Modular Design

Software Systems – Design – L6T4½

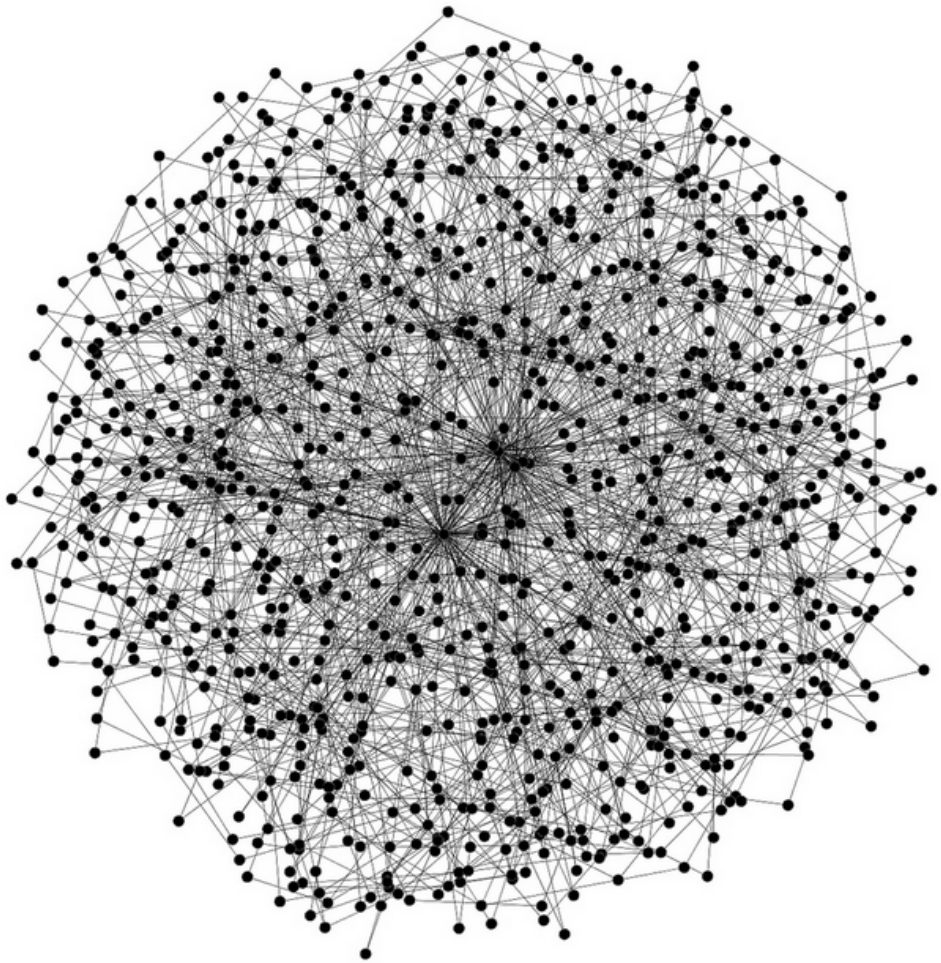
Peter Lammich

Complex Software Systems

- Complexity of software quickly increases

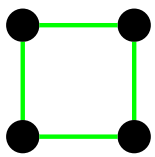
Complex Software Systems

- Complexity of software quickly increases
- and becomes unmanageable



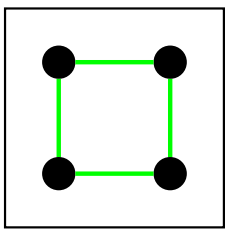
Modularization

Design complex system from smaller parts



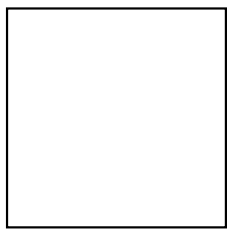
Modularization

Design complex system from smaller parts



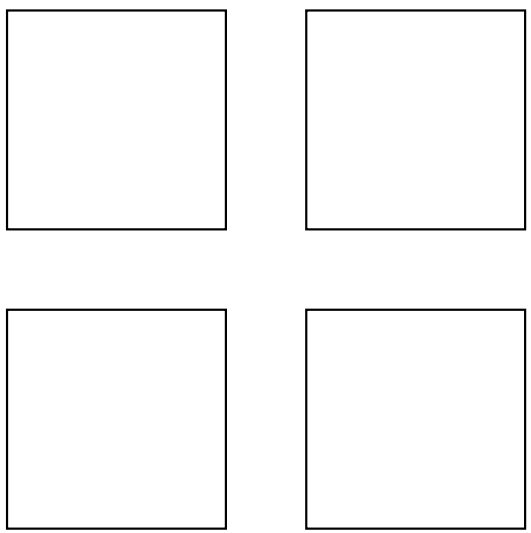
Modularization

Design complex system from smaller parts



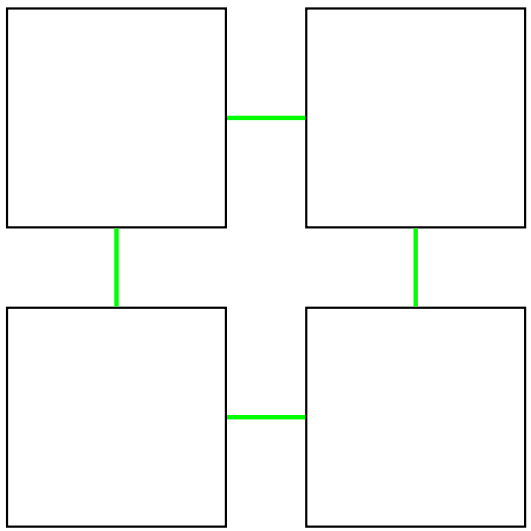
