ChronoLog Documentation

Project Information

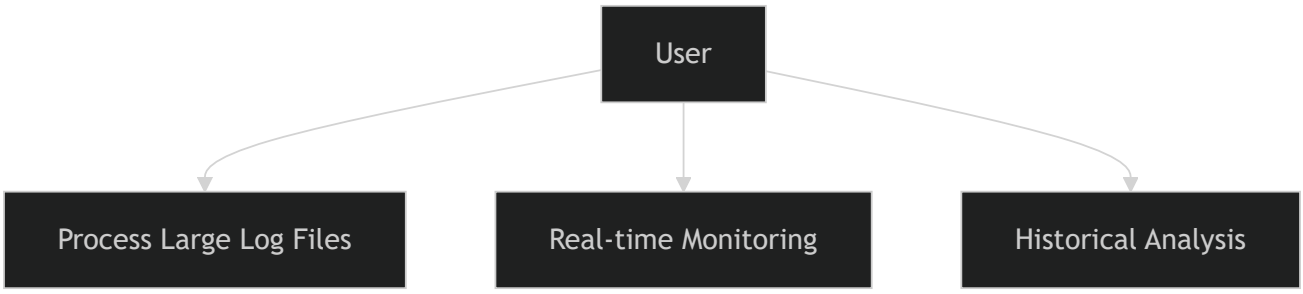
- **Project Name**: ChronoLog
- **Author**: Miro Slezák
- **Date**: 24.12.2025
- **Institution**: Střední průmyslová škola elektrotechnická, Praha 2, Ječná 30
- **Subject**: Information Technology

Overview

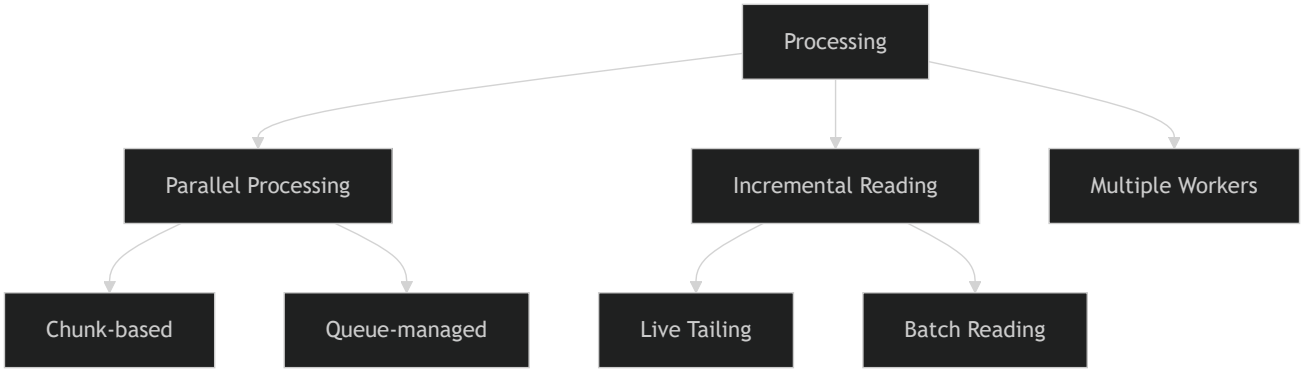
ChronoLog is a high-performance parallel log analyzer designed to process large log files efficiently. It supports both live (real-time) and batch processing modes, automatically detecting errors, warnings, and custom metrics while producing structured outputs for visualization.

