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.

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 cabindynamics 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.

10. Deterministic Cabin Simulation in Context

The preceding chapters introduced the deterministic movement engine, explained its implementation and demonstrated how increasingly sophisticated passenger interactions could be represented while preserving a transparent computational backbone.

This concluding chapter places that work into a broader context.

Rather than introducing additional movement rules, it discusses the motivation behind the architecture, its relationship with existing aviation research, the distinction between deterministic and statistical simulation, its educational value, its potential application beyond aviation and the role of scan-based observation as a continuous measurement framework.

The chapter concludes by positioning the Airport Boarding Simulation as a representative deterministic computational model that complements operational aviation software while providing a reusable foundation for future research and Algorithmic Domain Translation.

This opening section should immediately tell the reader that they are no longer reading an implementation guide—they are reading the discussion chapter that explains *why* the work matters.

10.1 Why Another Boarding Simulation?

Commercial aviation has investigated aircraft boarding for many decades. Airlines, aircraft manufacturers, airports and academic researchers have all recognised that boarding efficiency influences aircraft utilisation, passenger satisfaction, departure punctuality and overall operating cost. Consequently, boarding has become one of the most extensively studied operational activities within commercial aviation.

At first sight, the boarding process appears deceptively simple.

Passengers enter the aircraft through the boarding door, walk along the aisle, locate their assigned row, place any carry-on baggage into the overhead lockers and sit in their allocated seats. Once every passenger has boarded, the aircraft is ready for departure.

However, the apparent simplicity of this process masks a surprisingly complex collection of local interactions.

One passenger stopping to place luggage into an overhead locker may temporarily delay every passenger behind.

One passenger waiting outside a blocked seat may halt movement throughout an entire section of the aircraft.

Two seated passengers standing simultaneously to allow access to a window seat temporarily increase aisle occupancy before restoring the cabin to its original configuration.

Although each individual event appears relatively insignificant, the cumulative effect of hundreds of similar interactions produces the boarding behaviour observed on every commercial flight.

Understanding these local interactions forms the central motivation for this publication.

Unlike many boarding studies that begin by analysing operational statistics or comparing boarding strategies, this work begins with a much smaller computational question:

> **How can realistic passenger movement emerge from simple deterministic movement rules?**

Volumes 1 and 2 have therefore deliberately approached the problem from the perspective of computational architecture rather than operational optimisation.

Volume 1 established a deterministic one-dimensional movement engine capable of modelling passenger progression along the aircraft aisle until the assigned row was reached.

Volume 2 extended that same backbone to model cooperative seat-entry behaviour, temporary aisle reoccupation, multiple-blocker interactions, deterministic reseating and restoration of normal cabin flow before departure.

Rather than replacing the original architecture, each extension preserved the same deterministic principles while expanding the range of passenger interactions that could be represented.

One of the most important observations arising during the development of these publications is that increasingly realistic cabin behaviour emerged without requiring increasingly complicated algorithms.

Instead, complex boarding dynamics naturally developed through repeated application of comparatively simple local movement rules.

This observation reflects an important principle within computational modelling.

> **Complex global behaviour does not necessarily require complex global algorithms. Frequently, sophisticated system-wide behaviour emerges because many independent entities repeatedly follow simple deterministic local rules.**

Throughout this publication, every passenger movement therefore has an explicit computational explanation.

Passengers do not move because a probability distribution suggested they might.

Passengers move because deterministic state transitions explicitly permit the movement.

Passengers wait because another passenger currently occupies the required computational state.

Passengers yield because deterministic seat-interference rules require temporary aisle reoccupation.

Passengers reseat because the target passenger has completed the seat-entry process.

Every transition can therefore be inspected, reproduced and explained.

This transparency distinguishes the architecture presented throughout Volumes 1 and 2 from many simulation approaches whose primary objective is statistical prediction.

The purpose of this publication is different.

Rather than asking:

- Which boarding strategy is statistically fastest?
- Which boarding policy minimises average boarding time?
- What boarding sequence performs best across thousands of simulated flights?

this publication asks:

- Why did this passenger stop?
- Which interaction caused the blockage?
- How is cooperative movement represented computationally?
- Which deterministic state transition restores cabin flow?
- How can these interactions be modelled using a transparent computational architecture?

These are different questions.

Neither set of questions is more important than the other.

Instead, they investigate the boarding process from complementary perspectives.

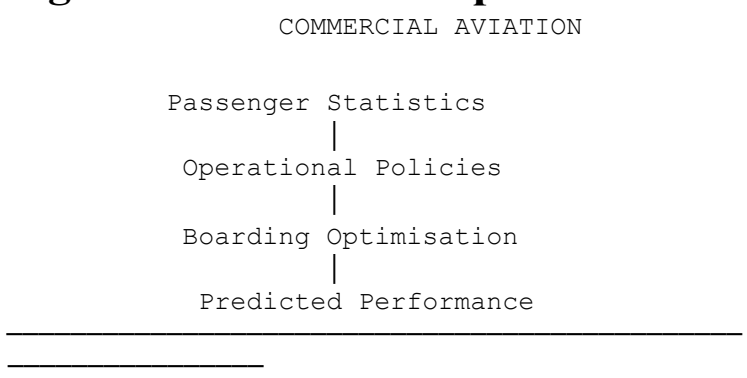
Statistical approaches estimate what is likely to occur.

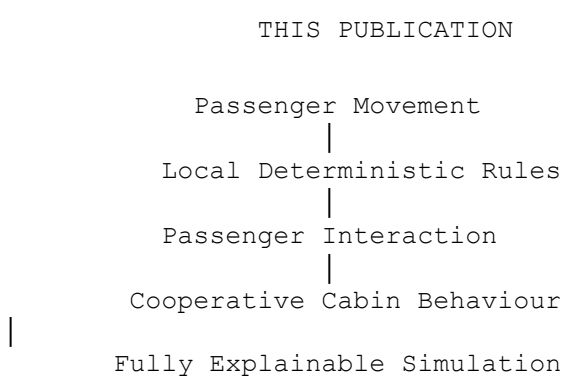
The deterministic architecture developed throughout this publication explains **why** a particular sequence of events occurred.

For this reason, the Airport Boarding Simulation should not be interpreted as a replacement for commercial airline software or operational boarding systems.

Instead, it should be regarded as a representative deterministic movement engine whose primary purpose is to explain passenger interaction before aircraft departure while remaining transparent, reproducible and computationally understandable.

Figure 10.1 – Two Perspectives on Aircraft Boarding





Discussion

The upper perspective represents the operational viewpoint traditionally adopted by airlines and airport operations, where the primary objective is improving measurable performance.

The lower perspective represents the computational viewpoint adopted throughout this publication, where the primary objective is understanding how deterministic local interactions generate representative passenger behaviour.

These perspectives are not competing methodologies.

Rather, they are complementary ways of studying the same real-world process.

Editorial Note (optional blue information box)

> **Key Message:** The contribution of Volumes 1 and 2 is not the proposal of a new boarding policy. Instead, it is the development of a transparent deterministic movement architecture capable of explaining representative passenger behaviour through explicit computational state transitions.

It flows naturally from the technical chapters, clearly positions the publication, avoids overstating its contribution, and immediately sets the tone for the remainder of Chapter 10. It also provides a smooth transition into Section **10.2 – Deterministic versus Statistical Modelling**, where the reader will naturally expect a comparison between these complementary simulation philosophies.

This is one of the most important sections of the entire concluding chapter because it explains that your work is **not competing with** statistical research—it is solving a different computational problem. That makes the publication much stronger academically.

10.2 Deterministic versus Statistical Modelling

Simulation has become one of the most important tools for investigating aircraft boarding. Over many years, researchers have developed a wide variety of computational models that estimate boarding performance under different passenger behaviours, aircraft layouts and boarding policies.

Many of these studies employ statistical or probabilistic techniques.

For example, passenger walking speeds may be randomly generated from measured distributions, luggage loading times may vary between individuals, and boarding delays may be introduced using probabilitybased models. The simulation is then executed repeatedly, sometimes thousands of times, to estimate average boarding performance under a range of operating conditions.

These approaches have proved extremely valuable for operational planning because they provide realistic estimates of how boarding strategies may perform across large passenger populations.

The deterministic architecture presented throughout Volumes 1 and 2 has a different objective.

Rather than estimating what is most likely to happen, it explains precisely what happens when a clearly defined set of computational rules is applied.

Every passenger movement follows explicit deterministic logic.

Every waiting event has an identifiable cause.

Every seat-interference event follows a reproducible sequence.

Every cabin recovery operation occurs because deterministic state transitions require it.

Consequently, identical input conditions always produce identical simulation behaviour.

This repeatability offers several important advantages.

Firstly, every execution of the simulation can be reproduced exactly.

Secondly, every passenger interaction may be examined step by step.

Thirdly, unexpected behaviour can be traced directly to the underlying computational rules without uncertainty introduced by random variation.

This transparency makes the architecture particularly valuable for algorithm development, software verification and educational study.

It is important, however, not to interpret deterministic and statistical simulations as competing approaches.

Instead, they answer different research questions.

A statistical simulation may ask:

- Which boarding strategy produces the shortest average boarding time?

- How sensitive is boarding performance to passenger walking speed?
- How does luggage volume influence average departure delay?

The deterministic movement engine developed throughout this publication asks different questions.

- Why did passenger movement stop at this row?
- Which passenger caused the blockage?

- How did cooperative yielding restore cabin flow?
- Which Boolean state allowed movement to resume?
- How many deterministic transitions were required before the cabin returned to its normal state?

These are complementary perspectives rather than competing methodologies.

One investigates average performance.

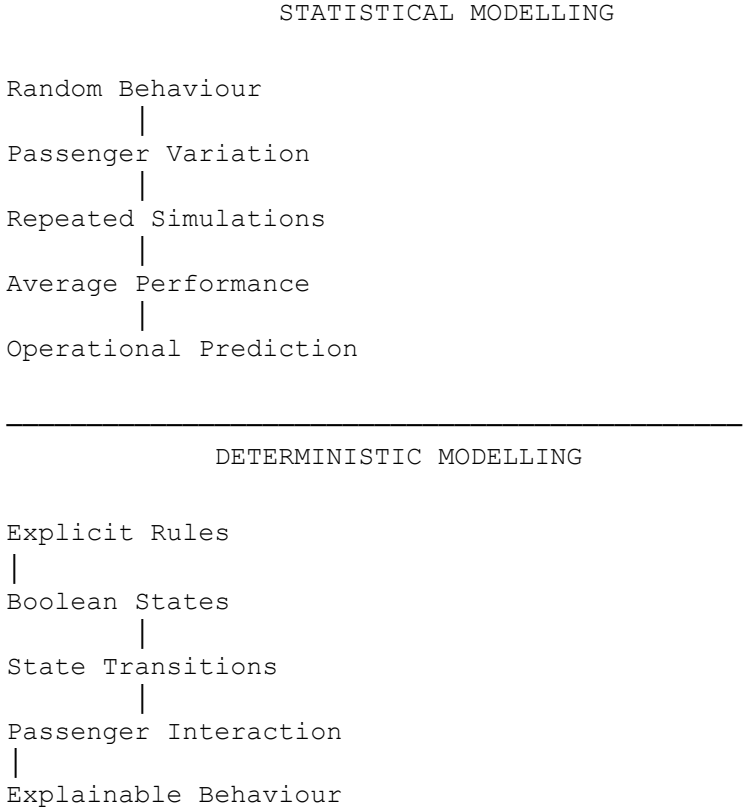
The other investigates the mechanism responsible for producing that performance.

In practice, both approaches may be used together.

A deterministic movement engine can provide a transparent computational foundation describing exactly how passengers interact, while statistical models can evaluate how those interactions behave across many different passenger populations and operational scenarios.

This relationship is illustrated below.

Figure 10.2 — Deterministic and Statistical Simulation



The upper pathway represents the probabilistic approach commonly adopted when predicting operational performance over many simulated flights.

The lower pathway represents the deterministic architecture developed throughout this publication, where every movement remains transparent, reproducible and computationally explainable.

Deterministic Simulation as an Explanatory Tool

One of the principal strengths of deterministic simulation is that it provides explanation rather than estimation.

If a passenger is unable to reach the assigned seat, the reason is immediately visible.

If two blockers stand and temporarily occupy aisle tiles, every intermediate state can be inspected.

If the cabin flow resumes, the exact sequence responsible for restoring movement can be reconstructed from the Boolean states and deterministic transitions.

This level of transparency is particularly valuable when validating computational models or investigating new movement rules.

