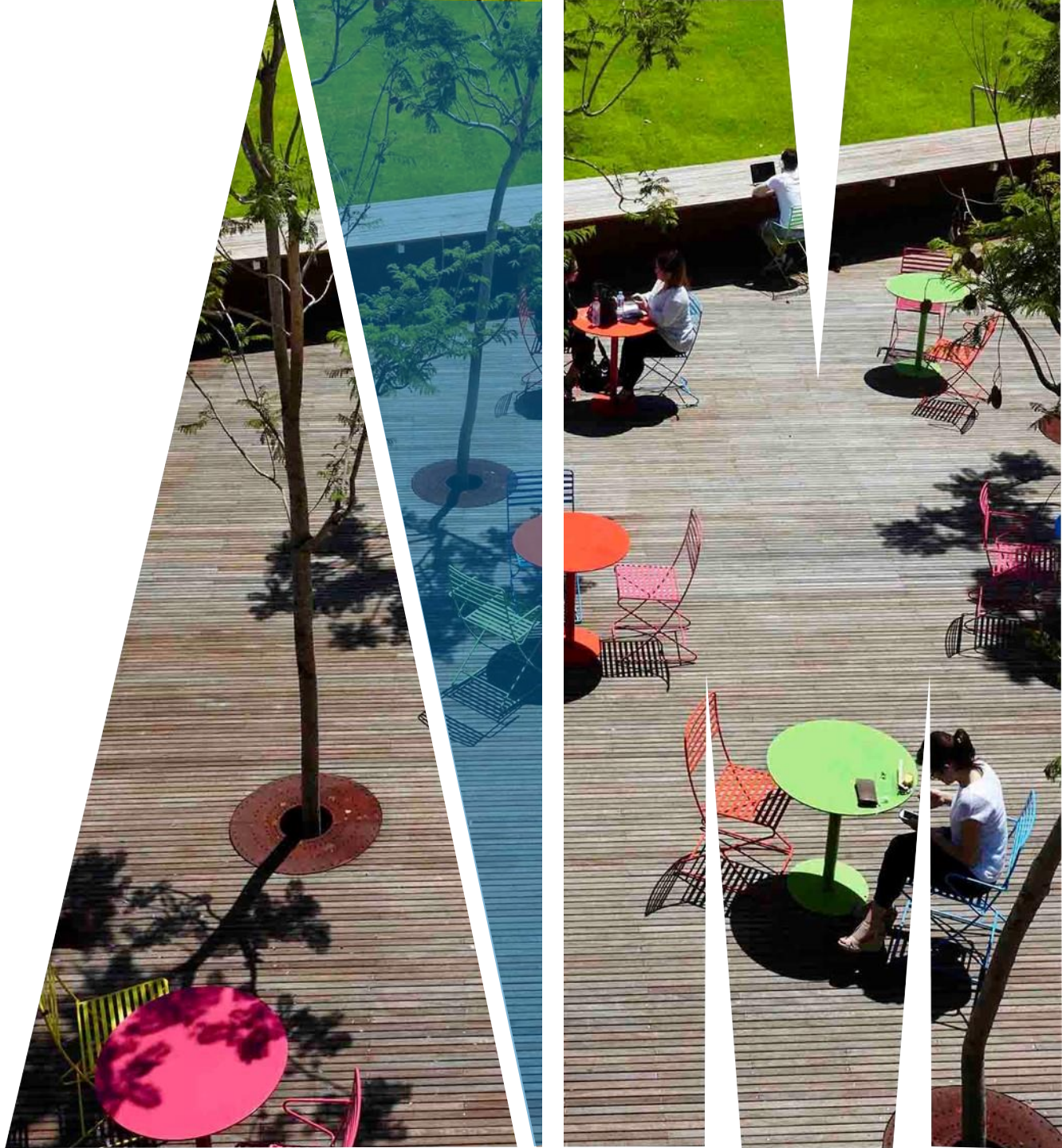
MONASH University

FIT2099 Object-Oriented Design and Implementation

Connascence

(Part 1): Definitions and static connascence



MONASH University

Outline

What is Connascence?

Static Connascence

Dynamic Connascence

FORMAL CHARACTERISTICS OF
GOOD SOFTWARE DESIGN

Characteristic	Definition
Simplicity	The amount and type of software elements needed to solve a problem
Coupling	Measure of interdependence between two or more modules
Cohesion	The degree to which a software module is strongly related and focused in its responsibilities
Information hiding	A component encapsulates its behaviors and data, hiding the implementation details from other components
Performance	The analysis of an algorithm to determine its performance in terms of time (i.e., speed) and space (i.e., memory usage).
Security	A set of technical controls intended to protect and defend information and information systems

Voorhees D.P. (2020) **Characteristics of Good Software Design**. In: Guide to Efficient Software Design. Texts in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-030-28501-2_11 (open resource)

WHAT IS CONNASCENCE?

