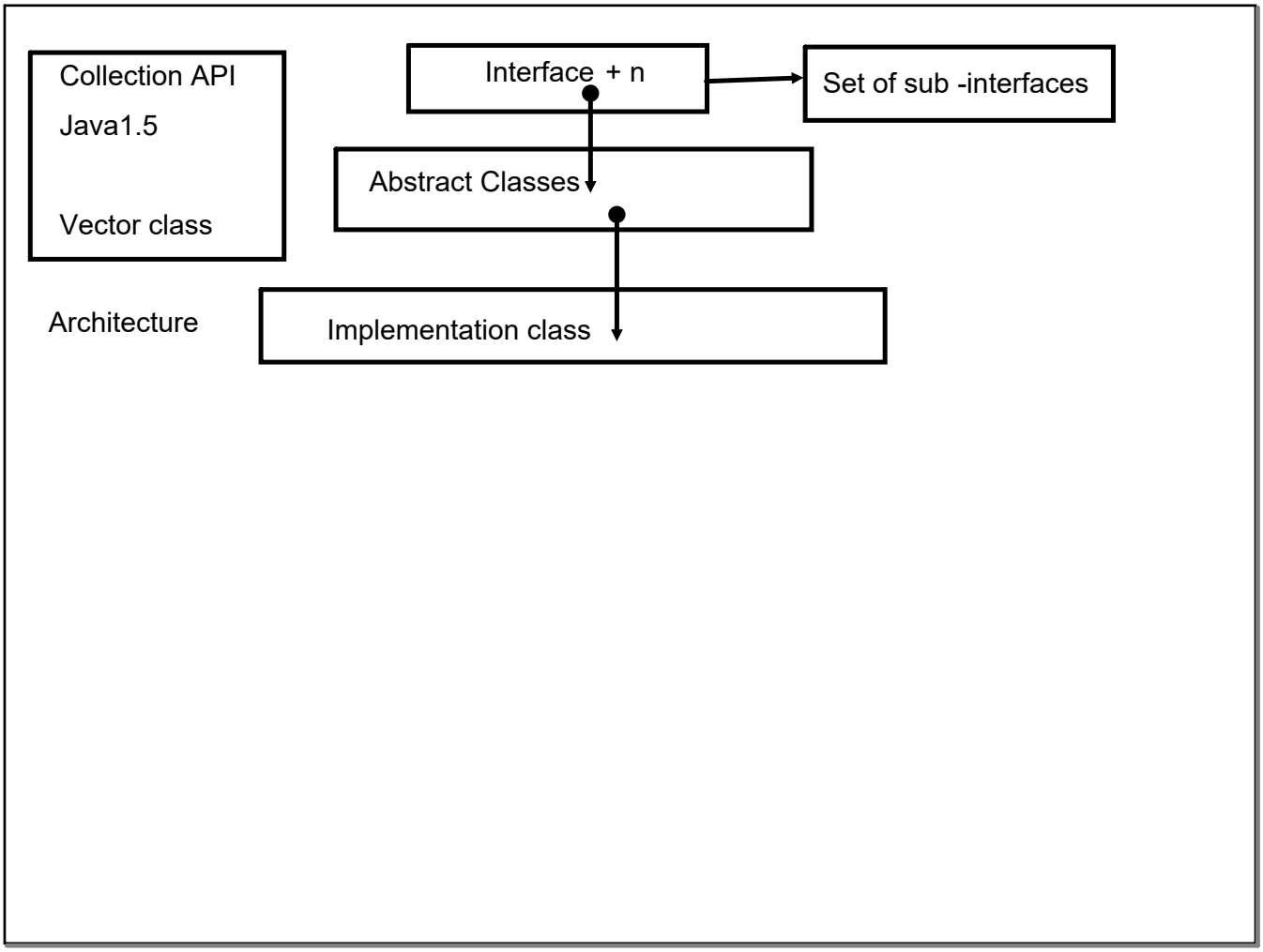
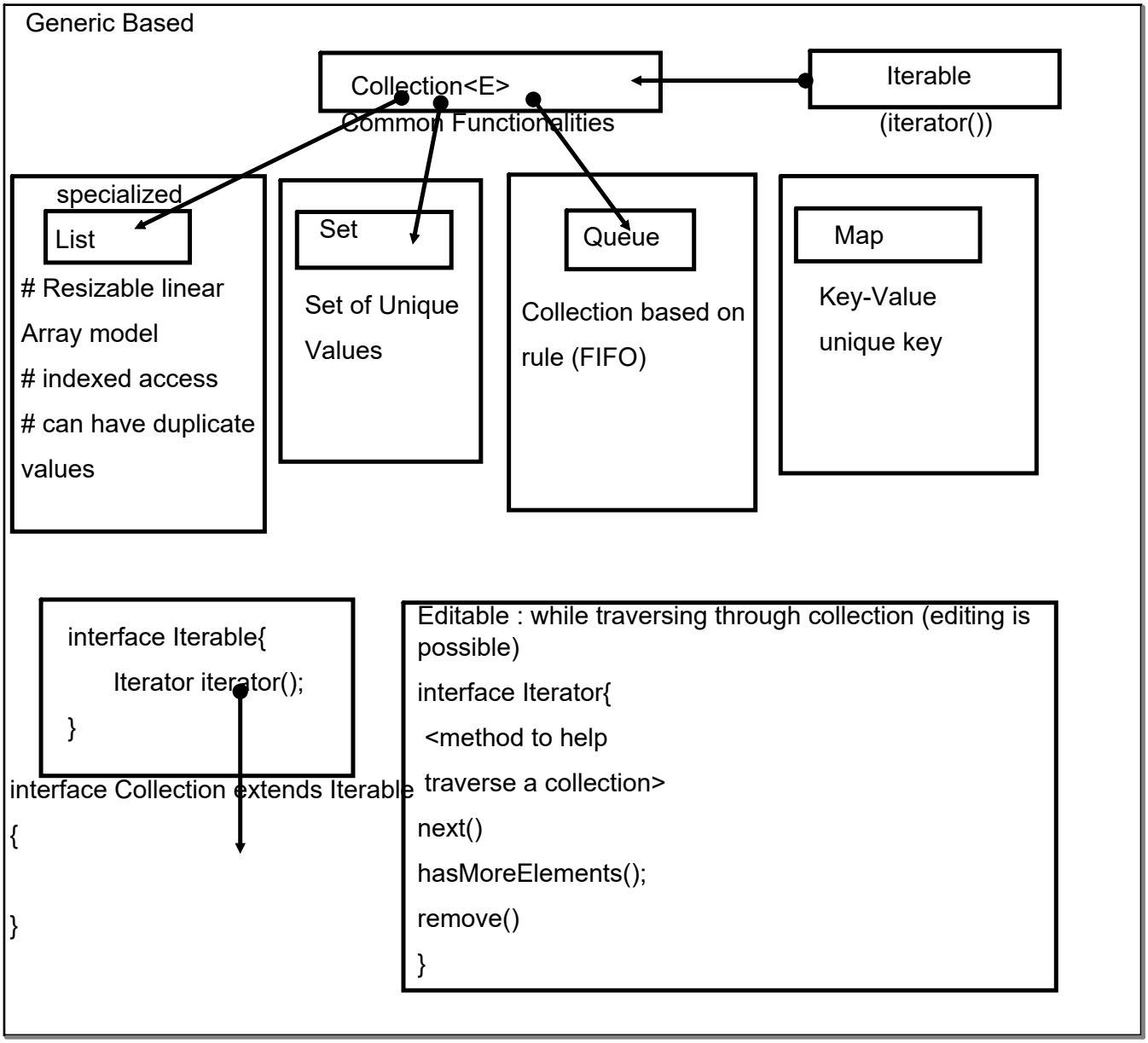
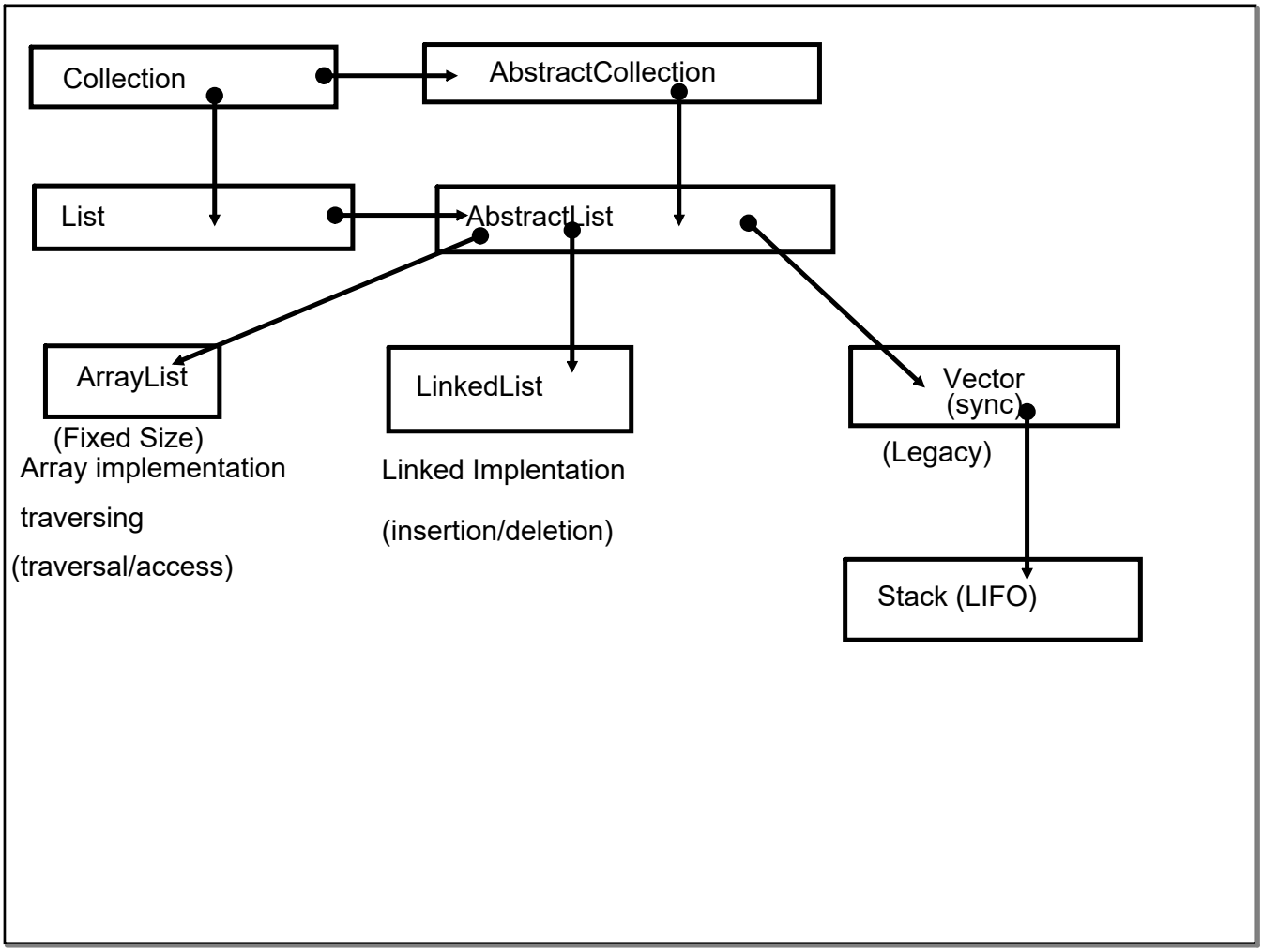
