kiana-mohammadinik-urgent-care-simulation

2026-02-09

Instructions

Choose ONE of the three questions below and answer it using simulation.
Your submission must include:

- Well-commented code (I should be able to follow your logic)
- A written analysis that answers the actual questions being asked (not just reporting numbers)
- At least one plot that directly supports a conclusion
- At least one sensitivity analysis (change an assumption and explain what changes)
- A clear recommendation in plain English, supported by your simulation results

You may make additional reasonable assumptions, but you must state and justify them.

Required (for any problem you choose):

Make one recommendation that optimizes something under a constraint.

- Queueing: minimize total cost *or* meet a service-level constraint
- Inventory: maximize profit while accounting for salvage/stockouts (or another realistic feature)
- Reliability: maximize uptime per dollar under a budget constraint

Question 1: Urgent Care Center Queueing Systems

You operate an urgent care center that:

- Opens at 8am and closes at 5pm
- Customer arrivals follow a Poisson process
  - Rate = 4 per hour, except 12–1pm, when rate = 6 per hour
- Service times are i.i.d. random variables on the positive real line *(you must choose and justify a service-time distribution)*

You have 2 servers.

You are considering three queuing systems:

1. Single pooled line with parallel servers
2. Tandem service (In-N-Out style): customers must complete service at Server 1, then Server 2

3. Separate lines, one per server (grocery-store style)

Core Tasks

- Simulate 1 work week (5 days).
- Repeat the week simulation 7 times and average results.
- Estimate and report:
  - Mean waiting time
  - Distribution of waiting times (not just the mean)
- Compare the systems.

Decision Requirement (required)

Assume:

- Waiting cost = $0.50 per customer-minute
- Server cost = $30 per hour per server

Your task is to:

- Choose the "best" system and clearly define what "best" means
- Report the total expected daily cost
- Make a clear operational recommendation

Add ONE realism feature (choose one)

Pick one of the following additions and incorporate it in your model:

A. Staffing tradeoff: allow 2 vs 3 servers during 12–1pm and choose the best option under total cost

Hard

- In System 3 (separate lines), allow customers to switch lines
- Compare outcomes with and without switching

Note: If you want a concrete service-time option, you may use either:

- Lognormal service times (e.g., mean 10 min, sd 8 min), or
- A mixture model (e.g., 80% "quick" mean 6 min, 20% "long" mean 20 min)
  Extra Credit (+10%, choose ONE)
  - Event-driven simulation: implement an event-based simulator (priority queue) rather than time-stepping

  Grading Emphasis You are graded on:

- Correct simulation logic
- Clear explanation and interpretation
- Thoughtful modeling assumptions
- Decision quality and justification
- Code readability and documentation

This is not about closed-form answers. It's about modeling, simulation, and decision-making under uncertainty.

Event list helpers + arrivals + service distribution

```r
# 1) GLOBAL TIME CONSTANTS

# minutes in workday (8am-5pm)
T_END   <- 540
# 12:00
T_LUNCH_START <- 240
# 1:00
T_LUNCH_END   <- 300

rate_per_min <- function(t){
  # piecewise arrival rate (per minute)
  if (t < T_LUNCH_START) return(4/60)
  if (t < T_LUNCH_END)   return(6/60)
  return(4/60)
}


# 2) ARRIVAL GENERATION (EVENT-DRIVEN)

# For piecewise-constant rates: generate next arrival time from current t
next_arrival_time <- function(t_now){
  t <- t_now

  while(TRUE){
    lam <- rate_per_min(t)
    if (lam <= 0) return(Inf)

    # boundary of current segment
    t_bound <- if (t < T_LUNCH_START) T_LUNCH_START else if (t < T_LUNCH_END) T_LUNCH_END else T_END

    # sample exponential interarrival with current lam
    dt <- rexp(1, rate = lam)
    t_candidate <- t + dt

    if (t_candidate < t_bound) {
      return(t_candidate)
    } else {
      # if we hit the boundary before the arrival, move to boundary and try again
      t <- t_bound
```

```
      if (t >= T_END) return(Inf)
    }
  }
}

generate_arrivals_one_day <- function(){
  t <- 0
  arr <- c()
  while(TRUE){
    t <- next_arrival_time(t)
    if (!is.finite(t) || t >= T_END) break
    arr <- c(arr, t)
  }
  arr
}


# 3) SERVICE TIME DISTRIBUTION CHOICE

# Lognormal option: mean ~ 10, sd ~ 8 minutes
# Converting (mean, sd) to lognormal parameters:
lognormal_params <- function(mean, sd){
  var <- sd^2
  sigma2 <- log(1 + var/mean^2)
  sigma  <- sqrt(sigma2)
  mu     <- log(mean) - 0.5*sigma2
  list(mu = mu, sigma = sigma)
}

svc_params <- lognormal_params(mean = 10, sd = 8)

sample_service_time <- function(n = 1){
  # returns positive service times in minutes
  rlnorm(n, meanlog = svc_params$mu, sdlog = svc_params$sigma)
}

# 4) EVENT LIST (PRIORITY QUEUE)

# Implement as a data.frame and always pop the smallest time.
add_event <- function(event_list, time, type, server = NA_integer_, cust_id = NA_integer_){
  new <- data.frame(time = time, type = type, server = server, cust_id = cust_id)
  rbind(event_list, new)
}

pop_next_event <- function(event_list){
  idx <- which.min(event_list$time)
  ev  <- event_list[idx, , drop = FALSE]
  event_list <- event_list[-idx, , drop = FALSE]
  list(event = ev, event_list = event_list)
}
```

This arrival method (used above) works because since arrivals are Poisson with constant rate within each segment, interarrivals are exponential within the segment. If a sampled interarrival would cross into the next

segment, we "jump to the boundary" and sample again using the new rate. That gives an exact NHPP for this piecewise-constant case.

Also, sorting the event list works because his simulation will have on the order of:

- 4/hr × 9 hr = 36 customers/day (a bit more because lunch)
- plus service completion events

So the event list stays small. Sorting is fast and clean.

System 1 (Pooled line, 2 servers) : Event-driven simulator in R

```
simulate_day_system1 <- function(n_servers = 2, seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  # Customer storage
  # create customers on the fly as arrivals happen
  customers <- data.frame(
    cust_id = integer(),
    arrival_time = numeric(),
    service_start = numeric(),
    service_end = numeric(),
    wait_time = numeric()
  )

  # System state
  # pooled FIFO queue of customer IDs
  queue <- integer()
  # if Inf => idle
  busy_until <- rep(Inf, n_servers)
  server_job <- rep(NA_integer_, n_servers)

  # find an idle server index (or NA if none)
  idle_server <- function(){
    idx <- which(busy_until == Inf)[1]
    if (length(idx) == 0) return(NA_integer_)
    idx
  }

  # start service for next customer in queue if server is idle
  start_service_if_possible <- function(t_now, event_list){
    s <- idle_server()
    while (!is.na(s) && length(queue) > 0) {
      cid <- queue[1]
      queue <<- queue[-1]

      st <- t_now
      svc <- sample_service_time(1)
      et <- t_now + svc

      # update customer record
      customers$service_start[customers$cust_id == cid] <<- st
```

```
    customers$service_end[customers$cust_id == cid]   <<- et
    customers$wait_time[customers$cust_id == cid]     <<- st - customers$arrival_time[customers$cust_

    # update server state
    busy_until[s] <<- et
    server_job[s] <<- cid

    # schedule departure event for this server
    event_list <- add_event(event_list, time = et, type = "depart", server = s, cust_id = cid)

    # check for another idle server
    s <- idle_server()
  }
  event_list
}

# Event list initialization
event_list <- data.frame(time = numeric(), type = character(), server = integer(), cust_id = integer()

# schedule first arrival
t_first <- next_arrival_time(0)
if (is.finite(t_first) && t_first < T_END) {
  event_list <- add_event(event_list, time = t_first, type = "arrival")
}

# customer id counter
next_id <- 1

# Main event loop
while (nrow(event_list) > 0) {
  popped <- pop_next_event(event_list)
  ev <- popped$event
  event_list <- popped$event_list

  t_now <- ev$time
  if (!is.finite(t_now)) break

  # If it is after closing, we stop new arrivals but let service finish. But, since we only schedule

  if (ev$type == "arrival") {
    # create new customer
    cid <- next_id
    next_id <- next_id + 1

    customers <- rbind(
      customers,
      data.frame(
        cust_id = cid,
        arrival_time = t_now,
        service_start = NA_real_,
        service_end = NA_real_,
        wait_time = NA_real_
      )
```

```
      )

      # join the pooled queue
      queue <- c(queue, cid)

      # schedule next arrival
      t_next <- next_arrival_time(t_now)
      if (is.finite(t_next) && t_next < T_END) {
        event_list <- add_event(event_list, time = t_next, type = "arrival")
      }

      # start service if any server idle
      event_list <- start_service_if_possible(t_now, event_list)

    } else if (ev$type == "depart") {
      s <- ev$server

      # free server
      busy_until[s] <- Inf
      server_job[s] <- NA_integer_

      # immediately start next customer if waiting
      event_list <- start_service_if_possible(t_now, event_list)
    }
  }

  # Return only customers who actually started service (removes weird edge cases)
  customers
}

# Quick summary helper
summarize_waits <- function(customers){
  waits <- customers$wait_time
  waits <- waits[is.finite(waits)]
  list(
    n_customers = length(waits),
    mean_wait = mean(waits),
    median_wait = median(waits),
    p90_wait = unname(quantile(waits, 0.90)),
    p95_wait = unname(quantile(waits, 0.95))
  )
}
```

