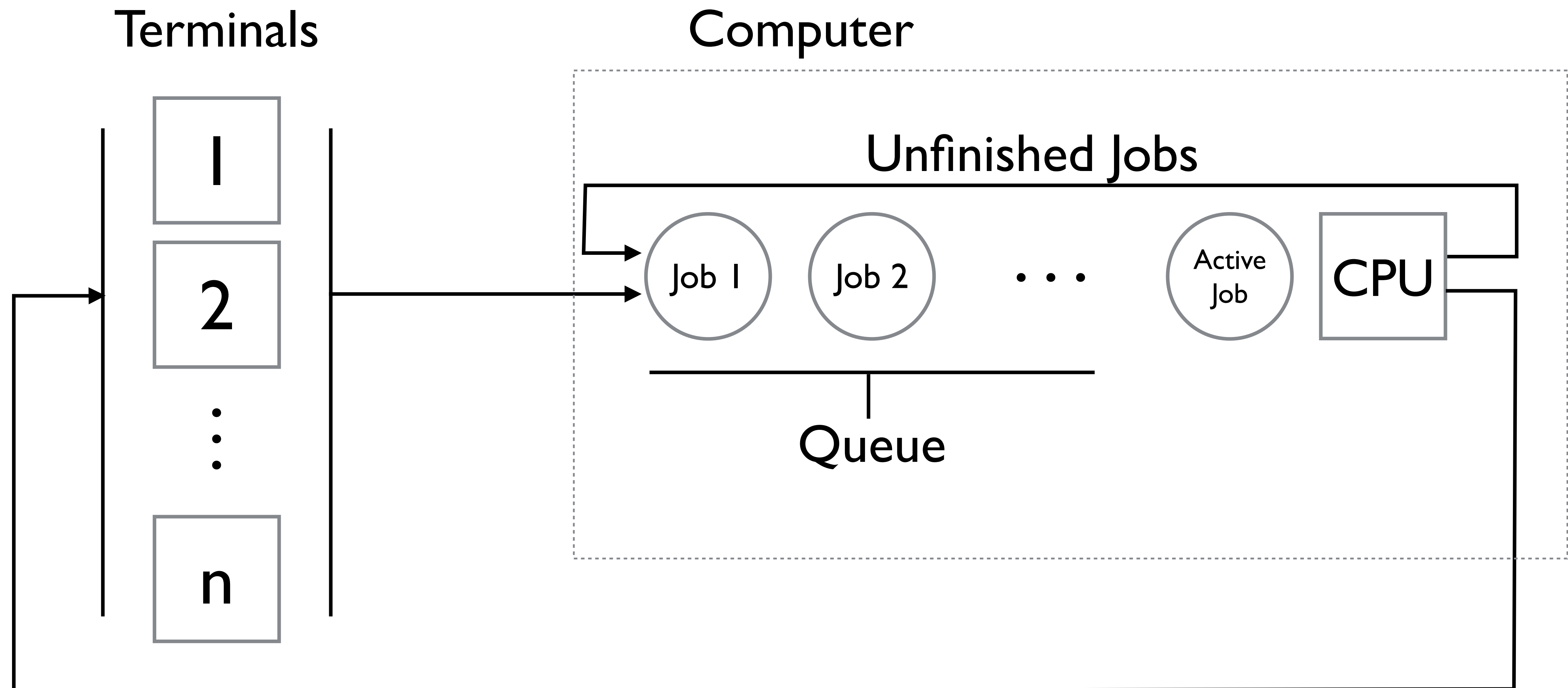
CS445: Modeling Complex Systems

Travis Desell

Averill M. Law, Simulation Modeling & Analysis, Chapter 2

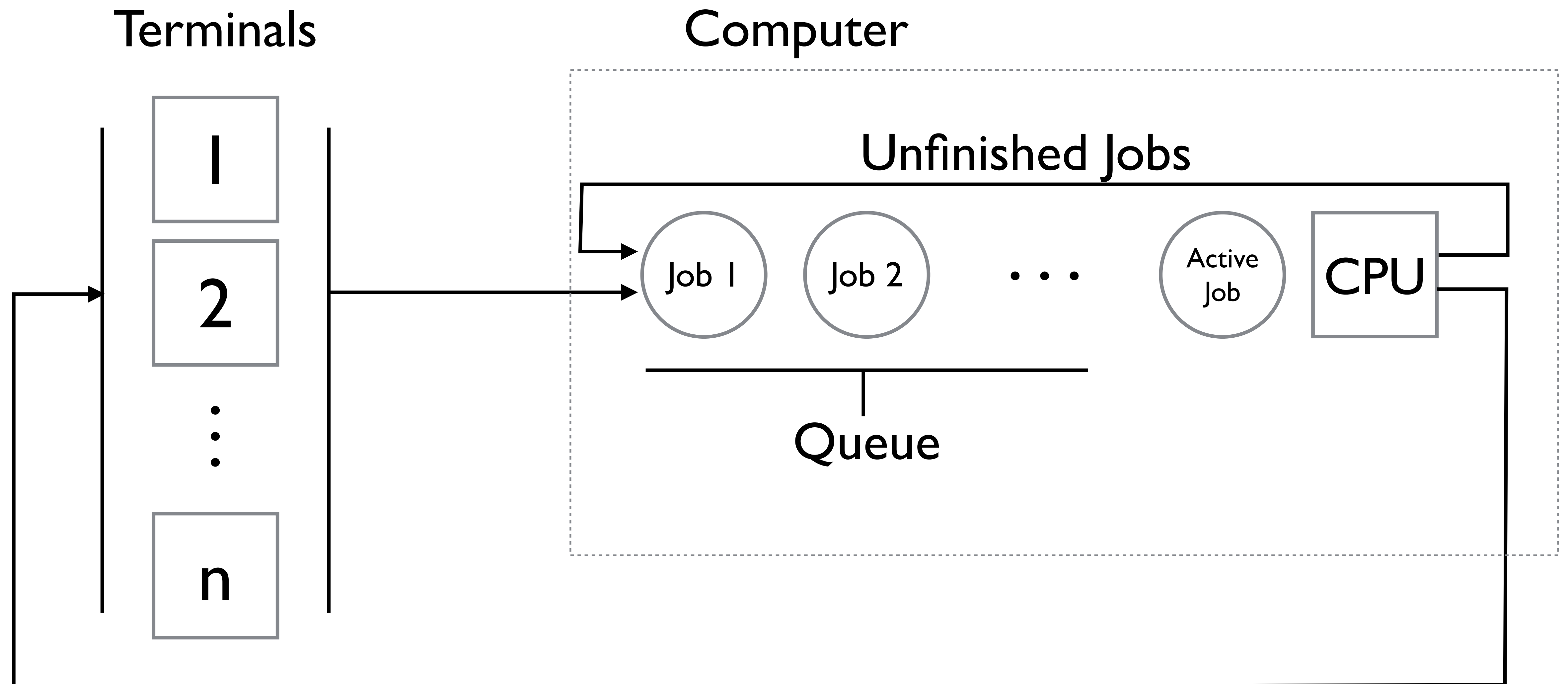
Time-Shared Computer Model

Time Shared Computer Model



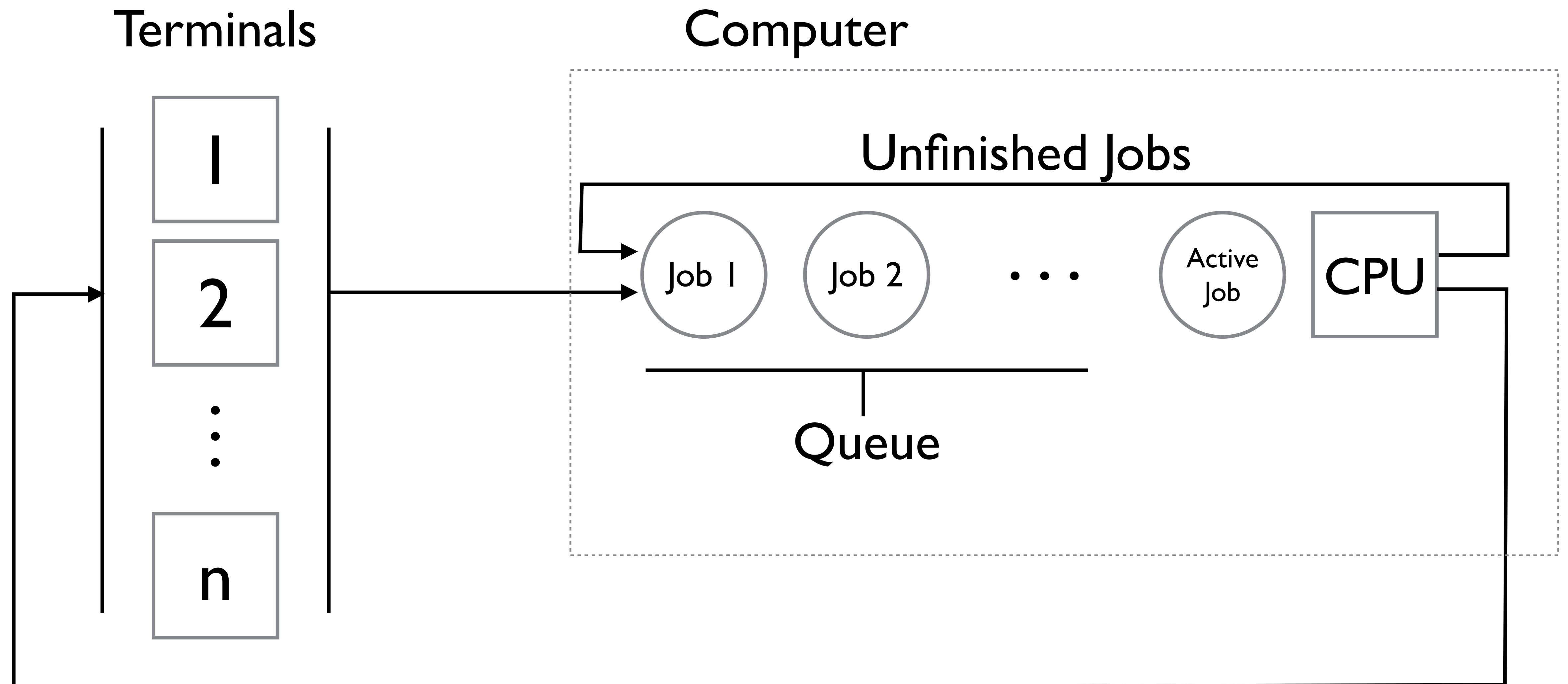
A company (or university) has a computer system consisting of a single central processing unit (CPU) and n terminals. This was a standard model for old mainframes, but is still applicable currently, as it is very similar as to how multiple processes are executed on a CPU by an operating system, or how multiple users utilize a supercomputer or high performance computing cluster.

Time Shared Computer Model



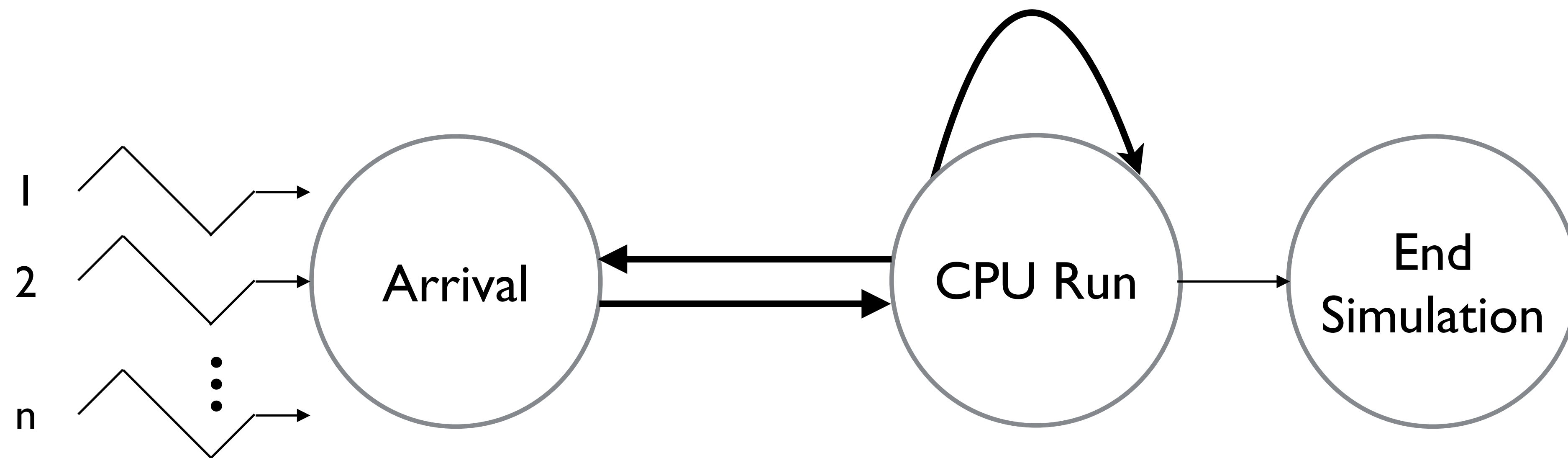
The operator of each terminal will "think" for awhile (the time is an exponential random variable with mean 25 seconds) and then sends to the CPU a job having service time exponentially distributed with mean 0.8 seconds.

