Algorithmic Domain Translation Series
Volume 2 - Airport Boarding Simulation
Advanced Cabin Dynamics

Publication Edition - Boolean Identity Version - People Movement Only Before Take-Off

Purpose of this publication

This volume extends the Volume 1 aircraft-aisle backbone into seat interference, temporary aisle reoccupation, multiple blocker yielding, reseating and cabin flow recovery. The final update explicitly models one independent dataset for each physical aisle and clarifies that passengers do not deviate to another aisle after choosing the aisle serving their seat bank.

1. Scope Boundary

This volume models standard economy-class seating before take-off only. It begins when the passenger has reached the assigned row, which is the completion point of Volume 1, and ends when the target passenger is seated, all displaced passengers have reseated, the aisle is restored and the cabin is ready for departure.

Main modelling simplification

The primary model assumes front-door boarding and local aisle assignment. Passengers walk from the front toward the rear. If seated passengers block access to the target seat, those blockers stand and temporarily enter rearward aisle tiles until the target passenger can sit.

2. Relationship to Volume 1

Volume	Question	Movement environment	Completion point
Volume 1 - Basic Aisle Dynamics	Can the passenger reach the assigned row?	One-dimensional aircraft aisle movement.	Passenger reaches assigned row and leaves the aisle model.
Volume 2 - Advanced Cabin Dynamics	Can the passenger reach the assigned seat?	Local cooperative seat-row movement plus temporary aisle reoccupation.	Target passenger sits, blockers reseat and aisle flow resumes.

3. Publication Infographic

The infographic below is the primary teaching artefact. It has been rebuilt on a larger canvas with clearer spacing, a per-aisle dataset model and a clearer multiple-blocker sequence.

