Testing ETL

Don't dig a tunnel with a toothpick

Iain McCowatt

http://exploringuncertainty.com

iain@exploringuncertainty.com

@imccowatt

imccowatt

CGI

Introduction: ETL

- What is ETL?
- Where is it used?
- How does it work?

Introduction: Challenges

- No GUI
- Tester skills
- Paucity of off-the-shelf tools

Introduction: Scope of Presentation

- Functional testing of ETL, primarily for accuracy
- Logic testing of ETL, i.e. verification, rather than validation

Context: Types of ETL Logic

Logic Type	Description	Extract	Transform	Load
Selection	Selection of source data	✓	✓	
Mapping	Simple transfer of data from source field to destination field		✓	
Transformation	Manipulation / normalization of data from source to destination		✓	
Load	Load strategy (e.g. append/merge) Referential Integrity			✓
Exception	Management of exceptions (invalid data, invalid lookups, orphan records etc.)		✓	✓

Context: Other Factors

- Project constraints: time, cost, quality
- Size
- Complexity
- Types of load (e.g. full, delta etc.)
- Data volume
- Degree of specification
- Tester skills

Test Strategy: Data

Approach	Indications	Contraindications
Production	Need to test with volume Need noise	Data privacy issues
Synthetic	Need to cover logic	Little visibility into the logic
Conditioned	Need volume, noise AND coverage	Data privacy issues Little visibility into the logic

Test Strategy: Oracles

Approach	Indications	Contraindications
Source Data	Minimal selection & transformation	Any complexity
Previous Implementation	Regression testing Previous implementation available	Significant change from previous implementation
Baseline Data	Regression testing Baseline data available	Significant change from previous implementation
Parallel Implementation	No appropriate Oracle is available, e.g. new implementation	Significant time and cost constraints Testers lack development skills

Test Strategy: Verification

Approach	Indications	Contraindications
Profiling	Logic is of minimal complexity	Data integrity is critical
Visual Diff	Data integrity is critical	Large data volume Multiple types of load Time or cost is critical
Automated Comparator	Data integrity is critical Testing with large data volumes or multiple types of load	Testers lack development skills

Case Study: Context

- Previous release overran: time critical
- Regulatory project: quality (data integrity) critical
- Significant transformation logic (>130 tables, >950 fields, >2200 conditions), some highly complex, but reasonably specified
- Multiple types of load with different logic (full, delta)
- Preference for testing with production data

Case Study: Strategy

Data	Oracle	Verification
Production	Source Data	Profiling
Synthetic	Previous Implementation	Visual Diff
Conditioned	Baseline Data	Automated Comparator
	Parallel Implementation	

Case Study: Strategy

Data	Oracle	Verification
Production		
Synthetic		
✓Conditioned		mparator

Client preference for production data
Reliability concerns suggested a need for "noisy" data

Complexity of transformations drove need for coverage

Conditioned data is the compromise

Case Study: Strategy

Data	Oracle	Verification
Production	Source Data	
Synthetic	Previous Implementation	Visual Diff
Conditioned ✓	Baseline Data	Automated Comparator
	✓Parallel Implementation	

Complex Transformations ruled this out

Significant changes from previous release ruled these out

With no other suitable Oracle, a Parallel Implementation was selected despite time and cost constraints