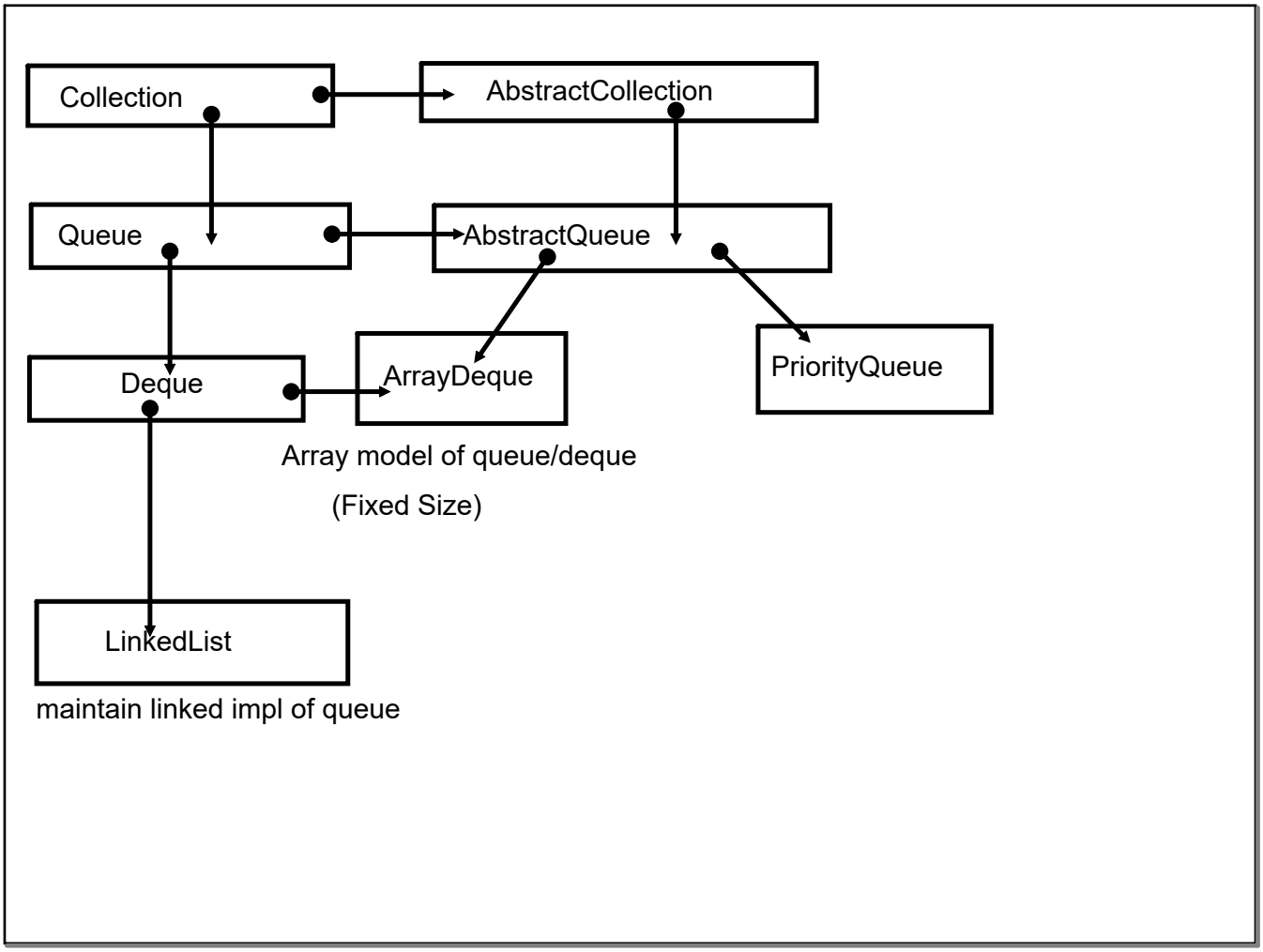
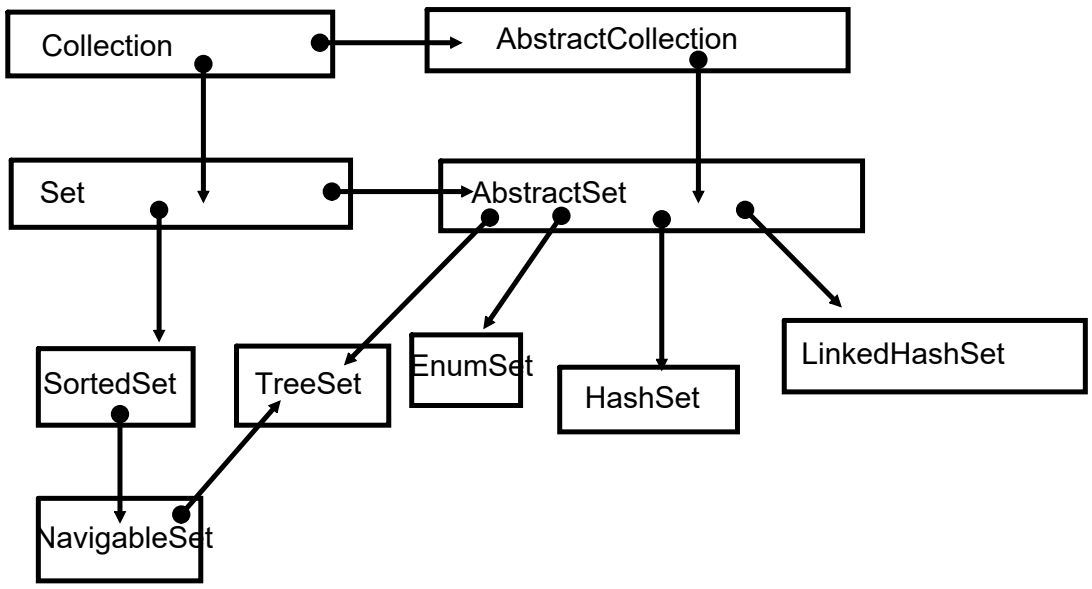

