UFR de mathématique et d'informatique
Université de Strasbourg

MASTER'S DEGREE IN COMPUTER SCIENCE
SCIENCE AND ENGINEERING OF NETWORKS, INTERNET, AND SYSTEMS

Defense presented by

Lucian MOCAN

Research Project (TER)

COMPILATION AND INTERPRETATION OF FUNCTIONS IN THE ALTHREAD LANGUAGE

May 15, 2025

Supervised by

Quentin BRAMAS

Agenda

1. Problem Context and Importance
2. Adding User-Defined Functions to Althread
3. Conclusion

1. Problem Context and Importance

Althread

- Educational programming language (University of Strasbourg)
- Written in Rust
- Model and verify distributed systems [1]
  - shared variables
  - processes
  - channels
  - always/never conditions
- Familiar syntax

```
shared { // block containing all global variables
    let Start = false; // synchronizes processes
}
program A() {   // program template A
    wait Start; // waits until Start == true
    // waits on the process' input channel
    wait receive in (x,y) => {
        print("received ", x, " ", y);
    };
}
main {
    // starts a process with program template A
    let pa = run A();
    let pb = run A();
    // creates and links an output channel to
    // the input channel of the process
    channel self.out1 (int, bool)> pa.in;
    channel self.out2 (int, bool)> pb.in;
    Start = true;
    send out (125, true); // send in the channel
    send out2 (125, false);
}
```

[1] Althread. Introduction to Althread Guide. 2025.
URL: https://althread.github.io/en/docs/guide/intro/

1. Problem Context and Importance

Althread

- Educational programming language (University of Strasbourg)
- Written in Rust
- **Model and verify distributed systems** [1]
  - shared variables
  - processes
  - channels
  - always/never conditions
- Familiar syntax

```
shared { // block containing all global variables
    let Start = false; // synchronizes processes
}
program A() {   // program template A
    wait Start; // waits until Start == true
    // waits on the process' input channel
    wait receive in (x,y) => {
        print("received ", x, " ", y);
    };
}
main {
    // starts a process with program template A
    let pa = run A();
    let pb = run A();
    // creates and links an output channel to
    // the input channel of the process
    channel self.out1 (int, bool)> pa.in;
    channel self.out2 (int, bool)> pb.in;
    Start = true;
    send out (125, true); // send in the channel
    send out2 (125, false);
}
```

[1] Althread. Introduction to Althread Guide. 2025.
URL: https://althread.github.io/en/docs/guide/intro/

1. Problem Context and Importance

Distributed systems

- Benefits of well-designed distributed systems [2]:
    - **resource sharing**
    - **dependability**
    - **scalability**

However…

[2] Maarten van Steen and Andrew S. Tanenbaum. Distributed Systems 4th edition. 2025. URL: https://www.distributed-systems.net/index.php/books/ds4/

1. Problem Context and Importance

Distributed systems

- Benefits of well-designed distributed systems [2]:
  - **resource sharing**
  - **dependability**
  - **scalability**

- Distributed systems design is complex [3][4]:
  - **no** global state
  - **no** global time-frame
  - **no** main coordinator
  - **no**n-determinism

[2] Maarten van Steen and Andrew S. Tanenbaum. Distributed Systems 4th edition. 2025. URL: https://www.distributed-systems.net/index.php/books/ds4/
[3] Gerard Tel. Introduction to Distributed Algorithms. 2nd ed. Cambridge University Press, 2000.
[4] Nancy A. Lynch. Distributed Algorithms. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1996. ISBN: 978-1-55860-348-6.

1. Problem Context and Importance

Solution: Models and Design Verification

"To a first approximation, we can say that accidents are almost always the result of incorrect estimates of the likelihood of one or more things." - C. Michael Holloway, NASA [5]

[5] Holloway, C. Michael. Why You Should Read Accident Reports. Presented at Software and Complex Electronic Hardware Standardization Conference, July 2005.

