# Parallel Part I

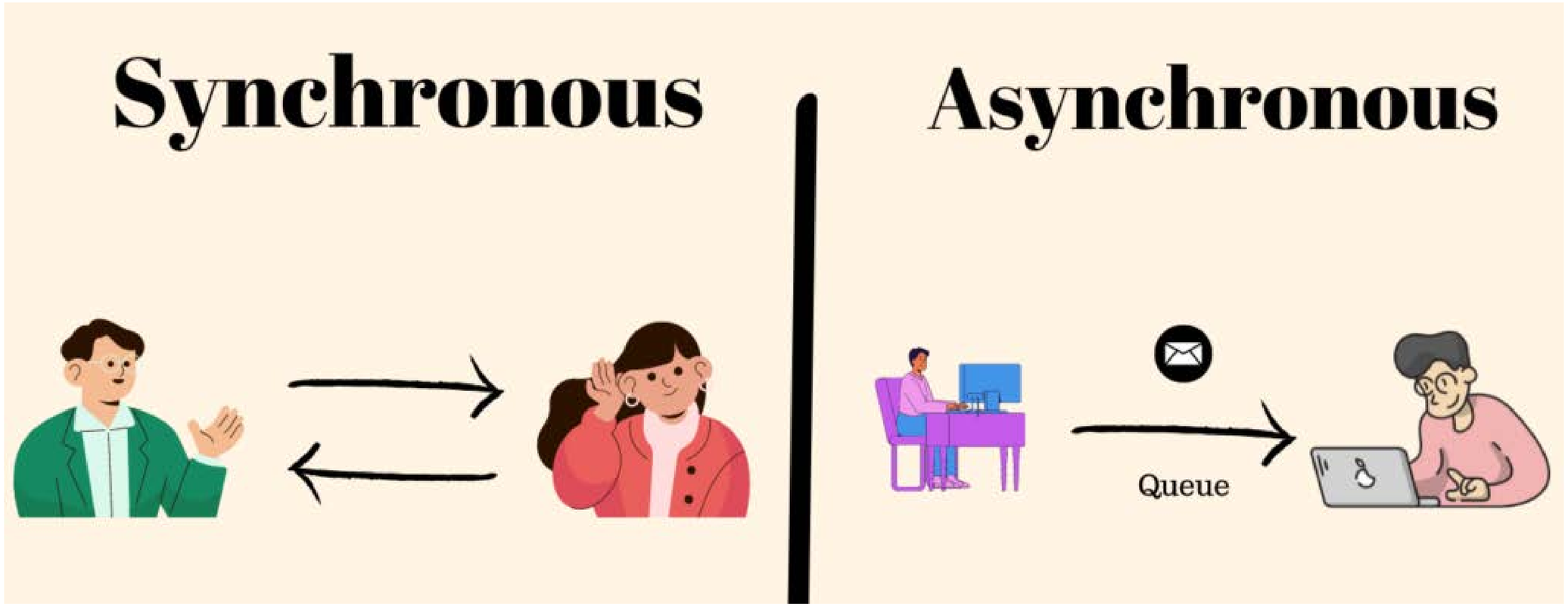
## Basic Tasks & Data Type in Parallel Programming

06016415 Functional Programming

IT KMITL
พระจอมเกล้าลาดกระบัง

# Outline

- Sequential VS Concurrent VS Parallel
- Parallel Mapping and Reduction
- Associativity
- Data Parallelism in Scala Collections

