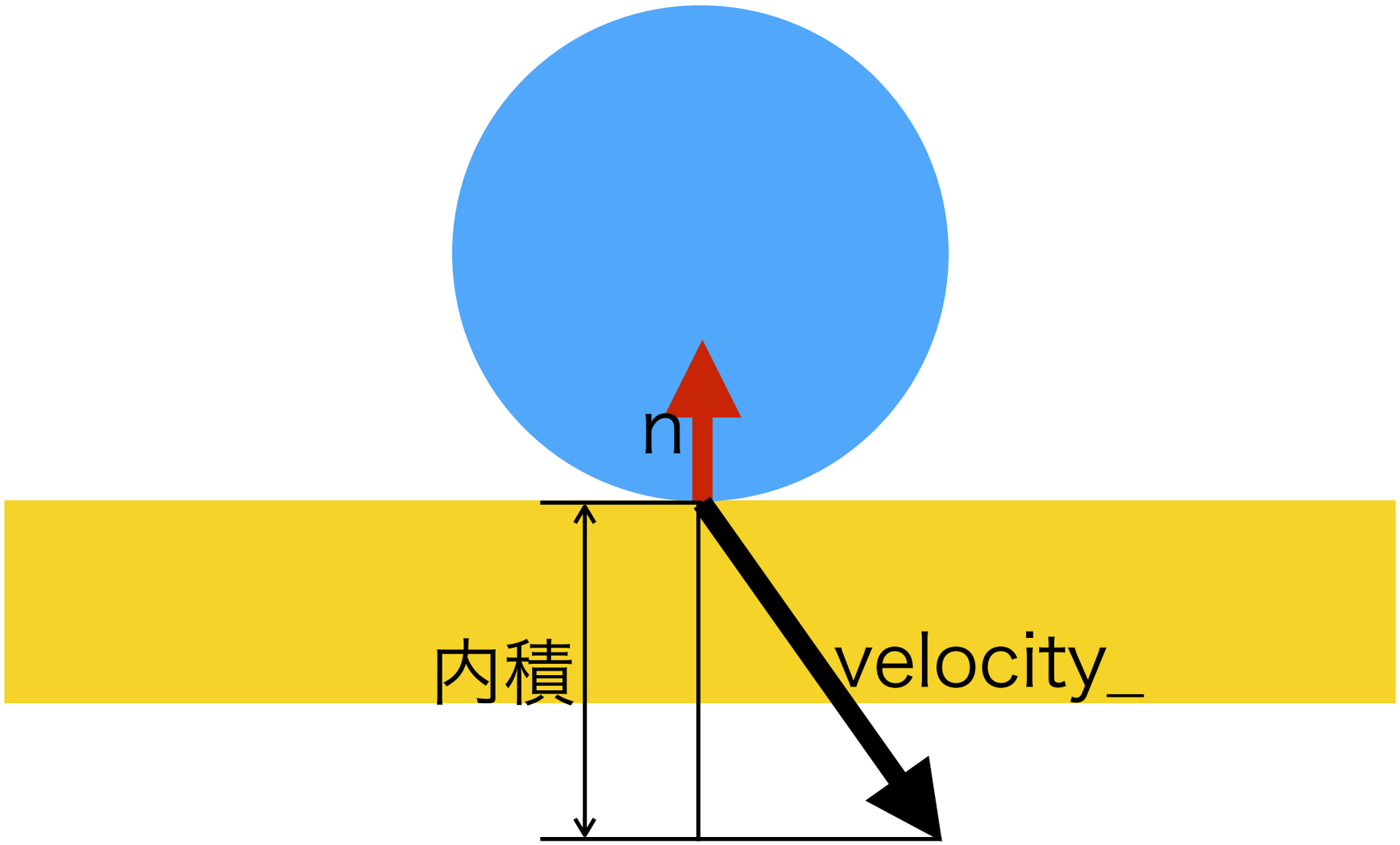
velocity_ と n の内積は？



ベクトルは平行移動し
ても意味は変わらない

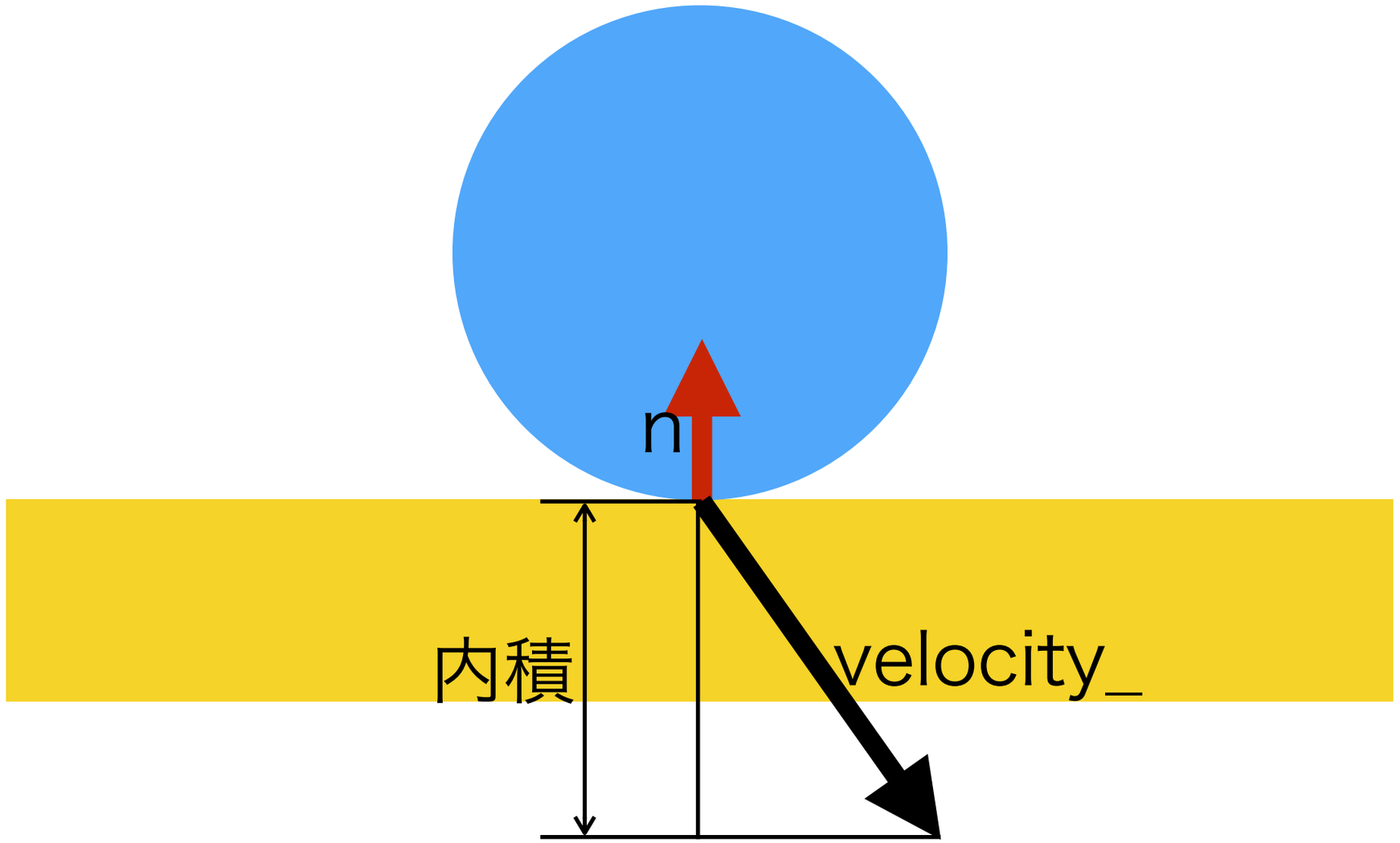


この長さが内積



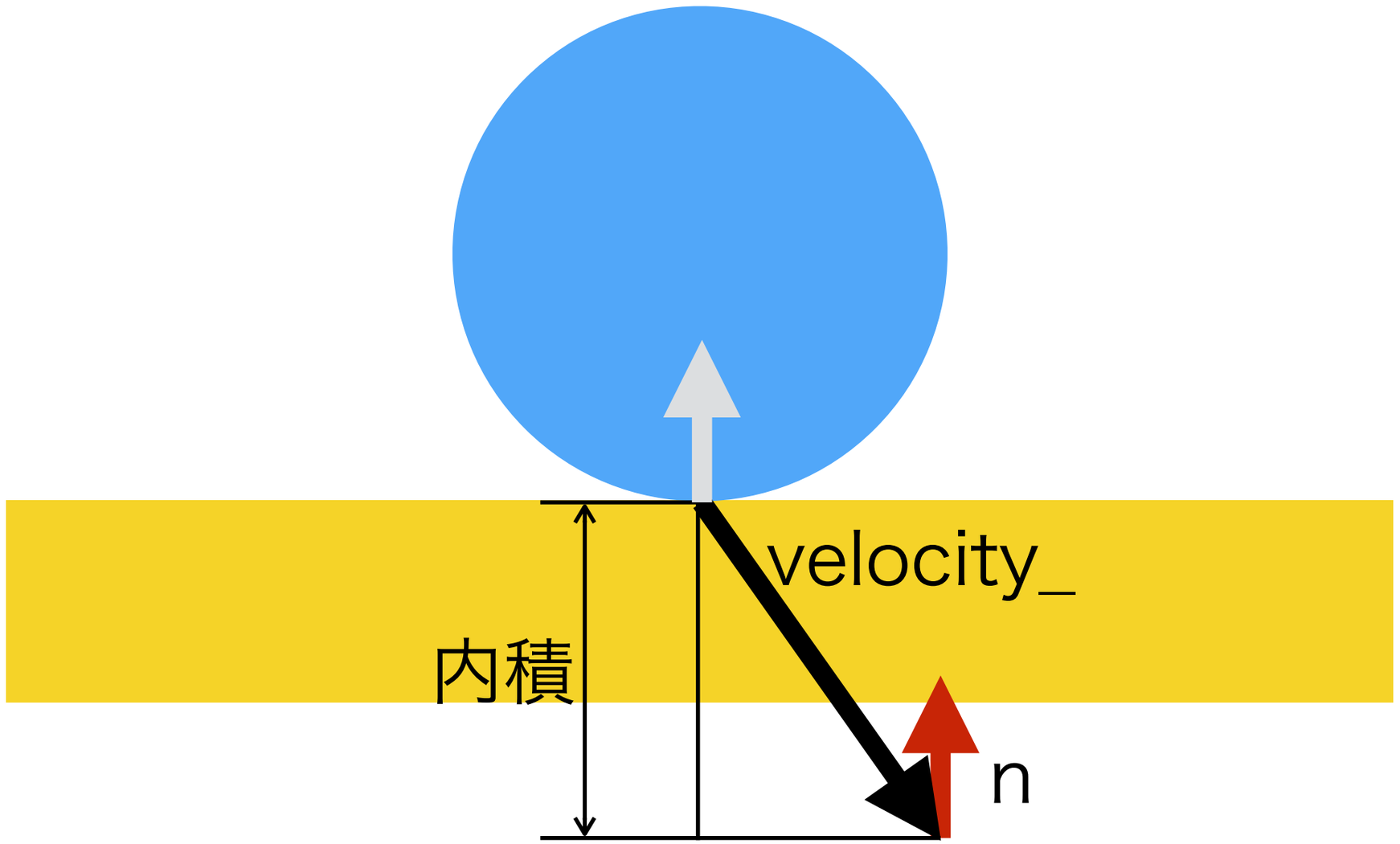
ただし

この図では負の値

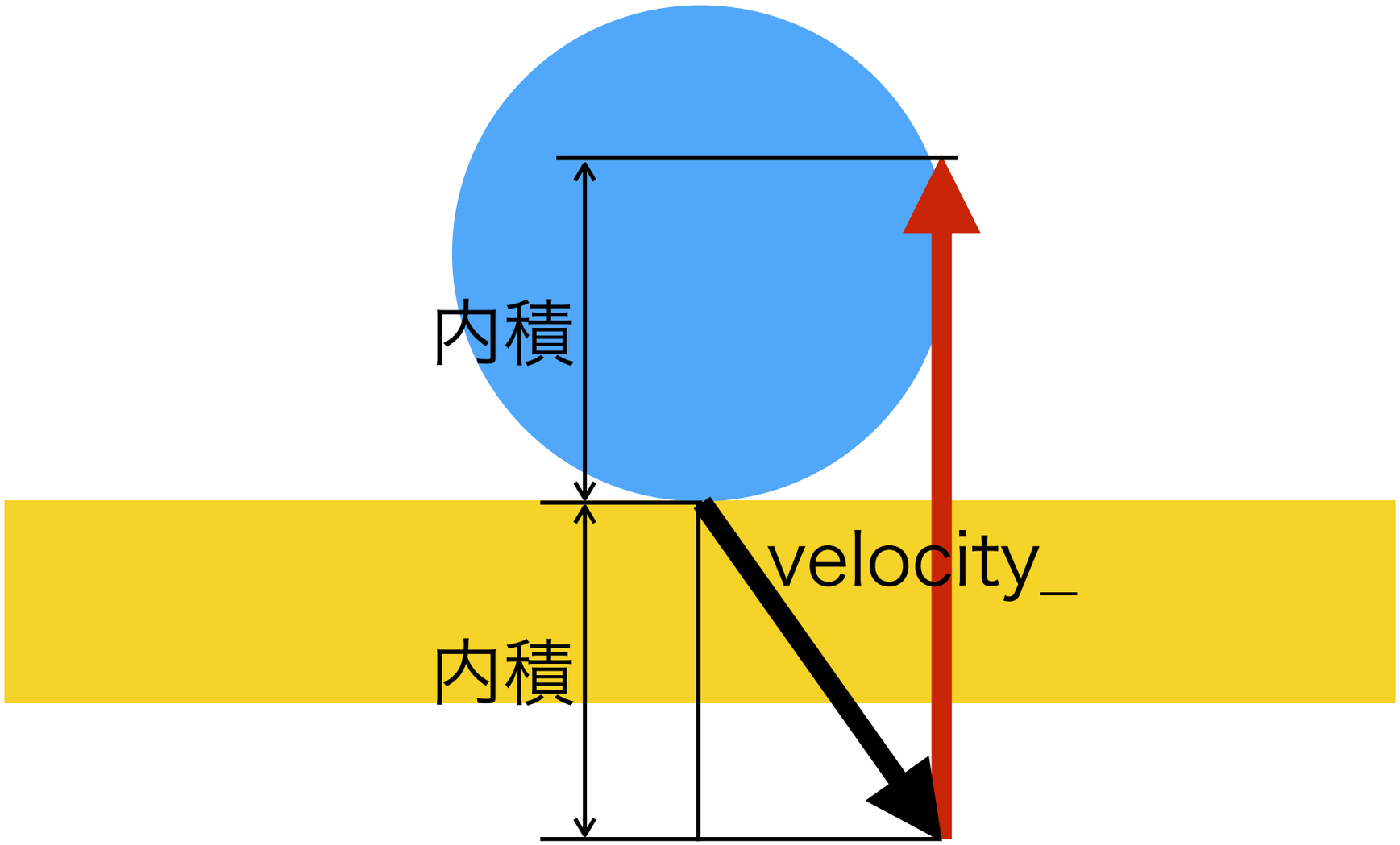


n がここにあると

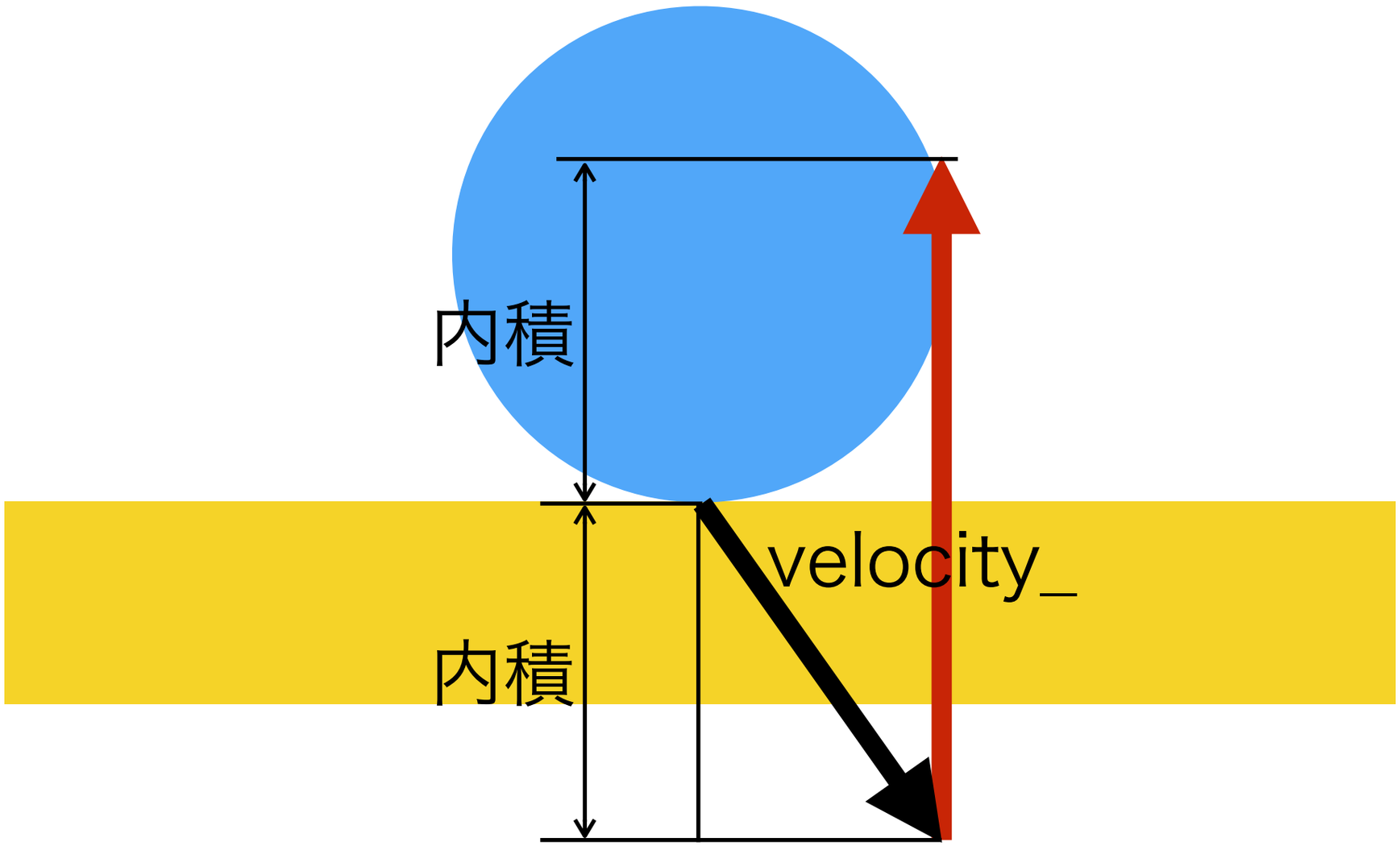
