Dzisiejszy wykład

- Wzorce projektowe
  - Visitor
  - Client-Server
  - Factory
  - Singleton

Wzorzec projektowy

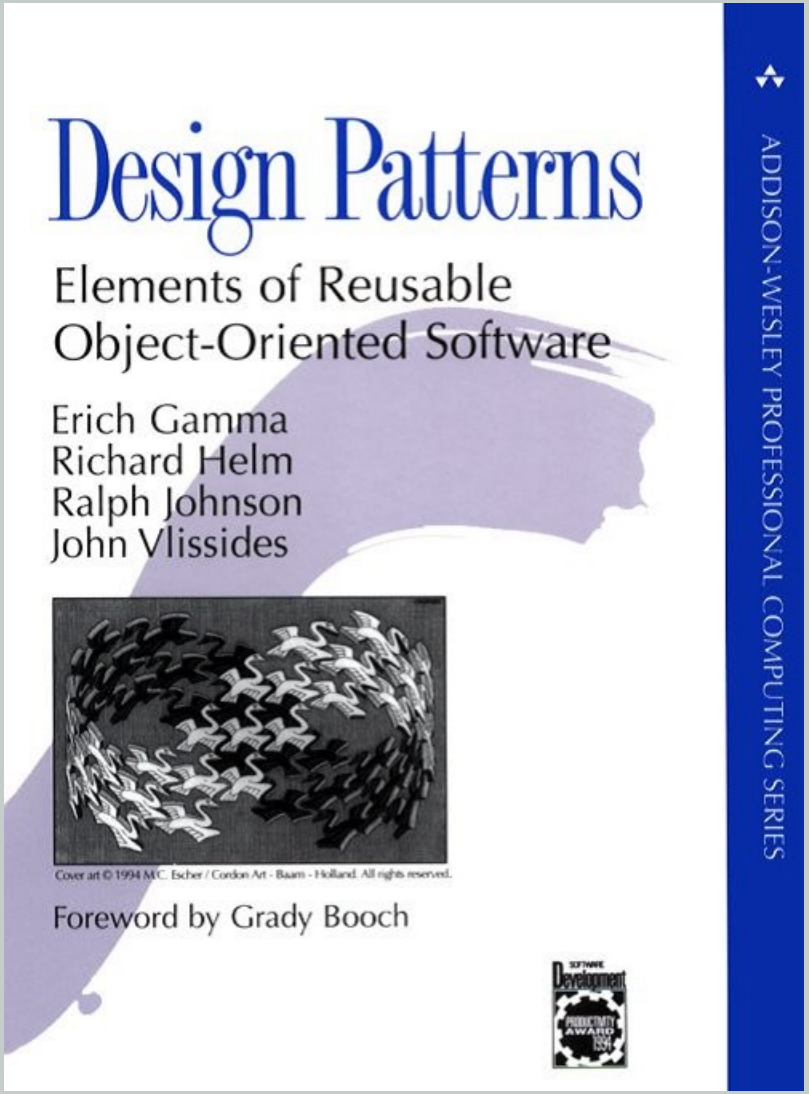
- Wzorzec
  - nazwana generalizacja opisująca elementy i relacje rozwiązania powszechnie występującego problemu projektowego
- Cztery podstawowe elementy wzorca
  - opisowa nazwa
  - rozwiązywany problem
  - rozwiązanie problemu
  - konsekwencje zastosowania wzorca
- Wzorzec opisuje zbiór klas i relacji między nimi, które będą razem tworzyć rozwiązanie problemu

Rozpoznawanie wzorców

- Duża część procesu programowania, od projektu do implementacji, opiera się na zastosowaniu pewnych podstawowych, dobrze poznanych wzorców do danej sytuacji
- Możliwość szybkiego znalezienia tych wzorców w projekcie odróżnia nowicjusza od eksperta
- Studiowanie zorganizowanej biblioteki wzorców może, teoretycznie, przyspieszyć proces uzyskiwania dobrego wyniku przez nowicjusza

