On Deductive Verification of an Industrial Concurrent Software Component with VerCors

Raúl E. Monti, **Robert**[1] **Rubbens**, Marieke Huisman

FMT - University of Twente

October 24, 2022

UNIVERSITY OF TWENTE.

[1] (Bob)

Previous presentations

1. “On the Industrial Application of Critical Software Verification with VerCors” (Huisman & Monti, 2020)
2. **“On Deductive Verification of an Industrial Concurrent Software Component with VerCors”** (Monti, Rubbens & Huisman, 2022)

Software Verification

- Systems are growing more complex
- Usage of concurrency is growing
- Traditional method for safety: testing

Software Verification

- Systems are growing more complex
- Usage of concurrency is growing
- Traditional method for safety: testing
- Problem: concurrency $\longrightarrow$ exponential interleavings

Testing shows the presence, not the absence of bugs

(Edsger W. Dijkstra)

$\Longrightarrow$ Software verification

Outline

1 Tunnel System & Architecture

2 VerCors

3 Concurrent Data Manager Verification

4 Presentation at Technolution & Results

Tunnel System & Architecture

About the Tunnel

- Blankenburgverbinding, A24 (connects Vlaardingen & Rozenburg)
- Software developed by Technolution
- We were asked to analyze during testing phase

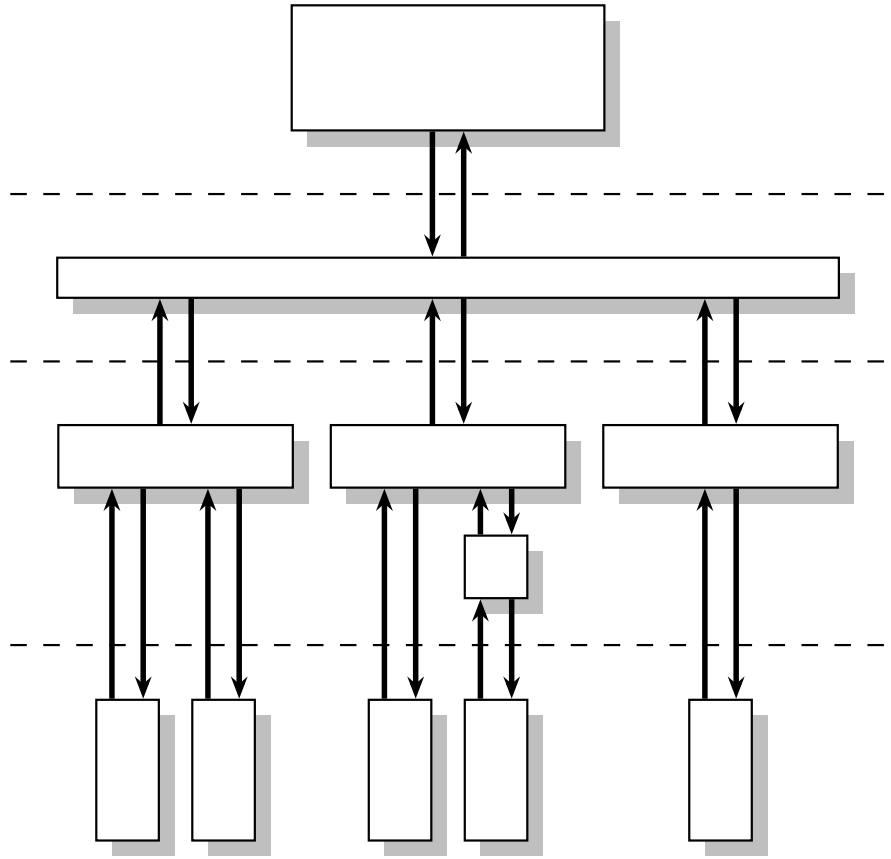
BSTTI

Operator

Man-Machine Interface (MMI)

Control, Instruct, Guard (3B)

Logical Function Fulfiller (LFV)



VerCors

Verification with VerCors

- Deductive verifier for concurrent Java, C, OpenCL, PVL
- Requires annotation of code with contracts

Verification with VerCors

- Deductive verifier for concurrent Java, C, OpenCL, PVL
- Requires annotation of code with contracts