Described by Meilir Page-Jones in early 1990s.

- based on earlier ideas of **cohesion** and **coupling**
- **cohesion**: how strongly do elements ***within* a code unit** depend on each other?
- **coupling**: how strongly do elements ***between different* code units** depend on each other

Connascence is about the way elements in an object-oriented design depend on each other

This is *one* way to think about **dependencies**

- but it's a useful one

Here, we present them in order from "weakest" to "strongest"

- a rule of thumb only

WHAT IS CONNASCENCE?

*I say that two elements of software are connascent if they are "**born together**" in the sense that they somehow share the same destiny.*

*More explicitly, I define two software elements A and B to be connascent if there is **at least one change that could be made to A that would necessitate a change to B** in order to preserve overall correctness.*
– Meilir Page-Jones

TYPES OF
CONNASCENCE

Static

- obvious **from code structure**
- can be **automatically identified** by IDE/analysis tools

Check: https://connascence.io/

TYPES OF
CONNASCENCE

Dynamic

- only obvious from **close inspection/execution**
- **can't be (easily)** identified by IDE
- generally, **more concerning**

Check: https://connascence.io/

CONNASCENCE OF NAME (CoN)

One of the weakest forms of connascence.

When two or more components must agree on the **name** of an entity.

```
public String getTitles(){
    String titles="";
    //Code to create the String
    return titles;
}
```

Connascence of name

```
public static void main(String[] args) {
    System.out.println("The movie titles are "
            + getTitles());
}
```

LEVELS OF CONNASCENCE

Static

- Name
- Type
- Meaning
- Position
- Algorithm

Dynamic

- Execution
- Timing
- Value
- Identity

CONNASCENCE OF TYPE (CoT)

When two or more components must agree on the **type** of an entity.

```
public void printRentalStatement(){
    Date rentalDate = new GregorianCalendar(2014, Calendar.FEBRUARY, 11).getTime();
    int rentalDays = rentalDaysSince(rentalDate);
    System.out.println("Article rented for" + rentalDays + "days");
}
```

Connascence of type

Connascence of type

```
public int rentalDaysSince(Date date){
    int numberOfDays=0;
    //Code to make calculations
    return (numberOfDays);
}
```

CONNASCENCE OF
MEANING/CONVENTION (CoM/CoC)

CoC is present when the **interpretation** of data in two elements **must be identical**.
CoC is when multiple components must agree on the interpretation of data values.
Basically: **magic numbers, magic strings, null/None, booleans**, etc.

```
public void displayRentalStatement(){
    double totalAmount= 0;
    //Code to retrieve rental articles
    switch (article.type){
        case "regular":
            totalAmount += ar
            break;
        case "discounted":
            totalAmount += article.price * .70;
            break;
    }
    //more code
    if (totalAmount > 200){
        '/Code to do something
```

Connascence of meaning

This means that case "regular" and case “discounted" will appear in at least two elements.

CONNASCENCE OF
MEANING/CONVENTION (CoM/CoC)

CoC is present when the **interpretation** of data in two elements **must be identical**.
CoC is when multiple components must agree on the interpretation of data values.
Basically: **magic numbers, magic strings, null/None, booleans**, etc.

```
public void displayRentalStatement(){
    double totalAmount= 0;
    //Code to retrieve rental articles
    switch (article.type){
        case "regular":
            totalAmount += article.price;
            break;
        case "discounted":
            totalAmount += article.price * .70;
            break;
    }
    //more code
    if (totalAmount > 200){
        '/Code to do something
```

What is the meaning behind “200” or “.70”? This is called a “Magic number” and can point to a code smell.

Connascence of meaning

CONNASCENCE OF
MEANING/CONVENTION (CoM/CoC)

CoC is present when the interpretation of data in two elements must be identical.
CoC is when multiple components must agree on the interpretation of data values.
Basically: magic numbers, magic strings, null/None, booleans, etc.

```
public void displayRentalStatement(){
    double totalAmount= 0;
    //Code to retrieve rental articles
    switch (article.type){
        case "regular":
            totalAmount += article.price;
            break;
        case "discounted":
            totalAmount += article.price * .70;
            break;
    }
    //more code
    if (totalAmount > 200){
        '/Code to do something
```

Compilers *can't* check for this sort of relationship

LEVELS OF CONNASCENCE

Static

- Name ✓
- Type ✓
- Meaning ✓
- Position
- Algorithm

Dynamic

- Execution
- Timing
- Value
- Identity

Connascence is getting stronger

CONNASCENCE OF POSITION (CoP)

When it matters ***where*** something is

E.g.: need to know which arguments can go where

- compiler can help if arguments are of different types

```
public void saveClient(String firstName, String lastName, String address){
    //Code to save a client
}
```

Connascence of position

```
client.saveClient("Michael", "Doe", "Melbourne");
```

CONNASCENCE OF POSITION (CoP)

This can also occur if return values are **NOT in an object**.