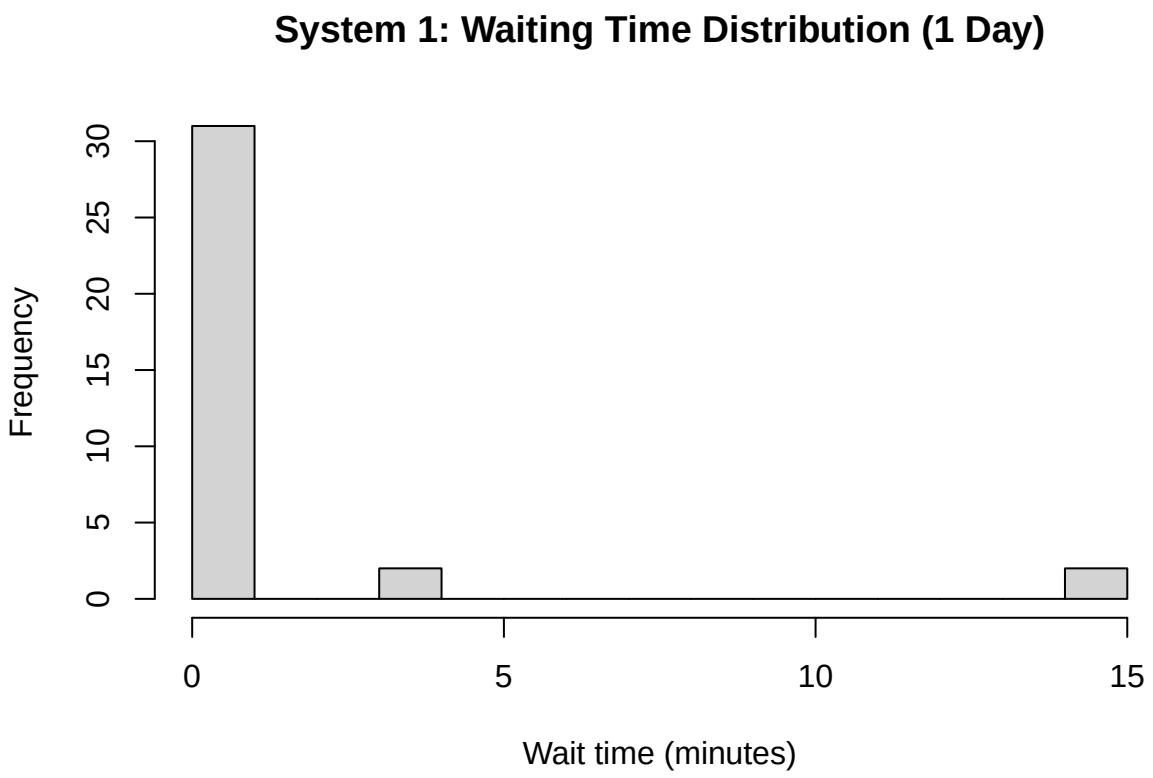
Test (1 day)

```
set.seed(1)
cust1 <- simulate_day_system1(n_servers = 2)
summarize_waits(cust1)
```

```
## $n_customers
## [1] 35
##
## $mean_wait
```

```
## [1] 1.05209
##
## $median_wait
## [1] 0
##
## $p90_wait
## [1] 2.249934
##
## $p95_wait
## [1] 6.892557
```

```r
# Basic plot: waiting time distribution
hist(cust1$wait_time, breaks = 20,
     main = "System 1: Waiting Time Distribution (1 Day)",
     xlab = "Wait time (minutes)")
```



Simulating 1 work week (5 days), repeat 7 weeks, average results, and compute daily costs for System 1

```r
simulate_week_system1 <- function(n_servers = 2, seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  # simulate 5 independent days (fresh randomness each day)
  days <- vector("list", 5)
  for (d in 1:5){
    days[[d]] <- simulate_day_system1(n_servers = n_servers)
```

```
    days[[d]]$day <- d
  }
  do.call(rbind, days)
}

# Run 7 independent weeks and return a list:
# - all customer-level data
# - week-level summaries
# - overall averages
run_system1_experiment <- function(n_weeks = 7, n_servers = 2, seed = 123){
  set.seed(seed)

  all_weeks <- vector("list", n_weeks)
  week_summaries <- data.frame(
    week = integer(),
    mean_wait = numeric(),
    median_wait = numeric(),
    p90_wait = numeric(),
    avg_customers_per_day = numeric(),
    total_wait_minutes_per_day = numeric()
  )

  for (w in 1:n_weeks){
    cust <- simulate_week_system1(n_servers = n_servers)
    cust$week <- w
    all_weeks[[w]] <- cust

    # daily totals for that week
    daily <- aggregate(wait_time ~ day, data = cust, FUN = function(x) sum(x, na.rm = TRUE))
    mean_total_wait_per_day <- mean(daily$wait_time)

    waits <- cust$wait_time
    waits <- waits[is.finite(waits)]

    week_summaries <- rbind(
      week_summaries,
      data.frame(
        week = w,
        mean_wait = mean(waits),
        median_wait = median(waits),
        p90_wait = unname(quantile(waits, 0.90)),
        avg_customers_per_day = nrow(cust)/5,
        total_wait_minutes_per_day = mean_total_wait_per_day
      )
    )
  }

  all_customer_data <- do.call(rbind, all_weeks)

  # Overall averages across the 7 week summaries
  overall <- data.frame(
    mean_wait = mean(week_summaries$mean_wait),
    median_wait = mean(week_summaries$median_wait),
```

```
    p90_wait = mean(week_summaries$p90_wait),
    avg_customers_per_day = mean(week_summaries$avg_customers_per_day),
    total_wait_minutes_per_day = mean(week_summaries$total_wait_minutes_per_day)
  )

  list(
    customers = all_customer_data,
    week_summaries = week_summaries,
    overall = overall
  )
}
```

1) Week + multi-week wrapper for System 1

```
compute_daily_cost <- function(total_wait_minutes_per_day, n_servers, hours_open = 9,
                               wait_cost_per_min = 0.50, server_cost_per_hr = 30){

  waiting_cost <- wait_cost_per_min * total_wait_minutes_per_day
  staffing_cost <- server_cost_per_hr * hours_open * n_servers
  total_cost <- waiting_cost + staffing_cost

  data.frame(
    waiting_cost = waiting_cost,
    staffing_cost = staffing_cost,
    total_cost = total_cost
  )
}
```