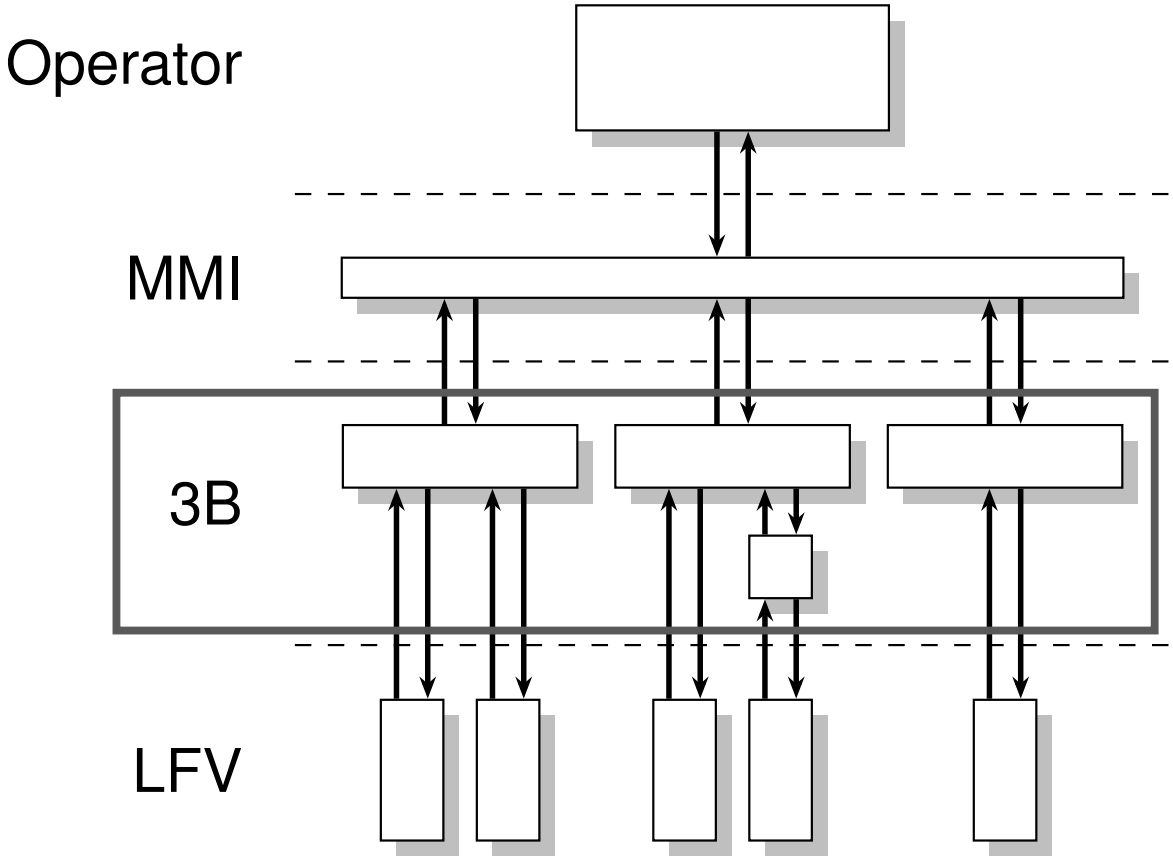
```
1 class Util {
2   //@ requires x > 0 && y > 0;
3   //@ ensures \result >= 2;
4   public static void add(int x, int y) {
5     return x + y;
6   }
7 }
```

Verification with VerCors: Concurrency

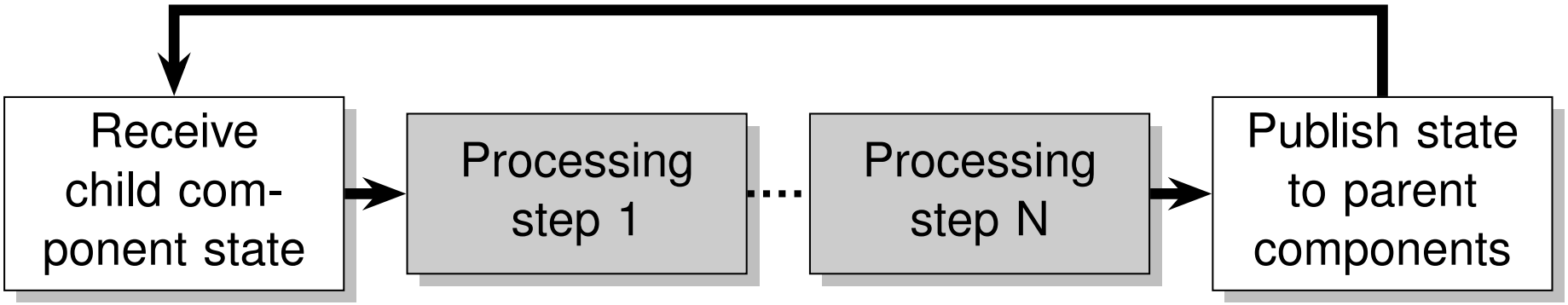
```
1  class Util {
2     int total;
3
4     //@ requires Perm(total, write);
5     //@ ensures Perm(total, write);
6     public static void add(int x) {
7        total += x;
8     }
9  }
```

