Lock-free Datenstrukturen

Eine Einführung in die lock-free Programmierung

Burak Ok

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU)

24. Januar 2017

Inhaltsverzeichnis

Inhaltsverzeichnis

Primitiven

- In Hardware und Software [Int][Arm][Cpp]

Primitiven

- In Hardware und Software [Int][Arm][Cpp]
- Memory barriers
  - Verhindert Umordnung von Lade- und Speicherbefehlen

Primitiven

- In Hardware und Software [Int][Arm][Cpp]
- Memory barriers
  - Verhindert Umordnung von Lade- und Speicherbefehlen
- test-and-set (TAS)
  - Setzt ein Bit auf 1, falls es 0 war

Primitiven

- In Hardware und Software [Int][Arm][Cpp]
- Memory barriers
  - Verhindert Umordnung von Lade- und Speicherbefehlen
- test-and-set (TAS)
  - Setzt ein Bit auf 1, falls es 0 war
- fetch-and-add (FAD)
  - Addiert eine Zahl
  - Gibt vorherige Zahl zurück

Primitiven

- In Hardware und Software [Int][Arm][Cpp]
- Memory barriers
  - Verhindert Umordnung von Lade- und Speicherbefehlen
- test-and-set (TAS)
  - Setzt ein Bit auf 1, falls es 0 war
- fetch-and-add (FAD)
  - Addiert eine Zahl
  - Gibt vorherige Zahl zurück
- linked-load und store-conditional (LL & SC)
  - LL lädt einen Wert
  - SC speichert einen Wert, falls seit LL nichts geändert wurde

Primitiven

- In Hardware und Software [Int][Arm][Cpp]
- Memory barriers
  - Verhindert Umordnung von Lade- und Speicherbefehlen
- test-and-set (TAS)
  - Setzt ein Bit auf 1, falls es 0 war
- fetch-and-add (FAD)
  - Addiert eine Zahl
  - Gibt vorherige Zahl zurück
- linked-load und store-conditional (LL & SC)
  - LL lädt einen Wert
  - SC speichert einen Wert, falls seit LL nichts geändert wurde
- compare-and-swap (CAS)
  - Speichert einen Wert, falls Adresse einen bestimmten Wert enthält

Vorteile

Vorteile

- Höherer Durchsatz
- Keine Deadlocks

Nachteile

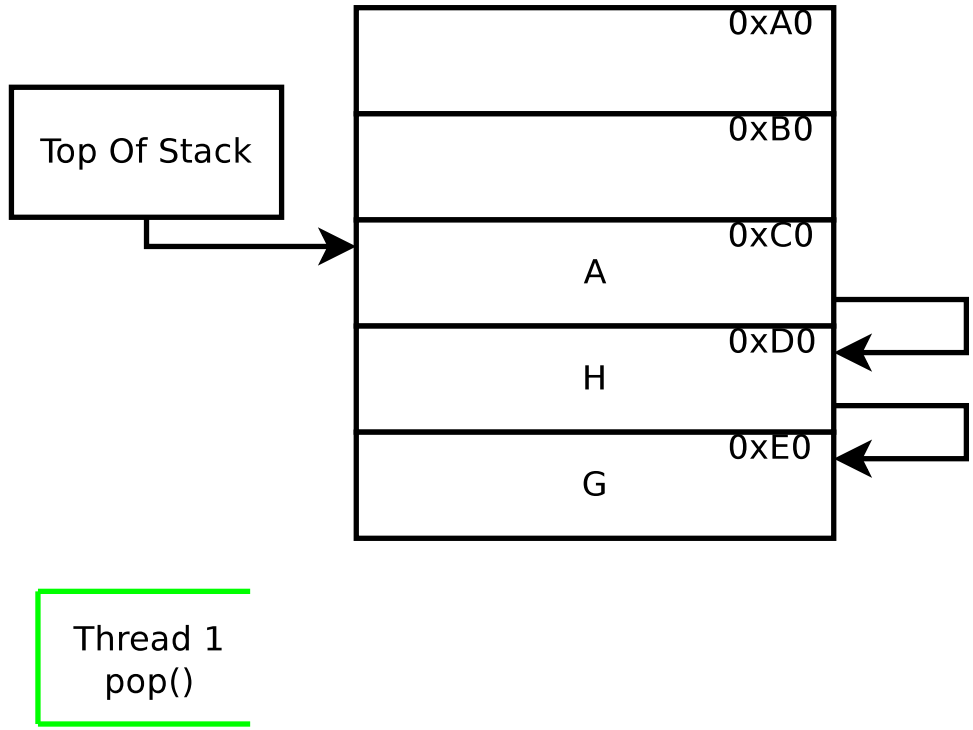
- Nicht wait-free

Nachteile

- Nicht wait-free
- ABA Problem bei CAS

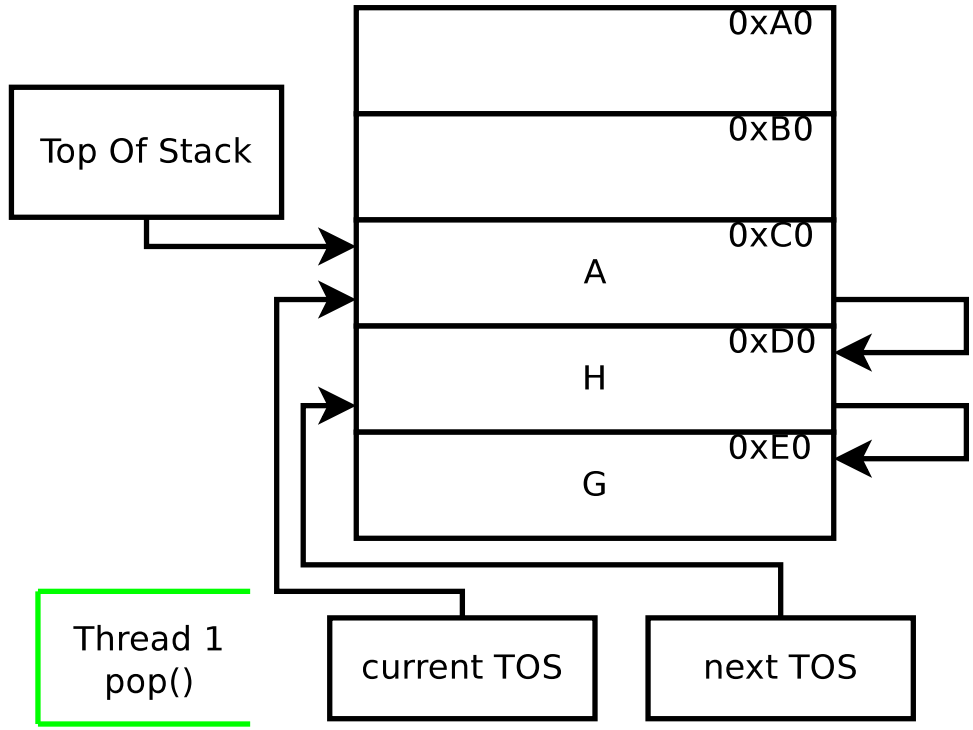
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