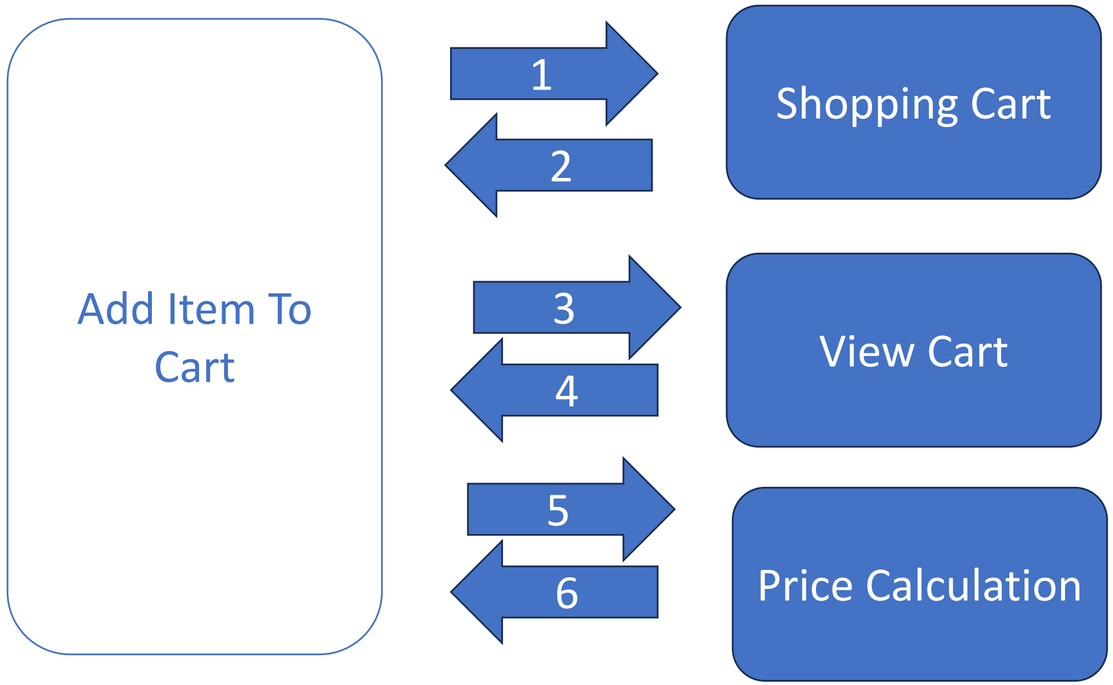



*request/response*

*event-based*

issarapong@it.kmitl.ac.th

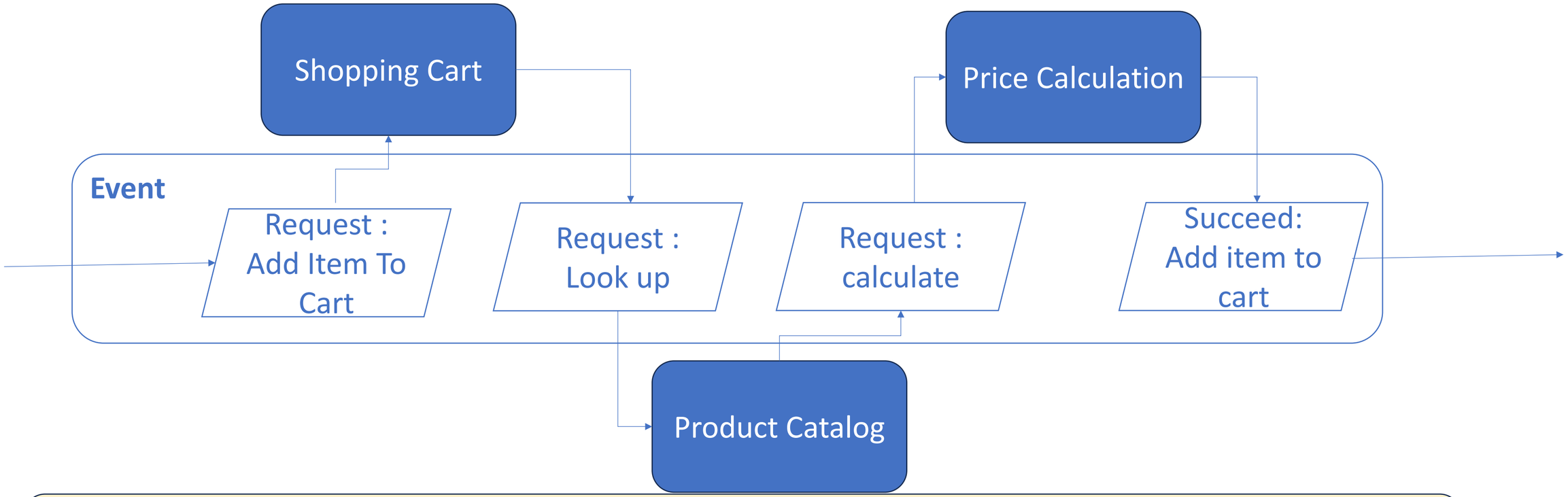
# Synchronous (request/response)




With synchronous communication, a call is made to a remote server, which blocks until the operation completes.

issarapong@it.kmitl.ac.th

# Asynchronous (event-based)




With asynchronous communication, the caller doesn’t wait for the operation to complete before returning and may not even care whether the operation completes at all.

