# Полиморфизм. Виртуальные функции

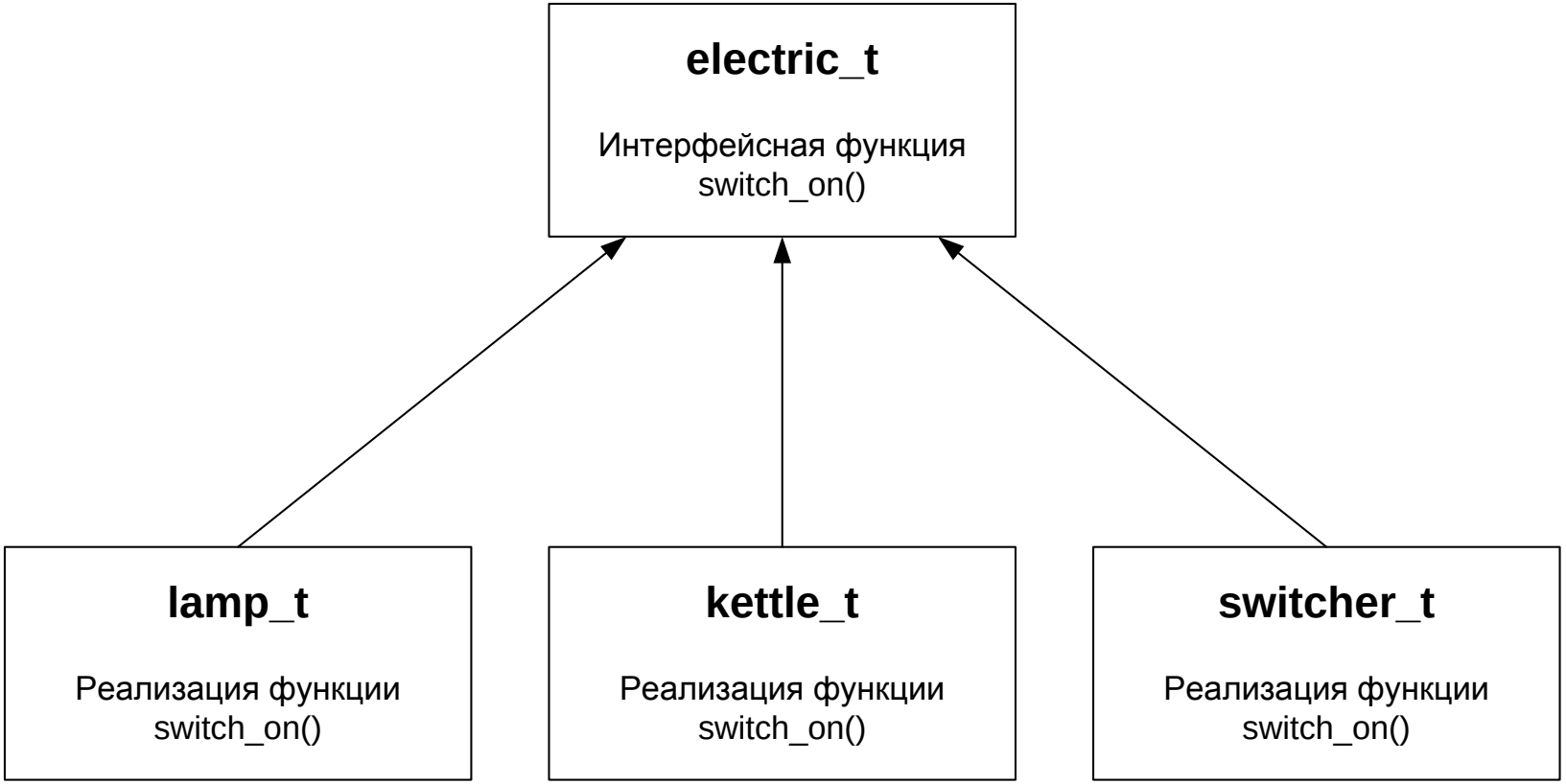
Лекция 9

# Обработка типизированных данных в процедурных языках

```
//! Общий тип записи.
struct electric_t {
   //! Поле, определяющее
   //! тип записи.
   int m_type;
[…]
};

//! Тип лампы.
const gc_lamp = 1;
//! Тип чайника.
const gc_kettle = 2;
//! Тип переключателя.
const gc_switcher = 3;
```

```
//! Процедура запуска устройства.
int
switch_on( electric_t & device ) {

   switch( device.m_type ) {
      case gc_lamp:
         […]
      case gc_kettle:
         […]
      case gc_switcher:
         […]
   };
}
```