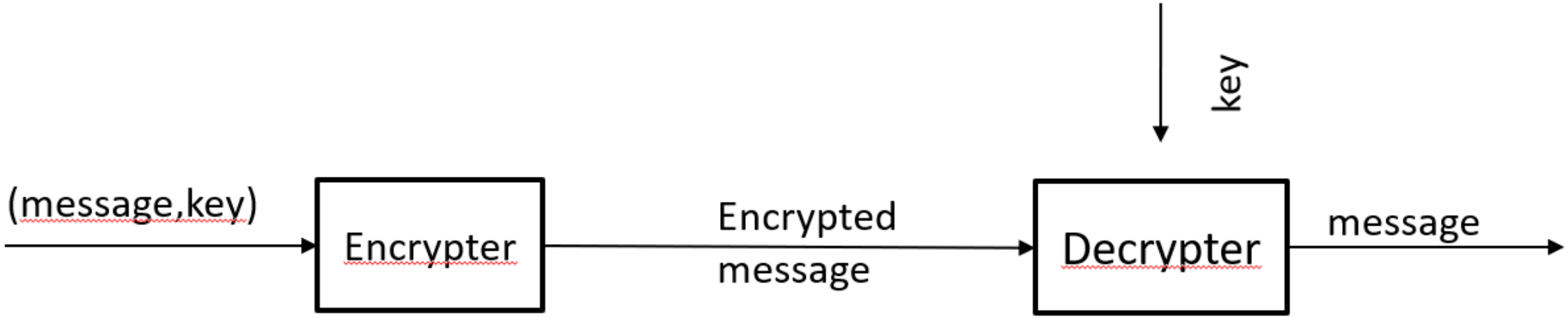
```
public int[] getFrequentClientPoints(int customerId, int movies){
    //Code to calculate the points
    int[] intArray = new int[2];
    intArray[0] = 5000; //total frequent client points
    intArray[1] = 250; // average frequent client points
    return (intArray);
    }
```

Connascence of position

```
public void printRentalStatement(int customerId){
    //Code to print the statement
    int[] frequentPoints = getFrequentClientPoints(customerId,movies)
    int totalPoints= frequentPoints[0];
    int averagePoints= frequentPoints[1];
}
```

CONNASCENCE OF
ALGORITHM (CoA)