Business Requirements

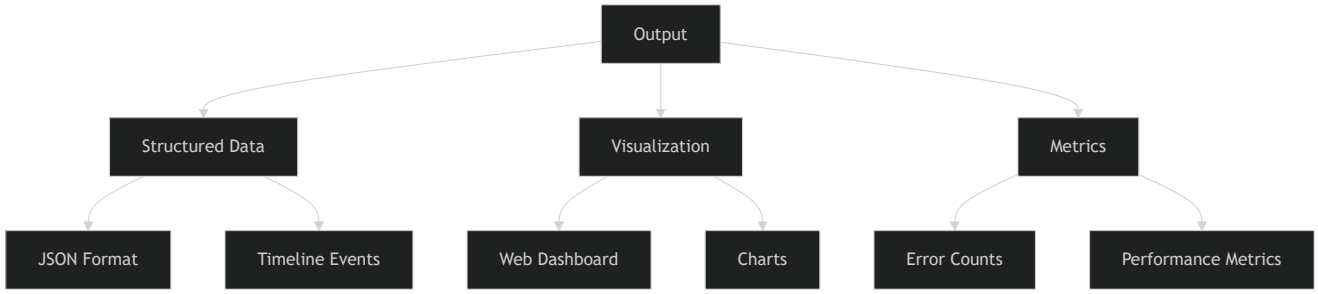
User Requirements



Processing Requirements

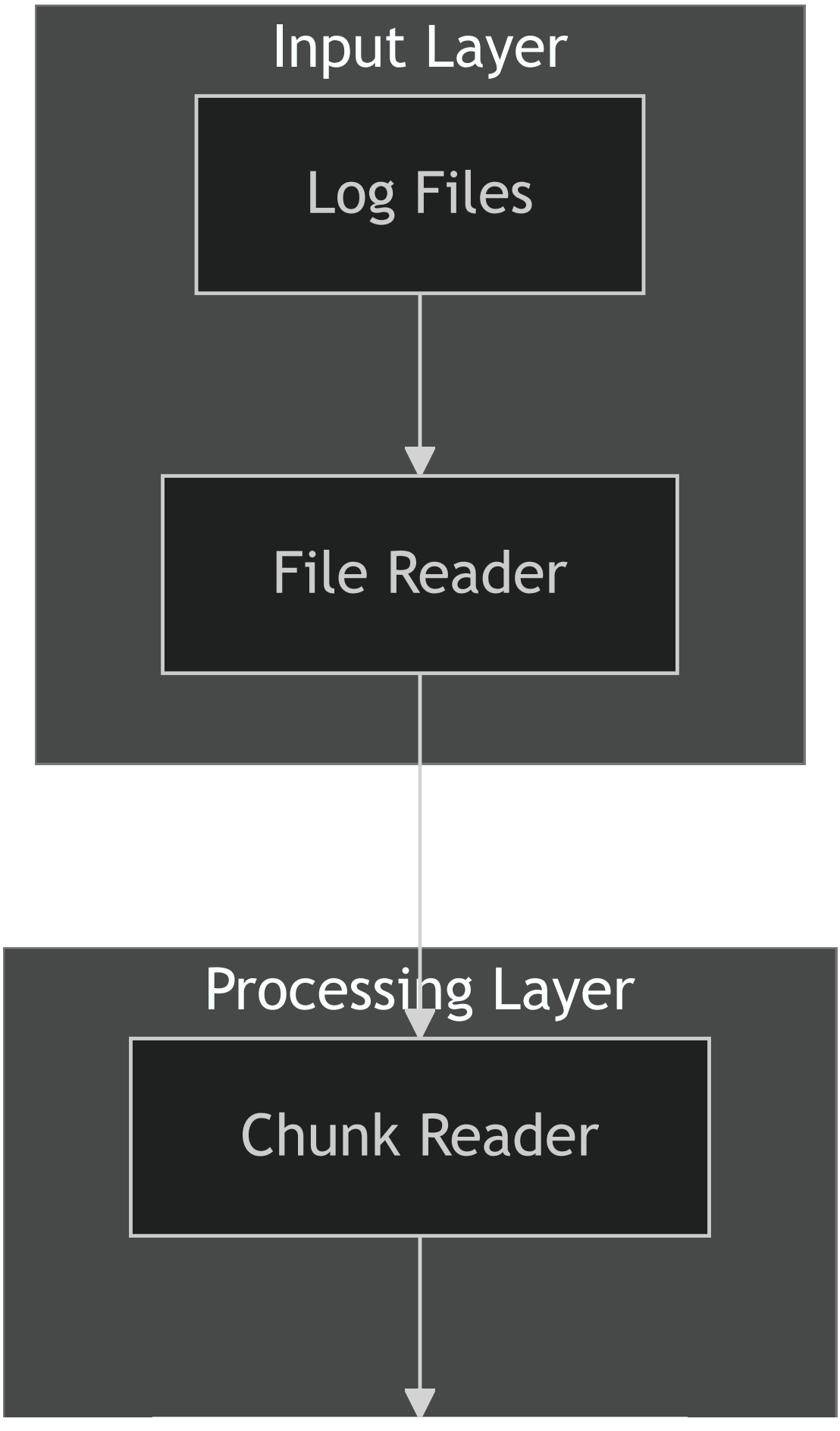


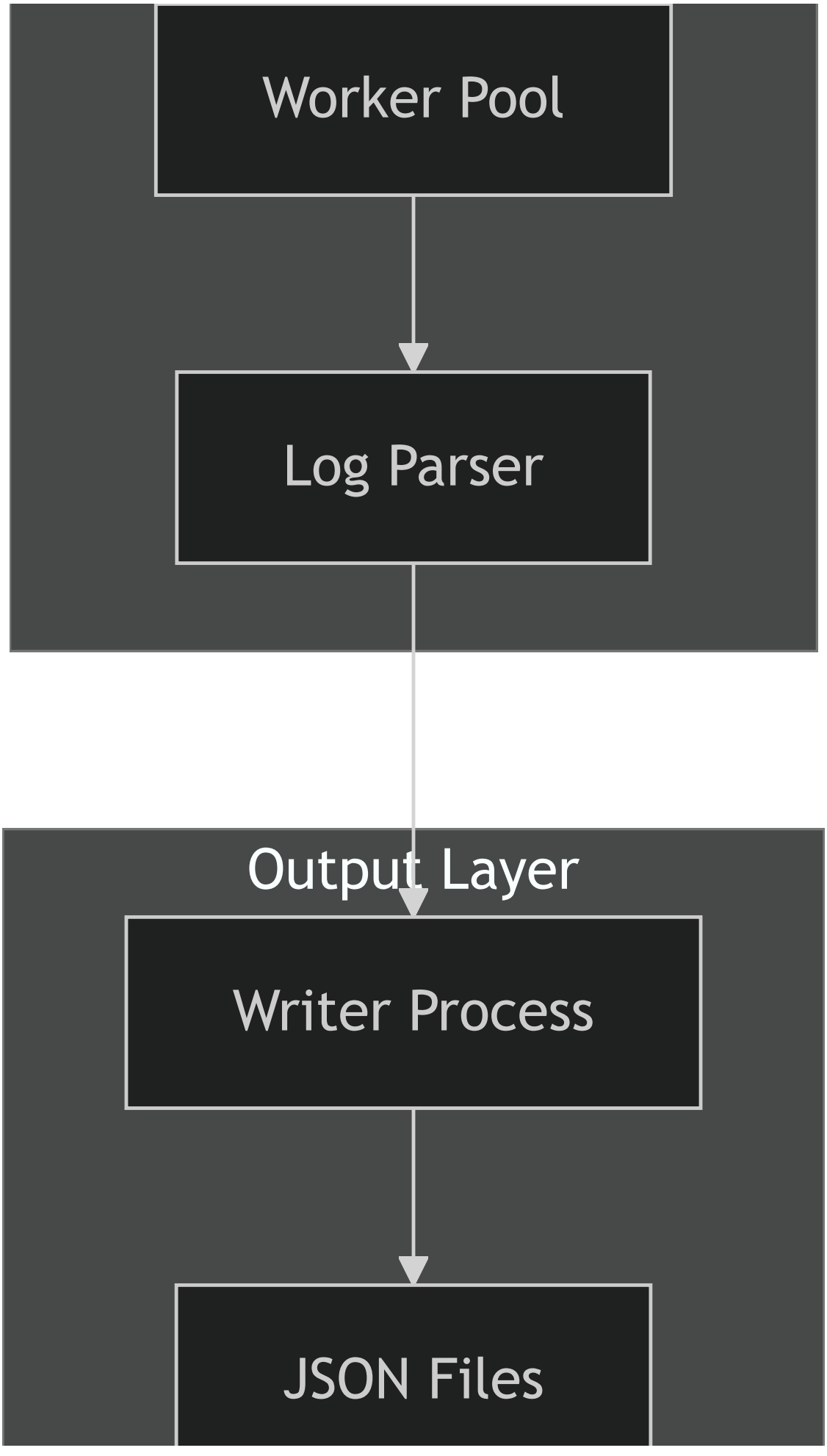
Output Requirements

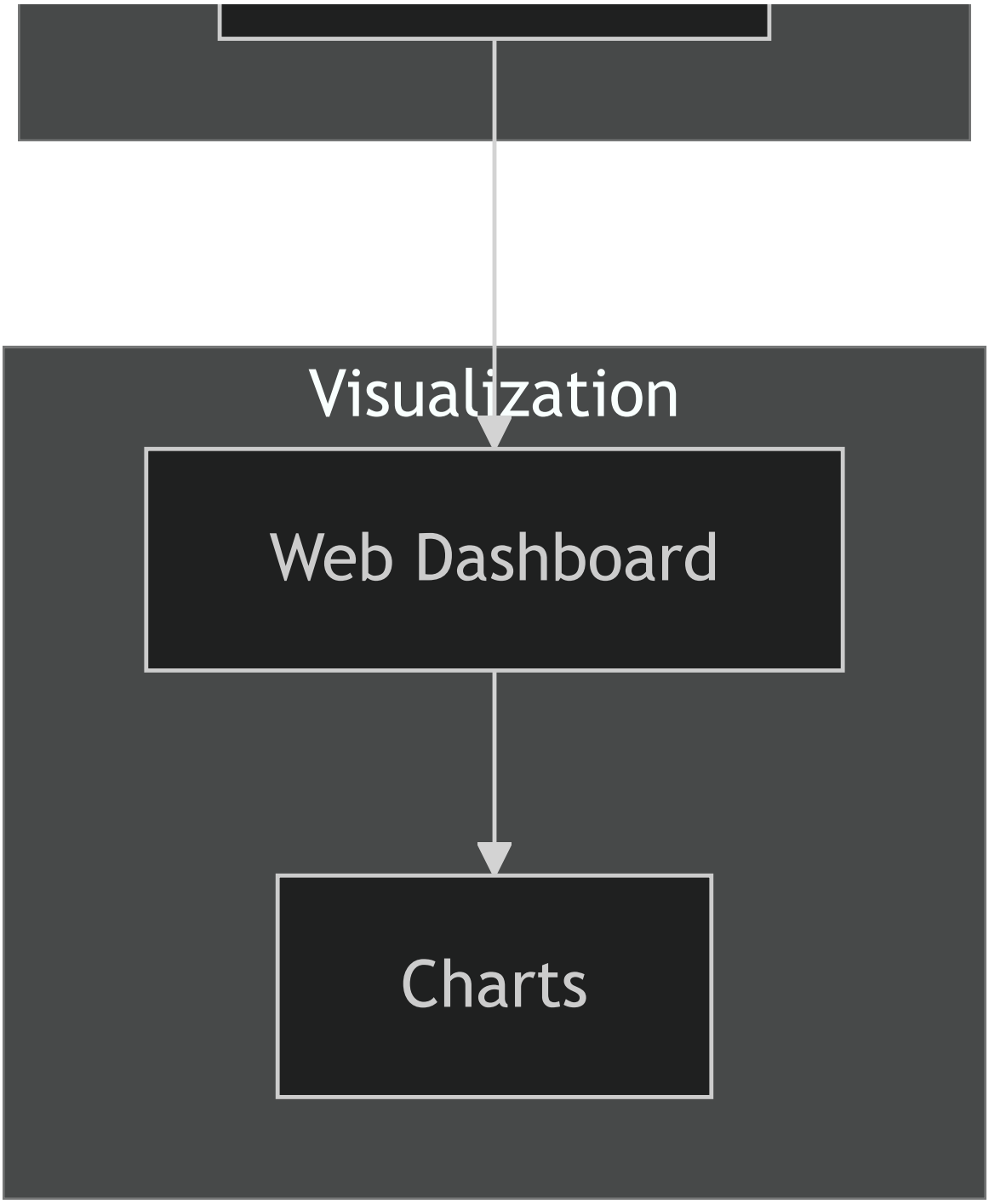


System Architecture

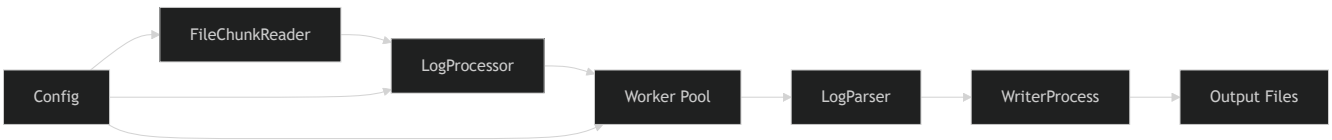