issarapong@it.kmitl.ac.th

## Sequence VS Parallel

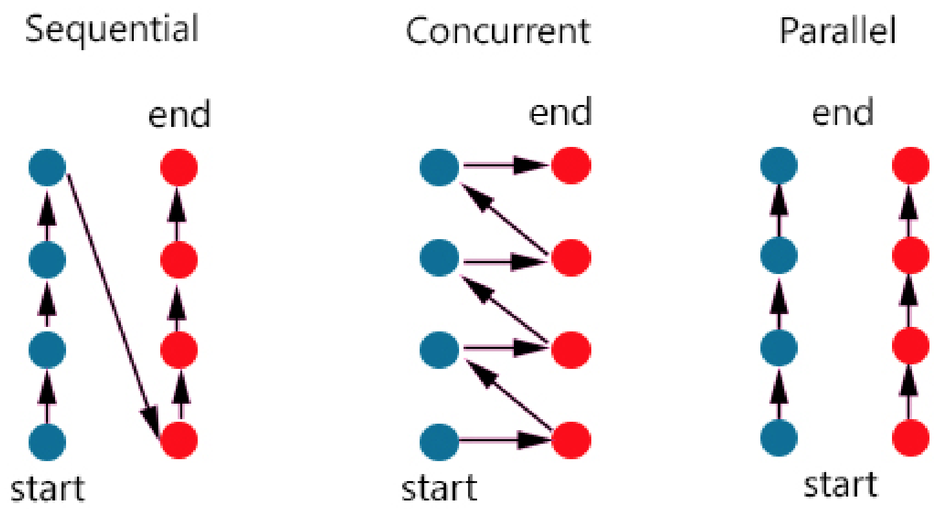
- **Sequential programming:** เป็นการประมวลโปรแกรม (program executing) ทีละคำสั่ง ณ เวลาหนึ่งๆ
- **Parallel programming:** เป็นการประมวลโปรแกรมแบบขนาน โดยมีหลายส่วนดำเนินการไปพร้อมกัน (multiple parts of program execute ) ณ เวลาหนึ่งๆ

- **ข้อดี** คือ ในกรณีที่ทำงานขนานกัน จะใช้เวลาในการรอน้อยลง จนกว่างานที่กำหนดจะเสร็จสิ้น หรือ ทำให้มีประสิทธิภาพสูงขึ้น
- **ข้อเสีย** คือ หากกำหนดผิดพลาดอาจเกิด non-determinism คือ ให้ผลลัพธ์ที่แตกต่างกันในแต่ละครั้งที่ป้อนข้อมูลแบบเดียวกัน

## Concurrency VS Parallel

- **Concurrency programming:** โดยทั่วไป คือ การ ทำให้โครงสร้างของโปรแกรมเป็น งานที่เกิดขึ้นไปพร้อมๆกัน (concurrent tasks) ซึ่งมีการสื่อสารระหว่างกันและกับสภาพแวดล้อมภายนอก
  - ตัวอย่างในกรณี Software เช่น การรับประกันความปลอดภัยในการเข้าถึงทรัพยากรที่ใช้ร่วมกัน
  - ตัวอย่างในกรณี Hardware เช่น OS และ JVM มี threads บน CPU คอร์เดียว

- **Parallel programming:** เป็นการประมวลโปรแกรมแบบขนาน โดยมีหลายส่วนดำเนินการไปพร้อมกัน (multiple parts of program execute ) ณ เวลาหนึ่งๆ ซึ่งจะเร่งความเร็วในการคำนวณแบบขนาน
  - ในกรณี Hardware เราจะใช้วิธีนี้ เพื่อแก้ปัญหาของ concurrency ที่อาจจะเป็นอุปสรรค
  - เช่น จัดการกับปัญหาแบบแยกส่วน isolation และทำให้เกิดงานใหม่ที่ไม่ได้ต้องการ เป็นต้น




**Sequential VS Concurrent VS Parallel**

- **Mapping** : คือ การแบ่งออกเป็นสองส่วน และเรียกใช้การเรียกซ้ำแบบขนานสำหรับทั้งสองส่วน ตามขนาดที่กำหนด โดยสามารถใช้ Mapping แบบอนุกรมได้

- **Folding/reducing :** คือการพับ หรือ การลดโครงสร้างข้อมูล
  - ทำได้ยากกว่าการทำ Mapping เนื่องจากในกรณีทั่วไป การคำนวณจะขึ้นอยู่กับแต่ละส่วน
  - *ตัวอย่าง val a = Array(3,2,1) และเราเรียก a.reduceLeft(_ - _) ซึ่งไม่สามารถดำเนินการแบบขนานได้ เนื่องจากการประเมิน (3-2)-1 ขึ้นอยู่กับการประเมินของ 3-2*