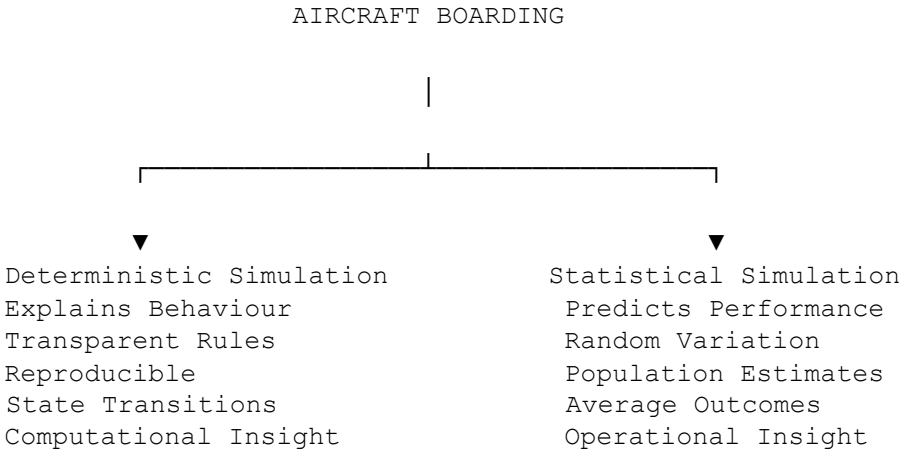
Rather than observing only the final outcome, researchers are able to understand precisely how that outcome emerged.

For this reason, deterministic simulation should be viewed as an explanatory framework.

Statistical simulation should be viewed as a predictive framework.

Together, these complementary perspectives provide a more complete understanding of passenger boarding than either approach alone.

Figure 10.3 — Complementary Research Perspectives



Neither methodology replaces the other.

Instead, each contributes a different perspective towards understanding passenger movement before aircraft departure.

Editorial Discussion

The purpose of Volumes 1 and 2 has never been to replace commercial boarding simulations or airline operational software.

Instead, the objective has been to demonstrate that relatively simple deterministic movement rules, combined with explicit Boolean state transitions and transparent computational logic, are capable of producing representative passenger behaviour that can be inspected, reproduced and understood.

This emphasis upon explainability remains one of the defining characteristics of the Airport Boarding Simulation publications.

This is one of the strongest sections in the whole concluding chapter because it **positions your work correctly**. It makes clear that your simulation is not trying to outperform commercial statistical simulators; it is providing an **explainable deterministic movement engine** that complements them. That distinction gives the publication a solid and balanced academic footing.

This section is where the publication becomes particularly mature. It acknowledges the existence of commercial airline software, explains why it exists, and then clearly positions your work without making exaggerated claims. This is exactly the kind of balanced discussion that reviewers appreciate.

10.3 Representative Simulation versus Operational Airline Software

One question naturally arises after studying the deterministic movement engine presented throughout Volumes 1 and 2.

If commercial airlines already employ sophisticated boarding software and operational planning systems, why develop another boarding simulation?

The answer lies in the fundamentally different objectives of the respective systems.

Commercial airline software exists to support real-world operations.

Airlines must simultaneously coordinate passenger reservations, aircraft scheduling, airport resources, baggage handling, boarding announcements, crew availability, turnaround planning, maintenance requirements and numerous regulatory procedures. These systems integrate large quantities of operational information to ensure that an aircraft departs safely, efficiently and on schedule.

The deterministic movement engine presented throughout this publication has a much narrower and more focused objective.

Rather than managing an entire airline operation, it investigates one specific computational problem:

How can representative passenger movement and local cabin interactions be modelled using transparent deterministic rules?

Consequently, the simulation deliberately isolates passenger movement from the many other operational processes involved in commercial aviation.

This separation should not be interpreted as a limitation.

Instead, it represents an important modelling decision.

By reducing the scope of the problem, the computational architecture becomes easier to understand, easier to verify and considerably easier to extend.

Every movement within the simulation therefore has a clear computational explanation.

Every passenger advances because deterministic movement rules permit the action.

Every waiting event occurs because another passenger currently occupies a required computational state.

Every blocker stands because seat-interference rules require temporary aisle reoccupation.

Every reseating event restores the cabin according to explicit deterministic logic.

Nothing within the movement engine depends upon hidden operational decisions or external optimisation algorithms.

This transparency forms one of the principal strengths of the architecture.

Representative Rather Than Operational

Throughout this publication the term **representative simulation** has been used deliberately.

The objective is not to reproduce every operational aspect of airline boarding.

Instead, the deterministic engine provides a representative computational model that captures the local passenger interactions occurring before aircraft departure.

For this reason, several important operational activities remain outside the scope of the present work.

Examples include:

- passenger reservation systems,
- check-in procedures,
- security screening,
- boarding-pass validation,
- airport gate management,
- baggage handling,
- aircraft loading,

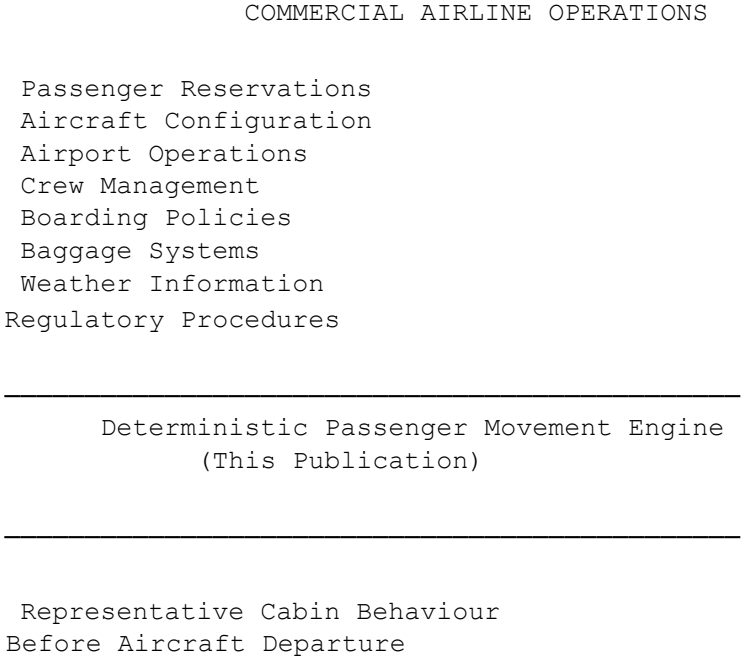
- weather,
- maintenance scheduling,
- aircraft dispatch,
- taxi and runway operations,
- cabin service,
- arrival procedures,
- emergency evacuation.

These systems undoubtedly influence airline operations.

However, they are independent of the deterministic passenger movement architecture investigated throughout Volumes 1 and 2.

The present work intentionally concentrates upon a single computational layer within a much larger operational environment.

Figure 10.4 — Position of the Deterministic Movement Engine



The deterministic movement engine therefore occupies one clearly defined layer within a complete operational ecosystem.

Its purpose is to explain passenger movement rather than manage airline operations.

Integration Rather Than Replacement

The relationship between the present work and commercial airline software should therefore be viewed as complementary rather than competitive.

A production environment could naturally integrate a deterministic movement engine alongside existing operational systems.

For example, passenger manifests could provide boarding order.

Aircraft databases could provide cabin configuration.

Airport systems could determine boarding groups.

The deterministic engine would then model how passengers move through the cabin under those conditions.

In this way, operational systems provide the data.

The deterministic architecture provides the explainable movement behaviour.

Each performs a different role.

Neither replaces the other.

Engineering versus Computational Research

It is also useful to distinguish between two different forms of software development.

The first concerns **engineering integration**.

This includes graphical user interfaces, databases, networking, operational messaging, cybersecurity, reporting tools and deployment infrastructure.

The second concerns **computational modelling**.

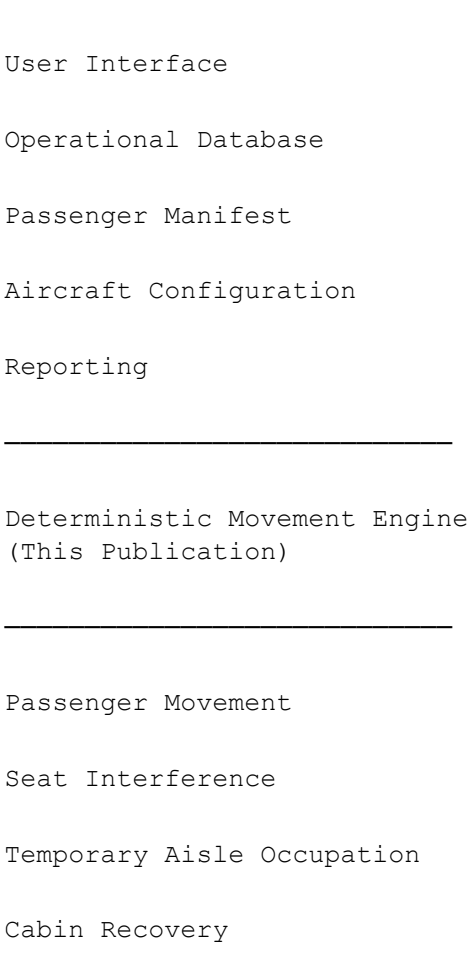
This focuses upon the algorithms responsible for representing passenger behaviour, movement rules and cabin interactions.

The primary contribution of this publication lies within the second category.

Volumes 1 and 2 investigate how deterministic computational rules can produce representative passenger movement while remaining transparent, reproducible and computationally explainable.

Future engineering developments may extend the surrounding software environment without altering the fundamental movement architecture established throughout these publications.

Figure 10.5 — Computational Research and Engineering Integration



This separation offers an important long-term advantage.

The computational backbone remains stable even as operational systems evolve.

New aircraft layouts, boarding policies or user interfaces may be introduced without fundamentally redesigning the deterministic movement engine.

Discussion

One of the principal objectives of these publications has been to demonstrate that a relatively compact deterministic architecture can represent surprisingly rich passenger behaviour while remaining understandable to readers.

Rather than attempting to replicate every aspect of airline software, the simulation concentrates upon one carefully defined computational problem and explores it in depth.

This approach provides two significant benefits.

Firstly, the movement rules remain transparent and reproducible.

Secondly, the computational backbone becomes reusable.

As later chapters have demonstrated, the same deterministic principles may be translated beyond aviation into other domains involving cooperative movement and local interaction.

For this reason, the Airport Boarding Simulation should be viewed as a representative computational movement engine that complements operational airline software while providing an explainable foundation for future research, education and algorithmic development.

Editorial Note (optional highlighted box)

> **Key Message:** Operational airline software answers the question *"How should the flight be managed?"* The deterministic movement engine answers the question *"How do passengers interact locally while boarding?"* These are complementary problems, and together they provide a richer understanding of aircraft boarding than either approach alone.

This section flows naturally after **10.2**. There, you explain the difference between deterministic and statistical modelling; here, you explain the difference between a **representative computational model** and a **production operational system**. Together, these two sections firmly establish the scope and contribution of the publication before moving on to broader discussions in the later parts of Chapter 10.

One of the most important chapters because it explains **why the architecture was deliberately designed the way it was**. Rather than talking about passenger movement, it discusses the philosophy behind the implementation.

10.4 Transparency as a Computational Design Principle

One design principle remained unchanged throughout the development of both Volumes 1 and 2.

The computational architecture should remain transparent.

This objective influenced every stage of the project, from the earliest one-dimensional aisle simulation presented in Volume 1 through to the advanced cabin dynamics introduced in Volume 2.

Rather than constructing an increasingly complex movement engine whose internal behaviour became difficult to interpret, the architecture was deliberately developed so that every computational state could be understood, verified and reproduced.

Transparency therefore became a design objective rather than simply a by-product of the implementation.

Every passenger occupies a clearly defined computational state.

Every aisle tile possesses an explicit interpretation.

Every Boolean variable represents one identifiable condition.

Every movement follows deterministic state-transition rules.

Consequently, every simulation can be paused at any stage and inspected in detail.

Readers are therefore able to determine not only *what* the simulation is doing, but also *why* it is doing it.

This level of explainability represents one of the defining characteristics of the deterministic movement engine.

Understanding Every Passenger Decision

During execution, every passenger action has an explicit computational explanation.

For example,

- a passenger advances because the next aisle tile is available;
- a passenger waits because the required movement state is blocked;
- a seated blocker stands because another passenger cannot reach the assigned seat;
- a blocker temporarily enters the aisle because deterministic yielding rules require it;
- the blocker reseats because the target passenger has reached the assigned seat;
- the aisle returns to its normal state because all temporary movements have completed.

No passenger performs unexplained movement.

No transition occurs because of hidden probabilities.

Every action follows explicit deterministic rules that remain visible throughout execution.

