**194.076 Modeling and Simulation**

# Discrete Event Simulation Case Study: Dining Philosophers Problem

Assylbek Tleules, 12432843 - Curriculum Number: UE 066 645 [*]
Arian Ahmad, 01529233 - Curriculum Number: UE 066 926 [†]
Hasan Hüseyin Günes, 12229237 - Curriculum Number: UE 066 645[‡]

February 8, 2026
Supervisor: Madlen Martinek

[*]TODO: ADD DESCRIPTION
[†]TODO: ADD DESCRIPTION
[‡]TODO: ADD DESCRIPTION

## Contents

## 1 Introduction

When multiple processes or systems share a limited set of resources, several problems can occur. If the coordination is done poorly, the whole system can get stuck in a deadlock, where no further progress is possible. In addition, if the system cannot ensure a certain fairness level, individual components may experience very long waiting times compared to others, also called starvation. The Dining Philosophers problem[1] is a simple but realistic example of such situations. It helps to understand why these issues can occur and provides a useful basis for copmaring different methods that aim to prevent them.

As shown in Figure 1, in the Dining Philosophers problem, philosophers sit around a table with one chopstick placed between each pair of neighboring philosophers. Over time, each philosopher repeats the cycle of thinking, being hungry and eating. To eat, a philosopher needs two chopsticks (the left and right one), while one chopstick can only be held by one philosopher at a time. Since neighboring philosophers share one chopstick, such conflicts can occur. Depending on the rule for picking up chopsticks, a situation could arise where every philosophers holds one chopstick and waits for the second one to become available. Another issue regarding the fairness could be that some philosophers may wait for much longer then others, when they want to eat.

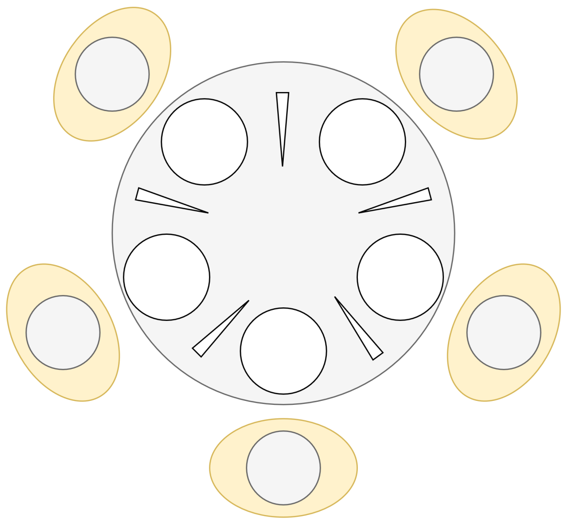

Figure 1: Dining philosophers sitting at a round table, each with a bowl of food and chopsticks.[2]

Based on a discrete event simulation (DES) model, this work aims to implement the Dining Philosophers system and analyze its behavior under different configurations and rules. First, the base concept without any additional methods is inspected. Then, methods are introduced to the concept, which aim to detect and prevent deadlocks and to reduce starvation cases. Finally, we consider further scenarios such as varying the number of philosophers and allowing rule violations by individual philosophers. The different settings are compared using the simulation outcomes.

[1] https://en.wikipedia.org/wiki/Dining_philosophers_problem
[2] https://diningphilosophers.eu/

## 2 Model

As already mentioned, a DES approach is used to realize this problem. The model is implemented in Python, and the source code is publicly available in a GitHub Repository[3].

### 2.1 General Modeling Approach

The DES contains three core components: entities, events and event schedule. The entities are philosophers and chopsticks. Philosophers can be in the states `THINKING`, `HUNGRY` or `EATING`, whereas the chopsticks form the shared resources that can be acquired and released. Events are used to illustrate changes and resource interactions, such as a philosopher finish thinking and becoming hungry or attempting to acquire chopsticks. The event schedule is implemented as a time-ordered priority queue and defines the timeline of the events. The simulation time progress is given by repeatedly processing the next event in time-oriented order.

The initial setup looks as follows: five philosophers, each starting in the `THINKING` state, are seating around a round table, with one chopstick placed between neighboring philosophers (= five chopsticks). Random delays are introduced every time a philosophers starts thinking or eating, to model asynchronous, more realistic behavior, specifically for thinking durations, acquisition retries when chopsticks are unavailable, and eating durations. These delays are mainly sampled from exponential distributions, while in the case, where additional scenarios are inspected, these distributions are varied (see Section 2.5).

The default delay distributions sampled from are as follows: We model thinking times using an exponential distribution with mean 5, and eating times using an exponential distribution with mean 3. Exponential distributions are a standard choice for stochastic durations, since they are non-negative and capture memoryless, asynchronous timing. The selected means reflect our assumption that philosophers typically spend more time on their main hobby, thinking, than eating. Unless stated otherwise, one time unit corresponds to one minute. Most simulation runs use a time horizon of 120 time units.

For the thinking times, we chose an exponential distribution with mean 5 and the eating time also en exponential distribution with mean 3. Generally, one time units in our case represents one minute. In most cases, one simulation has a duration of 120 time units.

### 2.2 Base Model

The base model captures the default simulation behaviour and **is not** a deadlock-prevention strategy. Each philosopher starts in the `THINKING` state. After a random thinking duration, the philosopher becomes `HUNGRY` and attempts to acquire two chopsticks sequentially: first one chopstick (left by default), then the second one. If the second chopstick is unavailable, the philosopher keeps the first chopstick and retries after a short random delay. Once both chopsticks are acquired, the philosopher enters the `EATING` state for a random duration, releases both chopsticks, and returns to `THINKING`.

Since no coordination rule is enforced, the system may either progress smoothly or enter a deadlock, depending on the timing of events. In practice, this outcome is sensitive to the random seed: some runs are effectively "lucky", while others reach the deadlocked configuration. We use this baseline as a reference point for comparing explicit prevention strategies (e.g., a priority rule such as lowest-ID first), highlighting the trade-off between simplicity and guaranteed progress.

[3]`https://github.com/tleules23/ModSim2025`

## 2.3 Deadlock Detection / Prevention

The goal of this part is to detect and mitigate deadlocks while still allowing philosophers to eat. In our model, a deadlock occurs when all philosophers are in the `HUNGRY` state and each of them holds exactly one chopstick. In this configuration, no philosopher can acquire the second chopstick, and the system stops making progress.

Deadlock detection is executed inside `process_event` after each event is handled, so the check is performed at every state change in the system. The function `check_deadlock()` implements the structural definition of deadlock by counting how many philosophers are in the `HUNGRY` state and how many of them hold exactly one chopstick. A deadlock is detected when both values equal the total number of philosophers.

When a deadlock is identified, the simulation increments `deadlock_count`, stores the current simulation time in `deadlock_times`, and writes a corresponding entry to the event log. `halt_on_deadlock=True` stops the simulation immediately after the first detection in order to illustrate the deadlock situation clearly. In other configurations, the simulation continues running so that different prevention strategies can be compared under the same detection logic.

We compare three different approaches for chopstick acquisition and deadlock handling:

**Left-first (baseline rule):** In the baseline model, each philosopher attempts to acquire chopsticks in a fixed order: left first, then right. This behaviour is implemented in `get_chopstick_order()`, which returns the tuple `(left, right)` when no priority strategy is active. Chopstick acquisition is handled in two steps by `try_acquire_chopsticks()`. The philosopher first tries to take the initial chopstick if it is available and, if the second chopstick cannot be obtained immediately, retries after a short delay.

Because all philosophers follow the same symmetric rule, randomised start times often result in reasonably balanced eating counts. However, this approach does not eliminate circular waiting. In the case of synchronised starts, all philosophers may pick up their left chopstick at the same time and then block each other indefinitely while waiting for the right one. As a result, the left-first strategy can appear fair, but it does not guarantee progress.

**Priority ordering (lowest-ID first):** In this strategy, each philosopher always attempts to pick up the lower-ID chopstick first. This behaviour is implemented in `get_chopstick_order()`, which returns the tuple $(\min(left, right), \max(left, right))$ when `prevention_strategy == "priority"`. By enforcing a single global ordering of resources, the circular-wait condition is eliminated, and deadlock is prevented by design.