High-Level Architecture


Input Layer
Log Files
File Reader
Processing Layer
Chunk Reader

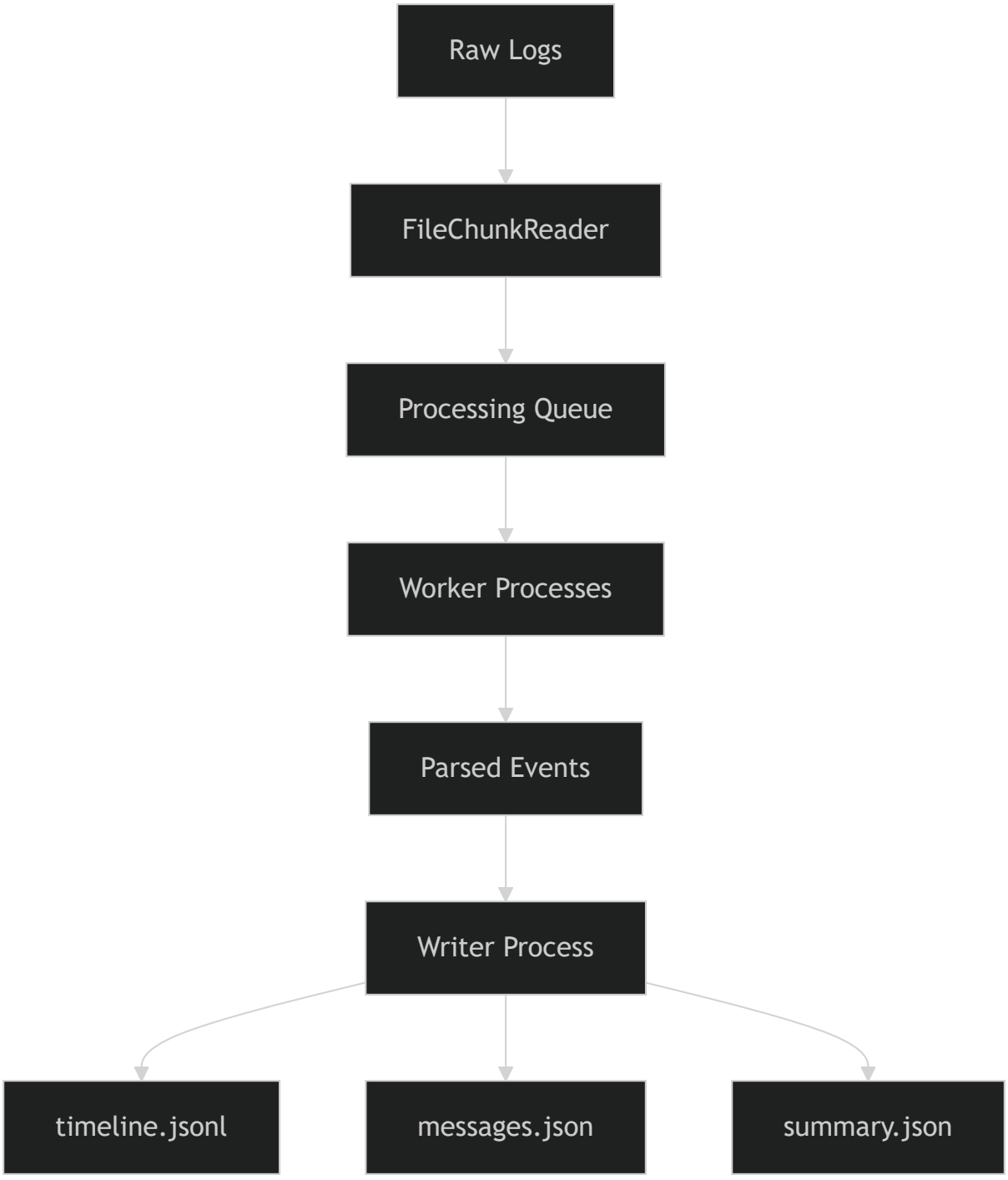

Worker Pool
Log Parser
Output Layer
Writer Process
JSON Files



Core Components

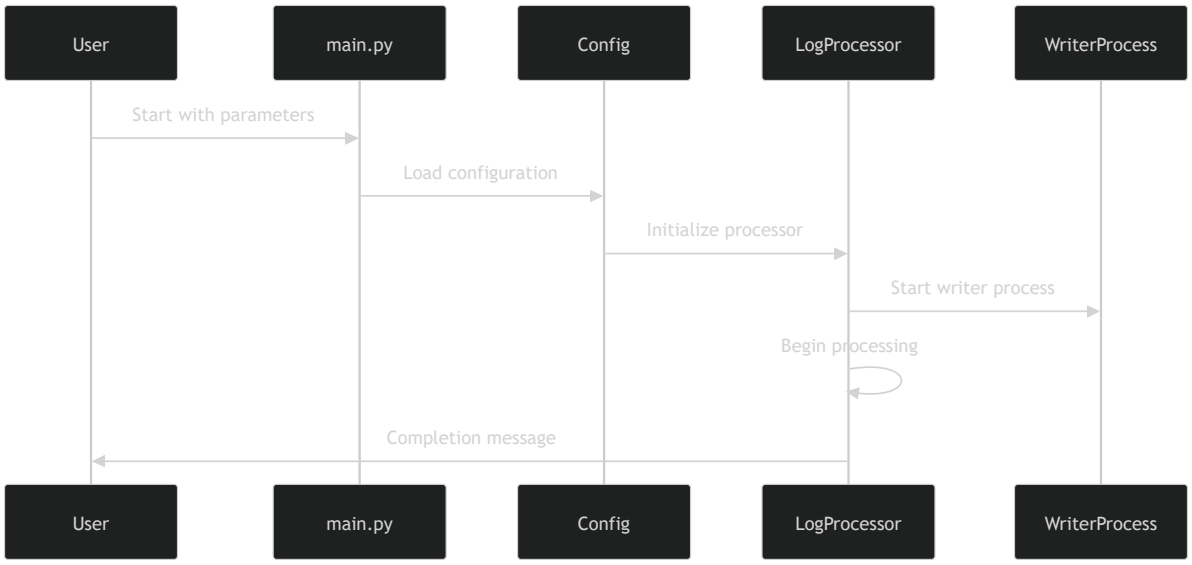


Data Flow Components

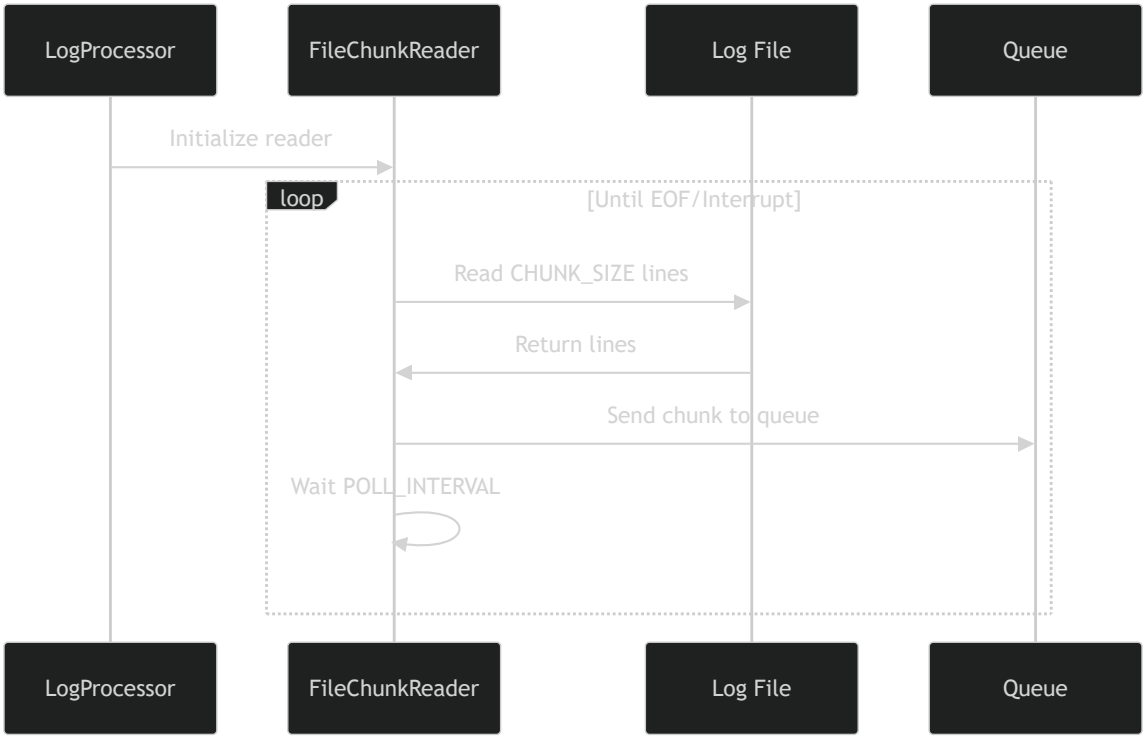


Application Flow