Modularization

Design complex system from smaller parts



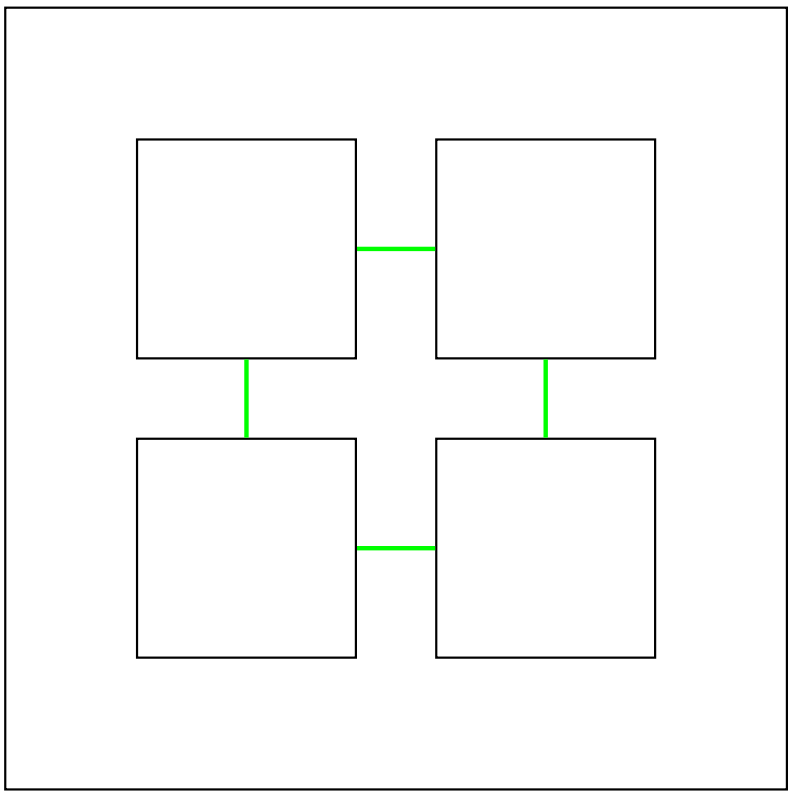
Modularization

Design complex system from smaller parts



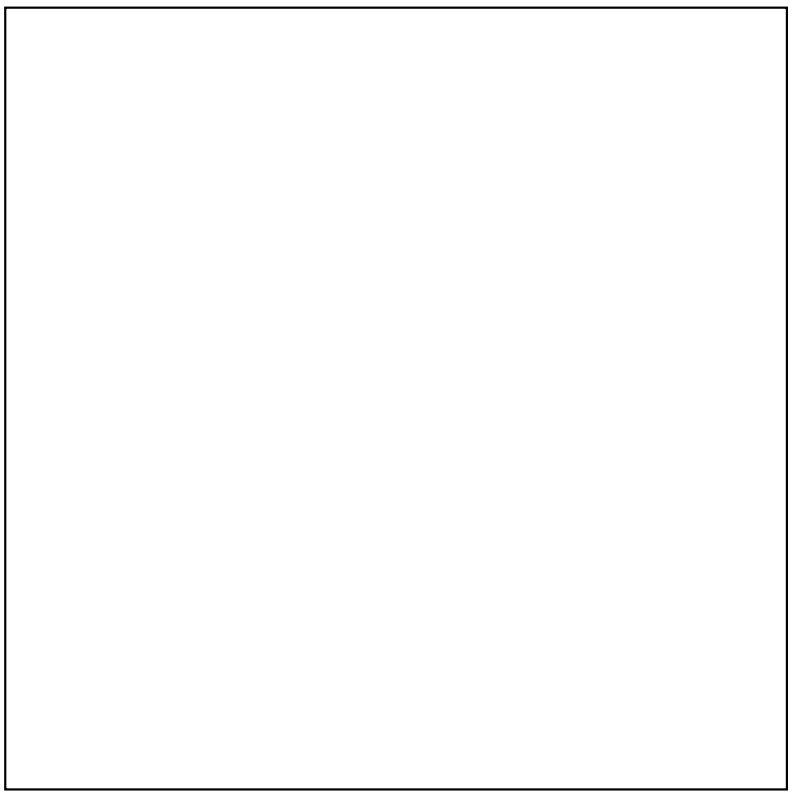
Modularization

Design complex system from smaller parts



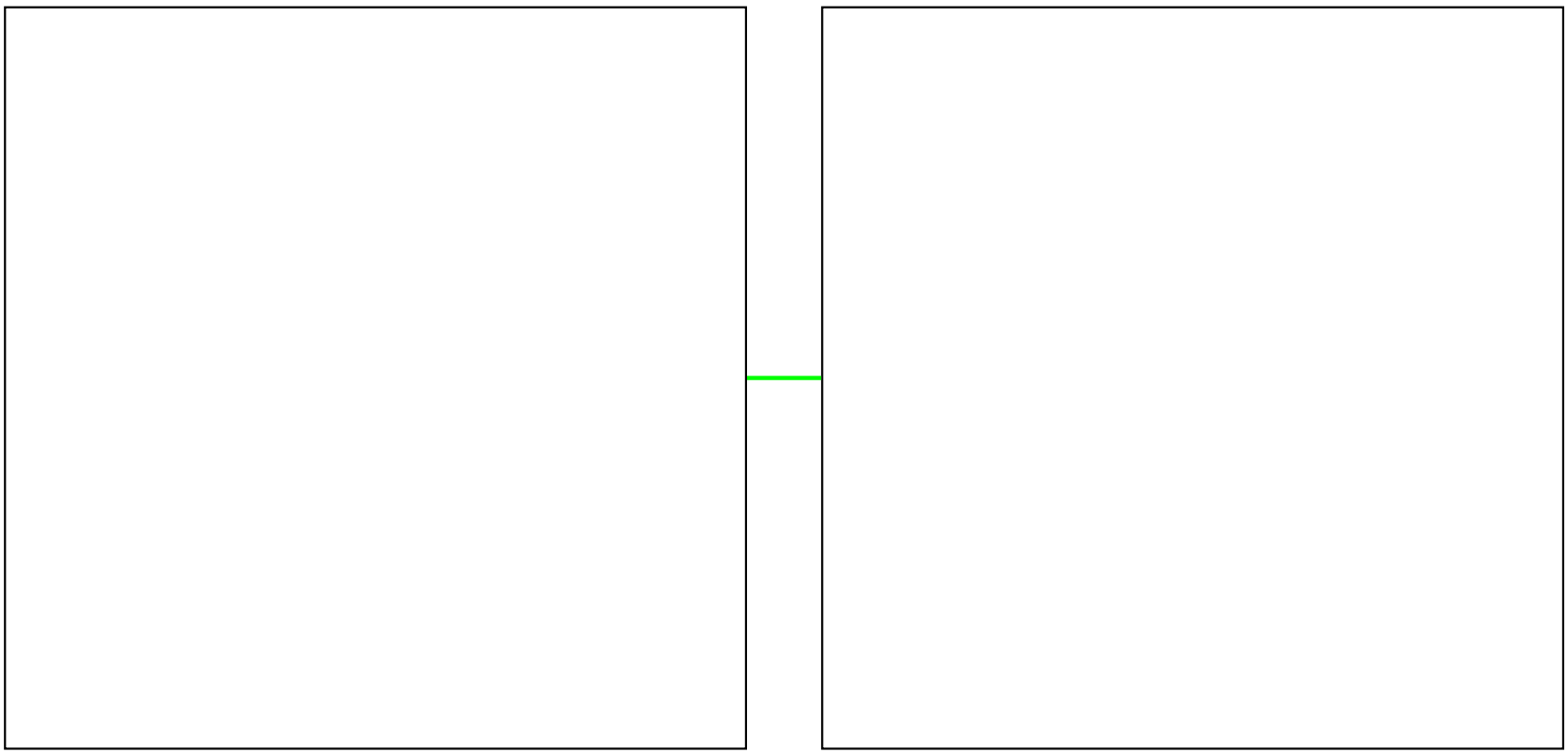
Modularization

Design complex system from smaller parts

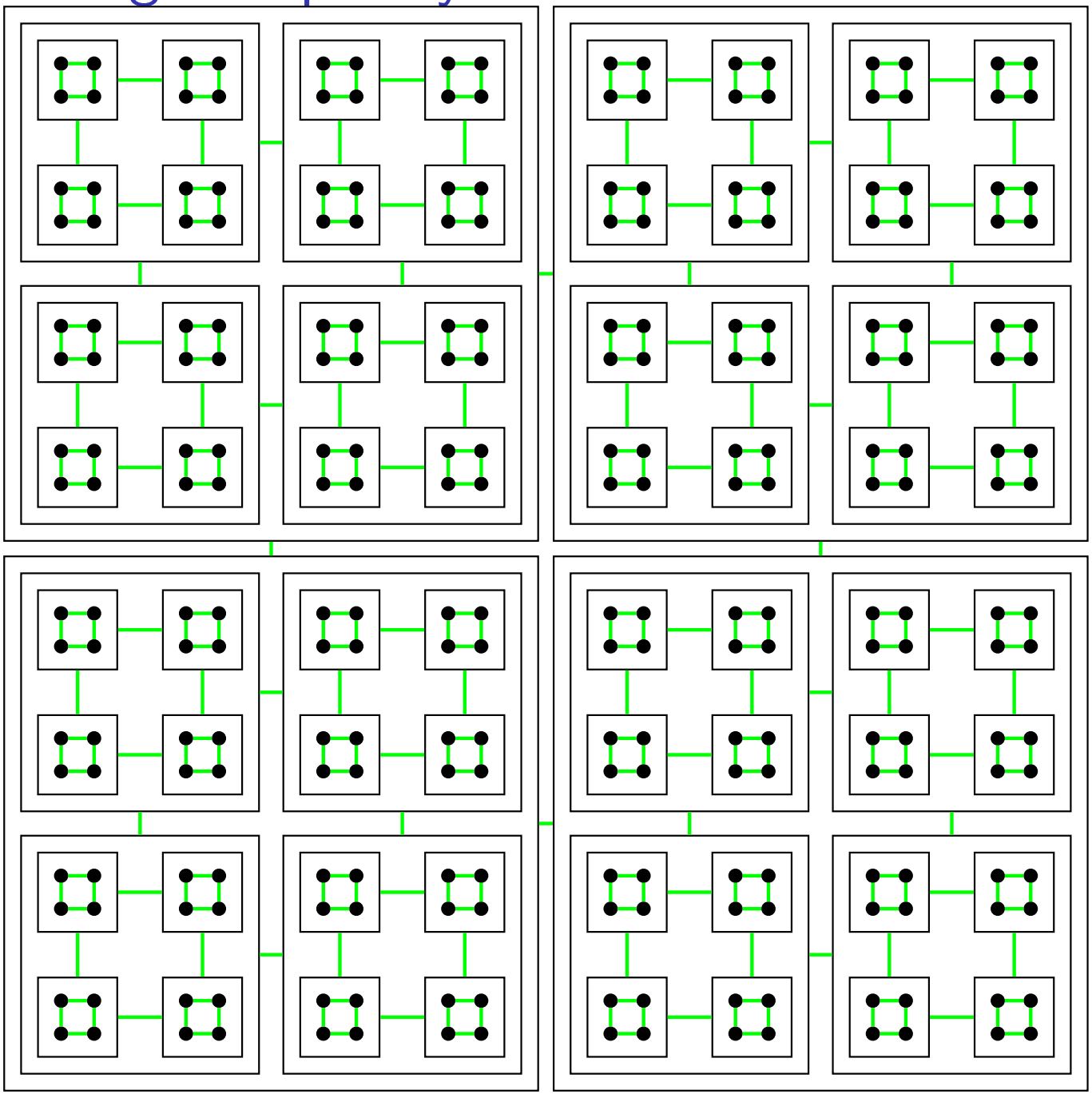


Modularization

Design complex system from smaller parts



Hiding Complexity



Advantages

- separates independent functionality
- decreases complexity
- increases re-usability
- provides natural parallelization of work
- ...

https://en.wikipedia.org/wiki/Modular_programming

Advantages

- separates independent functionality
- decreases complexity
- increases re-usability
- provides natural parallelization of work
- ...

Only way to design complex systems!

https://en.wikipedia.org/wiki/Modular_programming

Advantages

- separates independent functionality
- decreases complexity
- increases re-usability
- provides natural parallelization of work
- ...

Only way to design complex systems!

But needs to be done right!

https://en.wikipedia.org/wiki/Modular_programming

Five Essential Elements of Modular Design

Purpose	each module has well-defined functionality
Interface	well-defined and documented
Implementation	correct, tested, performant, minimal
Encapsulation	don't break the interface!
Connection	minimize dependencies between modules

In Practice

- Multiple layers to structure software system
- In Java
  - method
  - class / interface
  - package
  - archive (JAR)

Metrics for Modularization

Coupling degree of interdependence between modules [WP]

Cohesion degree to which components inside module belong together [WP]

Metrics for Modularization

Coupling degree of interdependence between modules [WP]

Cohesion degree to which components inside module belong together [WP]