The main drawback of this approach is reduced fairness. Chopstick 0 becomes a shared bottleneck, and philosophers that depend on it (in particular IDs 0 and 4) may experience longer waiting times. In the results, this effect can appear as biased eating counts across philosophers. The trade-off is therefore clear: guaranteed deadlock prevention comes at the cost of potential unfairness.

**Timeout strategy:** In the timeout approach, a philosopher who holds one chopstick waits only a fixed amount of time to acquire the second one. In the implementation, when the second chopstick is unavailable and `prevention_strategy == "timeout"`, the simulator schedules a `TIMEOUT_RELEASE` event after `timeout_duration`. If the philosopher is still in the `HUNGRY` state at that time, `handle_timeout_release()` releases any held chopstick and schedules a retry.

This mechanism breaks circular waiting without enforcing a global resource ordering. While it does not provide a strict mathematical guarantee of deadlock freedom, deadlocks are resolved in practice once timeouts are triggered. Compared to the priority strategy, it also reduces systematic bias, since philosophers are not locked into a fixed acquisition order. The main cost of this approach is additional retries and waiting, which can slightly reduce overall throughput.

## 2.4 Starvation Avoidance

Deadlocks are not the only issues that can occur during the simulation runs. Because of the mostly stochastic nature of the dining philosopher problem, given by the unordered sequence of chopstick acquisitions by the philosophers, it is possible that each participant in the simulation run is waiting indefinitely for his turn to eat. A fitting example would be the case where both neighbors of the same philosopher always take the chopsticks first. Long enough waiting times can lead to starvation, which for the dining philosophers problem is considered as a failed run if even one philosopher starves. In our model, starvation does not directly imply death or termination; it refers to unfair scheduling where a philosopher may wait indefinitely while others continue to eat. In the current baseline implementation biased anyone can attempt to grab free chopsticks and fast or lucky philosophers may finish thinking earlier, retry more aggressively or acquire chopsticks repeatedly.

This possibility of indefinite waiting times requires the addition of a mechanism that ensures fairness between the members for acquiring the chopsticks.

### 2.4.1 Fairness Mechanism

In order to combat starvation, the model is using an aging-based scheduling strategy in which hungry philosophers, stored in a queue, are dynamically prioritized by the time elapsed since their last eating event. By default, each philosopher keeps track of their most recent eating event time. Before every chopstick acquisition attempt the model calculates the passed time since the last eating event for every hungry philosopher:

$$priority = current\_time - last\_ate\_time$$

Based on that, the hungry philosopher with the largest elapsed time since the last eating event is considered the highest-priority candidate. For the model, this means that only this philosopher is allowed to acquire the chopsticks for eating, and other hungry philosophers must wait until they become the highest-priority candidate. After finishing eating, the philosopher's last eating time is updated to the current simulation time and they are removed from the hungry queue. This ensures that hungry philosophers who have waited the longest don't indefinitely have to wait for their order to eat and starve out. The additional fairness in the model, indicated by a lower deviation of the eating counts between philosophers, comes with a set of trade-offs, specifically the longer overall waiting times and the lower average and total eating counts due the non-arbitrary selection for the chopstick acquisition.

```
When a philosopher becomes HUNGRY:
    add philosopher to hungry_queue

When chopstick acquisition is attempted:
    select philosopher p from hungry_queue
        with maximum (current_time    p.last_ate_time)

    if p can acquire both chopsticks:
        p eats
        update p.last_ate_time
        remove p from hungry_queue
    else:
        schedule a retry for p
```

Listing 1: Priority-based starvation avoidance (pseudocode).

## 2.5 Scenario Variations

After defining the base model and the different mechanisms to cope with issues like deadlocks and starvation, we extend the analysis beyond the standard setting to inspect differences in outcomes under more challenging conditions. Applied extensions are described below.

We will vary the number of philosophers and look at how the results are changed, especially in terms of deadlock runs.

Next, we will vary stochastic characteristics of thinking times and eating times. In the context of thinking times, we want to look at two different uniform distributions: $unif(2, 30)$ and $unif(2, 4)$. We chose these distributions with the corresponding limits to ensure that we inspect a higher and lower scale for thinking durations.

In addition, we will also look at two different configurations of a a truncated normal distribution for eating times:

- $norm(8, 3)$ with truncated values resulting in range [1,16]
- $norm(15, 3)$ with truncated values resulting in range [1,30]

We chose the normal distribution here, since it seems to also be realistic, when it comes to eating times. In most cases, the eating time should not change that much on average, while there could also be cases, where the philosopher just takes a snack, or eats more than normally. We think, the normal distribution covers such cases.

As an extension of the base rules, we introduce selfish philosophers to the whole concept. In this context, selfish means that a philosopher may not follow some of the ground rules of the system. Some of the characteristics of the selfish philosophers are listed below.

- Ignoring the Timeout strategy, used for Deadlock Prevention
- Ignoring the Priority Ordering Strategy and just deciding randomly on which chopstick to take first
- No immediately dropping of chopsticks after finishing eating

Apart from that, we will again extend the basic rules and introduce conspirator philosophers. Conspirator philosophers are two philosophers, who try to block the one victim philosopher in between them, by keeping the victim's required chopsticks unavailable as often as possible. The conspirator philosophers will have the following characteristics:

- Conspiracy is active while the victim is hungry. Otherwise, the conspirators are acting normally.
- Targeted acquisition: When the victim is hungry, each conspirator prioritizes the victim-critical chopstick, which the corresponding conspirator can acquire.

- Targeted holding: After finishing eating, a conspirator keeps holding the victim-critical (target) chopstick for an additional time.
- Conspirators retry faster when the victim is hungry, whereas the victim retries slower (simulating the case that f.e. the conspirators block him from retrying fast)

## 2.6 Evaluation Methodology

In order to draw reliable conclusions, we do not rely on a single simulation run. For this reason, all simulations are done using a Monte Carlo Simulation. Depending on the scenario and parameter setting, each configuration is repeated between 50 and 100 times.

The evaluation is primarily based on summary statistics across runs, including the mean and standard deviation of measured performance and fairness metrics. For instance, the total number of eating events, the average number of eats per philosopher, as well as dispersion measures such as the minimum and maximum number of eats across philosophers are reported. Listing 2 shows an example report generated from a Monte Carlo simulation (using same initial and further seeds). The focus of this Listing is on the structure of the reported metrics rather than on the specific numerical values.

```
Runs: 100
Avg total eats: 47.96 +/- 6.62
Avg eats/philosopher: 9.59 +/- 1.32
Avg min/max eats: 7.51 / 11.79
Avg Gini (fairness): 0.088
Deadlock runs: 17/100
Avg first deadlock time: 76.99
Per-philosopher avg eats: [9.34, 9.54, 9.61, 9.75, 9.72]
Per-philosopher std eats: [2.11, 2.35, 2.04, 2.14, 1.91]
Per-philosopher avg thinking time: [49.74, 48.37, 48.97, 50.78, 49.28]
Per-philosopher avg hungry time:   [43.5, 43.47, 41.51, 41.06, 43.68]
Per-philosopher avg eating time:   [26.76, 28.16, 29.53, 28.16, 27.04]
Per-philosopher std thinking time: [12.37, 11.51, 12.69, 11.14, 12.09]
Per-philosopher std hungry time:   [11.61, 12.65, 12.81, 12.0, 11.4]
Per-philosopher std eating time:   [7.54, 8.03, 8.26, 8.34, 7.52]
Zero-eat rate by philosopher: [0.0, 0.0, 0.0, 0.0, 0.0]
```

Listing 2: Example Report Output

To monitor simulation progress and to enable debugging before running large Monte Carlo batches, we use logging during individual runs. Listing 3 illustrates a short example from the event log of an individual simulation run.

```
[t=0.27] Philosopher 0 is now HUNGRY
[t=0.38] Philosopher 0 is now EATING (count: 1)
[t=0.46] Philosopher 1 is now HUNGRY
[t=0.57] Philosopher 3 is now HUNGRY
[t=1.01] Philosopher 3 is now EATING (count: 1)
```

Listing 3: Snippet of an Example Simulation Logging

## 3 Simulation Results

Talking about results (in a storytelling way, so beginning with basic model, then extending with deadlock detection and so on, as in the notebook) + of course showing some plots

We document the results of the simulation as far as possible without potentially subjective interpretation. This is usually done with the help of plots.