Time Shared Computer Model



The jobs arriving at the CPU join a single queue but are served in a *round robin* fashion instead of a FIFO manner. The CPU allocates each job a *time slice* or *quantum* of length $q = 0.1$ second. If you have taken Operating Systems this should be familiar. Each job is processed for its quantum, and if it is not finished by the end of the quantum, it is placed back into the queue.

Time Shared Computer Model



The model can be described with three events, given a minimized event-graph.

There are *n* initial arrival events, one per terminal. This will put the event into the queue or start a CPU run if there are no jobs being run.

The CPU run event will process a time slice, which will either start another arrival event after the thinking time if the job completes, or start another CPU run event when the next time slice has ended.

When the specified number of jobs have completed, this will generate an end simulation event.

Time Shared Computer Model

Let R_i be the response time of the *i*th job to finish service. This is defined as the time elapsing between the instant the job leaves its terminal and the instant it is finished being processed by the CPU. We can vary the number of terminals n to evaluate a variety of information about the simulation.

Given a number of job completions (1000):

1. What is the expected average response time for a job? (continuous time statistic)
2. What is the time average number of jobs waiting in the queue? (discrete time statistic)
3. What is the utilization of the CPU? (continuous time statistic)

It is also possible to answer questions such as:

1. Given n users, how many terminals can it have on the system and still provide users with an average response time of no more than 30 seconds?

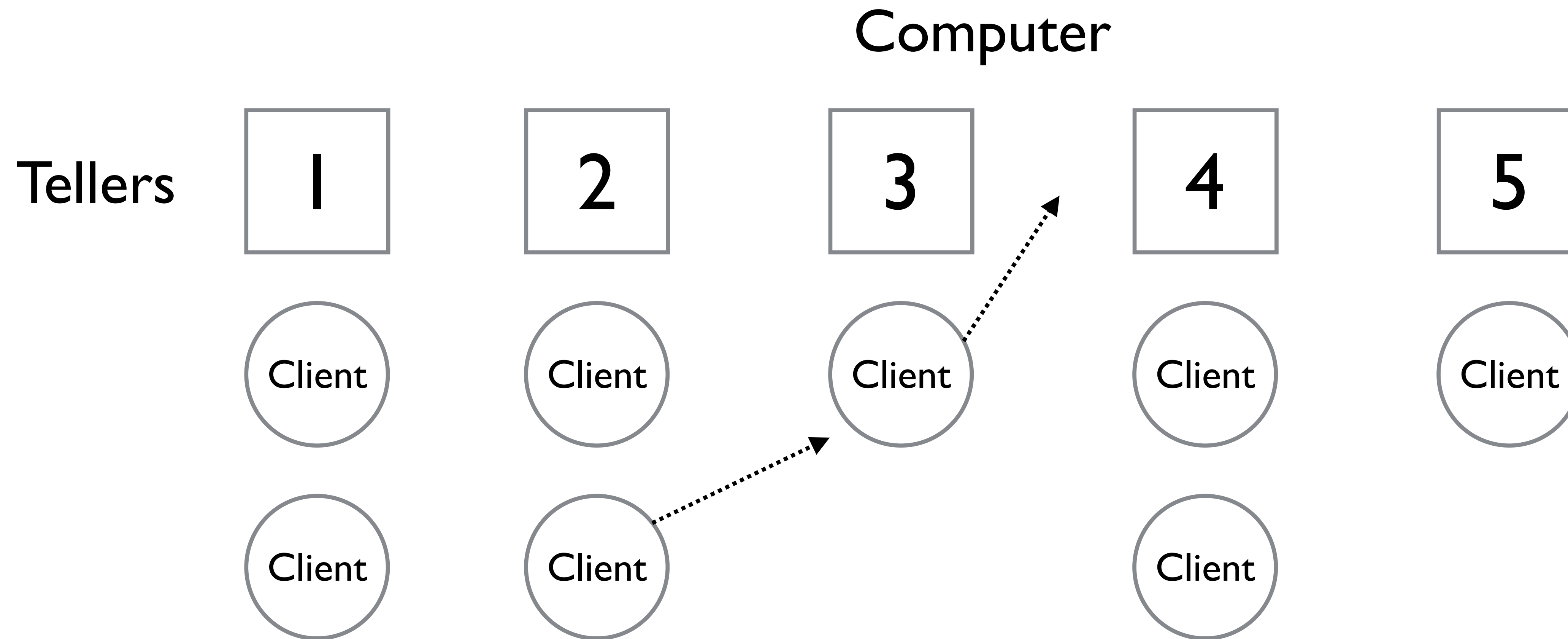
Required Data Structures

This simulation just requires one queue (probably implemented as a linked list), to hold the jobs being processed by the CPU. It will need to have a method to push to the end of the queue, and pop the front of the queue.

c++'s queue has push_back and pop_front: http://www.cplusplus.com/reference/queue/queue/

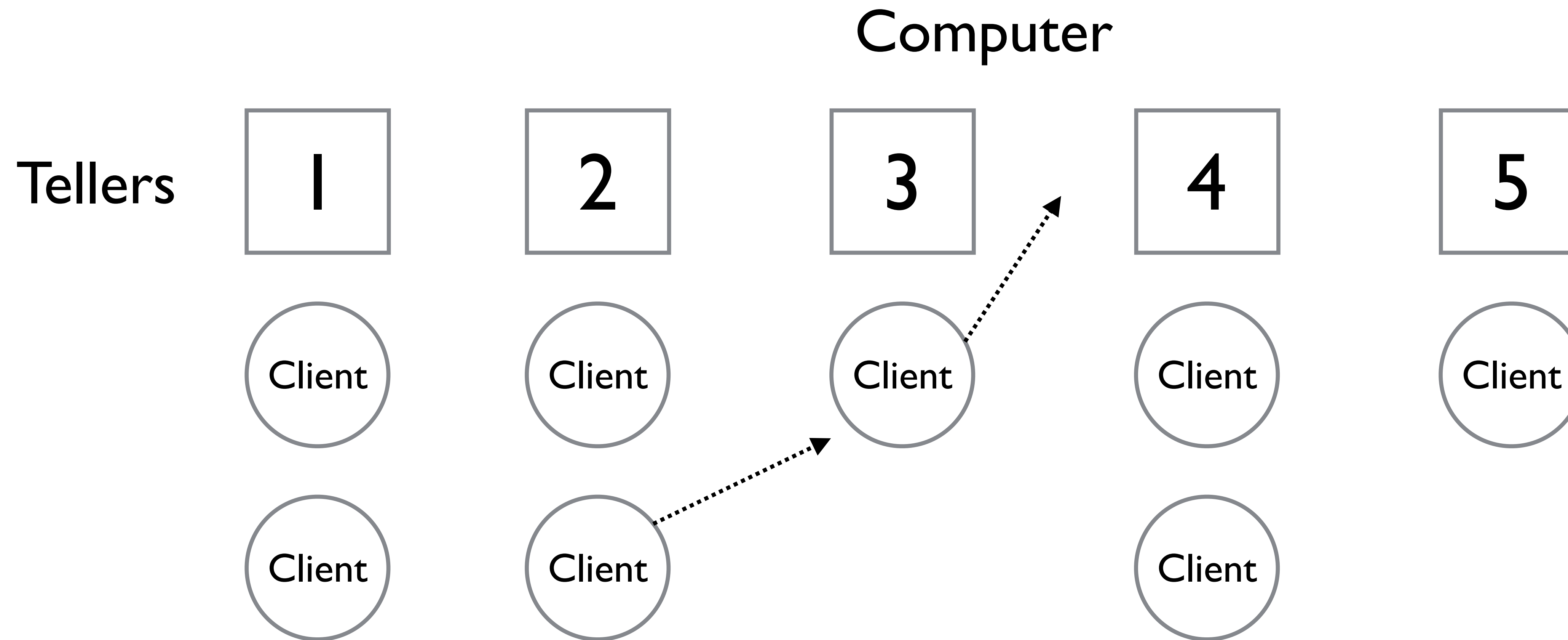
Multi-Teller Bank with Jockeying

Time Shared Computer Model



A company (or university) has a computer system consisting of a single central processing unit (CPU) and n terminals. This was a standard model for old mainframes, but is still applicable currently, as it is very similar as to how multiple processes are executed on a CPU by an operating system, or how multiple users utilize a supercomputer or high performance computing cluster.

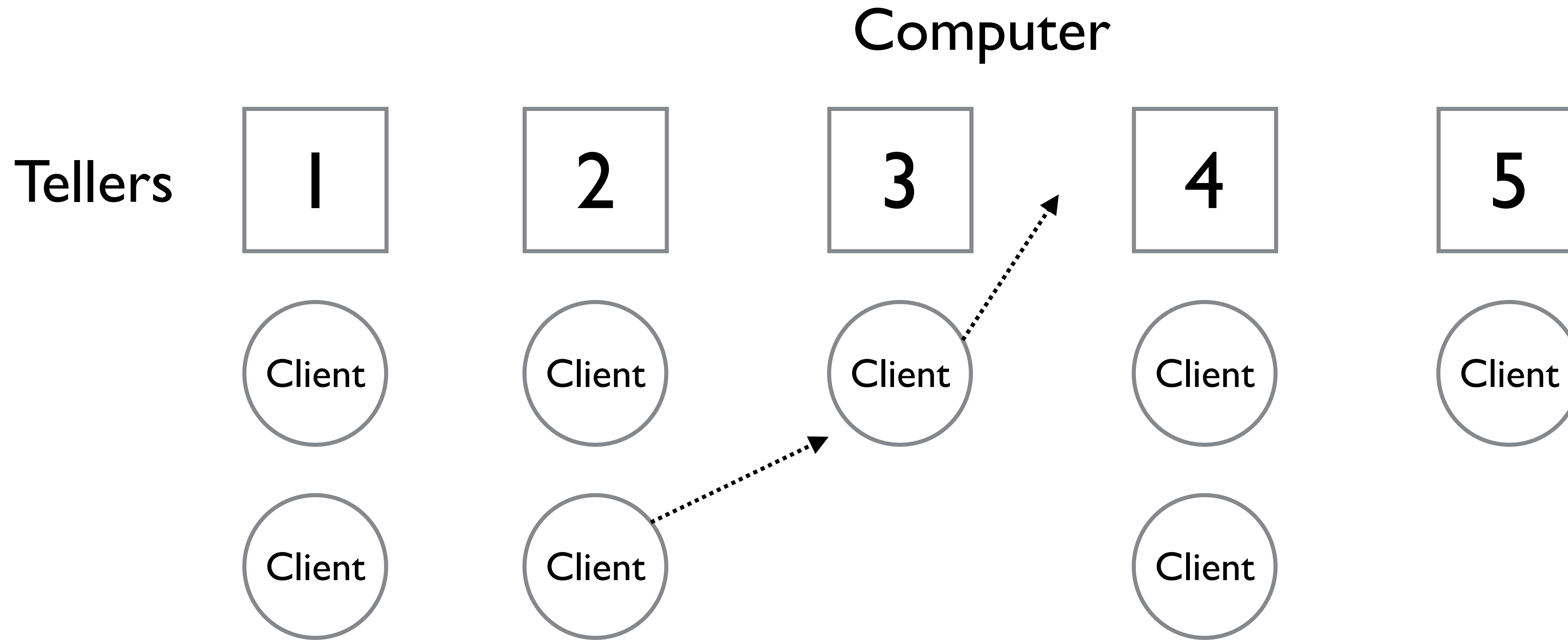
Multiteller Bank with Jockeying



Another common situation worth simulating is a multiteller bank with jockeying.