2) Cost calculation (daily)

```
res1 <- run_system1_experiment(n_weeks = 7, n_servers = 2, seed = 1)

res1$overall
```

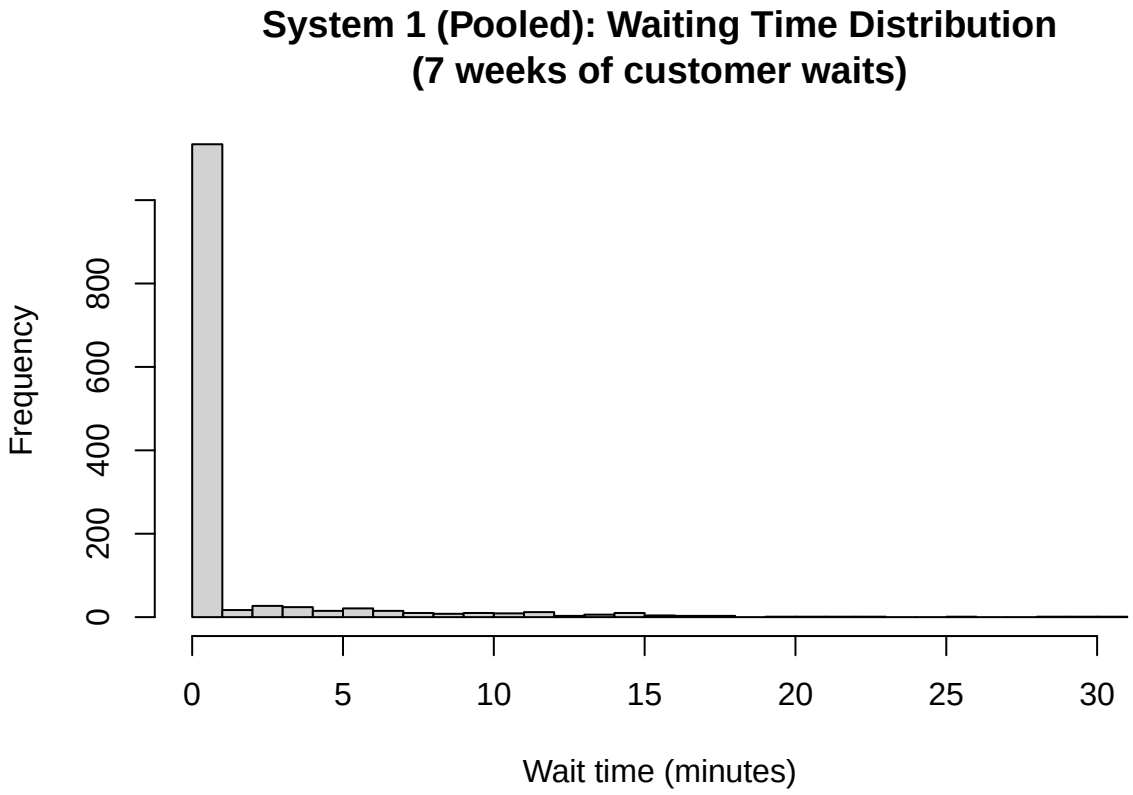
3) Run System 1 (2 servers) + plot

```
##   mean_wait median_wait p90_wait avg_customers_per_day
## 1   1.15835           0 4.342446              38.25714
##   total_wait_minutes_per_day
## 1                   44.76268
```

```
# Total expected daily cost (using the averaged daily total wait)
cost1 <- compute_daily_cost(
  total_wait_minutes_per_day = res1$overall$total_wait_minutes_per_day,
  n_servers = 2
)
cost1
```

```
##   waiting_cost staffing_cost total_cost
## 1     22.38134           540   562.3813
```

```
# Plot: waiting time distribution
hist(res1$customers$wait_time, breaks = 30,
     main = "System 1 (Pooled): Waiting Time Distribution\n(7 weeks of customer waits)",
     xlab = "Wait time (minutes)")
```



System 2:

```
simulate_day_system2 <- function(seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  # Customer storage
  customers <- data.frame(
    cust_id = integer(),
    arrival_time = numeric(),
    s1_start = numeric(), s1_end = numeric(), wait1 = numeric(),
    s2_start = numeric(), s2_end = numeric(), wait2 = numeric(),
    total_wait = numeric(),
    total_time_in_system = numeric()
  )

  # Two FIFO queues
  q1 <- integer()
```

```r
q2 <- integer()

# Server states (single server at each stage)
s1_busy_until <- Inf
s1_job <- NA_integer_

s2_busy_until <- Inf
s2_job <- NA_integer_

s1_idle <- function() s1_busy_until == Inf
s2_idle <- function() s2_busy_until == Inf

# Event list
event_list <- data.frame(time = numeric(), type = character(), server = integer(), cust_id = integer()

# customer id counter
next_id <- 1

# helper: start service at server 1 if possible
start_s1_if_possible <- function(t_now, event_list){
  if (s1_idle() && length(q1) > 0) {
    cid <- q1[1]; q1 <<- q1[-1]

    st <- t_now
    svc <- sample_service_time(1)
    et <- t_now + svc

    customers$s1_start[customers$cust_id == cid] <<- st
    customers$s1_end[customers$cust_id == cid]   <<- et
    customers$wait1[customers$cust_id == cid]    <<- st - customers$arrival_time[customers$cust_id ==

    s1_busy_until <<- et
    s1_job <<- cid

    event_list <- add_event(event_list, time = et, type = "depart1", server = 1, cust_id = cid)
  }
  event_list
}

# helper: start service at server 2 if possible
start_s2_if_possible <- function(t_now, event_list){
  if (s2_idle() && length(q2) > 0) {
    cid <- q2[1]; q2 <<- q2[-1]

    st <- t_now
    svc <- sample_service_time(1)
    et <- t_now + svc

    customers$s2_start[customers$cust_id == cid] <<- st
    customers$s2_end[customers$cust_id == cid]   <<- et
    customers$wait2[customers$cust_id == cid]    <<- st - customers$s1_end[customers$cust_id == cid]

    # total wait and total time in system now known
```

```
    customers$total_wait[customers$cust_id == cid] <<-
      customers$wait1[customers$cust_id == cid] + customers$wait2[customers$cust_id == cid]

    customers$total_time_in_system[customers$cust_id == cid] <<- et - customers$arrival_time[customers

    s2_busy_until <<- et
    s2_job <<- cid

    event_list <- add_event(event_list, time = et, type = "depart2", server = 2, cust_id = cid)
  }
  event_list
}

# Schedule first arrival
t_first <- next_arrival_time(0)
if (is.finite(t_first) && t_first < T_END) {
  event_list <- add_event(event_list, time = t_first, type = "arrival")
}

# Main event loop
while (nrow(event_list) > 0) {
  popped <- pop_next_event(event_list)
  ev <- popped$event
  event_list <- popped$event_list
  t_now <- ev$time

  if (ev$type == "arrival") {
    cid <- next_id
    next_id <- next_id + 1

    customers <- rbind(
      customers,
      data.frame(
        cust_id = cid,
        arrival_time = t_now,
        s1_start = NA_real_, s1_end = NA_real_, wait1 = NA_real_,
        s2_start = NA_real_, s2_end = NA_real_, wait2 = NA_real_,
        total_wait = NA_real_,
        total_time_in_system = NA_real_
      )
    )

    # join queue 1
    q1 <- c(q1, cid)

    # schedule next arrival
    t_next <- next_arrival_time(t_now)
    if (is.finite(t_next) && t_next < T_END) {
      event_list <- add_event(event_list, time = t_next, type = "arrival")
    }

    # start service at s1 if idle
    event_list <- start_s1_if_possible(t_now, event_list)
```

```
    } else if (ev$type == "depart1") {
      # server 1 finishes; customer moves to queue 2
      s1_busy_until <- Inf
      s1_job <- NA_integer_

      cid <- ev$cust_id
      q2 <- c(q2, cid)

      # immediately try starting s1 (next in q1) and s2 (from q2)
      event_list <- start_s1_if_possible(t_now, event_list)
      event_list <- start_s2_if_possible(t_now, event_list)

    } else if (ev$type == "depart2") {
      # server 2 finishes, customer exits system
      s2_busy_until <- Inf
      s2_job <- NA_integer_

      # start next at s2 if possible
      event_list <- start_s2_if_possible(t_now, event_list)
    }
  }

  customers
}
```