# Построение ООП иерархии

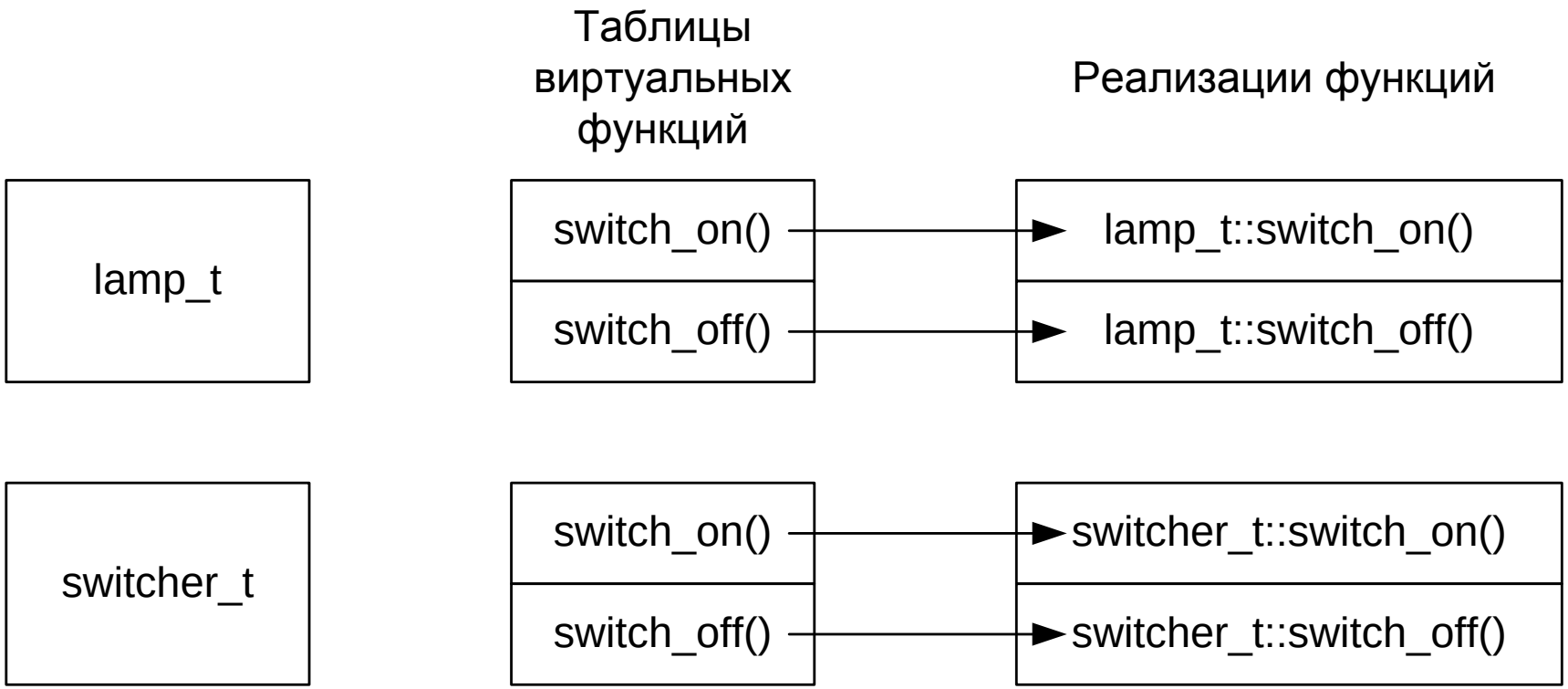


# Обработка типизированных данных в ООП языках

```
// Общий суперкласс.
class electric_t {
   // Объявление интерфейсной функции.
   virtual int
   switch_on() = 0;
};

// Подкласс лампы.
class lamp_t: public electric_t {
   // Объявление наличия
   // функции реализации.
   virtual int
   switch_on();
};

 // Реализация
 // функции в подклассах.
 int
 lamp_t::switch_on() {
    // Реализация для lamp_t.
    [..]
 }
```

```
// Использование функции
// в программе.

void
func( electric_t * device )
{
   device->switch_on();
}

int
main()
{
   lamp_t lamp;

   func( &lamp );
}
```