Good modularization: low coupling and high cohesion

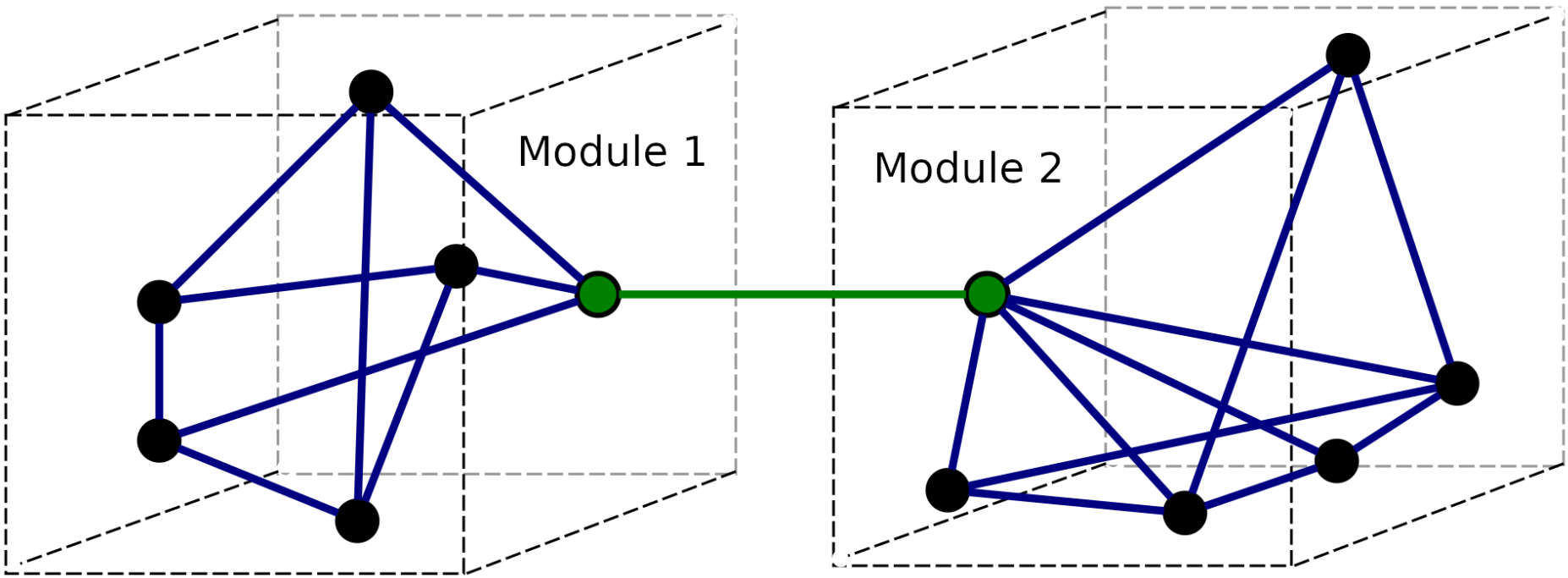


Image source

Metrics for Modularization

Coupling degree of interdependence between modules [WP]

Cohesion degree to which components inside module belong together [WP]

Good modularization: low coupling and high cohesion

Bad modularization: high coupling and low cohesion

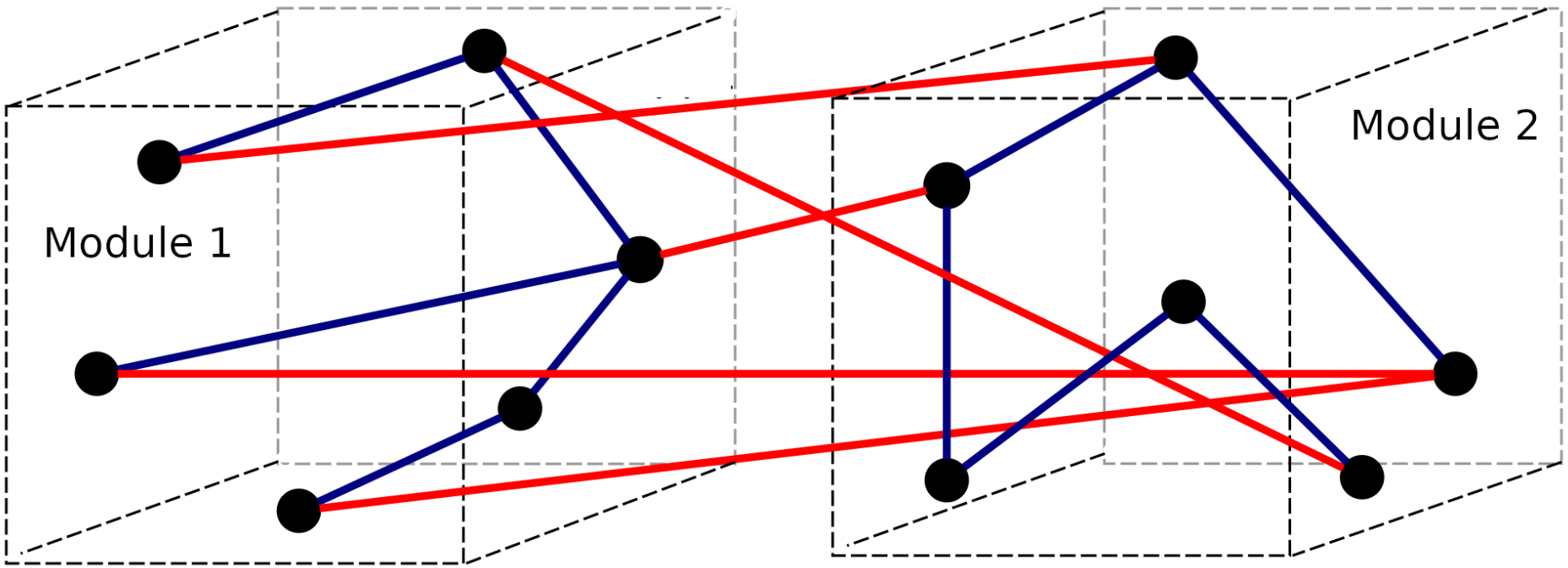


Image source

Classical Types of Coupling

(from low (good) to high (bad))

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

```
double avg(double [] xs)
```

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

```
double getAverageAge(Customer [] xs)
```

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

control coupling pass parameters to control module's behaviour

```
void setHtmlOutput(boolean enabled)
```

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

control coupling pass parameters to control module's behaviour

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

control coupling pass parameters to control module's behaviour

common coupling modules share global data

- Avoid if possible!