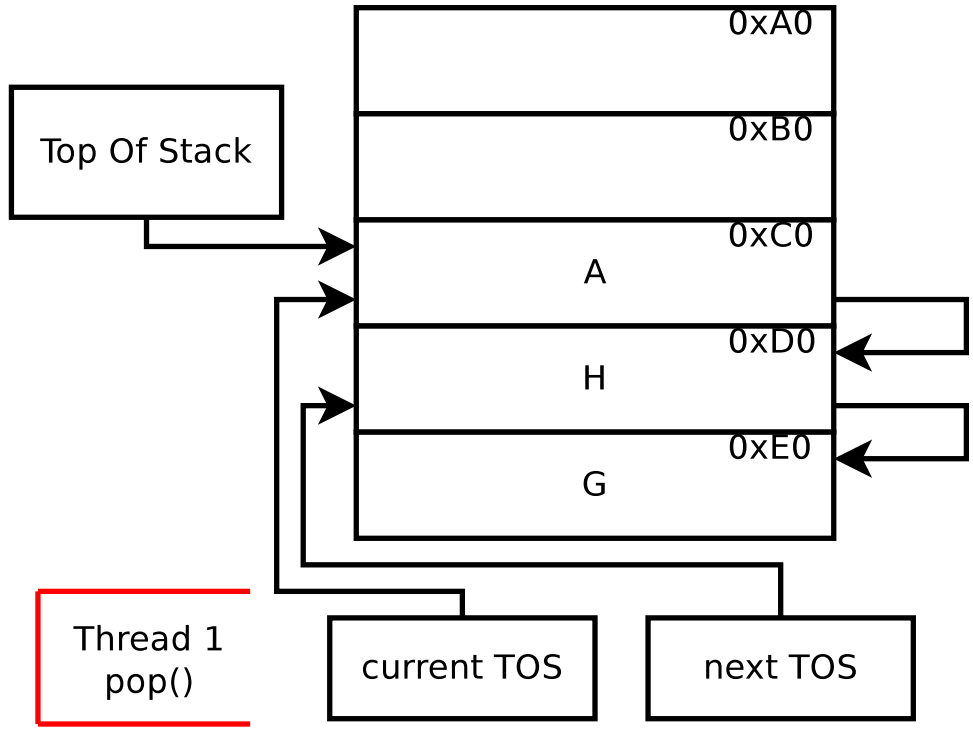
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



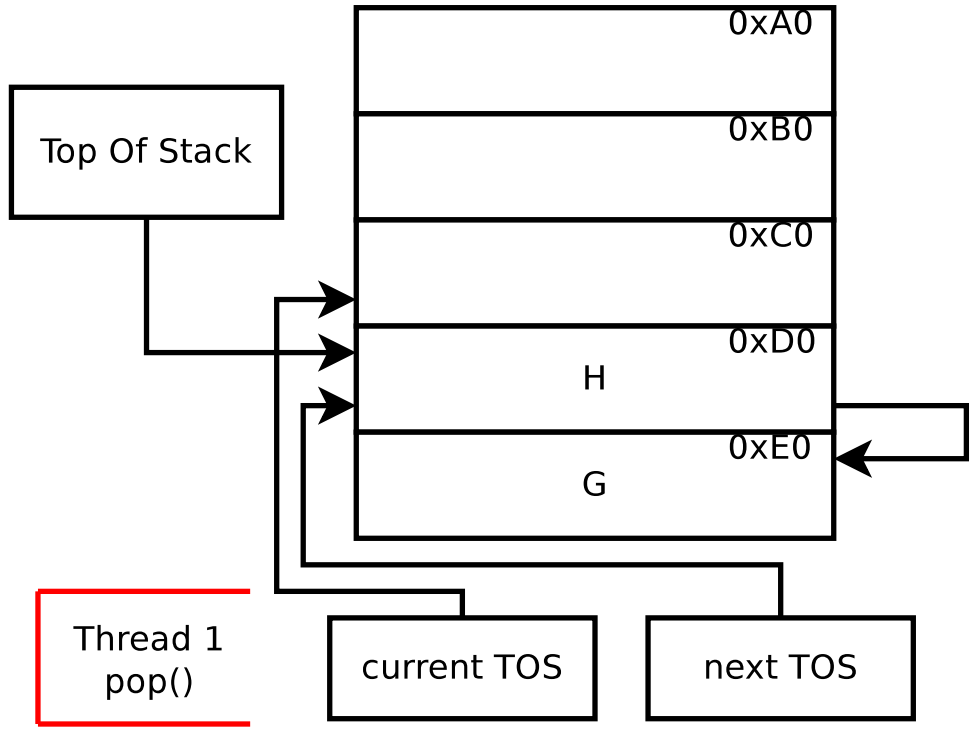
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



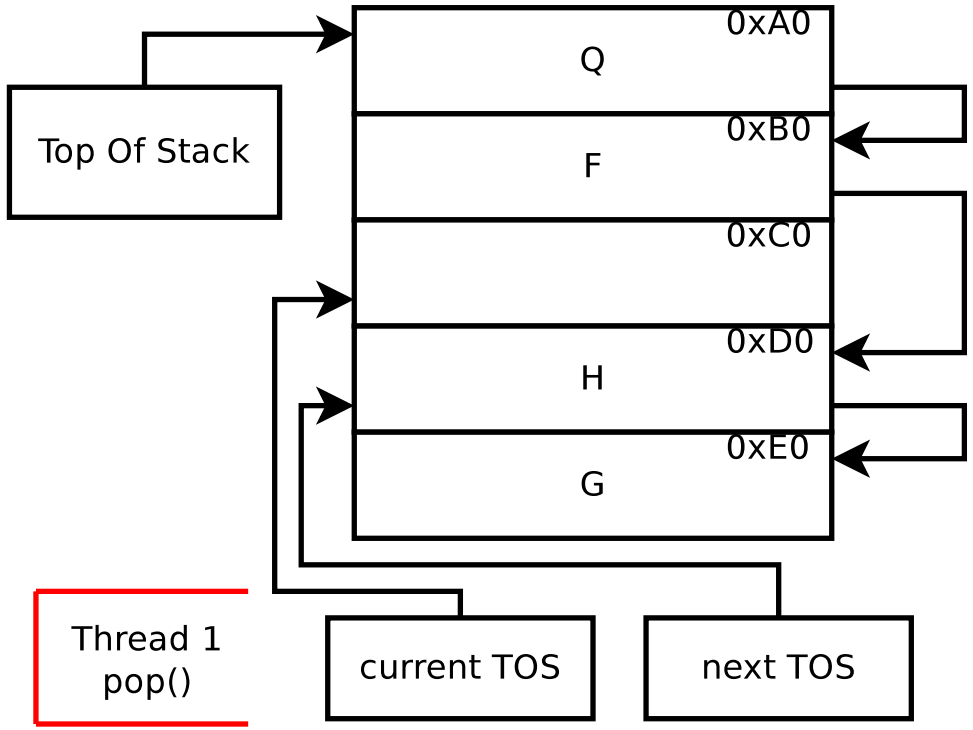
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