# Виртуальные функции

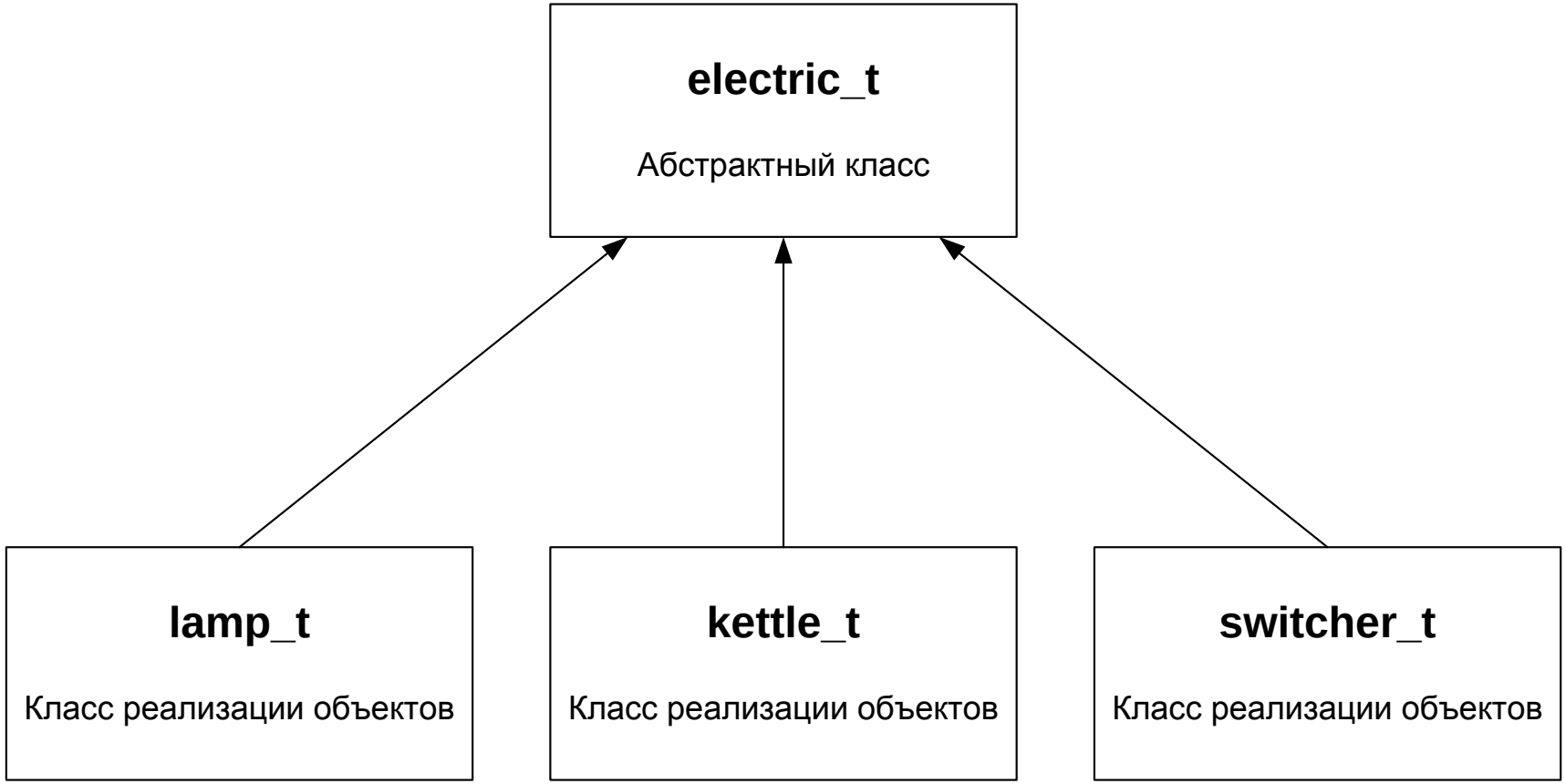


# Виртуальные функции

```
// Общий суперкласс.
class electric_t {
   public:
      // Объявление интерфейсной
      // функции.
      virtual void
      switch_on();
};

// Подкласс лампы.
class lamp_t: public electric_t {
   public:
      // Объявление наличия
      // функции реализации.
      virtual void
      switch_on();
};
```

```
// Реализация функции.
void
electric_t::switch_on() {
   m_is_on = true;
};

// Реализация функции.
void
lamp_t::switch_on() {
   if ( m_voltage > c_max ) {
      electric_t::switch_on();
   }
};
```

# Абстрактные и конкретные классы



# Чисто виртуальные функции

```
class electric_t {
   public:
      //! Включить устройство.
      virtual void
      switch_on() = 0;

      //! Выключить устройство.
      virtual void
      switch_off() = 0;

      //! Состояние устройства как строка.
      virtual std::string
      to_string() = 0;
};
```

# Определение типа объекта

```
class base_t {
[…]
};

class derived_t : public base_t {
[…]
};

void
detect_derived( base_t * p ) {

   if ( dynamic_cast<derived_t*>(p) )
      std::cout << "is derived class" << std::endl;
   else
      std::cout << "isn't derived class" << std::endl;