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

control coupling pass parameters to control module's behaviour

common coupling modules share global data

Classical Types of Coupling

(from low (good) to high (bad))

data coupling communication with (primitive type) parameters

stamp coupling communication with complex structures/classes

control coupling pass parameters to control module's behaviour

common coupling modules share global data

content coupling module refers to inside of other module

- Don't ever do that!
- In Java: prevent by correct public/private declarations!

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional single, well-defined function

```
class BasicArith {
  static double add(double a, double b)
  static double sub(double a, double b)
  ...
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional single, well-defined function

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- functional: single, well-defined function
- sequential: sequence of functions, output $\rightarrow$ input related

```
class MyCompiler {
  static Tokens lex(File f);
  static AST parse(Tokens tks);
  static AST annotate(AST ast);
  static Binary codegen(AST ast);
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- functional single, well-defined function
- sequential sequence of functions, output $\rightarrow$ input related

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- **functional** single, well-defined function
- **sequential** sequence of functions, output → input related
- **communicational** functions on same body of data

```
class CustomerDB {
  double getAverageSpending();
  Customer getByName(String name);
  void delete(Customer c);
  ...
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional single, well-defined function

sequential sequence of functions, output $\rightarrow$ input related

communicational functions on same body of data

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- **functional** single, well-defined function
- **sequential** sequence of functions, output $\rightarrow$ input related
- **communicational** functions on same body of data
- **procedural** functions related to task of software

```
class BookLending {
  Customer lookupCustomer(int id);
  Book lookupBook(int id);
  bool canLoan(Customer, Book);
  Loan createLoan(Customer, Book);
  void registerLoan(Loan)
  void displayLoan(Loan);
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional single, well-defined function

sequential sequence of functions, output $\rightarrow$ input related

communicational functions on same body of data

procedural functions related to task of software

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- **functional** single, well-defined function
- **sequential** sequence of functions, output $\rightarrow$ input related
- **communicational** functions on same body of data
- **procedural** functions related to task of software
- **temporal** functions, executed at same time in program

```
class PostmortemOperations {
  void closeOpenFiles();
  void createErrorLog();
  void notifyUser();
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional	single, well-defined function
sequential	sequence of functions, output $\rightarrow$ input related
communicational	functions on same body of data
procedural	functions related to task of software
temporal	functions, executed at same time in program

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

- **functional** single, well-defined function
- **sequential** sequence of functions, output $\rightarrow$ input related
- **communicational** functions on same body of data
- **procedural** functions related to task of software
- **temporal** functions, executed at same time in program
- **logical** functions that technically do the same

```
class UserNotifications {
  void displayBooking(Booking);
  void displayFlightDetails(Flight);
  void displayLoginScreen();
}
```

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional	single, well-defined function
sequential	sequence of functions, output $\rightarrow$ input related
communicational	functions on same body of data
procedural	functions related to task of software
temporal	functions, executed at same time in program
logical	functions that technically do the same

Classical Types of Cohesion

(from high (good) to low (bad))

Elements are grouped b/c they perform ...

functional	single, well-defined function
sequential	sequence of functions, output $\rightarrow$ input related
communicational	functions on same body of data
procedural	functions related to task of software
temporal	functions, executed at same time in program
logical	functions that technically do the same
coincidental	functions that are not further related

```
class Miscellaneous {
  void thisFitsNowhereElse();
  void funThatIdidntKnowWhereToPut();
  void wasToLazyToCreateModuleForThat();
  ...
}
```

Conclusions

- Modular design essential for complex systems
- Strive for low coupling and high cohesion
- Don't over-design!