1. Problem Context and Importance

Solution: Models and Design Verification

- Formal specification
- Detect flaws **early**
  - implementation-level testing is not enough
  - save resources : time, money
  - improved time-to-market [6]
- Get the advantages of a well-designed distributed system

[6] Chris Newcombe et al. Formal Methods at Amazon Web Services. Tech. rep. Amazon Web Services, 2015. URL: https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf

1. Problem Context and Importance

Solution: Models and Design Verification

- **CSP** (Communicating Sequential Processes) used for the International Space Station systems in 1999 [7]

- **The model checker SPIN** (PROMELA) [8] used for the "Distributed Systems" course at the University of Strasbourg

- **TLA+** (PlusCal) used by Intel, Microsoft and Amazon [9]

However…

[7] Jan Peleska and Bettina Buth. "Formal Methods for the International Space Station ISS". In: Correct System Design: Recent Insights and Advances. Ed. by Ernst-Rüdiger Olderog and Bernhard Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 363–389. ISBN: 978-3-540-48092-1. DOI: 10.1007/3-540-48092-7_16. URL: https://doi.org/10.1007/3-540-48092-7_16.
[8] Gerard J. Holzmann. "The model checker SPIN". In: IEEE Transactions on Software Engineering 23.5 (1997). URL: https://spinroot.com/spin/Doc/ieee97.pdf.
[9] Leslie Lamport. Industrial Use of TLA+. https://lamport.azurewebsites.net/tla/industrial-use.html.

1. Problem Context and Importance

Solution: Models and Design Verification

- **CSP** (Communicating Sequential Processes) used for the International Space Station systems in 1999 [7]
- **The model checker SPIN** (PROMELA) [8] used for the "Distributed Systems" course at the University of Strasbourg
- **TLA+** (PlusCal) used by Intel, Microsoft and Amazon [9]

- **Unusual and confusing syntax for students**
- **Difficult to setup**
- **Not education-oriented**

[7] Jan Peleska and Bettina Buth. "Formal Methods for the International Space Station ISS". In: Correct System Design: Recent Insights and Advances. Ed. by Ernst-Rüdiger Olderog and Bernhard Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 363–389. ISBN: 978-3-540-48092-1. DOI: 10.1007/3-540-48092-7_16. URL: https://doi.org/10.1007/3-540-48092-7_16.
[8] Gerard J. Holzmann. "The model checker SPIN". In: IEEE Transactions on Software Engineering 23.5 (1997). URL: https://spinroot.com/spin/Doc/ieee97.pdf.
[9] Leslie Lamport. Industrial Use of TLA+. https://lamport.azurewebsites.net/tla/industrial-use.html.

1. Problem Context and Importance

Althread

- Known design languages are modular (modules, inline functions)
- No modularity (apart from process templates)
- Built-in functions (print, methods for lists) [10]
- Limited reutilisability and readability
- Limited educational effect

[10] Althread Project. Althread Guide: Using Programs. 2025. URL: https://althread.github.io/docs/guide/program/simple-process.

2. Adding User-Defined Functions to Althread

Syntax

- C/Rust inspired

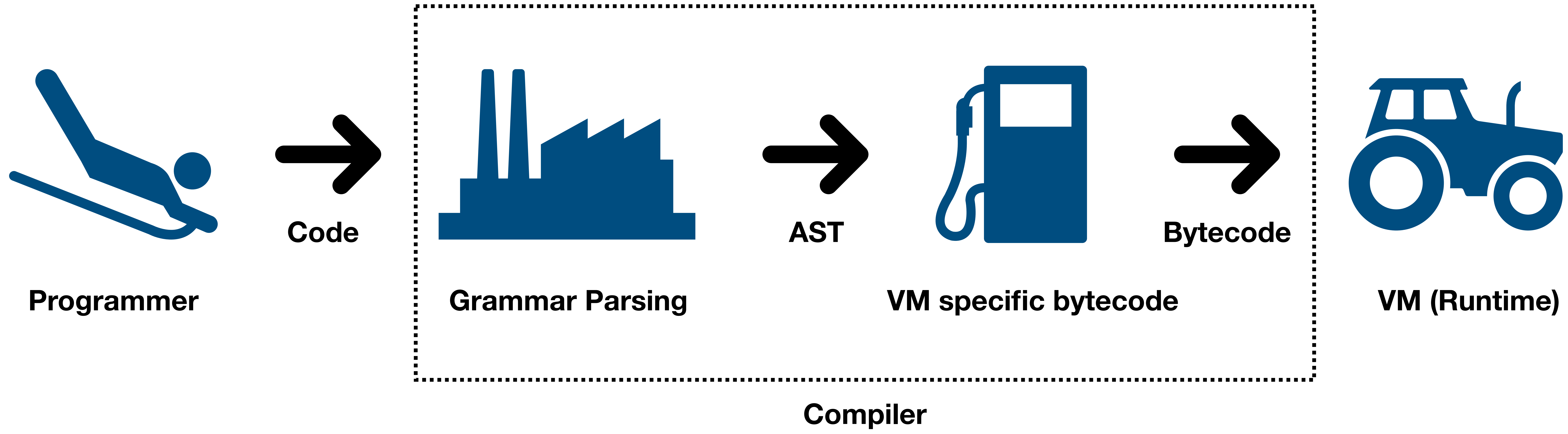
```
fn <function_name>
  (<param1>: <type1>, <param2>: <type2>) -> <return_type>
{
    <statements>;
    return <expression>;
}
```

