Path Expressions Revisited – Towards Compiler-enforced Reusable Synchronization Patterns

PLOS 2025

13th October 2025

Thomas Alexander Hövelmann
(thomas-alexander.hoevelmann@tu-dortmund.de)

Olaf Spinczyk
(olaf@uos.de)

Alexander Krause
(alexander.krause@cs.tu-dortmund.de)

Horst Schirmeier
(horst.schirmeier@tu-dresden.de)

Peter Ulbrich
(peter.ulbrich@tu-dortmund.de)

Technische Universität Dresden

technische universität dortmund

UNIVERSITÄT OSNABRÜCK

What are Path Expressions and what are they good for?

```
01::    int readcount = 0;
02::    semaphore x = 1, wsem = 1;
03::
04::    void reader() {
05::        semWait(x);
06::        readcount++;
07::        if (readcount == 1) semWait(wsem);
08::        semSignal(x);
09::        READUNIT();
10::        semWait(x);
11::        readcount--;
12::        if (readcount == 0) semSignal(wsem);
13::        semSignal(x);
14::    }
15::
16::    void writer() {
17::            semWait(wsem);
18::            WRITEUNIT();
19::            semSignal(wsem);
20::    }
```

On the left: excerpt from fig. 5.25 from Stallings, William: Operating Systems - Internals and Design Principles. 2017

What are Path Expressions and what are they good for?

```
01::    int readcount = 0;
02::    semaphore x = 1, wsem = 1;
03::
04::    void reader() {
05::        semWait(x);
06::        readcount++;
07::        if (readcount == 1) semWait(wsem);
08::        semSignal(x);
09::        READUNIT();
10::        semWait(x);
11::        readcount--;
12::        if (readcount == 0) semSignal(wsem);
13::        semSignal(x);
14::    }
15::
16::    void writer() {
17::            semWait(wsem);
18::            WRITEUNIT();
19::            semSignal(wsem);
20::    }
```

Functional code
Synchronization code

On the left: excerpt from fig. 5.25 from Stallings, William: Operating Systems - Internals and Design Principles. 2017

What are Path Expressions and what are they good for?

```
01::    int readcount = 0;
02::    semaphore x = 1, wsem = 1;
03::
04::    void reader() {
05::        semWait(x);
06::        readcount++;
07::        if (readcount == 1) semWait(wsem);
08::        semSignal(x);
09::        READUNIT();
10::        semWait(x);
11::        readcount--;
12::        if (readcount == 0) semSignal(wsem);
13::        semSignal(x);
14::    }
15::
16::    void writer() {
17::            semWait(wsem);
18::            WRITEUNIT();
19::            semSignal(wsem);
20::    }
```

```
01::    pathExpSynchronizer synch ( „( { reader } , writer )*“ );
02::
03::
04::    void reader() {
05::        synch.req(„reader“);
06::
07::
08::
09::        READUNIT();
10::        synch.term(„reader“);
11::
12::
13::
14::    }
15::
16::    void writer() {
17::            synch.req(„writer“);
18::            WRITEUNIT();
19::            synch.term(„writer“);
20::    }
```

Functional code
Synchronization code

On the left: excerpt from fig. 5.25 from Stallings, William: Operating Systems - Internals and Design Principles. 2017

What are Path Expressions and what are they good for?

```
01::    int readcount = 0;
02::    semaphore x = 1, wsem = 1;
03::
04::    void reader() {
05::        semWait(x);
06::        readcount++;
07::        if (readcount == 1) semWait(wsem);
08::        semSignal(x);
09::        READUNIT();
10::        semWait(x);
11::        readcount--;
12::        if (readcount == 0) semSignal(wsem);
13::        semSignal(x);
14::    }
15::
16::    void writer() {
17::            semWait(wsem);
18::            WRITEUNIT();
19::            semSignal(wsem);
20::    }
```

```
01::    @PE( „( { reader } , writer )*“ )
02::
03::    @PE-Synchronized( „reader“ )
04::    void reader() {
05::        READUNIT();
06::    }
07::
08::    @PE-Synchronized( „writer“ )
09::    void writer() {
10::        WRITEUNIT();
11::    }
12::
13::
14::
15::
16::
17::
18::
19::
20::
```