A) one-day event-driven simulator (tandem)

```
simulate_week_system2 <- function(seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  days <- vector("list", 5)
  for (d in 1:5){
    days[[d]] <- simulate_day_system2()
    days[[d]]$day <- d
  }
  do.call(rbind, days)
}

run_system2_experiment <- function(n_weeks = 7, seed = 123){
  set.seed(seed)

  all_weeks <- vector("list", n_weeks)
  week_summaries <- data.frame(
    week = integer(),
    mean_total_wait = numeric(),
    p90_total_wait = numeric(),
    mean_time_in_system = numeric(),
    avg_customers_per_day = numeric(),
    total_wait_minutes_per_day = numeric()
  )
```

```
  for (w in 1:n_weeks){
    cust <- simulate_week_system2()
    cust$week <- w
    all_weeks[[w]] <- cust

    # total waiting minutes per day (sum total_wait within each day)
    daily_totals <- aggregate(total_wait ~ day, data = cust, FUN = function(x) sum(x, na.rm = TRUE))
    mean_total_wait_per_day <- mean(daily_totals$total_wait)

    waits <- cust$total_wait
    waits <- waits[is.finite(waits)]

    tis <- cust$total_time_in_system
    tis <- tis[is.finite(tis)]

    week_summaries <- rbind(
      week_summaries,
      data.frame(
        week = w,
        mean_total_wait = mean(waits),
        p90_total_wait = unname(quantile(waits, 0.90)),
        mean_time_in_system = mean(tis),
        avg_customers_per_day = nrow(cust)/5,
        total_wait_minutes_per_day = mean_total_wait_per_day
      )
    )
  }

  all_customer_data <- do.call(rbind, all_weeks)

  overall <- data.frame(
    mean_total_wait = mean(week_summaries$mean_total_wait),
    p90_total_wait = mean(week_summaries$p90_total_wait),
    mean_time_in_system = mean(week_summaries$mean_time_in_system),
    avg_customers_per_day = mean(week_summaries$avg_customers_per_day),
    total_wait_minutes_per_day = mean(week_summaries$total_wait_minutes_per_day)
  )

  list(
    customers = all_customer_data,
    week_summaries = week_summaries,
    overall = overall
  )
}
```

B) Week + 7-week experiment wrapper for System 2

```
res2 <- run_system2_experiment(n_weeks = 7, seed = 1)
res2$overall
```

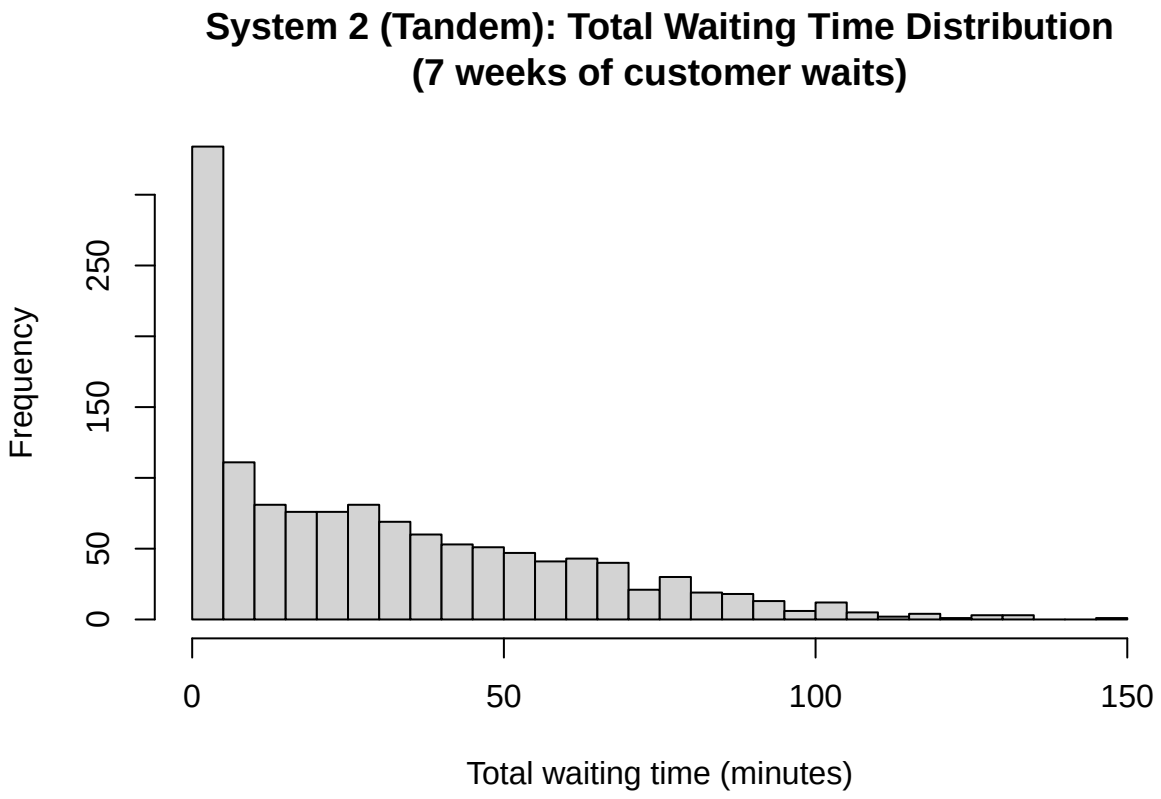
C) Run System 2 + required plot + daily cost

```
##   mean_total_wait p90_total_wait mean_time_in_system avg_customers_per_day
## 1        29.18839       70.91036            49.46122              37.17143
##   total_wait_minutes_per_day
## 1                   1111.618
```

```
# Daily cost: still 2 servers total (one at each stage), 9 hours/day
cost2 <- compute_daily_cost(
  total_wait_minutes_per_day = res2$overall$total_wait_minutes_per_day,
  n_servers = 2
)
cost2
```

```
##   waiting_cost staffing_cost total_cost
## 1     555.8091           540   1095.809
```

```
# Plot: distribution of total waiting time
hist(res2$customers$total_wait, breaks = 30,
     main = "System 2 (Tandem): Total Waiting Time Distribution\n(7 weeks of customer waits)",
     xlab = "Total waiting time (minutes)")
```



System 3: one-day simulator (separate lines, optional switching)