Collection API
Java1.5
Vector class
Interface + n
Set of sub -interfaces
Abstract Classes
Implementation class
Architecture


Generic Based
Collection<E>
Common Functionalities
Iterable
(iterator())
specialized
List
Resizable linear
Array model
indexed access
can have duplicate
values
Set
Set of Unique
Values
Queue
Collection based on
rule (FIFO)
Map
Key-Value
unique key
interface Iterable{
Iterator iterator();
}
interface Collection extends Iterable
{
}
Editable : while traversing through collection (editing is possible)
interface Iterator{
<method to help
traverse a collection>
next()
hasMoreElements();
remove()
}


Collection
AbstractCollection
List
AbstractList
ArrayList
(Fixed Size)
Array implementation
traversing
(traversal/access)
LinkedList
Linked Implentation
(insertion/deletion)
Vector
(sync)
(Legacy)
Stack (LIFO)


Collection
AbstractCollection
Queue
AbstractQueue
Deque
ArrayDeque
PriorityQueue
Array model of queue/deque
(Fixed Size)
LinkedList
maintain linked impl of queue



Set : Unique Values

Enums : uniques

Hashing :

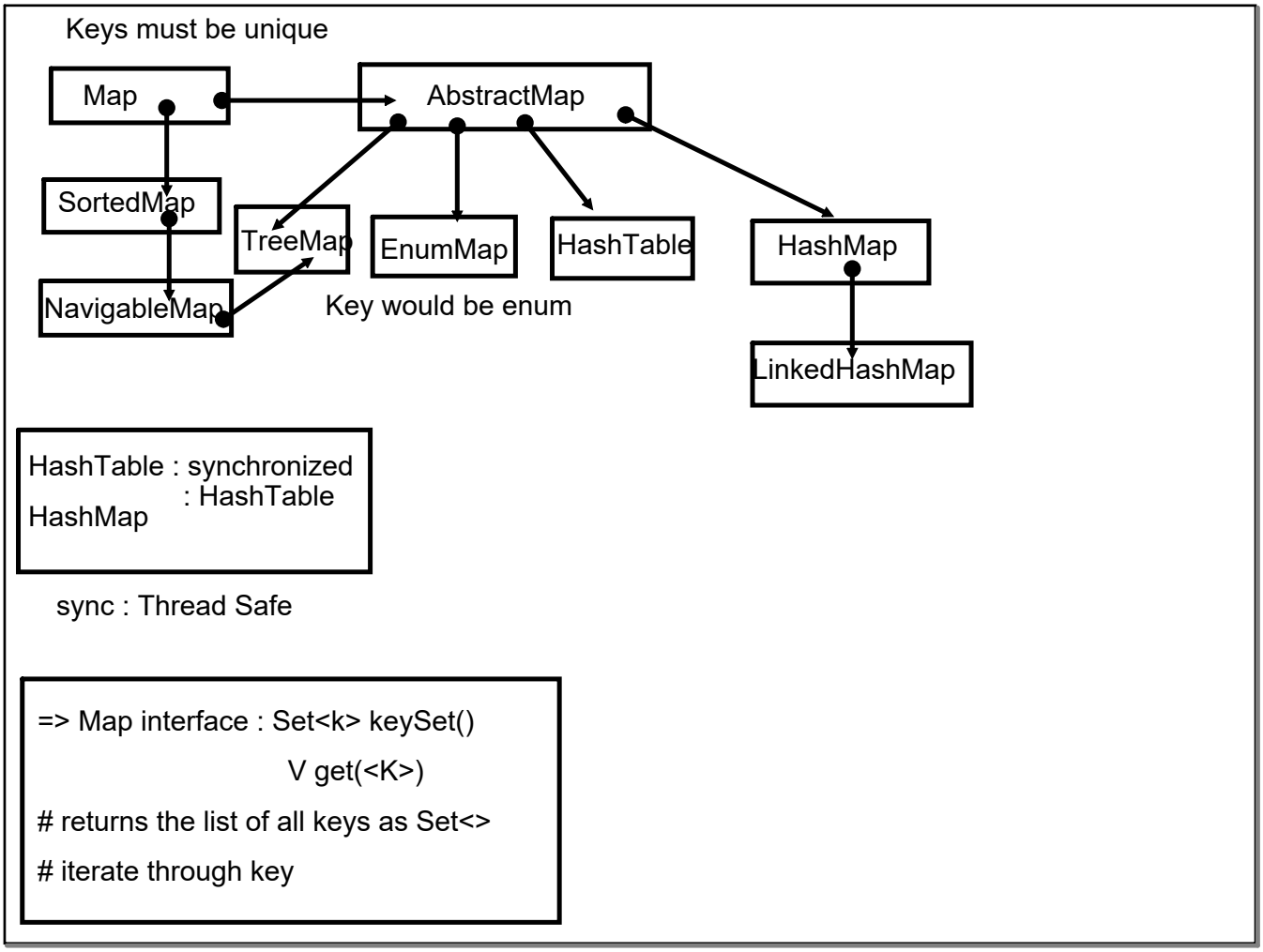
HashSet : Table Hashing

LinkedHashSet : Linked Hashing

SortedSet : maintain data in sorted

NavigableSet : Quick Navigation functionalities

Red-Black Tree


Keys must be unique
Map
AbstractMap
SortedMap
TreeMap
EnumMap
HashTable
HashMap
NavigableMap
Key would be enum
LinkedHashMap
HashTable : synchronized
: HashTable
HashMap
sync : Thread Safe
=> Map interface : Set<k> keySet()
V get(<K>)
returns the list of all keys as Set<>
iterate through key

Algorithm Classes

1. Arrays.
2. Collections.

@ Provide common functionalities can be applied on Array int [] and Collection

Collections.concurrentList(); // Thread safe equivalent

Java - 8

Functional-Programming
functional interface
default methods
static methods
lambdas
streams
methods references

DateTime API
Optional
Nashorn engine (javascript engine)
Extension in collection API

Traditional :(Pure OOPs) : Imperative

Functional : Declarative

Imperative :

How : focus

Pure OOPs

Embraces data mutability

Declarative

What : focus

Functional Programming (pure function)

