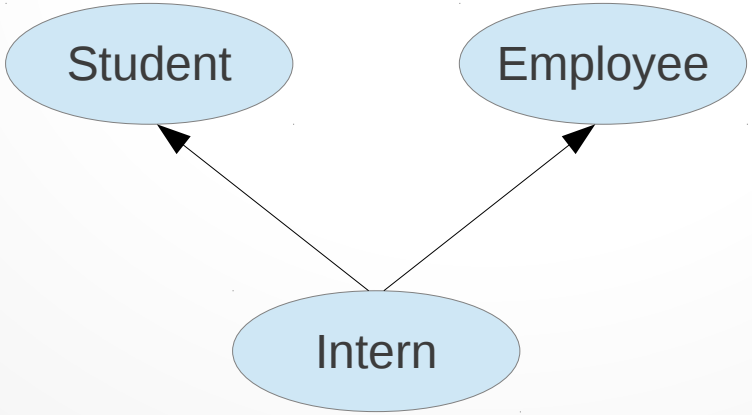
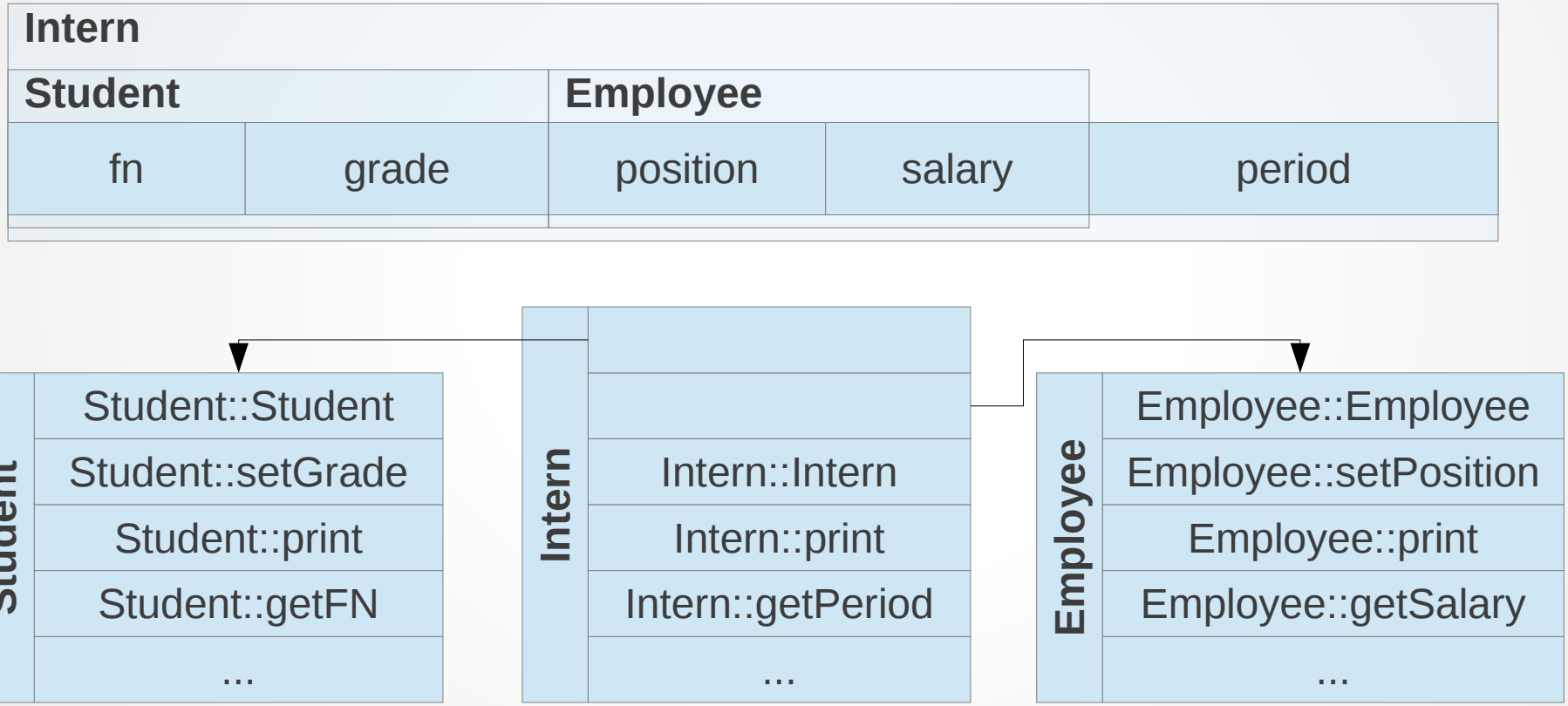
Множествено наследяване

Какво е множествено наследяване?