Switching rule (From my chosen difficulty level: Hard) At each service completion, we allow switching if the difference in queue lengths is >= 2. We move the last person in the longer line to the end of the shorter line. This would be helpful in scenarios where someone near the back decides to swap.

```r
simulate_day_system3 <- function(switching = FALSE, switch_threshold = 2, seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  customers <- data.frame(
    cust_id = integer(),
    arrival_time = numeric(),
    service_start = numeric(),
    service_end = numeric(),
    wait_time = numeric(),
    chosen_line = integer()
  )

  # Two separate FIFO queues (store customer IDs)
  q <- list(integer(), integer())

  # Two servers
  busy_until <- c(Inf, Inf)
  server_job <- c(NA_integer_, NA_integer_)

  is_idle <- function(s) busy_until[s] == Inf

  # Event list
  event_list <- data.frame(time = numeric(), type = character(), server = integer(), cust_id = integer()

  next_id <- 1

  # Choose a line at arrival: shorter queue (tie-break random)
  choose_line <- function(){
    len1 <- length(q[[1]])
    len2 <- length(q[[2]])
    if (len1 < len2) return(1L)
    if (len2 < len1) return(2L)
    sample(c(1L, 2L), 1) # tie
  }

  # Start service for a given server if idle and its queue nonempty
  start_service <- function(t_now, s, event_list){
    if (is_idle(s) && length(q[[s]]) > 0){
      cid <- q[[s]][1]
      q[[s]] <<- q[[s]][-1]

      st <- t_now
      svc <- sample_service_time(1)
      et <- t_now + svc

      customers$service_start[customers$cust_id == cid] <<- st
      customers$service_end[customers$cust_id == cid]   <<- et
      customers$wait_time[customers$cust_id == cid]     <<- st - customers$arrival_time[customers$cust_

      busy_until[s] <<- et
      server_job[s] <<- cid
```

```
    event_list <- add_event(event_list, time = et, type = "depart", server = s, cust_id = cid)
  }
  event_list
}

# Hard feature: allow switching after a departure event
do_switching_if_needed <- function(){
  len1 <- length(q[[1]])
  len2 <- length(q[[2]])
  diff <- abs(len1 - len2)

  if (diff >= switch_threshold){
    longer <- if (len1 > len2) 1L else 2L
    shorter <- if (longer == 1L) 2L else 1L

    # move last person from longer to end of shorter
    mover <- tail(q[[longer]], 1)
    q[[longer]] <<- q[[longer]][-length(q[[longer]])]
    q[[shorter]] <<- c(q[[shorter]], mover)
  }
}

# schedule first arrival
t_first <- next_arrival_time(0)
if (is.finite(t_first) && t_first < T_END){
  event_list <- add_event(event_list, time = t_first, type = "arrival")
}

# main loop
while (nrow(event_list) > 0){
  popped <- pop_next_event(event_list)
  ev <- popped$event
  event_list <- popped$event_list
  t_now <- ev$time

  if (ev$type == "arrival"){
    cid <- next_id
    next_id <- next_id + 1

    line <- choose_line()

    customers <- rbind(
      customers,
      data.frame(
        cust_id = cid,
        arrival_time = t_now,
        service_start = NA_real_,
        service_end = NA_real_,
        wait_time = NA_real_,
        chosen_line = line
      )
    )
```

```
      # join that line
      q[[line]] <- c(q[[line]], cid)

      # schedule next arrival
      t_next <- next_arrival_time(t_now)
      if (is.finite(t_next) && t_next < T_END){
        event_list <- add_event(event_list, time = t_next, type = "arrival")
      }

      # try starting service on that line if its server is idle
      event_list <- start_service(t_now, line, event_list)

      # if the other server is idle, it can start too
      other <- if (line == 1L) 2L else 1L
      event_list <- start_service(t_now, other, event_list)

    } else if (ev$type == "depart"){
      s <- ev$server

      # free server s
      busy_until[s] <- Inf
      server_job[s] <- NA_integer_

      # optional switching happens at service completions
      if (switching) do_switching_if_needed()

      # start service on both servers if possible
      event_list <- start_service(t_now, 1, event_list)
      event_list <- start_service(t_now, 2, event_list)
    }
  }

  customers
}
```

A)

```
simulate_week_system3 <- function(switching = FALSE, switch_threshold = 2, seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  days <- vector("list", 5)
  for (d in 1:5){
    days[[d]] <- simulate_day_system3(switching = switching, switch_threshold = switch_threshold)
    days[[d]]$day <- d
  }
  do.call(rbind, days)
}

run_system3_experiment <- function(n_weeks = 7, switching = FALSE, switch_threshold = 2, seed = 123){
  set.seed(seed)
```

```
  all_weeks <- vector("list", n_weeks)
  week_summaries <- data.frame(
    week = integer(),
    mean_wait = numeric(),
    p90_wait = numeric(),
    avg_customers_per_day = numeric(),
    total_wait_minutes_per_day = numeric()
  )

  for (w in 1:n_weeks){
    cust <- simulate_week_system3(switching = switching, switch_threshold = switch_threshold)
    cust$week <- w
    all_weeks[[w]] <- cust

    daily_totals <- aggregate(wait_time ~ day, data = cust, FUN = function(x) sum(x, na.rm = TRUE))
    mean_total_wait_per_day <- mean(daily_totals$wait_time)

    waits <- cust$wait_time
    waits <- waits[is.finite(waits)]

    week_summaries <- rbind(
      week_summaries,
      data.frame(
        week = w,
        mean_wait = mean(waits),
        p90_wait = unname(quantile(waits, 0.90)),
        avg_customers_per_day = nrow(cust)/5,
        total_wait_minutes_per_day = mean_total_wait_per_day
      )
    )
  }

  all_customer_data <- do.call(rbind, all_weeks)

  overall <- data.frame(
    mean_wait = mean(week_summaries$mean_wait),
    p90_wait = mean(week_summaries$p90_wait),
    avg_customers_per_day = mean(week_summaries$avg_customers_per_day),
    total_wait_minutes_per_day = mean(week_summaries$total_wait_minutes_per_day)
  )

  list(customers = all_customer_data, week_summaries = week_summaries, overall = overall)
}
```

B) Week + 7-week experiment wrappers (System 3)

```
# No switching
res3_no <- run_system3_experiment(n_weeks = 7, switching = FALSE, seed = 1)
res3_no$overall
```

C) Run System 3: without switching vs with switching + plot (distribution comparison)

```
##   mean_wait p90_wait avg_customers_per_day total_wait_minutes_per_day
## 1  2.674667 9.156186              37.71429                    100.7513
```

```
cost3_no <- compute_daily_cost(res3_no$overall$total_wait_minutes_per_day, n_servers = 2)
cost3_no
```

```
##   waiting_cost staffing_cost total_cost
## 1     50.37566           540   590.3757
```

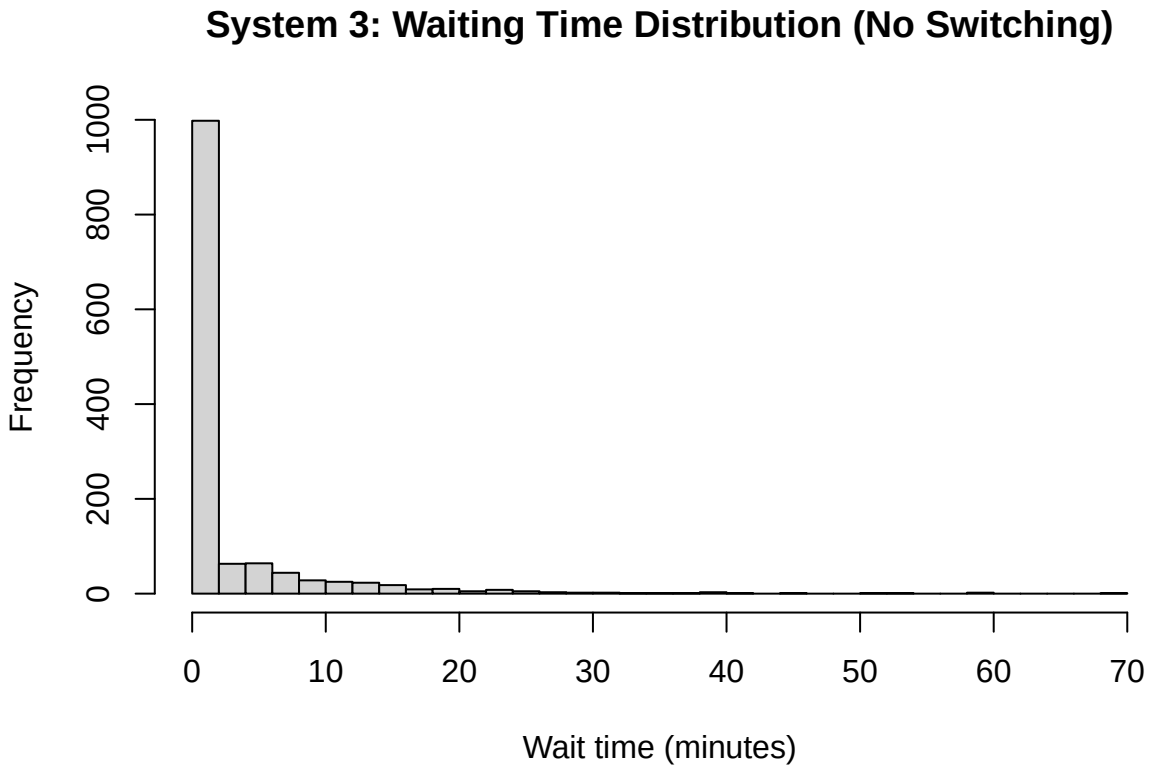
```
# With switching
res3_sw <- run_system3_experiment(n_weeks = 7, switching = TRUE, switch_threshold = 2, seed = 1)
res3_sw$overall
```

```
##   mean_wait p90_wait avg_customers_per_day total_wait_minutes_per_day
## 1  2.498084 8.233334                    38                    95.12854
```