}
```

```
int main()
{
   derived_t a;
   base_t b;

   detect_derived( &a );
   detect_derived( &b );

   return 0;
}
```

```
is derived class
isn't derived class
```

# Определение типа объекта

```
class base_t {
   virtual std::string
   type() const {
      return “base”;
   }
};

class derived_t : public base_t {
   virtual std::string
   type() const {
      return “derived”;
   }
};

void
detect_derived( base_t * p ) {
   if ( p->type() == “derived” )
      std::cout << "is derived class" << std::endl;
   else
      std::cout << "isn't derived class" << std::endl;
}
```

```
int main()
{
   derived_t a;
   base_t b;

   detect_derived( &a );
   detect_derived( &b );

   return 0;
}
```

```
is derived class
isn't derived class
```

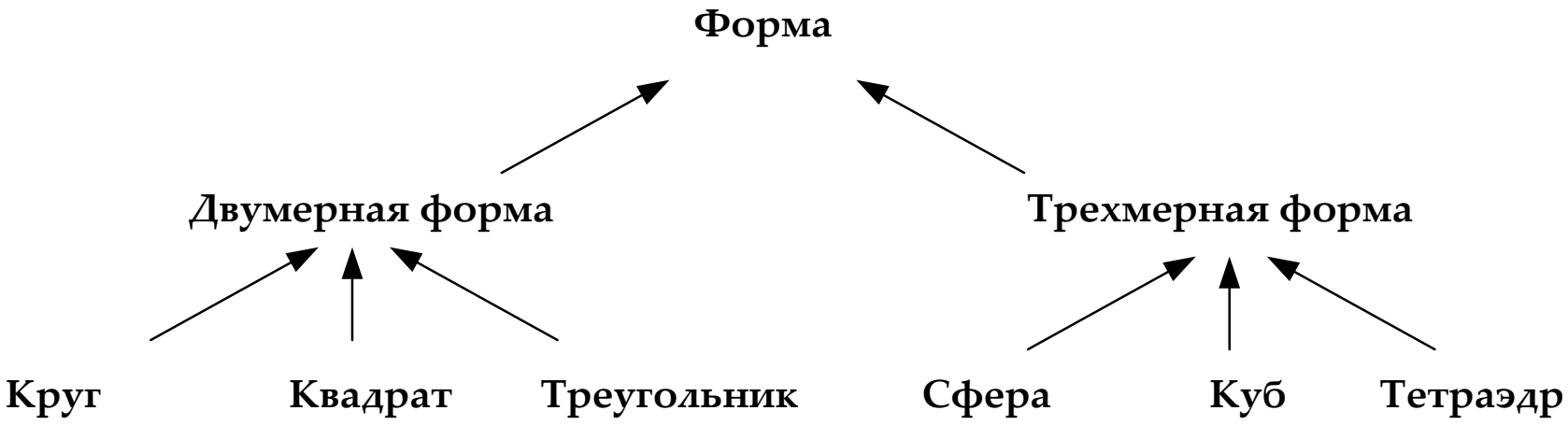
# Виртуальные деструкторы

```
// Общий суперкласс.
class electric_t
{
   public:
      virtual
      ~electric_t() {
         std::cout <<
            "electric is destroyed" << std::endl;
      }
};

// Подкласс лампы.
class lamp_t
{
   public:
      virtual
      ~lamp_t() {
         std::cout <<
            "lamp is destroyed" << std::endl;
      }
};
```

```
int main()
{
   lamp_t a;
   return 0;
};
```