Wzorce projektowe

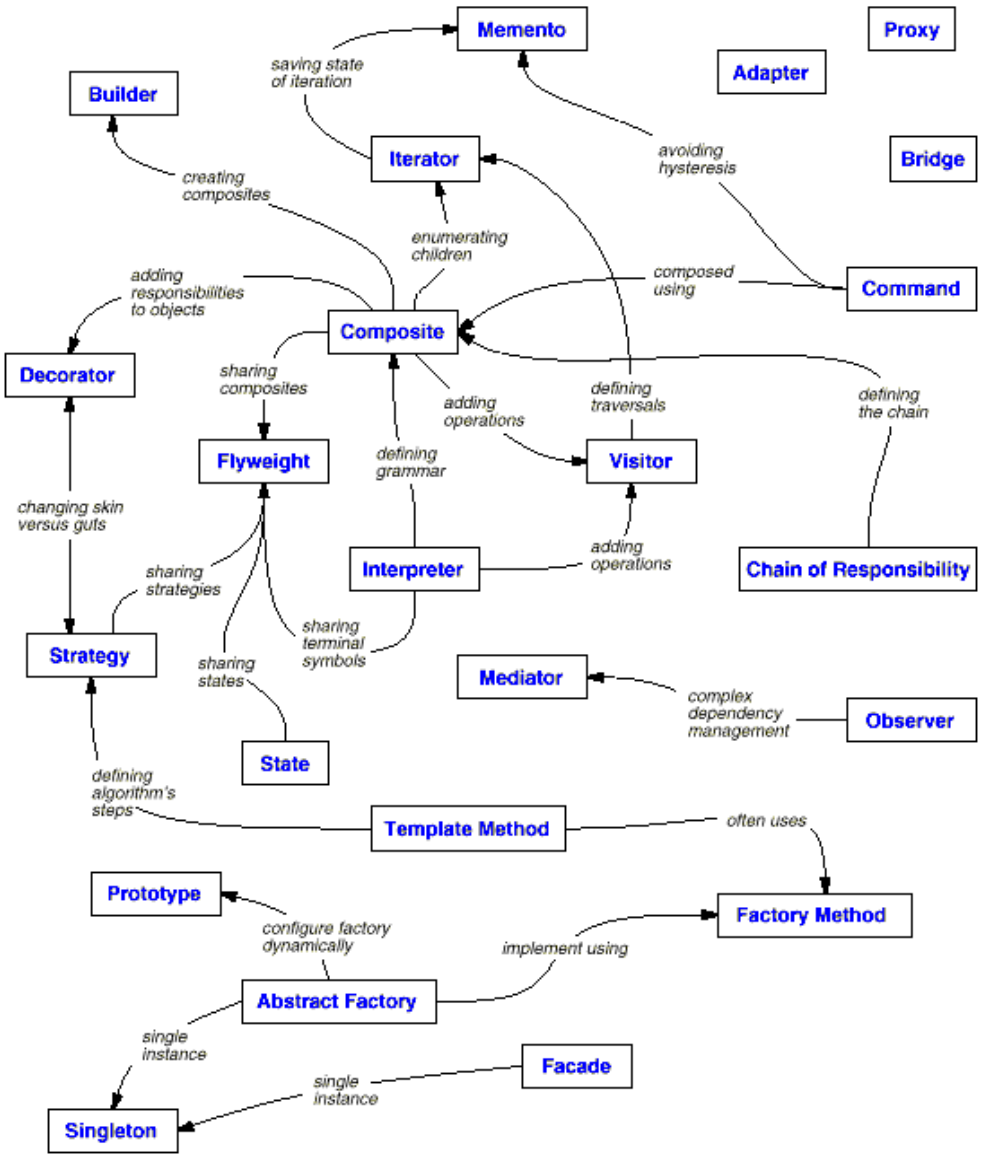
- Książka "Design Patterns" - Gamma, Helm, Johnson i Vlissides
- "Gang of Four" (GOF) Book
- Definiuje
  - Creational Patterns (5)
  - Structural Patterns (7)
  - Behavioral Patterns (11)
- Wiele innych książek na ten temat