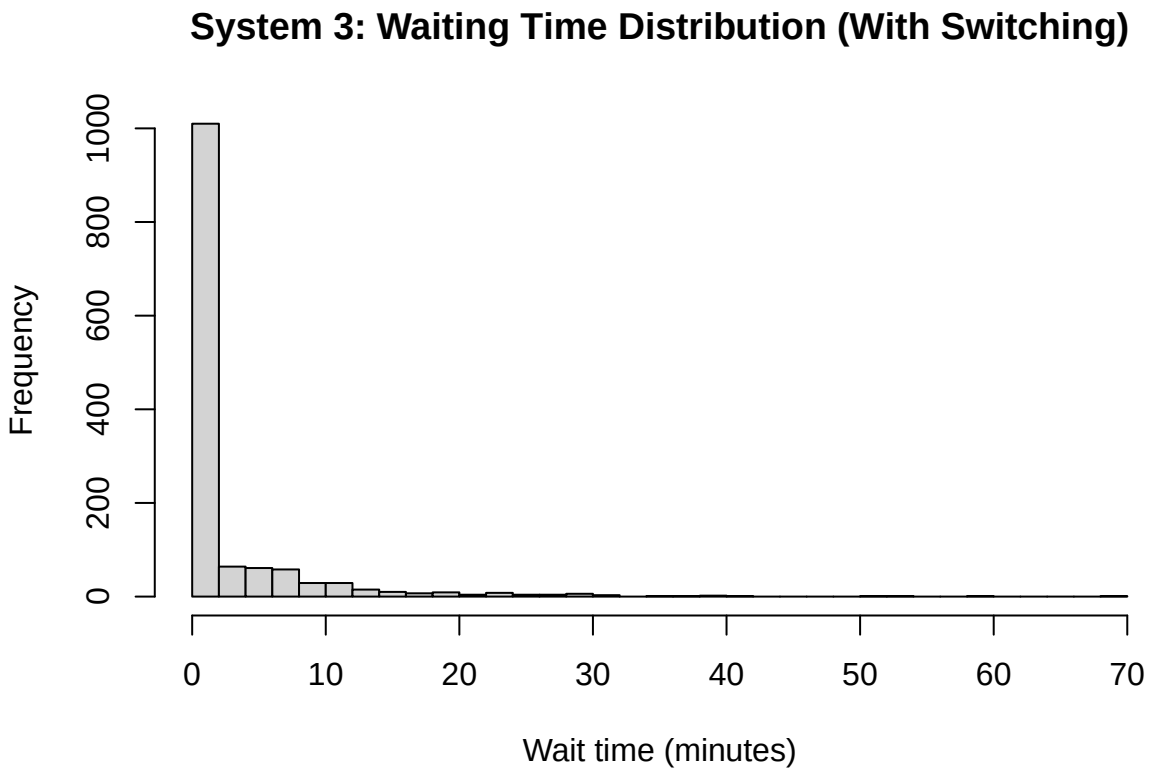
```
cost3_sw <- compute_daily_cost(res3_sw$overall$total_wait_minutes_per_day, n_servers = 2)
cost3_sw
```

```
##   waiting_cost staffing_cost total_cost
## 1     47.56427           540   587.5643
```

```
# plot (distribution comparison)
hist(res3_no$customers$wait_time, breaks = 30,
     main = "System 3: Waiting Time Distribution (No Switching)",
     xlab = "Wait time (minutes)")
```



```r
hist(res3_sw$customers$wait_time, breaks = 30,
     main = "System 3: Waiting Time Distribution (With Switching)",
     xlab = "Wait time (minutes)")
```



Compare all systems (same 5 days × 7 weeks framework)

```r
# Helper: build a single summary row for any system result
make_row <- function(system_name, mean_wait, p90_wait, total_wait_minutes_per_day, n_server_hours_per_da
  cost <- compute_daily_cost(
    total_wait_minutes_per_day = total_wait_minutes_per_day,
    n_servers = 0
  )

  # overwrite staffing cost using server-hours
  staffing_cost <- 30 * n_server_hours_per_day
  waiting_cost  <- 0.50 * total_wait_minutes_per_day
  total_cost    <- staffing_cost + waiting_cost

  data.frame(
    system = system_name,
    mean_wait_min = mean_wait,
    p90_wait_min = p90_wait,
    total_wait_min_per_day = total_wait_minutes_per_day,
    staffing_cost = staffing_cost,
    waiting_cost = waiting_cost,
    total_cost = total_cost
```

```
  )
}

compare_core_systems <- function(seed = 1){
  # System 1: pooled (2 servers all day)
  res1 <- run_system1_experiment(n_weeks = 7, n_servers = 2, seed = seed)
  row1 <- make_row(
    system_name = "System 1: Pooled (2 servers)",
    mean_wait = res1$overall$mean_wait,
    p90_wait = res1$overall$p90_wait,
    total_wait_minutes_per_day = res1$overall$total_wait_minutes_per_day,
    n_server_hours_per_day = 2 * 9
  )

  # System 2: tandem (1 server at stage 1 + 1 at stage 2 = 2 total all day)
  res2 <- run_system2_experiment(n_weeks = 7, seed = seed)
  row2 <- make_row(
    system_name = "System 2: Tandem (1+1 servers)",
    mean_wait = res2$overall$mean_total_wait,
    p90_wait = res2$overall$p90_total_wait,
    total_wait_minutes_per_day = res2$overall$total_wait_minutes_per_day,
    n_server_hours_per_day = 2 * 9
  )

  # System 3: separate lines, no switching (2 servers all day)
  res3_no <- run_system3_experiment(n_weeks = 7, switching = FALSE, seed = seed)
  row3 <- make_row(
    system_name = "System 3: Separate (no switching)",
    mean_wait = res3_no$overall$mean_wait,
    p90_wait = res3_no$overall$p90_wait,
    total_wait_minutes_per_day = res3_no$overall$total_wait_minutes_per_day,
    n_server_hours_per_day = 2 * 9
  )

  # System 3: separate lines, with switching
  res3_sw <- run_system3_experiment(n_weeks = 7, switching = TRUE, switch_threshold = 2, seed = seed)
  row4 <- make_row(
    system_name = "System 3: Separate (with switching)",
    mean_wait = res3_sw$overall$mean_wait,
    p90_wait = res3_sw$overall$p90_wait,
    total_wait_minutes_per_day = res3_sw$overall$total_wait_minutes_per_day,
    n_server_hours_per_day = 2 * 9
  )

  out <- rbind(row1, row2, row3, row4)
  out <- out[order(out$total_cost), ]
  out
}

comp <- compare_core_systems(seed = 1)
comp
```

```
##                                system mean_wait_min p90_wait_min
```

```
## 1        System 1: Pooled (2 servers)      1.158350     4.342446
## 4 System 3: Separate (with switching)      2.498084     8.233334
## 3   System 3: Separate (no switching)      2.674667     9.156186
## 2      System 2: Tandem (1+1 servers)     29.188391    70.910356
##   total_wait_min_per_day staffing_cost waiting_cost total_cost
## 1               44.76268           540     22.38134   562.3813
## 4               95.12854           540     47.56427   587.5643
## 3              100.75131           540     50.37566   590.3757
## 2             1111.61822           540    555.80911  1095.8091
```