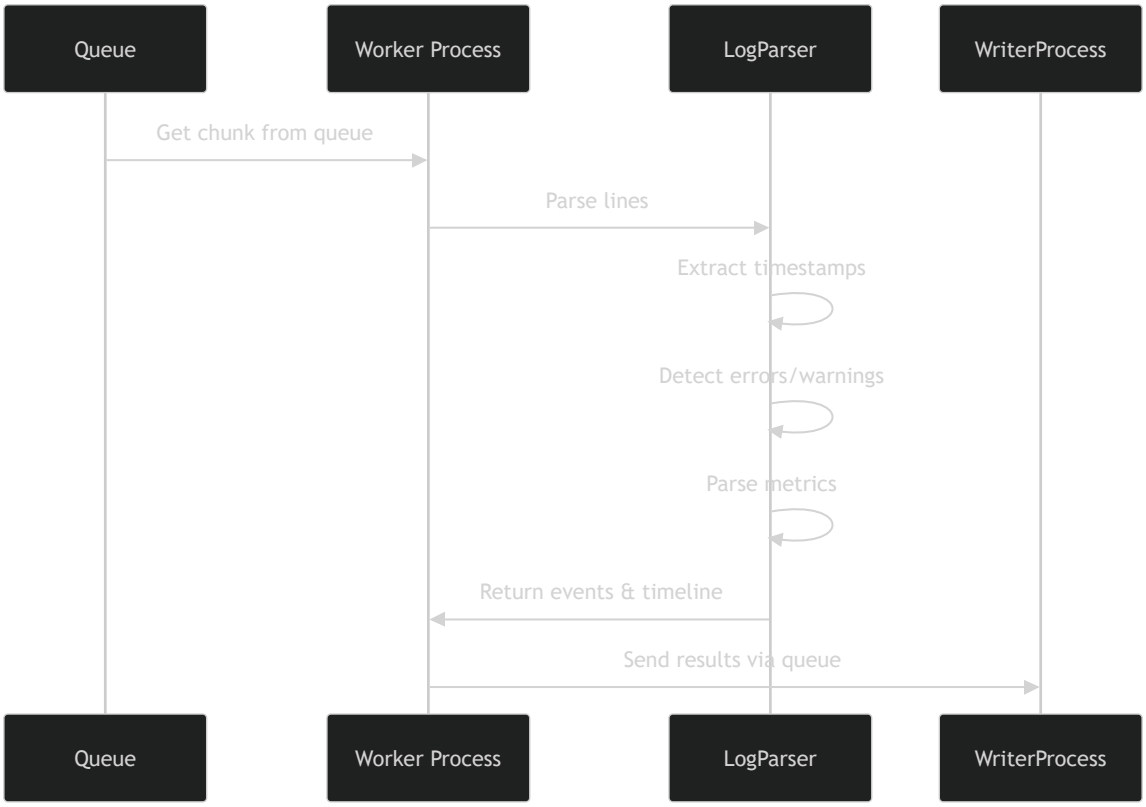
Main Application Flow



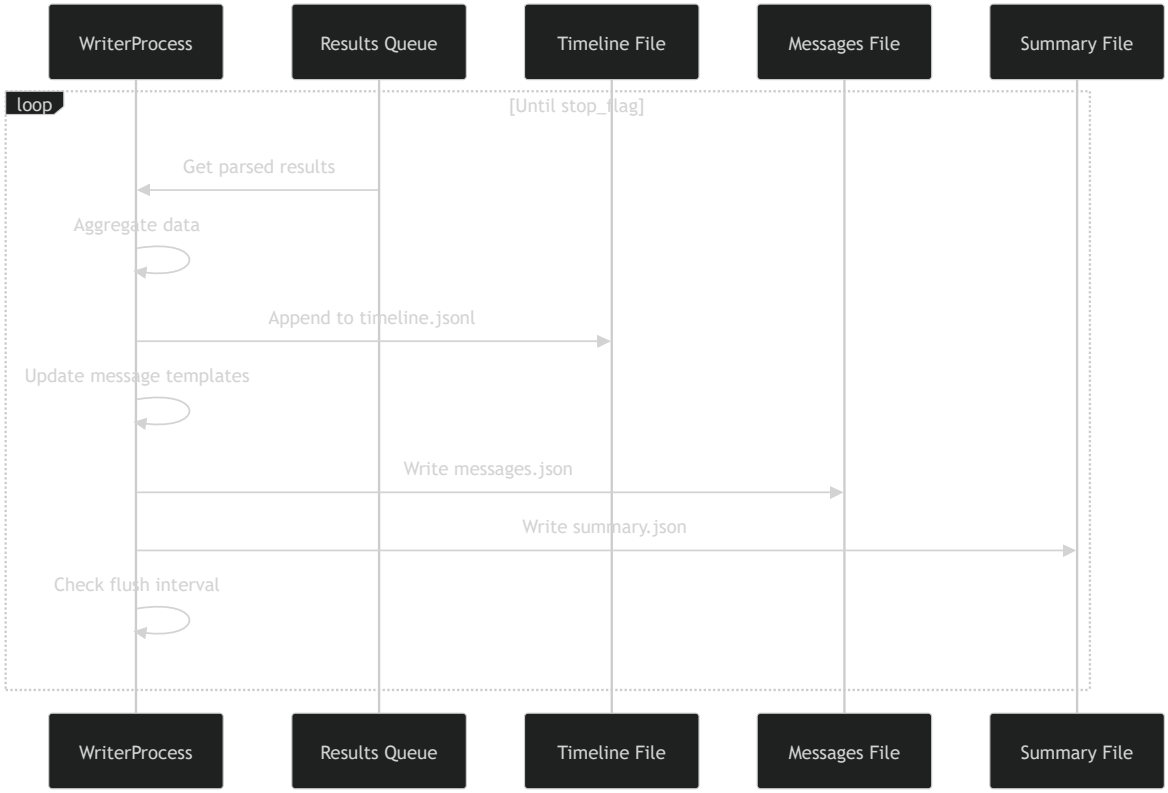
File Reading Flow



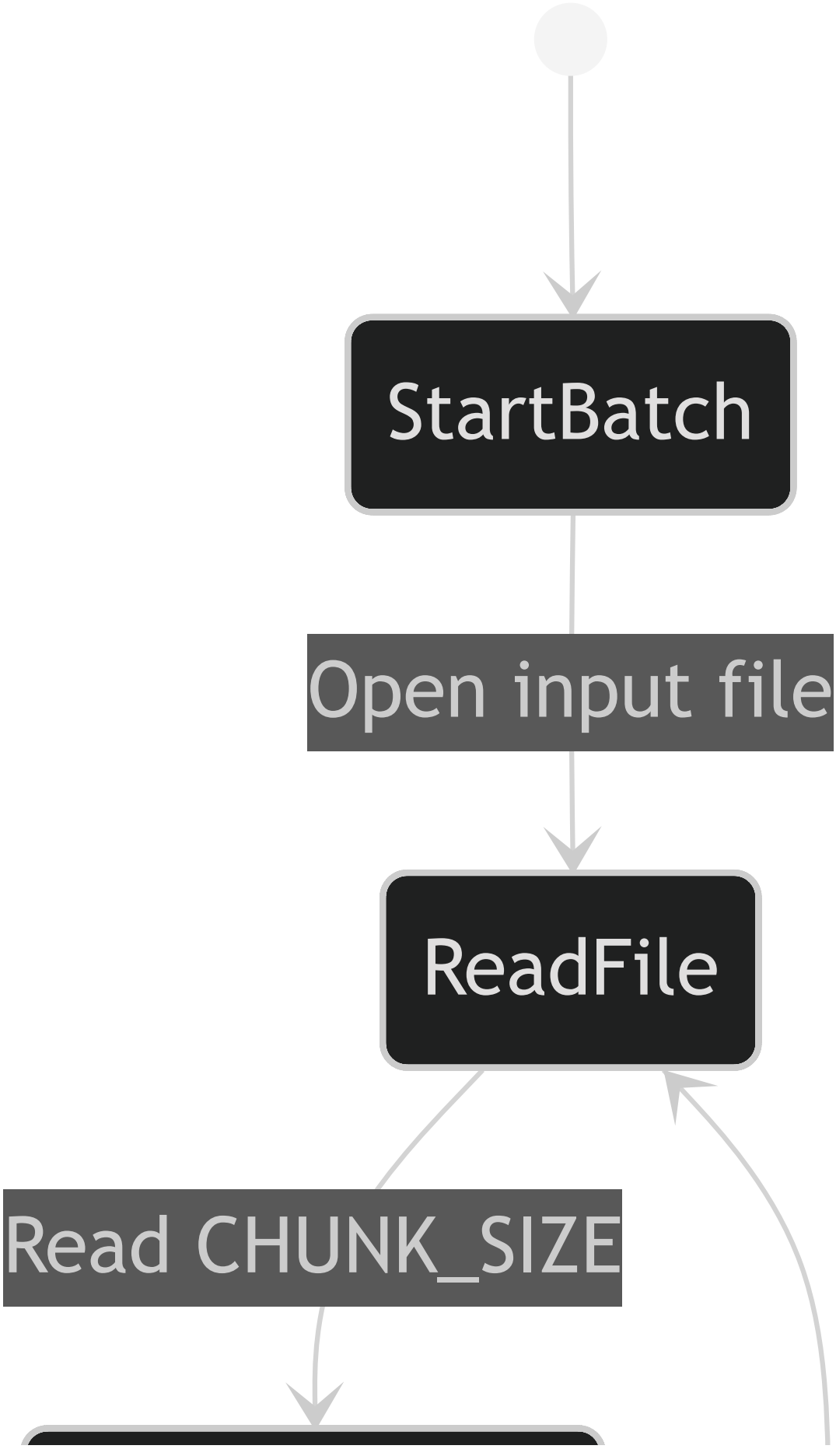
Worker Processing Flow

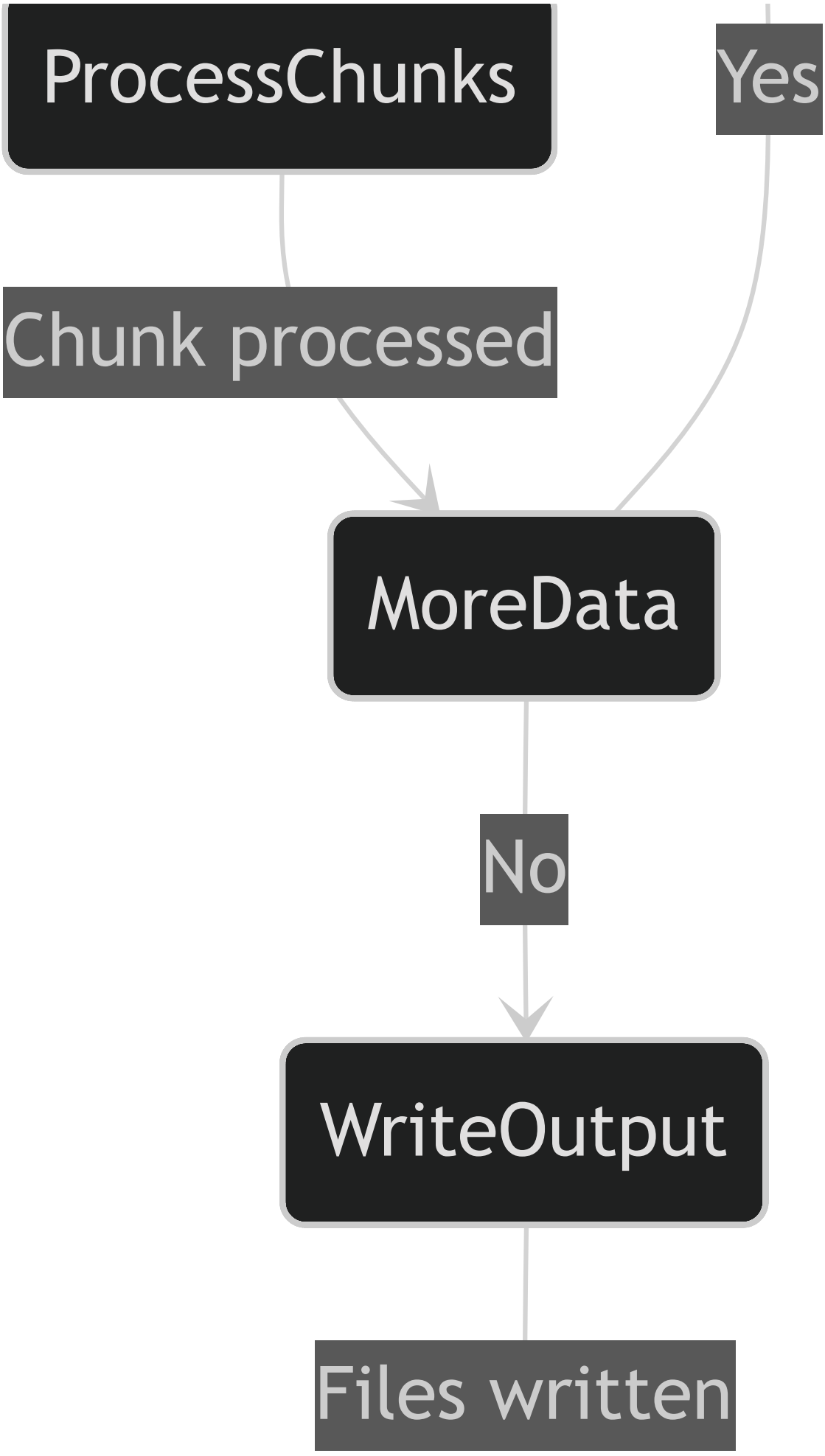


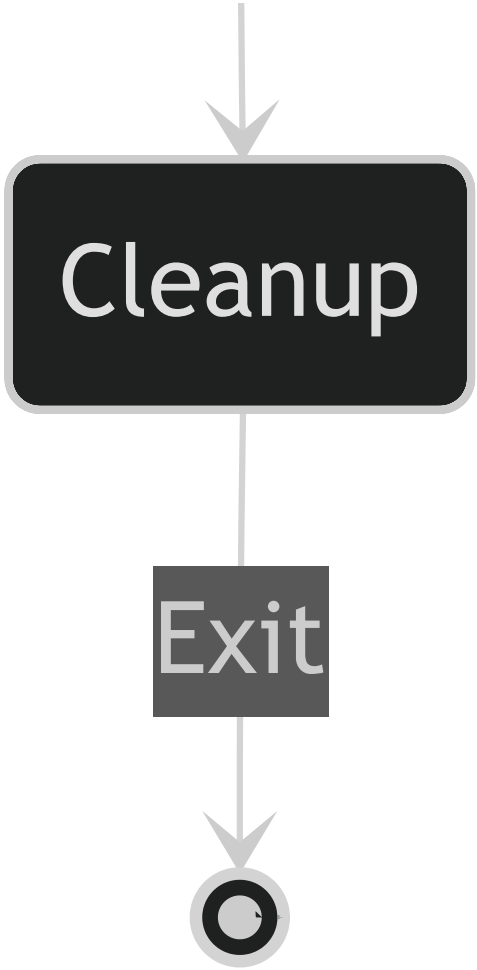
Writing Output Flow



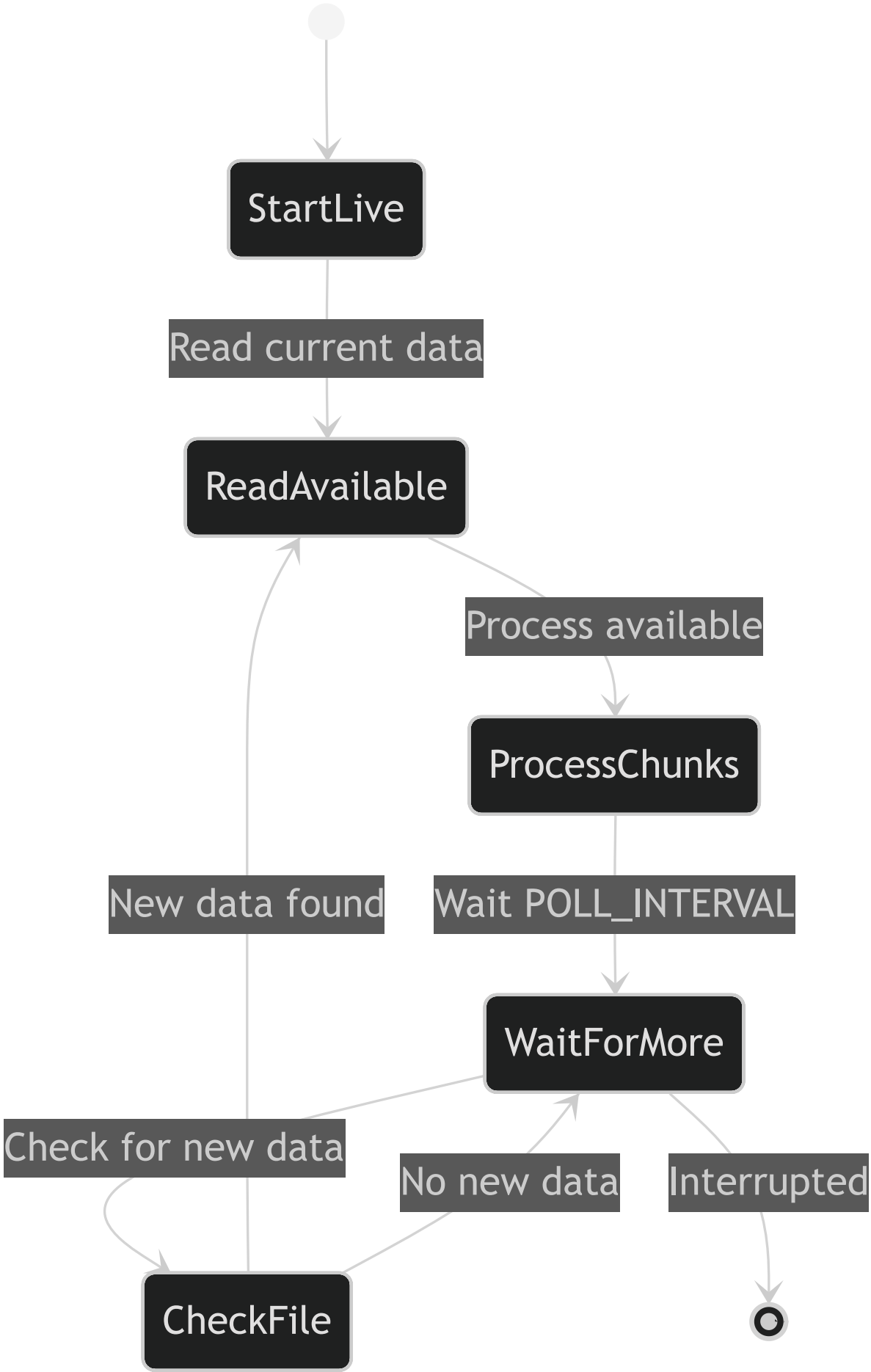
Batch Mode Flow


StartBatch
Open input file
ReadFile
Read CHUNK_SIZE


ProcessChunks
Chunk processed
MoreData
Yes
No
WriteOutput
Files written



Live Mode Flow


StartLive
Read current data
ReadAvailable
Process available
ProcessChunks
Wait POLL_INTERVAL
WaitForMore
Interrupted
No new data
Check for new data
New data found
CheckFile

Interfaces & Dependencies

Third-Party Libraries

```
# requirements.txt
flask==latest    # Web dashboard
```

External Services

- None - standalone application

System Requirements

- Python 3.9+
- Operating System: Windows/Linux/macOS
- Storage: Sufficient space for log files and output

Legal & Licensing

License Information

- **License**: MIT License
- **Copyright**: 2025 LostSoul
- **Permissions**: Commercial use, modification, distribution
- **Conditions**: Include original license
- **Limitations**: No warranty

Copyright Notice

All original code is MIT licensed. Third-party libraries maintain their respective licenses.