Data immutabilty

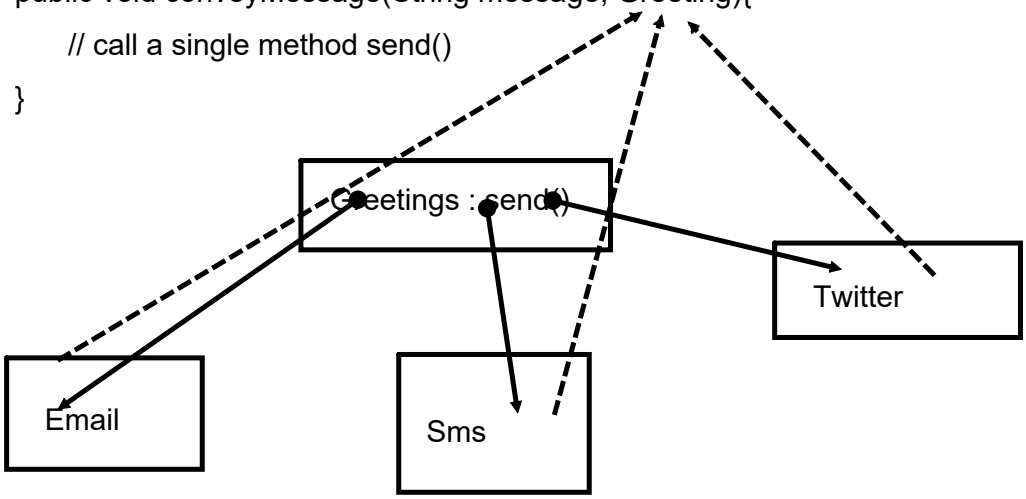
SQL style

Interfaces:

can have functions with definitions (default methods)

static methods with definitions

```
public void conveyMessage(String message, Greeting){
    // call a single method send()
}
```



LAmbda Syntax for pure function

1. not have any accessiblity modifier (nothing related to class)
2. Do not any name (anonymous function)
3. no return type
4. no param type
5. (<param>) -> { <definition>}

```
void fun(String str1, int n){
}
(str1,n) -> {
}

void fun(String str1){
}
str1 -> {
}

void fun(){
}
() -> {
}

void fun(String str1){
      // only single instruction
}
str1 -> single instruction

void fun(String str1){
      // instruction
      return a;
}

str1 -> {
      // instruction
      return a;
}
int add(int a, int b){
 return a+b;
}
// for single inst not bounded in braces return is default associated
(a, b) ->  a+b;
```

interface <reference> = <lambda expression>

only when interface is functional interface :

contains only one abstract methods

can have any number of default/static

refrence of an interface can refer to only those lambda expression whose signature matches with the only abstract method inside the interface

Runnable :

Comparator

Comparable

few protoypes have been identified which are common in use

api : functional interface

Consumer : void accept(<>), BiConsumer [two param], IntegerConsumer()

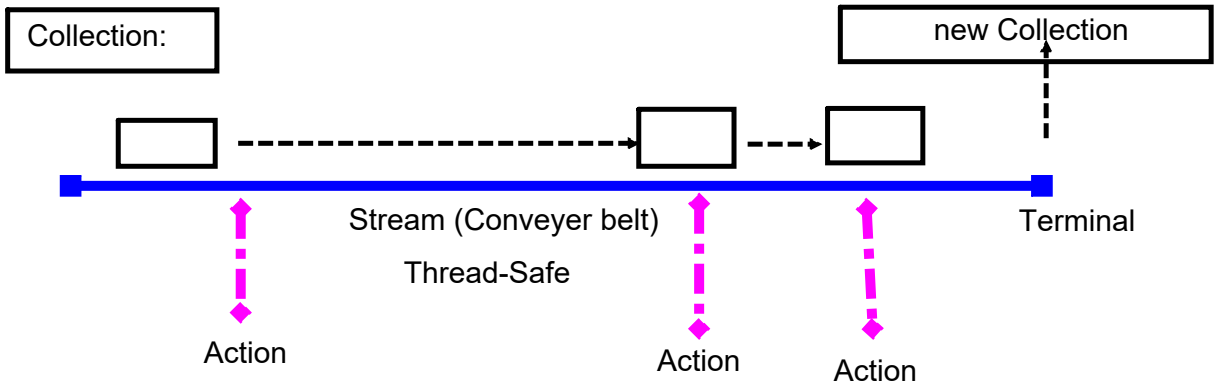
Predicate : boolean test(<>)

Function : <> apply(<>)

Supplier : <> get()

Streams : Safe/efficient way to process a collection

Not a data-structure/ not going to store any data | Lazy processing model



Stream method

1. action <intermediatory methods>
2. Terminal method (stops the stream) cannot add any new activity after it

Stream are not going to initiate if no terminal activity is there

stream : external mutable resource
inherently complex (JVM)

want to use calculator to perform addition of stream of number

1 2 3 4 5..... result : mutable (not thread safe)

1
3 Data inconsistency

25 : result 5,6,7

result = result + v
result = 25 + 5
result = 25 + 6

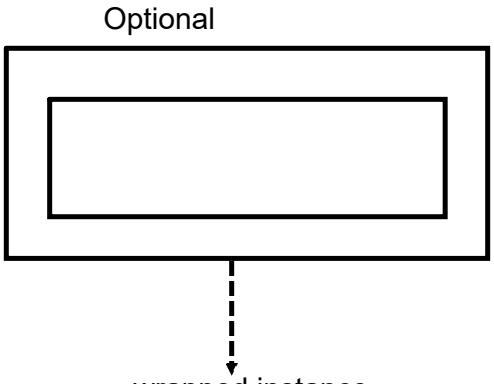
Unboxing : buffer

finalize() // gcm

Optional : to avoid null reference exception

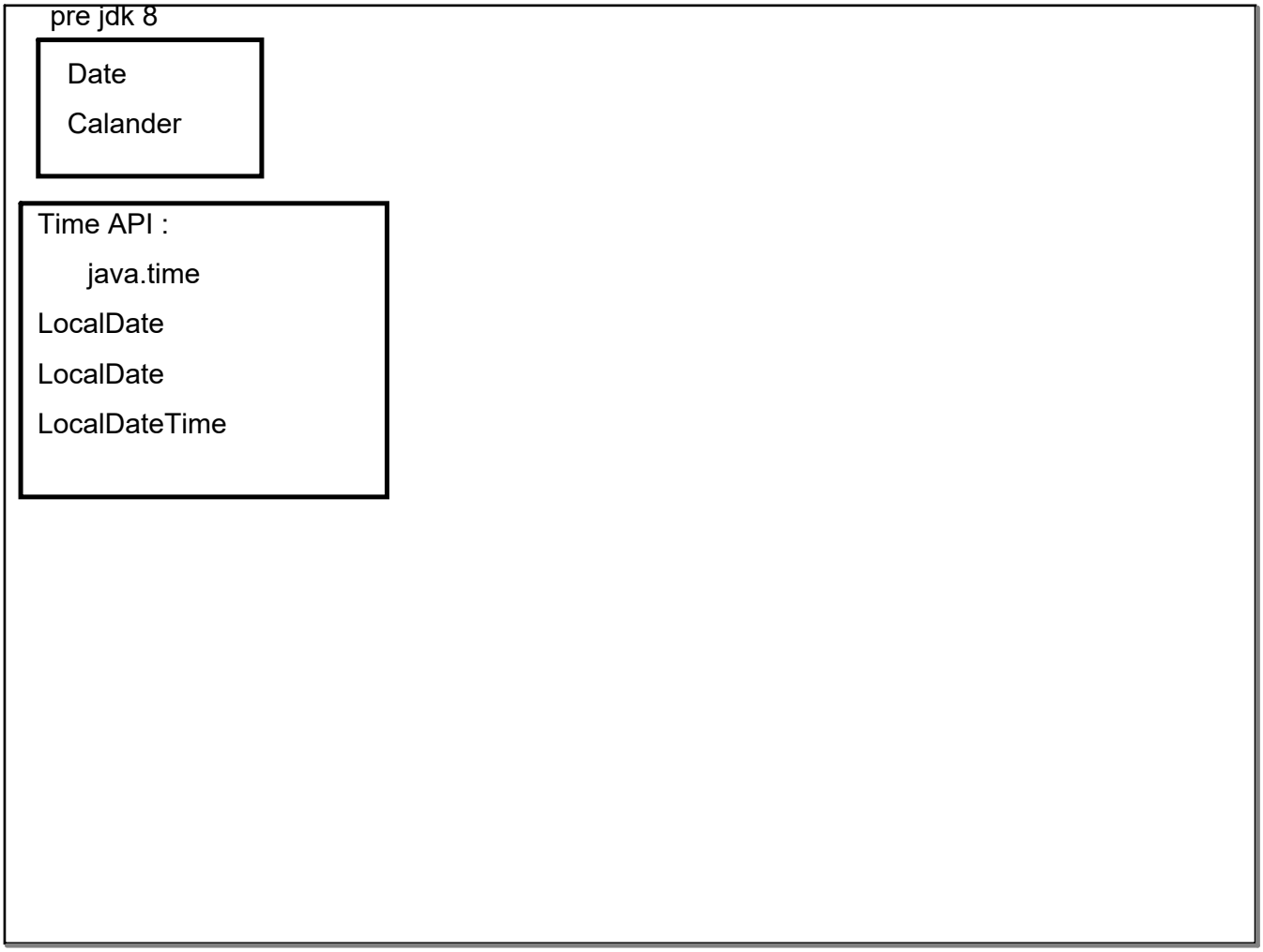
if any object being initialized by some external logic (method)