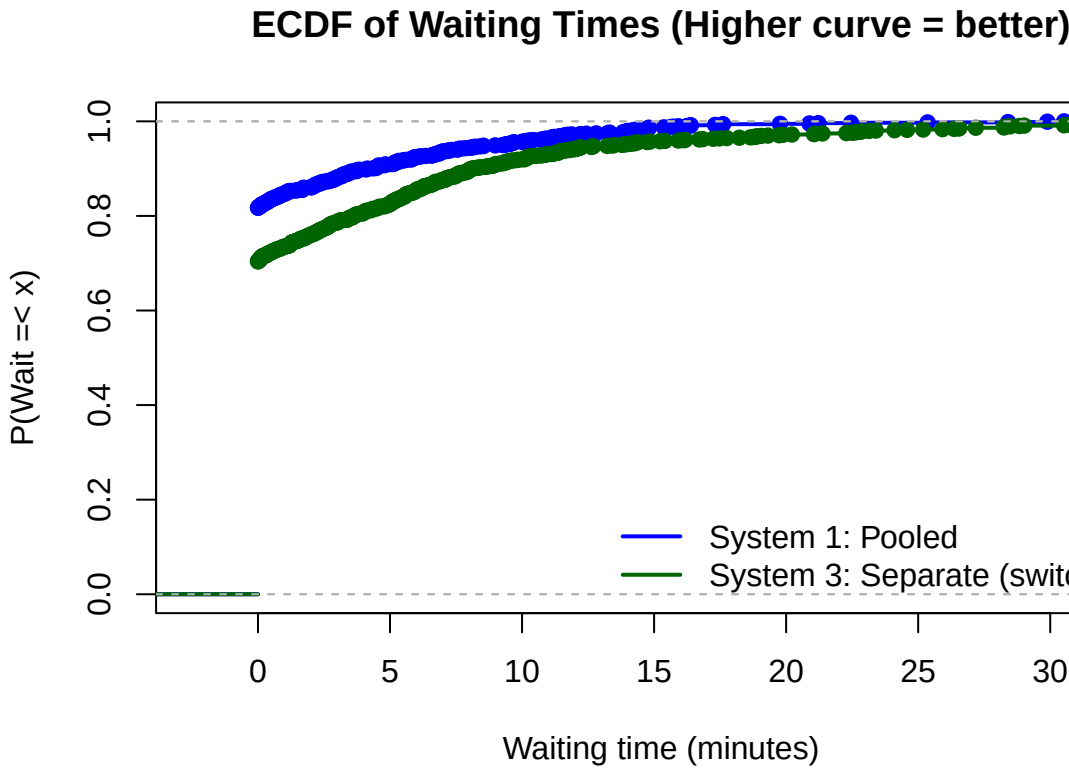
```r
switch_sensitivity <- function(thresholds = c(1,2,3), seed = 1){
  out <- data.frame()

  for (th in thresholds){
    res <- run_system3_experiment(n_weeks = 7, switching = TRUE, switch_threshold = th, seed = seed)

    # staffing is same: 2 servers for 9 hours
    staffing_cost <- 30 * (2 * 9)

    waiting_cost <- 0.50 * res$overall$total_wait_minutes_per_day
    total_cost <- staffing_cost + waiting_cost

    out <- rbind(out, data.frame(
      threshold = th,
      mean_wait = res$overall$mean_wait,
      p90_wait = res$overall$p90_wait,
      total_wait_min_per_day = res$overall$total_wait_minutes_per_day,
      total_cost = total_cost
    ))
  }

  out
}

tab_switch <- switch_sensitivity(c(1,2,3), seed = 1)
print(tab_switch)
```

Sensitivity analysis: best minimal add-on (switching threshold)

```
##   threshold mean_wait p90_wait total_wait_min_per_day total_cost
## 1         1  2.699255 9.285280              101.71028   590.8551
## 2         2  2.498084 8.233334               95.12854   587.5643
## 3         3  2.674667 9.156186              100.75131   590.3757
```

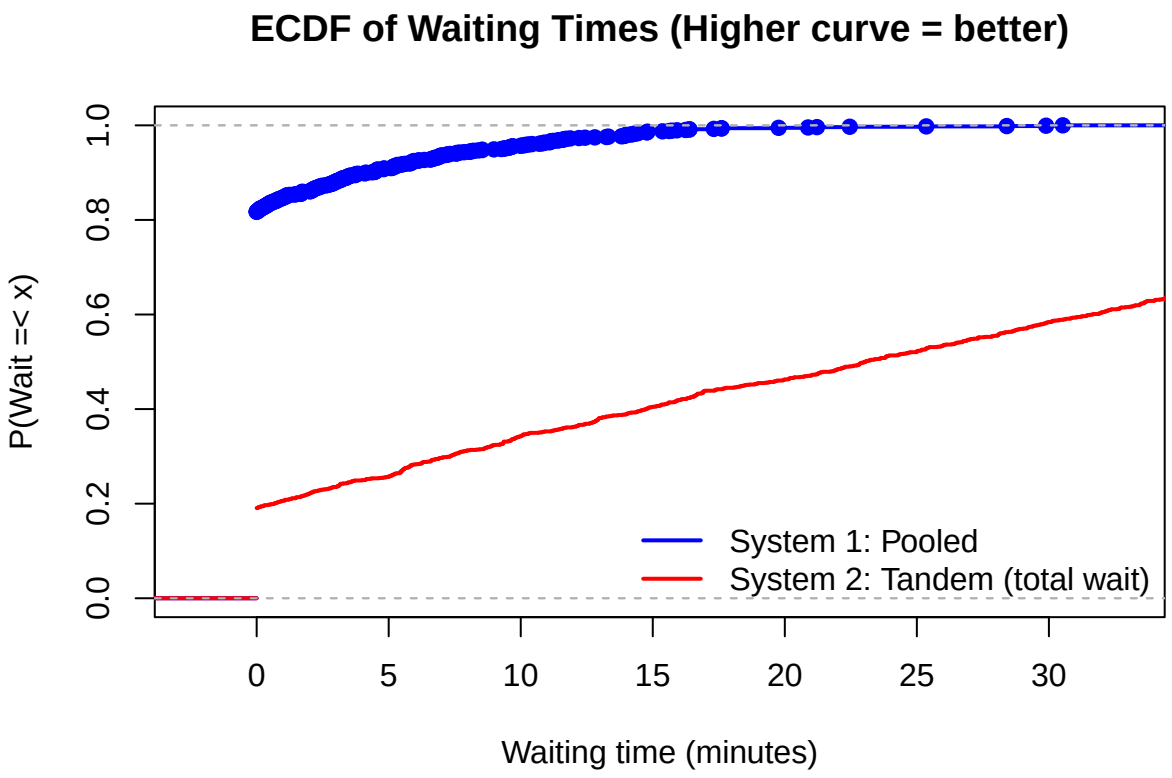
The results were not strictly monotone: a **moderate switching rule (threshold = 2)** produced the lowest mean and tail waiting times and the lowest cost among the switching scenarios. Both **more aggressive switching (threshold = 1)** and **less frequent switching (threshold = 3)** performed slightly worse, likely because excessive switching can create churn without consistently improving balance. Importantly, across all thresholds, System 3 remained more expensive than pooled service, so the overall conclusion (System 1 is lowest cost) did not change. Even at its best (threshold = 2), System 3's cost (**$587.56/day**) is still higher than System 1 pooled (**$562.38/day**).

```r
# compare pooled vs best separate (switching)
plot_ecdf_compare <- function(waits_a, waits_b, name_a, name_b,
                              col_a = "blue", col_b = "red", lwd = 2){
  waits_a <- waits_a[is.finite(waits_a)]
  waits_b <- waits_b[is.finite(waits_b)]

  ec_a <- ecdf(waits_a)
  ec_b <- ecdf(waits_b)

  plot(ec_a,
       main = "ECDF of Waiting Times (Higher curve = better)",
       xlab = "Waiting time (minutes)",
       ylab = "P(Wait =< x)",
       col = col_a, lwd = lwd)

  lines(ec_b, col = col_b, lwd = lwd)

  legend("bottomright",
         legend = c(name_a, name_b),
         col = c(col_a, col_b),
         lwd = c(lwd, lwd),
         bty = "n")
}


res1 <- run_system1_experiment(n_weeks = 7, n_servers = 2, seed = 1)
res3_sw <- run_system3_experiment(n_weeks = 7, switching = TRUE, switch_threshold = 2, seed = 1)
res2 <- run_system2_experiment(n_weeks = 7, seed = 1)

# Best vs separate-with-switching
plot_ecdf_compare(
  res1$customers$wait_time,
  res3_sw$customers$wait_time,
  "System 1: Pooled",
  "System 3: Separate (switching)",
  col_a = "blue", col_b = "darkgreen"
)
```