- Производен клас с повече от един основен клас
- Производният клас комбинира характеристиките и поведението на всичките си основни класове
- Пример:
  class Intern : public Student, public Employee { ... };



Физическо представяне



Жизнен цикъл на обект от производен клас

- За обекта са заделя памет (на стека или в динамичната памет)
- Извиква се конструктор, който
  - извиква конструкторите на основните класове в реда на наследяване
  - извиква конструкторите на всички съдържани обекти
- ... (работа с обекта)
- Достига се края на областта на действие на обекта
- Извиква се деструктор, който
  - извиква деструкторите на всички съдържани обекти
  - извиква деструкторите на всички основни класове в ред обратен на реда на наследяване
- Паметта на обекта се освобождава

Конструктори

- Конструкторите на производния клас трябва да указват как се конструират всяка една от наследените части
  - ако за някой от основните класове не е указан кой конструктор да се извика, тогава се извика този по подразбиране
- Пример:

```
Intern::Intern(int _fn, double _grade,
                    char const* _position, double _salary, int _period)
 : Student(_fn, _grade), Employee(_position, _salary),
period(_period)
```

Деструктори

- Деструкторите на основните класове се викат автоматично

Предефинирани функции

- Предефинираните функции могат да извикат съответна функция във всеки един от основните класове
- Пример:

```
void Intern::print() const {
  Student::print();
  Employee::print();
  cout << „Период на стажа: „ << period << endl;
}
```

Голямата четворка

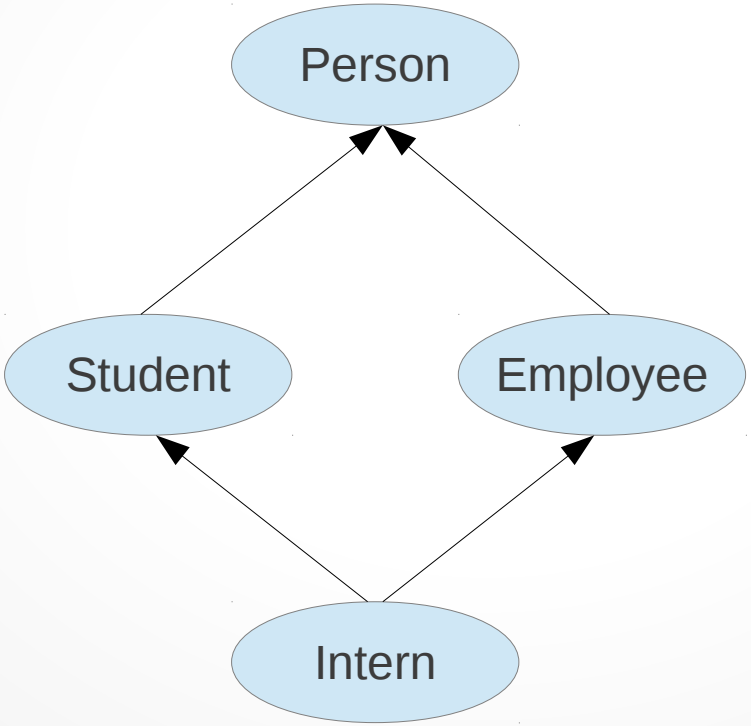
- Системно генерираните методи от голямата четворка правят това, което трябва
- Конструкторът по подразбиране на производния клас извиква съответните конструктори по подразбиране на основните класове
- Конструкторът за копиране на производния клас извиква съответните конструктори за копиране на основните класове
- Операцията за присвояване на производния клас извиква съответните операции за присвояване на основните класове
- Винаги редът е същия като реда на наследяване