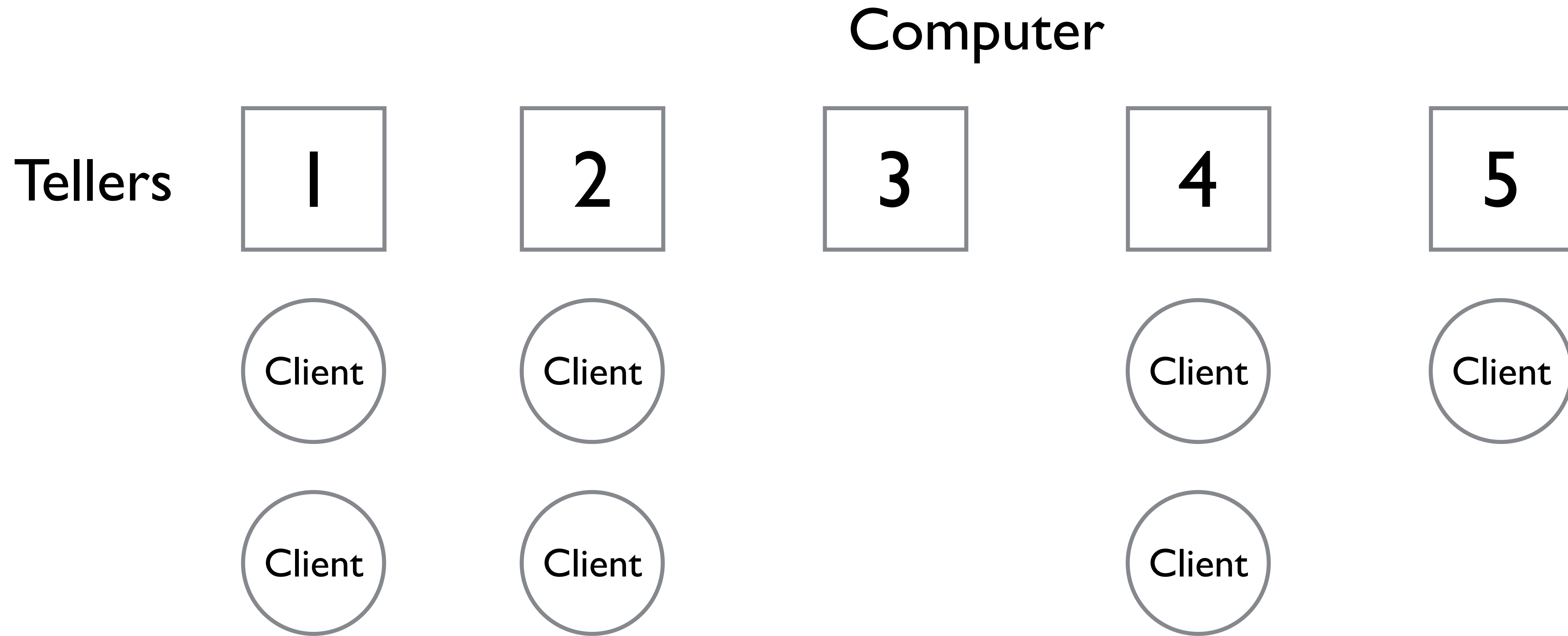
A bank with N tellers (in our case 5) opens at 9am and closes at 5pm but operates until all customers in the bank by 5pm have been served. Customers arrive determined by an IID (independent and identically distributed) exponential random variable with mean 1 minute. Customers are serviced with time IID exponential random with mean 4.5 minutes.

Multiteller Bank with Jockeying



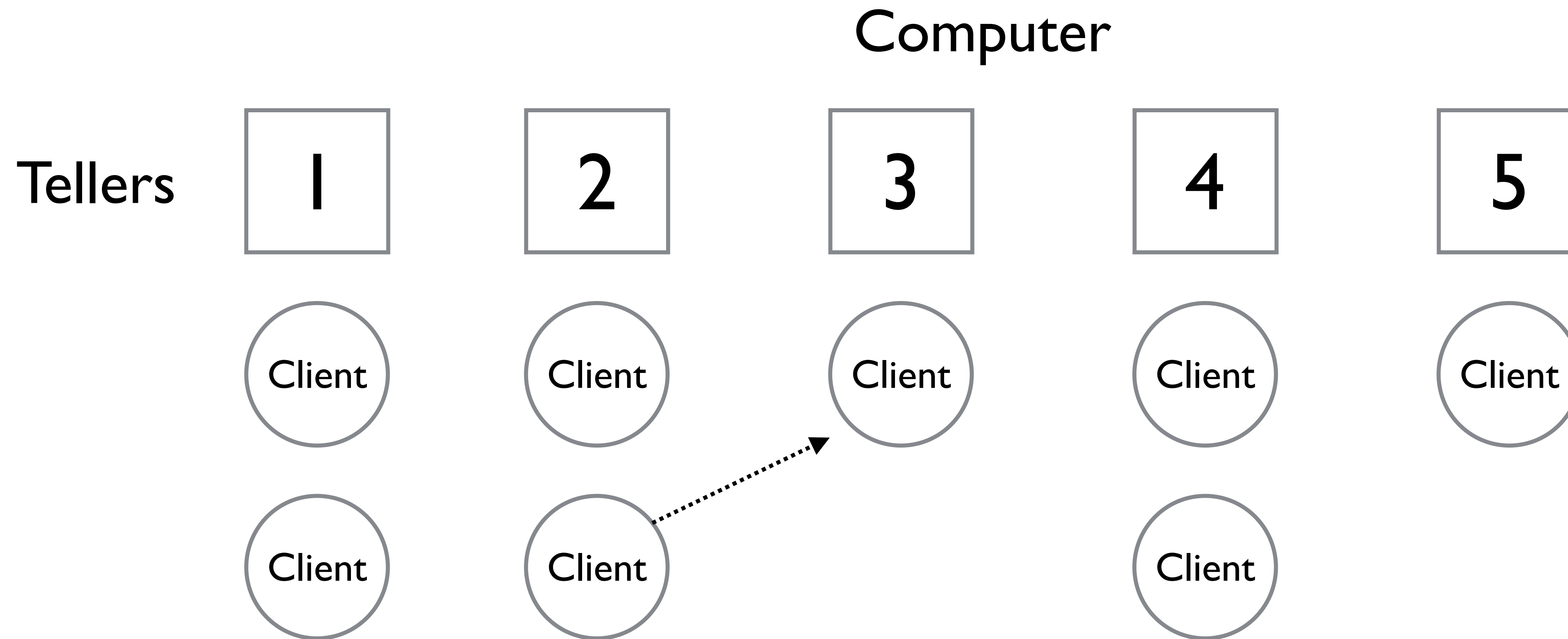
Each teller has their own separate queue. When a customer arrives, they will choose the leftmost shortest queue to enter.

Multiteller Bank with Jockeying



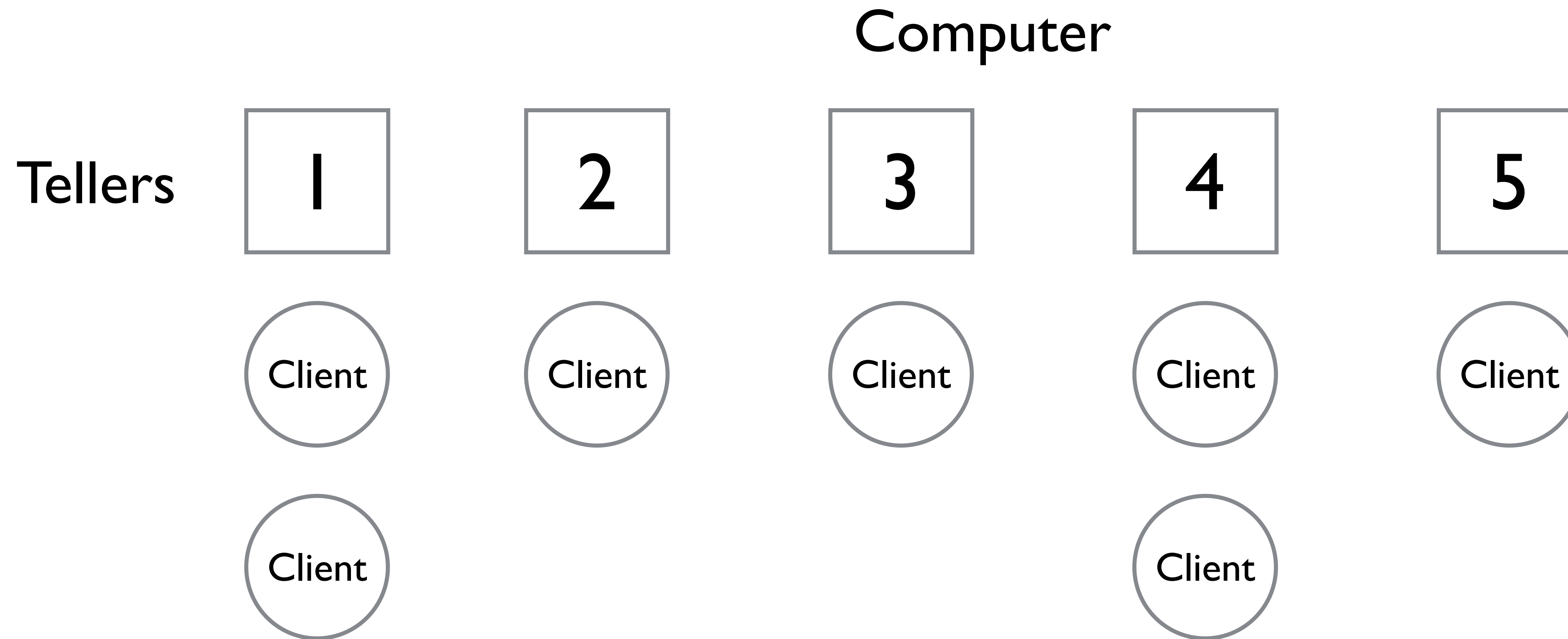
Clients process their queues in order, however if a customer could move up by going into another line, they will.

Multiteller Bank with Jockeying



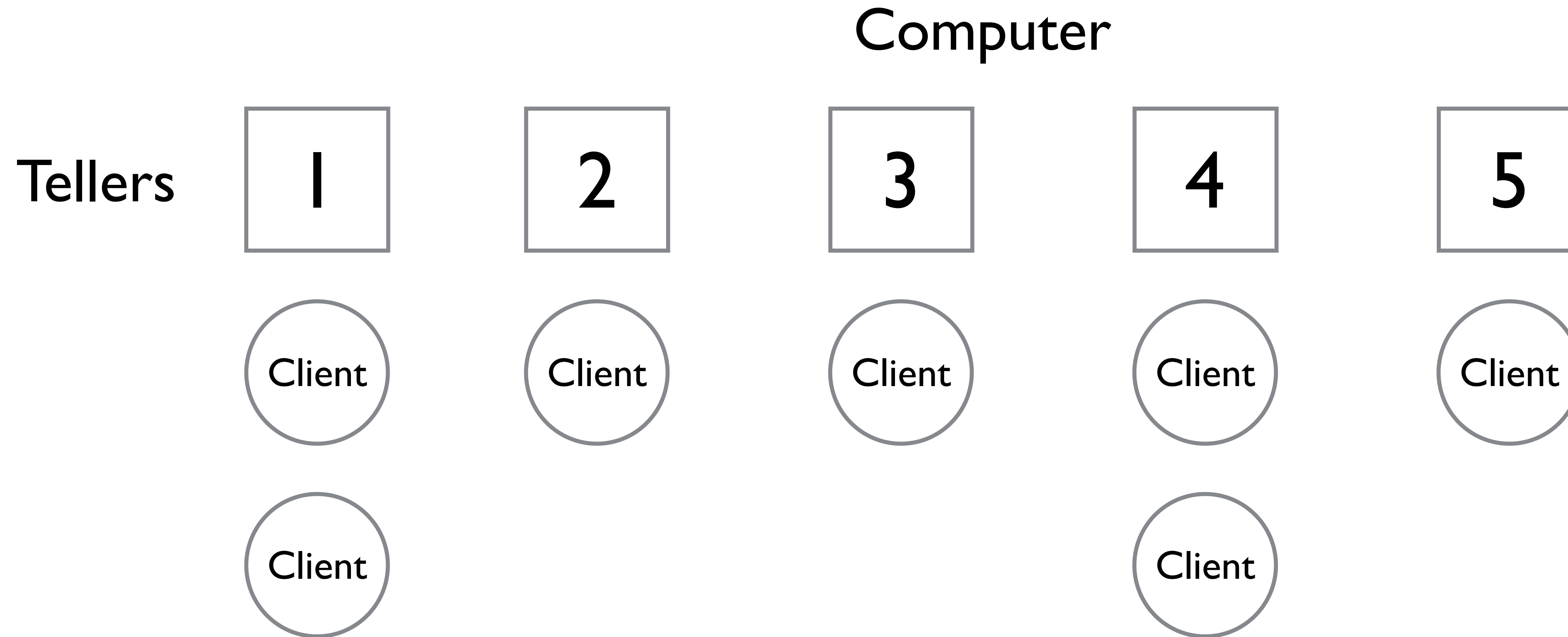
Clients process their queues in order, however if a customer could move up by going into another line, they will.