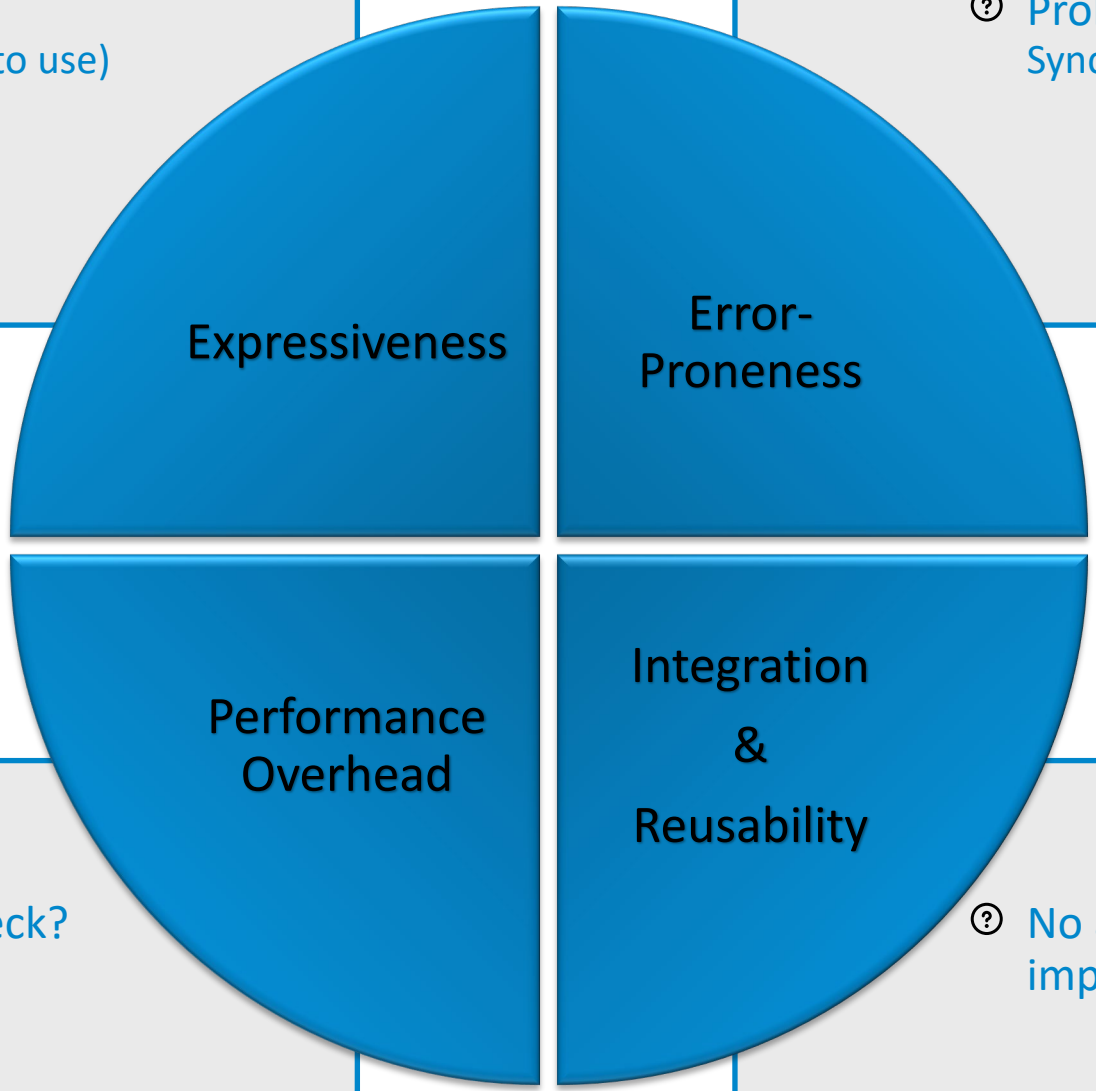
Functional code
Synchronization code

On the left: excerpt from fig. 5.25 from Stallings, William: Operating Systems - Internals and Design Principles. 2017

Why did Path Expressions not prevail?

- Not expressive enough?
  (cumbersome/hard/impossible to use)
- Too expressive?
  (… to verify)

- Problematic collateral evolution?
  Synchronization ↔ Functional code



- State machines as a bottleneck?

- No adequate tools in modern, imperative languages?