### 3.1 Base Model

In the single baseline run, only 4 eating events occurred in 120 time units and several philosophers ate zero times. This shows that single runs can be highly sensitive to randomness and are not reliable on their own. For that reason, we focus on Monte Carlo results. (Note: 0 deadlocks because they are not detected yet)

| Metric | Value |
|---|---|
| Number of runs | 100 |
| Average total eats | $45.12 \pm 9.68$ |
| Average eats per philosopher | $9.02 \pm 1.94$ |
| Average min / max eats | 7.01 / 11.16 |
| Average Gini (fairness) | 0.092 |
| Deadlock runs | 0 / 100 |
| Zero-eat rate | 0.0 |

Table 1: Baseline strategy (no prevention): Monte Carlo summary.

### 3.2 Deadlock Detection / Prevention

In the detection run with `halt_on_deadlock=True`, the simulation stops at the first deadlock (at t=19.26). This shows that the detection logic works and that a deadlock can occur even without prevention.

Monte Carlo results with randomized starts show deadlocks in 11 of 100 runs. This is expected because deadlock is not guaranteed under random timing; it depends on whether all philosophers reach the circular-wait condition within the simulation horizon. The average performance metrics are similar to the baseline because detection only stops runs that actually reach deadlock.

| Metric | Value |
|---|---|
| Number of runs | 100 |
| Average total eats | $45.12 \pm 9.68$ |
| Average eats per philosopher | $9.02 \pm 1.94$ |
| Average min / max eats | 7.01 / 11.16 |
| Average Gini (fairness) | 0.092 |
| Deadlock runs | 11 / 100 |
| Average first deadlock time | 71.91 |
| Zero-eat rate | 0.0 |

Table 2: Deadlock detection strategy: Monte Carlo summary.

The single-run results already show the main differences between the strategies. The left-first approach is prone to deadlock. In the randomized run, several deadlocks occurred during the simulation. With synchronized start, the system deadlocks immediately at time 0, which is expected because all philosophers try to pick up the same chopstick at the same time. This confirms that the baseline strategy is unsafe under unfavorable conditions.

The lowest-ID-first (priority) strategy avoids deadlocks in both randomized and synchronized starts. All philosophers are able to eat, but the eat counts are not fully balanced. Some philosophers eat more often than others, which indicates a small fairness bias caused by the fixed priority rule. However, no starvation occurs and the system always makes progress.

The timeout strategy also avoids deadlock in the single run and results in relatively balanced eat counts. Compared to the priority strategy, the total number of eats is slightly higher. Timeouts allow philosophers to release chopsticks when they wait too long, which helps the system recover from blocking situations.

The Monte Carlo results over 100 runs confirm these observations. The left-first strategy has the lowest average throughput and still produces deadlocks in 11% of the runs. Even though the fairness measure is relatively good when the system runs, deadlocks make this strategy unreliable. The priority strategy prevents deadlocks in all runs, but shows a small fairness imbalance due to priority ordering. The timeout strategy achieves the highest average throughput and good fairness. A small number of deadlocks (5%) still occur, but they are short and do not stop the simulation permanently.

Table 3 summarizes the Monte Carlo results and highlights the trade-off between deadlock safety, throughput, and fairness for the different strategies.

Table 3: Monte Carlo results (100 runs) for Task 3 strategies

| Strategy | Avg Total Eats | Avg Eats / Philosopher | Gini | Deadlock Runs |
|---|---|---|---|---|
| Left-First (Random) | $47.35 \pm 6.37$ | $9.47 \pm 1.27$ | 0.087 | 11 / 100 |
| Left-First (Sync) | 0.00 | 0.00 | 0.000 | 100 / 100 |
| Priority (Random) | $51.23 \pm 6.52$ | $10.25 \pm 1.30$ | 0.096 | 0 / 100 |
| Priority (Sync) | $51.39 \pm 6.33$ | $10.28 \pm 1.27$ | 0.104 | 0 / 100 |
| Timeout | $53.33 \pm 5.70$ | $10.67 \pm 1.14$ | 0.093 | 5 / 100 |

In addition to throughput and deadlock behavior, fairness was evaluated using the Gini coefficient, where lower values indicate a more even distribution of eat counts among philosophers. All strategies except left-first with synchronized start show relatively low Gini values, indicating good fairness overall. The priority strategy exhibits slightly higher Gini values compared to the other approaches, which reflects the bias introduced by fixed priority ordering. The timeout strategy achieves good fairness while maintaining high throughput, making it a balanced solution in practice.

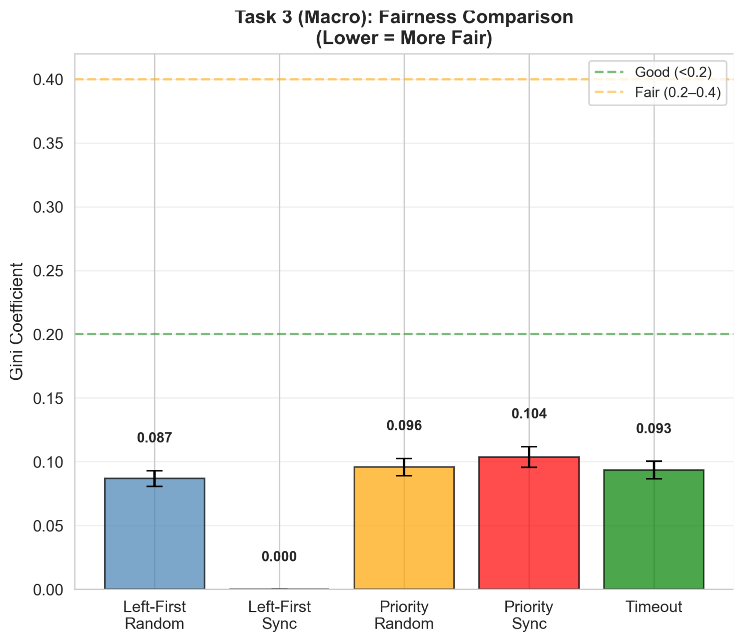

Figure 2: Fairness comparison using the Gini coefficient (lower values indicate more fairness)

Figure 3 shows the deadlock frequency across strategies, clearly illustrating the difference between deadlock-prone and deadlock-preventing approaches.

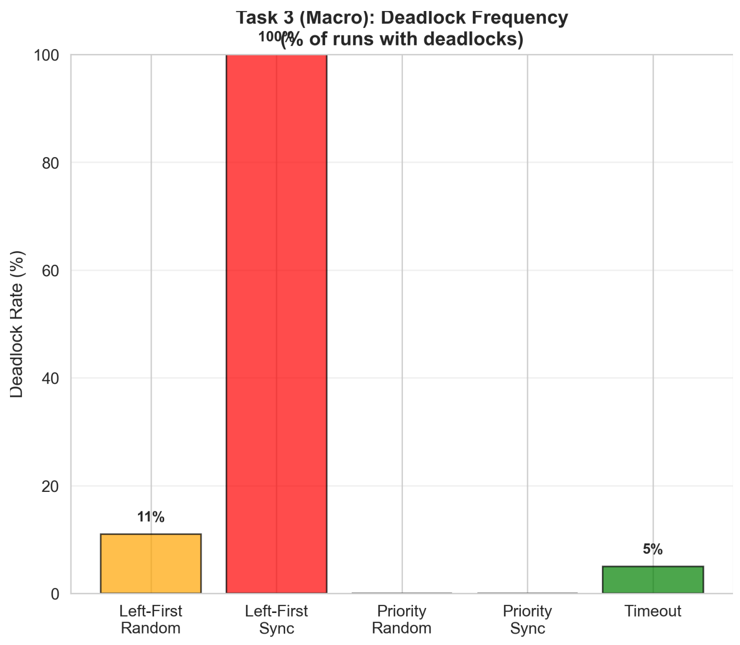

Figure 3: Deadlock frequency over 100 Monte Carlo runs for different strategies

Figure 4 compares the total system throughput. Strategies that avoid deadlock also achieve higher overall performance.

