Simulation of Complex Mining Port Operation, with Robust Code Testing to Facilitate Model Longevity

THE ANYLOGIC CONFERENCE 2025

Acknowledgement of Country

I’d like to acknowledge the Traditional Custodians of the land on which we meet today and pay my respects to Elders past, present and emerging. I also wish to extend this respect to other Aboriginal people and Torres Strait Islanders who are present.

Our team

Combining the best of Fortescue and Theia

- Fortescue’s internal modelling and analytics team
  - Access to Fortescue’s data systems and SMEs
- Theia Simulation
  - Deep mining and simulation expertise

About the presenters

Fortescue

Rhet Magaraggia

- Superintendent, Modelling and Analysis
- 8 years in Simulation
- Mining, Energy, Processing, Bulk and Cargo Shipping

THEIA

Chris Plottke

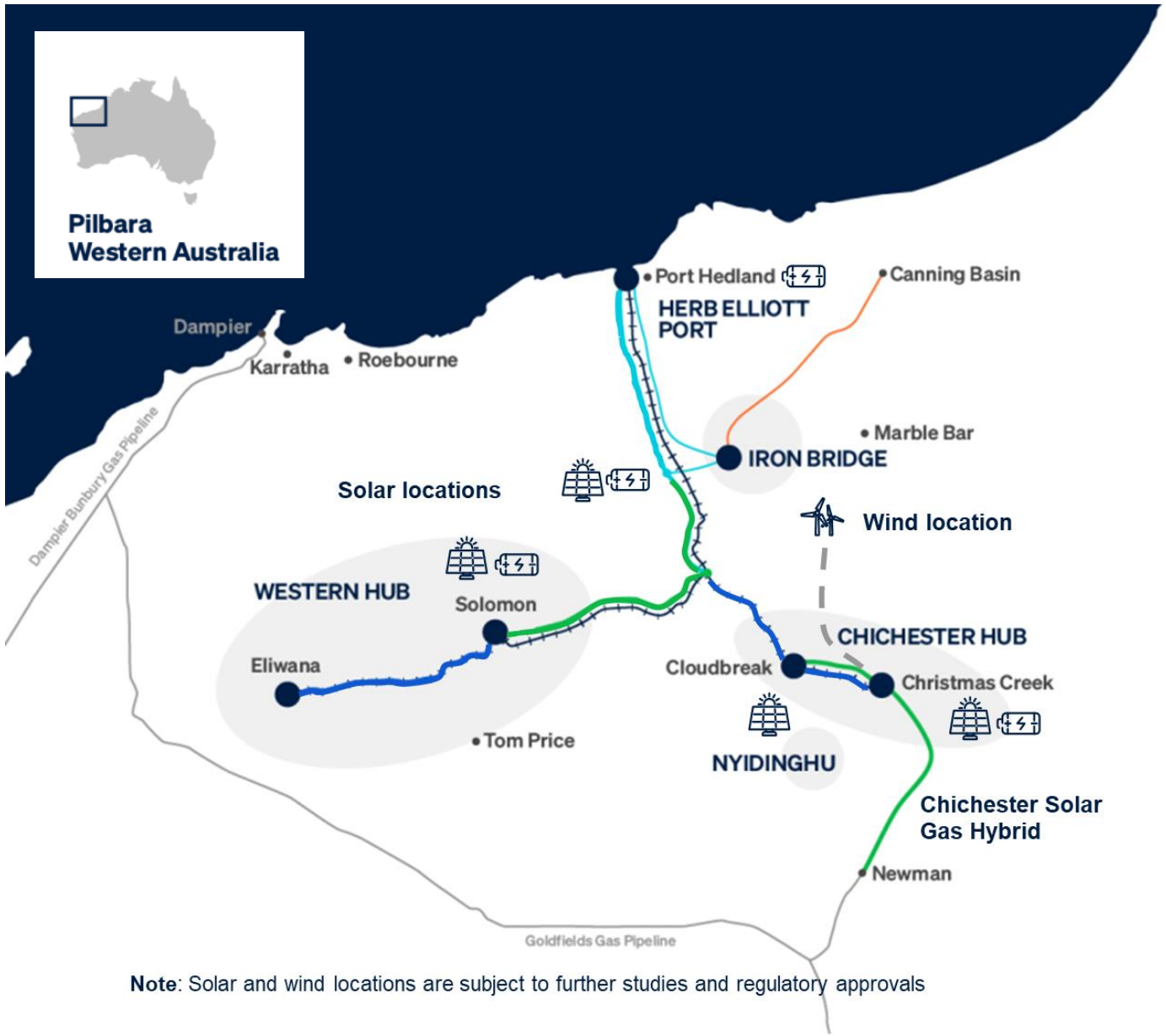
- Principal Consultant and Partner
- 19 years in Simulation
- Mining, Manufacturing, Food processing, Banking, Airlines