Definicja wzorca w GOF Book

- Nazwa - nazwa wzorca
- Cel
  - Co robi wzorzec?
  - Jakie jest jego uzasadnienie i cel?
  - Jakim konkretny problemem lub zagadnieniem się zajmuje?
- Motywacja
  - Scenariusz ilustrujący jak wzorzec rozwiązuje problem projektowy
- Stosowalność
  - W jakich sytuacjach wzorzec może być zastosowany?
  - Jakie są przykłady kiepskiego projektu, które wzorzec może poprawić?
  - Jak rozpoznać takie sytuacje?
- Struktura
  - Diagram UML dla składowych wzorca
- Uczestnicy
  - Klasy/obiekty we wzorcu i ich zadania
- Współpraca
  - Jak uczestnicy współpracują ze sobą
- Konsekwencje
  - Jak wzorzec rozwiązuje problem?
  - Jakie są kompromisy i rezultaty zastosowania wzorca?
  - Jakie aspekty struktury systemu mogą być zmieniane niezależnie?
- Implementacja
  - O jakich pułapkach, wskazówkach i technikach powinien wiedzieć programista?
  - Czy występują jakieś zależności od języka programowania?
- Przykładowy kod
- Znane zastosowania
  - Przykłady użycia
- Podobne wzorce
  - Inne blisko spokrewnione wzorce

Wzorce w GOF Book



Visitor Pattern

- Przejście przez wszystkie elementy z warunkowym zakończeniem

```
Go to first list element.
If test is satisfied, Quit.
While not at end of the list:
        Step to next list element.
        If test is satisfied, Quit.
```

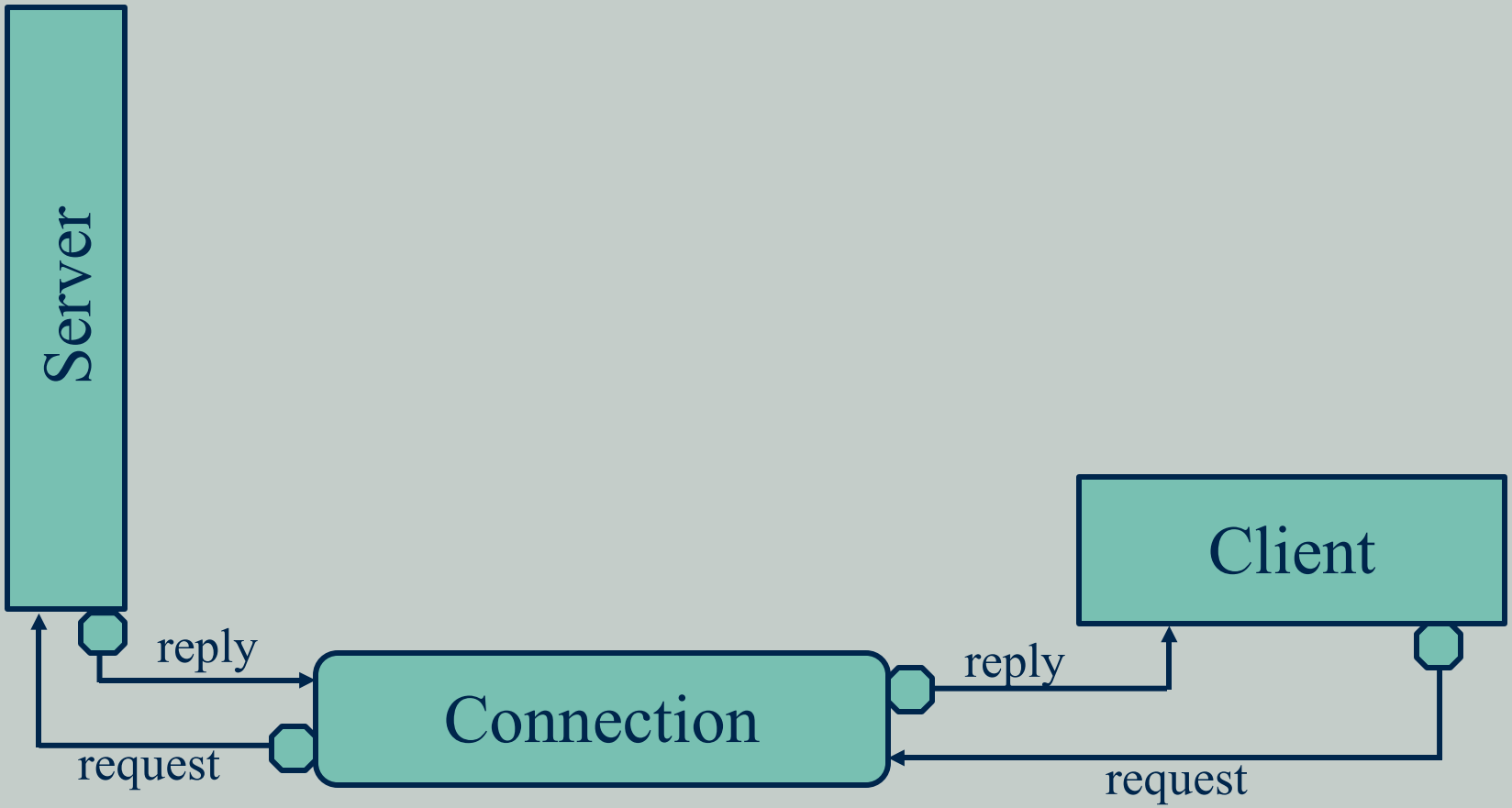
- Określa podstawowy wzorzec przeszukiwania listy
- Nie ma znaczenia, czy lista jest tablicą, listą z dowiązaniami czy czymś innym
- Nie podaje szczegółów warunku zakończenia
- Nie podaje, co się dzieje później