考えて



nを内積の
２倍ぶん伸ばしたい



ただし内積は
負が出ていたので
-2 倍する



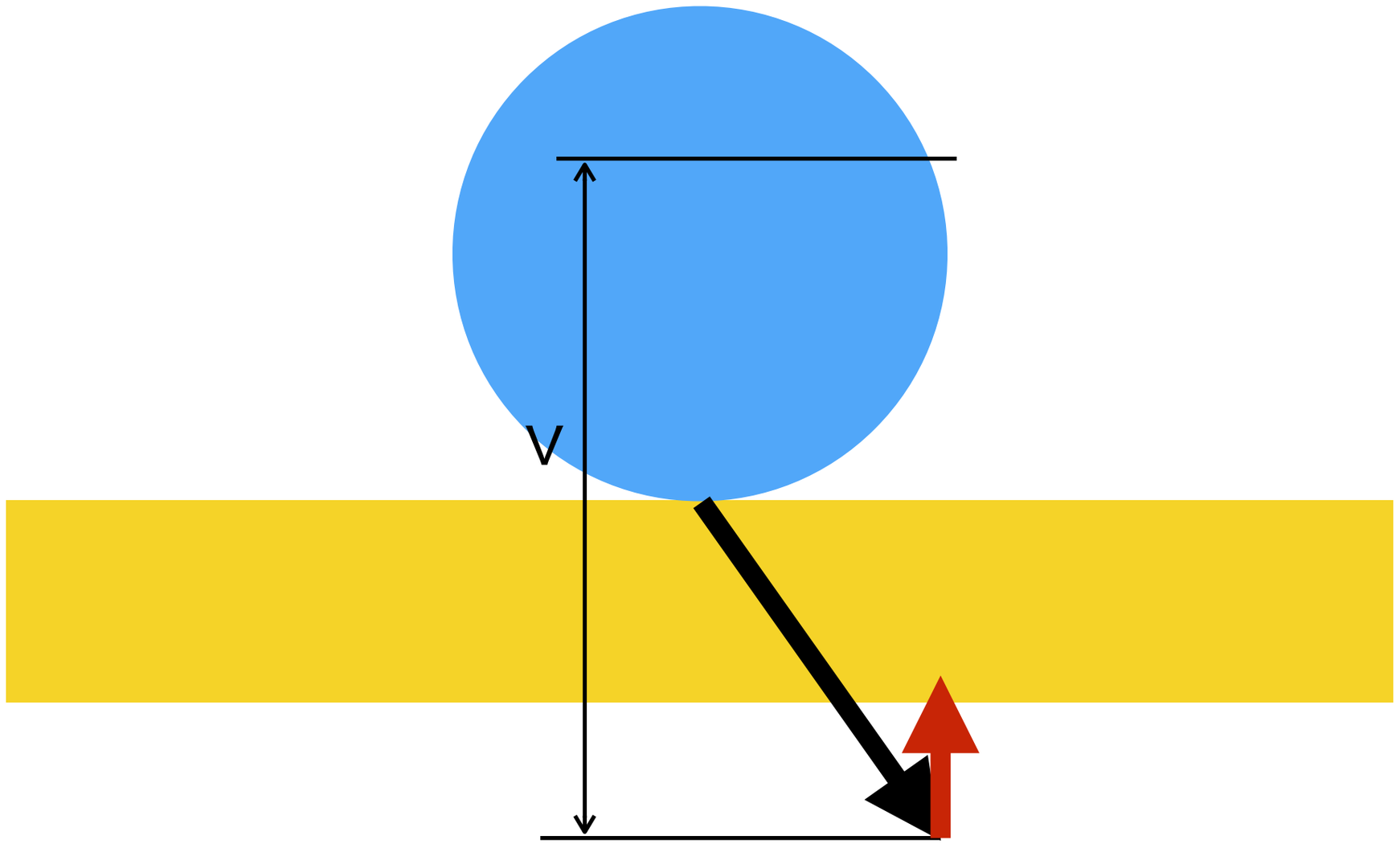
それがこれ

```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
      /* calculate the reflecting vector */
      Vector3 p = collision.contacts[0].point;
      Vector3 n = transform.position - p;
      n.Normalize();
      float v = -2 * Vector3.Dot(velocity_, n);
      var r = n * v;
      velocity_ += r; /* change the velocity vector */
    }
}
```

この v というのは

```
void OnCollisionE  er(Collision collision) {
      if (collisi .contacts.Length > 0) {/* hit for wall or paddle */
         /* calcu te the reflecting vector */
         Vector  = collision.contacts[0].point;
         Vector3  = transform.position - p;
         n.Normalize();
         float v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
      }
   }
```