THEIA

Yitping Kok

- Lead Consultant and Partner
- 5 years in Simulation
- Optimisation, integration with web apps, full stack dev, data engineer

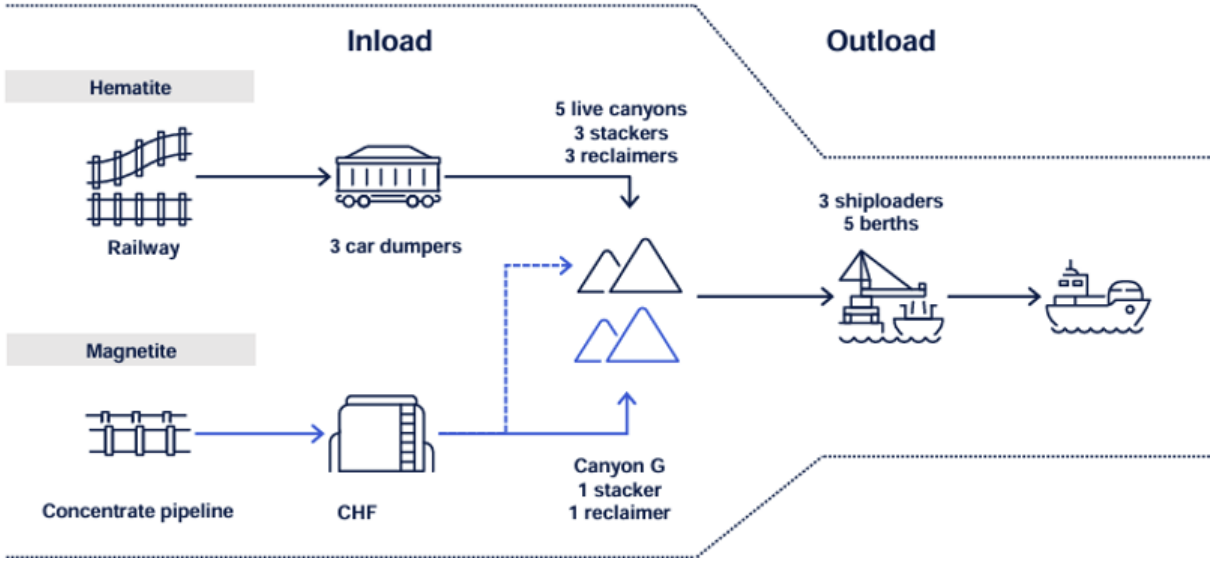
Overview of Fortescue Pilbara operations



Note: Solar and wind locations are subject to further studies and regulatory approvals

- Five mines
- Many 100's km of rail
- >100km slurry pipeline
- Complex and dynamic port

Focus on the port: All tonnes flow through here



Dynamic stockyard layout with multiple products

Priorities and rules

Multiple choices inload and outload

FY26 Guidance 195 – 205Mt

Well functioning port is key to achieving this

The usual stuff – maintenance, weather, failures, rates etc

We had clear targets for success…

Technical

- Fully inputs data driven & parameterized
- Upstream/downstream expandable (to mine, rail, marine) (use of interfaces)
- Modular Inload & Outload planning heuristics (able to test alternatives)
- Clear separation of model entities
- Code acceleration via robust libraries
- Clean implementation and codebase
- Fast! (< 1 minute ok, < 10 seconds great)

Process

- Straight-forward to test key operational and infra scenarios through inputs
- Minor experiments have week turnaround (not month)
- Well defined methodology to data drive the model and set up scenarios
- Full visibility in scenario generation process
- Well documented data & logic assumptions, and knowledge transfer to team
- Some input/outputs more closely aligned to business categories