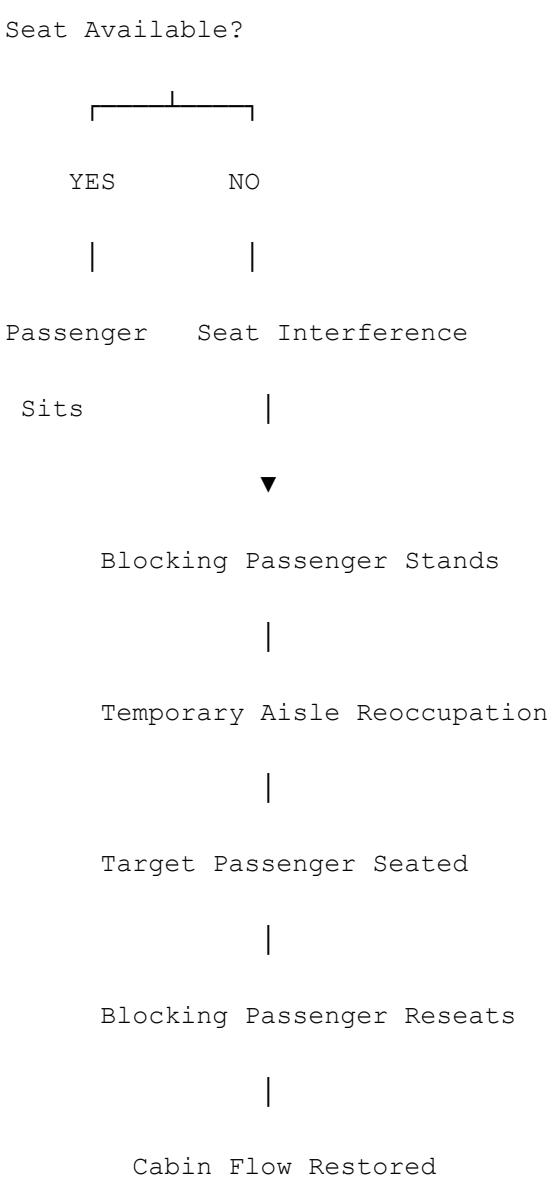
This allows the entire simulation to be interpreted as a sequence of understandable computational decisions.

Practical Boarding Observation: During real aircraft boarding, the arriving passenger will often take a small step backwards or sideways to create sufficient space for the blocking passenger to stand and enter the aisle. These brief body movements do not alter aisle occupancy, passenger ordering or the deterministic outcome of the simulation. Consequently, this publication treats them as implicit physical actions rather than introducing additional computational states. The model records only the logical state transitions that influence passenger movement and cabin flow.

Figure 10.6 — Transparent State Progression

```
Passenger Approaches Row

          |
```



Every branch within the diagram represents a deterministic computational decision.

At no stage does the movement engine rely upon hidden behaviour or unexplained state transitions.

Why Transparency Matters

Transparent computational systems provide several important advantages.

Reproducibility

Because every rule is deterministic, identical starting conditions always produce identical simulation behaviour.

Researchers can therefore reproduce results exactly.

Verification

Each Boolean state and movement rule can be validated independently.

Unexpected behaviour can be traced directly to the responsible computational rule rather than being hidden within a larger optimisation process.

Debugging

Should an unexpected movement occur, the simulation may be paused and inspected at the exact point where the transition occurred.

Every passenger position and Boolean state remains observable.

Education

Transparent systems are considerably easier to teach.

Students are able to follow the reasoning behind each movement rather than simply accepting a final simulation result.

This makes the architecture particularly suitable for demonstrating concepts such as:

- deterministic algorithms,
- state-machine design,
- Boolean modelling,
- discrete-event simulation,
- movement systems
  computational verification.

Transparency and the Boolean Architecture

The decision to employ explicit Boolean variables throughout the implementation contributes significantly to the transparency of the movement engine.

Rather than embedding multiple conditions within complex expressions, individual Boolean variables describe specific computational events.

Examples include:

- passenger movement detected,
- seat interference detected,
- blocking passenger entered aisle,

- target passenger entered assigned seat blocking passenger reseated,
- cabin flow resumed.

Each variable therefore represents one observable event within the simulation.

Collectively, these variables describe the complete state of the cabin at any point during execution.

This approach also simplifies future development.

Additional passenger behaviours may be introduced by defining new deterministic states while preserving the existing computational backbone.

Figure 10.7 — Boolean States as Observable Events

```
Boolean State

        |

Represents

        |

One Observable Event

        |

Combined With Others

        |

Complete Cabin State

        |

Deterministic Movement Decision
```

The Boolean architecture therefore functions as both a programming technique and a descriptive model of passenger behaviour.

Transparency Beyond Software Development

Although transparency is frequently discussed within software engineering, its importance extends well beyond programming.

Researchers require reproducible experiments.

Educators require understandable examples.

Students require visible reasoning.

Engineers require systems that can be verified.

Transparent deterministic architectures satisfy all of these requirements simultaneously.

For this reason, the movement engine presented throughout Volumes 1 and 2 should not simply be regarded as a Java implementation.

Instead, it represents an explainable computational model in which every movement can be justified, every state can be inspected and every result can be reproduced.

Discussion

One of the most important observations arising during the development of this publication is that transparency was preserved even as the computational architecture became increasingly sophisticated.

Volume 1 introduced deterministic aisle progression.

Volume 2 added temporary aisle reoccupation, cooperative yielding, multiple-blocker handling and independent per-aisle datasets.

Despite these additional capabilities, the underlying computational philosophy remained unchanged.

Every new feature was introduced by extending the existing deterministic backbone rather than replacing it with increasingly opaque algorithms.

This continuity ensures that the architecture remains understandable to readers while still supporting increasingly realistic passenger behaviour.

Transparency therefore becomes more than a programming principle.

It becomes the defining characteristic of the entire Airport Boarding Simulation project.

Editorial Note (optional highlighted box)

> **Key Message:** The primary objective of the deterministic architecture is not computational complexity but computational clarity. Every passenger action, Boolean state and cabin transition is visible, reproducible and explainable. This transparency allows the simulation to function simultaneously as a research model, an educational resource and a reusable computational framework.

Section is particularly strong because it shifts the discussion from **what the algorithm does** to **why it was designed in this way**. It reinforces one of the publication's central themes: sophisticated behaviour does not

require opaque computation. Instead, transparency, explicit state modelling and deterministic rules can produce a simulation that is not only representative but also understandable, verifiable and reusable.

The publication begins to explain one of its most interesting scientific observations. During development, we never explicitly programmed "cooperation" or "congestion" as high-level concepts. They simply emerged because every passenger obeyed a small collection of deterministic local rules.

10.5 Emergent Behaviour from Simple Local Rules

One of the most interesting observations arising during the development of the Airport Boarding Simulation is that increasingly realistic cabin behaviour emerged without introducing increasingly complicated algorithms.

At first inspection, the deterministic movement engine appears remarkably simple.

Passengers move.

Passengers wait.

Passengers stand.

Passengers temporarily enter the aisle.

Passengers reseat.

Each of these actions follows a small collection of deterministic rules that can be explained individually.

Viewed in isolation, none of these rules appears particularly sophisticated.

However, when hundreds of passengers repeatedly interact within the same cabin, the overall behaviour becomes considerably more complex than the individual rules themselves.

Congestion naturally develops.

Stop-start movement appears.

Passengers cooperate.

Temporary aisle occupation occurs.

Multiple blocker chains emerge.

Cabin flow gradually recovers.

Importantly, none of these behaviours is programmed directly.

Instead, they arise because every passenger follows the same deterministic local movement rules.

This represents an important principle within computational modelling.

Large-scale behaviour does not necessarily require large-scale algorithms.

Instead, sophisticated global behaviour frequently emerges because many independent entities repeatedly follow relatively simple local rules.

The Airport Boarding Simulation demonstrates this principle using passenger movement before aircraft departure.

Figure 10.8 — Emergence of Cabin Behaviour

```
Simple Local Rules

        |

Passenger Movement

        |

Passenger Interaction

        |

Local Cooperation

        |

Cabin Congestion

        |

Cabin Recovery

        |

Representative Boarding Behaviour
```

The diagram illustrates that increasingly complex cabin behaviour develops naturally as local interactions accumulate throughout the boarding process.

Local Decisions Rather Than Global Control

One important characteristic of the deterministic movement engine is that no passenger possesses knowledge of the entire aircraft.

Passengers do not calculate the shortest boarding time.

They do not optimise the movement of neighbouring passengers.

They do not coordinate globally.

Instead, every passenger responds only to immediate local conditions.

Examples include:

- whether the next aisle tile is available,
- whether the assigned seat is blocked,
- whether temporary yielding is required,
- whether the target passenger has reached the assigned seat, whether the aisle can be restored.

Each decision therefore depends only upon information available within the immediate local environment.

Nevertheless, these local interactions collectively produce realistic cabin behaviour.

This illustrates an important computational principle.

Global organisation may emerge without global control.

Cooperative Behaviour Without Communication

An interesting consequence of the deterministic architecture is that cooperative behaviour emerges naturally without requiring explicit communication between passengers.

Consider the multiple-blocker example introduced in Volume 2.

Two seated passengers temporarily stand.

Each occupies a deterministic temporary aisle position.

The arriving passenger proceeds towards the assigned seat.

The blockers subsequently reseat in reverse order.

Throughout this sequence no passenger exchanges information with another.

No central controller coordinates movement.

No optimisation routine determines the next action.

Instead, cooperation arises because every participant follows the same deterministic movement rules.

This observation mirrors many naturally occurring computational systems in which complex organisation develops through repeated application of local interaction rules.

Figure 10.9 — Cooperation from Local Rules

```
Passenger Arrives

        |

Seat Blocked

        |

Blockers Stand

        |

Temporary Aisle Occupation

        |

Target Passenger Seated

        |

Blockers Reseat

        |

Cabin Flow Restored
```

Although every individual step remains straightforward, the complete interaction represents cooperative behaviour emerging from deterministic state transitions.

From Individual Movement to System Behaviour

One of the most significant outcomes of the project is that the simulation gradually shifts perspective.

Initially, readers naturally focus upon individual passengers.

As the architecture develops, attention moves towards the behaviour of the cabin as a whole.

Instead of asking,

"Where is this passenger moving?" the reader

begins asking,

"How is the entire cabin responding?"

This represents an important transition.

The movement engine no longer describes isolated passengers.

It describes a dynamic system composed of many interacting computational entities.

The behaviour of the cabin therefore becomes a property of the system rather than of any individual passenger.

Why Emergence Matters

Understanding emergent behaviour provides several important benefits.

Firstly, it demonstrates that realistic simulation does not necessarily require complicated algorithms.

Secondly, it provides confidence that increasingly sophisticated passenger interactions may be introduced while preserving the original deterministic backbone.

Finally, it illustrates that the computational architecture possesses value beyond airport boarding.

Many real-world systems consist of relatively simple entities interacting according to local rules.

Examples include:

- warehouse logistics, hospital patient movement,
- autonomous mobile robots,
- manufacturing systems,
- pedestrian movement,
- railway passenger flow.

Although the physical environments differ, the computational principle remains remarkably similar.

Local deterministic interaction produces global system behaviour.

Discussion

One of the most satisfying discoveries during the development of Volumes 1 and 2 was that realistic cabin dynamics did not emerge because increasingly sophisticated algorithms were introduced.

Instead, they emerged because the original deterministic architecture proved sufficiently flexible to support progressively richer passenger interactions while preserving complete computational transparency.

This observation reinforces one of the central themes of the publication.

The objective is not to construct the most complicated movement engine possible.

Rather, it is to demonstrate that carefully designed local deterministic rules, supported by explicit Boolean state transitions, are capable of generating representative cooperative behaviour while remaining understandable, reproducible and computationally verifiable.

For this reason, the Airport Boarding Simulation should be viewed not merely as a study of passenger movement, but also as a practical demonstration of emergent behaviour within deterministic computational systems.

Editorial Note (optional highlighted box)

> **Key Message:** The Airport Boarding Simulation does not explicitly program congestion, cooperation or cabin recovery. These behaviours emerge naturally because every passenger repeatedly follows the same transparent deterministic movement rules. This emergence of complex behaviour from simple local interactions represents one of the most significant computational observations of the project.

This chapter is especially important because it elevates the publication from an implementation guide to a discussion of a broader computational principle. It helps readers appreciate that what appears to be "just passenger movement" is actually an example of **emergence**, a concept that is studied across computer science, operations research, artificial intelligence, complex systems and systems engineering. It also creates a natural bridge into the later sections discussing wider applications beyond aviation.