Multiteller Bank with Jockeying



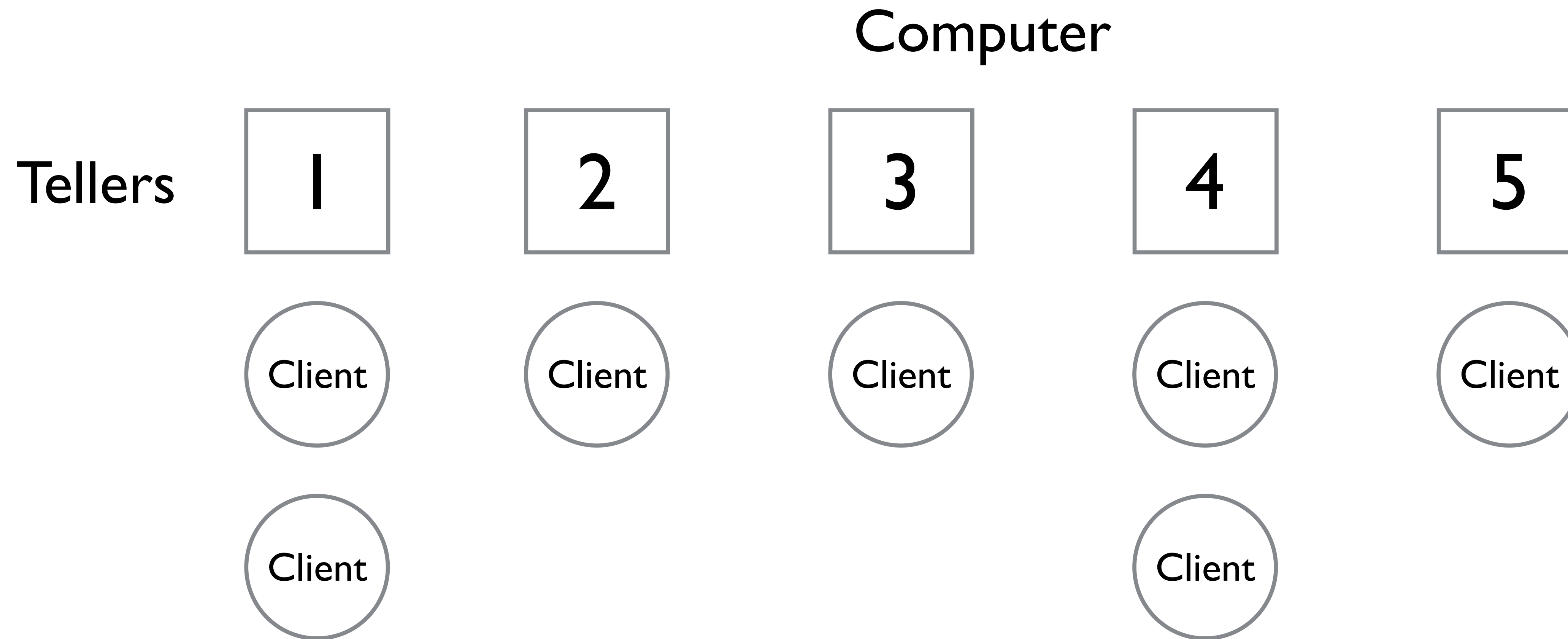
Clients process their queues in order, however if a customer could move up by going into another line, they will.

Multiteller Bank with Jockeying



The rule is formalized: If the completion of service at a teller i causes $n_j > n_i + 1$ for some other teller j, then the customer from the tail of queue j jockeys to the tail of queue i. If there are two or more such customers, the one from the closest, leftmost queue jockeys. If teller i is idle, the customer begins service at teller i.

Multiteller Bank with Jockeying

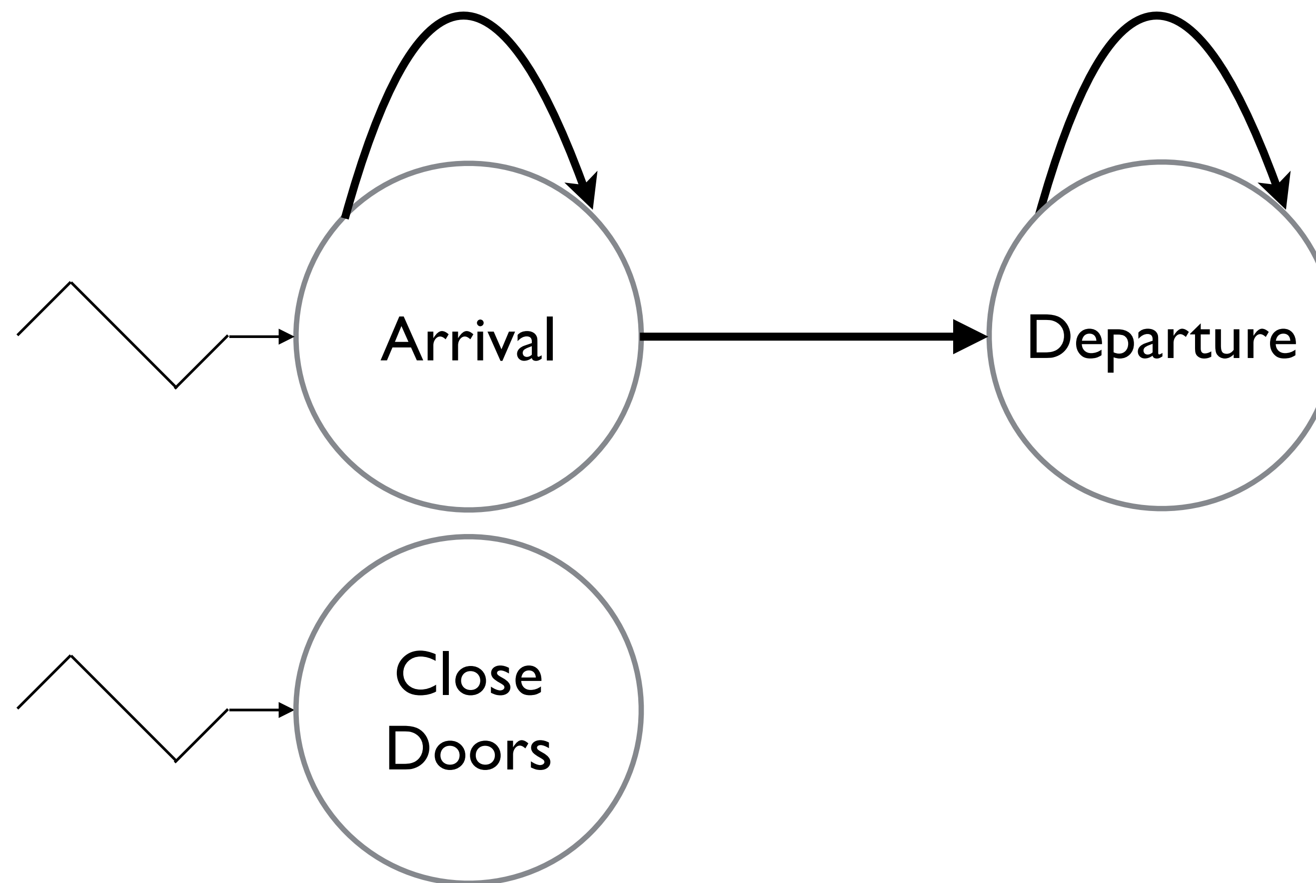


Some example questions that can be analyzed with this simulation are (given a number of tellers, *n*):

1. What is the estimated time-average total number of customers in the queue?
2. What is the expected average delay in the queue?
3. What's the expected maximum delay in the queue?

Bonus: What's the expected average time after 5pm that the bank closes? Note, to calculate this many day-long simulations will need to be run.

Multiteller Bank with Jockeying



We can specify this simulation with another simple event graph. In fact, it's identical to the event graph for the M/M/1 queue.

There are three events:

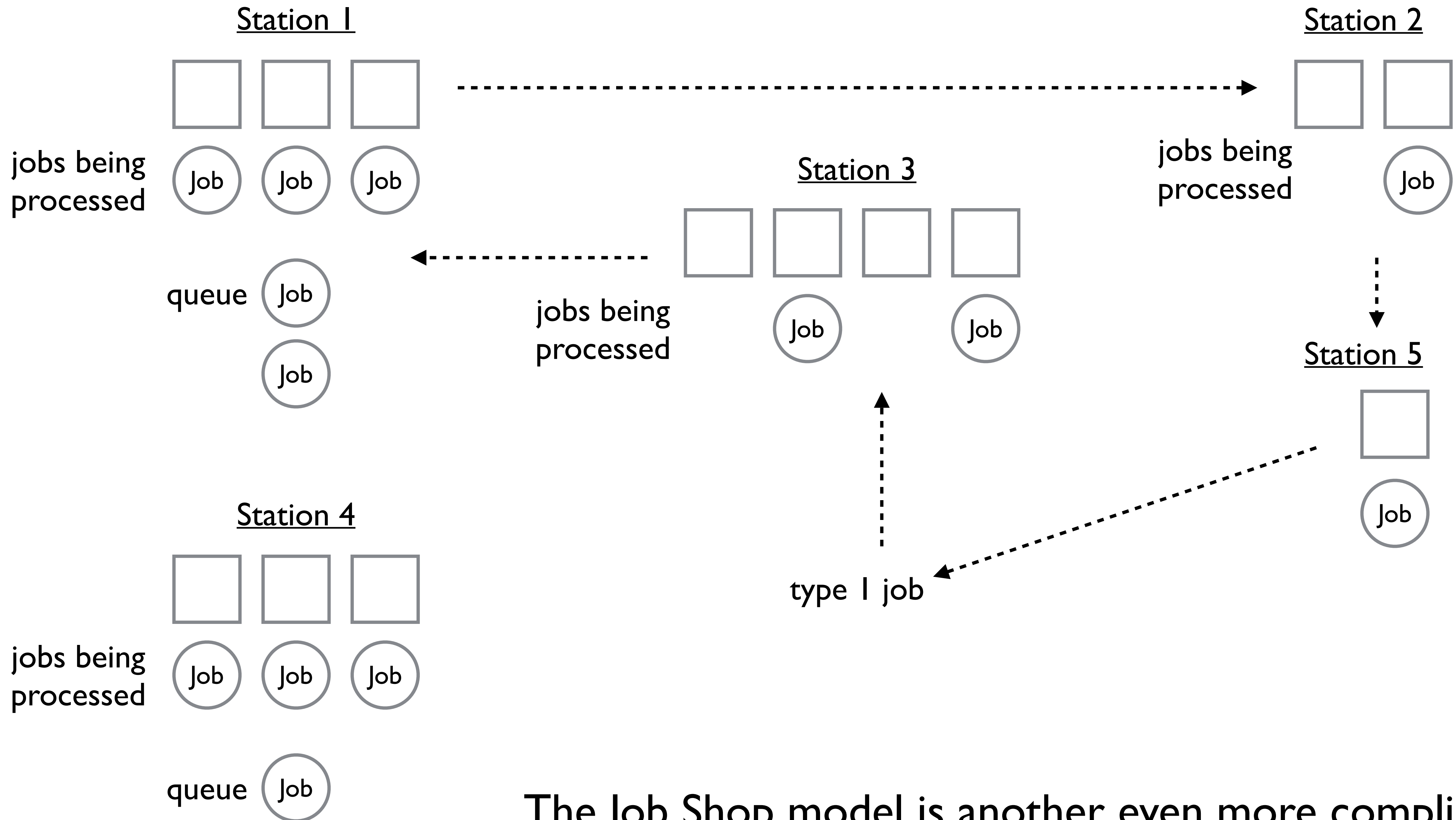
1. Arrival of a customer. This will handle scheduling the customer to a teller, and also generate the next customer arrival.
2. Departure of a customer. This will move the next customer up in line (if there is one) and perform the jockeying. This will also schedule the departure of any customer (the next in line or a jockeyed customer) if they are moved to the front of the line.
3. Closing of the doors at 5, which prevents any further arrivals. The simulation ends when there are no events left.