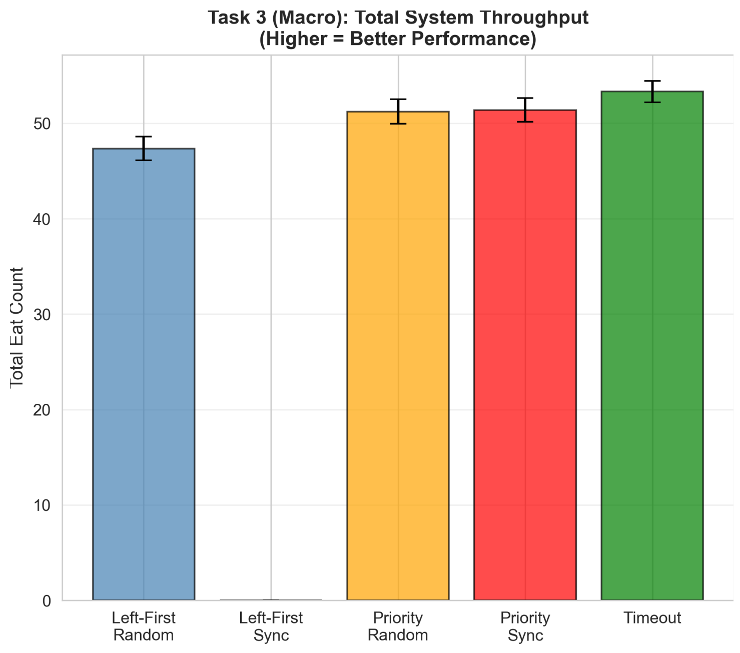

Figure 4: Average total system throughput for different strategies

Overall, the priority strategy is the safest choice if deadlock freedom is required, while the timeout strategy provides the best practical performance by balancing throughput and fairness. The left-first strategy is simple and fair in theory, but unsafe in practice due to its deadlock behavior.

### 3.3 Starvation Avoidance

In the following, the impact of the starvation avoidance mechanism is investigated on the simulation runs using Monte Carlo simulation. A model using priority ordering is used as the baseline for comparing the system with and without starvation avoidance. Both of the model configurations are executed over 50 independent runs with identical parameters and the performance metrics are collected for every philosopher in each turn.

The following boxplots summarize the resulting distributions of four key metrics, namely eating frequency, total hungry time, maximum hungry time per run, and fairness measured as the standard deviation of eating counts.

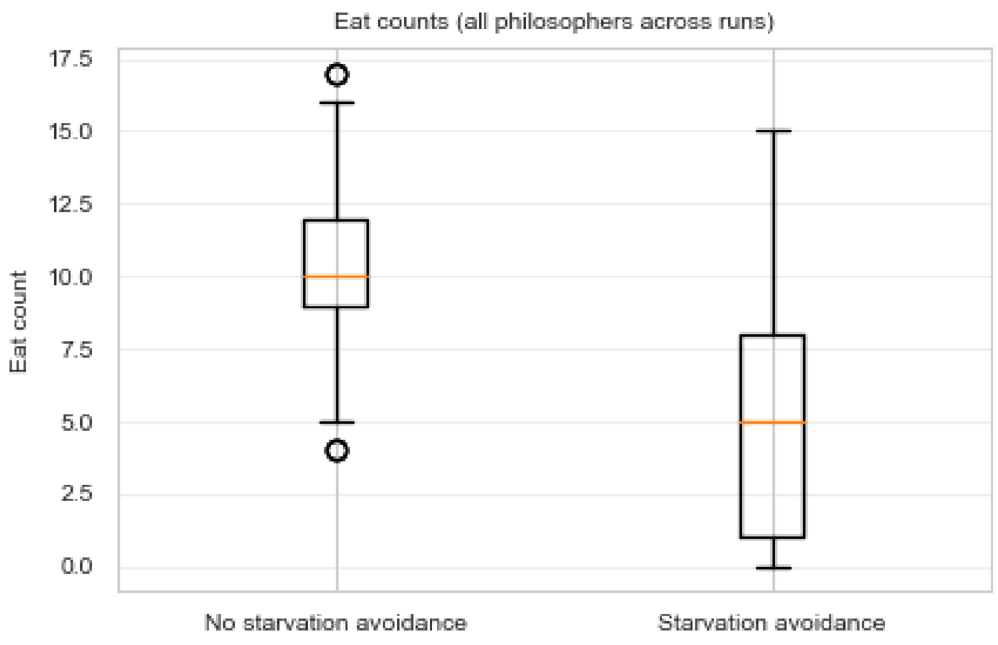

Figure 5: Eat counts across all philosophers and runs

The boxplot illustrated in Figure 5 shows the eat-count distribution, which reflects system throughput. With starvation avoidance enabled, the median eat frequency decreases and the distribution becomes more scattered. This is caused by the priority-based mechanism, which allows only the most-starved philosopher to attempt chopstick acquisition at a time, reducing parallelism. As a result, throughput is traded for improved control over starvation.

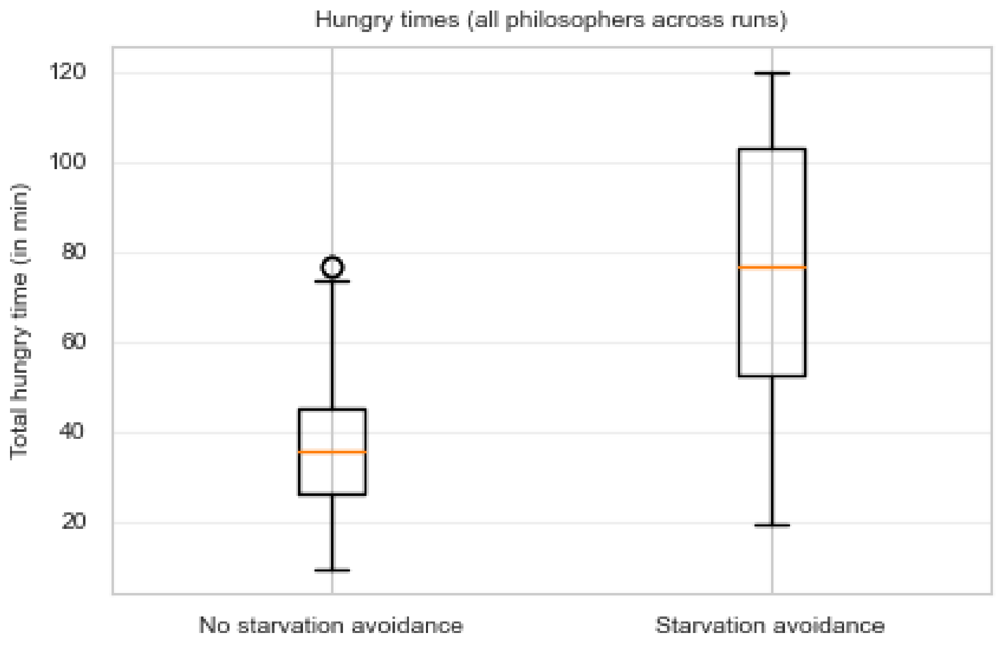

Figure 6: Eat counts across all philosophers and runs

Running the model with no starvation avoidance delivers a median hungry time of about 35 minutes, while the system with starvation avoidance enabled has a median of about 80 minutes with a large but structured spread (see Figure 6). This reflects longer waiting periods introduced by prioritizing the most starved philosopher and temporarily blocking others from progressing. One particular philosopher is allowed to eat while the others have to wait. Even though they have to wait longer to eat, the other philosophers don't have to wait a indefinite amount of time to get the chopsticks. This reflects bounded waiting instead of starvation.