この長さになる
（ベクトルではない）

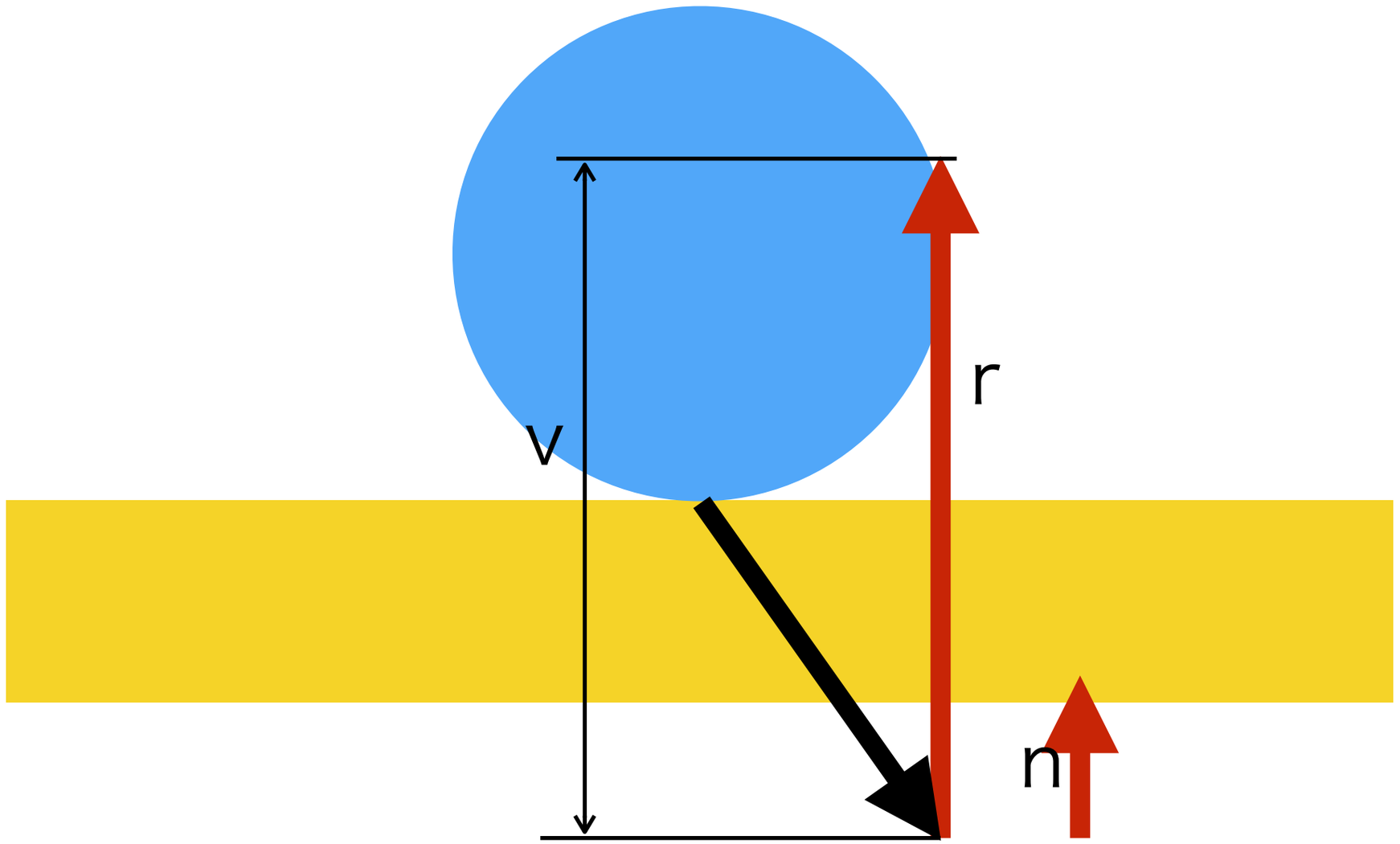


n を v 倍した r というのは

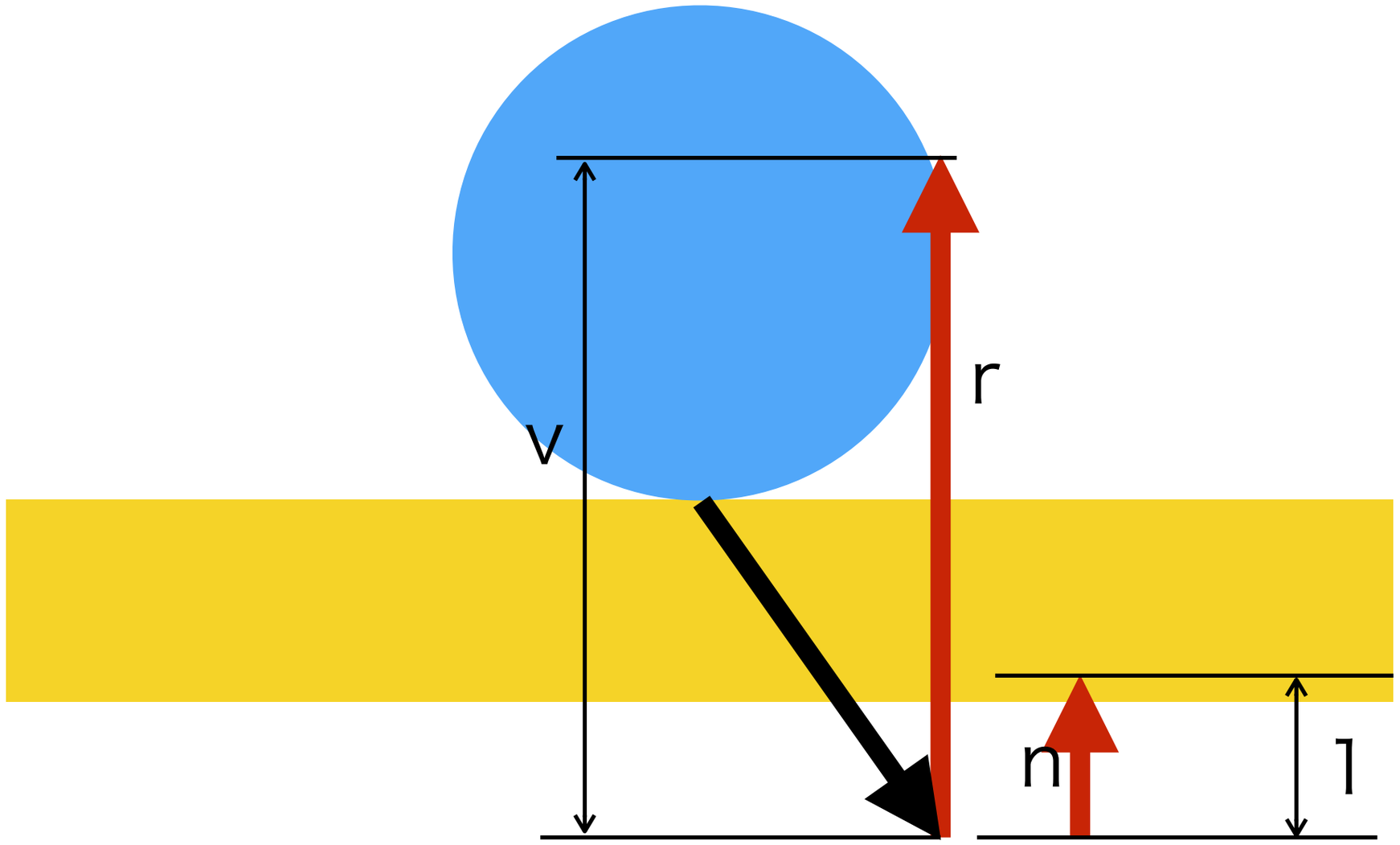
```
void OnCollisi Enter(Collision collision) {
      if (colli ion.contacts.Length > 0) {/* hit for wall or paddle */
         /* ca late the reflecting vector */
         Vecto  p = collision.contacts[0].point;
         Ve  n = transform.position - p;
         n.N ize();
         floa v = -2 * Vector3.Dot(velocity_, n);
         var r = n * v;
         velocity_ += r; /* change the velocity vector */
   }
}
```