Проблеми при множественото наследяване

- Усложнява се йерархията (става по-трудна за поддържане)
- Двусмислици при обръщения към компоненти на клас
  - коя от няколкото наследени компоненти се има предвид?
- Пример:
  bool Student::operator>(Student const& s) const
  { return grade > s.grade; }
  bool Employee::operator>(Employee const& e) const
  { return salary > e.salary; }
- Intern i1(40000, 5.00, „продавач“, 250, 30),
  i2(40001, 4.00, „шофьор“, 500, 20);
- if (i1 > i2) cout << „i1 е по-добър“;

Косвено повторно наследяване

- Deadly Diamond of Death



Физическо представяне

Intern								
Student				**Employee**				
Person				**Person**				
name	id	fn	grade	name	id	position	salary	period

- Всяка компонента на Person се повтаря
- Коя наследена компонента ще върне (Person)i?
- Ако Intern i, какво ще върне i.getName()?
- Раздвояване на личността!
- Какво всъщност искаме да се получи?

Желаното физическо представяне

Intern						
Person		**Student**		**Employee**		
name	id	fn	grade	position	salary	period

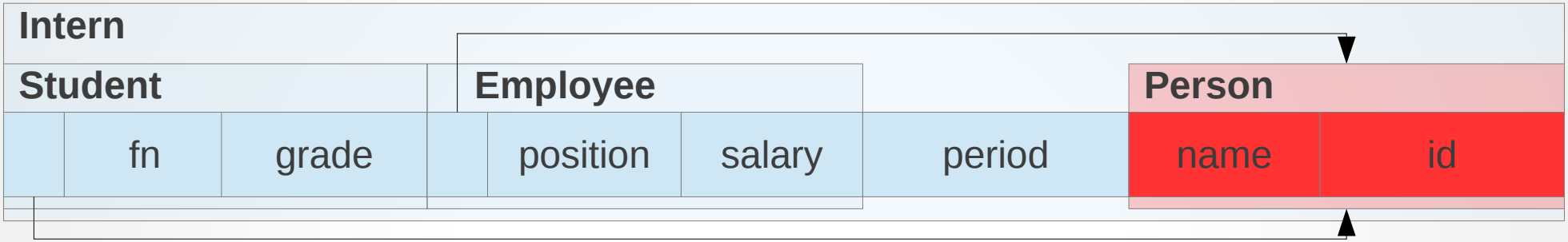
- Искаме да разрешим нееднозначността като поддържаме единствено копие на общия пра-основен клас
- Проблем: нарушаваме структурата на йерархията!
- Ако преобразуваме Intern до Employee, няма да можем да го преобразуваме после до Person!

Желаното физическо представяне

Intern						
Person		**Student**		**Employee**		
name	id	fn	grade	position	salary	period

- Искаме да разрешим нееднозначността като поддържаме единствено копие на общия пра-основен клас
- Проблем: нарушаваме структурата на йерархията!
- Ако преобразуваме Intern до Employee, няма да можем да го преобразуваме после до Person!

Правилното физическо представяне



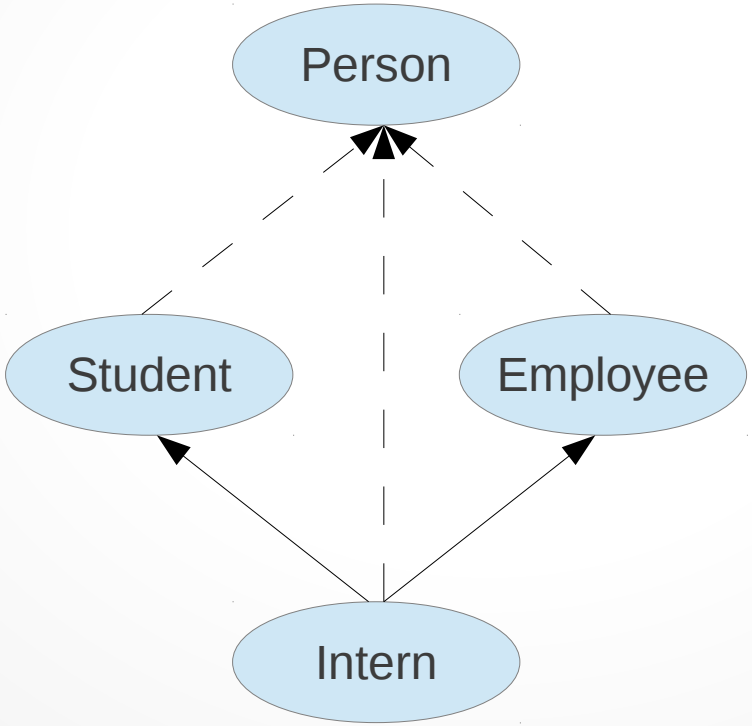
- Свързано представяне на веригата от наследници!
- Person е споделен основен клас за Student и Employee
- Такъв основен клас наричаме **виртуален**