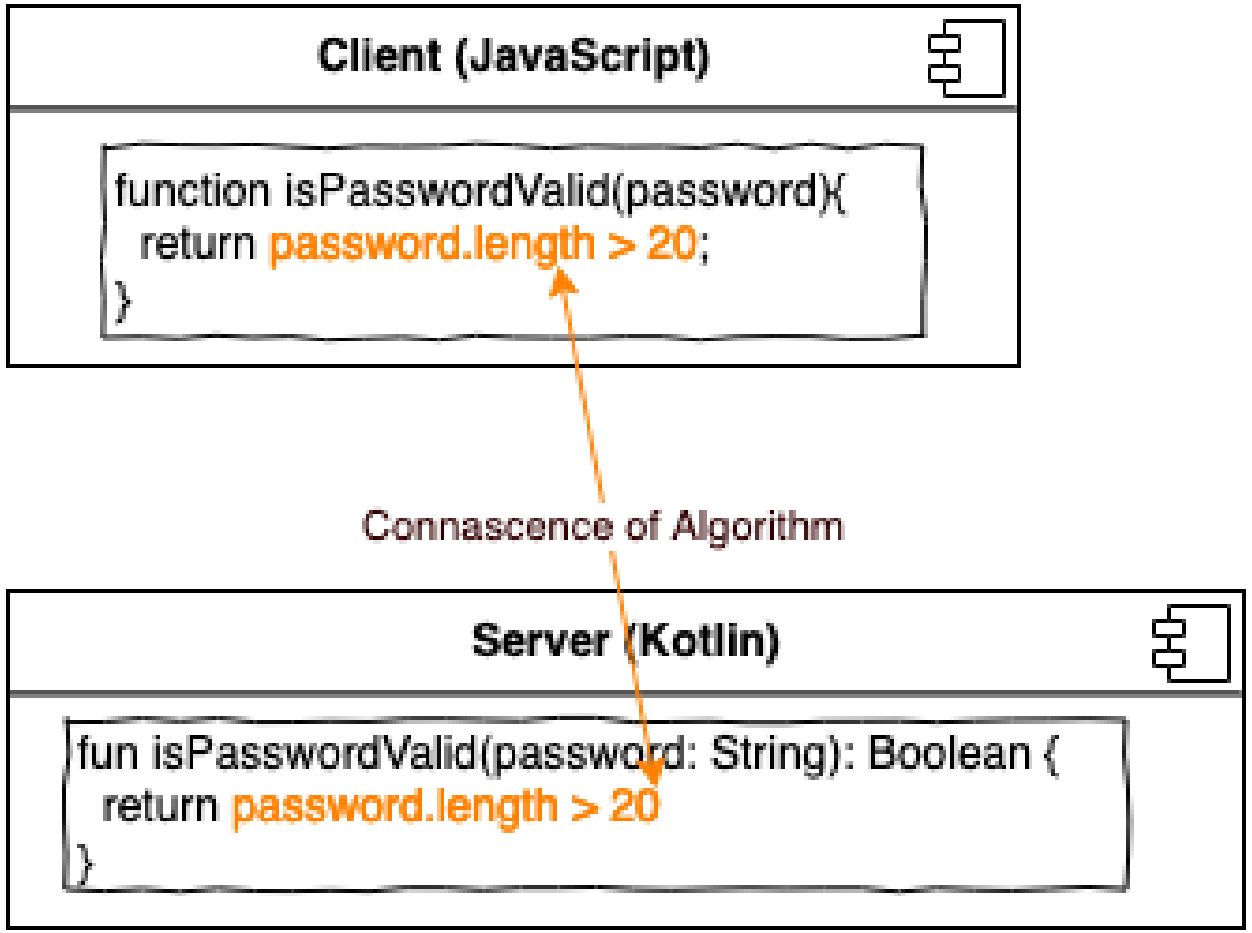
This connascence means that two components must agree **on a particular algorithm in order to work correctly.** Examples are encryption, e.g. SHA-2, SHA512, etc. If the sender changes it's encryption algorithm, the receiver has also to change it's decryption algorithm.



Encrypter and decrypter must implement the same crypto algorithm

CONNASCENCE OF
ALGORITHM (CoA)

CoA also occurs when two elements view or manipulate data **in the same way**.