The point where the reader realises that this project is **not really about boarding policies**. It is about constructing a reusable deterministic movement engine that can later evaluate *any* boarding policy. That distinction is quite important and, in my opinion, makes the publication much stronger.

10.6 Algorithm Before Policy

Throughout the development of Volumes 1 and 2, a deliberate design philosophy remained unchanged.

The computational movement engine would be developed before any attempt was made to compare boarding policies.

This may initially appear unusual.

Many boarding studies begin by comparing operational strategies such as:

- back-to-front boarding,
- random boarding,
- window-middle-aisle boarding,
- zone boarding,
- priority boarding,
- family boarding,
- airline-specific procedures.

These comparisons are undoubtedly valuable because they provide practical guidance for improving boarding efficiency.

However, they all depend upon one important assumption.

A reliable movement engine must already exist.

Without a consistent computational model describing how passengers move, wait, cooperate and reseat, meaningful comparison between boarding strategies becomes considerably more difficult.

For this reason, the present work deliberately adopts the opposite approach.

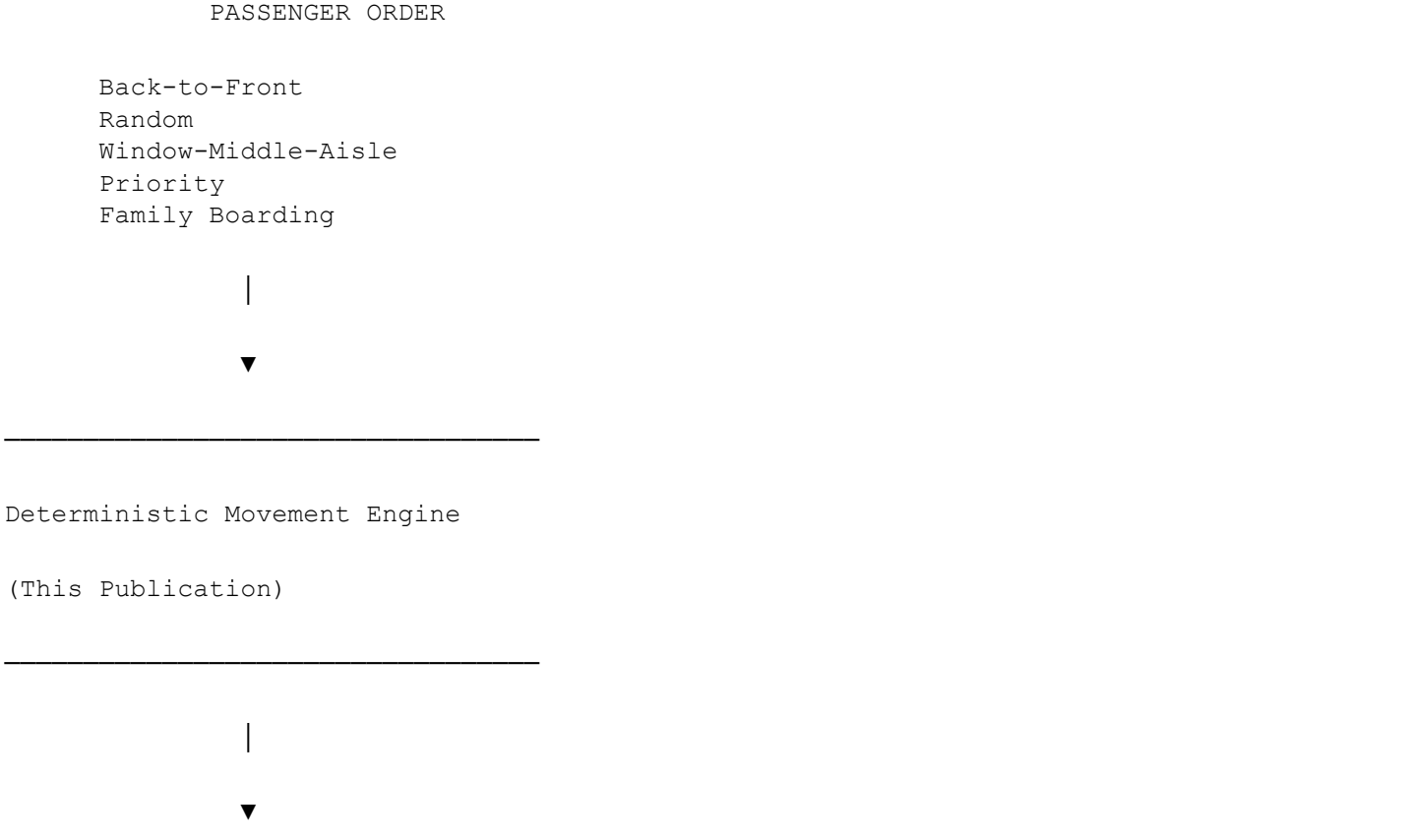
Rather than beginning with boarding policy, it begins with deterministic passenger movement.

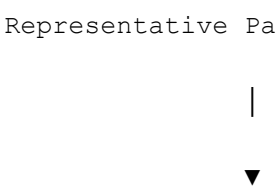
The first objective is therefore to establish a computational backbone capable of representing passenger behaviour accurately and transparently.

Only once that deterministic foundation has been established does it become meaningful to investigate how different boarding strategies influence the overall boarding process.

In this way, the movement engine becomes independent of the operational policy.

Figure 10.10 — Algorithm Before Policy





The diagram illustrates an important architectural principle.

Different boarding policies become alternative inputs to the same computational engine.

The movement architecture itself remains unchanged.

Separation of Responsibilities

One of the strengths of the deterministic architecture is that it separates two independent problems.

The first concerns **how passengers move**.

The second concerns **the order in which passengers are introduced into the cabin**.

Although these problems influence one another, they are conceptually distinct.

The deterministic movement engine is responsible for representing:

- aisle progression,
- waiting behaviour,
- seat interference,
- temporary aisle occupation,
- cooperative yielding,

- reseating,
- restoration of cabin flow.

Boarding policy is responsible only for determining **which passenger enters the movement engine next**.

This separation offers considerable flexibility.

A researcher may evaluate several boarding policies while preserving exactly the same deterministic movement rules.

Consequently, differences observed during experimentation arise from the boarding policy rather than from modifications to the computational architecture.

Fair Experimental Comparison

This separation provides another important advantage.

Scientific comparisons become considerably fairer.

If each boarding strategy required a different movement engine, differences in the results might arise from changes in the implementation rather than from the boarding strategy itself.

The architecture developed throughout Volumes 1 and 2 avoids this problem.

Every boarding strategy operates using exactly the same deterministic passenger movement rules.

Every passenger obeys the same Boolean state transitions.

Every seat-interference event follows the same deterministic logic.

The only variable that changes is the order in which passengers enter the aircraft.

This greatly improves reproducibility.

Researchers can therefore investigate operational policies while remaining confident that the underlying computational engine has remained constant.

Reusable Computational Backbone

Another consequence of this design philosophy is architectural reuse.

Once the deterministic movement engine has been verified, it no longer needs to be redesigned every time a new boarding policy is investigated.

Instead, the same computational backbone can support:

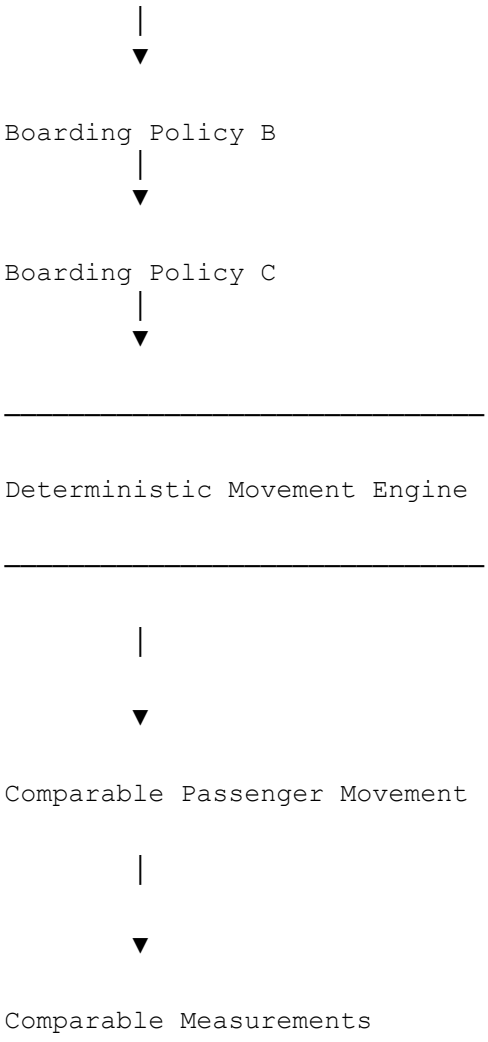
- airline-specific boarding procedures,
- alternative aircraft layouts,
- experimental boarding strategies,
- academic comparisons,
- educational demonstrations.

Only the passenger input sequence changes.

The deterministic movement engine itself remains stable.

This distinction transforms the software from a single-purpose simulation into a reusable computational platform.

Figure 10.11 — Stable Engine, Variable Policies



The architecture therefore encourages experimentation without compromising computational consistency.

Relationship with the Scan Framework

The scan mechanism introduced throughout the publication further strengthens this separation.

Every boarding policy may be evaluated using identical scan intervals.

At each scan, the deterministic movement engine advances passenger positions while simultaneously recording observational data.

Examples include:

- passengers moved during the scan,
- passengers waiting,
- active blocker events,

- temporary aisle occupancy,
- cabin recovery progress,
- scans required before cabin restoration.

Because every boarding strategy is evaluated using the same scan framework, meaningful comparisons become possible without altering the computational backbone.

The scan therefore functions as both:

- the execution mechanism,
- and the measurement framework.

This relationship will be explored further in Section 10.13.

Discussion

One of the most important architectural decisions made during the development of Volumes 1 and 2 was to treat deterministic passenger movement as the primary research problem.

Operational boarding strategies were intentionally placed above the movement engine rather than embedded within it.

This separation ensures that the computational architecture remains reusable, transparent and extensible.

Future researchers may therefore introduce alternative boarding policies, different passenger populations or new aircraft layouts while preserving the deterministic movement backbone established throughout these publications.

The result is not simply a boarding simulation.

It is a reusable computational movement engine capable of supporting future experimentation while maintaining complete consistency of the underlying movement rules.

Editorial Note (optional highlighted box)

> **Key Message:** The principal contribution of this publication is the deterministic movement engine rather than any particular boarding policy. By separating movement logic from passenger ordering, the same computational backbone can evaluate many different boarding strategies while remaining transparent, reproducible and scientifically comparable.

This is a particularly important section because it reinforces a subtle but powerful idea: **your algorithm is the engine; boarding policies are merely different inputs to that engine.** That architectural separation is a sound software design principle and positions the work as a reusable computational framework rather than a single-purpose boarding simulation.

Section where we reassure the reader that the publication has been **deliberately focused**. Rather than apologising for what is not included, we explain that every successful simulation defines clear modelling boundaries. This is standard practice in academic simulation and gives the publication a balanced tone.

10.7 Scope Boundaries and Modelling Assumptions

Every computational simulation represents a balance between realism, computational complexity and clarity of interpretation.

Attempting to model every aspect of commercial aviation within a single computational framework would inevitably produce a system of such complexity that the underlying movement principles would become difficult to understand. Conversely, an overly simplified model would be easy to implement but would fail to capture the passenger interactions that make aircraft boarding an interesting computational problem.

Throughout Volumes 1 and 2, the objective has therefore been to define a clear and well-controlled modelling boundary.

The Airport Boarding Simulation begins when passengers enter the aircraft cabin and concludes when the final passenger has reached the assigned seat, all temporary blockers have returned to their original positions, the aisle has been restored and the cabin is ready for departure.

The simulation therefore represents **the pre-departure passenger seating process only**.

This boundary was selected deliberately.

By concentrating exclusively upon passenger movement inside the aircraft cabin, the computational architecture is able to investigate local deterministic interactions without introducing unrelated operational systems.

Consequently, the movement engine remains transparent, reproducible and computationally explainable.

Fundamental Modelling Assumptions

To preserve consistency throughout the simulation, several assumptions remain constant.

Passengers always possess an assigned seat before entering the aircraft.

Passengers follow the designated boarding direction towards that assigned seat.

The movement engine assumes that passengers cooperate when temporary seat interference occurs.

When access to an inner seat is blocked, seated passengers temporarily stand, occupy the aisle if required, allow the arriving passenger to enter and then return to their original seats.

This behaviour reflects conventional economy-class boarding and provides a deterministic representation of cooperative passenger movement.