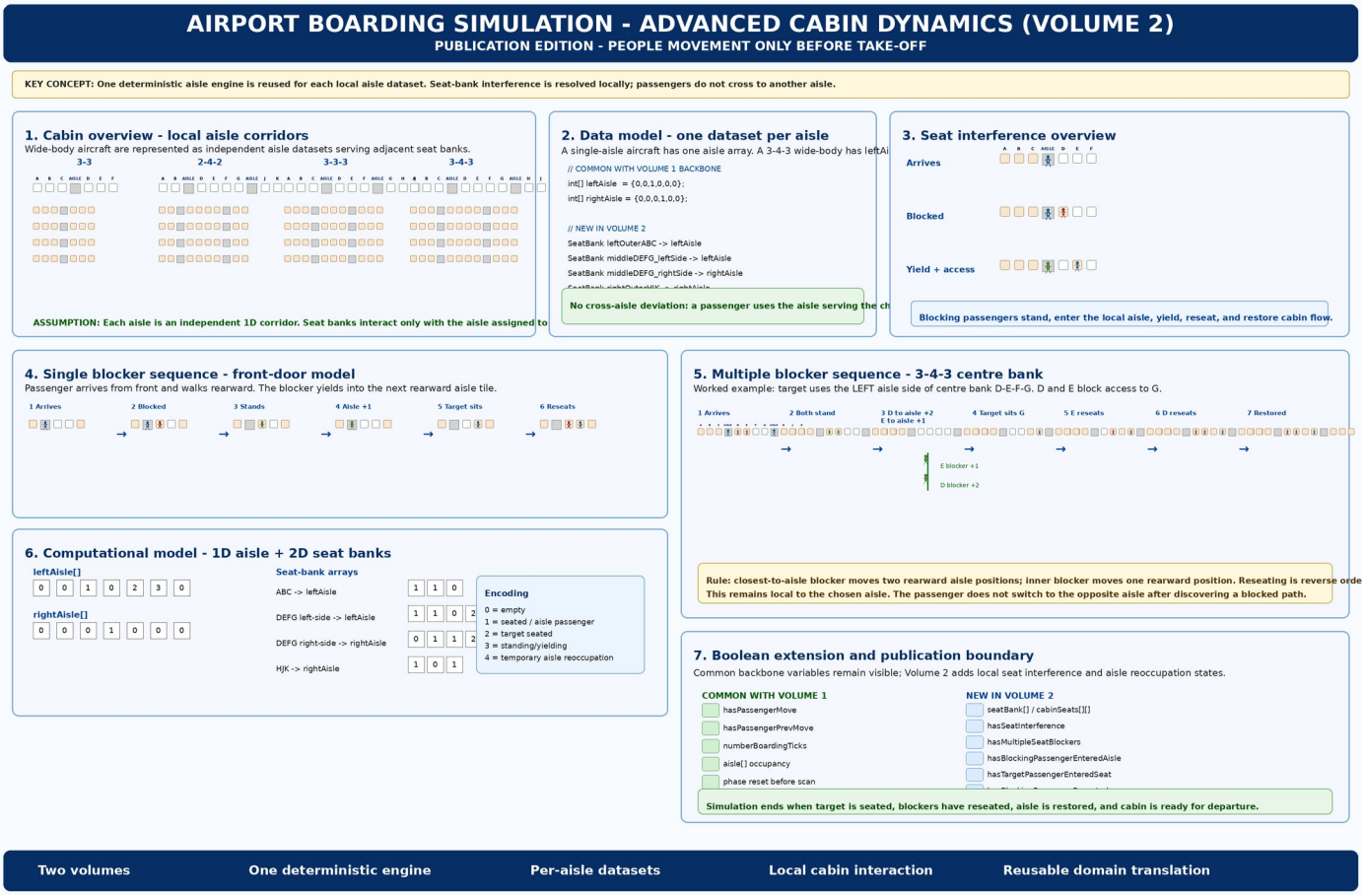

Figure 1 - Volume 2 publication infographic: per-aisle datasets, local seat-bank interference and final multiple-blocker yielding sequence.

4. Cabin Layouts and Independent Aisle Datasets

Wide-body aircraft introduce multiple physical aisles. The simulation does not merge all passengers into one global aisle array. Instead, each physical aisle maintains its own one-dimensional dataset. Seat banks are associated with the aisle that serves the passenger approach side.

Configuration	Local aisle datasets	Seat-bank association
3-3 narrow-body	centreAisle[]	ABC and DEF interact with the same aisle.
2-4-2 wide-body	leftAisle[], rightAisle[]	Left bank uses leftAisle; right bank uses rightAisle; the centre bank is assigned to the chosen approach side.
3-3-3 wide-body	leftAisle[], rightAisle[]	Outer banks use adjacent aisles; middle bank uses the selected serving aisle.
3-4-3 wide-body	leftAisle[], rightAisle[]	The centre DEFG bank may be approached from either side in principle, but a passenger does not switch aisles during the local interaction.

No cross-aisle deviation

A passenger assigned to a middle-bank seat may theoretically have less interference from the opposite aisle, but the published simulation does not allow them to discover a blocked path and then switch to the other aisle. Once the passenger is associated with an aisle dataset, the seat-interference event is resolved locally within that aisle corridor.

5. Seat Interference and Multiple Blockers

The single-blocker case is the basic extension: one seated passenger blocks the path, stands, enters the next rearward aisle tile, allows the target passenger to sit, then reseats.

Final multiple-blocker update

If two seated passengers block access to a target seat, the blocker closest to the aisle enters the rearward aisle tile two positions away, while the next blocker enters the rearward aisle tile one position away. After the target passenger sits, blockers reseat in reverse order so the aisle returns to its original clear state.

This is important for 3-4-3, 2-4-2 and other economy configurations where a passenger may need to pass more than one seated person. The aisle can become more occupied before it becomes clear again. This temporary increase in aisle occupancy is the defining advanced cabin-dynamics behaviour of Volume 2.

6. Boolean Identity Extension

Status	Variable / state	Role
COMMON WITH VOLUME 1 BACKBONE	aisle[]	The aircraft aisle remains a one-dimensional occupancy array.
COMMON WITH VOLUME 1 BACKBONE	hasPassengerMove	A passenger has begun or is continuing a movement event.
COMMON WITH VOLUME 1 BACKBONE	hasPassengerPrevMove	Tracks movement continuation across phases.
COMMON WITH VOLUME 1 BACKBONE	numberBoardingTicks	The simulation still advances through deterministic ticks/phases.
NEW IN VOLUME 2	leftAisle[] / rightAisle[]	Each physical aisle can maintain an independent dataset.
NEW IN VOLUME 2	seatBank[] / cabinSeats[][]	Represents local seat-row occupancy.
NEW IN VOLUME 2	hasSeatInterference	The target seat cannot be reached directly.
NEW IN VOLUME 2	hasMultipleSeatBlockers	More than one seated passenger blocks the access path.
NEW IN VOLUME 2	hasBlockingPassengerEnteredAisle	One or more seated blockers temporarily reoccupy aisle tiles.
NEW IN VOLUME 2	hasTargetPassengerEnteredSeat	The arriving passenger has reached the assigned seat.
NEW IN VOLUME 2	hasBlockingPassengerReseated	Displaced passengers have returned to their original seats.
NEW IN VOLUME 2	hasCabinFlowResumed	The row interaction is resolved and the aircraft is ready for departure preparation.