Case Study: Strategy

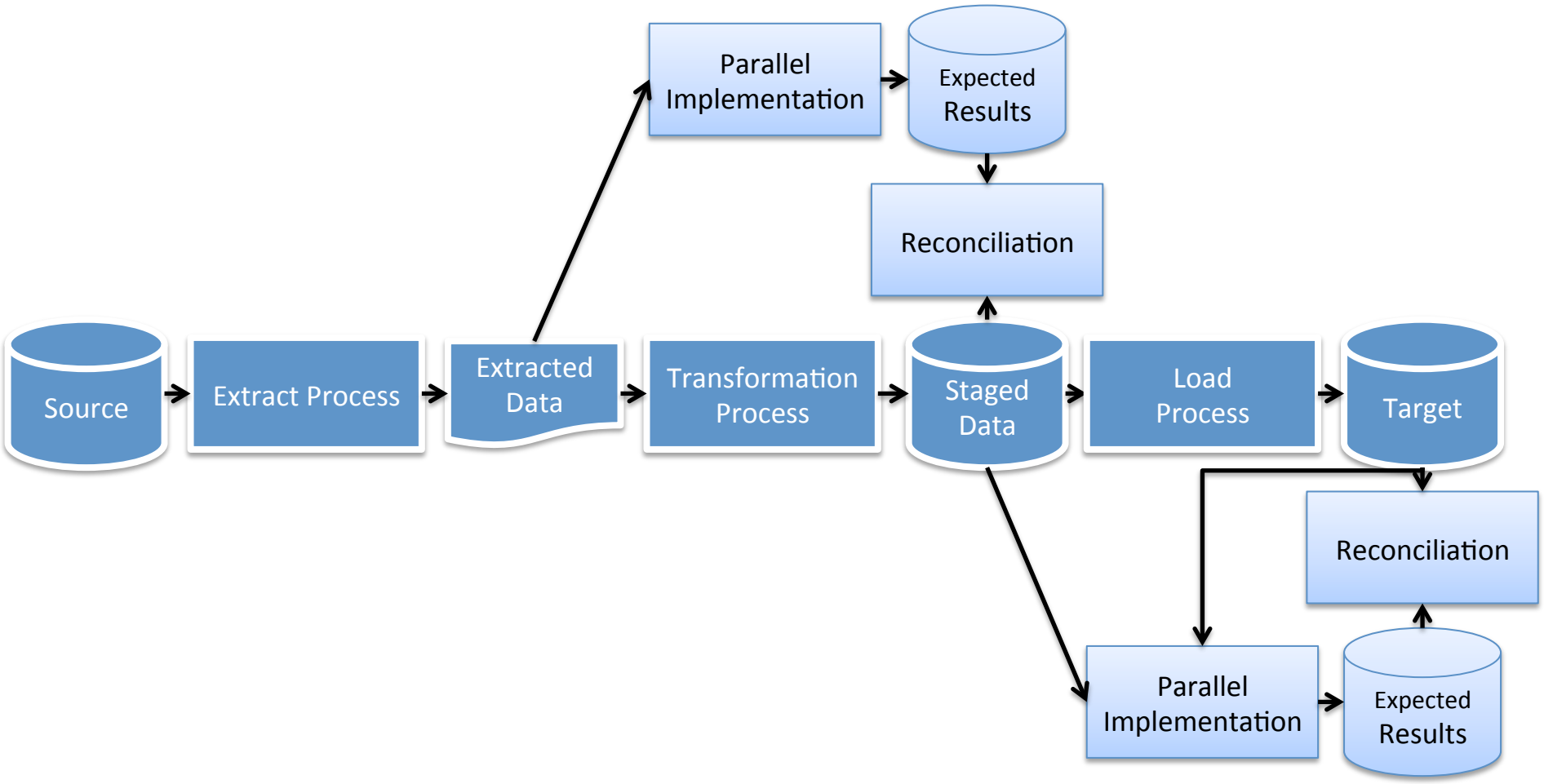
Data	Oracle	Verification
Production		Profiling
Synthetic	ientation	Visual Diff
		✓Automated Comparator
	lementation ✓	