if no explicit validation code is added to check null



forced to put validation

functionality to handle situation


pre jdk 8
Date
Calander
Time API :
java.time
LocalDate
LocalDate
LocalDateTime

Multi-Threading

Multi-Programming :

O/S to reside more than 1 exe program in memory , not necessarily executing them

Multi-Tasking :

O/S execute more than 1 program simultaneously , interleaved fashion (time sharing)

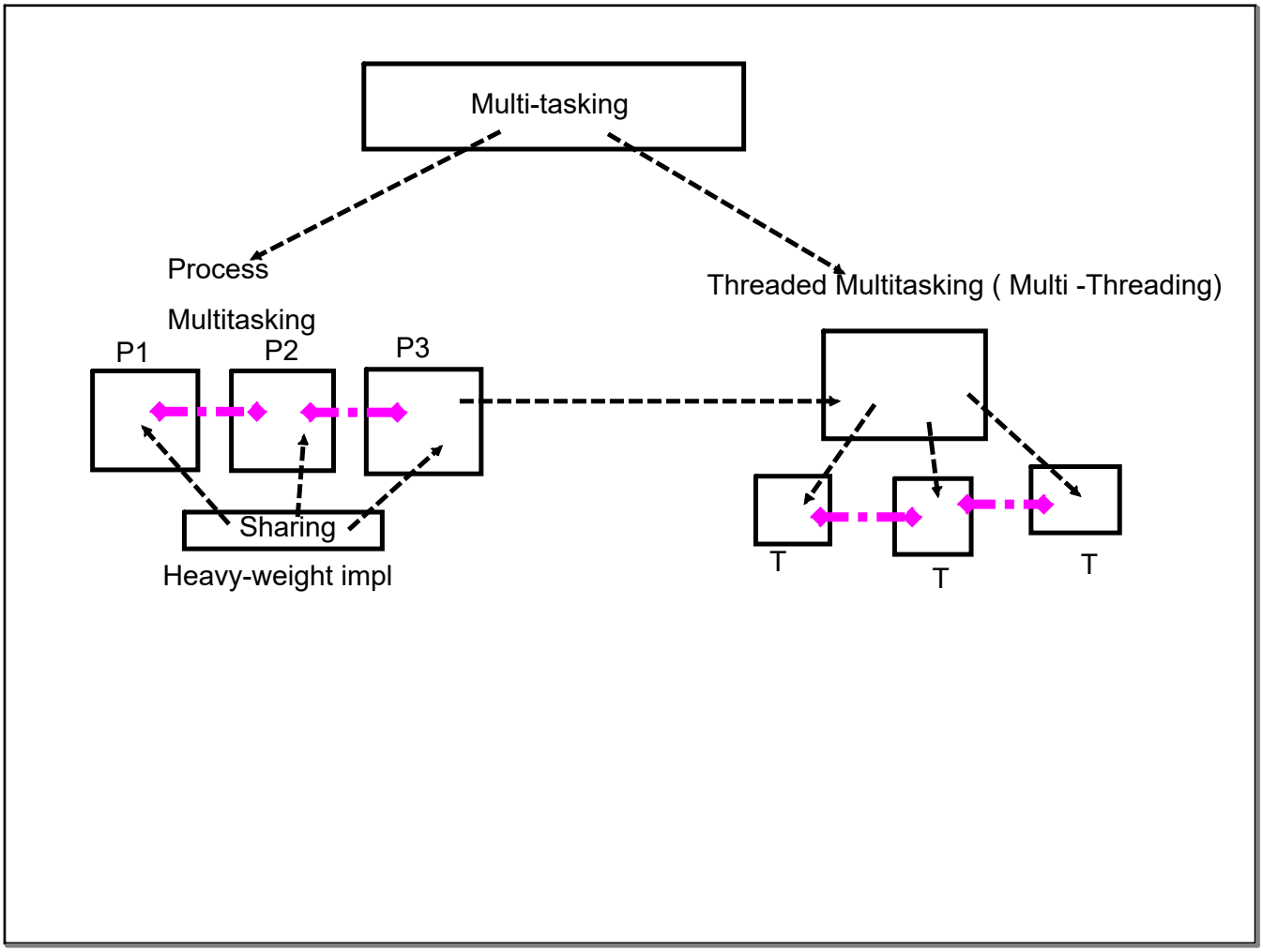
Multi-Processing

O/S to manage more than 1 processing units (parallel / array)

O/S

Multi-tasking :

multi-core processor : multi-processing

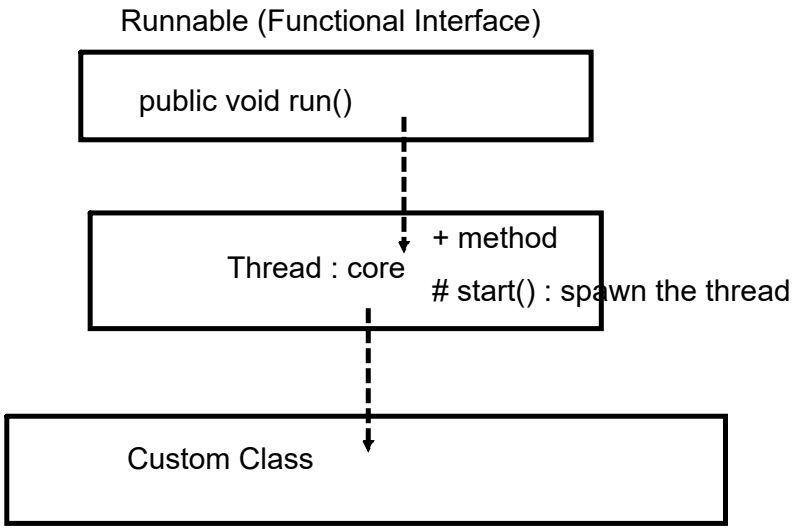

Multi-tasking
Process
Multitasking
Threaded Multitasking (Multi -Threading)
P1
P2
P3
Sharing
Heavy-weight impl
T
T
T