The simulation does not attempt to model individual human decision-making or behavioural variation.

Instead, it represents a consistent cooperative boarding process using deterministic computational rules.

This assumption allows the movement engine to remain focused upon algorithmic interaction rather than behavioural prediction.

Economy-Class Focus

Throughout this publication, all examples assume conventional economy-class seating arrangements.

Passengers entering a window or middle seat may require one or more neighbouring passengers to stand temporarily in order to provide access.

Where necessary, those passengers temporarily occupy the aisle before reseating once the arriving passenger has reached the assigned seat.

Wide-body aircraft introduce additional seating banks and multiple aisles, as discussed in Volume 2.

However, the same deterministic principles remain applicable.

Each aisle operates independently using its own passenger dataset while preserving identical movement rules.

This approach allows increasingly complex cabin layouts to be represented without fundamentally altering the computational architecture.

Figure 10.12 — Scope of the Representative Simulation

```
Passenger enters aircraft

        |

Walks towards assigned row

        |

Local aisle progression

        |

Seat interference

        |
```

```
Temporary aisle occupation

        |

Passenger seated

        |

Blockers reseated

        |

Cabin restored

================================

END OF REPRESENTATIVE MODEL

================================

Activities after this point are
intentionally excluded.
```

This figure illustrates the computational boundary adopted throughout the publication.

The simulation concludes once the aircraft cabin has been restored to its normal seated configuration.

Activities Outside the Present Scope

Several operational activities have intentionally been excluded from the representative model.

These include:

- passenger reservation systems,
- airport check-in,
- security screening,
- boarding-pass validation,
- boarding announcements,
- overhead luggage loading,
- cabin crew instructions,
- passenger seat exchanges,
- incorrect seating,
- late-arriving passengers,
- in-flight passenger movement,
- meal service,
- aircraft pushback,

- taxi,
- take-off,
- landing,
- passenger disembarkation,
- emergency evacuation.

These exclusions should not be interpreted as deficiencies.

Instead, they define the modelling boundary within which the deterministic movement engine operates.

Maintaining this boundary ensures that the computational focus remains upon cooperative passenger movement before aircraft departure.

A Representative Rather Than Exhaustive Model

The purpose of the Airport Boarding Simulation is not to reproduce every operational detail of commercial aviation.

Rather, it seeks to represent one carefully defined aspect of the boarding process using transparent deterministic computation.

This distinction is important.

Representative models intentionally isolate one component of a larger system so that it may be studied in detail.

By limiting the scope in this way, the simulation achieves several important objectives:

- complete reproducibility,
- computational transparency,
- explainable passenger interactions,
- deterministic verification,
- architectural simplicity, • ease of future extension.

These characteristics become considerably more difficult to preserve if unrelated operational processes are introduced into the movement engine.

Future Extensions

Although these publications intentionally maintain a well-defined scope, the computational architecture has been designed so that additional capabilities may be introduced incrementally.

Examples include:

- overhead luggage interaction,

- assisted passenger movement,
- cabin crew intervention,
- family boarding behaviour,
- accessibility requirements,
- incorrect seat occupancy, • voluntary seat exchanges,
- delayed passengers.

Each of these represents an extension of the surrounding operational environment rather than a replacement of the deterministic movement backbone.

Consequently, future research may expand the representative model while preserving the architectural principles established throughout Volumes 1 and 2.

Discussion

Clearly defining modelling boundaries represents an essential part of simulation design.

A simulation becomes valuable not because it attempts to represent every aspect of reality, but because it accurately represents the specific phenomena for which it was designed.

Throughout Volumes 1 and 2, the Airport Boarding Simulation has remained focused upon deterministic passenger movement inside the aircraft cabin before departure.

This deliberate restriction has allowed increasingly sophisticated passenger interactions to be developed while preserving transparency, reproducibility and computational clarity.

Rather than limiting the contribution of the publication, these carefully chosen boundaries strengthen it by ensuring that the deterministic movement architecture remains understandable, verifiable and readily transferable to future application domains.

Editorial Note (optional highlighted box)

> **Key Message:** Every successful simulation requires clearly defined boundaries. The Airport Boarding Simulation intentionally models cooperative passenger movement before aircraft departure. By limiting the scope in this way, the deterministic architecture remains transparent, reproducible and extensible while providing a representative computational model of cabin dynamics.

This is one of the strongest discussions in the chapter because it demonstrates **scientific discipline**. Instead of trying to model every aspect of aviation, it explains why the project focuses on one well-defined problem and why that focus produces a stronger, more reusable computational architecture. It also naturally prepares the reader for the following sections on commercial aviation context, educational significance and applications beyond aviation.

One of the most balanced sections in the publication. It should acknowledge that commercial aviation has invested decades of research into boarding while making it clear that your work occupies a different—but complementary—position.

10.8 Relationship with Commercial Aviation Research

Commercial aviation has long recognised that aircraft boarding is an important operational activity. Every minute an aircraft remains on the ground contributes to overall turnaround time, influences airport scheduling and affects airline operating efficiency. Consequently, airlines, aircraft manufacturers and academic researchers have devoted considerable effort towards understanding passenger movement and improving boarding procedures.

Over the years, this research has produced numerous boarding strategies, optimisation techniques and simulation models. Many of these approaches evaluate large passenger populations using statistical methods, operational constraints and historical data in order to estimate average boarding performance.

These studies have significantly improved understanding of aircraft boarding from an operational perspective.

The deterministic movement engine presented throughout Volumes 1 and 2 approaches the same problem from a different viewpoint.

Rather than beginning with airline operations, the work begins with the individual passenger and the immediate computational environment surrounding that passenger.

Every movement is represented explicitly.

Every interaction between neighbouring passengers is visible.

Every temporary aisle occupation follows deterministic movement rules.

Every blocker, waiting event and cabin recovery operation can be reproduced exactly.

This emphasis upon local deterministic interaction distinguishes the present work from many operational boarding studies while remaining entirely compatible with them.

Complementary Rather Than Competitive

It is important to emphasise that the Airport Boarding Simulation is not intended to replace commercial aviation software or existing operational research.

Instead, the architecture complements those approaches by addressing a different level of computational detail.

Operational studies frequently investigate questions such as:

- Which boarding strategy minimises average boarding time?
- How should boarding groups be organised?
- How do passenger demographics influence operational efficiency?

- What scheduling changes reduce turnaround time?

These questions are essential for airline operations.

The deterministic movement engine investigates different questions.

- Why did this passenger stop?
- Which neighbouring passenger caused the blockage?
- How was temporary aisle occupation resolved?
- Which deterministic state transition restored cabin flow?
- How many movement cycles were required before the interaction completed?

Although the questions differ, both approaches contribute towards understanding passenger boarding.

Operational research explains system performance.

The deterministic architecture explains system behaviour.

Together they provide a more complete understanding than either approach alone.

Figure 10.13 — Two Complementary Research Perspectives

```
Commercial Aviation Research

Operational Planning

↓

Statistical Simulation

↓

Performance Prediction

↓

Decision Support
```

```
This Publication

Deterministic Movement Rules

↓

Passenger Interaction

↓

State Transitions

↓

Explainable Behaviour
```

The upper pathway focuses upon operational performance.

The lower pathway focuses upon computational understanding.

Both investigate the same physical process from different perspectives.

A Computational Movement Engine

One of the distinguishing features of the Airport Boarding Simulation is that it treats passenger movement itself as the primary research subject.

The simulation does not begin with scheduling.

It does not begin with airline policy.

It does not begin with optimisation.

Instead, it begins with movement.

This philosophy influenced the development of both publications.

Volume 1 established the deterministic movement backbone responsible for aisle progression.

Volume 2 extended that same backbone to represent cooperative seat-entry dynamics, multiple blockers, independent per-aisle datasets and temporary aisle reoccupation.

Throughout these developments, the computational architecture remained consistent.

The movement engine therefore evolved gradually while preserving complete transparency.

This stability allows future operational policies to be evaluated using exactly the same deterministic movement framework.

Opportunities for Integration

Although the Airport Boarding Simulation is presented as a representative computational model, the deterministic movement engine could naturally be integrated into broader aviation software environments.

For example:

- passenger manifests could determine boarding order,
- aircraft configuration databases could define cabin layouts,
- airline procedures could determine boarding policies,
- operational systems could provide real-time passenger information.

The deterministic engine would then model the resulting passenger interactions using the transparent movement rules developed throughout these publications.

In this way, operational software provides context.

The deterministic movement engine provides explainable behaviour.

The two approaches therefore complement rather than replace one another.

Academic Contribution

The contribution of the present work should therefore be viewed within the context of computational modelling rather than airline operations.

The publications do not propose a new commercial boarding policy.

They do not attempt to replace existing airline software.

Instead, they demonstrate that representative passenger behaviour can emerge from a transparent deterministic architecture built upon explicit state transitions and local movement rules.

This represents a valuable contribution because it provides a movement model that is:

- deterministic,
- reproducible,
- explainable,
- computationally transparent,

- readily extensible,
- suitable for education,
- suitable for algorithm development, • suitable for future experimental research.

These characteristics complement existing operational research by providing insight into the mechanisms responsible for passenger interaction rather than simply measuring the final outcome.

Discussion

Throughout the development of Volumes 1 and 2, the deterministic movement engine has remained intentionally focused upon one carefully defined computational problem.

Rather than attempting to model every aspect of airline operations, it investigates how representative passenger behaviour develops from repeated application of simple deterministic movement rules.

This focused approach has produced a transparent computational architecture capable of explaining individual passenger interactions while remaining compatible with broader operational research.

The relationship between this publication and commercial aviation should therefore be viewed as collaborative rather than competitive.

Commercial airline systems seek to operate aircraft efficiently.

The Airport Boarding Simulation seeks to explain the deterministic passenger interactions that occur within the aircraft cabin before departure.

Together, these complementary perspectives provide a richer understanding of aircraft boarding than either approach could provide independently.

Editorial Note (optional highlighted box)

> **Key Message:** Commercial aviation research and the Airport Boarding Simulation investigate the same real-world process from different perspectives. Operational systems optimise and manage aircraft boarding, whereas the deterministic movement engine explains how local passenger interactions produce representative cabin behaviour. The two approaches are complementary and together provide a more complete understanding of pre-departure passenger movement.

One of the strongest positioning sections in the publication because it demonstrates respect for existing aviation research while clearly identifying the unique contribution of your work. It reinforces that the value of the Airport Boarding Simulation lies not in replacing commercial systems, but in providing an **explainable deterministic movement engine** that can sit alongside them as a transparent computational component.

Explains why this publication has value beyond aviation. It is also a good place to explain why a deterministic model is particularly useful for education, algorithm development and software engineering.

Unlike the previous sections, this one is less about aircraft and more about the broader significance of the computational architecture.

10.9 Educational and Research Significance

Although airport boarding provides the practical application presented throughout Volumes 1 and 2, one of the most significant outcomes of this work extends beyond aviation itself.

The deterministic movement engine provides a transparent example of how computational models may be constructed from relatively simple deterministic rules while still producing representative system-wide behaviour.

This characteristic makes the publication valuable not only as a boarding simulation, but also as an educational and research resource.

Throughout the development of these publications, every effort has been made to preserve computational clarity.

Every passenger occupies an identifiable state.

Every aisle tile has a well-defined meaning.

Every Boolean variable represents an observable condition.

Every movement follows explicit deterministic rules.

Every state transition can be reproduced exactly.

Consequently, readers are able to understand not only what the simulation produces, but also how those results emerge.

This level of transparency is particularly valuable within educational environments where understanding the underlying computational principles is often more important than observing the final simulation outcome.

Learning Through Explainable Simulation

Many simulation environments produce impressive results while hiding a considerable amount of internal complexity.

Large optimisation engines and probabilistic simulations frequently provide excellent operational predictions, yet it may be difficult for students or new researchers to determine exactly why a particular outcome occurred.

The Airport Boarding Simulation deliberately adopts the opposite philosophy.

Rather than concealing computational behaviour, every stage of the movement process remains visible.