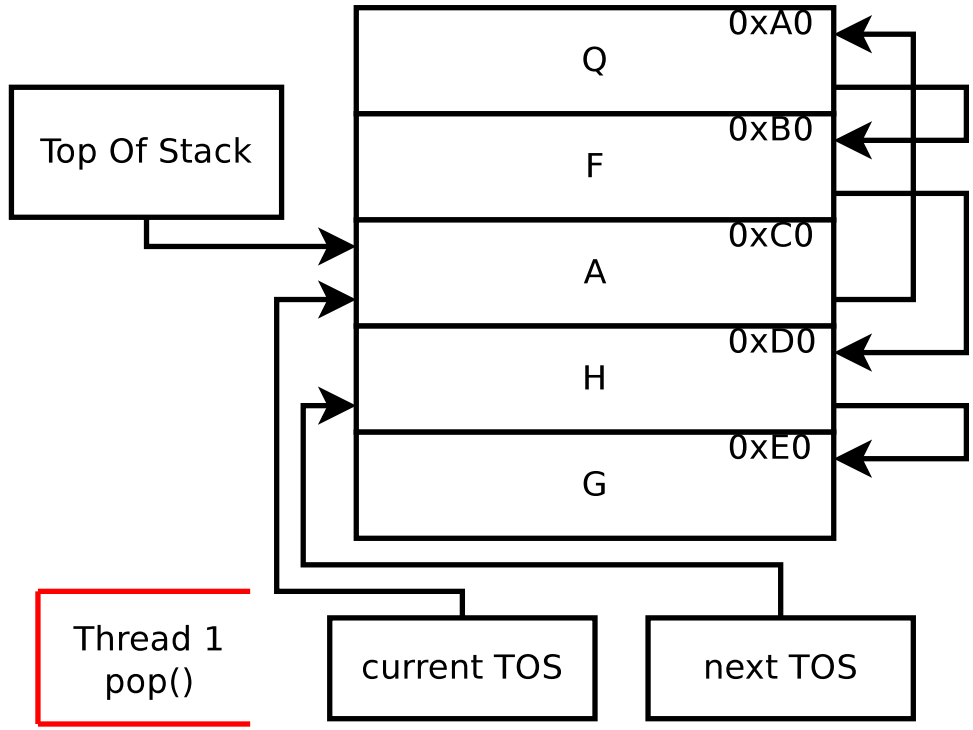
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



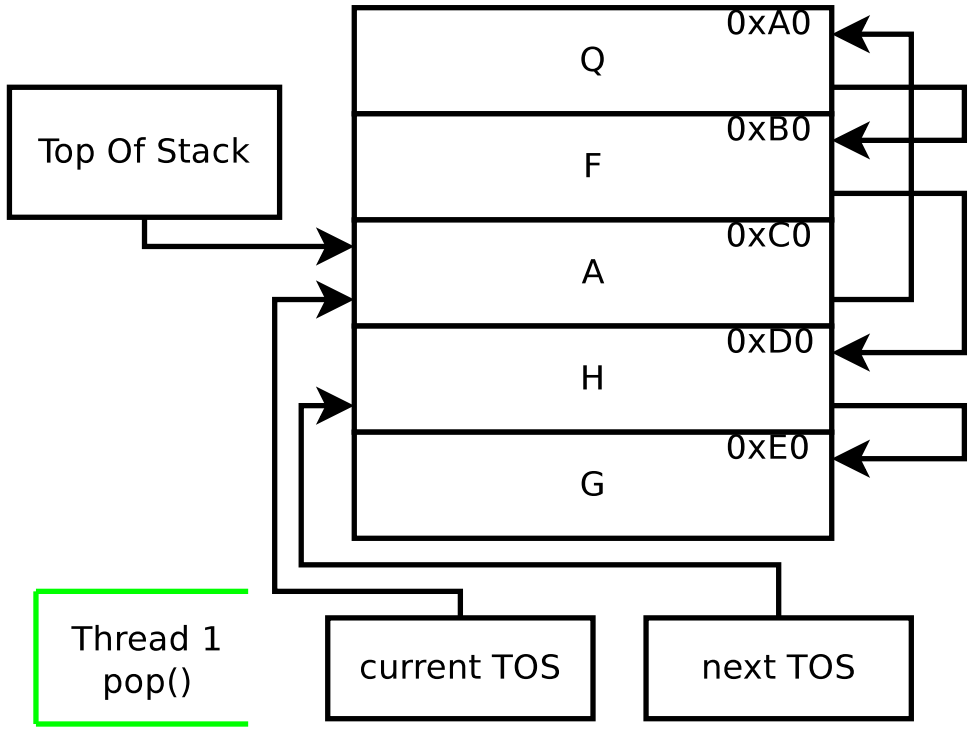
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



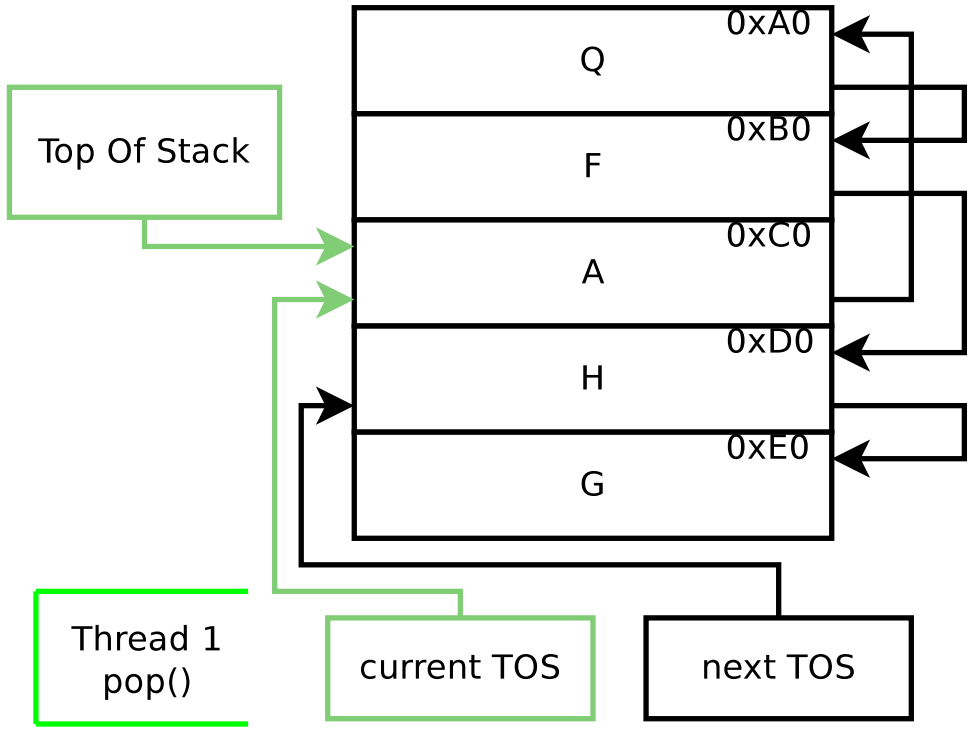
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