7. Data Model

Volume 2 still uses the Volume 1 aisle array, but on wide-body aircraft the aircraft is represented as multiple local aisle datasets rather than one giant corridor. Local seat-bank arrays then connect to the aisle that serves them.

```
// COMMON WITH VOLUME 1 BACKBONE
int[] leftAisle  = new int[] {0, 0, 1, 0, 0, 0};
int[] rightAisle = new int[] {0, 0, 0, 1, 0, 0};

// NEW IN VOLUME 2: local seat-bank representation
// 0 = empty, 1 = seated, 2 = target seated, 3 = standing/yielding, 4 = temporary aisle reoccupation
int[] middleBankDEFG_leftSide  = new int[] {1, 1, 0, 0};
int[] middleBankDEFG_rightSide = new int[] {0, 0, 1, 1};

// The chosen side determines which aisle dataset is used.
// The passenger does not cross from leftAisle[] to rightAisle[] during a blocked seating event.
```

8. Reference Implementation

The following code is included directly in the guide. Comments marked COMMON WITH VOLUME 1 BACKBONE identify preserved areas from the original movement engine. Comments marked NEW IN VOLUME 2 identify the advanced cabin-dynamics extension, including per-aisle datasets and the final multiple-blocker update.

```
import java.util.Arrays;

/*
```

```
================================================================================
AIRPORT BOARDING SIMULATION - ADVANCED CABIN DYNAMICS (VOLUME 2)
PUBLICATION EDITION REFERENCE IMPLEMENTATION
================================================================================
Purpose:
- Reuse the Volume 1 one-dimensional aisle movement backbone.
- Add local economy seat-bank interference.
- Use one independent aisle dataset for each physical aisle.
- Do not allow cross-aisle deviation after the passenger has chosen/entered an aisle.
- Model people movement only before take-off.
- Include single-blocker and multiple-blocker handling.
================================================================================
*/
public class AirportBoardingAdvancedCabinDynamics_Publication {
    // COMMON WITH VOLUME 1 BACKBONE
    static final int AISLE_EMPTY = 0;
    static final int ARRIVING_PASSENGER_IN_AISLE = 1;

    // NEW IN VOLUME 2 - ADVANCED CABIN DYNAMICS
    static final int TEMPORARY_BLOCKER_IN_AISLE = 4;
    static final int SEAT_EMPTY = 0;
    static final int SEATED_PASSENGER = 1;
    static final int TARGET_PASSENGER_SEATED = 2;
    static final int STANDING_YIELDING_PASSENGER = 3;

    public static void main(String[] args) {
        System.out.println("VOLUME 2 - ADVANCED CABIN DYNAMICS");
        runSingleAisleExample();
        runWideBodyPerAisleExample();
    }

    static void runSingleAisleExample() {
        int[] cabinAisle = {0,0,1,0,0,0};
        char[] rightBankDEF = {'D','E','F'};
        int[] seats = {SEATED_PASSENGER, SEAT_EMPTY, SEATED_PASSENGER};
        simulateSeatInterference("3-3 narrow-body, passenger to 13E", cabinAisle, 2, rightBankDEF, seats, 1, true);
    }

    static void runWideBodyPerAisleExample() {
        // 3-4-3: A B C | LEFT AISLE | D E F G | RIGHT AISLE | H J K
        // Each physical aisle has its own dataset. A centre-bank passenger is associated
        // with the aisle used to approach that seat-bank side. The passenger does not
        // switch to the opposite aisle after finding blockers.
        int[] leftAisle  = {0,0,1,0,0,0,0};
        int[] rightAisle = {0,0,0,0,0,0,0};

        char[] middleBankDEFG = {'D','E','F','G'};
        int[] seatsFromLeftSide = {SEATED_PASSENGER, SEATED_PASSENGER, SEAT_EMPTY, SEAT_EMPTY};

        simulateSeatInterference("3-4-3 centre bank approached from LEFT aisle, target G", leftAisle, 2, middleBankDEFG, seatsFromLeftSide, 3, true);
    }

    static void simulateSeatInterference(String name, int[] aisle, int rowAisleIndex, char[] seatNames, int[] seatBank, int targetSeatIndex, boolean
aisleIsLeftOfBank) {
        boolean hasPassengerReachedRow = true;
        boolean hasSeatInterference = false;
        boolean hasMultipleSeatBlockers = false;
        boolean hasBlockingPassengerEnteredAisle = false;