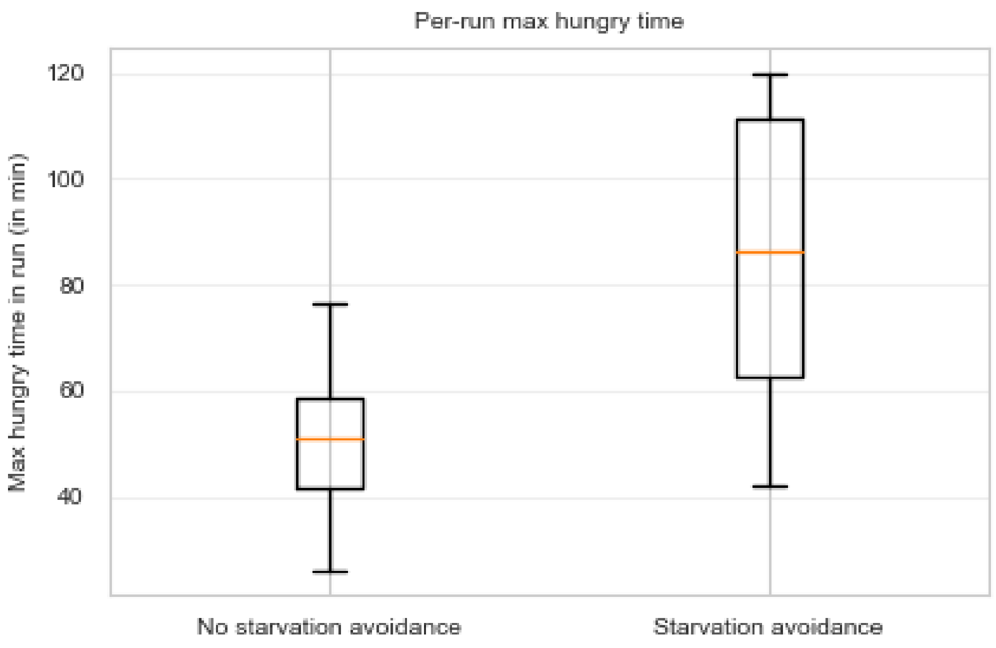

Figure 7: Eat counts across all philosophers and runs

Figure 7 shows the distribution of the tracked maximum hungry times across all simulation runs for both model settings. Without starvation avoidance enabled, the system's maximum hungry time depends on whether starvation occurs in a given run. In contrast, when starvation avoidance is enabled, the maximum hungry time is determined by the number of philosophers, the eating and thinking times, and the priority queue order. This can be observed in the right boxplot of the figure, where the spread of maximum hungry times is larger, spanning approximately from 60 to 115, while the baseline case ranges from about 40 to 60.

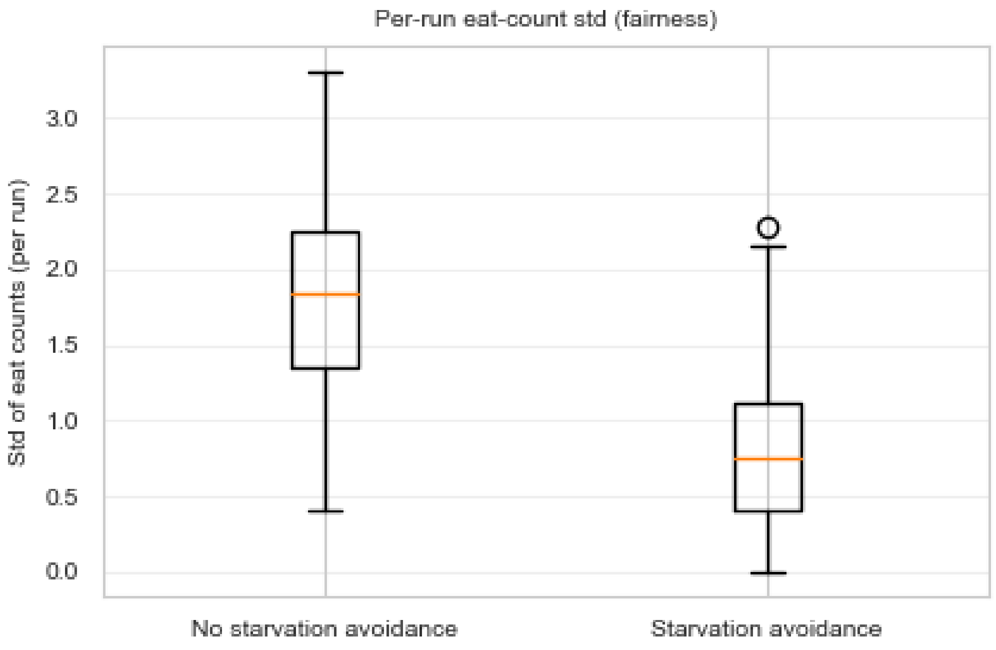

Figure 8: Eat counts across all philosophers and runs

Figure 8 illustrates the distribution of the per-run standard deviation of eating counts across all simulation runs for both model settings. In the absence of starvation avoidance, the standard deviation varies widely between runs, reflecting differences in how often individual philosophers are able to eat. With starvation avoidance enabled, the variation in eating counts is consistently lower, as the priority-based selection limits disproportionate access to chopsticks. This effect is visible in the right boxplot, where the values are concentrated in a narrower range, indicating improved fairness among philosophers.

## 3.4 Scneario Variations

After analyzing deadlock and starvation behavior under the standard configuration, we now turn to additional scenario variations. In this part, each simulation is run for 800 time units and execute 100 MC runs. Unless stated otherwise, we use the base model with deadlock prevention enabled, but without the starvation avoidance mechanism.

### 3.4.1 Varying Number of Philosophers

First, we increase the number of philosophers (thus automatically also number of chopsticks) to 10, and compare simulation outcomes for the left-first (LF), priority ordering (PO) and timeout (TO) strategies. For the TO strategy, the timeout duration is set to 2 time units, if not stated otherwise. We also count the number of "persistent deadlocks", which means that the deadlock state does not change for a certain amount of time, such that we can assume, that we really are in a deadlock. This is important, since for example in the Timeout Strategy, it can be the case that the deadline situation appears just for a short while and is solved then by the timeout strategy itself.

Figure 9 compares three deadlock prevention strategies for ten philosophers by showing the average number of eating events per philosopher as bars. The standard deviation across the 10

Monte Carlo runs is also indicated by the black thin error bars, Above each bar, the annotations report the deadlock rate (percentage of runs that ended in a deadlock) and the Gini coefficient as a fairness measure (lower = more equal access to eating). So, with 10 philosophers, the differences between the strategies become very clear. We see that in terms of eats per philosopher, the LF strategy performs the worst, also containing a certain amount of deadlock cases. The PO strategy (See Priority bar in Figure) performs better, but it is less fair, since the Gini value is higher, Timeout achieves the best overall result in this setup: no deadlocks, the highest throughput (71.1), and the lowest Gini (0.038), so the eating opportunities are distributed more evenly while keeping the system stable.