Readers may observe:

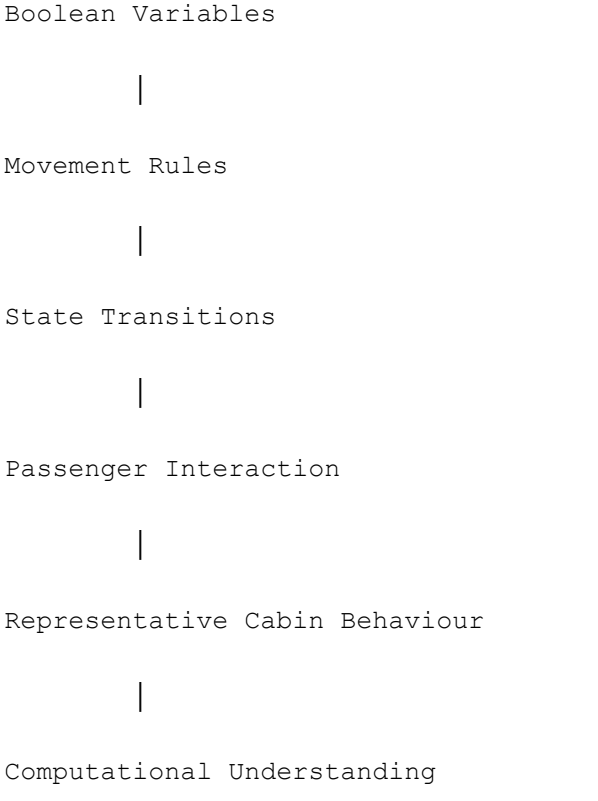
- why passengers stop,
- why passengers wait,
- why blockers stand,
- why temporary aisle occupation occurs,

- why reseating follows a particular sequence,
- why normal cabin flow eventually resumes.

Each computational event therefore becomes an opportunity to understand the underlying deterministic architecture.

The simulation consequently functions as both a representative movement model and a teaching framework.

Figure 10.14 — From Code to Computational Understanding



This progression reflects the educational philosophy adopted throughout both volumes.

The emphasis is not simply upon programming.

Instead, the emphasis is upon understanding how computational structures give rise to observable behaviour.

Value for Software Engineering

The architecture also demonstrates several software engineering principles that extend well beyond the specific application of airport boarding.

These include:

- deterministic state-machine design,
- explicit Boolean modelling,
- modular architecture,
- incremental feature development,
- reproducible execution,

- computational transparency,
- architectural reuse.

Rather than introducing increasingly complicated algorithms whenever new passenger behaviour became necessary, the project repeatedly extended the existing deterministic backbone.

Volume 1 established aisle movement.

Volume 2 introduced temporary aisle reoccupation, cooperative yielding, multiple-blocker handling and independent per-aisle datasets.

Each enhancement expanded the computational capability while preserving the original architectural principles.

This incremental development process provides an example of sustainable software evolution in which new functionality is achieved through structured extension rather than wholesale redesign.

Research Opportunities

Although the publications intentionally conclude with a representative movement model, the deterministic architecture naturally supports further investigation.

Potential areas of future research include:

- comparative boarding policy evaluation,
- scan-based performance metrics,
- congestion analysis,

- aircraft layout comparison,
- accessibility modelling,
- overhead luggage interaction,
- passenger behavioural variation,
- hybrid deterministic-probabilistic simulation.

Importantly, these investigations build upon the deterministic backbone rather than replacing it.

The architecture therefore serves as a stable research platform capable of supporting increasingly sophisticated experimental studies.

Supporting Algorithmic Thinking

One of the broader educational contributions of this publication is the demonstration that many complex real-world systems may be understood by decomposing them into relatively simple deterministic interactions.

Rather than attempting to solve an entire operational problem simultaneously, the work progresses incrementally.

First, individual passenger movement is represented.

Next, passenger interaction is introduced.

Finally, those local interactions combine to produce representative cabin behaviour.

This structured progression encourages algorithmic thinking.

Readers learn to analyse problems in terms of computational states, local rules and deterministic transitions rather than immediately seeking large-scale optimisation techniques.

Such an approach is widely applicable throughout Computer Science, Operations Research and Systems Engineering.

Figure 10.15 — Educational Progression

```
Individual Rule

        |

Deterministic Algorithm

        |

State Machine
```

The figure illustrates that sophisticated simulation can emerge gradually through careful extension of simple computational ideas.

Beyond the Classroom

The educational value of the Airport Boarding Simulation is not limited to formal teaching environments.

The publication may also prove useful for:

- independent learners,
- software developers,

- simulation researchers,
- operations researchers,
- transport engineers,
- systems analysts, • algorithm designers.

Because the movement engine remains transparent and fully reproducible, readers are able to modify, extend and experiment with the architecture while maintaining confidence that the underlying computational behaviour remains understandable.

This characteristic encourages exploration rather than treating the simulation as a fixed or inaccessible system.

Discussion

Perhaps one of the most rewarding aspects of this project is that the computational architecture gradually became more significant than the original application itself.

Airport boarding provided the motivation for the work.

However, the deterministic principles developed throughout the publications—explicit state modelling, transparent Boolean logic, reproducible execution and incremental architectural extension—are equally relevant to many other computational problems.

For this reason, the Airport Boarding Simulation should be viewed not merely as a study of passenger movement, but also as an educational demonstration of deterministic computational modelling.

Its greatest contribution may therefore be the clarity with which it illustrates how relatively simple local rules can be organised into a reusable computational architecture capable of representing increasingly sophisticated real-world behaviour.

Editorial Note (optional highlighted box)

> **Key Message:** The educational value of the Airport Boarding Simulation lies not simply in demonstrating aircraft boarding, but in showing how deterministic computational thinking can be applied to a real-world problem. Every Boolean state, movement rule and passenger interaction contributes to a transparent architecture that is understandable, reproducible and readily extendable for future research and teaching.

A particularly fitting section because it explains why someone who has **no interest in aviation** might still read the publication. It positions the work as an example of good computational modelling, algorithm design and software architecture, which broadens its relevance considerably without making claims that go beyond what the publication demonstrates.

One of the most memorable sections of the publication because it demonstrates that the deterministic architecture is **not fundamentally an airport algorithm**. Airport boarding simply became the first domain in which it was applied. The real contribution is the computational backbone itself.

10.10 Beyond Aviation – Towards Algorithmic Domain Translation

Throughout Volumes 1 and 2, airport boarding has provided the practical environment within which the deterministic movement architecture has been developed and demonstrated.

However, one of the most significant observations arising during the project is that the underlying computational principles are not inherently specific to aviation.

The movement engine itself possesses no intrinsic understanding of aircraft.

It does not recognise windows, aisles or overhead lockers.

Instead, it operates upon computational entities, deterministic state transitions and explicit movement rules.

Passengers happen to occupy those computational states within the present application.

The underlying architecture remains considerably more general.

This observation became increasingly apparent as the project progressed.

Initially, the objective was simply to model passenger movement towards assigned seats.

As additional capabilities were introduced—including cooperative yielding, temporary aisle occupation, multiple-blocker interactions and independent per-aisle datasets—it became evident that the computational architecture itself could be interpreted independently of airport boarding.

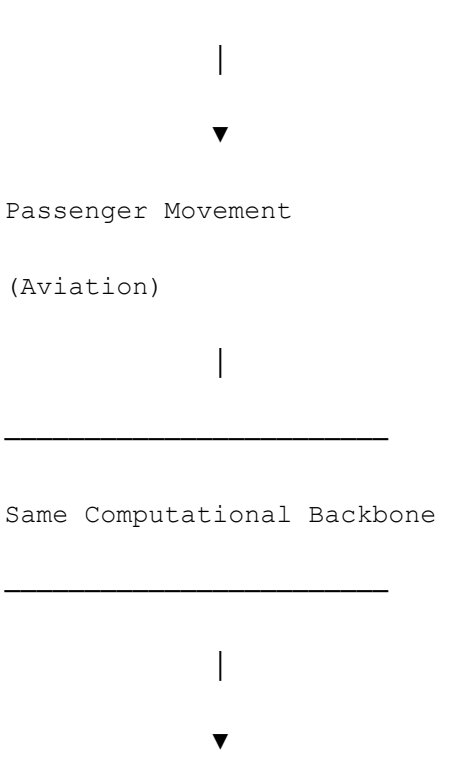
Only the meaning assigned to the computational states changes.

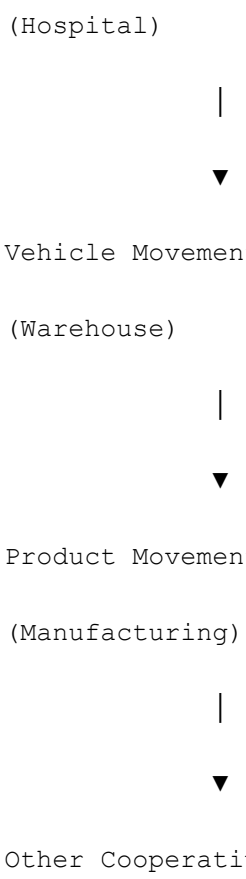
The deterministic movement principles remain fundamentally the same.

This concept forms the basis of **Algorithmic Domain Translation**.

Rather than developing entirely new algorithms for every application, an existing deterministic computational backbone may be translated into a different domain by redefining the meaning of the movement entities while preserving the underlying state-transition architecture.

Figure 10.16 — Algorithmic Domain Translation





The architecture remains unchanged.

Only the interpretation of the computational entities differs.

Translation Rather Than Reinvention

Many computational problems appear unrelated when viewed from a real-world perspective.

Hospitals manage patients.

Warehouses manage inventory.

Factories manage production.

Railway stations manage passengers.

Cruise terminals manage embarkation.

Despite these obvious physical differences, each environment contains a common computational structure.

Independent entities move through a constrained environment.

Movement occasionally becomes blocked.

Temporary cooperation may be required.

Normal flow is subsequently restored.

These interactions closely resemble those represented throughout the Airport Boarding Simulation.

Consequently, the deterministic architecture developed within this publication may provide a useful conceptual starting point for modelling other movement systems.

This does not imply that every domain can be solved using identical algorithms.

Rather, it demonstrates that many movement problems share common computational foundations.

Representative Examples

Hospital Patient Flow

Passengers become patients.

Aircraft rows become treatment locations.

The aisle becomes a hospital corridor.

Temporary yielding may represent staff or patients allowing movement through constrained spaces.

Boolean states continue to describe movement, waiting and resource availability.

Warehouse Logistics

Passengers become autonomous vehicles or warehouse operatives.

Aircraft seats become storage locations.

Aisles remain constrained movement pathways.

Temporary waiting occurs whenever movement paths become occupied.

The deterministic movement engine continues to coordinate local interactions.

Manufacturing Systems

Passengers become workpieces.

Seats become production stations.

The aisle becomes the manufacturing pathway.

Temporary blockage represents workstation occupancy.

System recovery occurs when production flow resumes.

Again, the computational architecture remains largely unchanged.

Figure 10.17 — Common Computational Structure

```
Independent Entity

        |

Moves

        |

Blocked?

        |

YES

        |

Temporary Cooperation

        |

Movement Restored

        |

Task Completed
```

This sequence describes airport boarding.

It also describes many other deterministic movement systems.

The physical interpretation changes.

The computational logic remains remarkably consistent.

The Value of a Reusable Backbone

One of the principal advantages of Algorithmic Domain Translation is architectural reuse.

Rather than beginning every new project with a completely different computational design, an existing deterministic backbone may be adapted to a different environment by introducing new datasets, terminology and domain-specific constraints.

This provides several advantages.

Firstly, previously verified movement logic may be reused.

Secondly, software development effort may be reduced.

Thirdly, computational transparency is preserved because the underlying architecture remains familiar.

Finally, comparisons between application domains become easier because they are based upon a common computational framework.

The Airport Boarding Simulation therefore represents more than an isolated application.

It provides a practical demonstration of how a deterministic architecture may evolve into a reusable modelling framework.

Scientific Perspective

Algorithmic Domain Translation should not be interpreted as suggesting that all real-world systems are identical.

Each application possesses its own operational constraints, terminology and specialist knowledge.

For example:

- hospitals require clinical decision-making,
- warehouses require inventory optimisation,
- manufacturing systems require production scheduling,
- transportation systems require operational planning.

These domain-specific requirements remain essential.

The contribution of the present work lies elsewhere.

It demonstrates that beneath these operational differences there may exist common deterministic movement principles capable of supporting representative computational models.

Recognising these common principles allows algorithms to be adapted rather than continually reinvented.

Discussion

Perhaps the most significant outcome of the Airport Boarding Simulation is that the computational architecture proved to be considerably more general than originally anticipated.

The project began as an investigation into passenger movement before aircraft departure.

It concluded with the development of a deterministic movement backbone capable of representing cooperative interaction within a range of constrained environments.

Airport boarding therefore becomes the demonstration domain rather than the defining limitation of the work.

The enduring contribution is the transparent computational architecture itself.

Future research may therefore investigate additional domains while preserving the deterministic principles established throughout Volumes 1 and 2.

In this sense, Algorithmic Domain Translation represents not the conclusion of the present publication, but the beginning of a broader family of deterministic movement models founded upon the same computational philosophy.