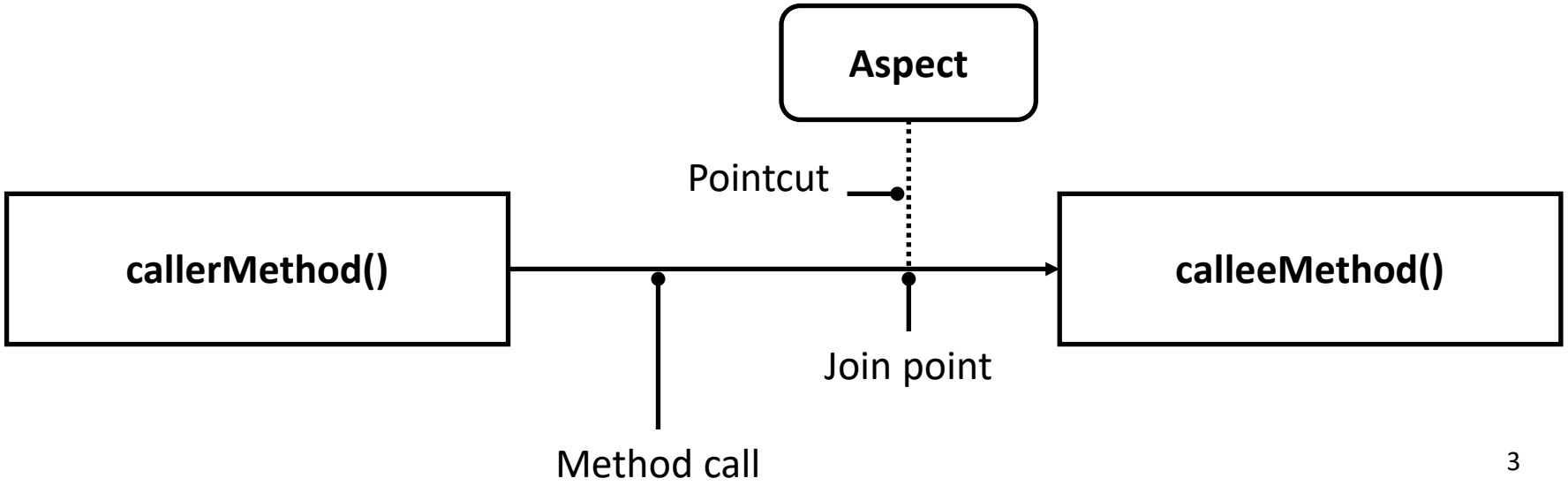
Step 1: A toy example from Path Pascal

- **First C++-prototype based on *Path Pascal*[1]**
  1. Implementation of Path Pascal in C++ based on templates

[1]R. H. Campbell, T. J. Miller. 1978. *A path Pascal Language.*

Step 1: A toy example from Path Pascal

- **First C++-prototype based on *Path Pascal*[1]**
  1. Implementation of Path Pascal in C++ based on templates
  2. Separation of Concerns:
     Utilization of Aspect-oriented Programming (AOP) using AspectC++[2] annotations

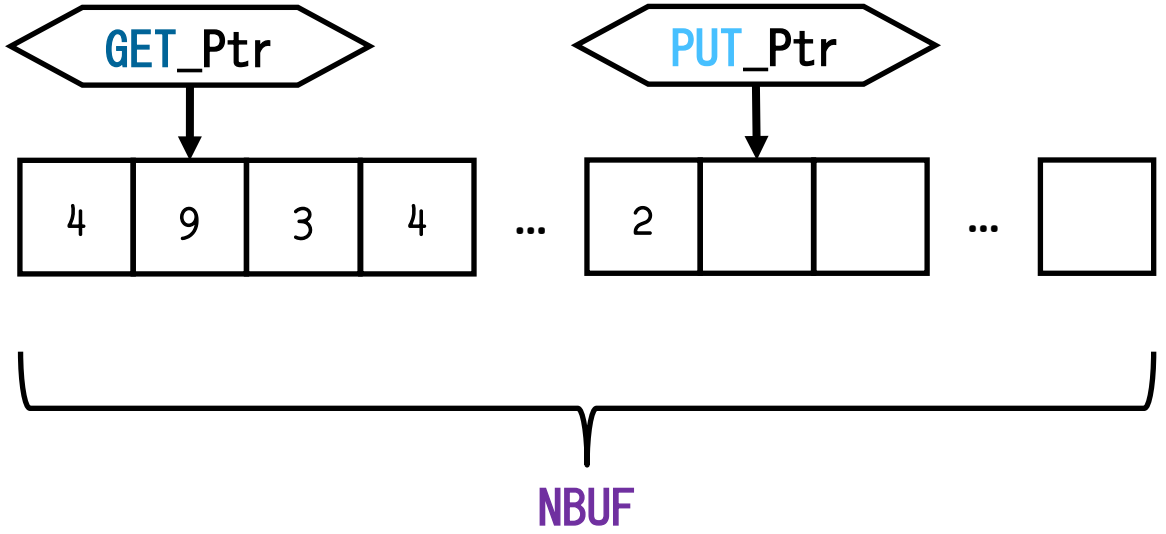


3

[1]R. H. Campbell, T. J. Miller. 1978. *A path Pascal Language.*
[2]O. Spinczyk, A. Gal, W. Schröder-Preikschat. 2002. *AspectC++: an aspect-oriented extension to C++.*
[3]Adopted from Aspektorientierte Programmierung – Wikipedia