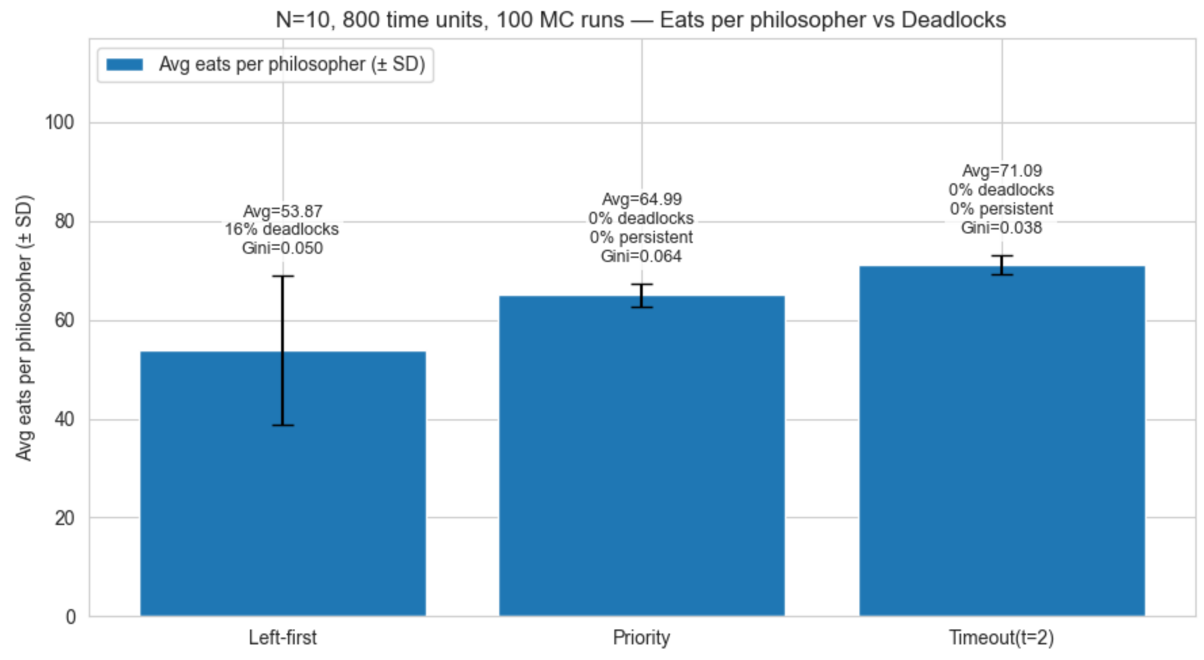

Figure 9: Comparison Simulation Results for 10 Philosophers

To see how the results change with a lower number of philosopher, we also set the number of philosophers to three and re-run with the same configurations. The results are shown in Figure 10 again shows the same statistics for three philosophers. Here, the throughput is very similar for all strategies and fairness is also comparable. However, the whole model is highly unstable now, if we look at the deadlock rates. Although the persistent deadlock rate is 0% for the timeout strategy, we still see that the occurring deadlock state rate is very high. However, we see that this is fully prevented in the priority ordering case. This indicates that for three philosophers, priority is clearly the most robust choice, because the other strategies frequently reach a deadlock even though their average performance looks similar.

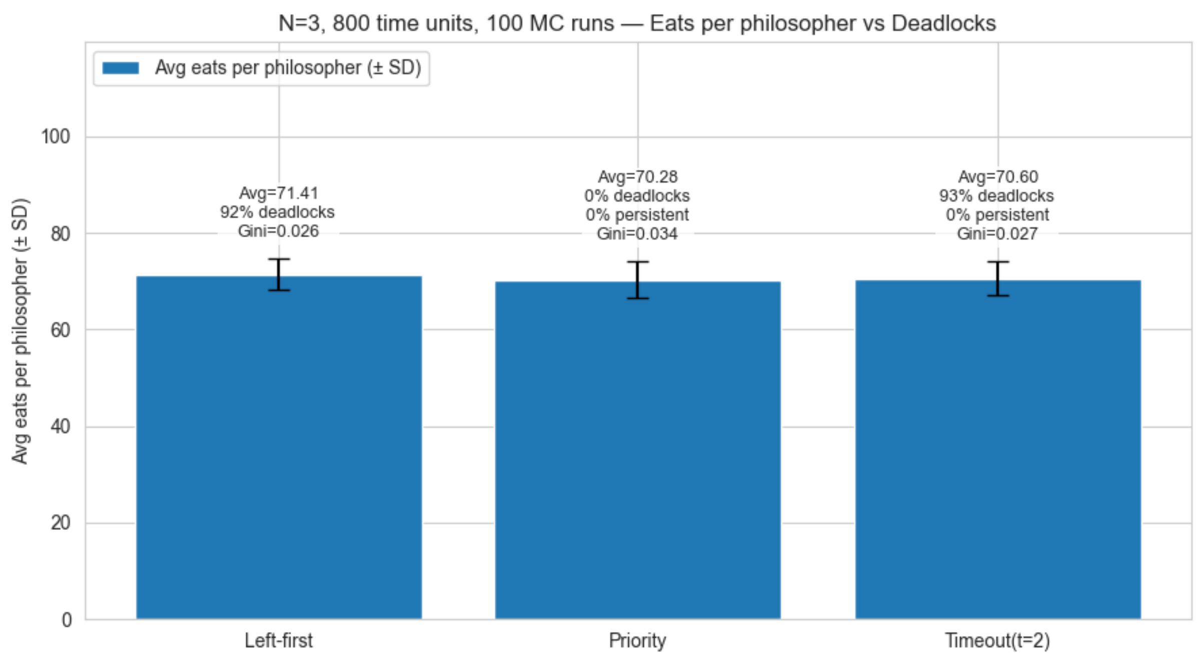

Figure 10: Comparison Simulation Results for 3 Philosophers

For the next part, we will inspect results with number of philosophers set to five, as in the standard model.

### 3.4.2 Varying Thinking Time

Now, we want to vary the thinking time. For this reason, we want to look at two different uniform distributions: $unif(2, 30)$ and $unif(2, 4)$. Figure 11 shows a stacked bar chart, which illustrates, how each philosopher's simulation time is distributed across the three states thinking, hungry (= waiting), and eating, for two different thinking-time distributions. We see that when the thinking time is wide ($unif(2, 30)$), philosophers spend most of the time thinking, and only a small number is waiting in the hungry state. As a result, contention for chopsticks is low, deadlocks do not occur (0%), but the throughput is lower. With short thinking times ($unif(2, 30)$), philosophers become hungry much more frequently. The plot shows that the reduced thinking time has largely moved into hungry time, meaning philosophers spend much more time waiting for chopsticks because of higher contention. However, the throughput still rises because philosophers cycle through the think–eat loop more often. This higher contention also slightly increases the deadlock probability to 2%.

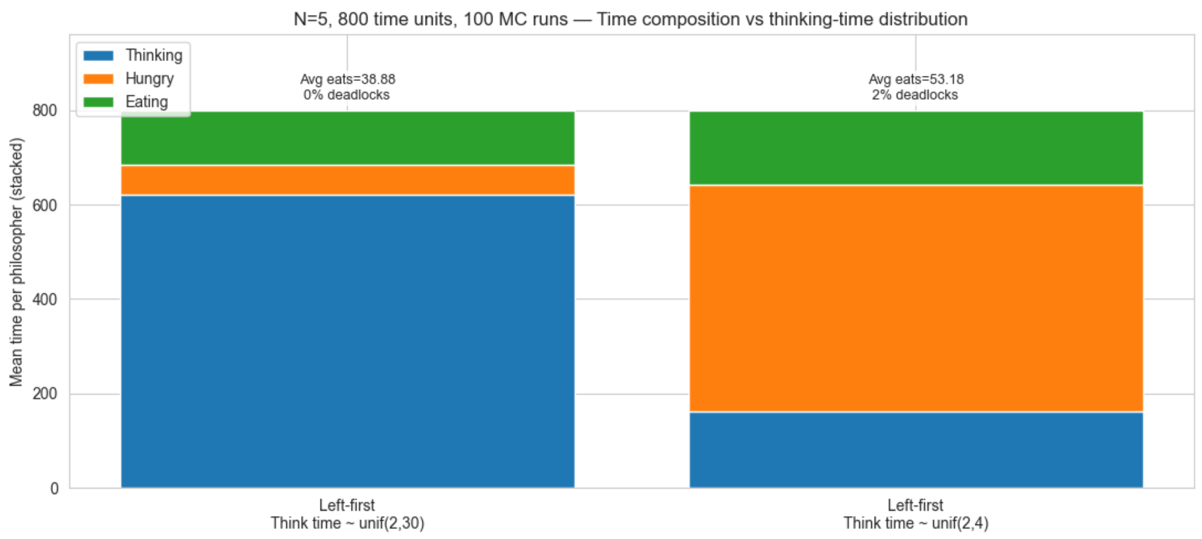

Figure 11: Time Composition vs Thinking Time Distribution for Left-first strategy

Since we want to get cases, where the deadlock possibility is high enough, we will set the distribution for the thinking times to $unif(0.3, 3)$ for the remaining part of this section.

### 3.4.3 Varying Eating Time

As already mentioned, we will vary between two different configurations of a truncated normal distribution:

- $norm(8, 3)$ with truncated values resulting in range [1,16]
- $norm(15, 3)$ with truncated values resulting in range [1,30]

With these configurations, we aim to see how the duration of the eating time with thinking times sampled from $unif(0.3, 3)$ will impact the results.