Required Data Structures

A queue is required for each teller (which can be implemented in a queue). This queue will hold the customers in line and the customer being served for that teller. You don't need as many queues as the book specifies (2n+1), if you implement things using events and classes given the in class assignment code.

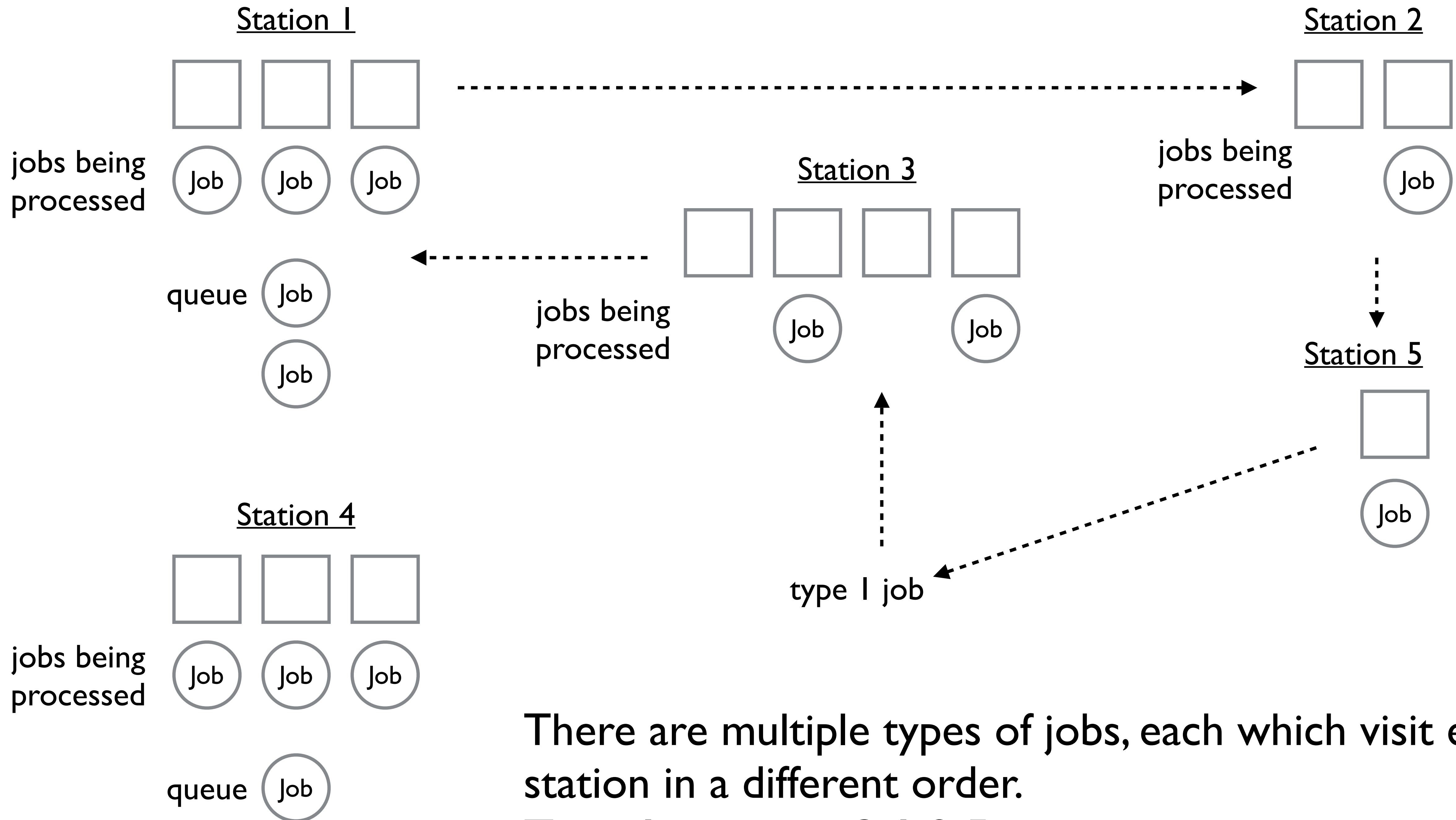
Job Shop Model

Job Shop Model



The Job Shop model is another even more complicated model, simulating jobs as they move through a shop.

Job Shop Model



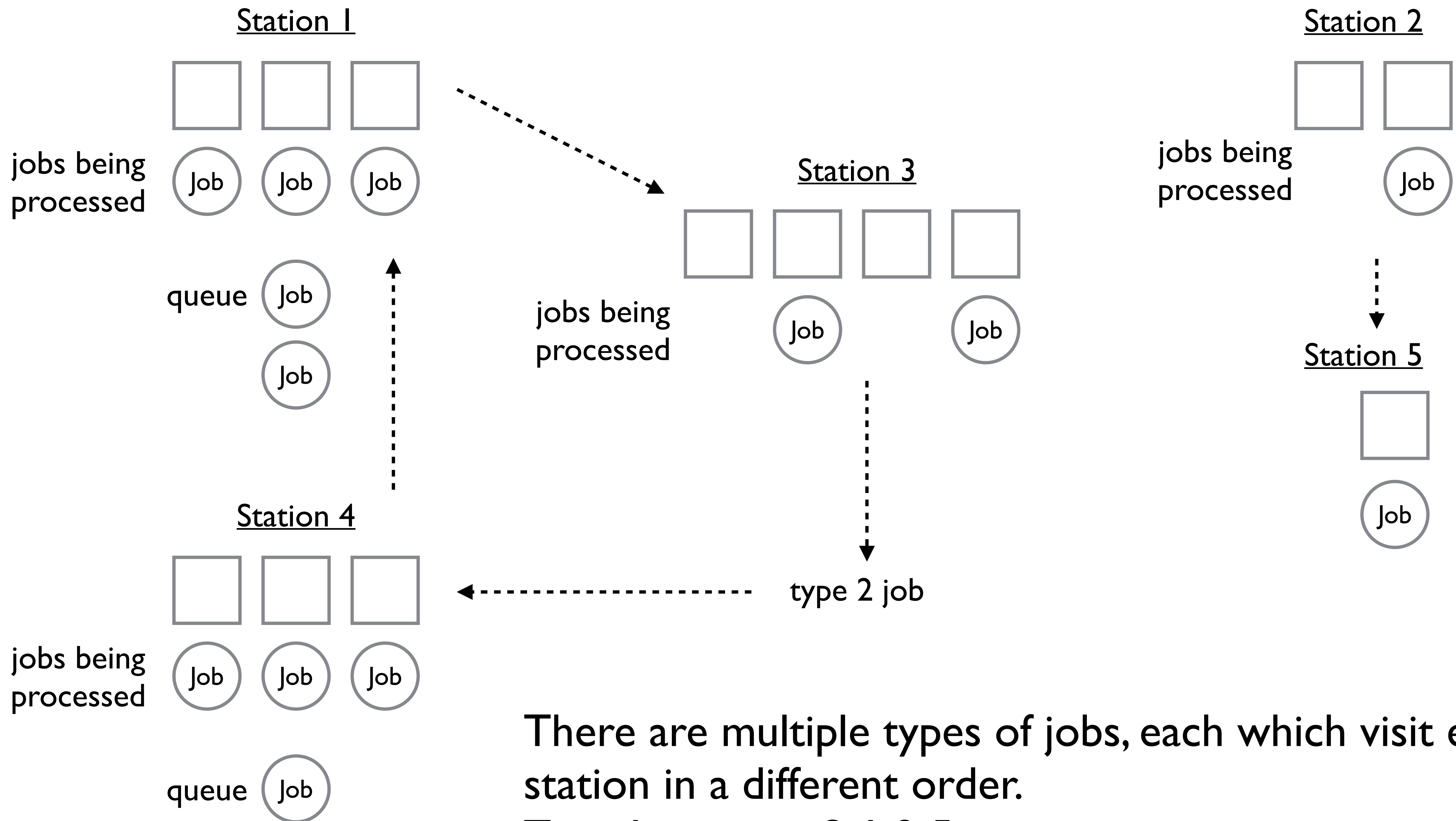
There are multiple types of jobs, each which visit each station in a different order.

Type 1: stations 3 1 2 5

Type 2: stations 4 1 3

Type 3: stations 2 5 1 4 3

Job Shop Model



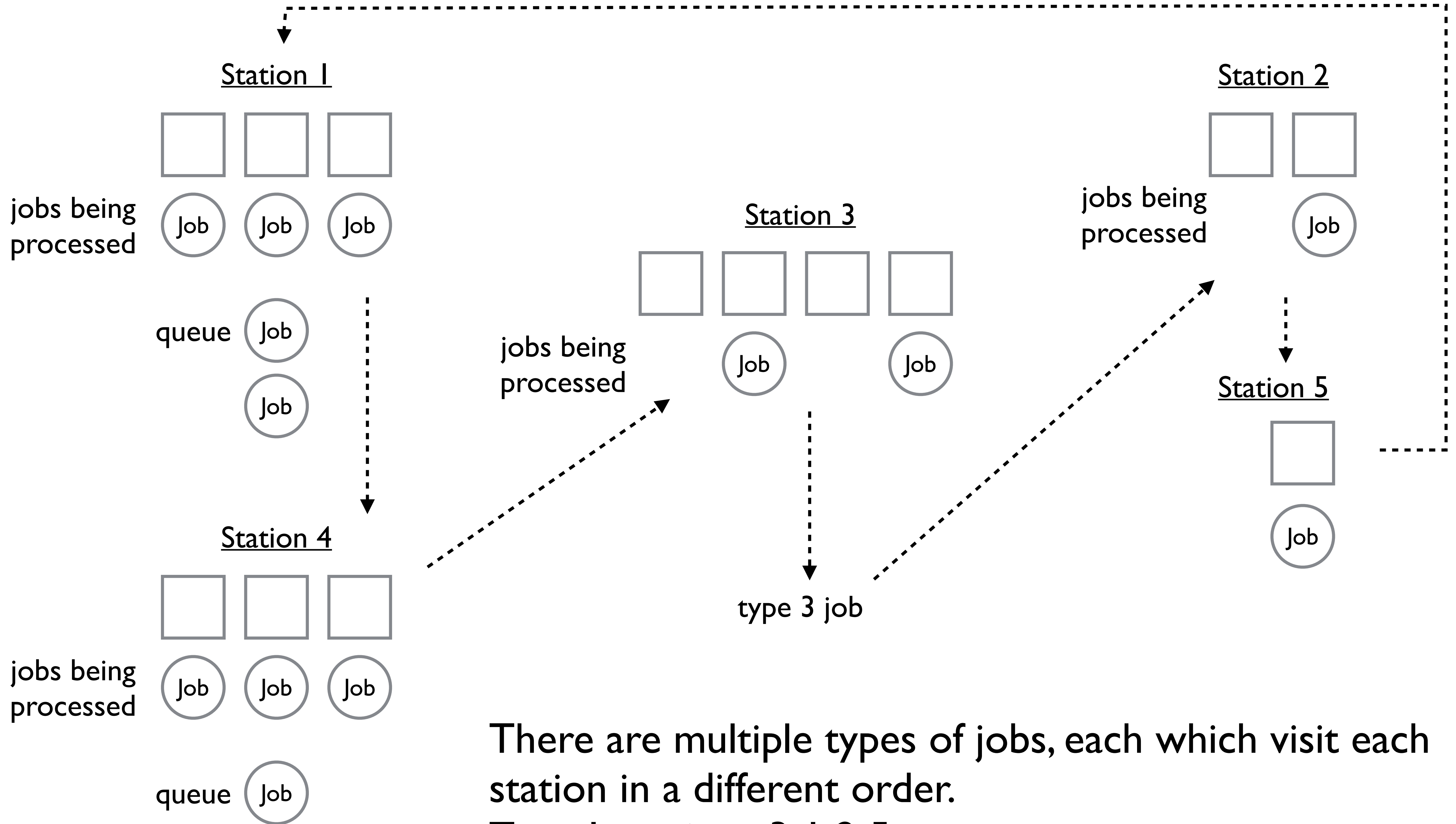
There are multiple types of jobs, each which visit each station in a different order.