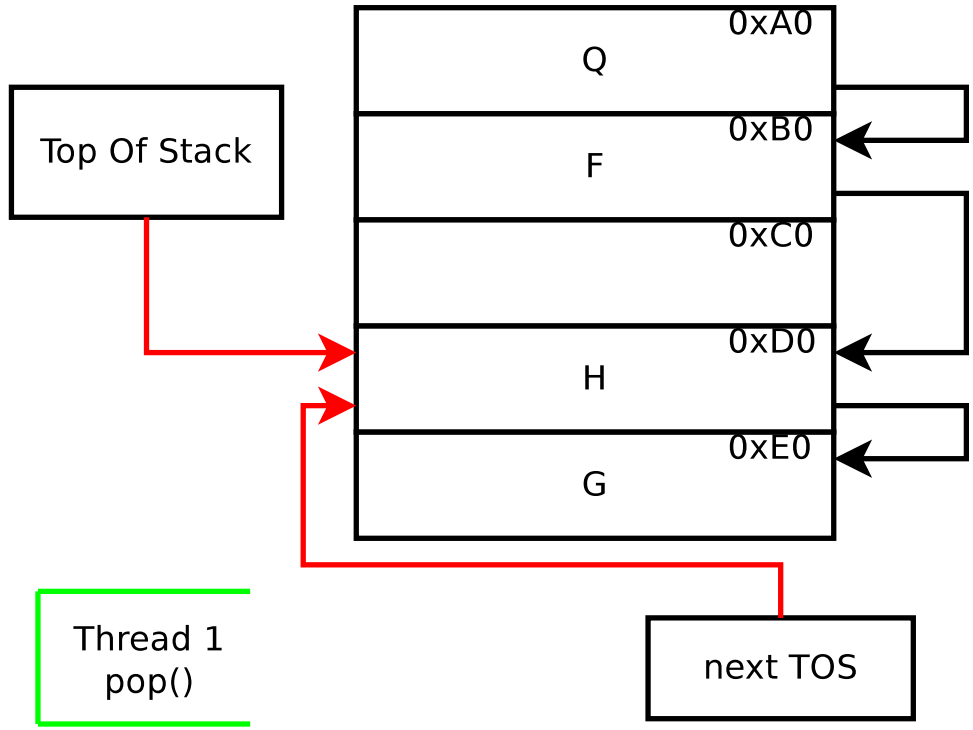
Nachteile

- Nicht wait-free
- ABA Problem bei CAS



Nachteile

- Nicht wait-free
- ABA Problem bei CAS



Nachteile

- Nicht wait-free
- ABA Problem bei CAS
- Schwer richtig zu implementieren

Nachteile

- Nicht wait-free
- ABA Problem bei CAS
- Schwer richtig zu implementieren

  It's easy to write lock-free code that appears to work, but it's very difficult to write lock-free code that is correct and performs well.

  [Her]

Inhaltsverzeichnis

Korrektheitsbeweis

1. Linearisierbarkeit
   - Änderungen scheinen nach “aussen” atomar

Korrektheitsbeweis

1. Linearisierbarkeit
   - Änderungen scheinen nach “aussen” atomar
2. Sicherheit
   - Queue bleibt immer eine Queue

Korrektheitsbeweis

1. Linearisierbarkeit
   - Änderungen scheinen nach “aussen” atomar
2. Sicherheit
   - Queue bleibt immer eine Queue
3. Lebendigkeit
   - Operationen erfolgen irgendwann

Korrektheitsbeweis

1. Linearisierbarkeit
   - Änderungen scheinen nach “aussen” atomar
2. Sicherheit
   - Queue bleibt immer eine Queue
3. Lebendigkeit
   - Operationen erfolgen irgendwann
     - Lock-free: In fester Zeit, falls kein Wettstreit

Definitionen und Initialisierung

```
structure pointer_t
   {ptr: pointer to node_t, count: uint}
structure node_t
   {value: data_type, next: pointer_t}
structure queue_t
   {Head: pointer_t, Tail: pointer_t}

initialize(Q: pointer to queue_t)
   node = new_node()
   node->next.ptr = NULL
   Q->Head = Q->Tail = <node, 0>
```

Definitionen und Initialisierung

- Verkette Liste

```
structure pointer_t
   {ptr: pointer to node_t, count: uint}
structure node_t
   {value: data_type, next: pointer_t}
structure queue_t
   {Head: pointer_t, Tail: pointer_t}

initialize(Q: pointer to queue_t)
   node = new_node()
   node->next.ptr = NULL
   Q->Head = Q->Tail = <node, 0>
```