Plot to support conclusion:

```
# Best vs tandem (total wait)
plot_ecdf_compare(
  res1$customers$wait_time,
  res2$customers$total_wait,
  "System 1: Pooled",
  "System 2: Tandem (total wait)",
  col_a = "blue", col_b = "red"
)
```



3) Realism feature + sensitivity: 2 vs 3 servers during 12–1pm (choose cheaper total cost)

```
simulate_day_system1_lunch_extra <- function(seed = NULL){
  if (!is.null(seed)) set.seed(seed)

  customers <- data.frame(
    cust_id = integer(),
    arrival_time = numeric(),
    service_start = numeric(),
    service_end = numeric(),
    wait_time = numeric(),
    server = integer()
  )

  queue <- integer()

  # 3 servers exist, but server 3 can only START jobs during lunch (240-300)
  busy_until <- c(Inf, Inf, Inf)
  server_job <- c(NA_integer_, NA_integer_, NA_integer_)

  server_can_start <- function(s, t_now){
    if (s %in% c(1,2)) return(TRUE)
    # server 3:
    (t_now >= T_LUNCH_START) && (t_now < T_LUNCH_END)
  }
```

```r
idle_servers <- function(t_now){
  which(busy_until == Inf & sapply(1:3, function(s) server_can_start(s, t_now)))
}

start_service_if_possible <- function(t_now, event_list){
  ids <- idle_servers(t_now)
  while (length(ids) > 0 && length(queue) > 0){
    s <- ids[1]
    cid <- queue[1]
    queue <<- queue[-1]

    st <- t_now
    svc <- sample_service_time(1)
    et <- t_now + svc

    customers$service_start[customers$cust_id == cid] <<- st
    customers$service_end[customers$cust_id == cid]   <<- et
    customers$wait_time[customers$cust_id == cid]     <<- st - customers$arrival_time[customers$cust_
    customers$server[customers$cust_id == cid]        <<- s

    busy_until[s] <<- et
    server_job[s] <<- cid

    event_list <- add_event(event_list, time = et, type = "depart", server = s, cust_id = cid)

    ids <- idle_servers(t_now)
  }
  event_list
}

event_list <- data.frame(time = numeric(), type = character(), server = integer(), cust_id = integer()
t_first <- next_arrival_time(0)
if (is.finite(t_first) && t_first < T_END){
  event_list <- add_event(event_list, time = t_first, type = "arrival")
}

next_id <- 1

while (nrow(event_list) > 0){
  popped <- pop_next_event(event_list)
  ev <- popped$event
  event_list <- popped$event_list
  t_now <- ev$time

  if (ev$type == "arrival"){
    cid <- next_id; next_id <- next_id + 1
    customers <- rbind(customers, data.frame(
      cust_id = cid, arrival_time = t_now,
      service_start = NA_real_, service_end = NA_real_, wait_time = NA_real_, server = NA_integer_
    ))

    queue <- c(queue, cid)
```

```
      t_next <- next_arrival_time(t_now)
      if (is.finite(t_next) && t_next < T_END){
        event_list <- add_event(event_list, time = t_next, type = "arrival")
      }

      event_list <- start_service_if_possible(t_now, event_list)

    } else if (ev$type == "depart"){
      s <- ev$server
      busy_until[s] <- Inf
      server_job[s] <- NA_integer_

      # after any departure, try to start jobs again
      event_list <- start_service_if_possible(t_now, event_list)
    }
  }

  customers
}

simulate_week_system1_lunch_extra <- function(seed = NULL){
  if (!is.null(seed)) set.seed(seed)
  days <- vector("list", 5)
  for (d in 1:5){
    days[[d]] <- simulate_day_system1_lunch_extra()
    days[[d]]$day <- d
  }
  do.call(rbind, days)
}

run_system1_lunch_extra_experiment <- function(n_weeks = 7, seed = 123){
  set.seed(seed)

  week_summaries <- data.frame(week=integer(), mean_wait=numeric(), p90_wait=numeric(), total_wait_minu
  all_weeks <- vector("list", n_weeks)

  for (w in 1:n_weeks){
    cust <- simulate_week_system1_lunch_extra()
    cust$week <- w
    all_weeks[[w]] <- cust

    daily_totals <- aggregate(wait_time ~ day, data = cust, FUN = function(x) sum(x, na.rm=TRUE))
    mean_total_wait_per_day <- mean(daily_totals$wait_time)

    waits <- cust$wait_time
    waits <- waits[is.finite(waits)]

    week_summaries <- rbind(week_summaries, data.frame(
      week=w,
      mean_wait = mean(waits),
      p90_wait = unname(quantile(waits, 0.90)),
      total_wait_minutes_per_day = mean_total_wait_per_day
    ))
```

```
  }

  overall <- data.frame(
    mean_wait = mean(week_summaries$mean_wait),
    p90_wait = mean(week_summaries$p90_wait),
    total_wait_minutes_per_day = mean(week_summaries$total_wait_minutes_per_day)
  )

  list(customers = do.call(rbind, all_weeks), week_summaries = week_summaries, overall = overall)
}
```

Pooled system with a lunch-only extra server

```
# Baseline pooled: 2 servers all day
res1_base <- run_system1_experiment(n_weeks = 7, n_servers = 2, seed = 1)
row_base <- make_row(
  "System 1: Pooled (2 servers all day)",
  mean_wait = res1_base$overall$mean_wait,
  p90_wait = res1_base$overall$p90_wait,
  total_wait_minutes_per_day = res1_base$overall$total_wait_minutes_per_day,
  n_server_hours_per_day = 2*9
)

# Lunch staffing: +1 server during 12-1 => adds 1 server-hour/day
res1_lunch <- run_system1_lunch_extra_experiment(n_weeks = 7, seed = 1)
row_lunch <- make_row(
  "System 1: Pooled (3rd server during 12-1)",
  mean_wait = res1_lunch$overall$mean_wait,
  p90_wait = res1_lunch$overall$p90_wait,
  total_wait_minutes_per_day = res1_lunch$overall$total_wait_minutes_per_day,
  n_server_hours_per_day = 2*9 + 1
)

rbind(row_base, row_lunch)[order(c(row_base$total_cost, row_lunch$total_cost)), ]
```

Running the staffing tradeoff

```
##                                     system mean_wait_min p90_wait_min
## 1      System 1: Pooled (2 servers all day)     1.1583496     4.342446
## 2 System 1: Pooled (3rd server during 12-1)     0.7836459     2.017229
##   total_wait_min_per_day staffing_cost waiting_cost total_cost
## 1               44.76268           540     22.38134   562.3813
## 2               29.31863           570     14.65931   584.6593
```

Decision definition: I define the best queueing **system as the one that minimizes total expected daily cost = staffing cost + waiting cost, where waiting cost is** $0.50 per customer-minute and staffing cost is $30 per server-hour (9 hours/day).

Final recommendation: Based on 7 simulated work weeks (5 days each), the system with the lowest expected daily total cost is System 1 (single pooled line with two parallel servers). Under System 1, the estimated mean waiting time is 1.16 minutes and the 90th percentile waiting time is 4.34 minutes, with an average of 44.76 total waiting minutes per day. This corresponds to an expected daily waiting cost of $22.38, staffing cost of $540, and total expected daily cost of $562.38. In comparison, separate lines performed worse even with switching ($587.56/day, mean wait 2.50 min), and tandem service performed dramatically worse ($1095.81/day, mean total wait 29.19 min). Therefore, I recommend the urgent care operate a single pooled queue feeding two servers.