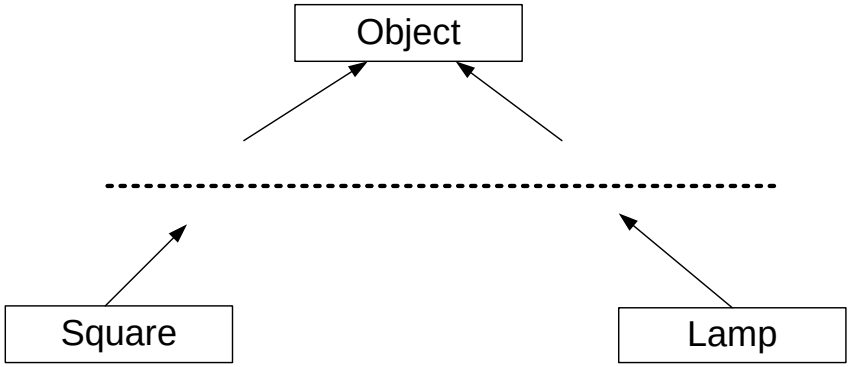
```
lamp is destroyed
electric is destroyed
```

# Выполнение типового действия

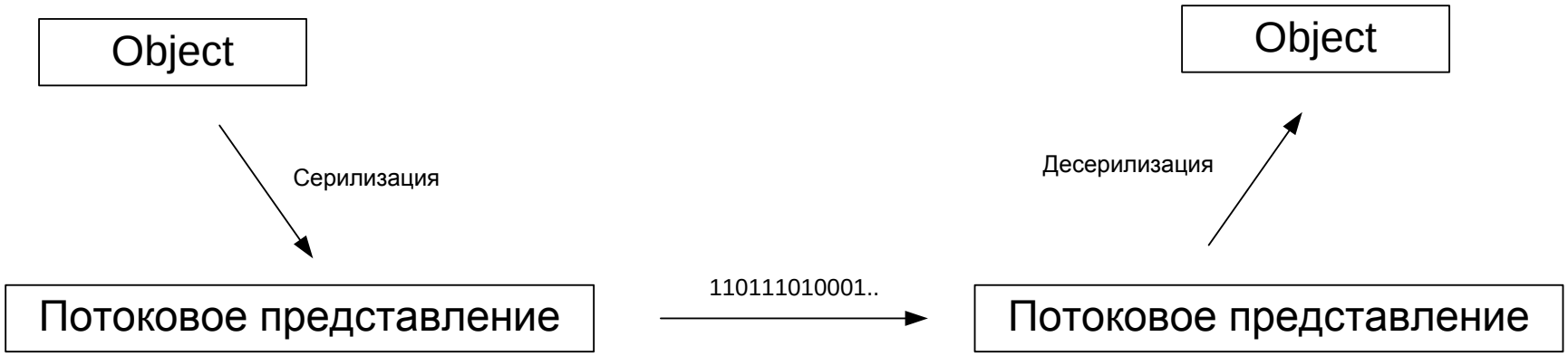


# Преобразование в строку

```
// Базовый класс для всех объектов.
class object_t
{
    public:
       // Преобразование объекта в строку.
       virtual std::string
       to_string() const;
};

class square_t : public object_t
{
   public:
      virtual std::string
      to_string() const
      {
         std::strstream result;
         result <<
            "Object square. Edge of a square is " <<
            m_a << " m" << std::ends;
         return result.str();
      }

    private:
       // Ребро квадрата.
       float m_a;
};
```



