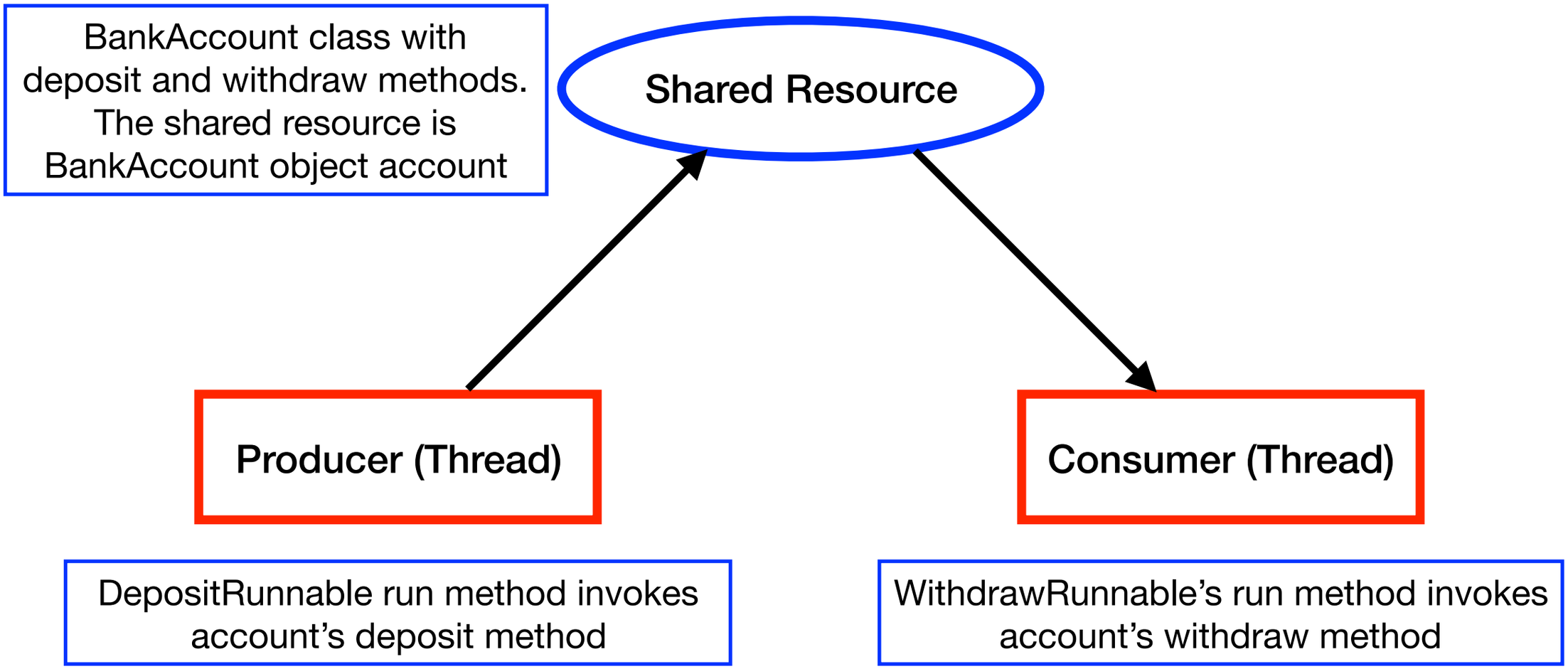
Diagram for Thread Interaction (BankAccount)