LEVELS OF CONNASCENCE

Static

- Name ✓
- Type ✓
- Meaning ✓
- Position ✓
- Algorithm ✓

Dynamic

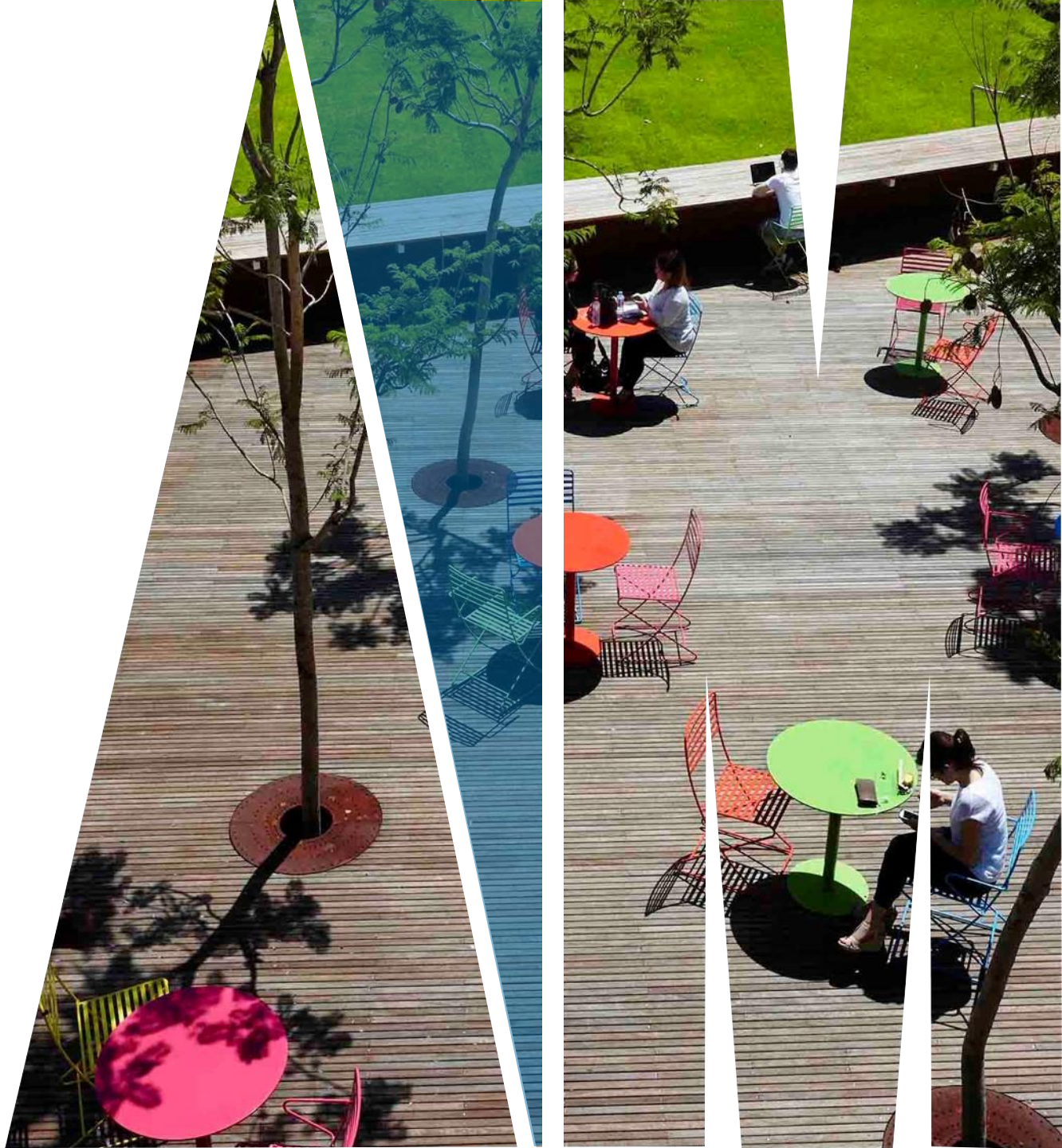
- Execution
- Timing
- Value
- Identity

Connascence is getting stronger

MONASH University

FIT2099 Object-Oriented Design and Implementation

Connascence (Part 2): Dynamic connascence



MONASH University

CONNASCENCE OF
EXECUTION (ORDER) (CoE)

When things need to be executed in a particular **order** if they are to work

Connascence of execution

```
email = new Email();
email.setRecipient("one@example.com");
email.setSender("me@monash.edu");
email.send();
email.setSubject("Hello World"); ?
```

CONNASCENCE OF
EXECUTION (ORDER) (CoE)

Saving an object **AFTER** it has been updated.

Connascence of execution

```
public void changeProductName(int id, String newTitle){
    Product product = getProduct(id);
    product.setTitle(newTitle);
    product.setUpdatedOn(LocalDateTime.now());
    save(product);
    }
```

CONNASCENCE OF TIMING (CoT)

When two events must happen with constraints on **timing**

This is not easy to show

Typically occurs in:

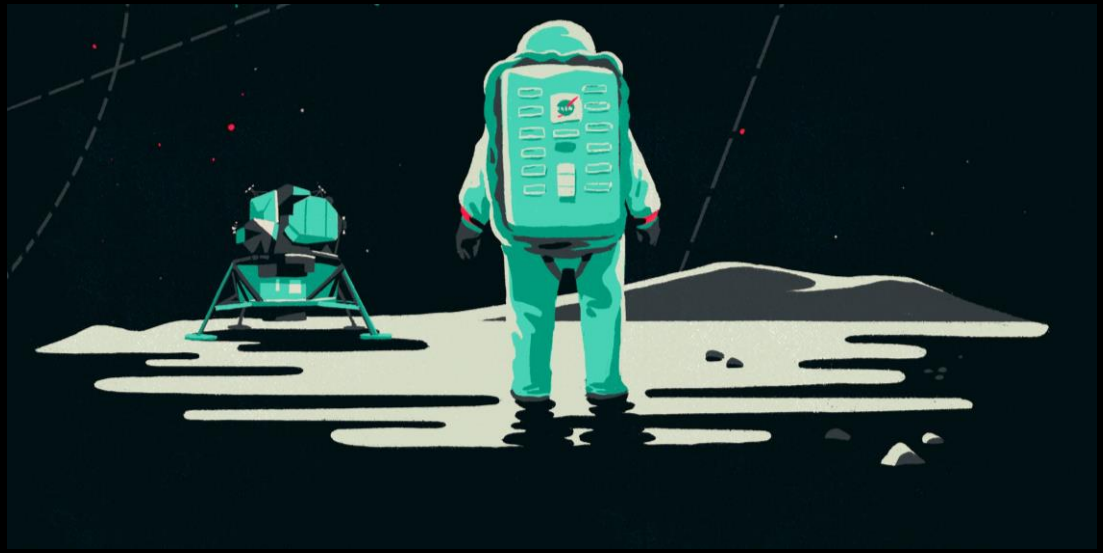
- **parallel** computing
- interacting with **hardware**
  - especially real-time computing
- **distributed** computing