Visitor - przykład

```
class calculate {
public: static int total;
 void operator()(string v) { total += v.length(); }
 int getSum() const { return total; }
};
int calculate::total = 0;

int main() {
 list<string> alist;
 calculate fobj;
 string value;

 cout << "Enter strings, press ^D when done" << endl;
 cin >> value;
 while (cin) {
   alist.push_back(value);
 cin >> value;
}

 for_each(alist.begin(), alist.end(), fobj);
 cout << fobj.getSum() << endl;
}
```

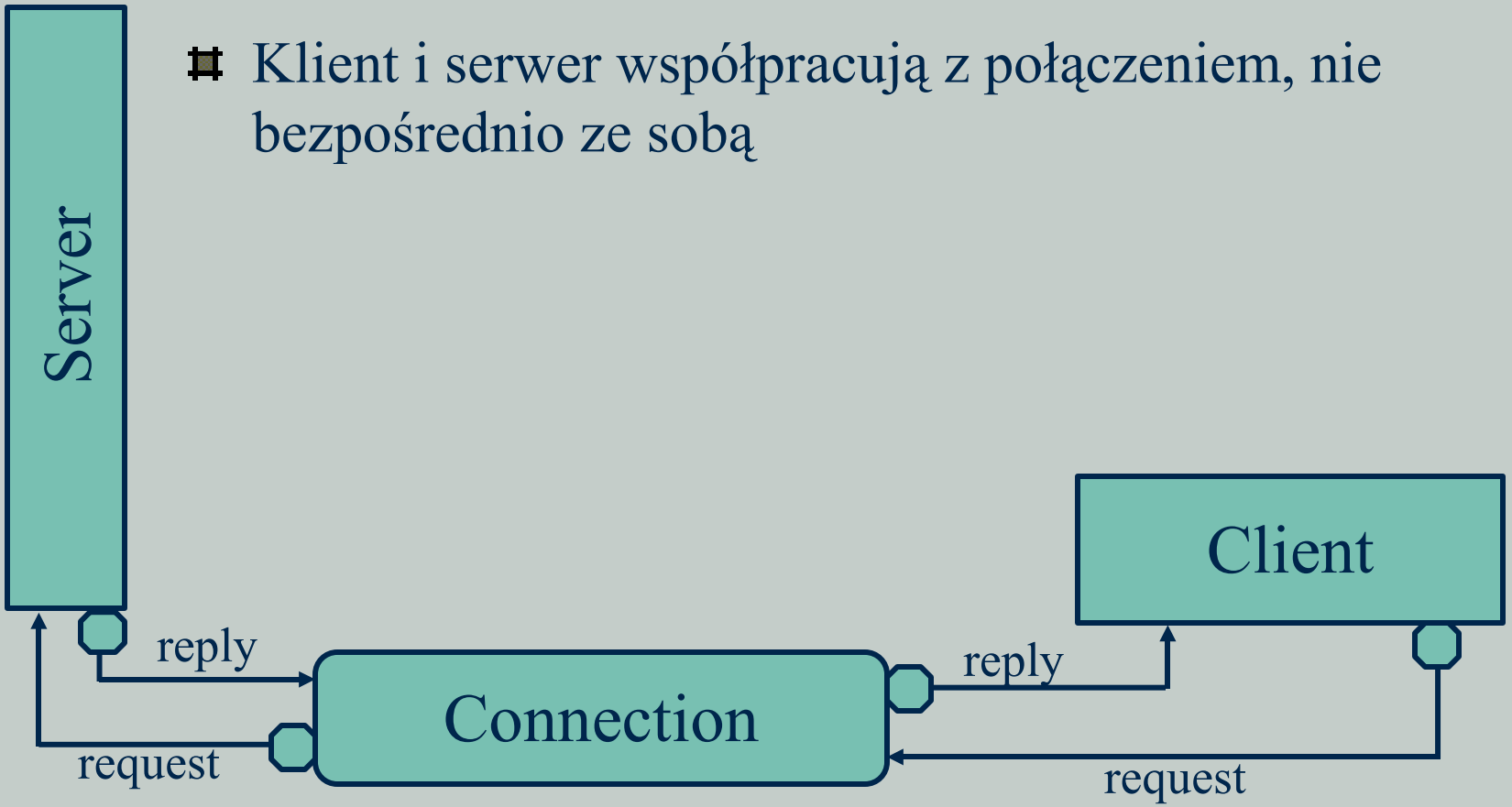
Client-Server

- Problem: dostarczanie usług dla wielu luźno połączonych klientów



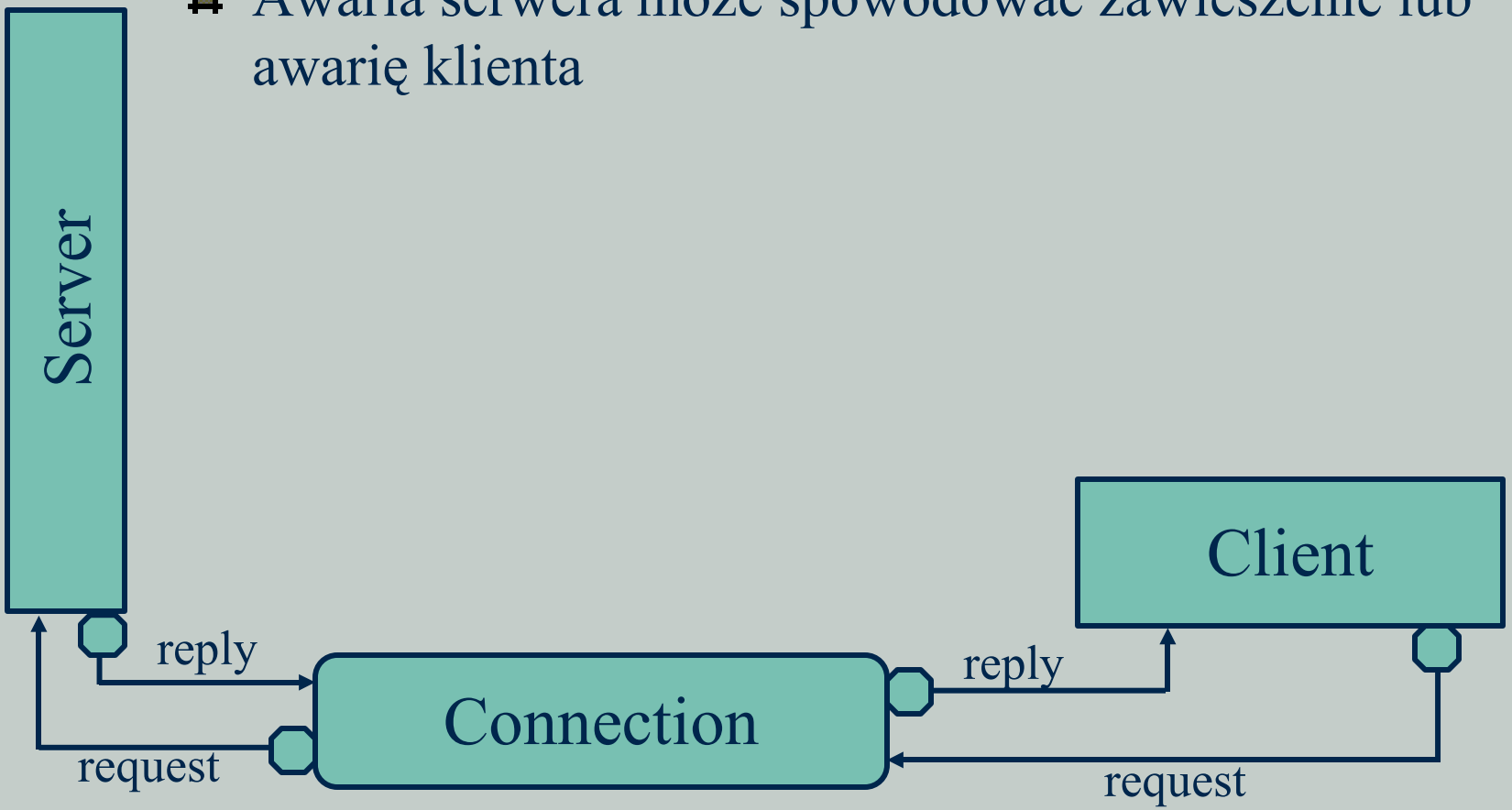
Elementy i zadania

- Klient generuje żądanie, które jest przesyłane do serwera, który z kolei generuje odpowiedź
- Połączenie przenosi pytania i odpowiedzi między klientem i serwerem
- Klient i serwer współpracują z połączeniem, nie bezpośrednio ze sobą



