Sa proceduralnog na OO programiranje: polimorfizam

- Ideja — osloniti se na strukture podataka i na (dinamičko) ulančavanje pokazivača i pokazivače na funkcije, a onda i na dinamičko vezivanje:

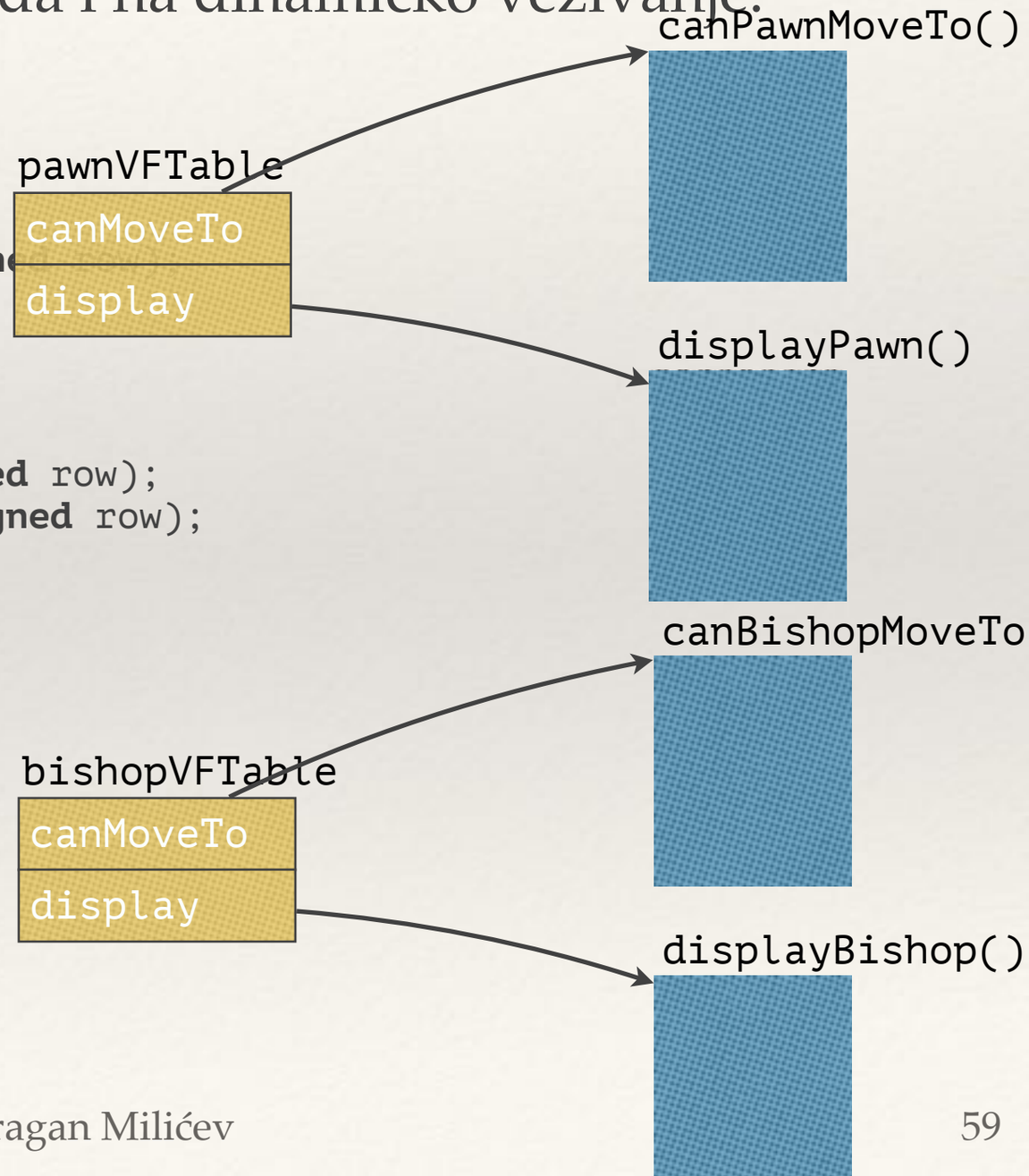
```
// Table of pointers to implementations
// of virtual functions (“virtual table”)

struct Figure_VFTable {
  int (*canMoveTo) (Figure* fig, unsigned col, unsigne
  int (*display) (Figure* fig, …);
  …
};

int canPawnMoveTo (Figure* fig, unsigned col, unsigned row);
int canBishopMoveTo (Figure* fig, unsigned col, unsigned row);
…

Figure_VFTable pawnVFTable;
pawnVFTable.canMoveTo = &canPawnMoveTo;
pawnVFTable.display = &displayPawn;
…

Figure_VFTable bishopVFTable;
bishopVFTable.canMoveTo = &canBishopMoveTo;
bishopVFTable.display = &displayBishop;
…
```



Sa proceduralnog na OO programiranje: polimorfizam

- Kod za funkcije generiše prevodilac, a prikazane strukture (tabele) postoje po jedna za svaku klasu (vrstu objekta) i inicijalizuju se statički, za vreme prevođenja
- Svaki objekat ima pokazivač na takvu tabelu pokazivača na implementacije virtuelnih funkcija koje odgovaraju svakoj pojedinoj klasi (vrsti objekta), tzv. *virtual table pointer*
- Ovaj pokazivač potrebno je inicijalizovati za svaki objekat, u zavisnosti od njegove vrste:

```
struct Figure {
  Figure_VFTable* vtp;  // Virtual table pointer
  FigureColor color;
  …
};

void initPawn (Figure* fig,…) {
  fig->vtp = &pawnVFTable;
  …
}
```