Famous example: Apollo 11 lunar module guidance computer

CONNASCENCE OF
TIMING (CoT)

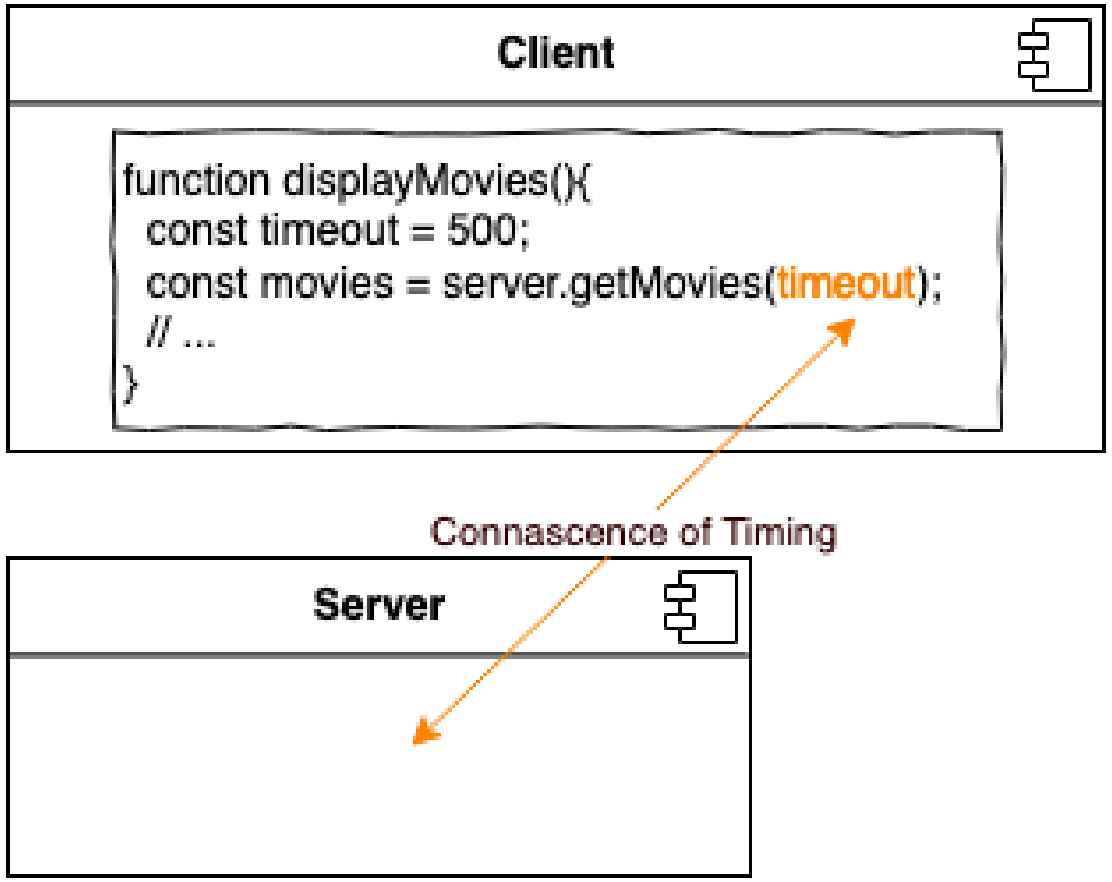
Famous example: Apollo 11 lunar module guidance computer

Memory overload



Connascence of timing (CoT)

TIMING (CoT)



CONNASCENCE OF VALUES (CoV)

When two values in the system are semantically linked

- so if you change one, the other might need to change

```
public void customerReturnProduct(int productId, int userID){
    // some code
    customerStatistics.productsReturnedByCustomer -= 1;
    }
```

Connascence of values

```
public void returnProduct(int productId){
    // some code
    globalStatistics.productsReturnedByAllCustomers -= 1;
}
```

CONNASCENCE OF VALUES (CoV)

When two values in the system are **semantically linked**

– so if you change one, the other might need to change

```
public class Unit {

    ...
    private HashMap<Integer, Student> enrolledStudents = new
HashMap<Integer, Student>();
    ...
    public void enrolStudent(Student student) {
        enrolledStudents.put(student.getStudentId(), student);
    }
```

map key and id attribute of student must be equal – and stay that way

If you ***ever*** change the student ID of a student, you're going to have to change it in many, many places.