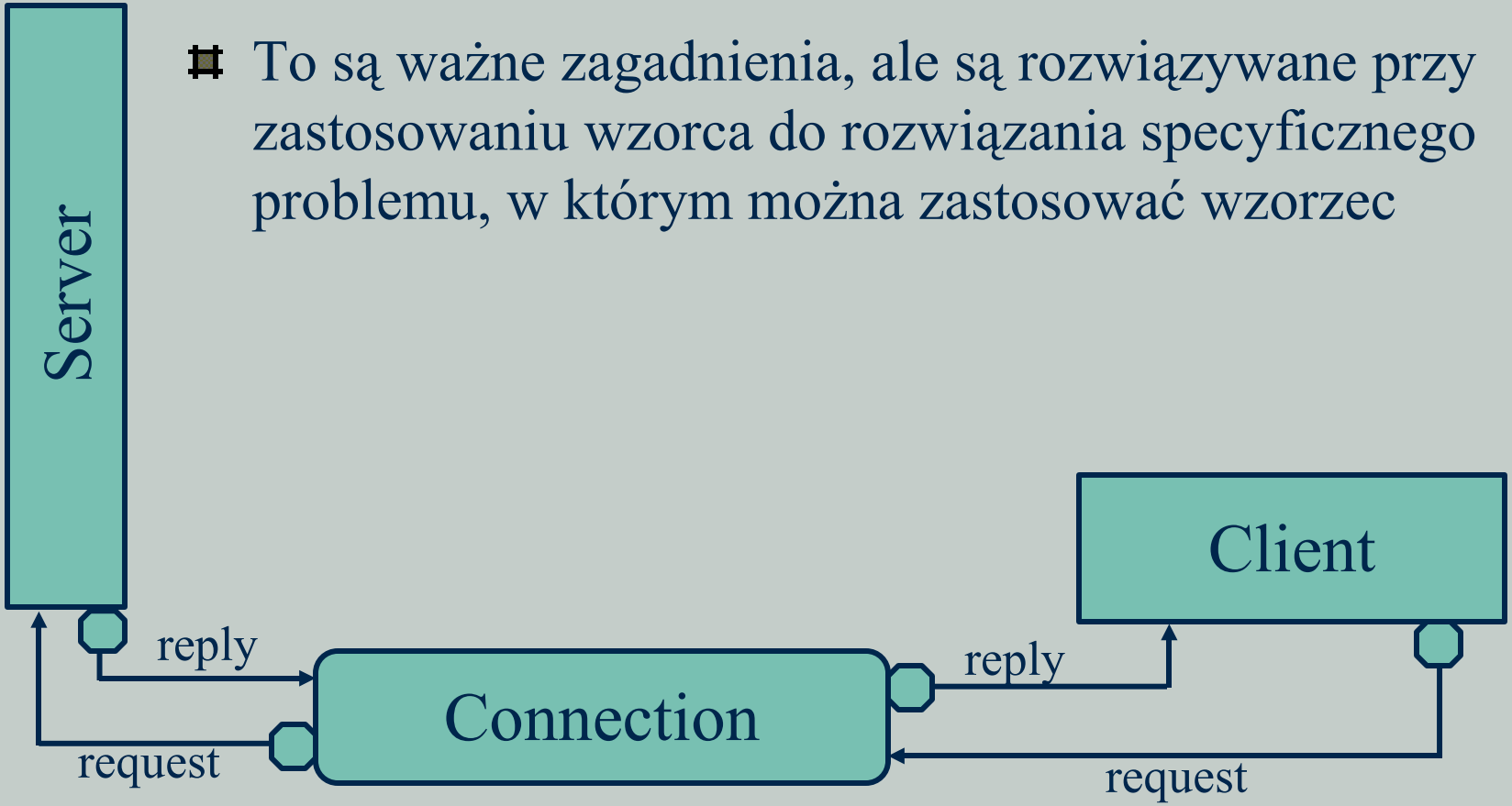
Konsekwencje

- Klient i serwer są niezależne od implementacji, z wyjątkiem typów przekazywanych komunikatów
- Łatwo uzyskać model obsługi "wiele do jednego"
- Awaria serwera może spowodować zawieszenie lub awarię klienta



Elastyczność

- Natura usług żądanych i dostarczanych nie ma znaczenia dla wzorca
- Natura połączenia (pipe, socket, bufor) nie ma znaczenia dla wzorca
- To są ważne zagadnienia, ale są rozwiązywane przy zastosowaniu wzorca do rozwiązania specyficznego problemu, w którym można zastosować wzorzec



Factory Method Pattern

- Cel: zdefiniowanie interfejsu do tworzenia obiektów nieznanego typu
- Przykład
  - Program z poleceniem "New" w menu File
  - Kod standardowy dla wszystkich aplikacji
  - Konkretny typ dokumentu zależy od aplikacji
    - Word - nowy dokument tekstowy
    - Excel - nowy arkusz kalkulacyjny
    - etc.
- Jak wyrazić zachowanie "New", jeżeli nie wiemy, jaki nowy obiekt stworzyć?
- Rozwiązanie: Factory Method

Factory Method

- Uczestnicy
  - Product (tutaj: dokument)
  - ConcreteProduct (WordDocument lub SpreadsheetDocument)
  - Creator (aplikacja)
    - klasa abstrakcyjna, która ma metodę Factory
    - virtual Product* Create()=0
  - ConcreteCreator (np. Word)
    - zastępuje metodę Factory i tworzy konkretny typ dokumentu
    - virtual Product* Create() { return new WordDocument(); }

Singleton Pattern

- Cel: Zapewnić, że istnieje tylko pojedynczy obiekt danej klasy i sposób do odwołania się do tego obiektu
- Realizowane przez definiowanie chronionego lub prywatnego konstruktora
- Przykład implementacji

```
class Singleton {
 protected:
   Singleton() { /* do whatever might be needed here */ }
 private:
   static Singleton* theInstance;
 public:
   static Singleton* Instance() {
      if (theInstance == NULL) {
         theInstance = new Singleton();
      }
      return theInstance;
   }
};
Singleton* Singleton::theInstance = NULL;
```