  - **ใช้ Associativity** แทนได้
    - เช่น f(a,f(b,c)) = f(f(a,b),c) for every a, b, and c.
    - def f(a:Int, b: Int) = a+b เป็น Associativity
      - (a+b)+c = a+(b+c)
    - *Associativity?? def g(a:Int, b: Int) = a-b*

# Map

```
val lastNames = List("Smith","Jones","Frankenstein","Bach","Jackson","Rodin").par
println(lastNames.map(_.toUpperCase))
```

# Reduce

```
val strings = List("abc","def","ghi","jk","lmnop","qrs","tuv","wx","yz").par
println(strings)
val alphabet = strings.reduce(_+_)
println(alphabet)
```

# Associativity

- **Associativity** คือ การจัดกลุ่มของการดำเนินการ ซึ่งช่วยให้เราทำสิ่งต่างๆ ควบคู่กันไปได้ เพราะเราสามารถกำหนด ในแบบที่เราต้องการได้ ซึ่งถ้าฟังก์ชัน เป็นแบบเชื่อมโยง ก็จะสามารถดำเนินการได้ ซึ่งมีผลต่อการคำนวณ
  - f(a,f(b,c)) = f(f(a,b),c) for every a, b, and c.
  - def f(a:Int, b: Int) = a+b เป็น Associativity
    - (a+b)+c = a+(b+c)
  - $(A \cup B) \cup C = A \cup (B \cup C)$
  - $A \cup B = B \cup A$
  - Boolean conjunction: (a && b) && c = a && (b && c) and a && b = b && a;
- *?? $(A \times B) \times C = A \times (B \times C)$*
- *?? $A \times B = B \times A$*
- *?? Sum of floats: a+b = b+a but (a+b)+c = a+(b+c)*

# Associativity

ประเภทของ Associativity

- Left Associative ((a op b) op c) → คำนวณจากซ้ายไปขวา
- Right-Associative (a op (b op c)) → คำนวณจากขวาไปซ้าย

- ใน Scala การกำหนด Associativity ของ Operator : (Colon)
- ถ้าชื่อ Operator ลงท้ายด้วย : (Colon) จะเป็น Right-Associative
- ถ้าไม่ลงท้ายด้วย : (Colon) จะเป็น Left-Associative

# Associativity

- **Associativity** มีบทบาทสำคัญในการ **Parallel Computation** เพราะมันช่วยให้เราสามารถแบ่งงานออกเป็นหลายส่วนและรันพร้อมกันได้โดยที่ไม่ต้องกังวลว่าลำดับการคำนวณจะเปลี่ยนผลลัพธ์

- Associativity ช่วยให้ Parallel Computation ปลอดภัยนิยาม:
- ถ้า Operator มีคุณสมบัติ Associativity, เราสามารถ เปลี่ยนกลุ่มของการคำนวณ ได้โดยไม่เปลี่ยนผลลัพธ์กฎ

**การบวก**:(1+2)+3=1+(2+3) = 6

- Associative → สามารถรันแบบขนานได้

- Non-Associative → ต้องรันแบบ Sequential

# Associativity

```
var sum = 0
val list = (1 to 1000).toList.par
print(list.foreach(sum += _))
println(sum)
```

```
val list2 = (1 to 1000).toList.par
println(list2.reduce(_-_))
```

# Associativity

- ถ้าต้องการ Parallel Computation อย่างปลอดภัย → ต้องใช้ Associative Operation

| **Associativity** | **Parallel** ได้? | ตัวอย่าง |
|---|---|---|
| Associative | ใช้ Parallel Computation ได้ | +, *, max, min, &&, \|\| |
| Non-Associative | ใช้ Parallel Computation ไม่ได้ | -, /, ^, :: |