LEVELS OF CONNASCENCE

Static

- Name ✓
- Type ✓
- Meaning ✓
- Position ✓
- Algorithm ✓

Dynamic

- Execution ✓
- Timing ✓
- Value ✓
- Identity

CONNASCENCE OF IDENTITY (CoI)

CoI is the **strongest form** and occurs when the same object must be referenced at two or more locations.

This often happens when you obtain **an object from a database in different parts of your code** and want to **update** that object. The update must be in consistent once it is put back again in the database.

CONNASCENCE OF IDENTITY (CoI)

```
public void userRequestedTitleChange(Product product, String newTitle){
    changeProductTitle(product.id, newTitle);
    displayProductInformation(product);
}
```

How do you know these objects are the same?

```
public void changeProductTitle(id product, String newTitle){
    Product product = getProduct(id);
    product.setTitle(newTitle);
    product.updatedOn(LocalTime.now());
    save(movie);
}
```

CONNASCENCE OF IDENTITY (CoI)

```
public class Person {
   private String name;
   private Set<Person> parents;

   public Person(String name, Person parenta,
                  Person parentb) {
      this.name = name;
      parents = new HashSet<Person>();
      parents.add(parenta);
      parents.add(parentb);
   }

   public Set <Person> getParents() {
      return parents;
   }

}
```

CONNASCENCE OF IDENTITY (CoI)

```
public class Person {
   private String name;
   private Set<Person> parents;

   public Person(String name, Person parenta,
                  Person parentb) {
      this.name = name;
      parents = new HashSet<Person>();
      parents.add(parenta);
      parents.add(parentb);
   }

   public Set <Person> getParents() {
      return parents;
   }

   public boolean isSibling(Person a) {
      for (Person parent : parents) {
         if (a.getParents().contains(parent)) {
            return true;
         }
      }
      return false;
  }
}
```

CONNASCENCE OF IDENTITY (CoI)

```java
public class Person {
   private String name;
   private Set<Person> parents;

   public Person(String name, Person parenta,
                 Person parentb) {
      this.name = name;
      parents = new HashSet<Person>();
      parents.add(parenta);
      parents.add(parentb);
   }

   public Set <Person> getParents() {
      return parents;
   }

   public boolean isSibling(Person a) {
      for (Person parent : parents) {
         if (a.getParents().contains(parent)) {
            return true;
         }
      }
      return false;
  }
}
```

```java
public static void main(String[] args) {
   Person gina = new Person("Gina Meares", null, null);
   Person fred = new Person("Fred Meares", null, null);
   Person anna = new Person("Anna Meares", gina, fred);
   Person kerrie = new Person("Kerrie Meares", gina, fred);


}
```

CONNASCENCE OF IDENTITY (CoI)

```java
public class Person {
   private String name;
   private Set<Person> parents;

   public Person(String name, Person parenta,
                 Person parentb) {
      this.name = name;
      parents = new HashSet<Person>();
      parents.add(parenta);
      parents.add(parentb);
   }

   public Set <Person> getParents() {
      return parents;
   }

   public boolean isSibling(Person a) {
      for (Person parent : parents) {
         if (a.getParents().contains(parent)) {
            return true;
         }
      }
      return false;
  }
}
```

```java
public static void main(String[] args) {
   Person gina = new Person("Gina Meares", null, null);
   Person fred = new Person("Fred Meares", null, null);
   Person anna = new Person("Anna Meares", gina, fred);
   Person kerrie = new Person("Kerrie Meares", gina, fred);

   Person gina2 = new Person("Gina Meares", null, null);
   Person fred2 = new Person("Fred Meares", null, null);
   Person kerrie2 = new Person("Kerrie Meares", gina2,
fred2);

}
```

CONNASCENCE OF IDENTITY (CoI)

```java
public class Person {
   private String name;
   private Set<Person> parents;

   public Person(String name, Person parenta,
                 Person parentb) {
      this.name = name;
      parents = new HashSet<Person>();
      parents.add(parenta);
      parents.add(parentb);
   }

   public Set <Person> getParents() {
      return parents;
   }

   public boolean isSibling(Person a) {
      for (Person parent : parents) {
         if (a.getParents().contains(parent)) {
            return true;
         }
      }
      return false;
  }
}
```

```java
public static void main(String[] args) {
   Person gina = new Person("Gina Meares", null, null);
   Person fred = new Person("Fred Meares", null, null);
   Person anna = new Person("Anna Meares", gina, fred);
   Person kerrie = new Person("Kerrie Meares", gina, fred);

   Person gina2 = new Person("Gina Meares", null, null);
   Person fred2 = new Person("Fred Meares", null, null);
   Person kerrie2 = new Person("Kerrie Meares", gina2,
fred2);

   if (anna.isSibling(kerrie)) {
      System.out.println("Sisters rule");
   }

   if (anna.isSibling(kerrie2)) {
      System.out.println("Duplicate sisters too?");
   }
}
```

In this example, to be considered **siblings** the parent references must point to exactly the same person object. If they are duplicates, no deal!