Definitionen und Initialisierung

- Verkette Liste
- Mindestens einen Knoten

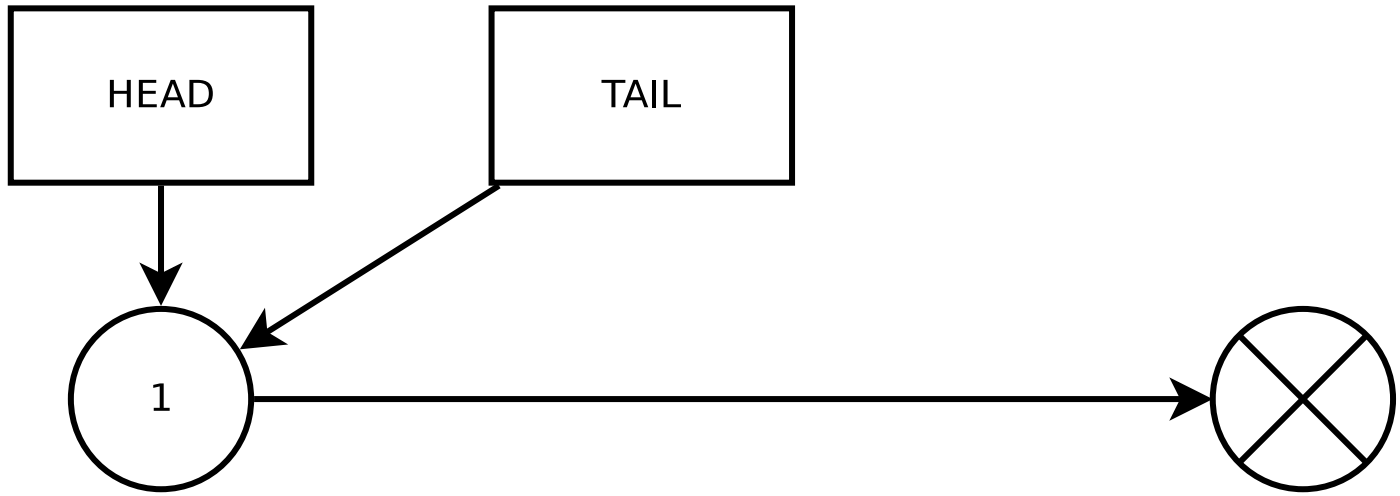
```
structure pointer_t
   {ptr: pointer to node_t, count: uint}
structure node_t
   {value: data_type, next: pointer_t}
structure queue_t
   {Head: pointer_t, Tail: pointer_t}

initialize(Q: pointer to queue_t)
   node = new_node()
   node->next.ptr = NULL
   Q->Head = Q->Tail = <node, 0>
```

Definitionen und Initialisierung

- Verkette Liste
- Mindestens einen Knoten
- Modifikationszähler im *pointer_t*

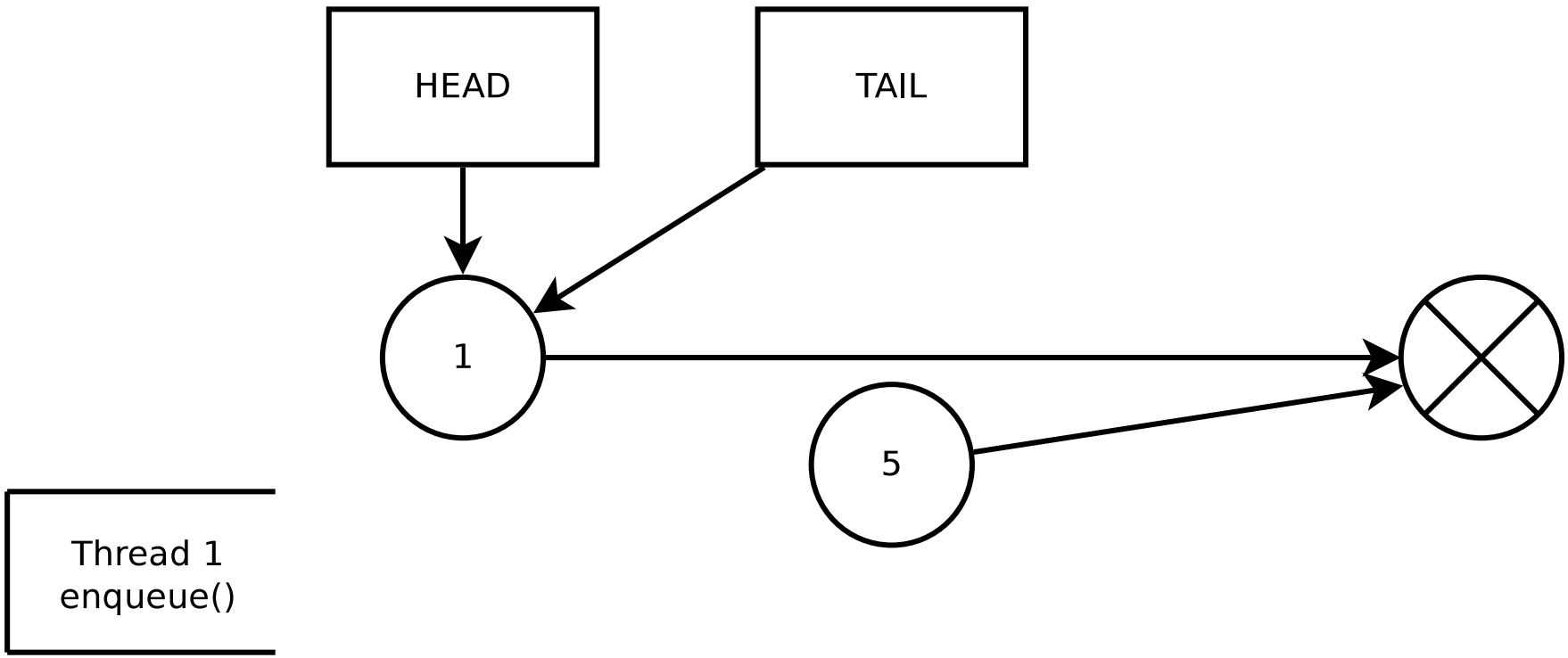
```
structure pointer_t
   {ptr: pointer to node_t, count: uint}
structure node_t
   {value: data_type, next: pointer_t}
structure queue_t
   {Head: pointer_t, Tail: pointer_t}

initialize(Q: pointer to queue_t)
   node = new_node()
   node->next.ptr = NULL
   Q->Head = Q->Tail = <node, 0>
```

