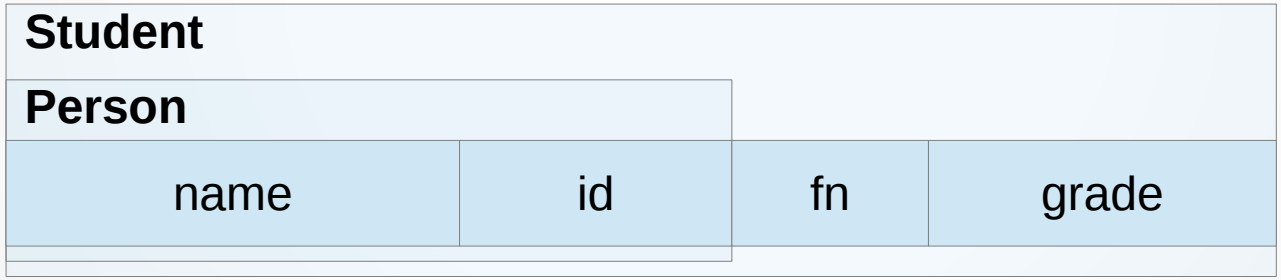
Множествено наследяване

Част 2

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

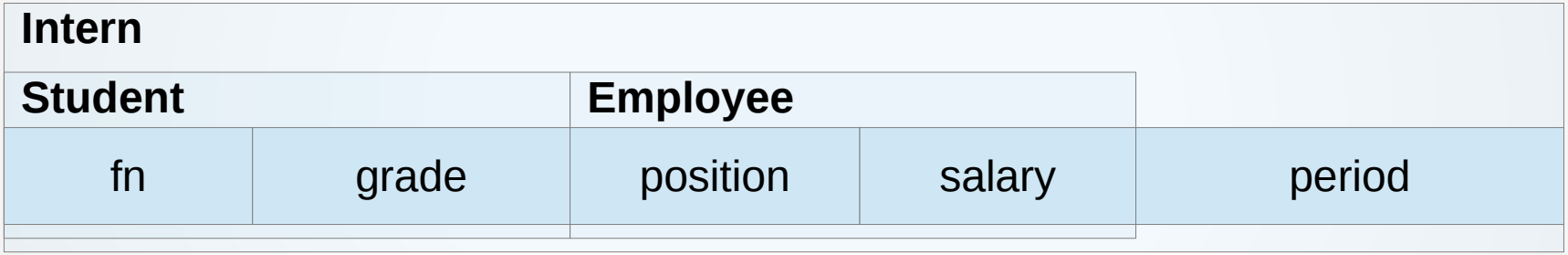
- При множествено наследяване (Student → Intern ← Employee), ако преобразуваме обект от клас Intern до обект от клас Employee, те няма да имат един и същ адрес в паметта
  - понеже наследената част може да не е в началото на обекта

<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

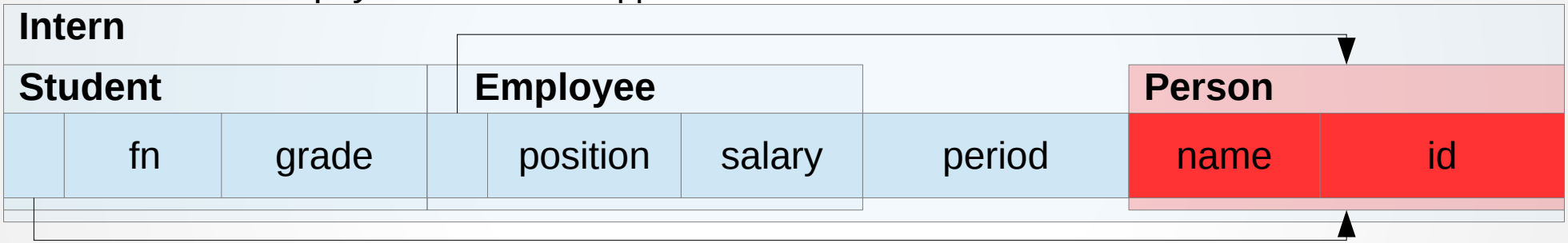
- Ако се опитаме да преобразуваме основен в производен клас, компилаторът автоматично преизчислява началния адрес според схемата на наследяване



