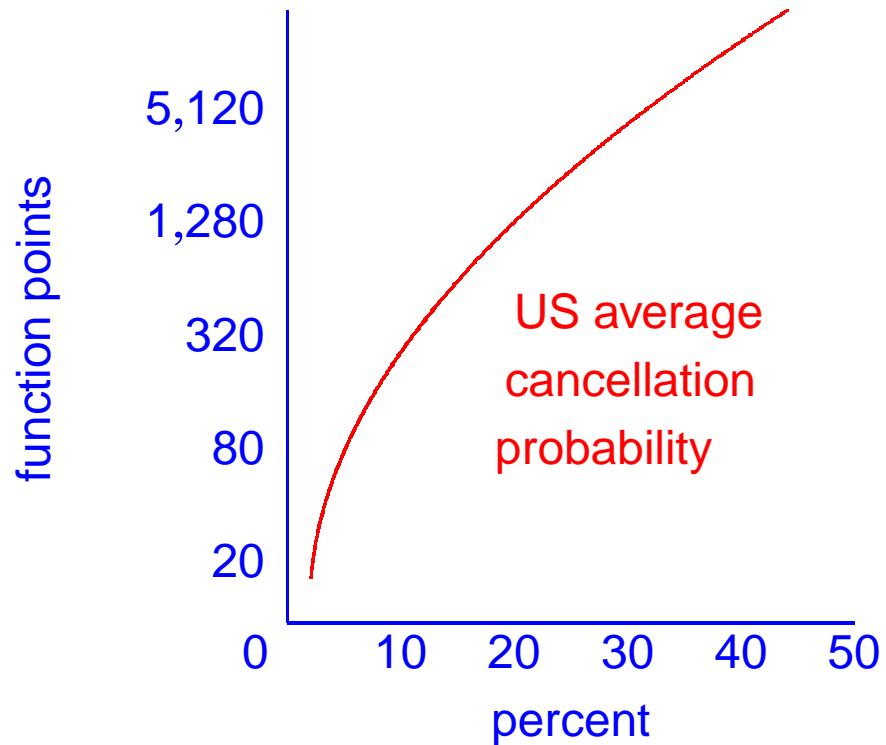


# Decomposition

Size and complexity are the enemy.

**decomposition:** combats size

**abstraction:** combats complexity



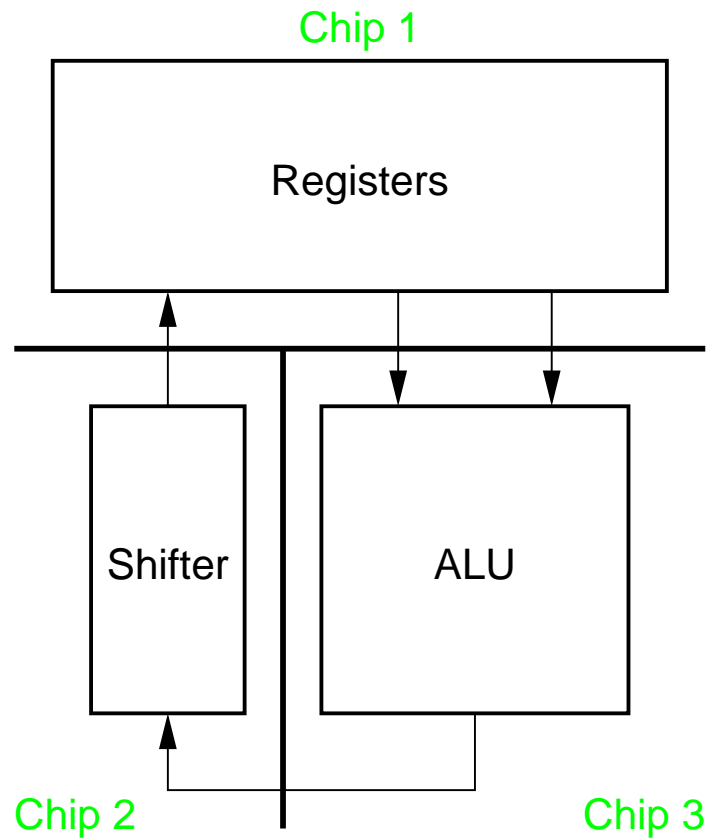
# Decomposition

- breaks a large project into smaller pieces
- the smaller pieces are easier to understand
- different pieces can be given to different programmers  
(parallel processing)

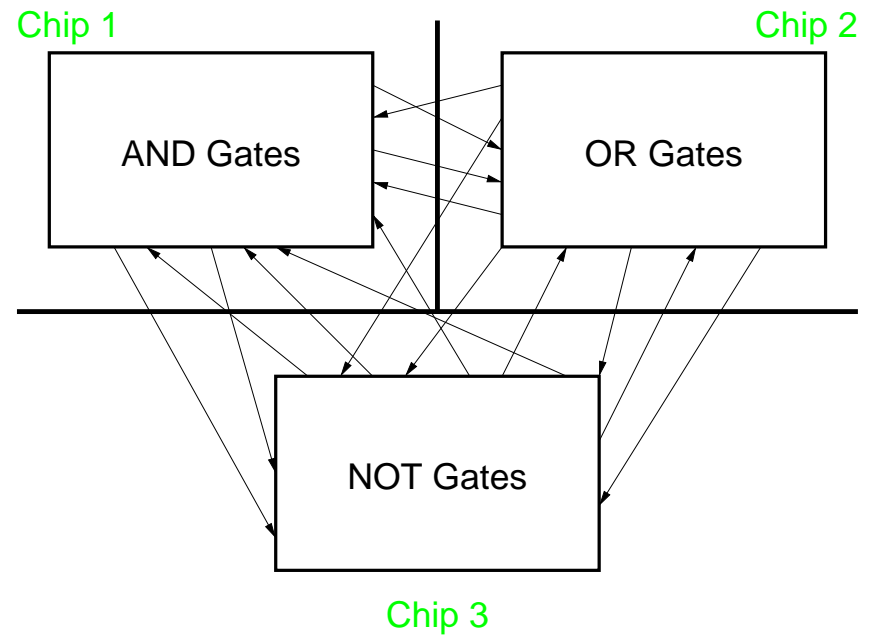
From the Macquarie Dictionary:

**decompose** /dikəm'pɒʊz/, *v.*, **-posed**, **-posing**, *-v.t.* **1.** to separate or to resolve into constituent parts or elements; disintegrate. *-v.i.* **2.** to rot; putrefy.  
[F *décomposer*, from *dé-* DIS-<sup>1</sup> + *composer* COMPOSE]

# MSI CPU Design Example



# MSI CPU Design Error



# Decomposition Evaluation Criteria

**Cohesion:** Interrelation between the various parts of a component

**Coupling:** Interrelation between the various components of a system

**Note:** A *component* can be at the level of a class, or at the level of a routine (procedure or function).

*In general, a system should have high cohesion and low coupling.*

# Cohesion

- ✓ Informational Cohesion
- ✓ Functional Cohesion
- ✗ Communicational Cohesion
- ✗ Procedural Cohesion
- ✗ Temporal Cohesion
- ✗ Logical Cohesion
- ✗ Coincidental Cohesion

## X Coincidental Cohesion

There is no conceptual link between the parts of a component.

Sometimes done in a misguided attempt to satisfy component size guidelines.

```
class EXTRAS
  ...
  feature
    reverse(s: STRING): STRING is
      ...
    factorial (n: INTEGER): INTEGER is
      ...
    palindrome(s: STRING): BOOLEAN is
      ...
  end -- class EXTRAS
```

## X Logical Cohesion

Components contains parts which are logically similar, but otherwise unrelated.

```
class OPEN_UNI_DATABASES
  ...
feature
  open_student_databases is
  ...
  open_unit_databases is
  ...
end -- class OPEN_UNI_DATABASES
```

## X Temporal Cohesion

Parts of a component implement actions that are performed at the same time.

```
class BATMAN_OUTING
```

```
...
```

```
feature
```

```
  illuminate_batcave is
```

```
...
```

```
  turbines_to_speed is
```

```
...
```

```
  open_bat_doors is
```

```
...
```

```
end -- class BATMAN_OUTING
```

```
class CLOSE_BUILDING
```

```
...
```

```
feature
```

```
  shutdown_aircon is
```

```
...
```

```
  shutdown_main_lights is
```

```
...
```

```
  alarm_external_doors is
```

```
...
```

```
end -- class CLOSE_BUILDING
```

## X Procedural Cohesion

The parts of a component are used in order to accomplish a task, but operate on separate data.

```
class ENROLMENT
  ...
feature
  write_student_data_record is
    ...
  write_student_number_in_lab_group is
    ...
end — class ENROLMENT
```

## ✗ Communicational Cohesion

The parts of a component are used in order to accomplish a task, and they operate on a shared data-structure, but the component could split into parts with more reusable, atomic actions.

```
class ACCOUNT
  ...
feature
  deposit_interest_and_print_statement is
  ...
end — — class ACCOUNT
```

## ✓ Functional Cohesion

A component implements a single simple action.

```
class ACCOUNT
  ...
  feature
    deposit_interest is
      ...
  end — class ACCOUNT
```

**Note:** These are typically a property of routine level components.

## ✓ Informational Cohesion

The parts of the component perform actions on the same data-structure.

```
class ACCOUNT
  ...
  feature
    deposit (amount: INTEGER) is
      ...
    withdraw (amount: INTEGER) is
      ...
  end -- class ACCOUNT
```

**Note:** This is typically a property of class level components.

## Coupling

✓ Data Coupling

✗ Stamp Coupling

✗ Control Coupling

✗ Common Coupling

## ✗ Common Coupling

The components share access to global data-structures.

```
class ACCOUNT
  ...
feature
  balance: INTEGER
  ...
end -- class ACCOUNT
```

```
class EMPLOYEE
  ...
feature
  pay is
  do
    ac.balance :=
      ac.balance + wage
  end
  ...
end -- class EMPLOYEE
```

## X Control Coupling

The components communicate by passing parameters that contain explicit control information.

```
class ACCOUNT
  ...
feature
  update_balance (trans_type: INTEGER; amt: INTEGER) is
    do
      inspect trans_type
      when 0 then -- deposit
        balance := balance + amt
      when 1 then -- withdrawal
        balance := balance - amt
      end
    end
```

## ✗ Stamp Coupling

The components communicate by passing data-structures that are only partly relevant.

```
phone_book.add_entry(current_employee)
```

## ✓ Data Coupling

The components communicate only by passing relevant data-structures.

```
phone_book.add_entry(current_employee.last_name,  
                      current_employee.initials ,  
                      current_employee.phone_number)
```