これ。

r の長さは v

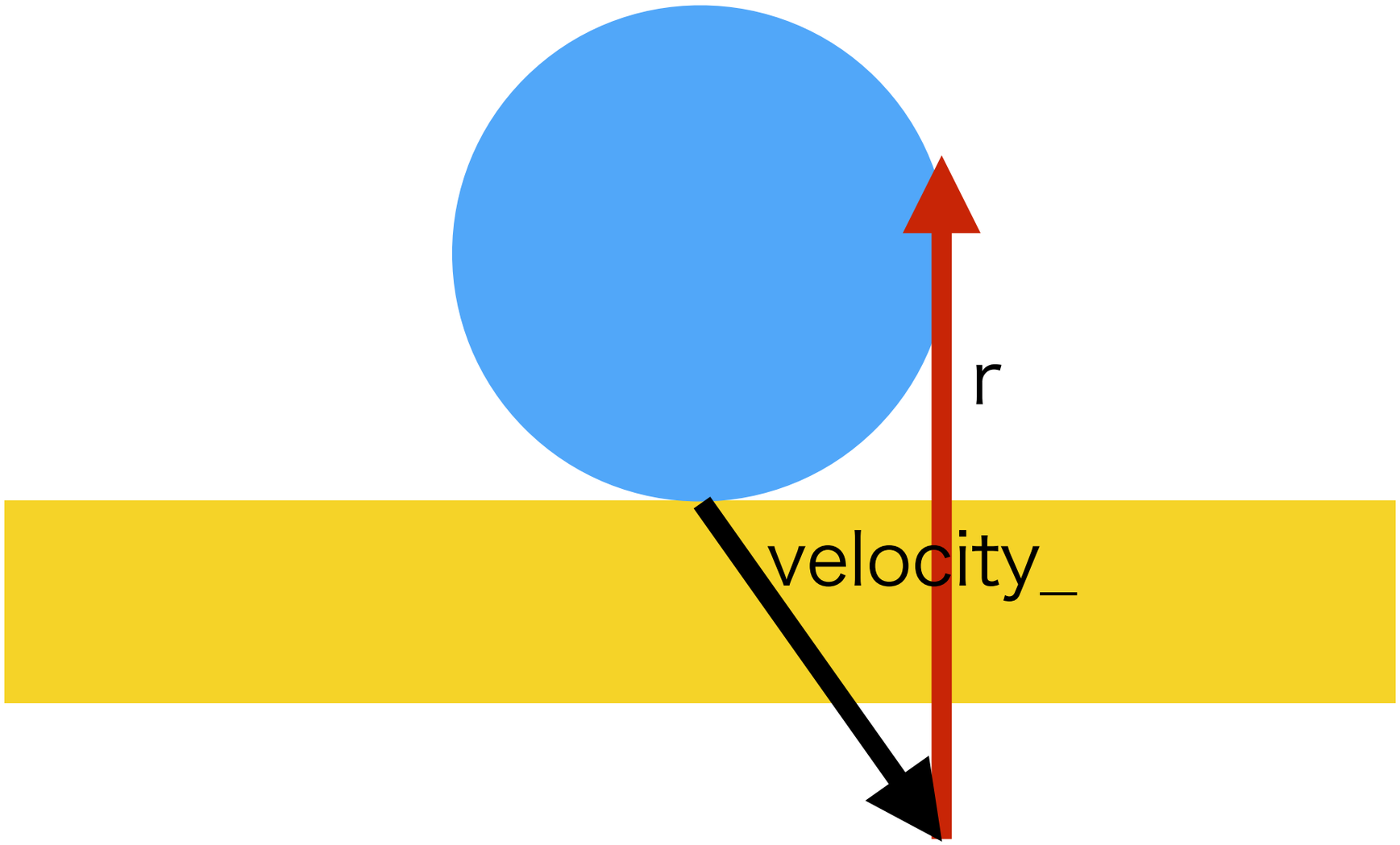


このために
n の長さが 1 で
ある必要があった

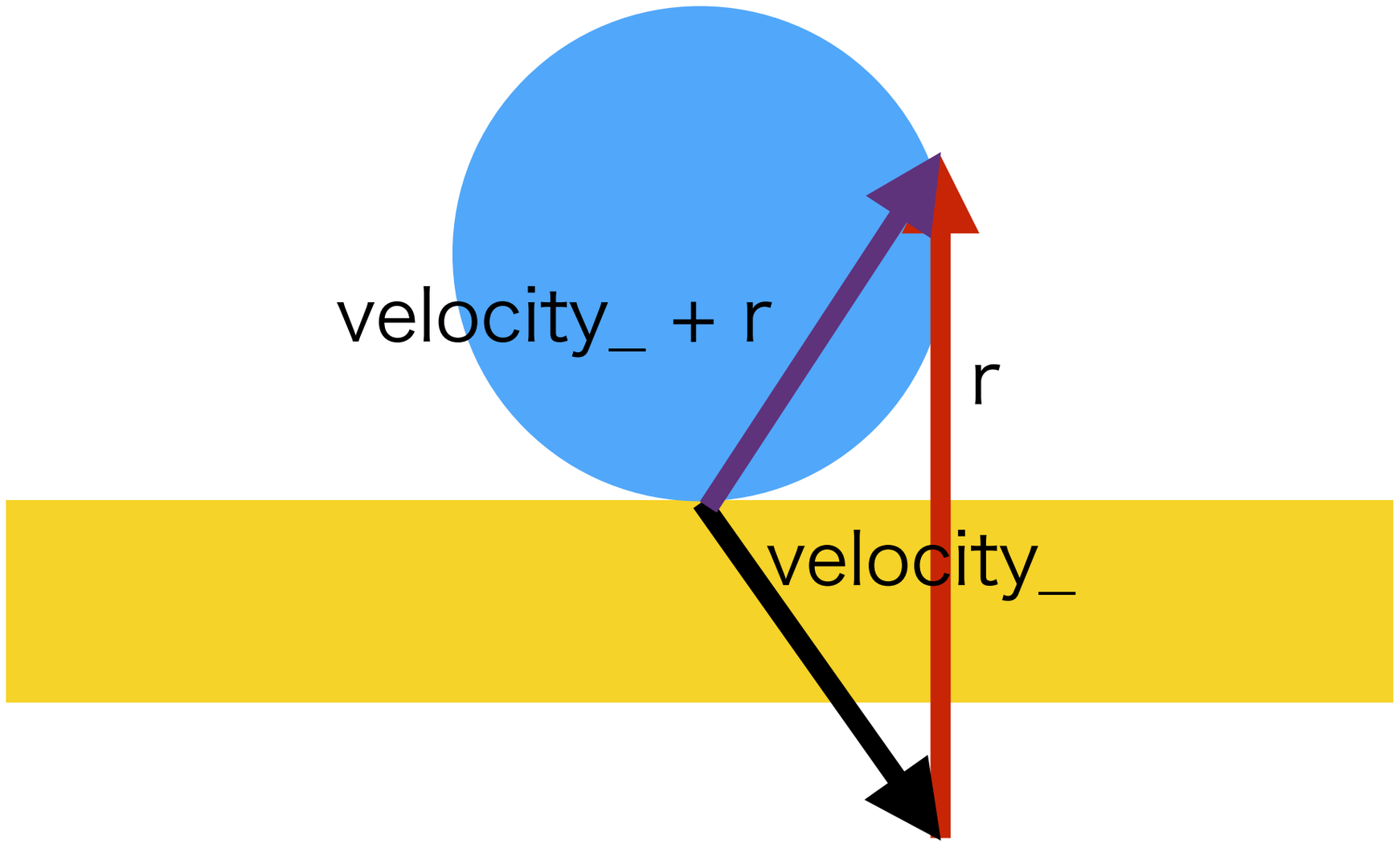


このふたつのベクトル

を足すと



こうなる！



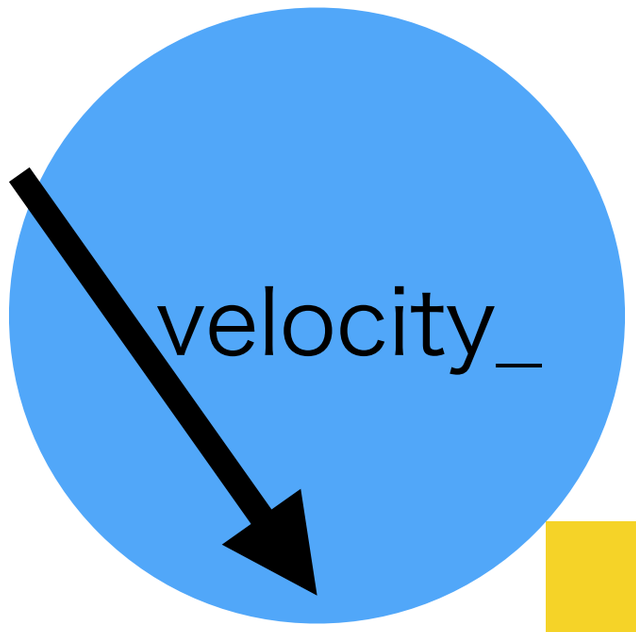
新 velocity_ に
velocity_ + r
を格納する

これがそれ

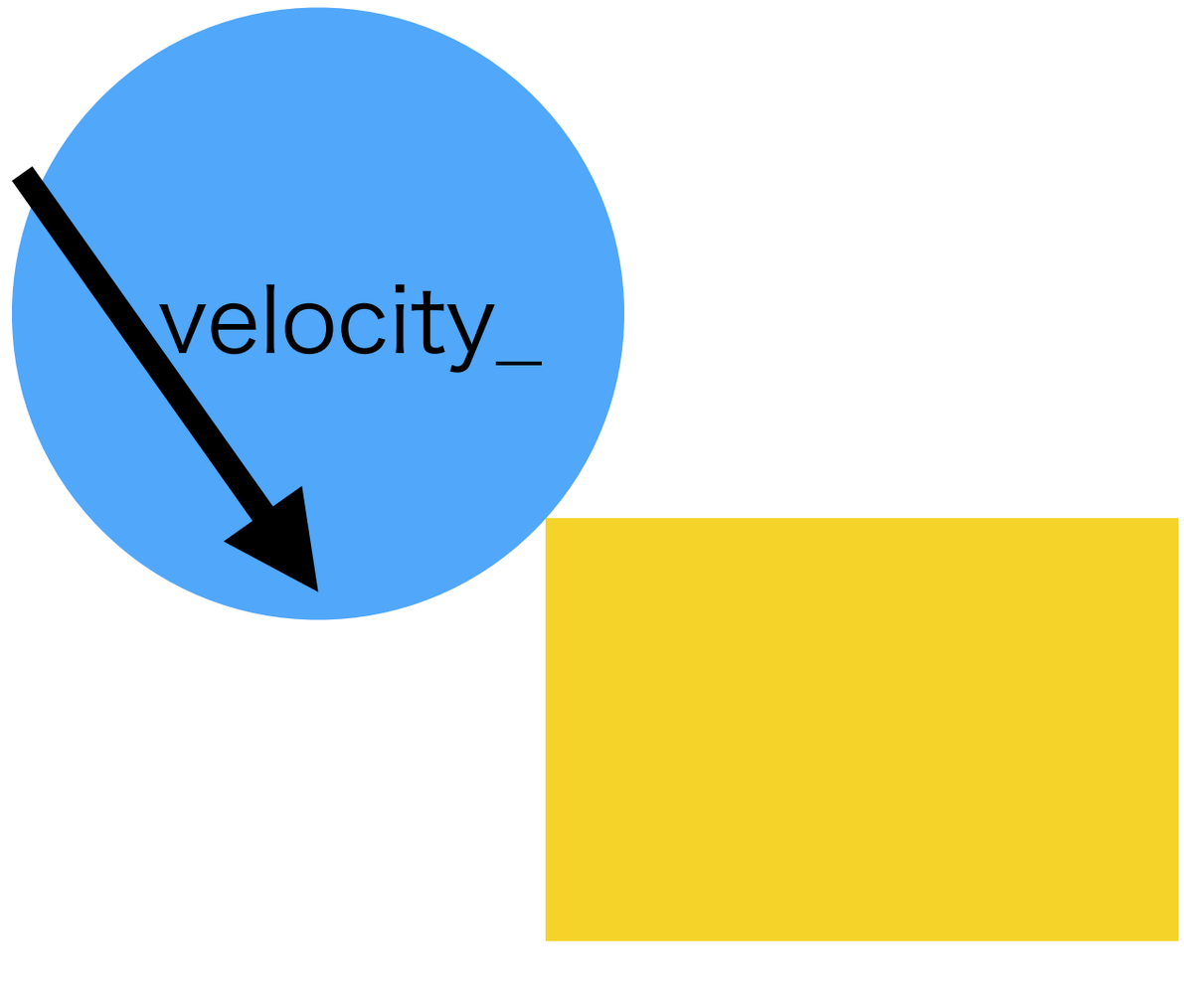
```
void OnCollisionEnter(Collision collision) {
    if (collision.contacts.Length > 0) {/* hit for wall or paddle */
        /* calculate the reflecting vector */
        Vector3 p = collision.contacts[0].point;
        Vector3 n = transform.position - p;
        n.Normalize();
        float v = -2 * Vector3.Dot(velocity_, n);
        var r = n * v;
        velocity_ += r; /* change the velocity vector */
    }
}
```