        boolean hasTargetPassengerEnteredSeat = false;
        boolean hasBlockingPassengerReseated = false;
        boolean hasCabinFlowResumed = false;
```

```
    System.out.println("\n--- " + name + " ---");
    printState("Initial", aisle, seatNames, seatBank);

    int[] blockers = findBlockers(seatBank, targetSeatIndex, aisleIsLeftOfBank);
    hasSeatInterference = blockers.length > 0;
    hasMultipleSeatBlockers = blockers.length > 1;

    if (hasSeatInterference) {
        for (int i = 0; i < blockers.length; i++) {
            seatBank[blockers[i]] = STANDING_YIELDING_PASSENGER;
        }
        printState("Blockers stand", aisle, seatNames, seatBank);

        // Final Volume 2 update:
        // closest-to-aisle blocker yields farthest rearward (+blockers.length),
        // inner blocker yields nearer (+1). This creates space in the correct order.
        for (int i = 0; i < blockers.length; i++) {
            int blockerIndex = blockers[i];
            int rearwardOffset = blockers.length - i;
            int yieldTile = rowAisleIndex + rearwardOffset;
            aisle[yieldTile] = TEMPORARY_BLOCKER_IN_AISLE;
            seatBank[blockerIndex] = SEAT_EMPTY;
            System.out.println("Blocker from seat " + seatNames[blockerIndex] + " yields to aisle tile +" + rearwardOffset);
        }
        hasBlockingPassengerEnteredAisle = true;
        printState("Aisle temporarily reoccupied", aisle, seatNames, seatBank);
    }

    aisle[rowAisleIndex] = AISLE_EMPTY;
    seatBank[targetSeatIndex] = TARGET_PASSENGER_SEATED;
    hasTargetPassengerEnteredSeat = true;
    printState("Target passenger seated", aisle, seatNames, seatBank);

    // Reseat in reverse order: inner blocker first, closest-to-aisle blocker last.
    for (int i = blockers.length - 1; i >= 0; i--) {
        int blockerIndex = blockers[i];
        int rearwardOffset = blockers.length - i;
        int yieldTile = rowAisleIndex + rearwardOffset;
        aisle[yieldTile] = AISLE_EMPTY;
        seatBank[blockerIndex] = SEATED_PASSENGER;
        System.out.println("Blocker returns to seat " + seatNames[blockerIndex]);
    }
    hasBlockingPassengerReseated = true;
    hasCabinFlowResumed = true;
    printState("Cabin flow restored", aisle, seatNames, seatBank);
}

static int[] findBlockers(int[] seatBank, int targetSeatIndex, boolean aisleIsLeftOfBank) {
    int[] temp = new int[seatBank.length];
    int count = 0;
    if (aisleIsLeftOfBank) {
        for (int i = 0; i < targetSeatIndex; i++) if (seatBank[i] == SEATED_PASSENGER) temp[count++] = i;
    } else {
        for (int i = seatBank.length - 1; i > targetSeatIndex; i--) if (seatBank[i] == SEATED_PASSENGER) temp[count++] = i;
    }
    return Arrays.copyOf(temp, count);
}

static void printState(String label, int[] aisle, char[] names, int[] seats) {
    System.out.println(label);
    System.out.println("aisle = " + Arrays.toString(aisle));
    System.out.print("seats = ");
    for (int i=0;i<seats.length;i++) System.out.print(names[i] + ":" + seats[i] + " ");
```

```
        System.out.println();
    }
}
```

9. Operational Applications

Application	How Volume 2 helps
Boarding strategy evaluation	Counts and compares seat interference and temporary aisle reoccupation.
Aircraft layout analysis	Compares 3-3, 2-4-2, 3-3-3 and 3-4-3 layouts using the same movement engine.
Cabin crew training	Visualises where passengers must yield and where aisle conflict occurs.
Simulation research	Provides a deterministic, inspectable model before adding probabilistic or operational factors.

Publication Edition Statement

This final publication edition freezes Volume 2 as the advanced cabin-dynamics extension of Volume 1. It models local seat interference, per-aisle datasets, multiple blockers, temporary aisle reoccupation, reverse reseating and cabin flow recovery before take-off. It intentionally excludes in-flight movement, luggage handling, lavatory use, cabin service, arrival, deplaning and emergency procedures.