Importance of data integrity ruled this out

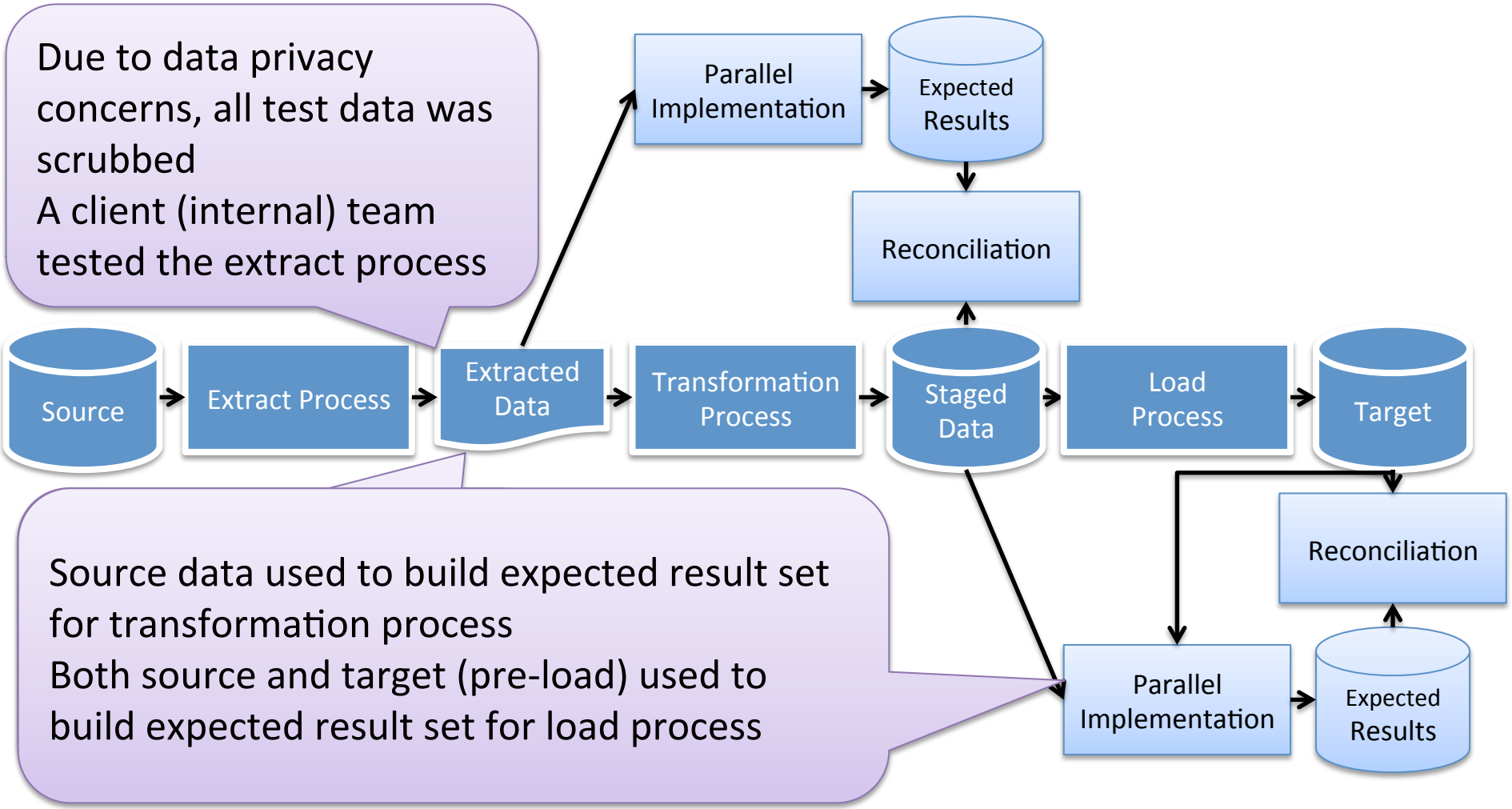
Given time and cost constraints, data volumes and repetitions with different types of load ruled this out