…to create a future-proof simulation model

- To answer Fortescue's current business questions; questions likely to arise soon; and questions further down the track -> 10+ years (harder to predict!)
- With quality and clarity of code to ensure efficient transition of individual modellers off and on the team

- Design week decision: incorporate unit testing into the model build
  - We had the theory
  - We had created a testing library
  - But would be our first "serious" use in an AnyLogic model

Why AnyLogic

Theia modellers* have used AnyLogic for 10+ years as our Simulation tool of choice

Robust and well supported simulation platform

Full support for Object Oriented Programming

Able to use Git workflows

Inbuilt libraries supporting various simulation styles

Easy to extend with custom Java code and libraries

Reasonably fast execution speed

Intuitive to use and easy to learn

Strong debugging capabilities

Easy to set up good animations

Inhouse libraries allow faster development

* Including work done previously at The Simulation Group and Boston Consulting Group

Our Test suite includes three types of tests

True Unit Tests

- Testing methods on agents and classes
- Independent of simulation environment
- Simple and direct

Sub-model Tests

- Recreating scaled down component of the model
- Running simulated environment over a short period of time
- Controlled interactions, verify expected outcome

Full-run Tests

- Run base case for full runtime duration
- Test subset of KPIs against expected range
- Detects changes that fundamentally change the model result

Experiment runs JUnit Suite of Test classes

- TestExperiment triggers JUnit tests

```
/**
 * TRunManager
 */
@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestEmptyMain.class,
    TestStateManager.class,
    TestBuffer.class,
    TestCircuit.class,
    TestJob.class,
    TestInloadJob.class,
    TestTrainSpawner.class,
    TestFailures.class,
```

- Test classes define Mock classes to use in tests and set up initial test environment in the constructor

```
/**
 * TestInloadJob
 */
public class TestInloadJob {

    public static class MockQueue extends TrainQueue {
        @Override
        public void removeTrain(TrainAgent train) {
            // do nothing
        }
    }
```

```
public TestInloadJob() {
    // Create new root object
    main = new EmptyMain(
        engine, new ArrayList<Pair<Class<?>, String>>(Arrays.asList(
                Pair.of(TrainAgent.class, "train"),
                Pair.of(TUL.class, "tul"),
                Pair.of(StackerReclaimer.class, "stacker"),
                Pair.of(PortJobScheduler.class, "scheduler"),
                Pair.of(JobExecutor.class, "executor"),
                Pair.of(Stockyard.class, "basicYard")
        ))
    );
    main.setParametersToDefaultValues();
```

- @Before & @After decorators allow additional setup and teardown
- Inside Test classes, JUnit framework detects and runs all @Test methods

```
/**
 * Check that all the ore flow is done and job completed after 102 minutes has elapsed.
 * Expected event sequence: 1m-job and train arrive, 2m-complete prework, 102m-complete work
 *
 * Start time: 1m
 * Final time: 102.0001m
 */
@Test
public void testOreFlowCompleted() {
    this.jobSubmissionEvent();
    this.trainArrivalEvent();
    scheduler.disablePlanning();
    engine.runFast(102.0001);
    assertEquals(100.0, job.getMovedTonnes(), 0.0);
    assertTrue(job + " has moved planned tonnes", job.haveMovedPlannedTonnes());
    assertEquals(JobState.COMPLETED, job.getState());
    assertEquals(0.0, train.getCurrentAmount(), 0.0);
    assertEquals(100.0, pile.getCurrentAmount(), 0.0);
}
```

- Require all tests passing to approve PR

```
JUnit version 4.13.2
.............................................
Time: 1.258

OK (126 tests)
```

AnyLogic features make unit testing challenging

- It’s hard to dynamically construct a test experiment for a specific unit test
- Agents behave differently in unit tests if not properly initialised
  - Particularly in terms of default variable values
- Animation and other features can be a source of runtime errors in tests
  - Have resorted to refactoring model code to have animation off flag

Testing requires an investment, but the benefits make it worthwhile!