Beispiel eines Dequeues

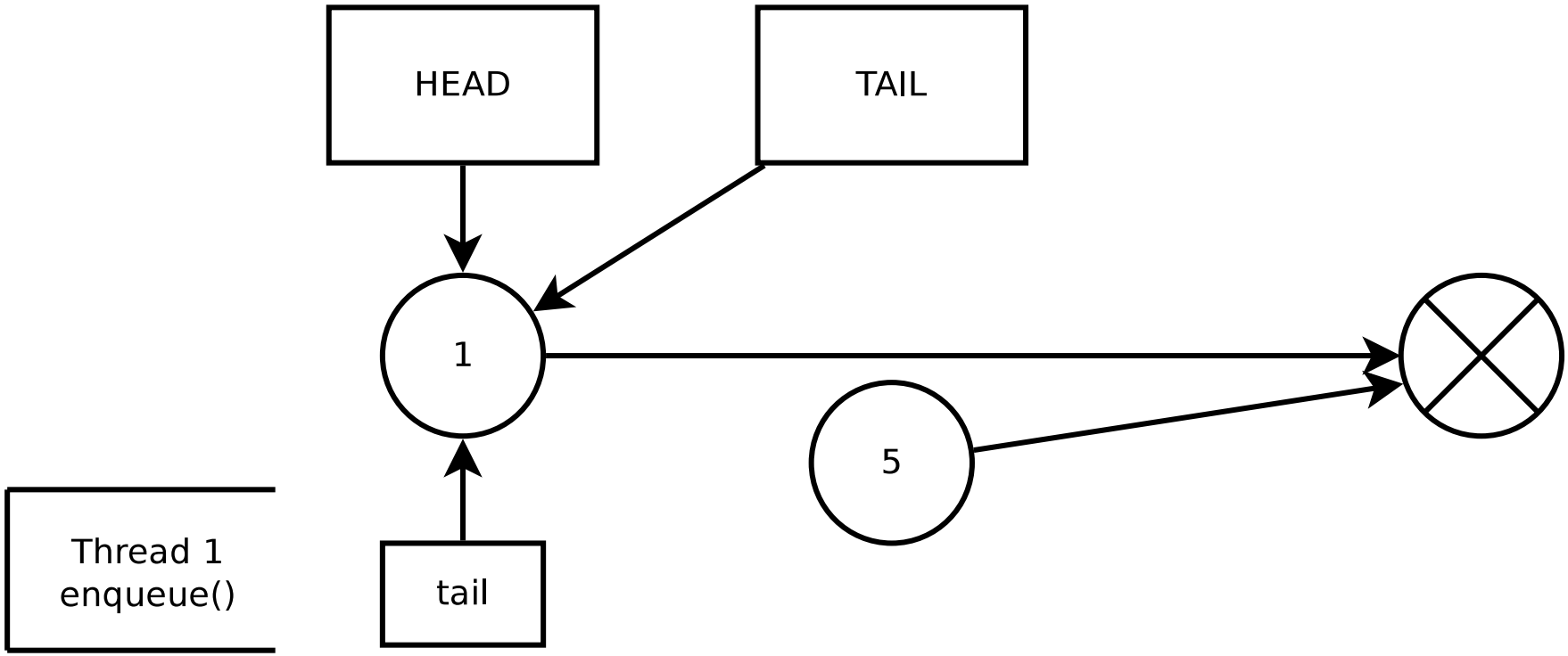


Thread 1
enqueue()