# Data Parallelism

ใช้กับข้อมูลจำนวนน้อย ๆ

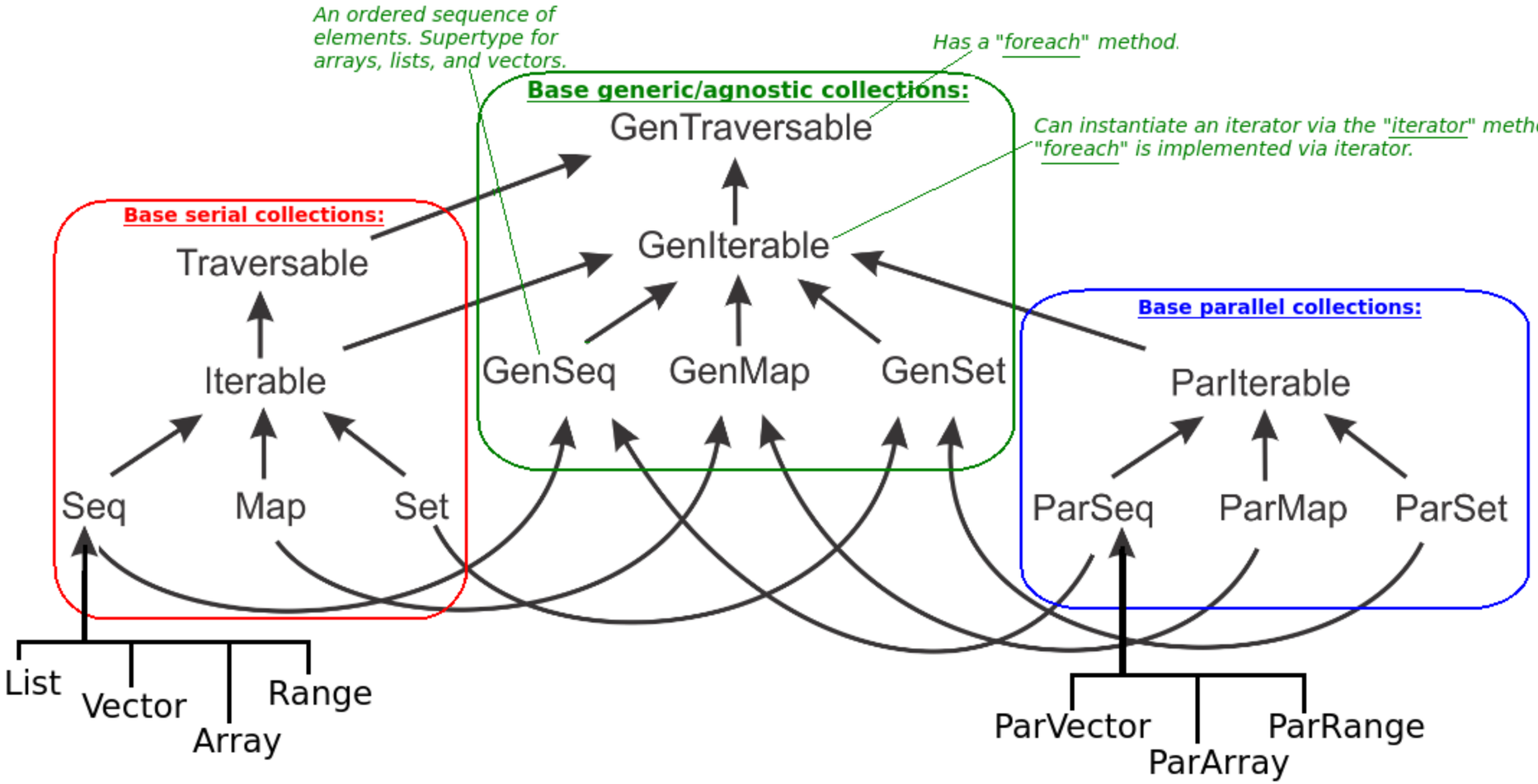
```
Set(1,2,3).par      // Returns a ParSet instance

Array(1,2,3).par    // Returns a ParArray instance

(1 to 5).par        // Returns a ParRange instance

List(1,2,3).par     // Returns a ParVector instance
```

| Data Type | ใช้สำหรับ | จุดเด่น | ข้อเสีย |
|---|---|---|---|
| Future | Asynchronous & Parallel | ใช้งานง่าย, อยู่ใน Standard Library | ควบคุมเธรดยาก, ไม่มี Error Handling ที่ดี |
| IO (Cats Effect) | Functional Parallelism | ปลอดภัย, ไม่มี Side Effect, ใช้ parTupled | ต้องเรียนรู้ Cats Effect |
| ZIO | Functional Parallelism & Concurrency | จัดการ Error & Resource ดี, รองรับ Concurrency | ต้องเรียนรู้ ZIO |
| Parallel Collection | Parallel Processing บน Collection | ใช้งานง่าย | ควบคุม Concurrency ไม่ดี, ไม่แนะนำสำหรับงานใหญ่ |




https://docs.scala-lang.org/overviews/parallel-collections/concrete-parallel-collections.html