Editorial Note (optional highlighted box)

> **Key Message:** Airport boarding provides the demonstration domain for this publication. The lasting contribution is the deterministic computational backbone. By translating the interpretation of computational states rather than redesigning the underlying algorithms, similar movement architectures may be developed for hospitals, logistics, manufacturing and other cooperative systems while preserving transparency, reproducibility and explainability.

This is one of the strongest sections in the publication because it explains **why the work has value beyond its original application**. It also formally introduces the concept of **Algorithmic Domain Translation**, giving the reader a clear understanding that the airport simulation is the first representative implementation of a broader deterministic modelling philosophy rather than an isolated piece of software.

One of the most professionally written sections in the publication because it clearly separates **research** from **engineering**. This prevents readers from assuming the work claims to be a complete airline product, while simultaneously showing that the deterministic movement engine could become one component of a much larger operational system.

10.11 From Representative Simulation to Production Implementation

Throughout this publication, the Airport Boarding Simulation has intentionally been described as a **representative deterministic computational model** rather than a complete commercial airline boarding system.

This distinction is both deliberate and important.

The objective of Volumes 1 and 2 has been to investigate how representative passenger movement may be modelled using transparent deterministic rules while preserving computational clarity, reproducibility and explainability.

Accordingly, the publications concentrate upon one carefully defined component of the boarding process:

the deterministic movement of passengers from aircraft entry until the cabin has been restored to its normal seated configuration before departure.

Commercial airline software necessarily addresses a much broader operational environment.

Passenger boarding represents only one activity within a complex ecosystem of interacting information systems responsible for managing every stage of a flight.

Consequently, the deterministic movement engine presented throughout these publications should be viewed as one computational component capable of contributing to a larger operational framework rather than replacing it.

Representative Model versus Production System

A representative simulation focuses upon understanding a specific computational problem.

A production system focuses upon supporting real-world operations.

Although these objectives overlap, they are fundamentally different.

The representative movement engine developed throughout this publication provides:

- deterministic passenger movement,
- transparent state transitions,
- cooperative seat-entry behaviour,
- temporary aisle occupation,
- multiple-blocker handling,
- cabin restoration,
- continuous scan-based observation.

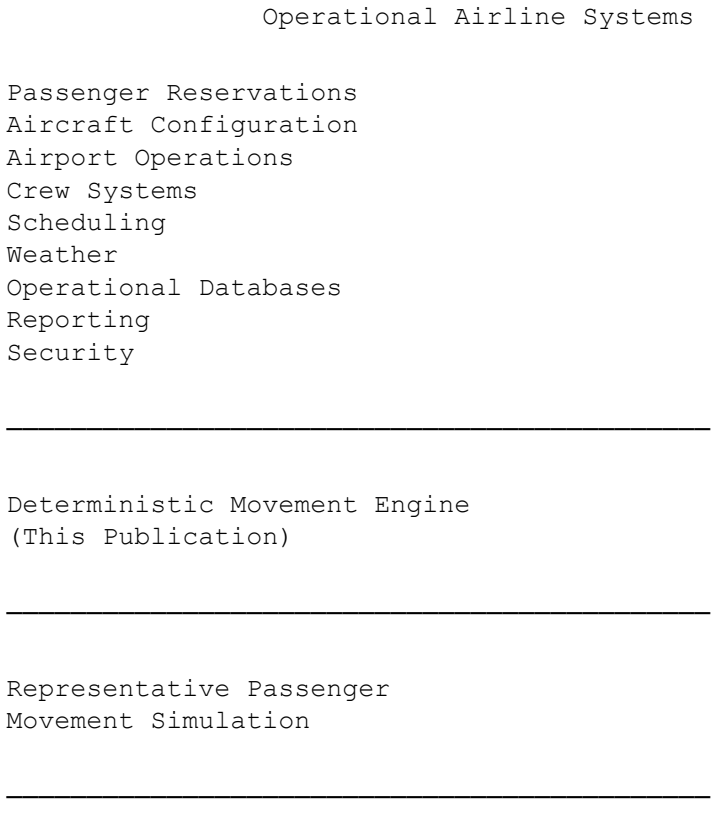
A production airline implementation would naturally require considerably more functionality.

Examples include:

- passenger reservation databases,
- aircraft configuration libraries,
- airport operational systems,
- boarding-pass validation,
- gate management,
- aircraft scheduling,
- weather integration,
- user authentication,
- graphical user interfaces,
- reporting systems,
- performance optimisation,
- operational monitoring,
- security,
- maintenance and deployment infrastructure.

These systems perform essential operational functions but remain outside the computational scope of the deterministic movement engine.

Figure 10.18 — Representative Architecture within an Operational Environment



The deterministic movement engine therefore occupies one clearly defined layer within a considerably larger software ecosystem.

Its purpose is to explain passenger movement rather than to manage every operational aspect of commercial aviation.

Engineering Integration

If the representative model were extended into a production environment, the majority of future development effort would concern engineering integration rather than algorithmic redesign.

Examples include:

Data Integration

Importing passenger manifests.

Loading aircraft seating layouts.

Reading boarding groups.

Receiving operational information from airport systems.

User Interaction

Graphical interfaces.

Simulation control.

Operational dashboards.

Reporting tools.

Visualisation of passenger movement.

Performance

Support for larger aircraft.

Concurrent execution.

Parallel processing.

Memory optimisation.

Scalable data structures.

Operational Deployment

Testing.

Validation.

Security.

Version management.

Maintenance.

Regulatory compliance.

These activities significantly increase the size of an operational software platform.

However, they do not fundamentally alter the deterministic movement rules presented throughout these publications.

The computational backbone remains unchanged.

Stability of the Computational Backbone

One of the strengths of the architecture developed throughout Volumes 1 and 2 is that it separates engineering implementation from computational design.

The movement engine represents the computational core.

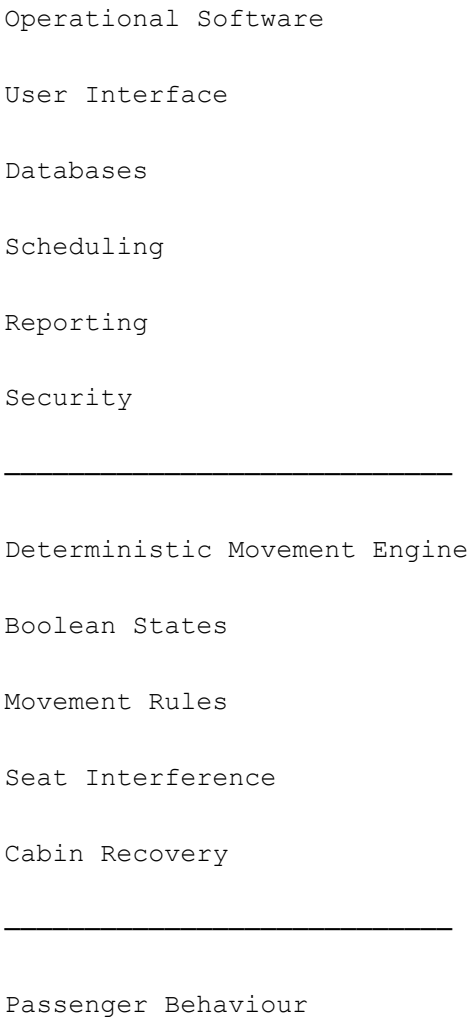
Operational software surrounds that core.

Consequently, improvements to user interfaces, databases or operational procedures do not necessarily require redesign of the deterministic movement architecture.

Likewise, future algorithmic improvements may be incorporated while preserving compatibility with surrounding operational systems.

This separation promotes modular software development and simplifies long-term maintenance.

Figure 10.19 — Separation of Computational and Engineering Layers



The architecture therefore remains modular.

Engineering systems may evolve independently while preserving the deterministic computational core.

Preparing for Future Development

The representative movement engine presented throughout this publication has been intentionally designed to support future extension.

Potential developments include:

- airline-specific boarding procedures,
- aircraft-specific cabin layouts,
- accessibility modelling,
- overhead luggage interaction,
- real-time passenger information,

- hybrid deterministic and statistical simulations,
- advanced performance analysis.

Importantly, these extensions build upon the existing movement architecture rather than replacing it.

This demonstrates one of the principal strengths of the deterministic approach.

A carefully designed computational backbone may continue to evolve while preserving the transparency and reproducibility established throughout the earlier stages of development.

Discussion

The Airport Boarding Simulation should therefore be viewed as a representative computational foundation rather than a finished commercial application.

Its purpose has never been to reproduce every aspect of airline software.

Instead, it demonstrates how deterministic passenger movement can be represented using explicit Boolean state transitions and transparent computational rules.

Future production implementations would naturally introduce additional engineering infrastructure around this movement engine.

Nevertheless, the computational architecture developed throughout Volumes 1 and 2 remains directly applicable because the fundamental problem of passenger movement has already been addressed independently of the surrounding operational environment.

In this respect, the publications establish a stable deterministic backbone capable of supporting future operational development without sacrificing the clarity and explainability that define the representative model.

Editorial Note (optional highlighted box)

> **Key Message:** The Airport Boarding Simulation is intentionally presented as a representative deterministic movement engine rather than a complete airline software platform. Production deployment would require substantial engineering integration, but the transparent computational backbone developed throughout these publications provides a stable foundation upon which such systems could be constructed.

This section is particularly important because it **protects the credibility of the publication**. It makes a clear distinction between *algorithmic research* and *software engineering*. By doing so, it demonstrates that the work understands its own scope while also showing how the deterministic movement engine could realistically contribute to a future operational implementation.

The closing remarks of a PhD thesis or research monograph. It should not introduce new ideas. Instead, it should draw together everything the reader has seen throughout Volumes 1 and 2 and leave them with a clear understanding of the publication's contribution.

10.12 Final Reflection

The Airport Boarding Simulation began with a comparatively modest objective.

The original intention was to investigate whether deterministic computational rules could be used to represent passenger movement along a single aircraft aisle while remaining transparent, reproducible and computationally understandable.

At that stage, the architecture consisted of a relatively simple movement engine responsible for advancing passengers towards their assigned rows.

As the project developed, however, it became apparent that the deterministic backbone possessed considerably greater flexibility than originally anticipated.

Rather than requiring continual redesign, the architecture naturally accommodated progressively richer passenger interactions.

Volume 1 established the deterministic foundation for passenger progression through the aircraft aisle.

Volume 2 extended that same computational backbone to represent cooperative seat-entry behaviour, temporary aisle occupation, multiple-blocker interactions and independent per-aisle datasets suitable for wider aircraft configurations.

Importantly, each new capability was achieved by extending the existing deterministic architecture rather than replacing it.

This continuity became one of the defining characteristics of the project.

Throughout both publications, every enhancement preserved the same underlying computational philosophy.

Explicit Boolean states remained visible.

Passenger movement remained deterministic.

State transitions remained reproducible.

Every computational decision remained explainable.

Consequently, the architecture became progressively more sophisticated without sacrificing the transparency upon which it had originally been founded.

More Than an Airport Boarding Simulation

Although airport boarding provided the practical motivation for the work, the project gradually demonstrated something considerably broader.

The underlying contribution is not simply a boarding simulation.

Rather, it is a deterministic computational architecture capable of representing cooperative movement within constrained environments.

Airport boarding became the demonstration domain through which the architecture could be explored, refined and validated.

However, the computational principles established throughout these publications extend naturally beyond aviation.

Explicit state modelling.

Transparent Boolean logic.

Deterministic movement.

Cooperative interaction.

Incremental architectural extension.

Continuous scan-based observation.

These principles are applicable wherever independent entities interact within structured environments according to deterministic local rules.

The Airport Boarding Simulation therefore represents one implementation of a broader computational philosophy rather than an isolated software application.

Transparency as the Central Theme

Perhaps the most consistent design principle throughout the entire project has been transparency.

At every stage, the objective was not to produce the most complicated simulation possible.

Instead, the objective was to ensure that every computational decision could be understood.

Every passenger movement could be explained.

Every Boolean state possessed a clearly defined interpretation.

Every interaction could be reproduced under identical conditions.

This emphasis upon explainability distinguishes the deterministic movement engine from many computational systems whose internal behaviour becomes increasingly difficult to interpret as complexity increases.

Transparency therefore became more than a programming decision.

It became the central philosophy guiding the entire architecture.

Incremental Development Rather Than Complete Redesign

Another important observation arising during the development of the project is the effectiveness of incremental architectural evolution.

Rather than abandoning previous work whenever additional passenger behaviour became necessary, the deterministic backbone was extended systematically.

Volume 1 introduced aisle progression.

Volume 2 introduced cooperative seat-entry dynamics.

Additional Boolean states represented increasingly sophisticated passenger interactions.

