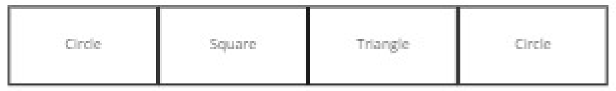
Setup:
Use Logging Queue from previous paper
PEQ seeds should be saved and used for non-PEQ use cases (to guarantee same distribution) (apples to apples)

1. Init vector of shapes
2. Draw (Iterate vector)
3. Update positions
4. Draw (Iterate vector)
5. Goto 3



Method Dispatch
1. Each struct and union implements same method with union using switch/case in every method.
2. Dispatch Method

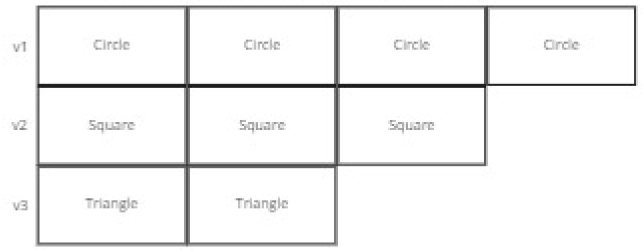
Suitability of this pattern
1. Large degree of variance of object size will waste space.
2.

What other patterns do we want to compare
1. CRTP
2. vanilla C++ virtual
3. Vector of same data structure (baseline)
4. N vectors of static types
   a. 1 vector per type

Testing Scenarios
1. Single random seed chosen as typical use case
2. Many seeds tested and statistical analysis provided

Stats gathered
1. Memory usage (loss of efficiency due to union of varying types)
2. Typical latency stats

Iterate over N vectors with same object composition of above



Use cases
1. Vector of dynamic objects
2. Queue between threads
   a. critical -> non-critical
   b. non-critical -> critical
   c. critical -> critical
   d. non-critical -> non-critical
      i. Really need for this complexity when performance is not a concern?
      ii. Need some further exploration

```
class DescDispatch
{
public:
   template <class T>
   static auto dispatch(T& x) {return x.desc;}
};

class DrawDispatch
{
public:
   template <class T>
   static auto dispatch(T& x) {return x.Draw();}
};

....

std::cout << "DD1 " << c.Dispatch<DescDispatch>() << std::endl;
 std::cout << "DD2 " << s.Dispatch<DescDispatch>() << std::endl;
 std::cout << "DD3 " << t.Dispatch<DescDispatch>() << std::endl;
 std::cout << "DD4 " << o.Dispatch<DescDispatch>() << std::endl;

c.Dispatch<DrawDispatch>();
```

```
template <class Dispatcher>
auto Dispatch()
{
   switch(m_st)
   {
      case ShapeType::Square:
      {
         auto &x = get_by_type<Square>();
         return Dispatcher::dispatch(x);
         break;
      }
      case ShapeType::Triangle:
      {
         auto &x = get_by_type<Triangle>();
         return Dispatcher::dispatch(x);
         break;
      }
      case ShapeType::Circle:
      {
         auto &x = get_by_type<Circle>();
         return Dispatcher::dispatch(x);
         break;
      }
      case ShapeType::Octagon:
      {
         auto &x = get_by_type<Octagon>();
         return Dispatcher::dispatch(x);
         break;
      }
      default:
         std::cout << "Unknown Shape" << std::endl;
         break;
   }
```