Type 1: stations 3 1 2 5

Type 2: stations 4 1 3

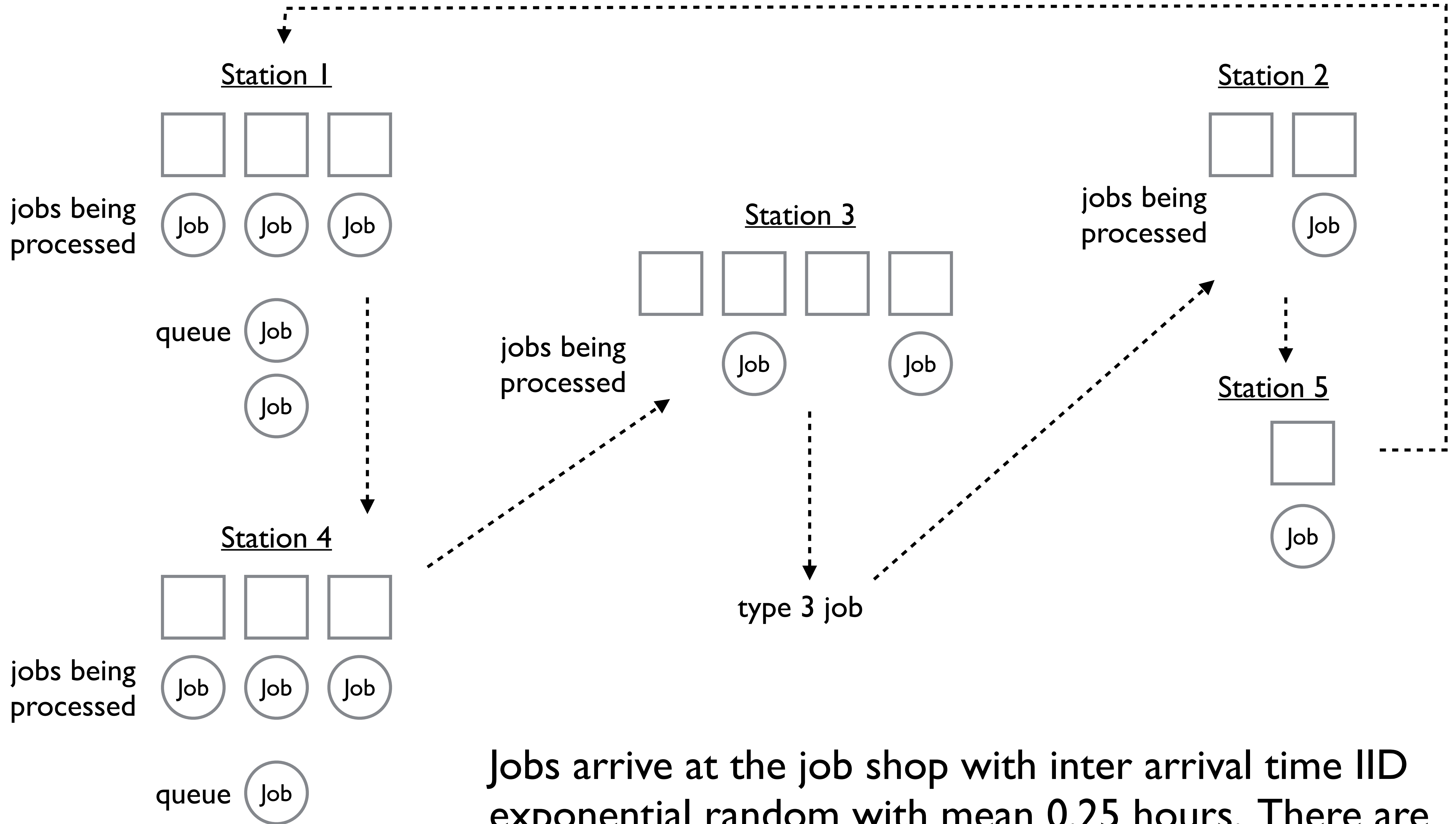
Type 3: stations 2 5 1 4 3

Job Shop Model



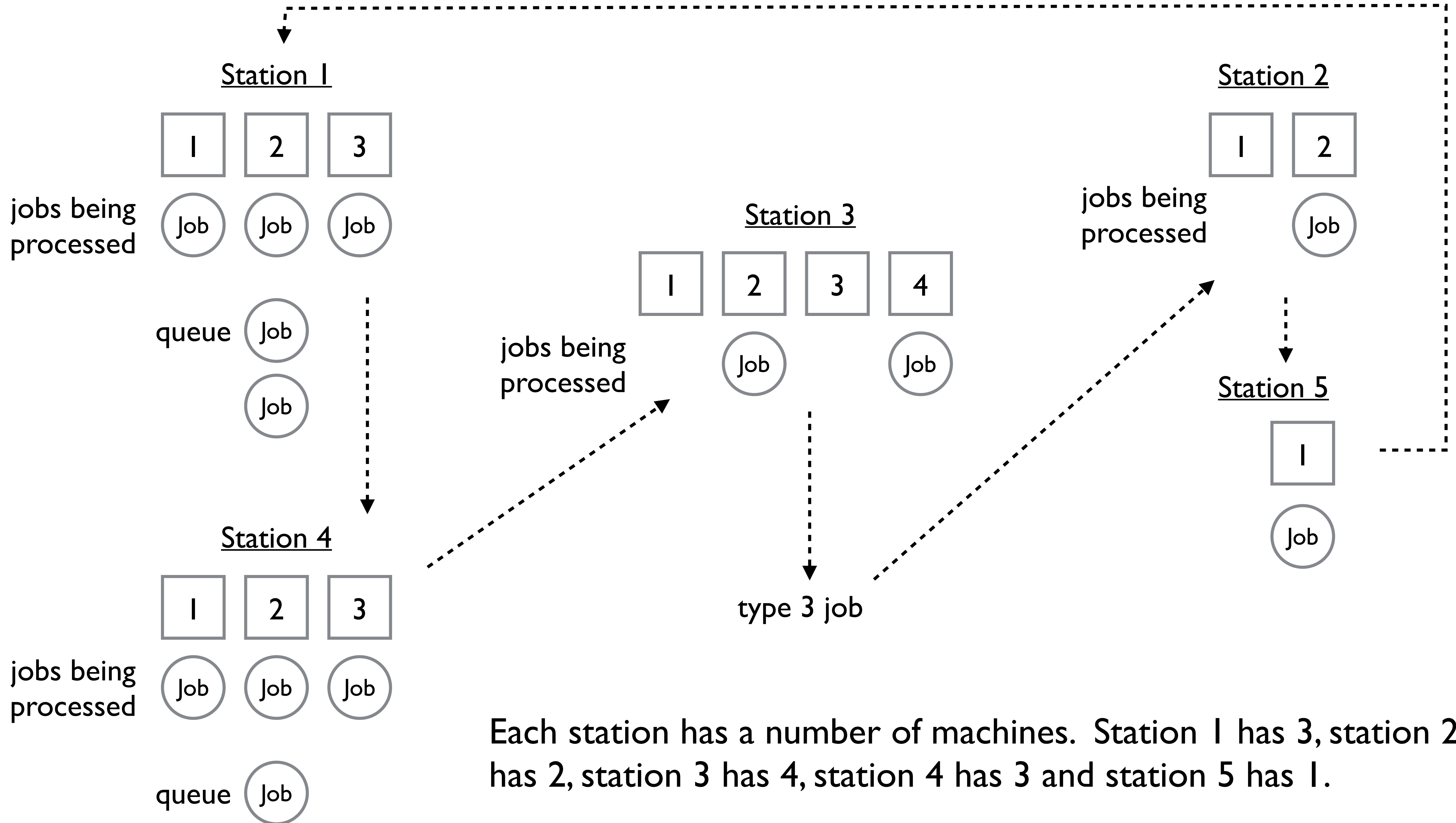
There are multiple types of jobs, each which visit each station in a different order.
Type 1: stations 3 1 2 5
Type 2: stations 4 1 3
Type 3: stations 2 5 1 4 3

Job Shop Model



Jobs arrive at the job shop with inter arrival time IID exponential random with mean 0.25 hours. There are the three types of jobs, with respective probabilities 0.3, 0.5 and 0.2

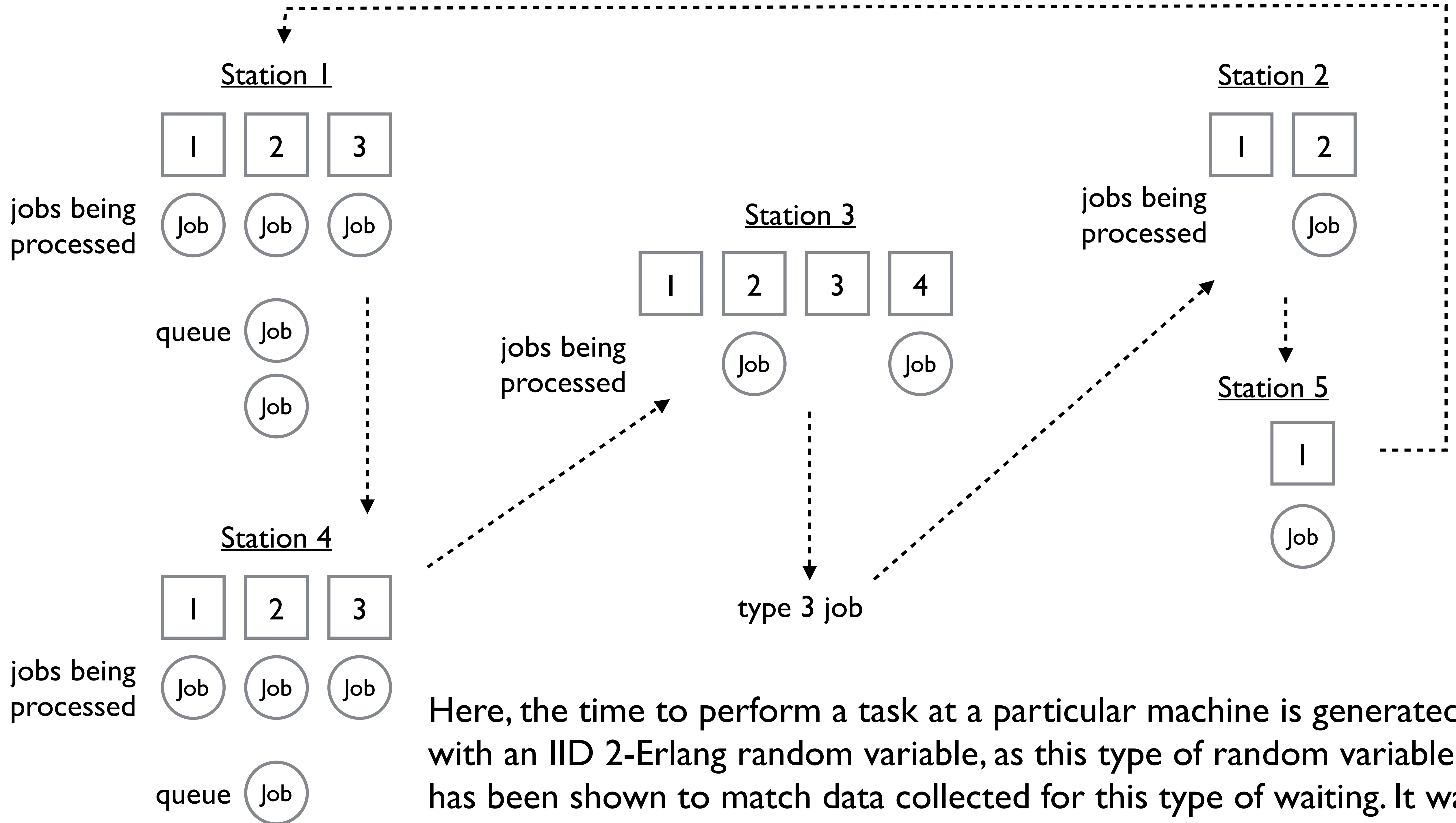
Job Shop Model



Each station has a number of machines. Station 1 has 3, station 2 has 2, station 3 has 4, station 4 has 3 and station 5 has 1.