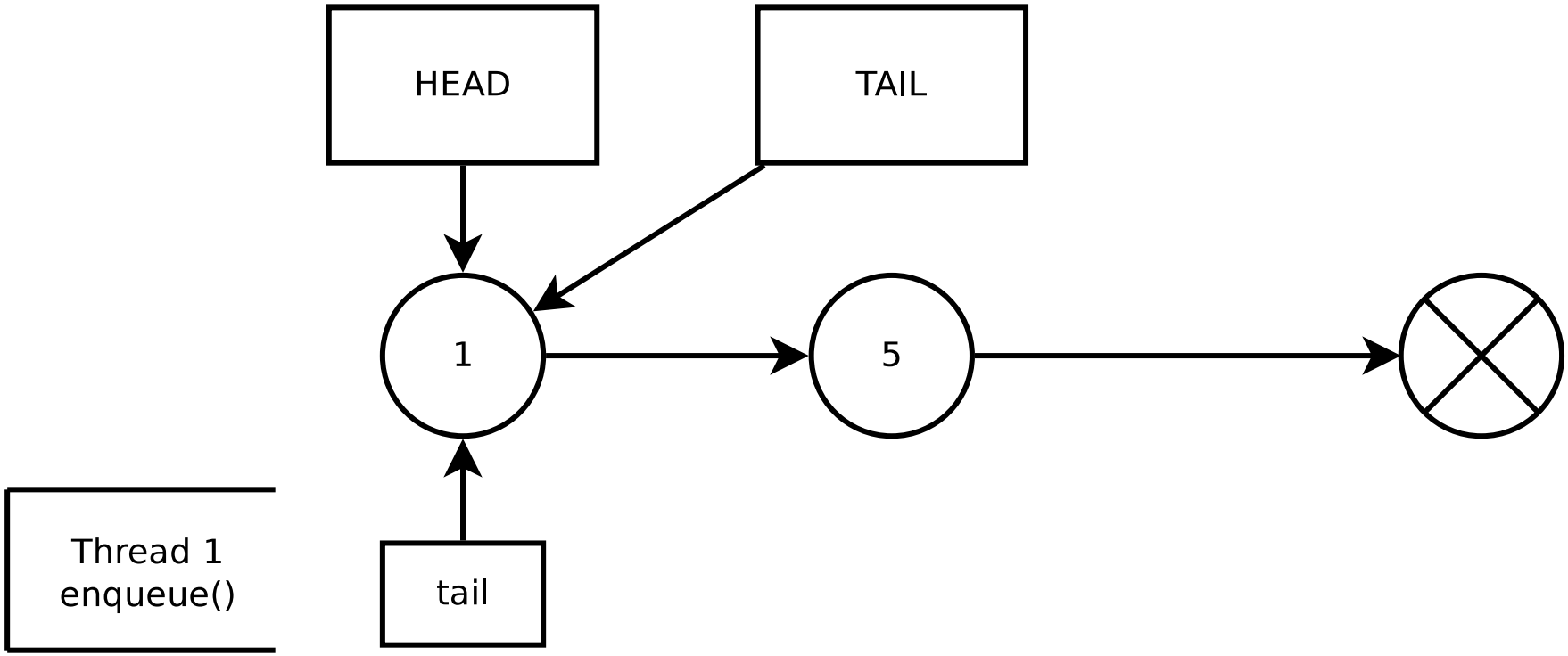
Beispiel eines Dequeues



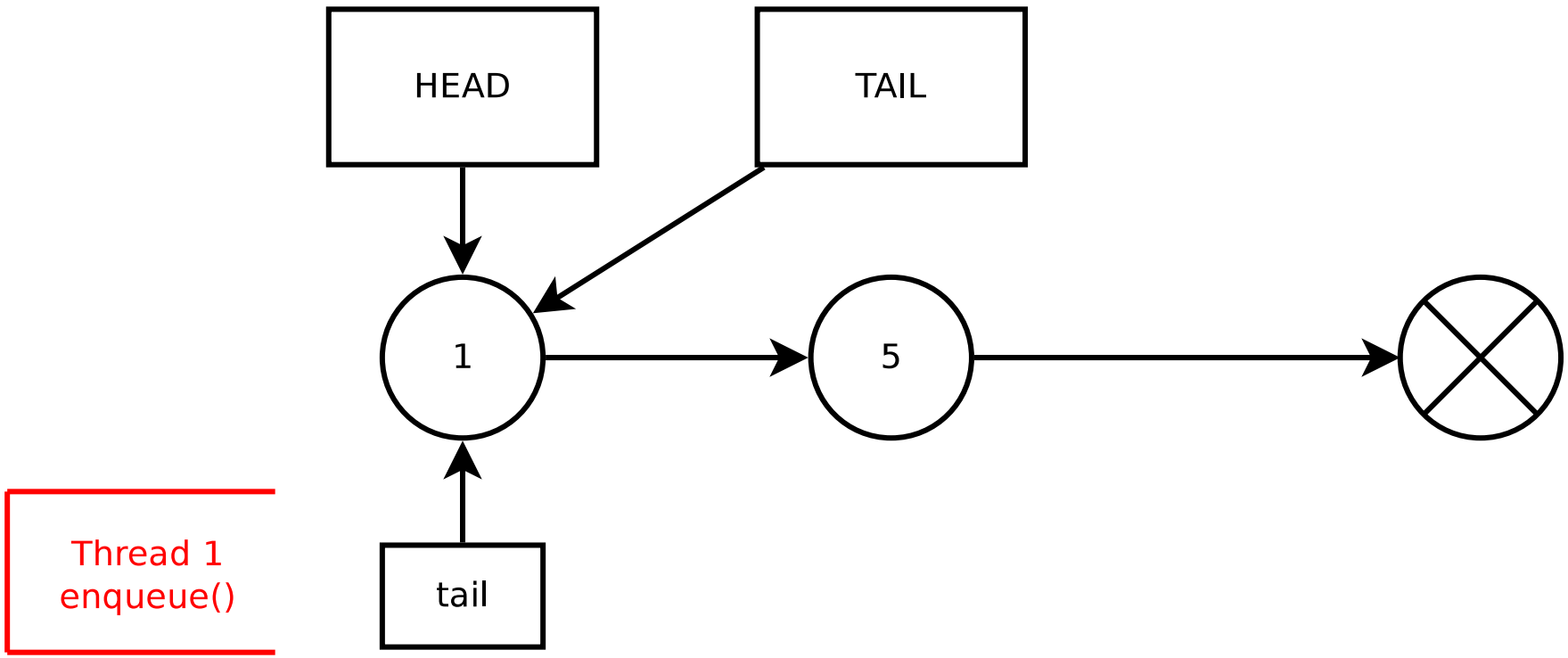
Beispiel eines Dequeues



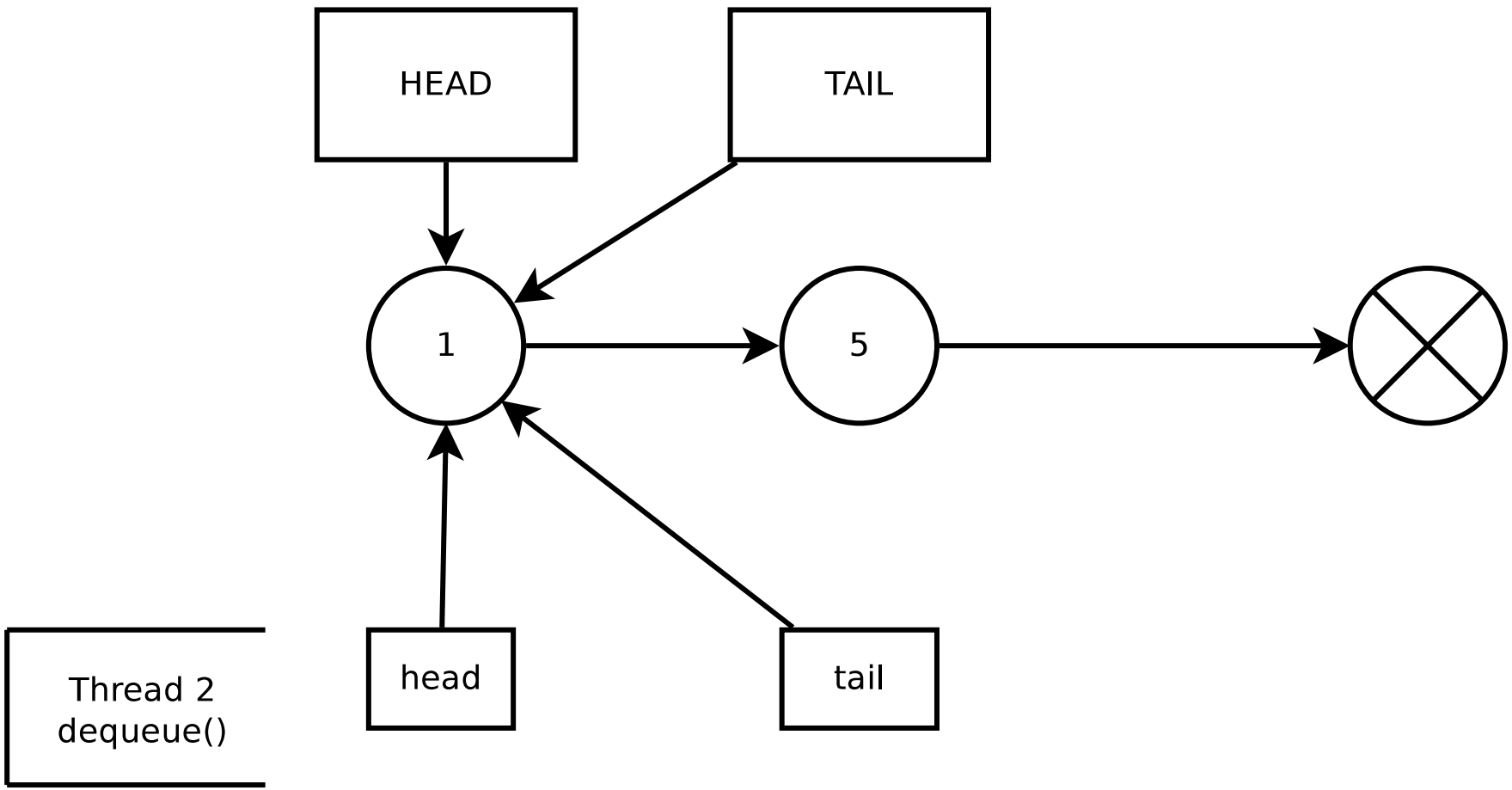