Виртуални основни класове

- class Student : **virtual** public Person { ... }
  class Employee : **virtual** public Person { ... }
- Един клас не може да е виртуален сам по себе си, той може да бъде наследен виртуално, т.е. да е **виртуален основен** клас
- Всеки клас може да бъде наследен както виртуално, така и обикновено
- Транзитивност:
  Ако класът A е виртуален основен за B и B е основен за C, то A автоматично е виртуален основен и за C

Виртуално косвено повторно наследяване

- Escaping the Deadly Diamond of Death



Особености на виртуалните класове

- Виртуалността на основните класове се разпространява автоматично по йерархията
- Затова всеки клас, трябва да се „грижи“ за всичките си виртуални основни класове, вместо да разчита за тази грижа на своите директни родители

Конструктори на виртуални класове

- Пример:

  Intern(char const* _name, char const* _id, int _fn, double _grade, char const* _position, double _salary, int _period) : period(_period),
  Student(_name, _id, _fn, _grade),
  Employee(_name, _id, _position, _salary) {}

- Колко пъти и кой конструктор на Person ще се извика?
  - един път конструкторът по подразбиране!
- Отговорността за извикване на конструктора на Person е на Intern
- Понеже Intern(...) не извиква конструктор на Person, се извиква конструкторът по подразбиране
- Компилаторът **игнорира** извикванията на конструкторите на Person в конструкторите на Student и Employee!

Конструктори на виртуални класове

- Пример:

  Intern(char const* _name, char const* _id, int _fn, double _grade, char const* _position, double _salary, int _period) : period(_period),
  Student(NULL, NULL, _fn, _grade),
  Employee(NULL, NULL, _position, _salary),
  Person(_name, _id) {}

- Ако клас Demonstrator наследява Intern, то конструкторът на Demonstrator също трябва да извиква явно конструктора на Person
- Конструкторите на виртуални основни класове винаги се извикват първи!

Други проблеми на косвеното повторно наследяване

- Двусмислицата е на концептуално ниво, не е достатъчно да използваме виртуално наследяване, за да я разрешим!
- Раздвояване ще се получи при операциите, които са общи за всички класове от йерархията
- Примери:
  - Intern::print ще отпечатва една и съща Person част два пъти
  - Intern::read ще въвежда Person частта два пъти
  - operator= ще копира Person частта два пъти

Избягване на повторенията

- Следваме примера на конструкторите на виртуални класове
- Всеки клас извиква директно операцията за всичките си виртуални основни класове
- Всеки клас има вариант на операцията, в който се пропуска извикването на операцията за виртуалните основни класове

