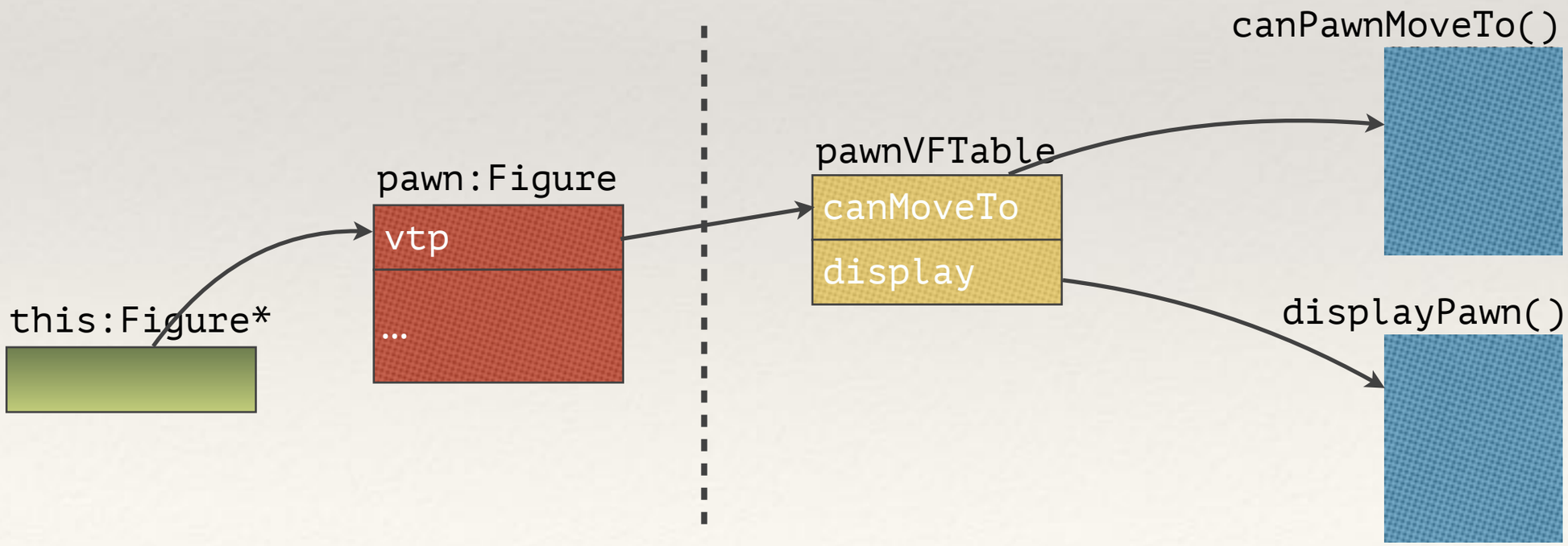
Sa proceduralnog na OO programiranje: polimorfizam

- Sada se *polimorfan* poziv realizuje jednostavno, *dinamičkim vezivanjem* (*dynamic binding*), preko lanca pokazivača, uvek istim kodom:

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

int display (Figure* this, …) {
  return this->vtp->display(this,…);
}
```

- Pogodnost: kod pozivaoca se nimalo ne menja proširenjem hijerarhije klasa (dodavanjem novih podvrsta)



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