Step 1: A toy example from Path Pascal

- **First C++-prototype based on *Path Pascal*[1]**
  1. Implementation of Path Pascal in C++ based on templates
  2. Separation of Concerns:
     Utilization of Aspect-oriented Programming (AOP) using AspectC++[2] annotations
  3. Adoption of the bounded buffer Path Pascal example[1]:
     `Par< NBUF, Seq< Mutex< PUT >, Mutex< GET > > >`



[1]R. H. Campbell, T. J. Miller. 1978. *A path Pascal Language.*
[2]O. Spinczyk, A. Gal, W. Schröder-Preikschat. 2002. *AspectC++: an aspect-oriented extension to C++.*
[3]Adopted from Aspektorientierte Programmierung – Wikipedia

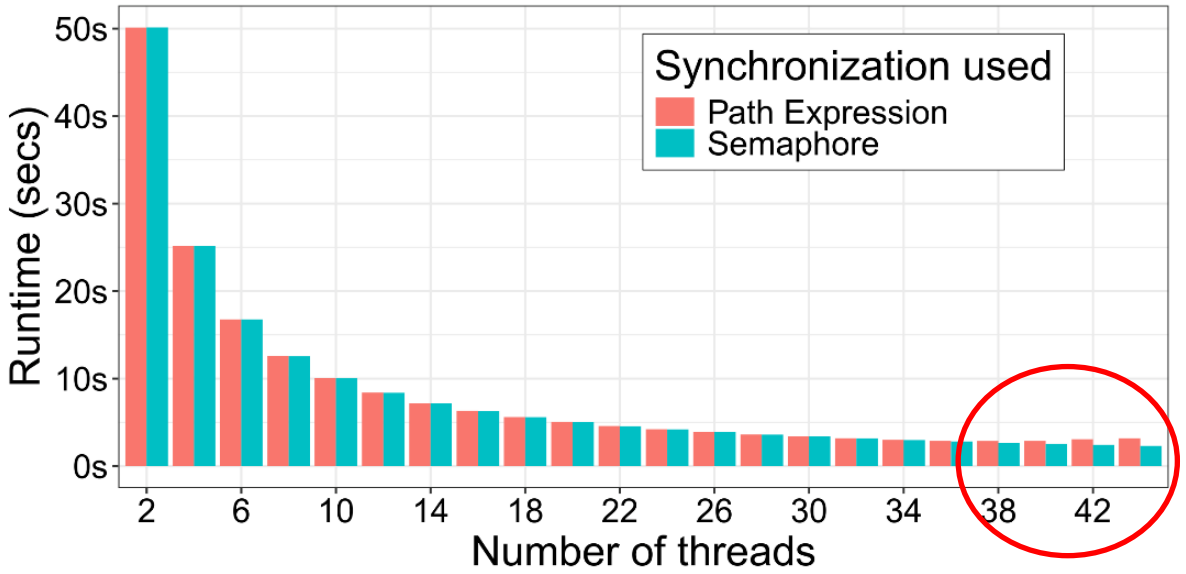
Step 1: A toy example from Path Pascal

- **First C++-prototype based on *Path Pascal*[1]**
  1. Implementation of Path Pascal in C++ based on templates
  2. Separation of Concerns:
     Utilization of Aspect-oriented Programming (AOP) using AspectC++[2] annotations
  3. Adoption of the bounded buffer Path Pascal example[1]:
     `Par< NBUF, Seq< Mutex< PUT >, Mutex< GET > > >`
  4. Performance benchmarks:
     - Comparison: with semaphore-based synchronization
     - Constant workload; increasing number of threads

→ Promissing performance results:

\+ Performance almost identical

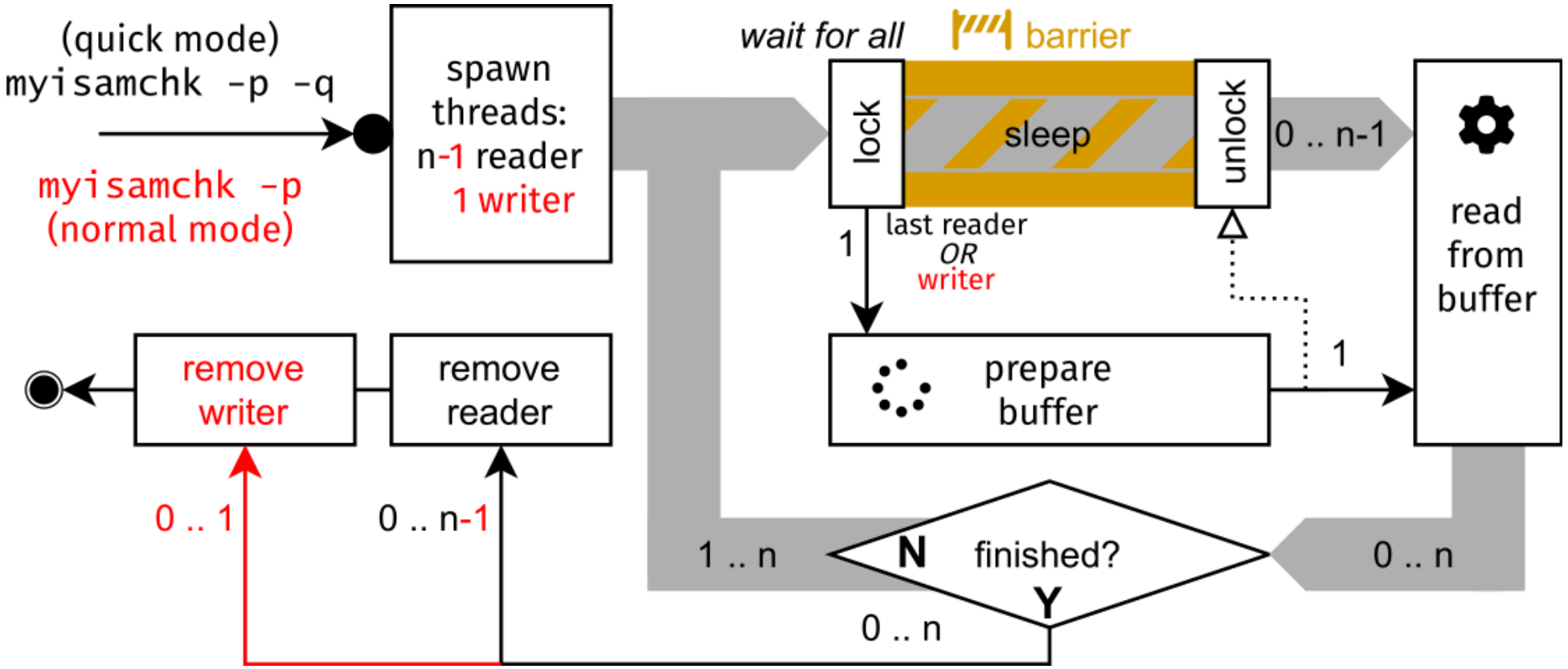
\- Bottleneck for high thread counts



[1]R. H. Campbell, T. J. Miller. 1978. *A path Pascal Language.*
[2]O. Spinczyk, A. Gal, W. Schröder-Preikschat. 2002. *AspectC++: an aspect-oriented extension to C++.*
[3]Adopted from Aspektorientierte Programmierung – Wikipedia

Step 2: A reality check with myisamchk (MySQL Server)

- **Practical example: myisamchk**[1]
  used for checking and repairing MyISAM tables



[1]https://github.com/mysql/mysql-server/

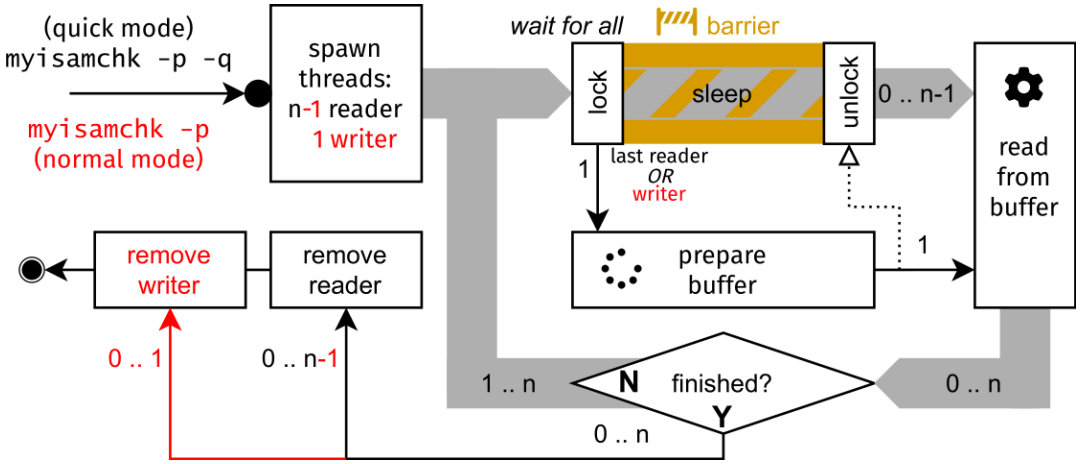
Step 2: A reality check with myisamchk (MySQL Server)