=> Thread class

=> Runnable

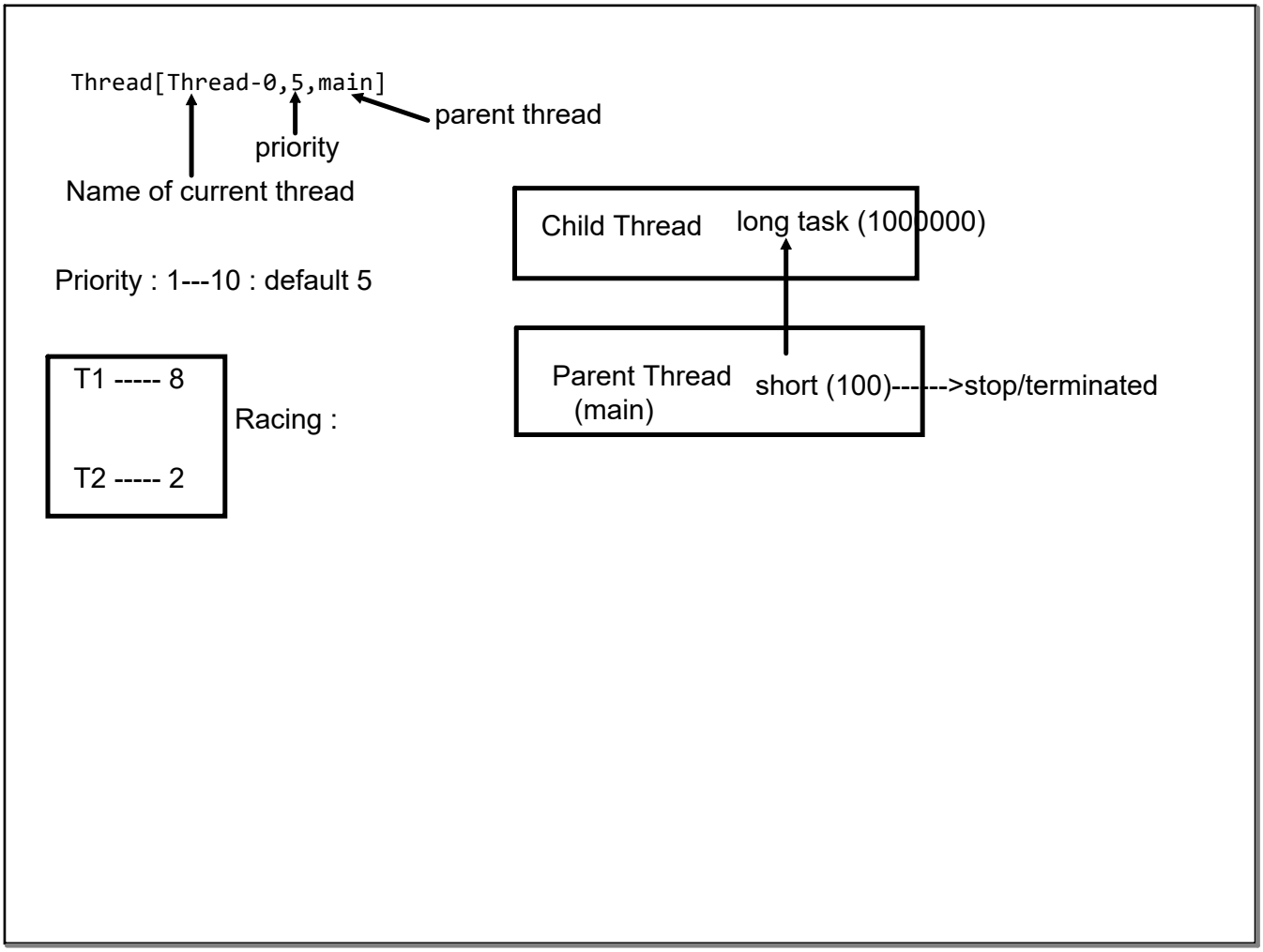
Sharable resources : data consistency

Inter-thread communication

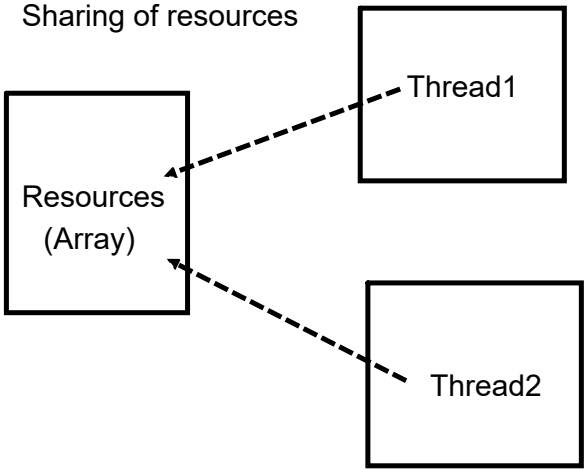


```
class Thread implements Runnable{
    public void run(){}  // empty
    public final void start(){
        // low level functionality to spawn a new thread
        // native coding : C/C++
        // call the run() : assigning thread capacity
    }
}
```

```
class MThread extends Thread {
    public void run(){  // overriding run of thread
        // thread activity
    }
}
```

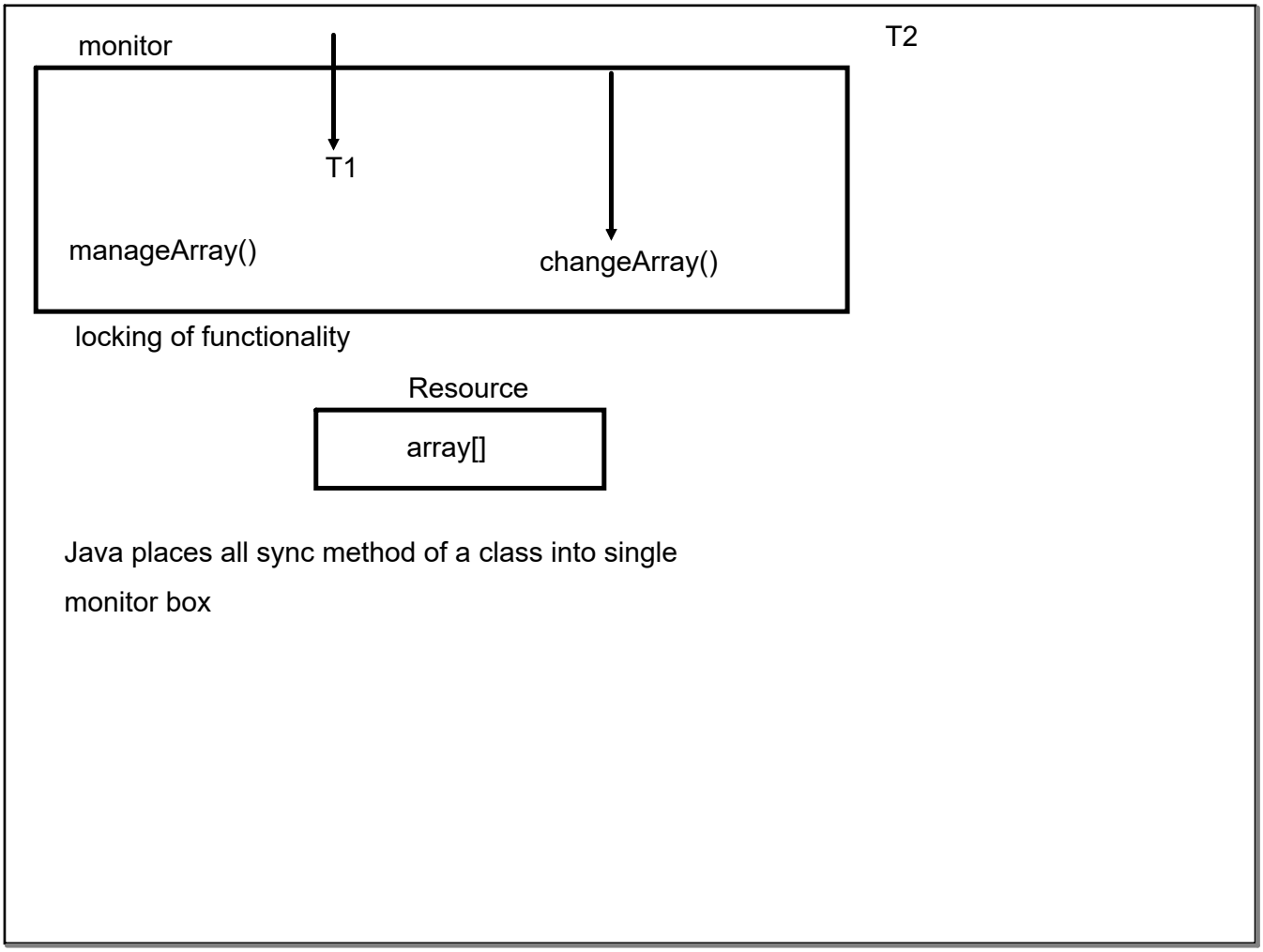

Thread[Thread-0,5,main]
parent thread
priority
Name of current thread
Priority : 1---10 : default 5
T1 ----- 8
T2 ----- 2
Racing :
Child Thread
long task (1000000)
Parent Thread
(main)
short (100)------->stop/terminated