Configuration

Environment Variables

```
# .env.example
INPUT_FILE_PATH=    # Path to input log file
CHUNK_SIZE=1000     # Lines per processing chunk
QUEUE_MAX_SIZE=10   # Max chunks in queue
POLL_INTERVAL=0.5   # Seconds between file checks
```

```
NUM_PROCESSES=3      # Worker processes count
OUTPUT_PATH=         # Output directory path
```

Configuration Files

- **.env**: Environment variables (optional)
- **config.py**: Central configuration management

Installation & Setup

Quick Start

```
# 1. Generate sample log
python bin/generate_sample_log.py

# 2. Analyze logs (batch mode)
python src/main.py

# 3. Start web dashboard
python vendor/frontend_demo/serve.py
```

Detailed Installation

1. **Prerequisites**: Python 3.9+ in PATH
2. **Dependencies**: `pip install -r requirements.txt`
3. **Project Structure**:

```
ChronoLog/
├── src/           # Core application
├── tests/         # Unit tests
├── input/         # Log files
├── output/        # Analysis results
├── vendor/        # Web dashboard
└── bin/           # Utilities
```

Error Handling

Common Error States

Error Type	Cause	Resolution
FileNotFound	Input file missing	Check INPUT_FILE_PATH
PermissionError	File access denied	Adjust file permissions

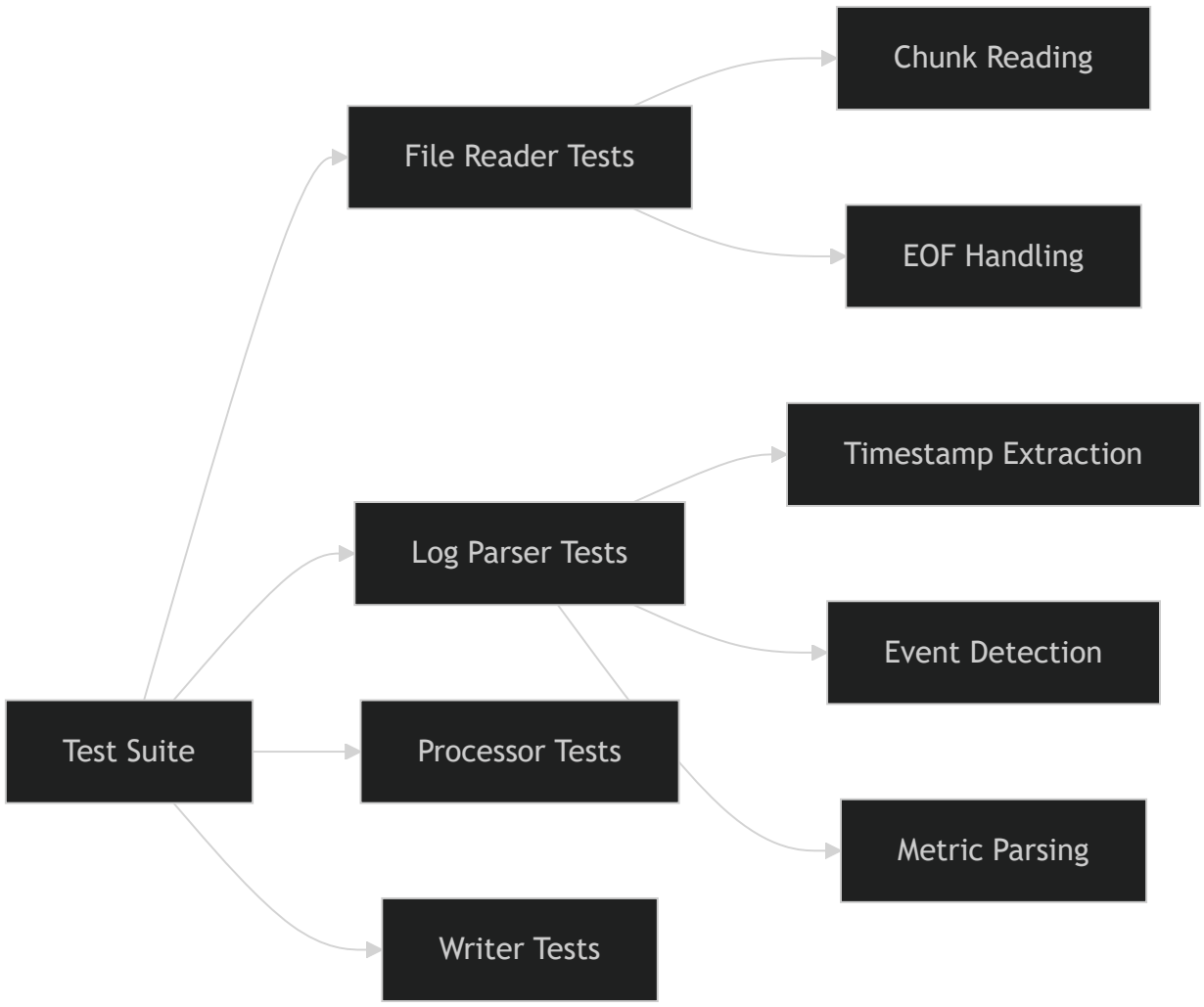
Error Type	Cause	Resolution
MemoryError	Queue overflow	Reduce CHUNK_SIZE or QUEUE_MAX_SIZE
ParserError	Malformed log lines	Check log format

Error Recovery

- **Automatic**: Queue timeouts and retries
- **Manual**: Configuration adjustment
- **Fallback**: Default file paths

Testing & Validation

Test Structure



Running Tests

```
# Method 1: Using unittest discovery
set PYTHONPATH=%CD%\src
python -m unittest discover -s tests -v

# Method 2: Using test runner
python tests/run_all_tests.py
```

Test Coverage

- **Unit Tests**: Individual component functionality
- **Integration Tests**: Component interactions
- **Performance Tests**: Large file processing

Versioning & Known Issues

Version History

- Current: Initial release
- Features: Batch/live processing, web dashboard

Known Limitations

- Log format assumes specific timestamp pattern
- Memory usage scales with queue size
- Web dashboard requires Flask installation

Data Schema

Output Files Structure

timeline.jsonl

```
{
  "time": "2025-11-23T12:00:00",
  "event": "error",
  "msg_id": 1,
  "msg_values": ["123"]
}
```

messages.json

```
{
  "id": 1,
  "template": "ERROR Database connection failed for user {num}"
}
```

summary.json

```
{
  "summary": {
    "error_count": 45,
    "warning_count": 120,
    "metrics": {
      "latency": {
        "count": 1000,
        "average": 275.5
      }
    }
  },
  "timeline_count": 50000,
  "unique_messages": 25
}
```

Network Configuration

Web Dashboard

- **Port**: 8000
- **Host**: 127.0.0.1 (localhost)
- **Protocol**: HTTP
- **Endpoints**:
  - `/` - Main dashboard
  - `/summary` - Statistics
  - `/messages` - Message templates
  - `/timeline` - Paginated events
  - `/timeseries` - Metric data

Performance Optimization

Tuning Parameters

```
# For large files (>1GB)
CHUNK_SIZE=5000
```

```
NUM_PROCESSES=4
QUEUE_MAX_SIZE=50

# For memory-constrained systems
CHUNK_SIZE=1000
QUEUE_MAX_SIZE=10
```

Best Practices

- Use SSD storage for large log files
- Monitor memory usage during processing
- Adjust workers based on CPU cores
- Use batch mode for historical analysis
- Use live mode for real-time monitoring

Maintenance

Cleanup

```
# Clear input/output directories
python bin/util_clear_dirs.py
```

Monitoring

- Check output file sizes
- Monitor processing queue
- Validate JSON output format
- Review error counts in summary