- **Practical example: myisamchk[1]**
  used for checking and repairing MyISAM tables
- **No. of MyISAM indices = No. of threads**
  known only at runtime

→ Second C++-prototype:

- PE language analogous to first prototype
- Implementation allows dynamic PE creation a runtime

- **Performance benchmarks:**
  - Integration of our synchronization via AspectC++
  - Comparison: **original** vs. **our** synchronization



→ Mixed performance results:

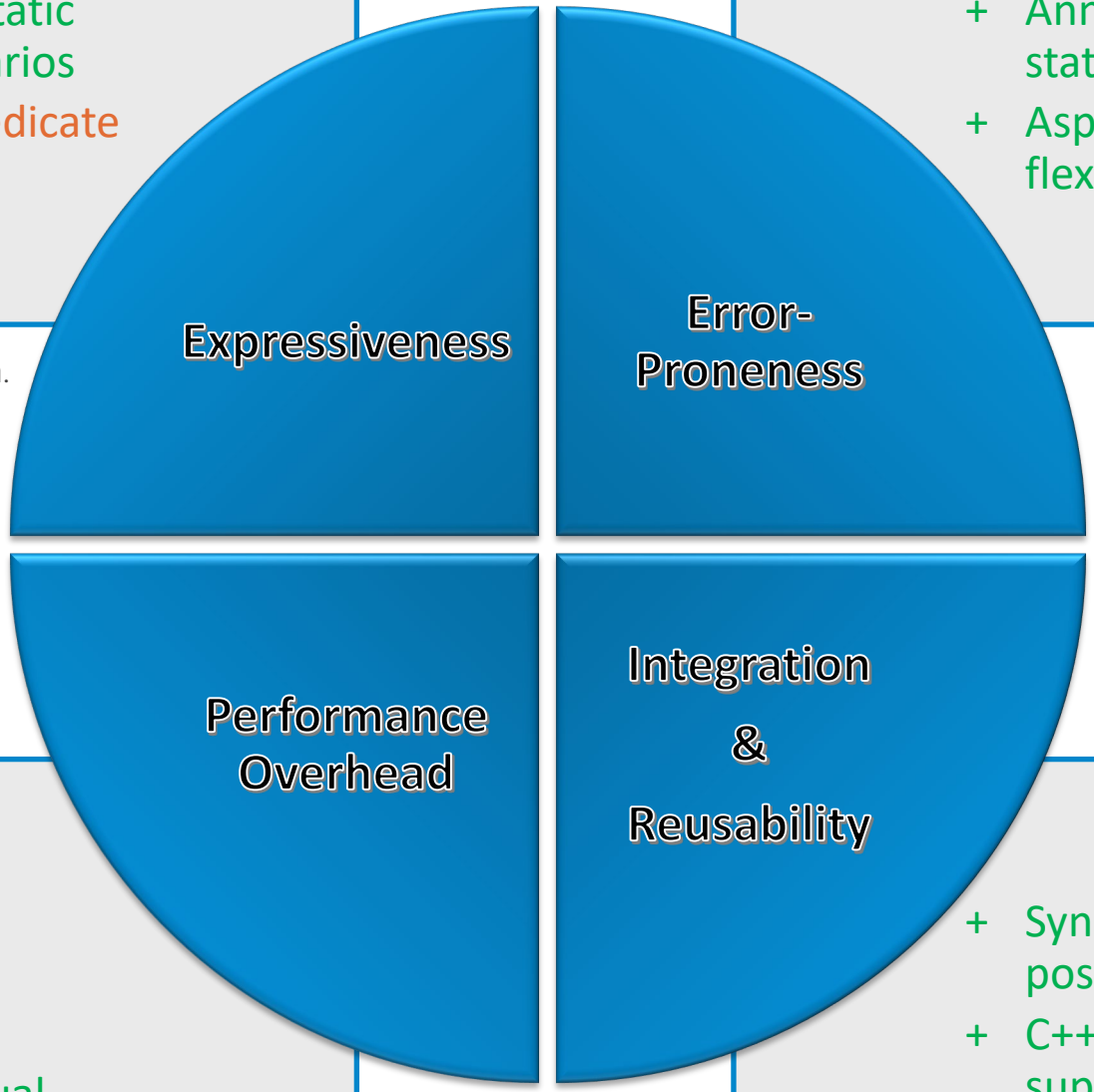
+ *Normal mode*: performance almost identical; bottleneck only for high thread counts (analogous to step 1)
- *Quick mode*: performs significantly worse (even than *normal mode*); runtime factor between 1.24 and 2.36

