Auto

Yossi Moalem

Auto

- Given the compiler knows the type, why should we write it?
- Uses template type deduction

What is the deduced type

```
int i = 5;
int & rl = i;
auto a = rl;
```

```
const int i = 5;
auto a = i;
```

```
auto i1 = 5;
auto i1 (5);
auto i3 {5};
auto i4 = {5}
```

```
auto i = -0xFFFFFF
```

See C++ weekly SE (48.5) for details

Auto, use cases:

Presentations

Auto is shorter

Easier on the slides!

Magical type

```
auto x = getMagical();
setMagic(x);
```

Possible compromise:

```
auto magicalPacket = getMagicalPacket()
```

Initialize just for the sake of auto

```
auto x = 0;
auto y = 0;
if (useRelative()) {
    x = relativeX();
    y = relativeY();
} else {
    x = absoluteX();
    y = absoluteY();
}
```

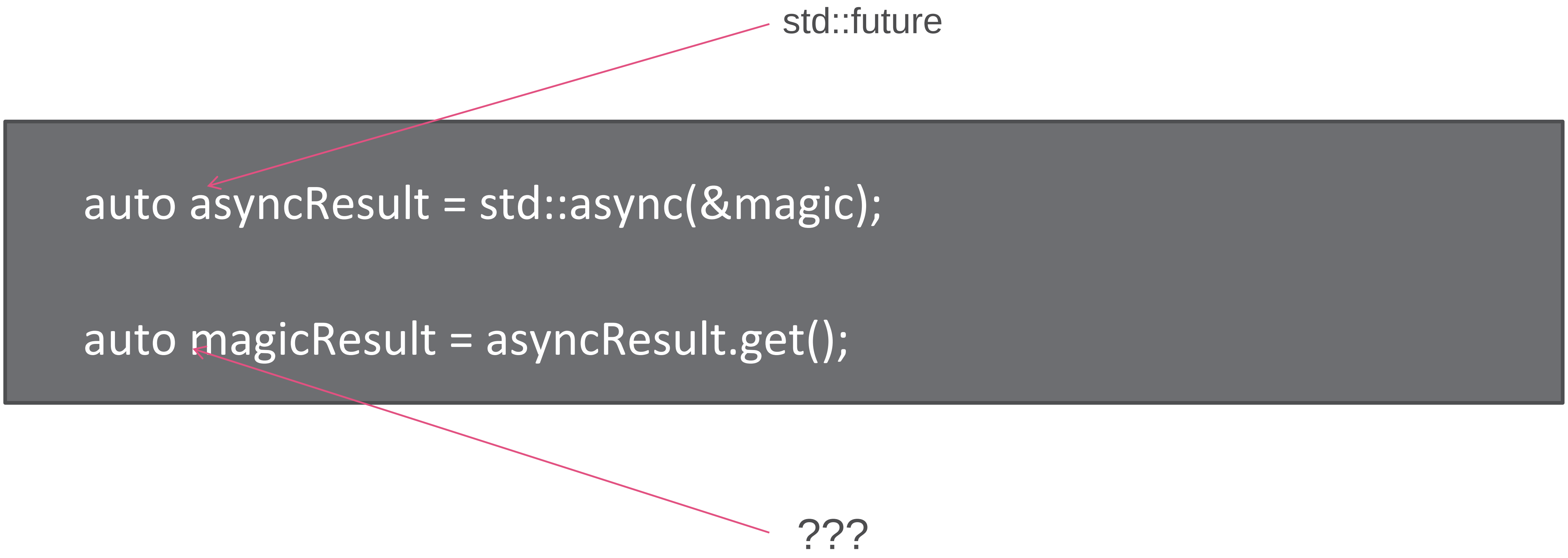
I prefer uninitialized variable then lie

- Compiler may detect
- Static analyzer may detect
- Memory analyzer may detect

Use auto for the sake of using auto

```
auto l = ( long ) 5;
```

Async result



std::future

```
auto asyncResult = std::async(&magic);

auto magicResult = asyncResult.get();
```

???

Prevent repetition

```
std::shared_ptr<ppt::default::connectionHandler> connectionHandler =
std::make_shared<ppt::default::connectionHandler>();
```

```
auto handler = std::make_shared<ppt::default::connectionHandler>();
```

Complex, non interesting types

```
std::function<int(int, int)> operator1 =   [](int x, int y) { … };

auto operator2 = [](int x, int y) {… };
```

- Operator1 is also slower
  - Another level of indirection
- And allocates memory on the heap

Long, easy to know types

```
std::map<int, std::string>::iterator currentItem=  itemMap.begin();
```

Vs.

```
auto currentItem= itemMap.begin();
```

Or actually,

```
for (auto const & currentItem: items )
```

Prevent needless copies

```
auto map = std::map<int,float>();
map.emplace(12, 42.5);


std::pair<int,float> const& firstPairWillCopy = *map.begin();
auto const & firstPairNoCopy = *map.begin();
```

Proxy object

```
void foo(bool f);
std::vector<bool> flags;
//initialize the vector
std::vector<bool> getFlags() {return flags;}

auto flag = getFlags()[0];
foo(flag); //undefined behavior
```

Case study

```
auto profit = getProfit(paid, cost);

if (profit == 0) …
```

```
auto getProfit( int paid, int cost) {
  auto revenue = paid – cost;
  return reduceVAT(revenue);
}
```

```
auto reduceVAT( int revenue) {
  auto VAT = getVAT();
    return (revenue *(100 – VAT)) / 100;
}
```

```
auto getVAT () {
      return 17.0;
}
```

Spot the bug

```
void doSomethingOnLastInstance (std::vector<Bar> & vec ) {
    for (auto i = vec.size() - 1; i => 0; --i ) {
            auto & currentElement = vec[i];
            if (currentElement.shouldDoSomething() ) {
                    currentElement.doSomething();
                    break;
              }
        }
}
```