Selected as the more efficient option

Case Study: Approach



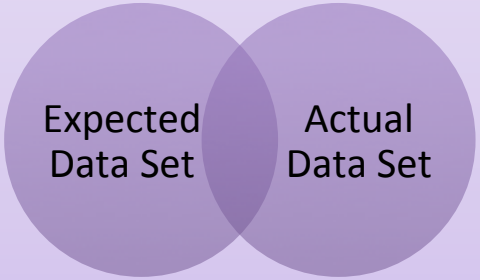
Case Study: Approach

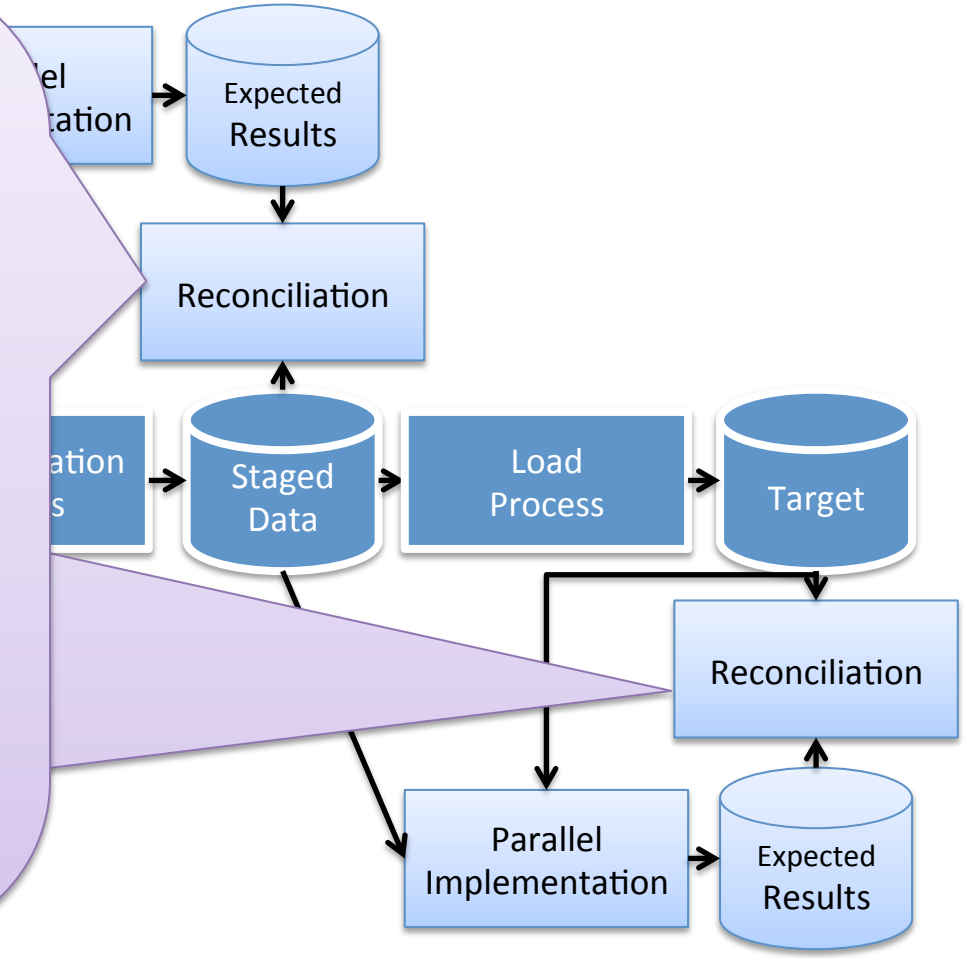


Case Study: Approach

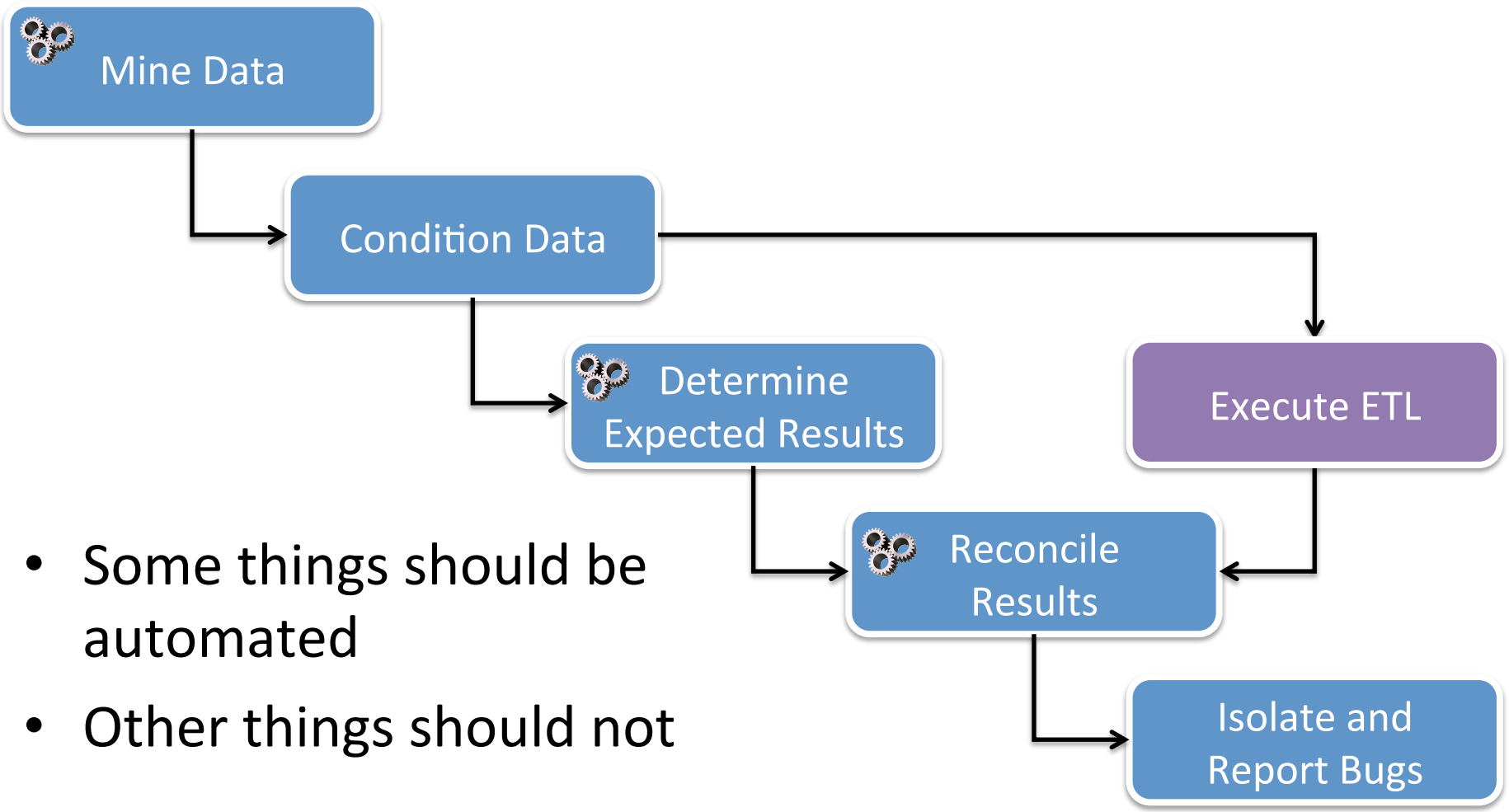
Reconciliation is a comparison of the expected result data set generated by the parallel implementation and the actual result data set generated by the ETL tool

Records in the relative complements (i.e. mismatches) indicate potential problems:





Case Study: Process

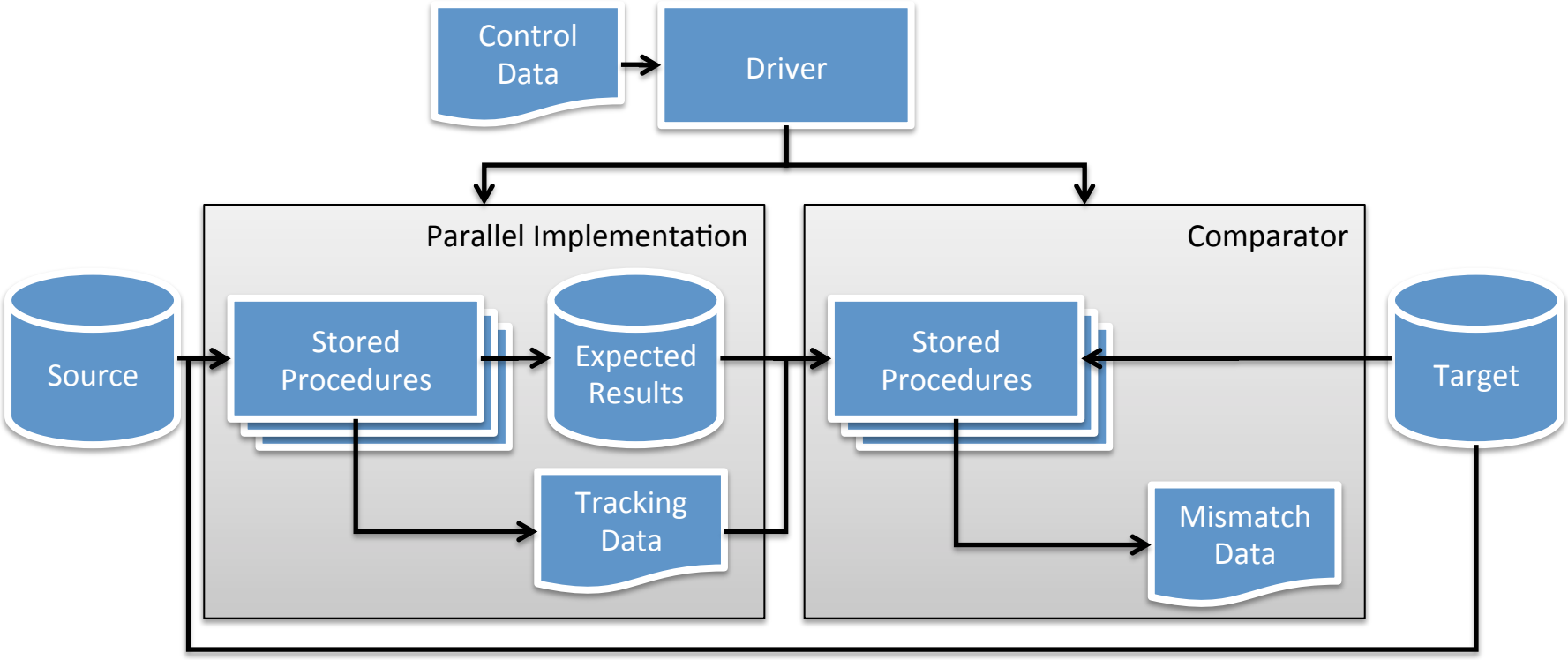


- Some things should be automated
- Other things should not

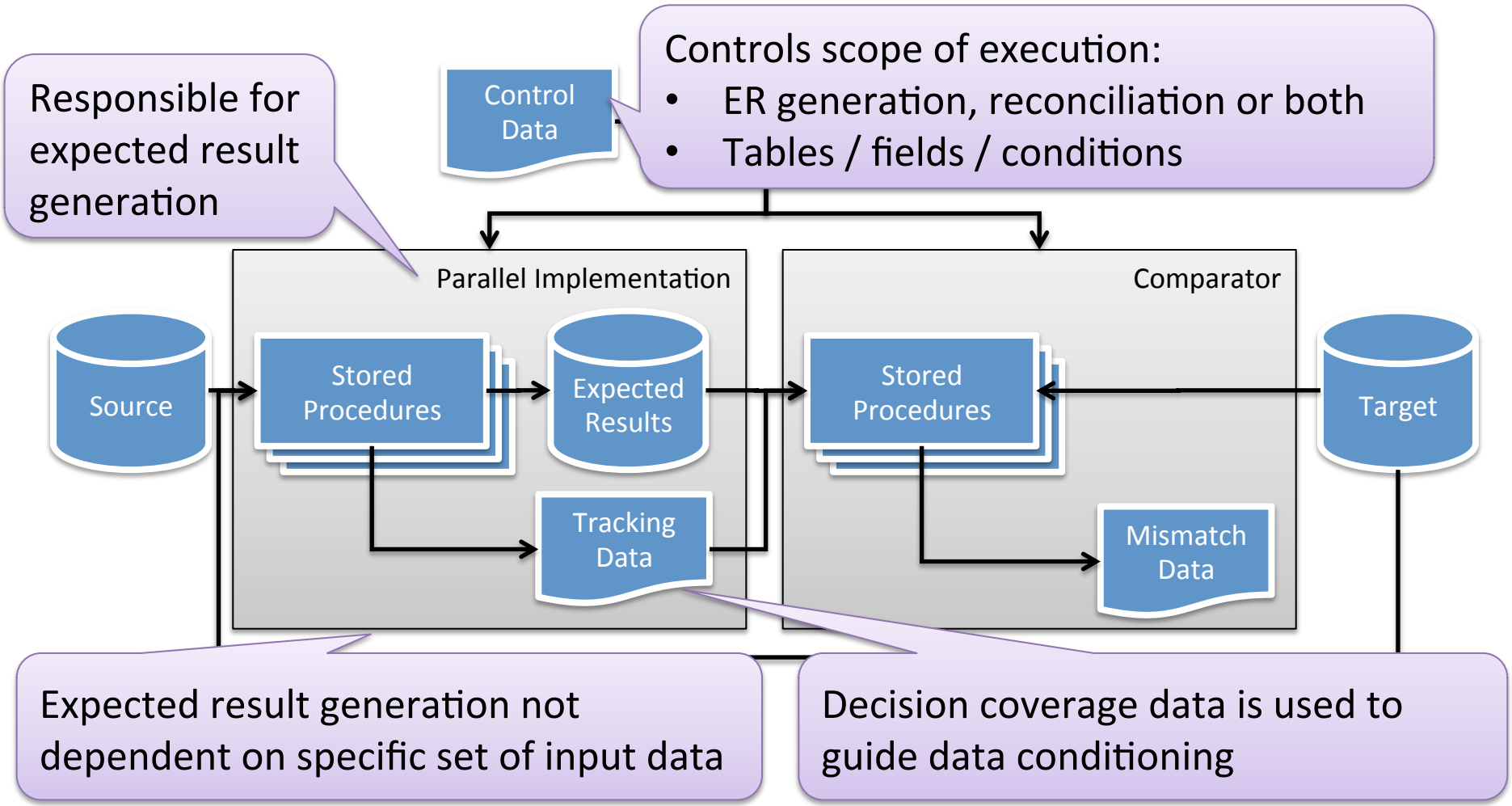
Case Study: Tool

- A custom tool was created using Oracle (SQL, PL/SQL)
- This serves as both a parallel implementation and comparator
- A generic framework was used, enabling:
  - Easier addition of new transformations
  - Control over scope of any given execution