[1]https://github.com/mysql/mysql-server/

Step 1 + Step 2 → Results

+ Traditional PEs suitable for static but limited in dynamic scenarios
- Language extensions like Predicate PEs[1] are adviced

+ Annotations are convenient and state of the art
+ AspectC++ annotations are very flexible

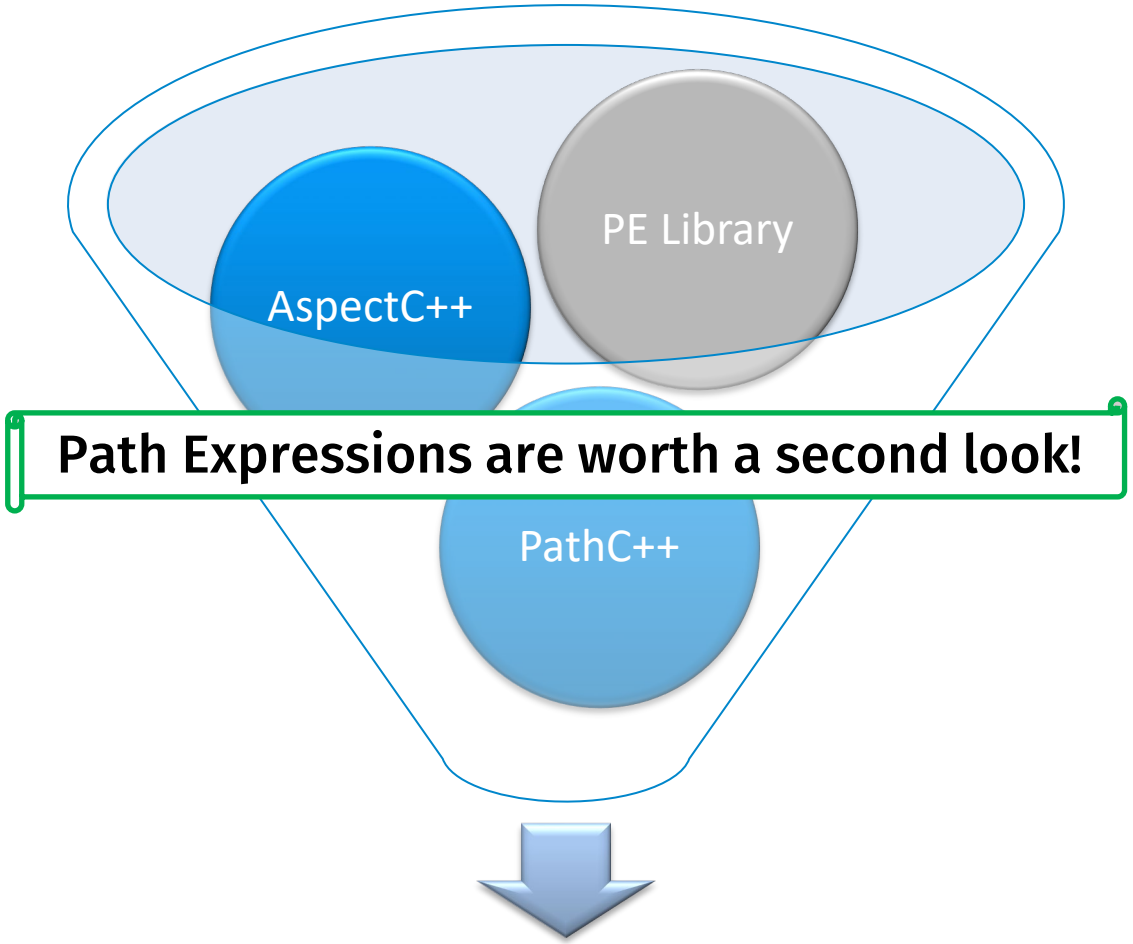


- Implementation & scenario dependend
- Unintuitive/Hard to predict
+ BUT: can be as good as manual

+ Synchronization pattern library possible
+ C++'s object-oriented features support this naturally