Figure 12 contains a grouped bar chart that compares how two different eating-time distributions affect performance for five philosopher under the three strategies. Across all strategies, increasing the eating time from $norm(8, 3)$ to $norm(15, 3)$ hugely reduces the average eat numbers. That makes sense, since we can see philosophers simply spend longer holding chopsticks, so fewer eating cycles fit into the fixed simulation horizon. Although PO achieves the highest throughput, it is consistently the least fair strategy. LF stays the most problematic in terms of stability, with high deadlock rates, even though its Gini value is low. All in all, we see that with the higher eating times, the number of eats drops massively, which makes sense.

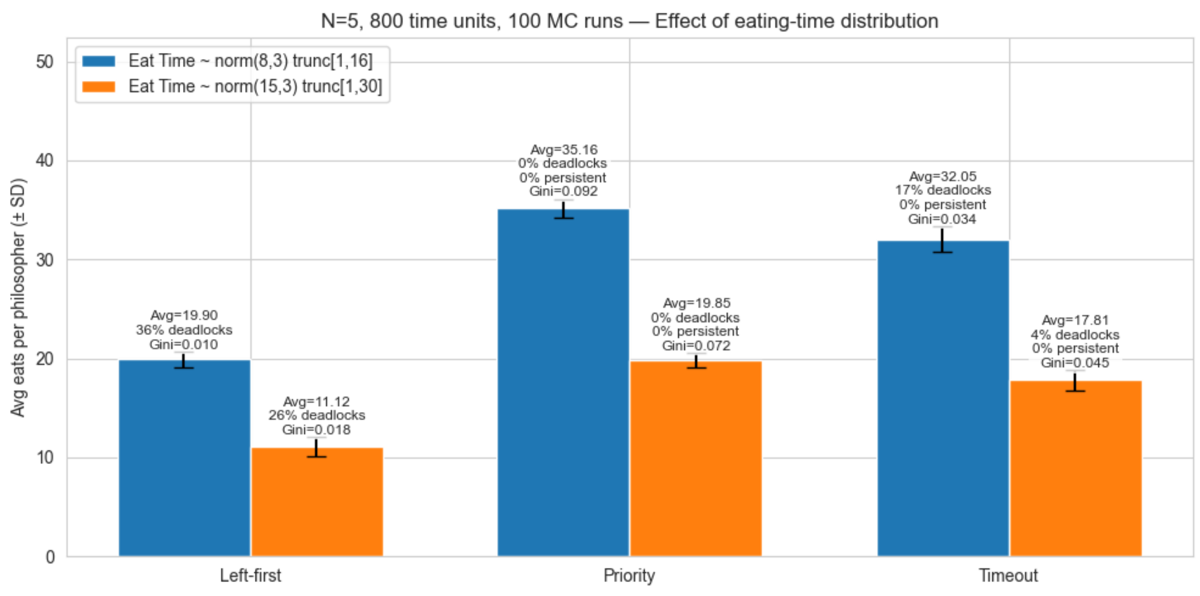

Figure 12: Effect of Eating Time Distribution

### 3.4.4 Selfish Philosophers

Now, we will continue with the following default settings: Five philosophers, simulation run time set to 800, 100 executed MC runs, thinking time sampled from $unif(0.3, 3)$ and eating time sampled from $norm(8, 3)$ (truncated: [1,16]).

First, we will use the model with the TO strategy and introduce a selfish philosopher, who will have the following characteristics:

- Ignoring the Timeout strategy
- No immediately dropping of chopsticks after finishing eating (extra holding time before release = 1 second)

Figure 13 compares the per-philosopher eating counts under the timeout strategy (timeout duration = 8) for a normal baseline run versus a run with one selfish philosopher (ID = 3). For the baseline case, we see that all people achieve nearly the same number of eats, so the outcome is fairly balanced. However, when philosopher 3 behaves selfishly, the distribution becomes strongly skewed: philosopher 3 increases its average eats substantially, while at least one other philosopher (ID = 2) drops notably. This makes sense, since the selfish philosopher has to impact at least one of his neighbors. We can also see that in the shift of the Gini index, it goes higher. Overall, the selfish behavior benefits the selfish philosopher directly, but it reduces fairness, which was somehow expected.

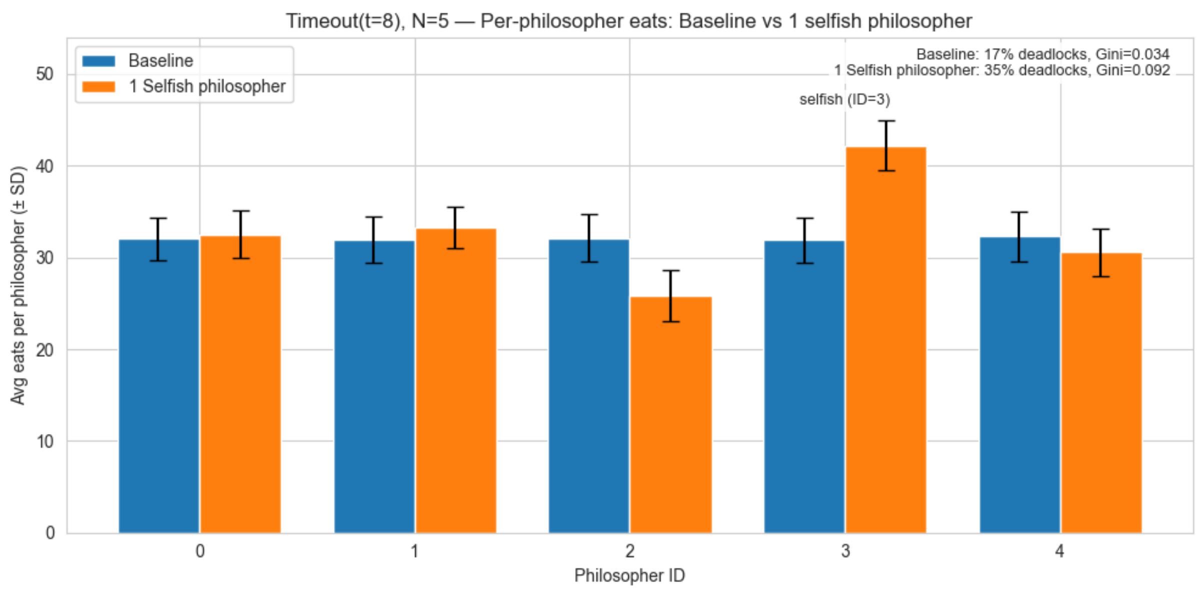

Figure 13: Effect of 1 Selfish Philosopher for Timeout strategy

Now, we will look at one selfish philosopher again, but with the PO strategy. Here, the selfish philosopher has the following characteristics:

- Ignoring the Priority Ordering Strategy and just deciding randomly on which chopstick to take first
- No immediately dropping of chopsticks after finishing eating (extra holding time before release = 1 second)

Figure 14 now shows the same statistics, but for the PO strategy. Besides, the selfish philosopher is philosopher with ID 1 in this case. We decides on that, because with the PO strategy, we already have an unfairness, benefiting the philosopher 3. After introducing the selfish philosopher, philosopher 1 increases its average eats substantially, while other philosophers, especially philosopher 2, lose some eats. We also see that the Gini also got higher. However, deadlocks remain at 0, which again shows that although PO is more unfair, it is very stable.