- Familiarity

- Intuitive function output notation

2. Adding User-Defined Functions to Althread

Compilation Pipeline



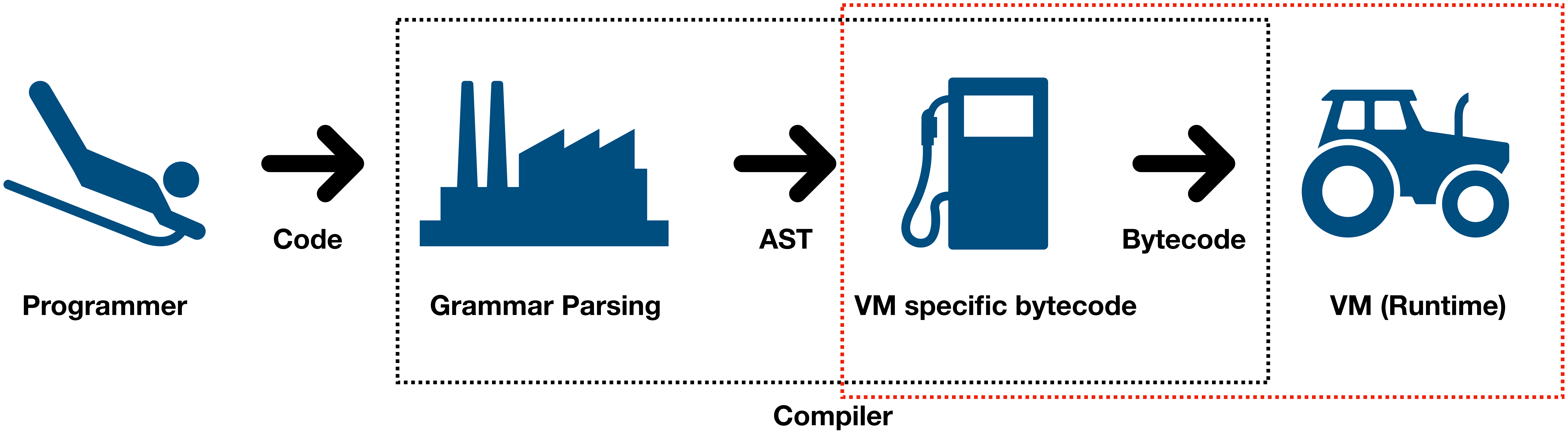
AST = Abstract Syntax Tree

VM = Virtual Machine

Bytecode = Instructions that the VM understands

2. Adding User-Defined Functions to Althread

Compilation Pipeline



AST = Abstract Syntax Tree

VM = Virtual Machine

Bytecode = Instructions that the VM understands

2. Adding User-Defined Functions to Althread

Function Definition

- Global knowledge of existing functions
- Call depth
- Valid block of instructions

```rust
pub struct CompilerState {
    pub global_table: HashMap<String, Variable>,
    pub program_stack: Vec<Variable>,
    pub current_stack_depth: usize,
    // other fields
    pub in_function: bool,
    pub user_functions:
            HashMap<String, FunctionDefinition>
}
```

Modified Althread source code. Filepath: interpreter/compiler/mod.rs
URL: https://github.com/lucianmocan/althread

2. Adding User-Defined Functions to Althread

Function Definition

```
pub struct FunctionDefinition {
    pub name: String,
    pub arguments: Vec<(Identifier, DataType)>,
    pub return_type: DataType,
    pub body: Vec<Instruction>,
    pub pos: Pos,
}
```

Modified Althread source code. Filepath: interpreter/compiler/mod.rs
URL: https://github.com/lucianmocan/althread