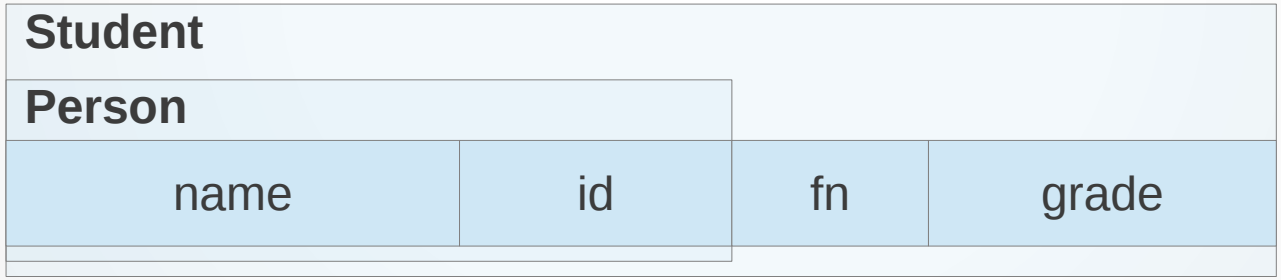
Избягване на повторенията

- Пример:

```
// operator<< извежда собствените член-данни
// и член-данните на обикновените основни класове
ostream& operator<<(ostream& os, <клас> const& obj) {
  os << (<невиртуален_основен_клас> const&)(*this);
  <извеждане_на_собствените_член_данни>
  return os;
}
// print извежда всичко като operator<<
// и в допълнение и член-данните на виртуалните основни класове
void <клас>::print() const {
  cout << (<виртуален_основен_клас> const&)(*this);
  cout << *this;
}
```

Преобразуване на типовете при наследяване

- При обикновено наследяване (Person -> Student) ако преобразуваме обект от клас Student до обект от клас Person, те ще имат един и същ адрес в паметта
  - понеже наследената част е винаги в началото на обекта



- Student s; Person &ps = s;
- cout << &s << ' ' << &ps; // извежда се един и същ адрес

Преобразуване на типовете при множествено наследяване

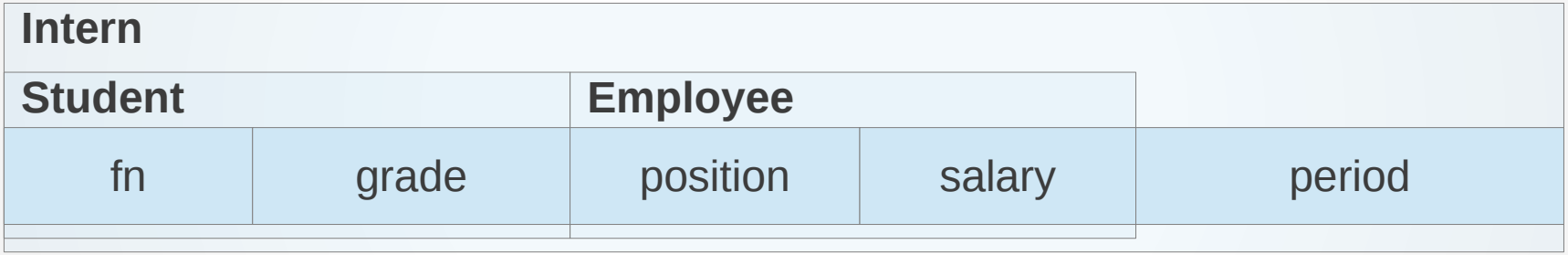
- При множествено наследяване (Student → Intern ← Employee), ако преобразуваме обект от клас Intern до обект от клас Employee, те няма да имат един и същ адрес в паметта
  - понеже наследената част може да не е в началото на обекта

<table>
<tr><td colspan="5">Intern</td></tr>
<tr><td colspan="2">Student</td><td colspan="2">Employee</td><td></td></tr>
<tr><td>fn</td><td>grade</td><td>position</td><td>salary</td><td>period</td></tr>
</table>

- Intern i; Employee &ei = i;
- cout << &i << ' ' << &ei; // извеждат се различни адреси!

Преобразуване на типовете при множествено наследяване

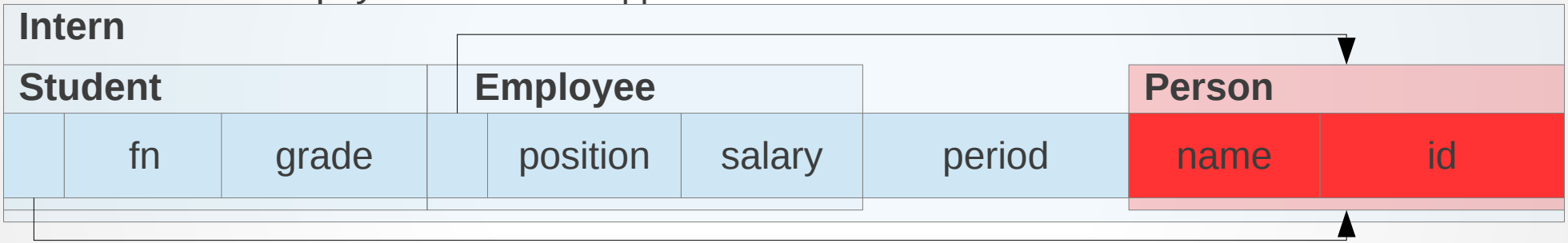
- Ако се опитаме да преобразуваме основен в производен клас, компилаторът автоматично преизчислява началния адрес според схемата на наследяване



- Intern i; Employee &ei = i; Intern& i2 = (Intern&)ei;
- cout << &i << ' ' << &ei; // извеждат се различни адреси!
- cout << &i << ' ' << &i2; // извеждат се еднакви адреси!