Sharing of resources



Any method blocking the thread execution will throw checked exception

Interrupted Exception



Java places all sync method of a class into single monitor box

locking of functionality : sync method [permanent] : Resource owner

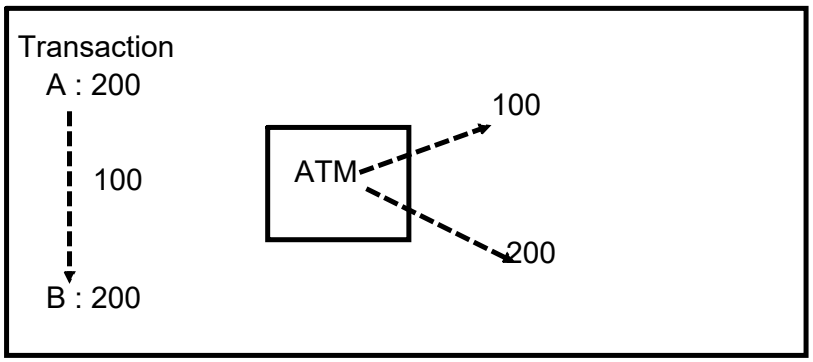
locking of objects : sync block [temp] : user

Concurrent-Collection API

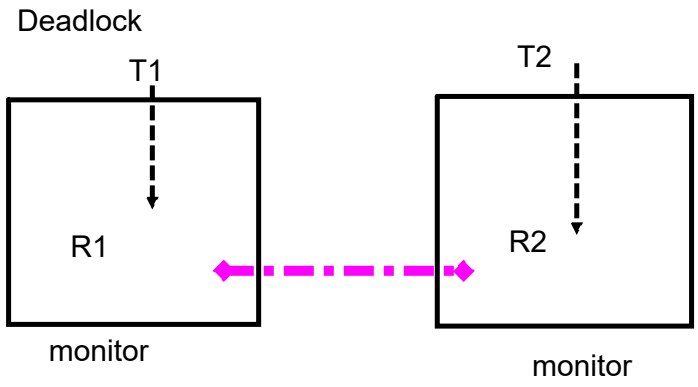
High level of concurrency with Thread Safety

synchronized : method/block : Wide Spectrum of locking

granular locking : ReentrantLock();