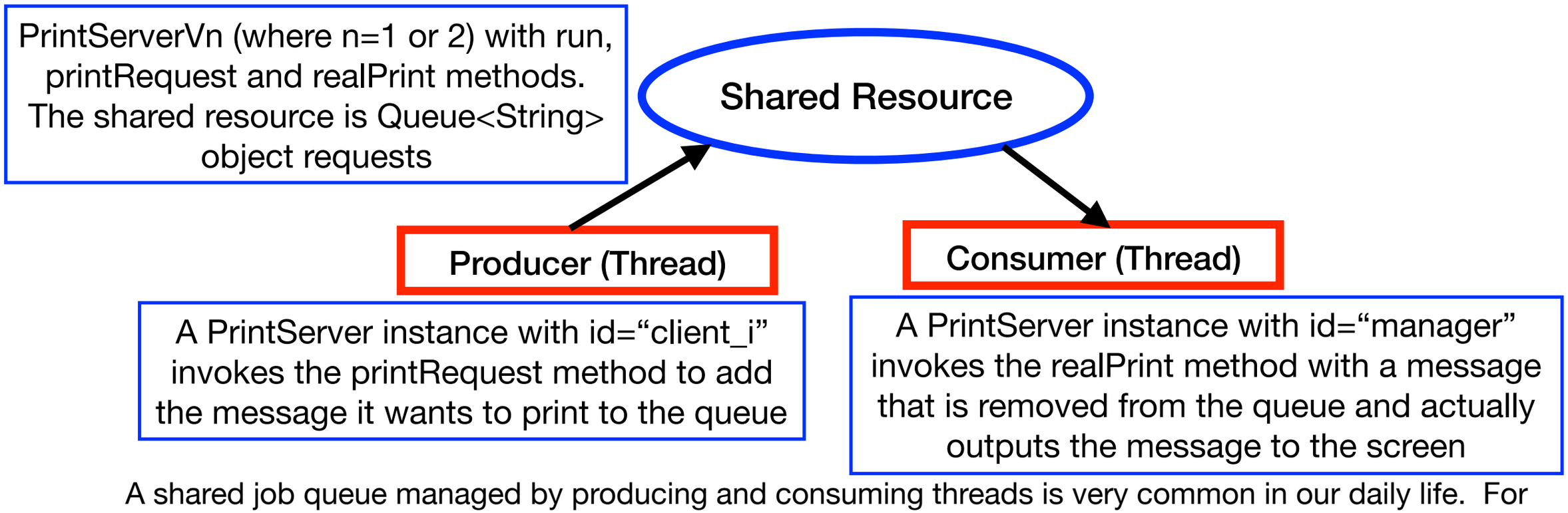
CSCE 314 Hyunyoung Lee



The producer-consumer model depicts nicely the interaction between multiple threads that concurrently share a resource to perform distributed tasks. Also, the producer-consumer model is consistent with the causal ordering of the events occurred at the producing and consuming threads.

Diagram for Thread Interaction (HW7 P1 PrintServer)