Преобразуване на типовете при виртуално наследяване

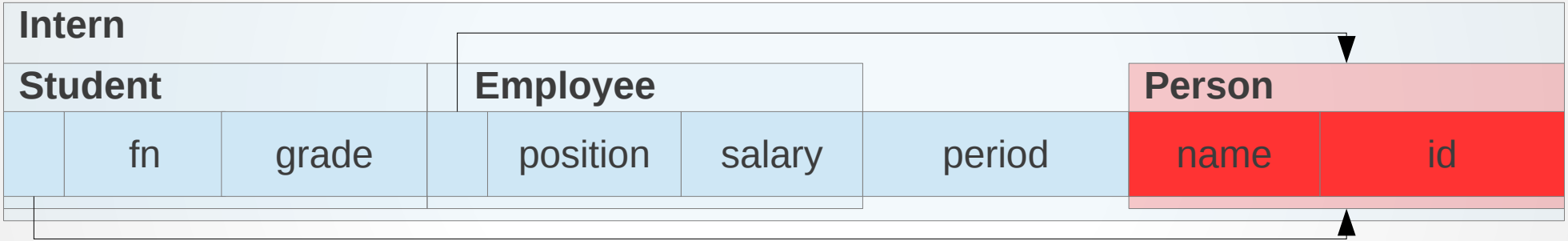
- При виртуално наследяване, ако преобразуваме обект от производен клас до обект от виртуален основен клас, те няма да имат един и същ адрес в паметта
  - понеже виртуалните наследени части не са в началото на обекта



- Intern i; Person &pi = i;
- cout << &i << ' ' << π // извеждат се различни адреси!
- Student& s = si; Person &psi = si;
- cout << &i << ' ' << π // извеждат се различни адреси!
- cout << &pi << ' ' << ψ // извеждат се еднакви адреси!

Операция за преобразуване reinterpret_cast

- Можем да забраним на компилатора да прави преизчисление на началните адреси с операцията за преобразуване reinterpret_cast
- „Вземи дословно същия начален адрес като преобразуваш“



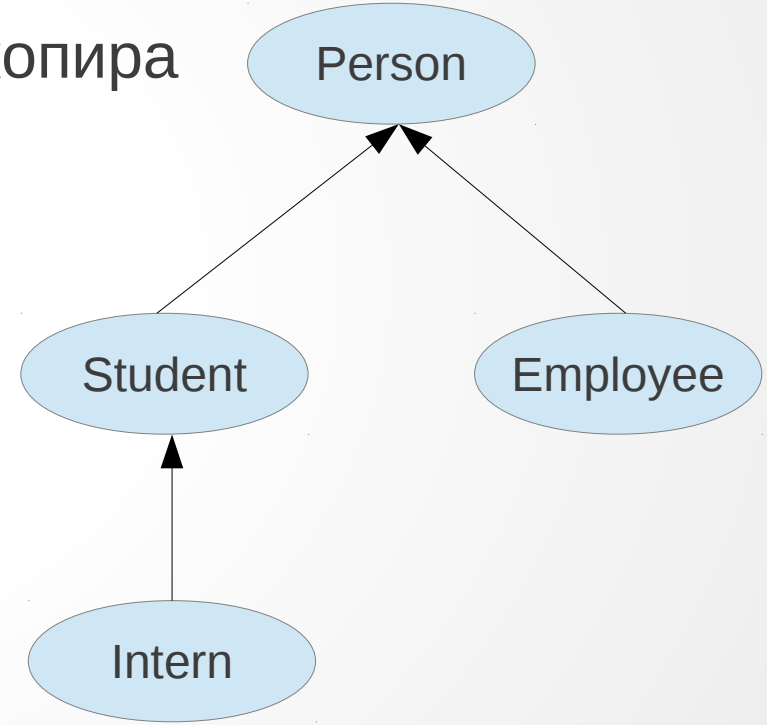
- Intern& i; Student &si = i; Intern& isi = reinterpret_cast<Intern&>si;
  - работи правилно, понеже i и si имат един и същ начален адрес
- Intern& i; Employee &ei = i; Intern& iei = reinterpret_cast<Intern&>ei;
  - обектът iei е невалиден, член-данните са Employee се интерпретират като член-данни на Student!