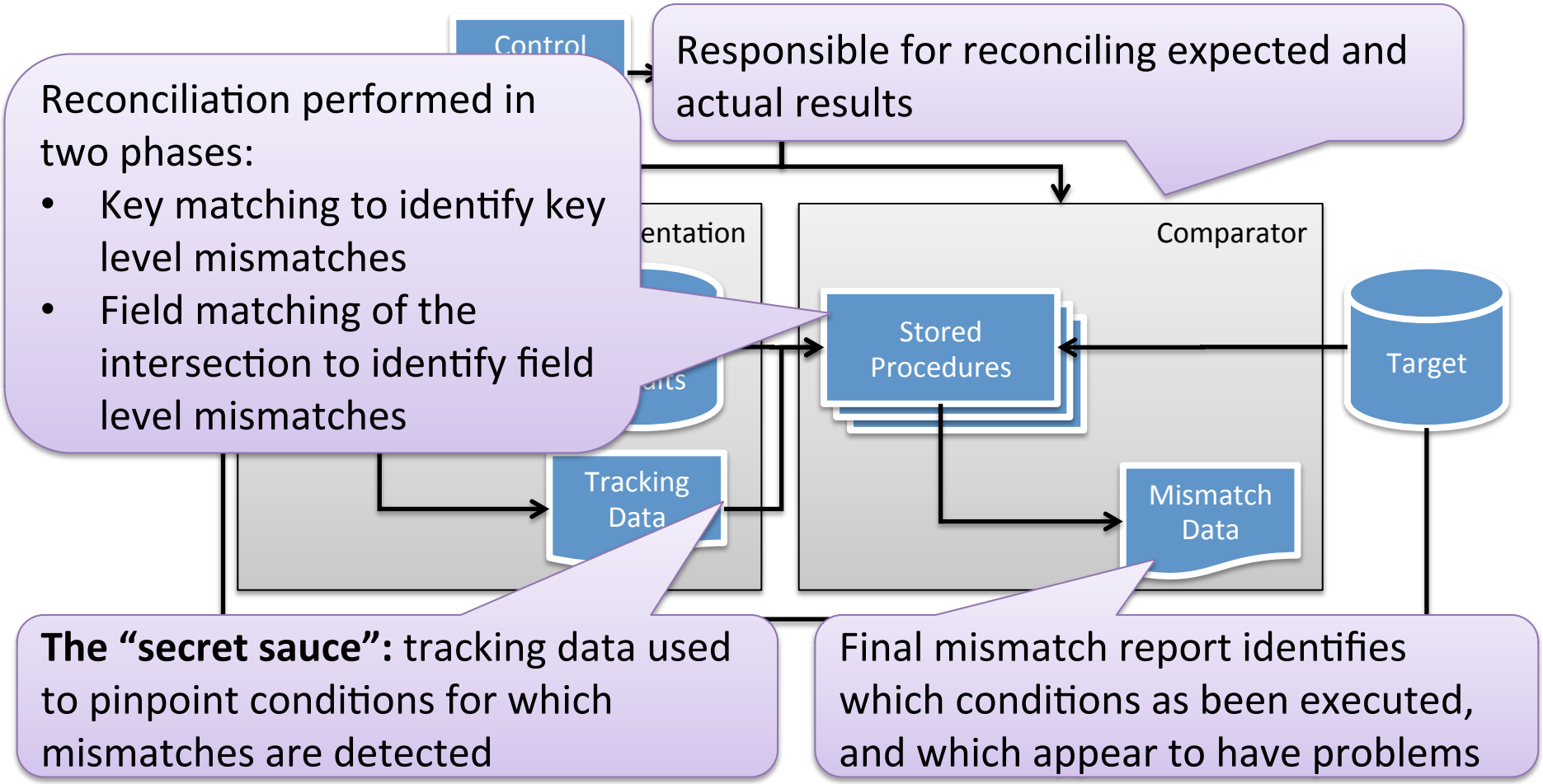
Case Study: Tool



Case Study: Tool



Case Study: Tool



Case Study: Tool

- On a shared “run-of-the-mill” development box, the tool was able to reconcile >10 Million cells (Rows x Columns) per minute, enabling execution and reconciliation of a complete load in a matter of hours
- This use of “large scale mechanized checking” enabled a degree of coverage not possible on the previous release, where a more manually intensive approach was applied

Case Study: A Note on Tools

- This is an example of a heavy weight Oracle and Comparator, designed for reuse and high performance with large data volumes
  - Other projects may require neither
- Similar tools can be constructed very quickly using off-the-shelf database software
  - For an earlier project, MS Access was used to construct a parallel implementation in a matter of days, and a comparator in under an hour

Case Study: Team

- Data literate, mix of manual testers and traditional automators
- Prior SQL but no prior PL/SQL experience
- Ready, willing and able to learn whatever needed to be learned in order to test
- Ready to challenge each other, and to be challenged

Case Study: Lessons Learned

- The tool was “just a tool”: human factors were more important, e.g.:
  - Availability of information
  - Responsibilities and handoffs
  - Communication of changes

Case Study: Lessons Learned

- The tool, and the testing, were heuristic in nature:
  - Tests (whether manual or automated) can have bugs in them: tests that do not reveal problems don't "pass", they simply do not reveal problems
  - Our checks were only as good as the requirements and design, and were blind to requirement and design bugs
  - We were forced to use composite keys where true keys could not be replicated, increasing our blindness to certain logic bugs
  - In some cases, conditions were not mutually exclusive, making isolation of any corresponding mismatch less precise (i.e. at a field rather than condition level)

Questions?