Concurrent Data Manager Verification

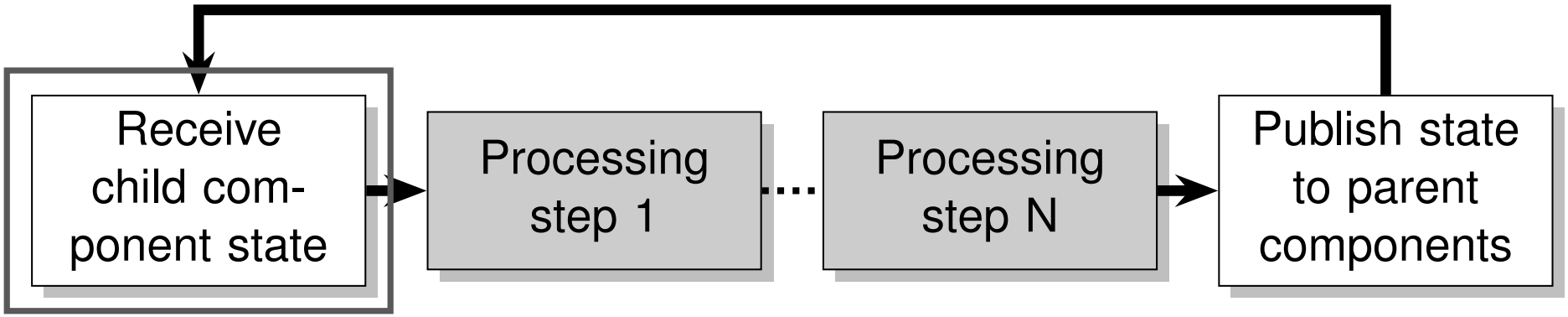
Context



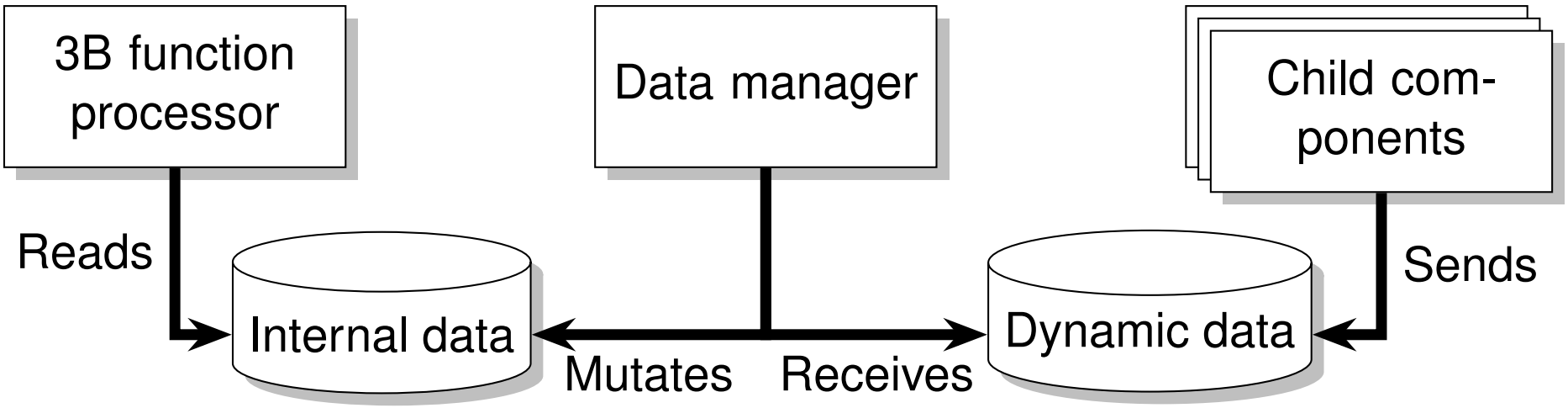
3B: Lifecycle



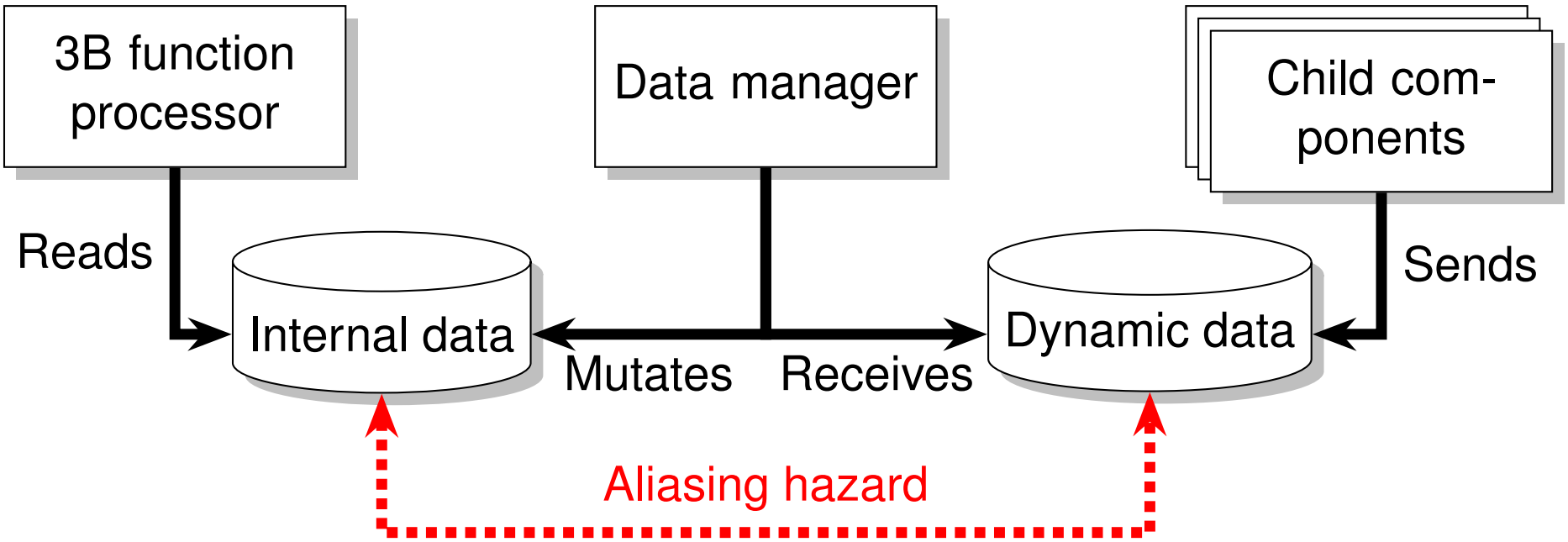
3B: Lifecycle



3B: State Management



3B: State Management



Detecting with VerCors: code

```
1  class Manager {
2      Data internal;
3      Data dynamic;
4      ...
5
6
7      void sync() {
8          ...
9          internal = dynamic;
10     }
11 }
```

Detecting with VerCors: code + annotations

```
1  class Manager {
2      Data internal;
3      Data dynamic;
4      ...
5      //@ ensures Perm(internal.value, write);
6      //@ ensures Perm(dynamic.value, write);
7      void sync() {
8          ...
9          internal = dynamic;
10     }
11 }
```

Detecting with VerCors: code + annotations

```
1  class Manager {
2      Data internal;
3      Data dynamic;
4      ...
5      //@ ensures Perm(internal.value, write);
6      //@ ensures Perm(dynamic.value, write);
7      void sync() {
8          ...
9          internal = dynamic;
10     }
11 }
```

Problem 2: private reference leak

```
1  class Manager {
2    private Data internal;
3    ...
4    synchronized Data get_internal() {
5      return internal;
6    }
7  }
8
9
10
```

Problem 2: private reference leak

```
1  class Manager {
2    private Data internal;
3    ...
4    synchronized Data get_internal() {
5      return internal;
6    }
7  }
8
9  Data d = get_internal(); // ok
10 int x = d.value;         // error!
```

Presentation at Technolution & Results

Presentation at Technolution: Iterate

- Define terms
- Narrow scope
- Present to smaller group

– Time consuming

“Good” formal methods presentation properties

- Introduce *only* key concepts of formalism