Основни недостатъци на множественото наследяване

- Усложнена йерархия
- Усложнено представяне в паметта
- Усложнена логика на компилатора
- Двусмислици при косвено повторно наследяване
- Евентуални конфликти между няколко йерархии

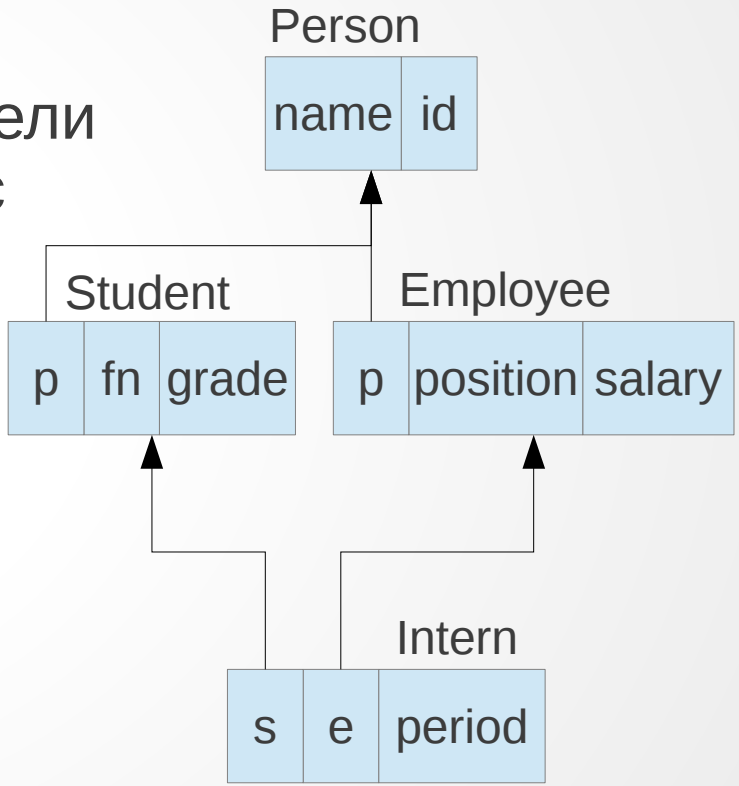
Алтернативно решение №1

- Единият основен клас се избира за главен
- Кодът на другите основни класове се копира
- Предимства:
  - опростяваме йерархията
  - можем да пропуснем да ненужните компоненти на Employee
- Недостатъци:
  - копиране на код
  - Intern не може да се разглежда като Employee



Алтернативно решение №2

- Делегиране вместо наследяване
- Всеки обект има един или повече указатели към „прототипни“ обекти от основен клас
- Предимства:
  - споделени данни в случай на диамант
  - динамичен контрол върху йерархията
- Недостатъци:
  - преобразуването се прави по време на изпълнение и е по-бавно
  - налага се явно извикване на наследени методи



Алтернативно решение №3

- Комбинация от №1 и №2
- Избираме си главна йерархия
- При нужда, присъединяваме вторична йерархия чрез делегиране
- Предимства:
  - за главната йерархия имаме предимствата на наследяването
  - за вторичните йерархии имаме гъвкавостта на делегацията
- Недостатъци:
  - позволява само една главна йерархия
  - не решава проблема с диаманта

Алтернативно решение №4

- Проблемите при множествено наследяване се появяват, когато имаме множествено наследени член-данни и реализации на член-функции
- Множествено наследяване на абстрактни класове (интерфейс)
  - класове, в които няма член-данни и реализации на член-функции
  - какво има тогава в такива класове?
  - само декларации на член-функции!
  - т.е., описание на операциите, които наследниците им поддържат
- Ще говорим за абстрактни класове в следващата лекция