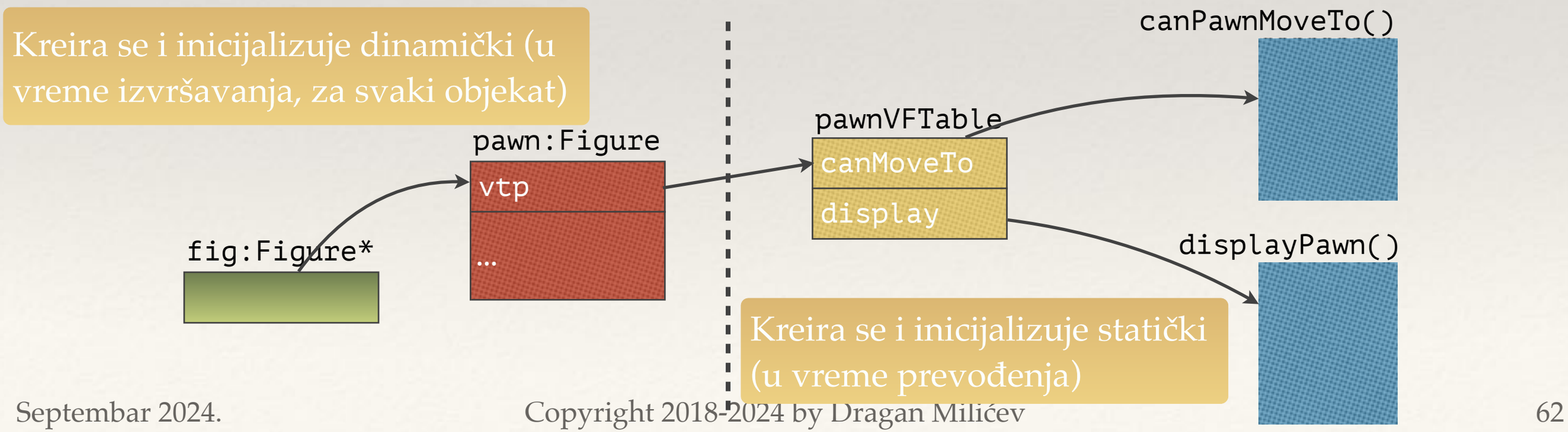
Sa proceduralnog na OO programiranje: polimorfizam

Upravo na ovakav način se na jeziku C++ implementira polimorfizam:

- za svaku klasu koja ima bar jednu virtuelnu funkciju članicu, prevodilac generiše *tabelu virtuelnih funkcija* (*virtual table*), sa pokazivačima koje (statički, u vreme prevođenja) inicijalizuje tako da ukazuju na verzije implementacija funkcija koje odgovaraju toj klasi (nasleđena ili redefinisana)
- u svakom objektu ovakve klase postoji *pokazivač na tabelu virtuelnih funkcija* (*virtual table pointer*); ovaj pokazivač inicijalizuje konstruktor; svaki konstrukor na tabelu koja odgovara baš toj klasi, tako da ga konstruktor osnovne klase najpre postavi na svoju tabelu (te osnovne klase), a onda konstruktor izvedene klase prepiše tako da ukazuje na tabelu te klase itd.
- za svaki poziv virtuelne funkcije objekta kome se pristupa preko pokazivača, npr. *fig->canMoveTo(…)*, prevodilac generiše kod koji taj poziv rešava dinamičkim vezivanjem, tj. indirektnim pristupom (memorijskim indirektnim adresiranjem) preko pokazivača na virtuelnu tabelu i pokazivača na funkciju u ulazu te tabele koji odgovara toj virtuelnoj funkciji u klasi



Sa proceduralnog na OO programiranje: polimorfizam

- Prema tome, efikasnost poziva virtuelne funkcije na jeziku C++ je ista kao i u ekvivalentnom C kodu sa dinamičkim vezivanjem, i tek nešto malo manja nego poziv statičkim vezivanjem (skok na adresu memorijski direktnim adresiranjem), ali je ceo mehanizam sakriven od programera i program je apstraktniji, kompaktniji i lakši za razumevanje i održavanje:

```
struct Figure_VFTable {
  int (*canMoveTo) (Figure* fig,…);
  int (*display) (Figure* fig, …);
  …
};

int canPawnMoveTo (Figure* fig,…);
int canBishopMoveTo (Figure* fig,…);
…

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

struct Figure {
  Figure_VFTable* vtp;
  FigureKind kind;
  …
};

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

int canMoveTo (Figure* fig,…) {
  return fig->vtp->canMoveTo(fig,col,row);
}
```

```
class Figure {
public:
  Figure ();
  virtual int canMoveTo (…);
  virtual int display (…);
  …
};

class Pawn : public Figure {
public:
  Pawn ();
  virtual int canMoveTo (…);
  virtual int display (…);
  …
};

…

…aFig->canMoveTo(…)…
```