これで跳ね返ります
めでたしめでたし。

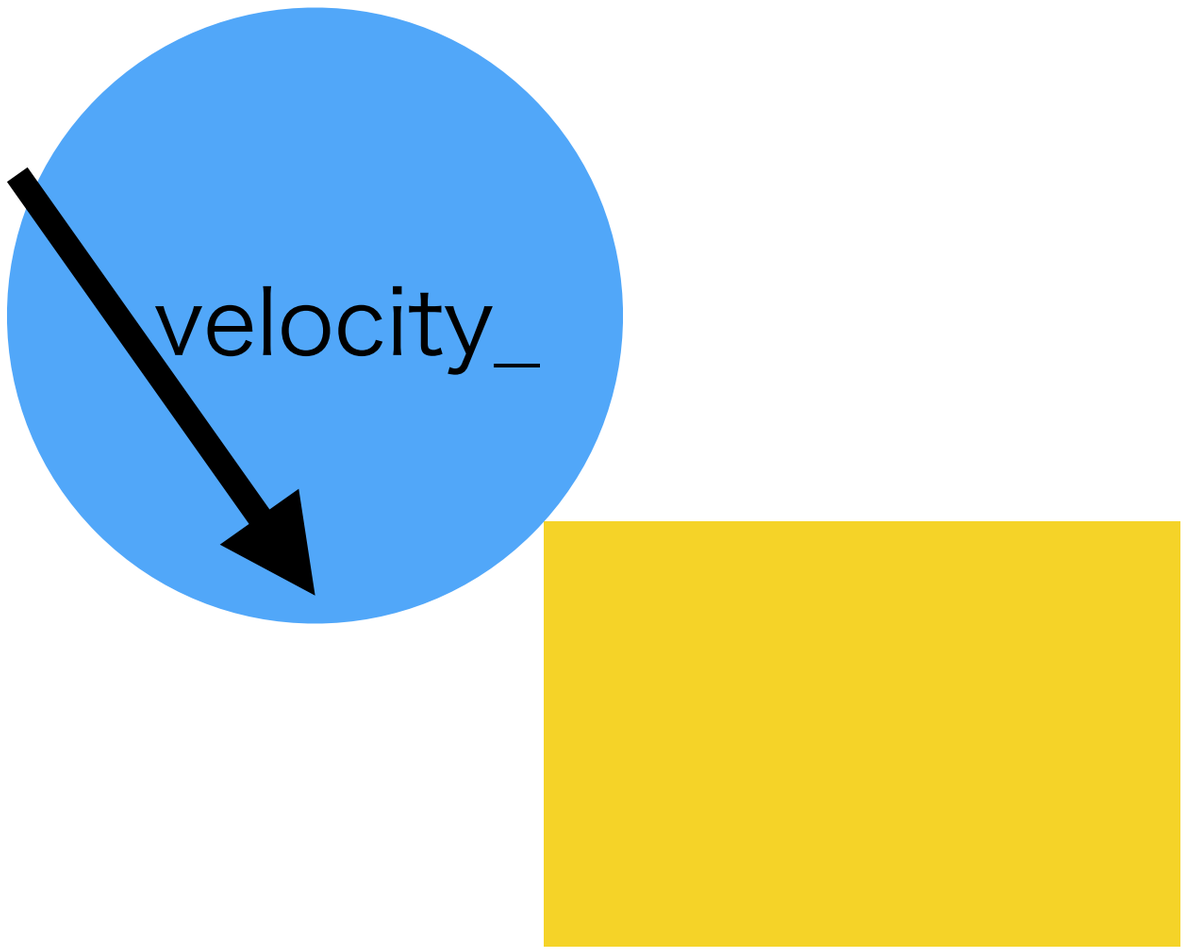
あれ、じゃあ
こういうカドに
ぶつかったらどうなるの？



結論から言うと、
このプログラムのままで
大きな問題はありません



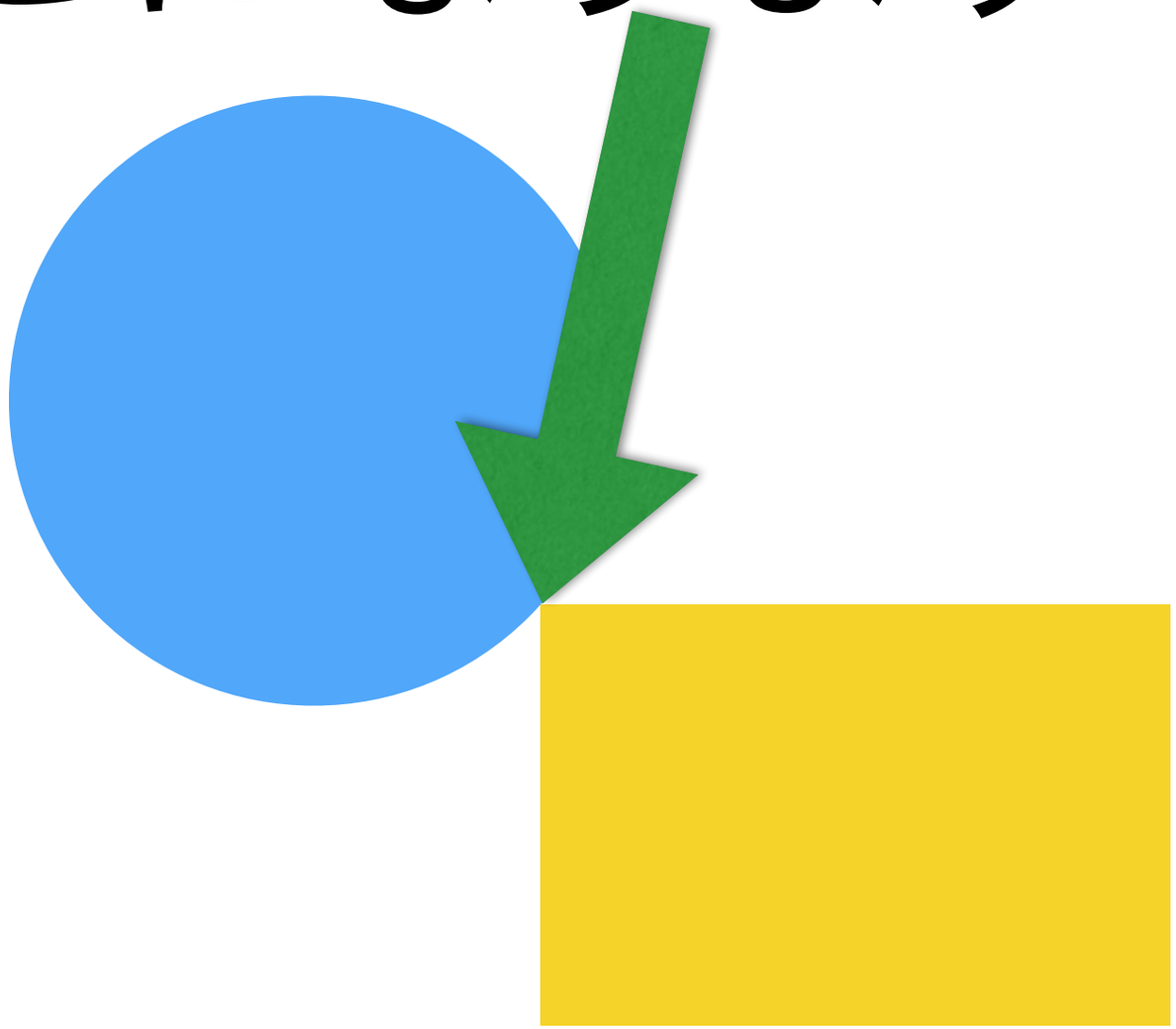
各自、考えてみて
ください



ヒント

```
collision.contacts[0].point;
```

はここになります



おしまい