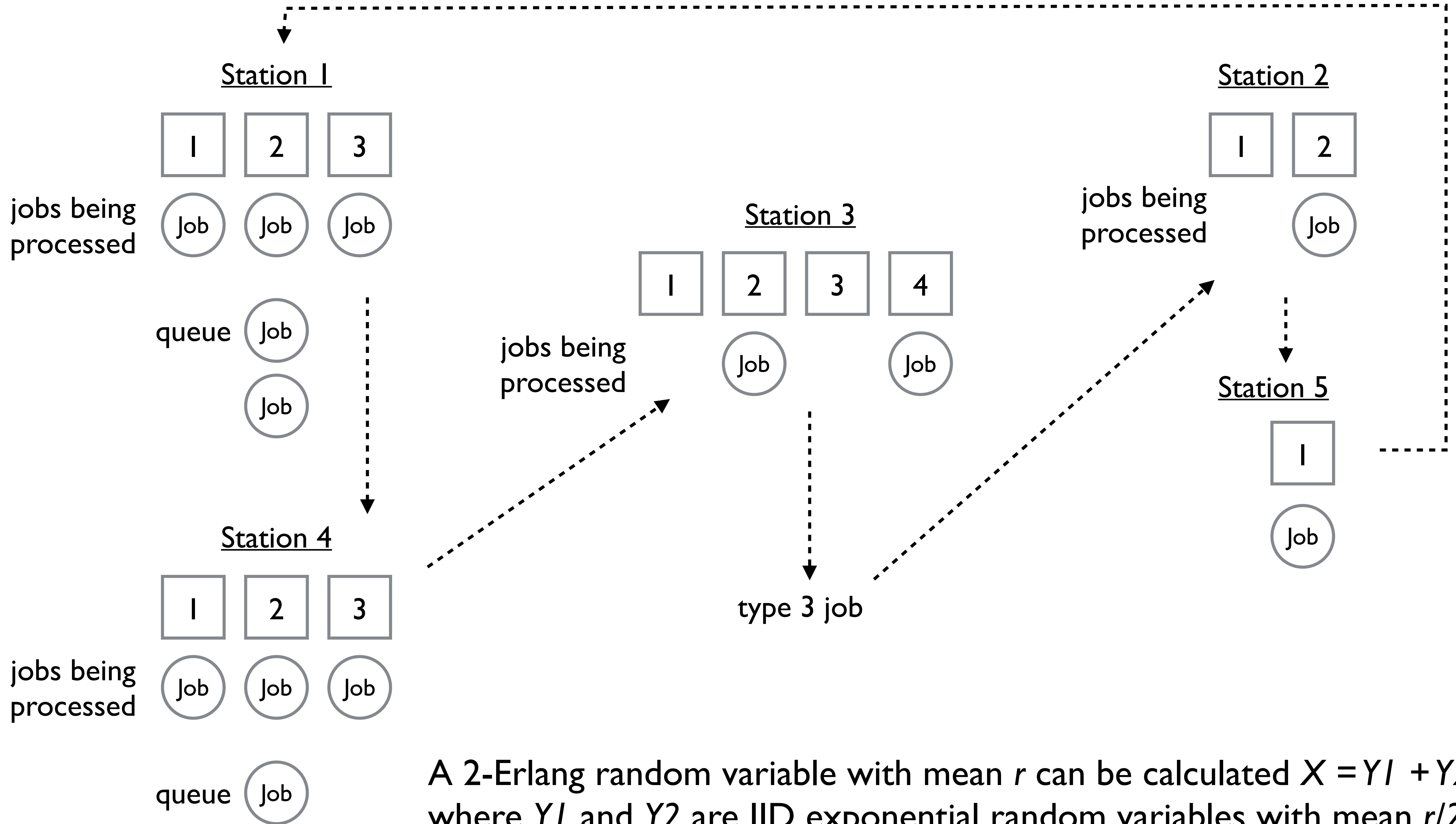
If a job arrives at a station it will enter a FIFO queue if the first machine in the station is in use. If not, it will start on the first machine and progress successively through them.

Job Shop Model



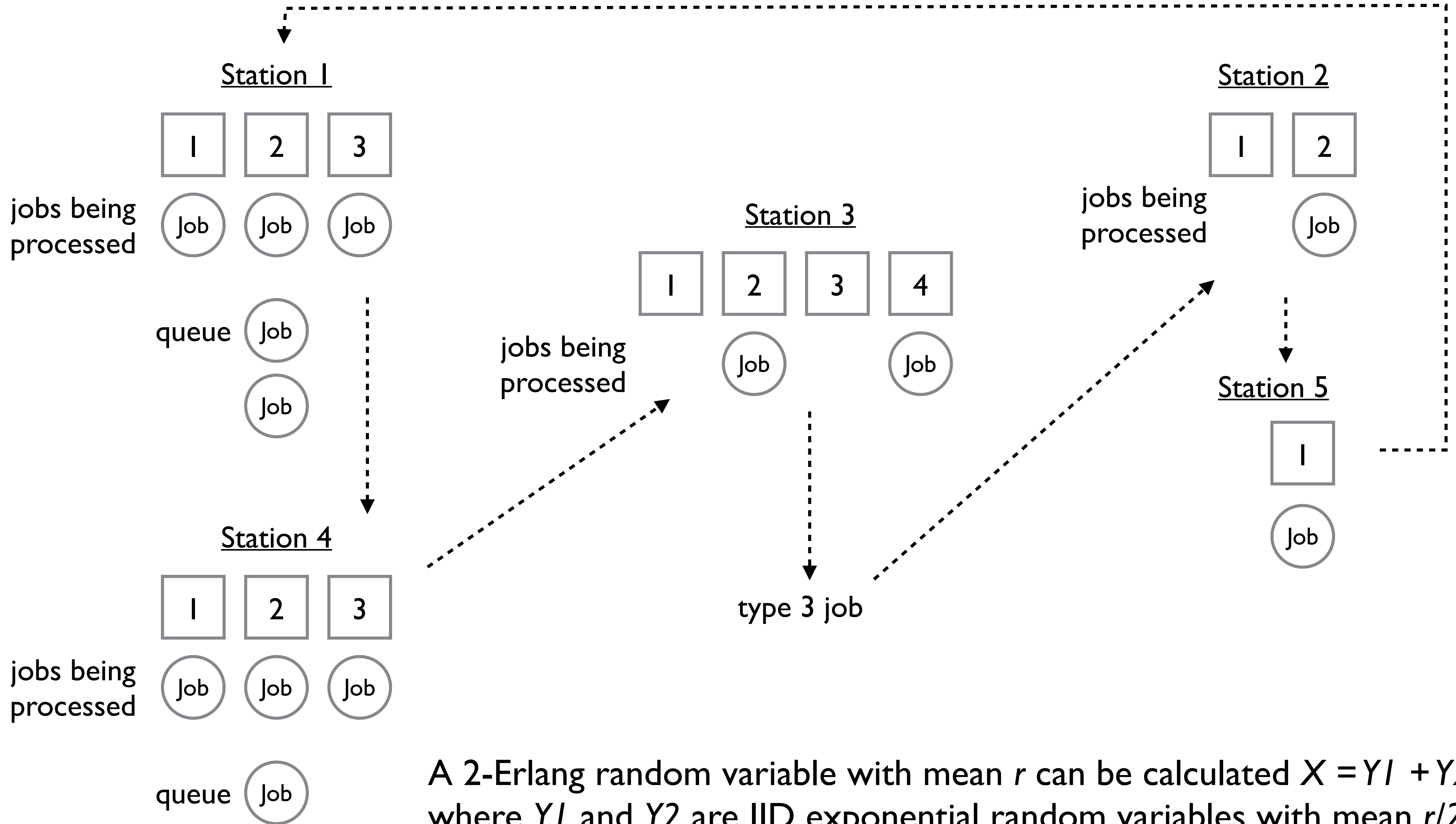
Here, the time to perform a task at a particular machine is generated with an IID 2-Erlang random variable, as this type of random variable has been shown to match data collected for this type of waiting. It was originally discovered by Erlang who gathered data to determine the number of telephone calls that might occur at the same time to operators of phone switching stations (see http://en.wikipedia.org/wiki/Erlang_distribution).

Job Shop Model



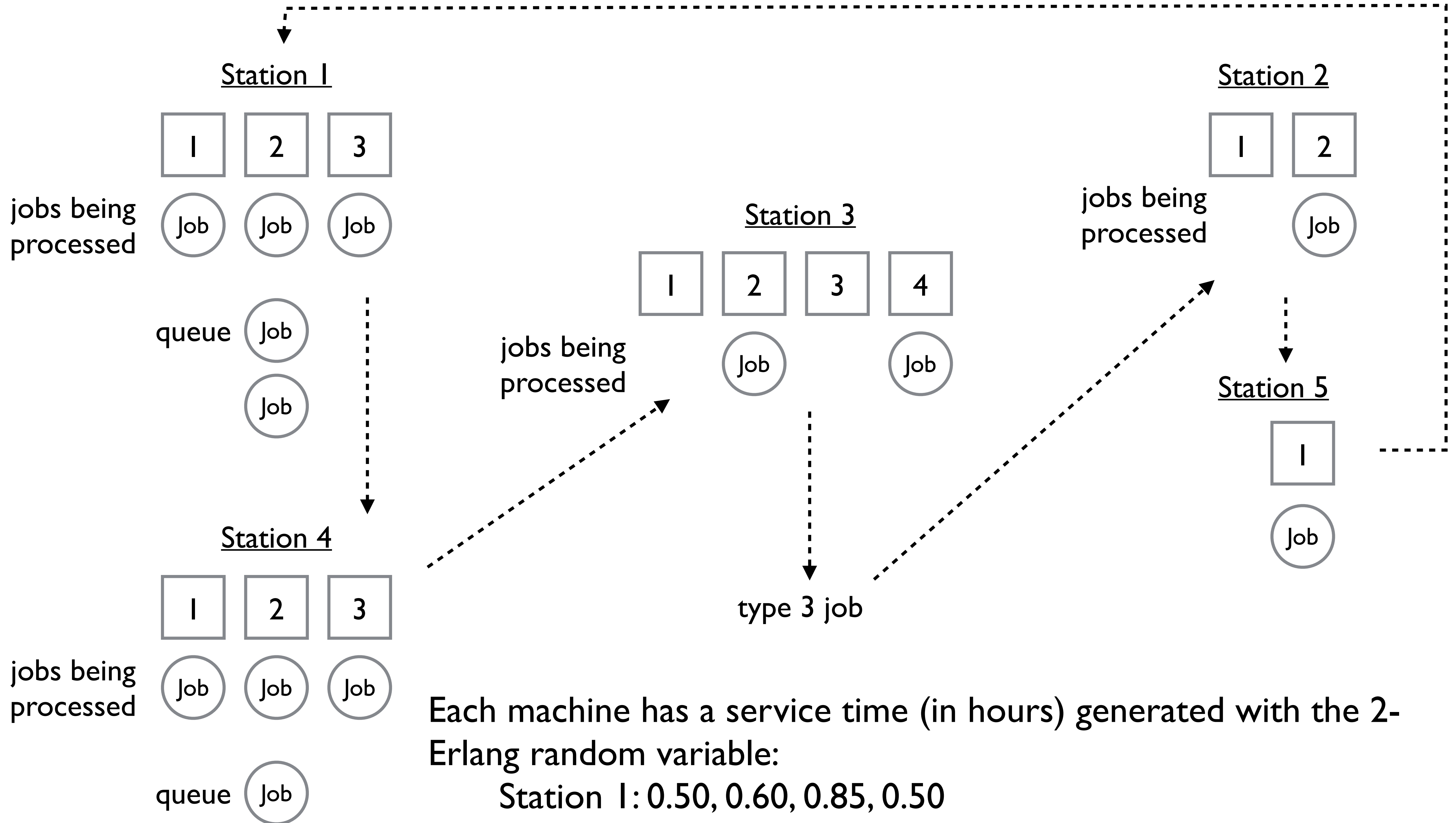
A 2-Erlang random variable with mean r can be calculated $X = Y1 + Y2$, where $Y1$ and $Y2$ are IID exponential random variables with mean $r/2$, alternately, X is known as a gamma random variable with shape parameter 2 and scale parameter r/2.

Job Shop Model



A 2-Erlang random variable with mean r can be calculated $X = Y1 + Y2$, where $Y1$ and $Y2$ are IID exponential random variables with mean $r/2$, alternately, X is known as a gamma random variable with shape parameter 2 and scale parameter r/2.