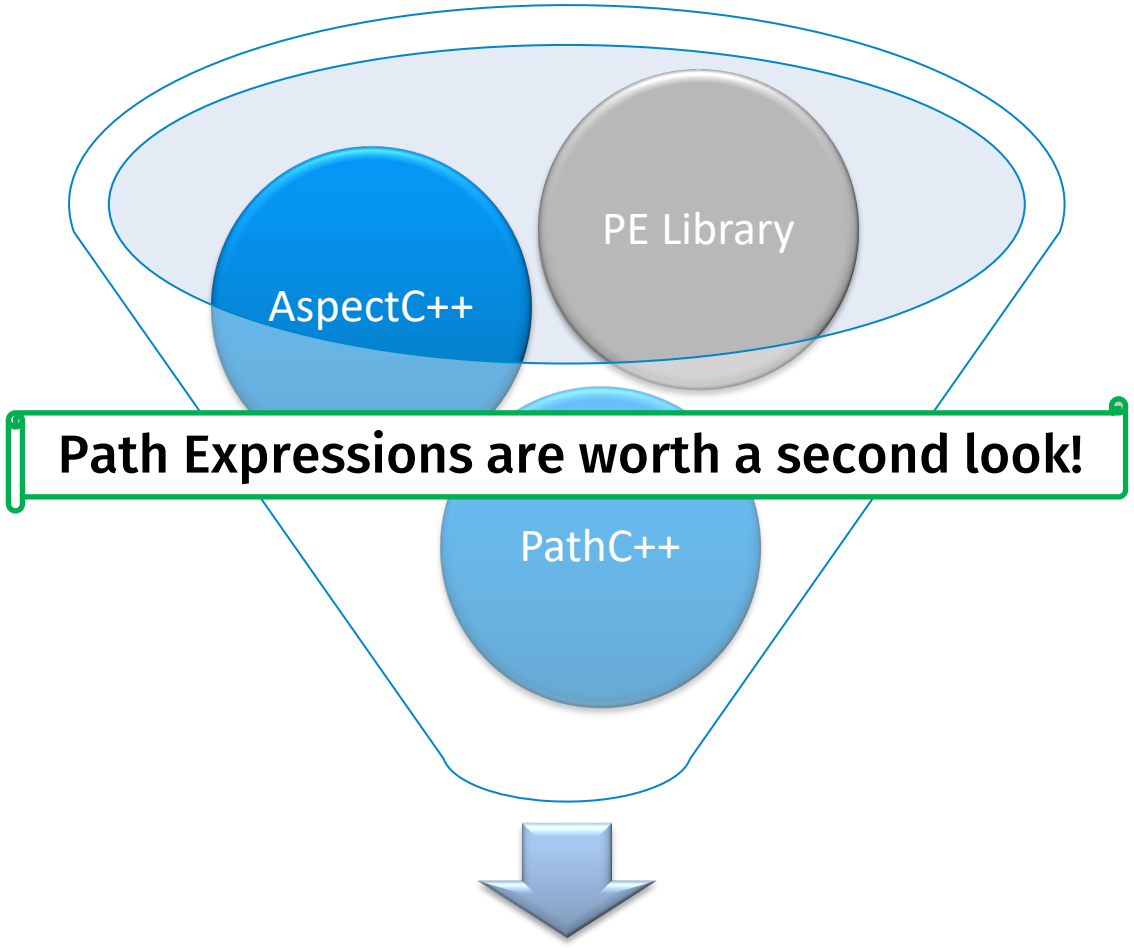
[1]S. Andler. 1979. Predicate path expression.

Conclusion



The Better Way to Synchronize?

Conclusion



The Better Way to Synchronize?

Try it yourself!

- PathC++: https://ess.cs.uos.de/git/software/path-cplusplus

- Please tell us about your experience and share complex synchronization patterns with us.
- We plan to work on:
  - Formal PE-language analysis
  - Extending the PE-language features of PathC++
  - Performance improvements
  - …

Path Expressions Revisited – Towards Compiler-enforced Reusable Synchronization Patterns

PLOS 2025

13th October 2025

Thomas Alexander Hövelmann
(thomas-alexander.hoevelmann@tu-dortmund.de)

Olaf Spinczyk
(olaf@uos.de)

Alexander Krause
(alexander.krause@cs.tu-dortmund.de)

Horst Schirmeier
(horst.schirmeier@tu-dresden.de)

Peter Ulbrich
(peter.ulbrich@tu-dortmund.de)

Technische Universität Dresden

technische universität dortmund

UNIVERSITÄT OSNABRÜCK

Backup: myisamchk experiments