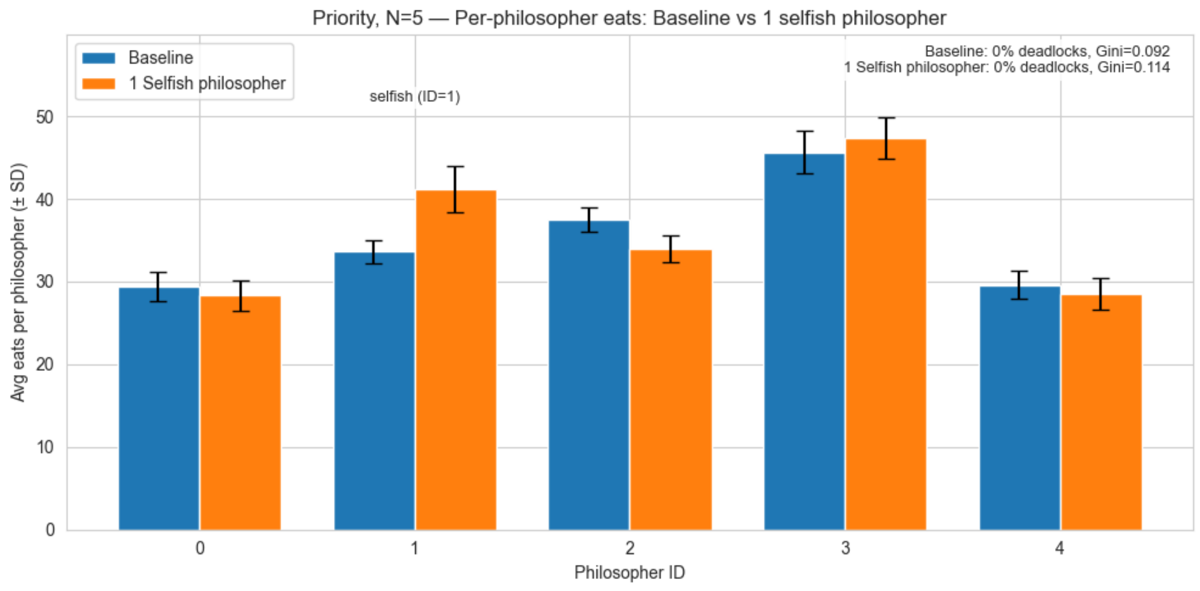

Figure 14: Effect of 1 Selfish Philosopher for Priority Ordering strategy

### 3.4.5 Conspirator Philosophers

Now we introduce conspirator philosophers. Their characteristics are already described in Section 2.5.

Figure 15 compares the baseline (PO strategy) model with a conspiracy scenario where philosophers 1 and 3 collude to block philosopher 2. In the baseline, priority ordering is again already somewhat unequal (Gini is high), but all philosophers still eat regularly. Under conspiracy, the effect becomes extreme. So, the victim philosopher 2 drops from roughly 37 eats to about 12.5, indicating clear starvation. At the same time, one conspirator, especially philosopher 3, benefits heavily, increasing to around 62 eats, while the other philosophers also gain slightly. We also see, this distribution slightly increased the Gini index, which again makes sense. However, we again see that deadlocks remain at 0%, which again speaks for the robustness of this strategy.

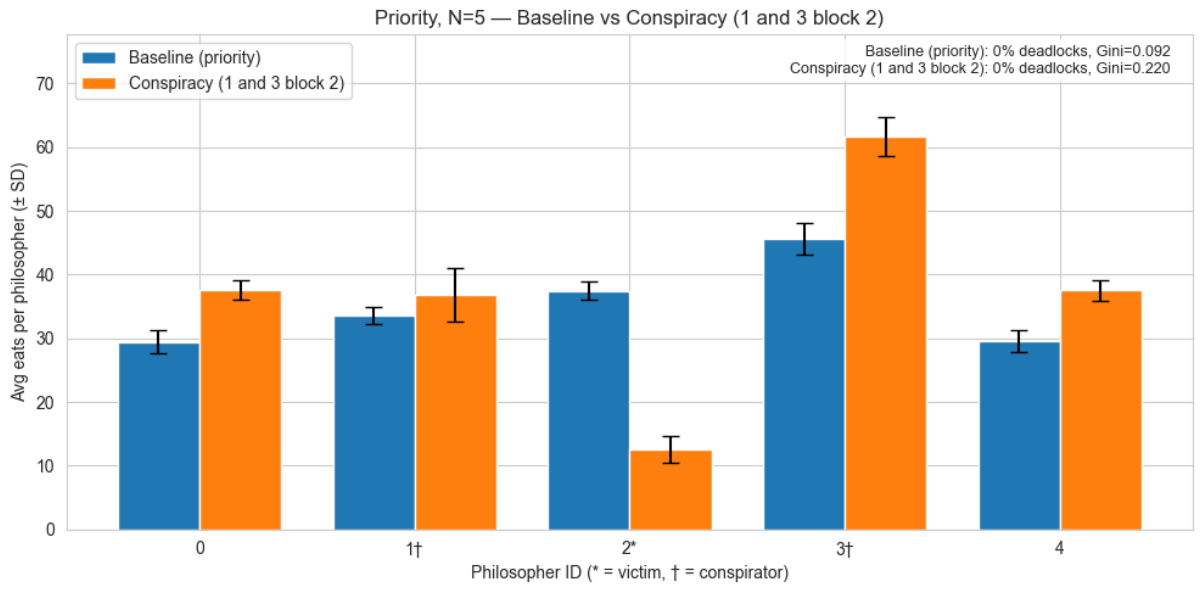

Figure 15: Effect of introducing Conspirators

In summary, the additional scenarios show that the system's results depend heavily on how much contention there is and whether all philosophers follow the intended rules. When the number of philosophers increases, the differences between the strategies become clearer: LF is still prone to deadlocks and shows high variability, while PO and the TO approach are much more stable and usually achieve higher throughput. Changing the timing distributions mainly affects where the simulation time is spent. Short thinking times lead to more simultaneous requests for chopsticks, so hungry time increases and deadlocks can become slightly more likely, whereas longer eating times reduce the overall number of eating events because chopsticks are occupied for longer periods. Finally, the selfish and conspiracy experiments highlight that preventing deadlocks is not enough on its own: even with a deadlock-free strategy like priority ordering, unfair behavior can create strong imbalances and clear starvation effects, which is reflected by the much higher Gini values and the per-philosopher results.

## 4 Discussion

This work investigated the Dining Philosophers problem using a discrete event simulation framework, with the focus set on deadlock occurrence, starvation, and fairness under different prevention strategies. By incrementally extending a simple baseline model with deadlock detection, prevention mechanisms, and starvation avoidance, we were able to systematically analyze the trade-offs between system efficiency, robustness, and fairness.

### Summary of Findings

Our initial model with the left-first strategy showed that while the system can operate smoothly under favorable timing conditions, it still can be suspect to deadlocks and unfair resource allocations. We introduced Monte Carlo experiments , which showed that deadlocks can occur on regular basis and that individual philosophers may experience very low or zero eating counts in some runs.

When we introduced deadlock detection into our model, it confirmed that such blocking states can arise naturally as an effect of even resource acquisition rules. Among the prevention strategies, priority-based was able to successfully eliminate deadlocks by setting a global resource order for the philosophers, while the timeout strategy resolved circular waiting dynamically without relying on strict ordering. Across multiple scenarios, the timeout strategy was able to achieve high throughput consistently while maintaining low deadlock rates.

Additionally the model included starvation avoidance to abolish indefinite waiting times using priority mechanism that effectively bounded waiting times and provided fairness by even eating distributions amonst the philosophers. However, this improvement in fairness came at the cost of reduced parallelism and lower overall model throughput, as only one philosopher was allowed to acquire resources at a time.

Scenario variations further showed how the system behavior depends on its parameter settings. Increasing the number of philosophers showed strong deviations between strategies, while shorter thinking times increased conflict and deadlock likelihood. The experiments with selfish and conspirator philosophers highlighted how uncooperative behavior can affect fairness and system stability negatively.

## Interpretation and Trade-offs

The results show that no single rule is best in all aspects. Symmetric rules like left-first can look fair and keep good throughput, but they can still deadlock. Priority ordering (lowest-ID first) removes deadlock by design, yet it introduces a bias because some chopsticks are targeted more often. The timeout strategy sits in between: it avoids long deadlocks in practice and keeps fairness closer to the baseline, but it adds retries and small overhead.

The starvation avoidance mechanism highlights another common trade-off. By giving priority to the longest-waiting philosopher, fairness improves, but total throughput usually drops because fewer philosophers can compete at the same time. This reflects a typical balance between efficiency and fairness in resource-sharing systems.

Overall, the Dining Philosophers model is alternative way to study coordination problems that appear in real systems such as locking in databases, scheduling in operating systems, and resource allocation in distributed services.

## References