Job Shop Model



Each machine has a service time (in hours) generated with the 2-Erlang random variable:

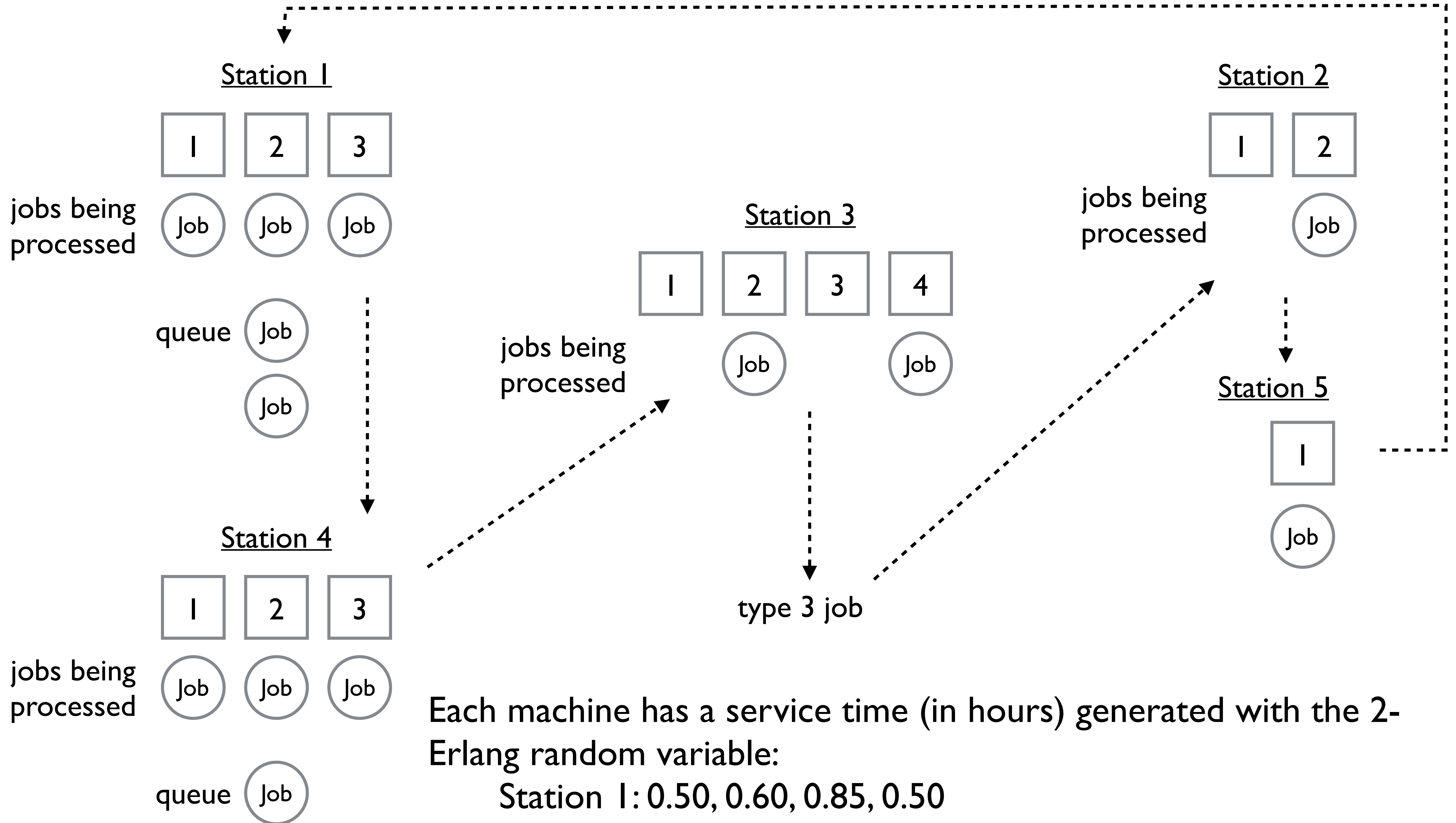
Station 1: 0.50, 0.60, 0.85, 0.50

Station 2: 1.10, 0.80, 0.75

Station 3: 1.20, 0.25, 0.70, 0.90, 1.00

And a job must pass through all stations to move on to the next station.

Job Shop Model

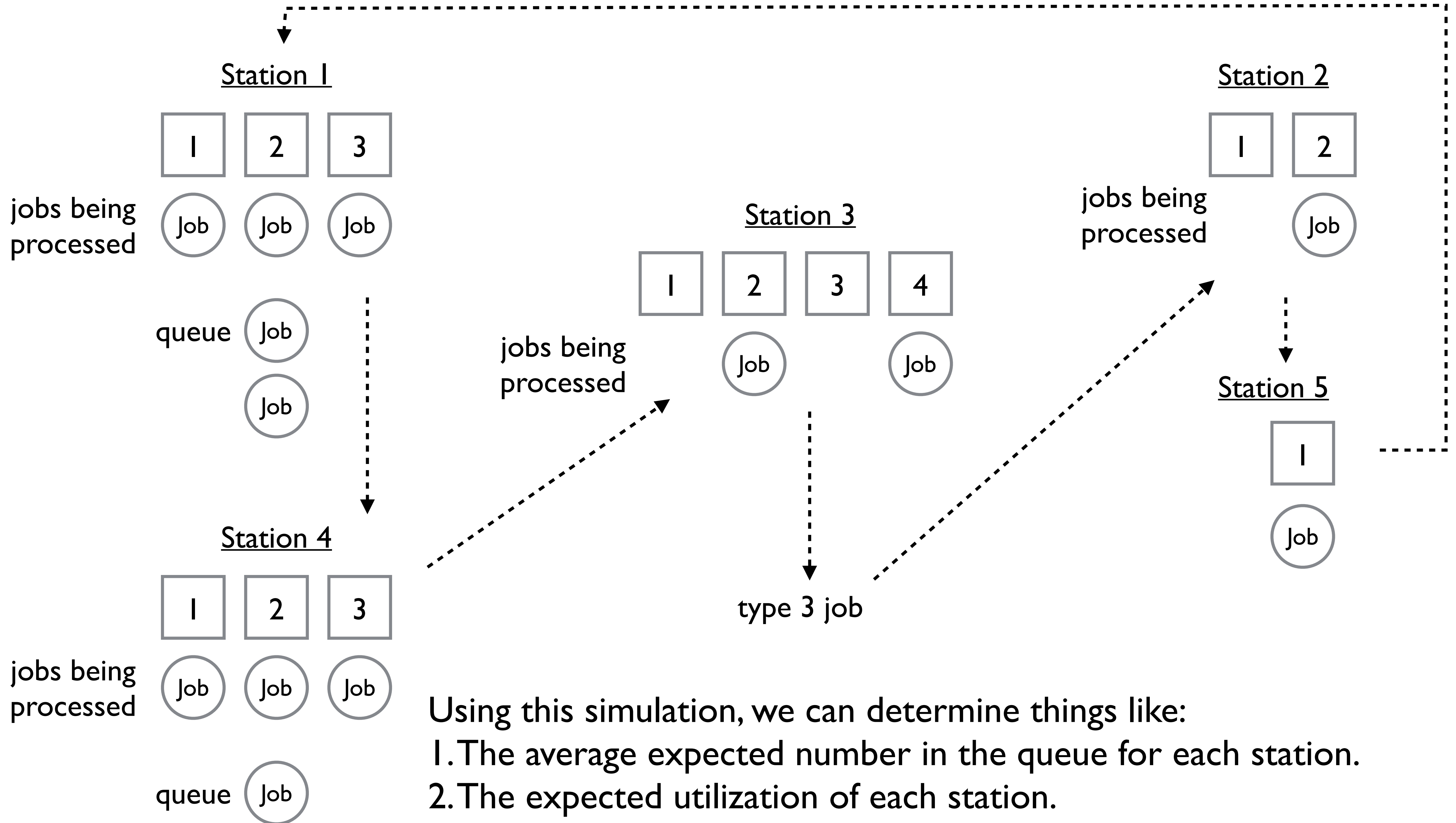


Each machine has a service time (in hours) generated with the 2-Erlang random variable:

Station 1: 0.50, 0.60, 0.85, 0.50
Station 2: 1.10, 0.80, 0.75
Station 3: 1.20, 0.25, 0.70, 0.90, 1.00

And a job must pass through all stations to move on to the next station.

Job Shop Model

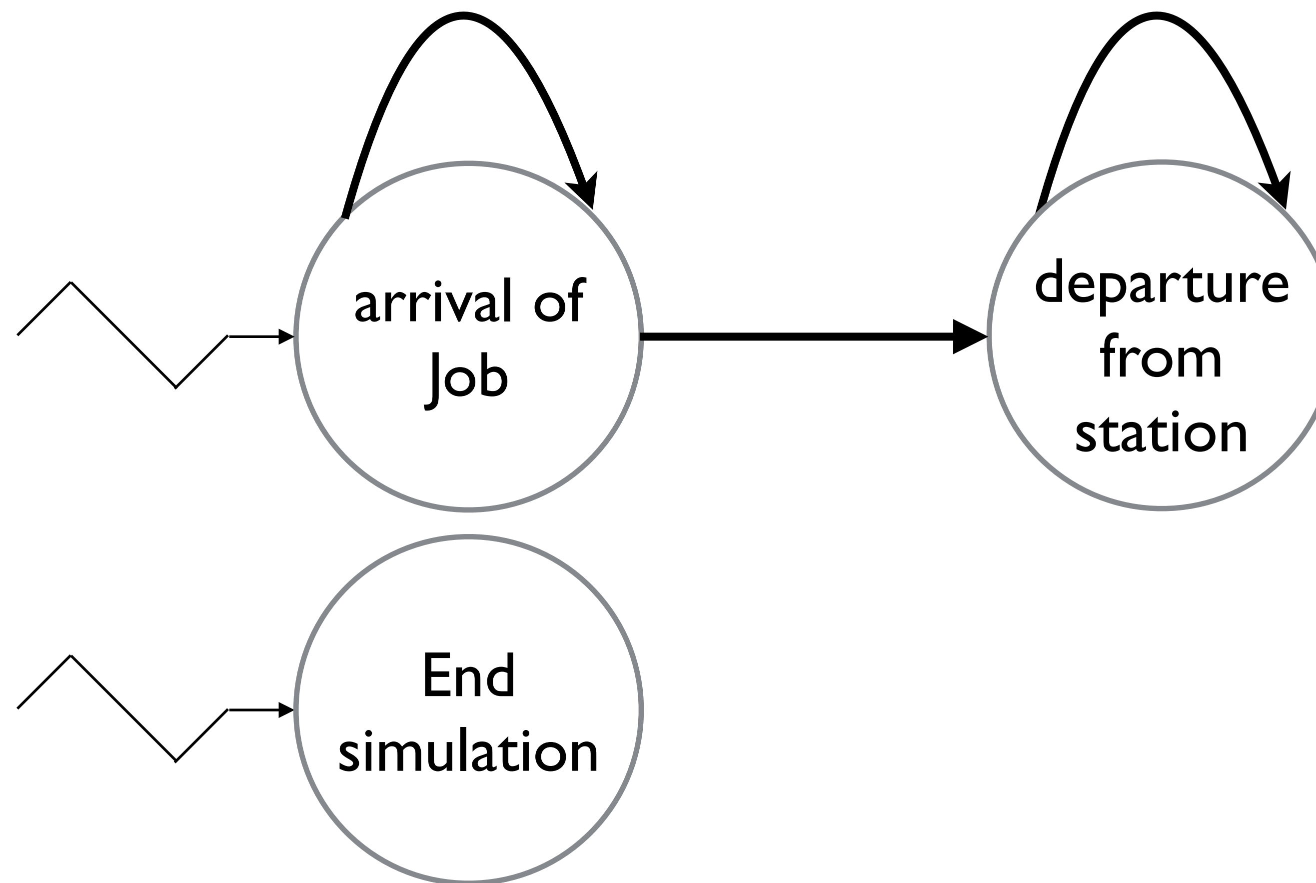


Using this simulation, we can determine things like:

1. The average expected number in the queue for each station.
2. The expected utilization of each station.
3. The expected average delay in queue for each station.

We could then use this information and simulation to determine which machines to purchase given a certain amount of money to best increase production, etc.

Job Shop Model



The event graph again looks similar!

There are three events:

1. Arrival of a job to the system. This will handle scheduling the job to its given first station, and also generate the next job arrival.
2. Departure of a job from a station. This will move the next job up in the FIFO queue (if there is one) and start processing of then next job at the station. This will also put the job in the next stations queue, or schedule it's departure.
3. Ending of the simulation.

Required Data Structures

For this, a queue is required at each station, to hold the jobs being processed. When a job starts being processed we can calculate the time for the next one to begin processing, moving things through the machines. We will also need data structures for each station, and the machines within it to keep track of what is where.

Conclusions

Conclusions

This lecture went over various more complicated simulations, to get an idea of how these can be described. Interestingly, they all can boil down to quite simple event graphs, however the actions of each event get progressively more complex. You'll have to implement one of these for your next assignment.