2. Adding User-Defined Functions to Althread

Function Call Instruction in the VM

1. Push arguments in memory
2. Switch to the function’s context
3. Execute and return
4. Switch back to the calling context

2. Adding User-Defined Functions to Althread

Function Call Instruction in the VM

1. Push arguments in memory
2. Switch to the function's context
3. Execute and return
4. Switch back to the calling context

```
struct StackFrame<'a> {
    return_ip: usize, // the instruction pointer to return to
    caller_fp: usize, // the frame pointer of the caller
    caller_code: &'a [Instruction], // the code of the caller
    expected_return_type: DataType
}
```

Modified Althread source code. Filepath: interpreter/vm/running_program.rs
URL: https://github.com/lucianmocan/althread

2. Adding User-Defined Functions to Althread

Results

```
fn fibonacci_recursive(n: int, a: int, b: int) -> int {
  if n == 0 {
    return a;
  } else {
    return fibonacci_recursive(n - 1, b, a + b);
  }
}

main {
    let n = 10;
    let res = fibonacci_recursive(n, 0, 1);
    print("Fibonacci recursive of " + n + ": " + res);
}

// Outputs:
// Fibonacci recursive of 10: 55
```

2. Adding User-Defined Functions to Althread

Results

```
shared {
    let Counter: int = 0;
}
fn increment_and_get() -> int {
    Counter = Counter + 1; // racing condition
    return Counter;
}
program Worker() {
    let c = increment_and_get();
    print("Worker " + c);
}
main {
    atomic {
        run Worker();
        run Worker();
    }
}
```

```
// Possible outputs
Worker 1
Worker 2

Worker 1
Worker 1
```

2. Adding User-Defined Functions to Althread

Results

```
shared {
    let Counter: int = 0;
}
fn increment_and_get() -> int {
  atomic {
    Counter = Counter + 1;
    return Counter;
  }
}
program Worker() {
    let c = increment_and_get();
    print("Worker " + c);
}
main {
    atomic {
        run Worker();
        run Worker();
    }
}
```

```
// Always outputs
Worker 1
Worker 2
```

3. Conclusion

At the moment

- Working user-defined functions
  - correctly integrated with most of Althread's existing features
  - function signature checks:
    - arguments' types and count check
    - return value type check
  - support for advanced functionalities:
    - recursion
    - nested function calls
- Extensive testing is required
- Unfinished work on function calls as expressions (e.g. *max(max(1,2), 3)*)

3. Conclusion

Future perspectives

- Check return coverage
- Add further modularity with function imports / modules
- Lambda functions, pattern matching ?
- Fix function unrelated existing grammar issues
- Documentation

References

[1] Althread. Introduction to Althread Guide. 2025. URL: https://althread.github.io/en/docs/guide/intro/

[2] Maarten van Steen and Andrew S. Tanenbaum. Distributed Systems 4th edition. 2025. URL: https://www.distributed-systems.net/index.php/books/ds4/

[3] Gerard Tel. Introduction to Distributed Algorithms. 2nd ed. Cambridge University Press, 2000.

[4] Nancy A. Lynch. Distributed Algorithms. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1996. ISBN: 978-1-55860-348-6.

[5] Holloway, C. Michael. Why You Should Read Accident Reports. Presented at Software and Complex Electronic Hardware Standardization Conference, July 2005.

[6] Chris Newcombe et al. Formal Methods at Amazon Web Services. Tech. rep. Amazon Web Services, 2015. URL: https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf

[7] Jan Peleska and Bettina Buth. "Formal Methods for the International Space Station ISS". In: Correct System Design: Recent Insights and Advances. Ed. by Ernst-Rüdiger Olderog and Bernhard Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 363–389. ISBN: 978-3-540-48092-1. DOI: 10.1007/3-540-48092-7_16. URL: https://doi.org/10.1007/3-540-48092-7_16.

[8] Gerard J. Holzmann. "The model checker SPIN". In: IEEE Transactions on Software Engineering 23.5 (1997). URL: https://spinroot.com/spin/Doc/ieee97.pdf.

[9] Leslie Lamport. Industrial Use of TLA+. https://lamport.azurewebsites.net/tla/industrial-use.html.

[10] Althread Project. Althread Guide: Using Programs. 2025. URL: https://althread.github.io/docs/guide/program/simple-process.