- 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

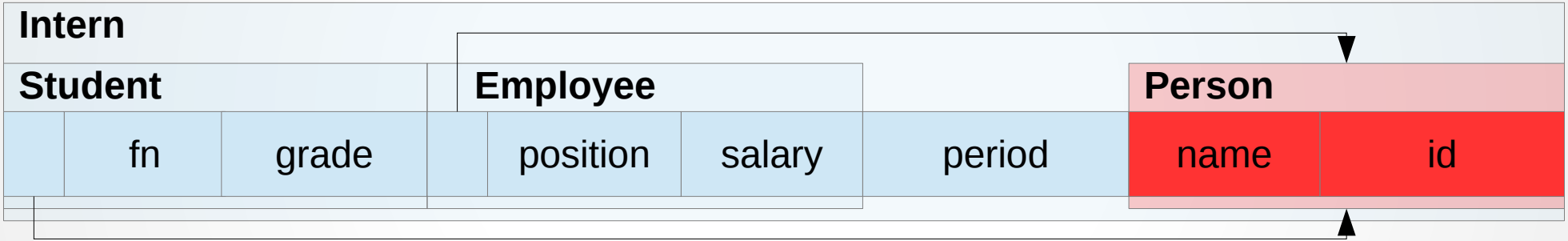
- При виртуално наследяване, ако преобразуваме обект от производен клас до обект от виртуален основен клас, те няма да имат един и същ адрес в паметта
  - понеже виртуалните наследени части не са в началото на обекта



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



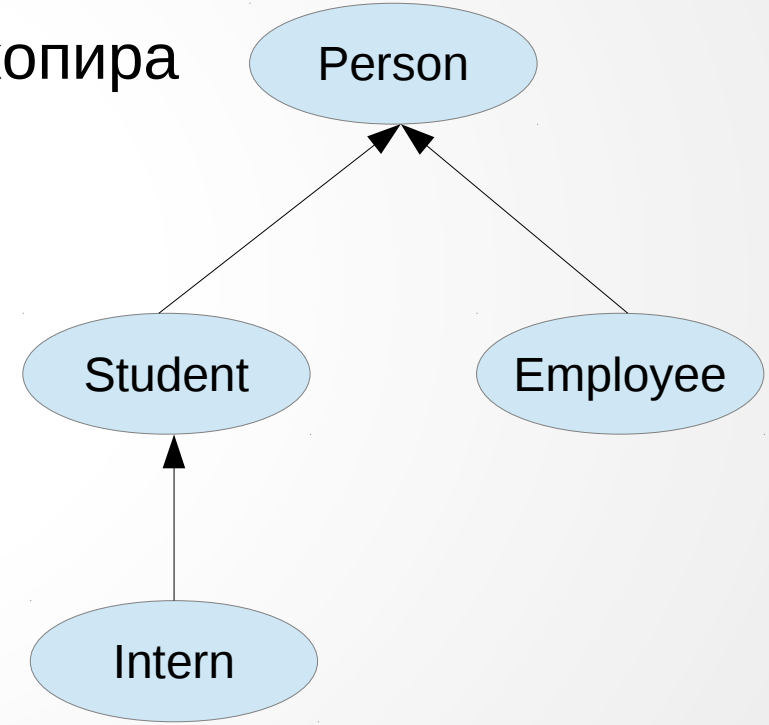
- Intern& i; Student &si = i; Intern& isi = reinterpret_cast<Intern&>si;
  - работи правилно, понеже i и si имат един и същ начален адрес
- Intern& i; Employee &ei = i; Intern& iei = reinterpret_cast<Intern&>ei;
  - обектът iei е невалиден, член-данните са Employee се интерпретират като член-данни на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

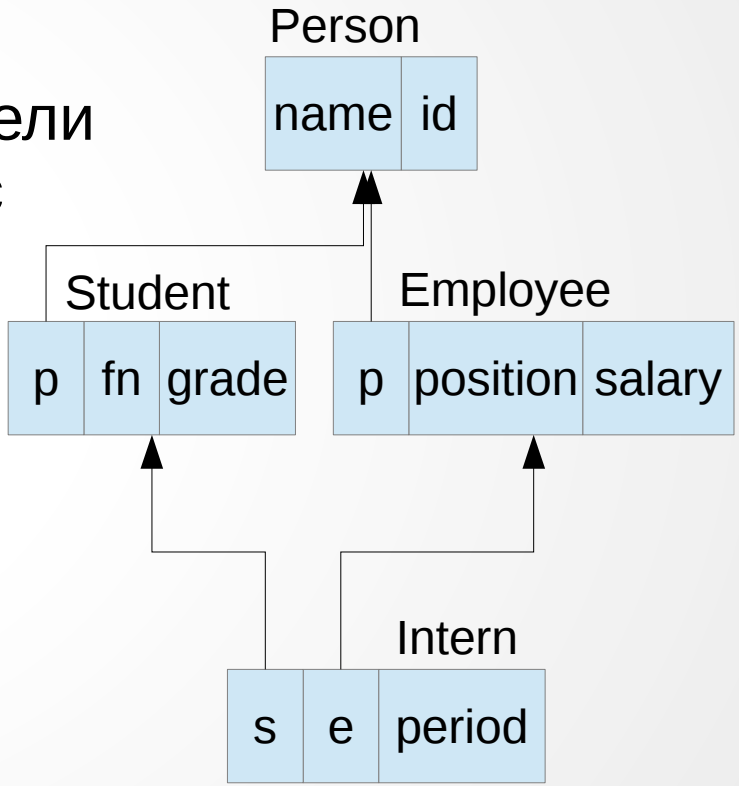
Алтернативно решение №1

- 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