CSCE 314 Hyunyoung Lee



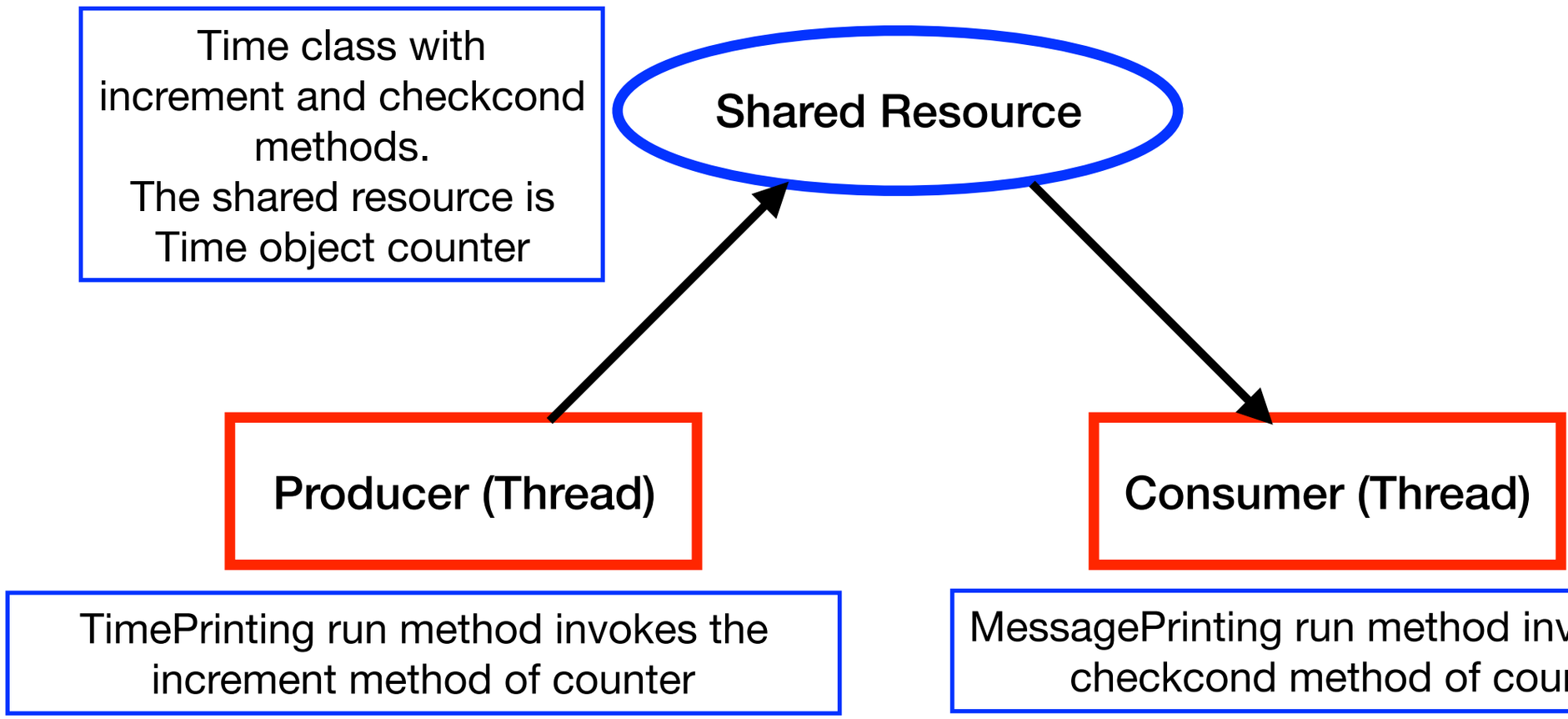
A shared job queue managed by producing and consuming threads is very common in our daily life. For example, a shared printer in the lab has a print job queue, to which print jobs are added (enqueued) by the students, and they are output (dequeued and printed) in a FIFO manner. Problem 1 of Homework 7 simulates this situation.

The main difference between this problem and BankAccount (or P2 Time) is that for this problem there are no separate classes for producer and consumer, whereas in BankAccount, the producer is DepositRunnable and the consumer is WithdrawRunnable (and in P2 Time, they are TimePrinting and MessagePrinting, respectively). For this problem, you will distinguish the producer and the consumer by keeping an id field in the PrintServer class, and when you construct a PrintServer object, you will pass its id as "client" or "manager". There must be only one manager, but there may be more than one clients, thus, give the client's id indexed such as "client_1", "client_2", and so on.

The reasoning behind this design is that in the view of clients, printRequest is requesting a task to the print server even though actual printing is done by the manager. If multiple client threads get scheduled to run before the manager thread gets scheduled, several messages may be added to the queue before manager gets scheduled and prints them.

Diagram for Thread Interaction (HW7 P2 TimeCount)

CSCE 314 Hyunyoung Lee



The increment method first acquires the lock, increments the counter value by one and outputs the value, invokes signalAll() on the condition object, and finally releases the lock (unlock()). The checkcond method acquires the lock, checks the condition (whether x many seconds have elapsed) on the counter by checking the counter value modulo x == 0; if so, outputs the message of the MessagePrinting thread which invoked the checkcond method; otherwise, invoke await() method on the condition object. Then, finally releases the lock.