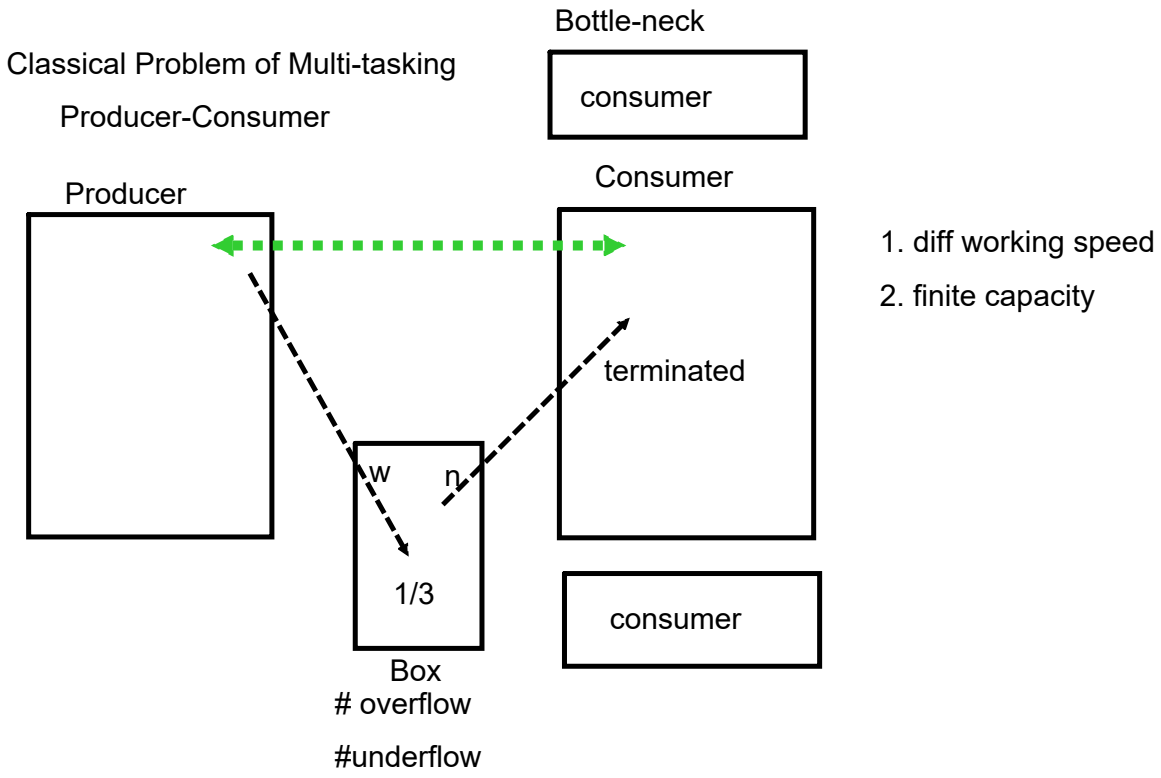

Transaction
A : 200
100
B : 200
ATM
100
200

Solution : Inter-Thread Communication



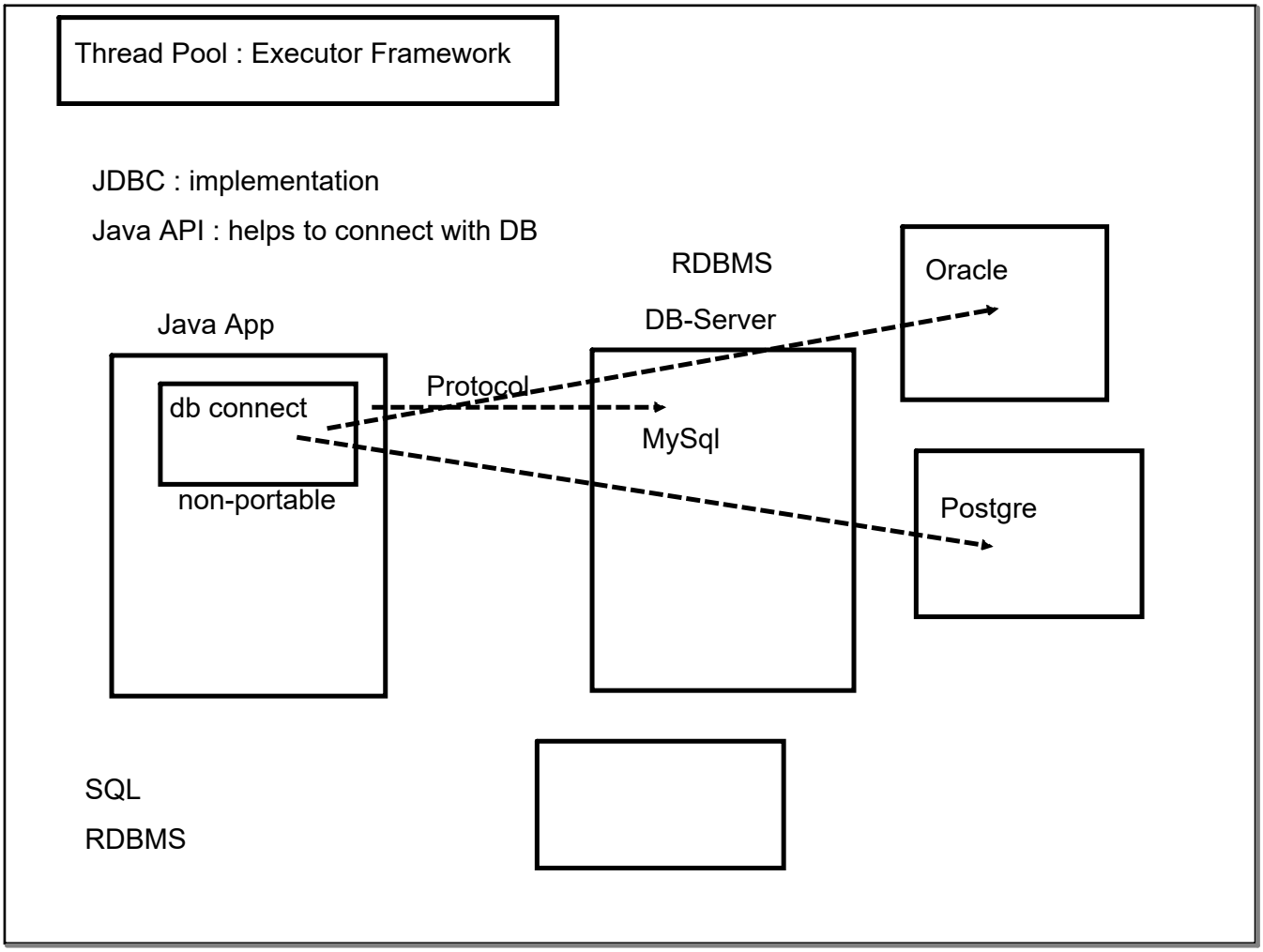
Classical Problem of Multi-tasking

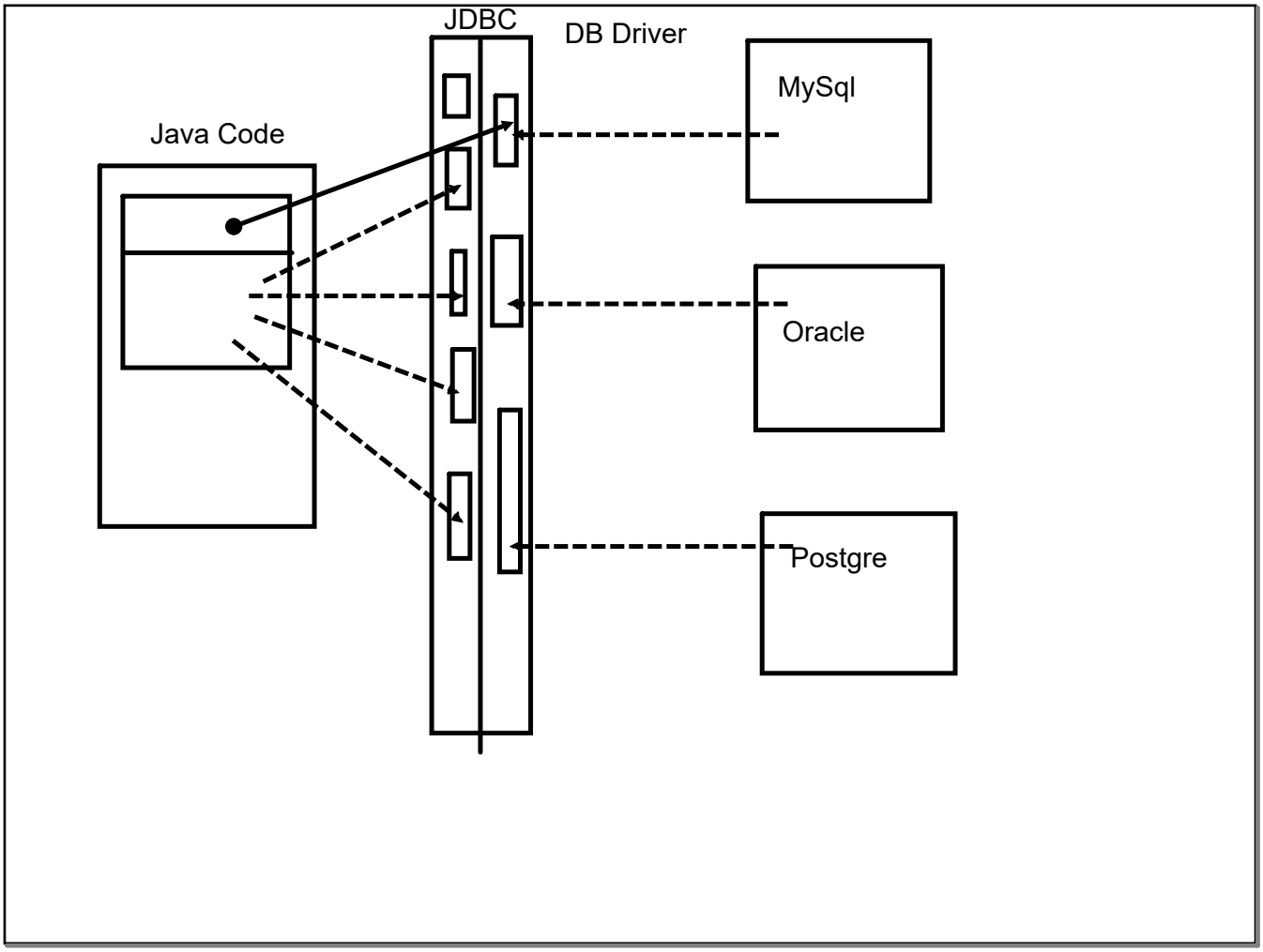
Producer-Consumer

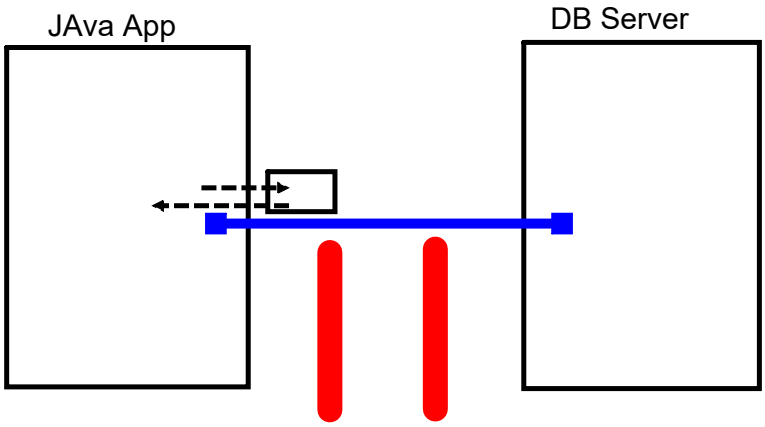


1. diff working speed
2. finite capacity

All Java object provides platform inter-thread comm


Thread Pool : Executor Framework
JDBC : implementation
Java API : helps to connect with DB
RDBMS
DB-Server
Oracle
Java App
Protocol
db connect
MySql
non-portable
Postgre
SQL
RDBMS


JDBC
DB Driver
MySql
Java Code
Oracle
Postgre



Steps:

1. Load the DB Driver
2. Setup a connection
3. Create a Statement