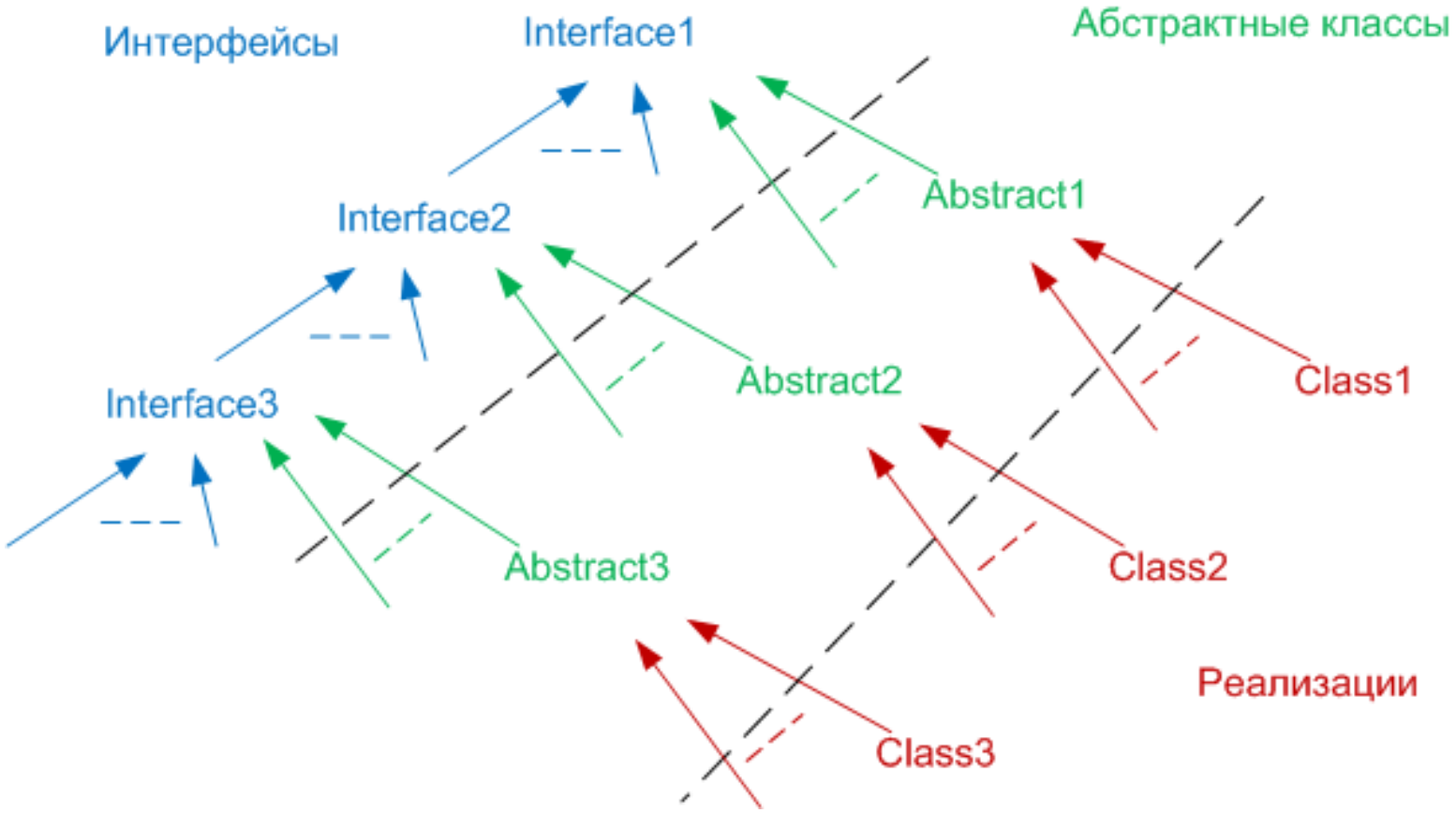
# Сериализация и десериализация



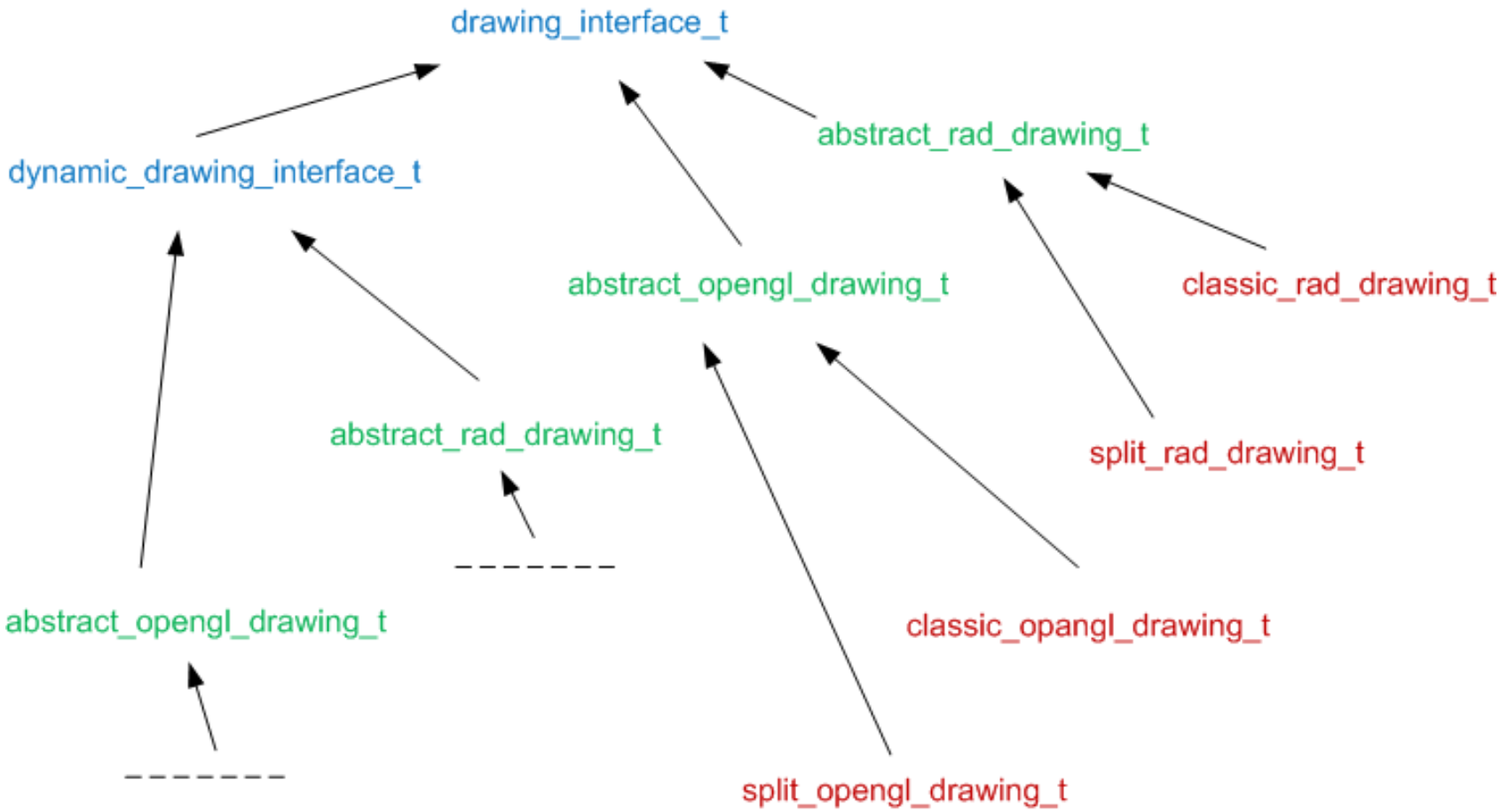
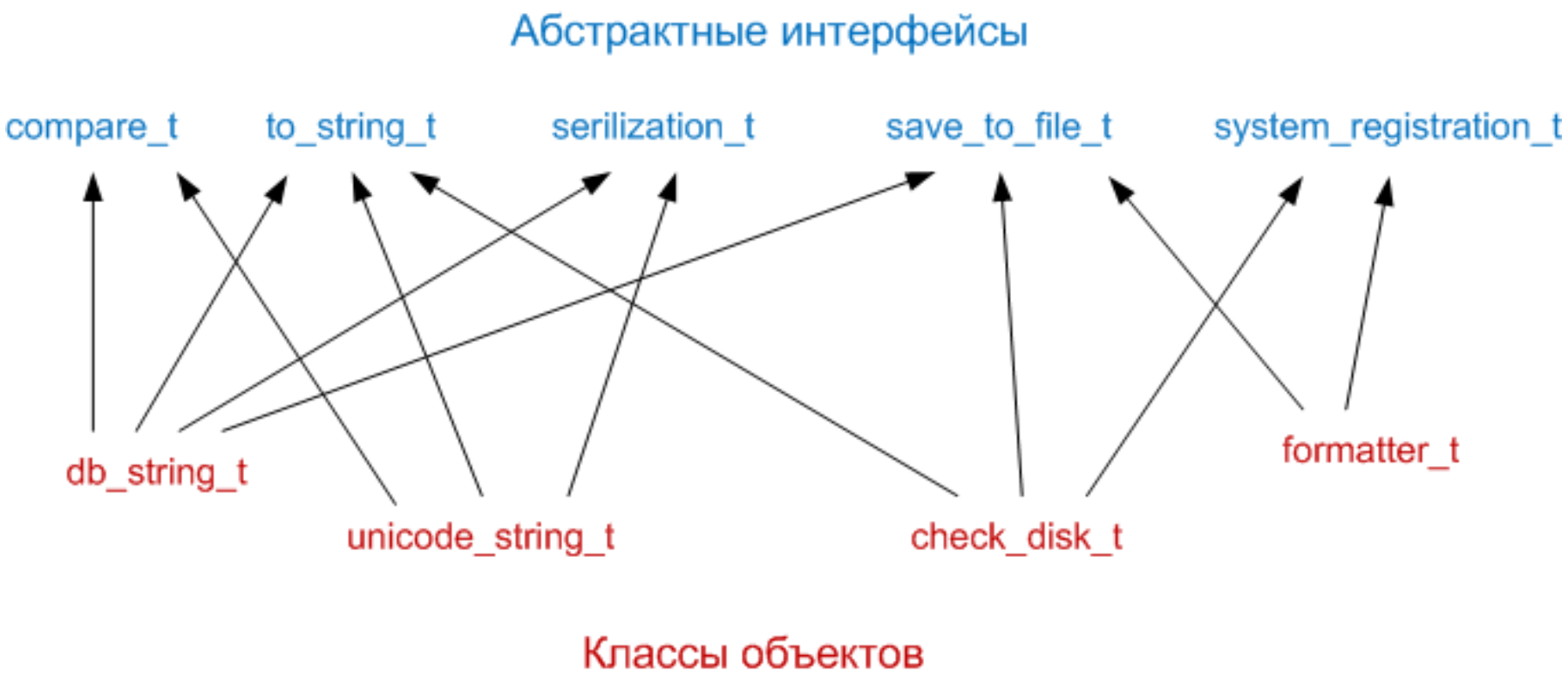
# Множественный полиморфизм

```
class electric_t {

   public:
      //! Соединить устройство с другим.
      virtual void
      connect( electric_t & electric );
[…]
};

void
func( electric_t * p ) {

[…]

   p->connect( device );

}

int main() {

   electric_t * device;

   […]

   func( device );

}
```

# Иерархия интерфейсов



# Иерархия интерфейсов



# Множественное наследование. Распределение интерфейсов.