Testing framework comes at a cost

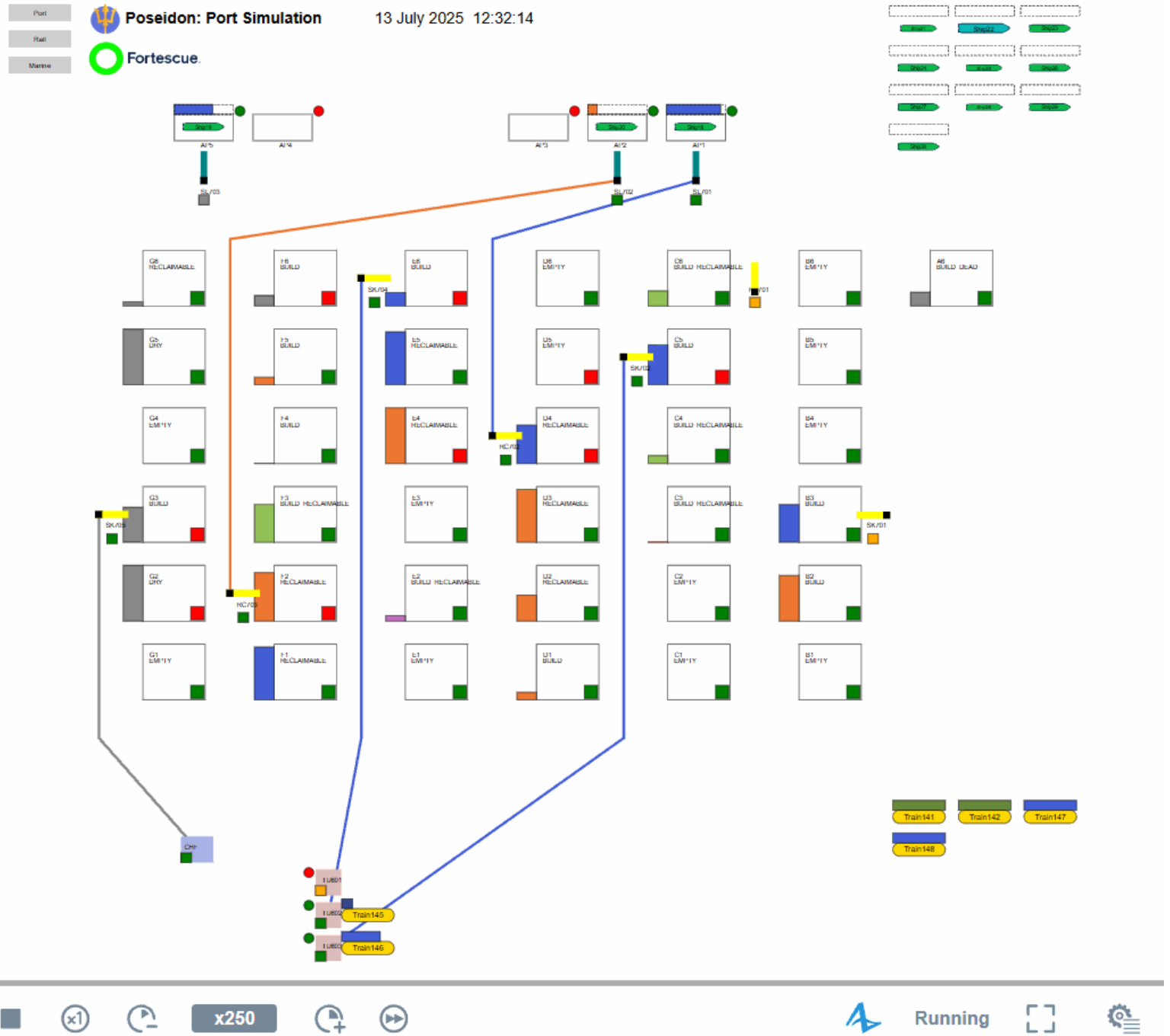
- Significant additional development time to include unit tests of new features
- Requires refactoring when new features break existing tests
- Need a pragmatic approach to what tests are appropriate