WHY CONNASCENCE MATTERS?

Recall definition: *More explicitly, I define two software elements A and B to be connascent if* ***there is at least one change that could be made to A that would necessitate a change to B in order to preserve overall correctness***

So… more connascence means:

- **harder to extend**
- more chance of **bugs**
- **slower** to write in the first place

WHY
CONNASCENCE MATTERS?

Summary

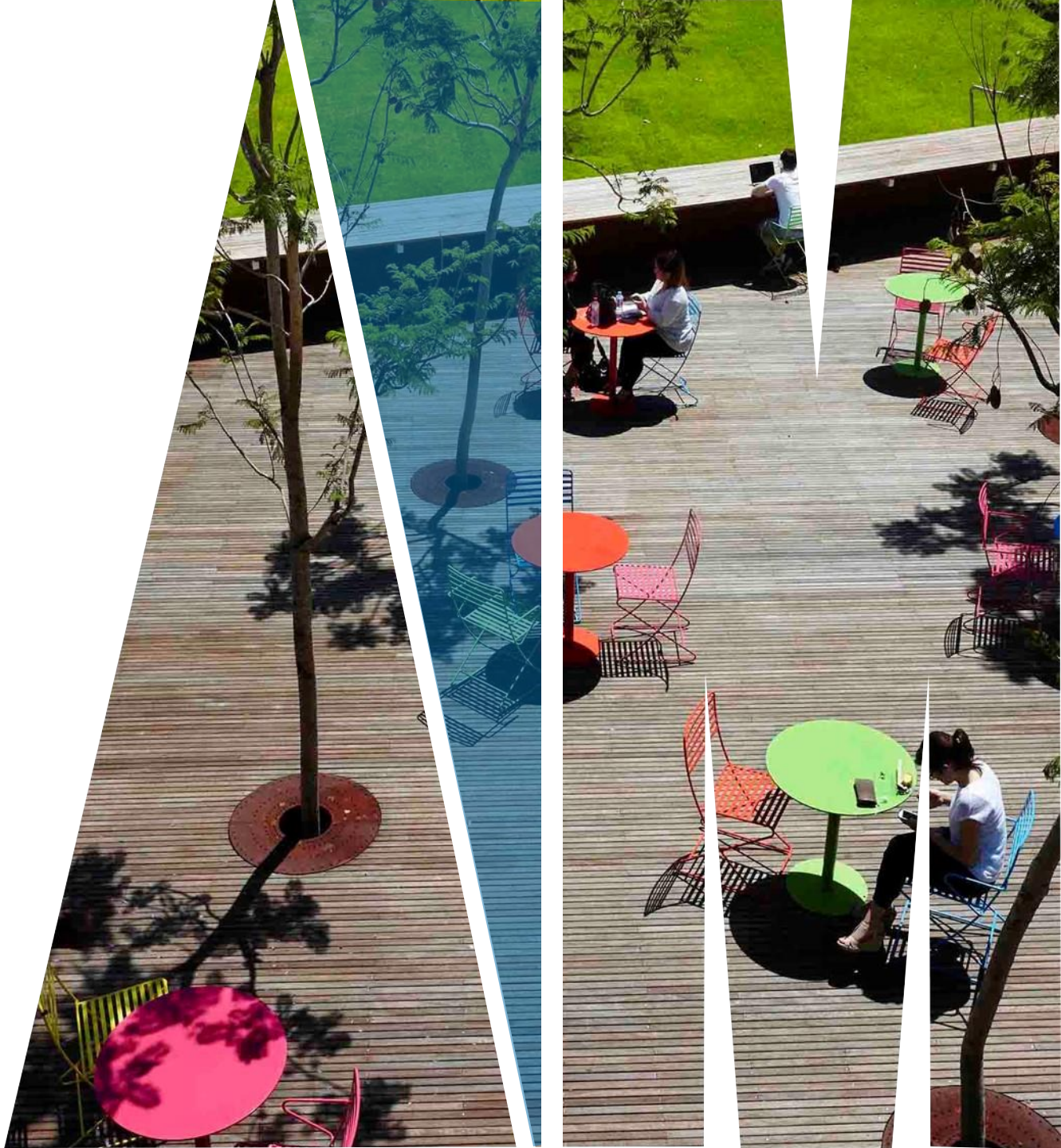
What is Connascence?

Static Connascence

Dynamic Connascence

MONASH University

Thanks



MONASH University