“Good” formal methods presentation properties

- Introduce *only* key concepts of formalism
- One example = one concept

“Good” formal methods presentation properties

- Introduce *only* key concepts of formalism
- One example = one concept
- Short examples

“Good” formal methods presentation properties

- Introduce *only* key concepts of formalism
- One example = one concept
- Short examples
- No unrelated concerns

“Good” formal methods presentation properties

- Introduce *only* key concepts of formalism
- One example = one concept
- Short examples
- No unrelated concerns
- Draw parallels to known concepts
  - But: avoid misunderstanding
  - E.g. no reuse of overloaded terms

Presentation results

- Testing vs. verification

Presentation results

- Testing vs. verification
- Annotation & specification culture

Presentation results

- Testing vs. verification
- Annotation & specification culture
- Similarities to Rust

Presentation results

- Testing vs. verification
- Annotation & specification culture
- Similarities to Rust
- Optimise for the common case

Presentation results

- Testing vs. verification
- Annotation & specification culture
- Similarities to Rust
- Optimise for the common case
- Library calls

Presentation results

- Testing vs. verification
- Annotation & specification culture
- Similarities to Rust
- Optimise for the common case
- Library calls
- Why not static analysis?

Conclusion

- Analysed: tunnel software component with VerCors
- Found: bug, weakness
- Presentation optimized to audience
- Determined properties of “good” formal methods presentations
- Collected presentation feedback

Future work:

- Reduce minimum required annotations
- Improve VerCors Java support