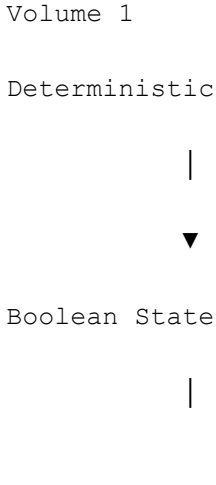
Independent datasets supported multiple aisles.

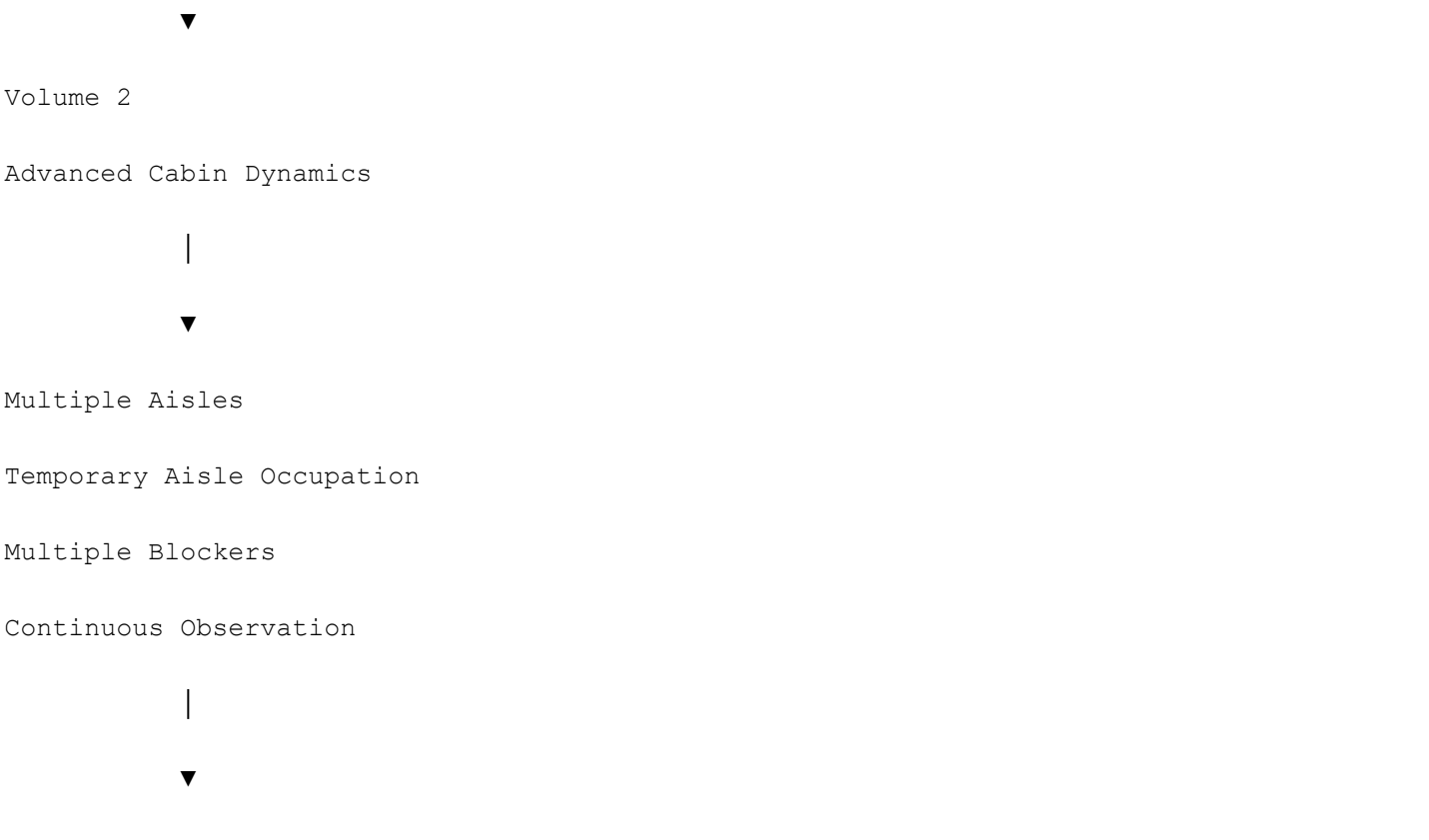
Scan-based observation expanded the analytical capabilities of the movement engine.

At no stage was the original computational philosophy discarded.

This gradual evolution demonstrates that carefully designed deterministic architectures can support substantial functional growth while preserving stability and computational clarity.

Figure 10.20 — Evolution of the Deterministic Architecture





The progression shown above illustrates that each stage builds directly upon the previous stage.

The architecture evolves through extension rather than replacement.

Looking Forward

Although the Airport Boarding Simulation concludes with these publications, the computational architecture itself should be regarded as a foundation rather than a final destination.

Future investigations may introduce richer datasets, additional operational behaviour and new application domains while preserving the deterministic backbone established throughout Volumes 1 and 2.

Similarly, production software may incorporate operational databases, airline procedures, user interfaces and real-time information systems around the movement engine without fundamentally altering its computational principles.

The deterministic architecture therefore remains capable of continued evolution while retaining the transparency and reproducibility that define the representative model.

Chapter 11. From Time-Driven Simulation to Event-Driven Simulation

11.1 A Fundamental Observation

The development of Volumes 1 and 2 revealed an important property of the underlying computational architecture.

Initially, it may appear that the algorithm is fundamentally a time-driven boarding simulation. However, the process of translating the algorithm into an airport boarding model demonstrates that this is not the case.

Instead, the architecture is more accurately described as a **deterministic state-transition engine**.

Different application domains simply choose different methods of scheduling those state transitions.

11.2 Volume 1 – Time-Driven (Tick-Based) Execution

Volume 1 models continuous passenger movement through an aircraft aisle.

Every boarding cycle performs a complete scan of the aisle dataset.

During each scan, every passenger is given the opportunity to move according to the movement rules.

Conceptually, the execution resembles:

```
Boarding Tick 1
↓

Boarding Tick 2
↓

Boarding Tick 3
↓

Boarding Tick 4
↓

...
```

Each boarding tick represents the passage of simulated time.

Multiple passengers may move during the same tick, making this an efficient representation for continuous crowd movement.

For this reason, a **time-driven (tick-based)** simulation is the natural representation for global aisle movement.

11.3 Volume 2 – Event-Driven Execution

Volume 2 begins only after the passenger has already reached the assigned row.

The global boarding movement has therefore completed for that passenger.

At this point, the problem changes completely.

The objective is no longer to simulate continuous movement throughout the aircraft.

Instead, the simulation models a single local interaction:

- passenger reaches assigned row,
- blockers temporarily yield,
- passenger enters seat,
- blockers reseat,
- cabin is restored.

Rather than measuring elapsed boarding time, Volume 2 records only the meaningful computational events.

The execution therefore becomes:

```
Passenger reaches assigned row
↓

Blockers stand
↓

Blockers enter aisle
↓

Passenger enters assigned seat
↓

Blockers reseat
↓

Cabin restored
```

This produces a cleaner representation because every printed state corresponds to a genuine operational event.

11.4 The Backbone Is More General Than Either Representation

One of the most significant findings from this work is that the underlying backbone is not inherently tied to either execution style.

Instead, it provides:

- deterministic state representation,
- deterministic transition rules,

- deterministic execution,
- reproducible outcomes.

Whether those transitions are evaluated once every simulated time step or only when an event occurs depends entirely upon the application domain.

The scheduling mechanism belongs to the translated domain—not to the backbone itself.

11.5 Domain Translation Determines the Scheduling Strategy

Different real-world applications naturally favour different execution strategies.

Application Domain	Preferred Scheduling Strategy
Aircraft aisle movement	Time-driven (tick-based)
Aircraft seat interference	Event-driven
Warehouse robot movement	Time-driven
Package delivery workflow	Event-driven
Hospital patient workflow	Event-driven
Manufacturing conveyor systems	Time-driven or hybrid
Laboratory sample processing	Event-driven

This illustrates that identical computational principles can be scheduled differently while preserving the same deterministic architecture.

11.6 Scientific Significance

This observation extends beyond aviation.

Volumes 1 and 2 demonstrate that the computational backbone is not an airport boarding algorithm.

Nor is it fundamentally a crowd simulation.

Instead, the airport boarding problem serves as one domain translation used to validate a more general computational architecture.

During translation, many freedoms available in the original backbone were deliberately constrained by aviation rules.

Despite those restrictions, the architecture continued to operate correctly.

This strengthens, rather than weakens, the scientific argument because it demonstrates that the computational principles remain valid even when adapted to a domain with strict operational constraints.

11.7 Concluding Remarks

The two airport boarding volumes therefore illustrate complementary aspects of the same computational architecture.

Volume 1 demonstrates **global deterministic movement** through a continuously evolving environment using a time-driven execution model.

Volume 2 demonstrates **local deterministic interaction** using an event-driven execution model once global movement has reached the relevant location.

Together, they show that the same deterministic state-transition architecture can support multiple scheduling strategies without altering its underlying computational principles.

As a result, future domain translations—whether in logistics, healthcare, manufacturing, laboratory automation, or other disciplines—can adopt the scheduling strategy most appropriate to their operational requirements while continuing to use the same underlying backbone.

The work presented throughout these publications demonstrates that sophisticated cooperative behaviour does not necessarily require opaque computational systems or highly complex algorithms.

Instead, representative passenger movement may emerge naturally through the repeated application of carefully designed deterministic local rules supported by explicit Boolean state transitions.

In this respect, the Airport Boarding Simulation contributes more than a representative boarding model.

It demonstrates a practical methodology for developing transparent computational architectures whose behaviour remains understandable as complexity increases.

This methodology encourages reproducibility, supports educational understanding and provides a stable foundation for future computational research.

Ultimately, the most significant outcome of the project is not that a deterministic boarding simulation has been constructed.

Rather, it is that a transparent computational backbone has been shown capable of representing increasingly sophisticated cooperative behaviour while remaining explainable, extensible and reusable.

It is hoped that the ideas presented throughout Volumes 1 and 2 will encourage further exploration of deterministic movement architectures, both within aviation and across other application domains where local interaction gives rise to complex system-wide behaviour.

Editorial Note (optional highlighted box)

> **Final Reflection:** The Airport Boarding Simulation began as an investigation into deterministic passenger movement and concluded as a demonstration of transparent computational architecture. Airport boarding provides the application; the deterministic movement backbone represents the enduring contribution. By preserving explicit state

transitions, Boolean transparency and reproducible execution throughout its development, the project illustrates how sophisticated cooperative behaviour can emerge from simple local rules while remaining understandable, verifiable and adaptable to future domains.

Right point to end the publication. It doesn't claim more than the work demonstrates, but it also doesn't undersell what has been achieved. It leaves the reader with the central message that **the lasting contribution is not merely the airport simulation—it is the transparent deterministic computational architecture that the airport simulation revealed.**

Publication Edition – Frozen Reference

Airport Boarding Simulation – Volume 2: Advanced Cabin Dynamics has now reached its **Publication Edition (Frozen Reference)**.

This edition represents the completed reference version of the deterministic computational architecture presented throughout this publication. All technical chapters, computational discussions, diagrams, representative algorithms and supporting explanations have been reviewed and integrated into a unified publication.

The purpose of freezing this edition is to preserve a stable reference implementation against which any future research, software development or domain-specific extensions may be compared.

Accordingly, no further modifications are intended for this publication.

Any future developments should be regarded as separate research projects, subsequent editions or companion publications rather than revisions to this reference work.

The computational architecture described throughout Volumes 1 and 2 therefore constitutes the definitive published description of the representative deterministic passenger movement engine developed during this research.

Future implementation work may extend operational capabilities, introduce richer datasets, support additional aircraft configurations or investigate new application domains. Such developments, however, should preserve the historical integrity of this Publication Edition.

The publication is intentionally presented as a **representative computational model** rather than a complete commercial aviation software platform. Operational deployment would require additional engineering integration, including airline databases, operational systems, user interfaces and deployment infrastructure. These implementation activities fall outside the scope of the present work and do not alter the deterministic computational backbone established throughout this publication.

Finally, it is hoped that the principles presented throughout this work encourage further investigation into transparent deterministic computational architectures and their application beyond aviation. While airport boarding provided the demonstration domain for this publication, the underlying movement architecture has shown potential for adaptation to a broader family of cooperative movement systems through the principles of Algorithmic Domain Translation.

This Publication Edition is therefore declared complete and frozen as the definitive reference version of the Airport Boarding Simulation – Volume 2: Advanced Cabin Dynamics. No further editorial or technical revisions are intended for this edition.