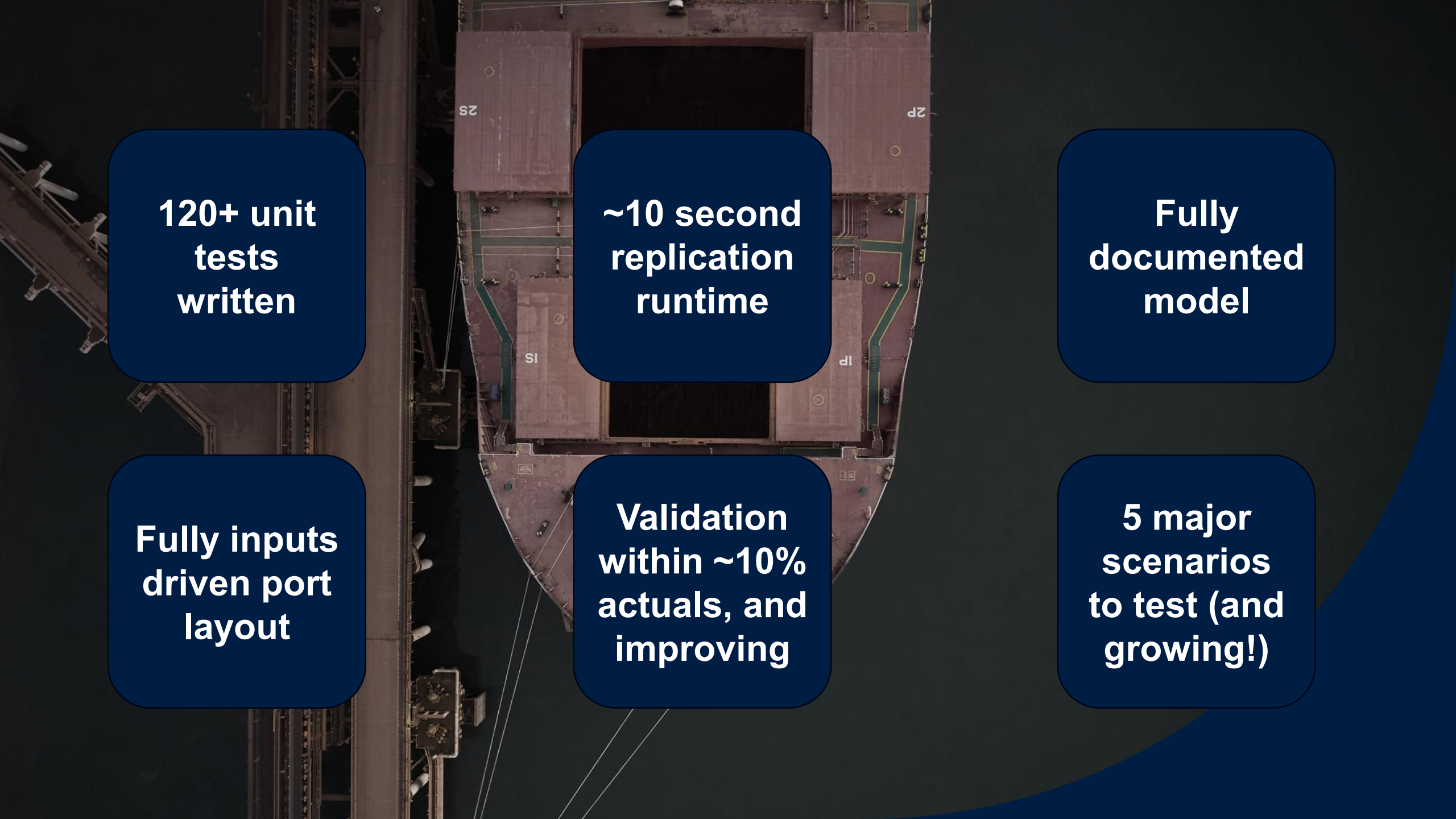
But has many benefits

- Verifies that new features are working as intended
- Detects straight away when there's a breaking change to old functionality
  - Helps find the source of bugs quickly
- Has enabled rapid development of features, avoiding excessive debugging

Animation

- Ship queue, train queue are battery limits
- Focus on stockyard process
- Visual focus on jobs and equipment states





120+ unit tests written

~10 second replication runtime

Fully documented model

Fully inputs driven port layout

Validation within ~10% actuals, and improving

5 major scenarios to test (and growing!)


Fortescue
THEIA
Clarity from complexity
www.theia.com.au
We are Fortescue
fortescue.com