Beispiel eines Dequeues



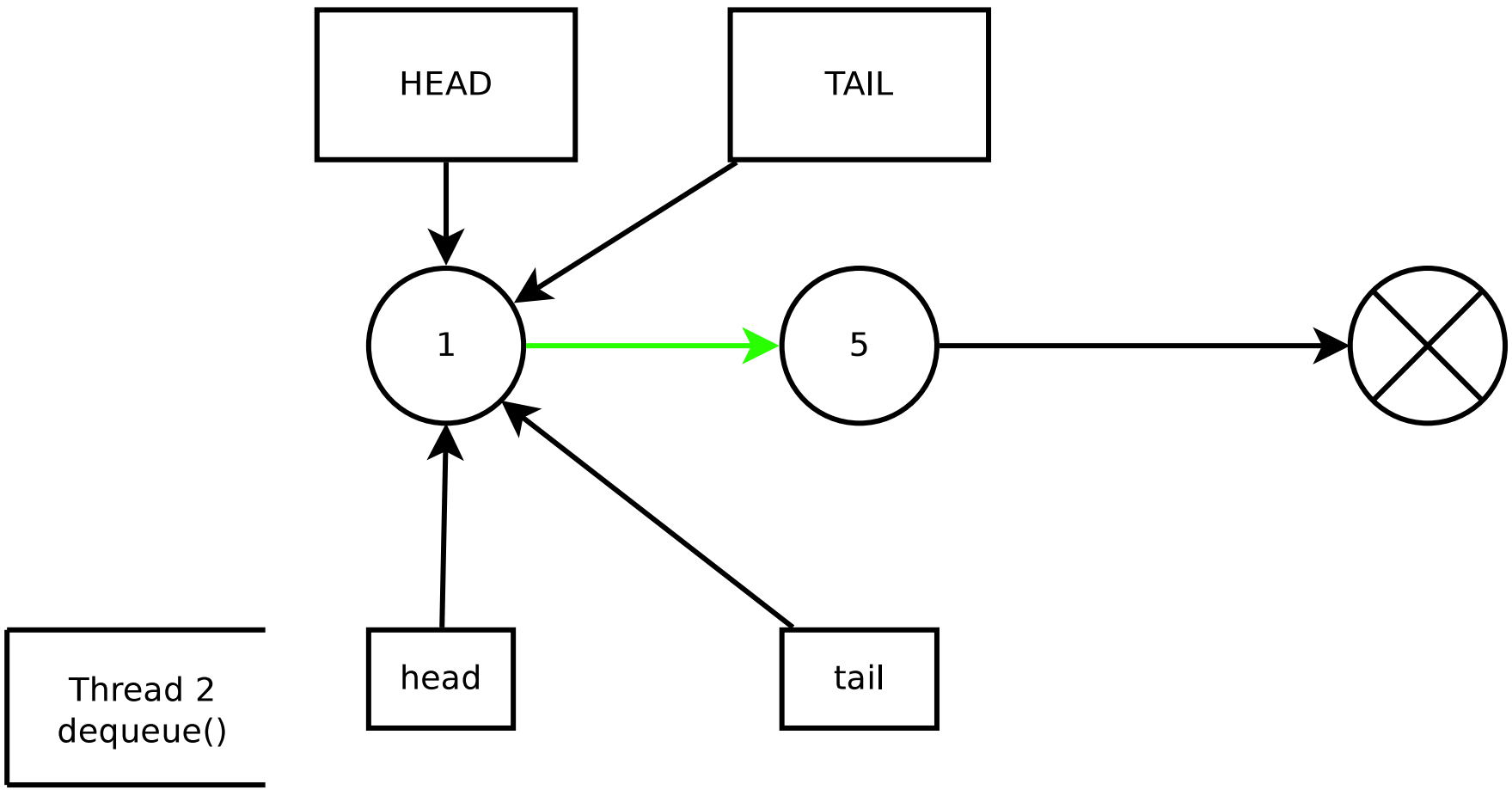
Beispiel eines Dequeues



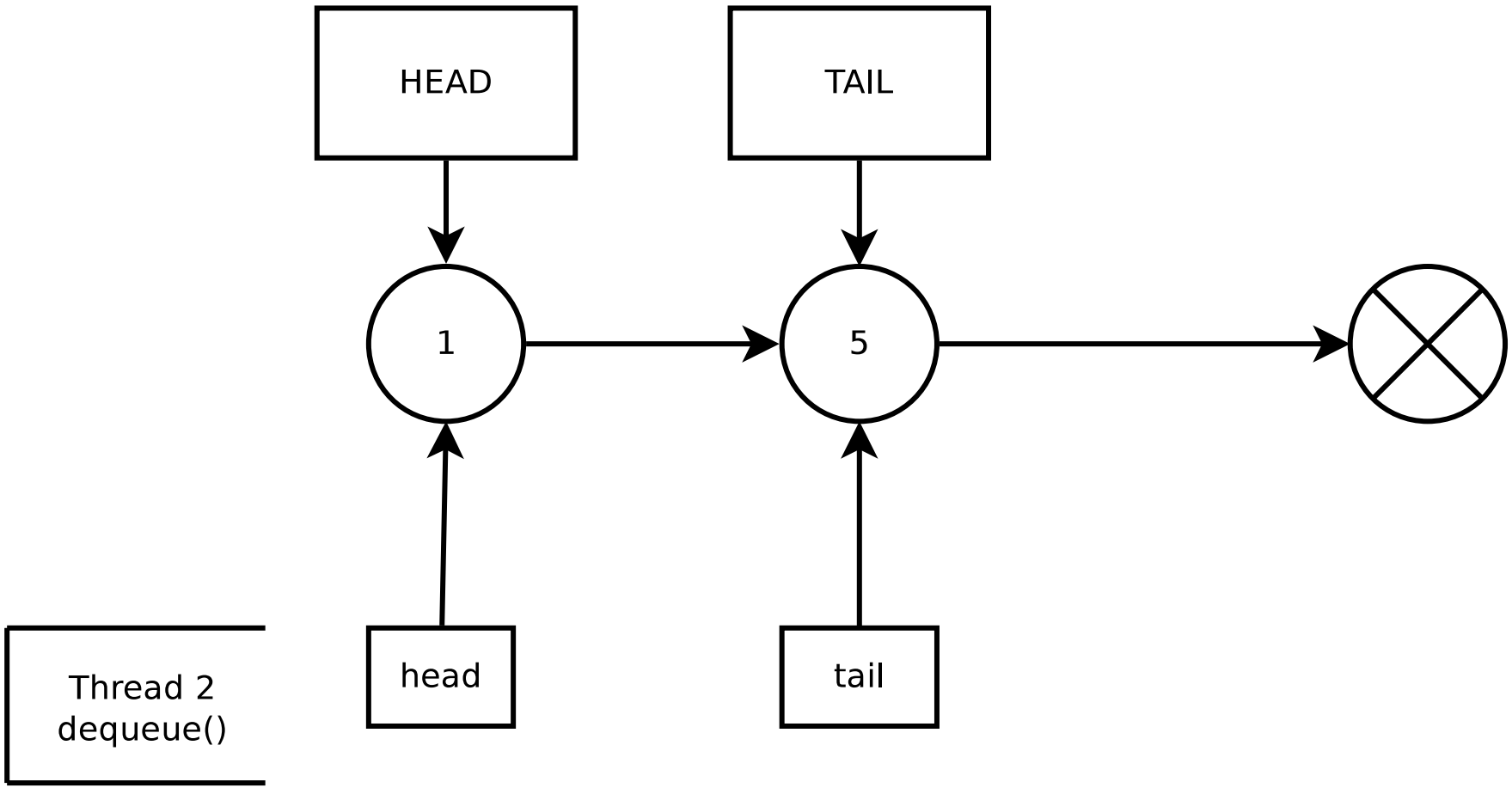
Beispiel eines Dequeues



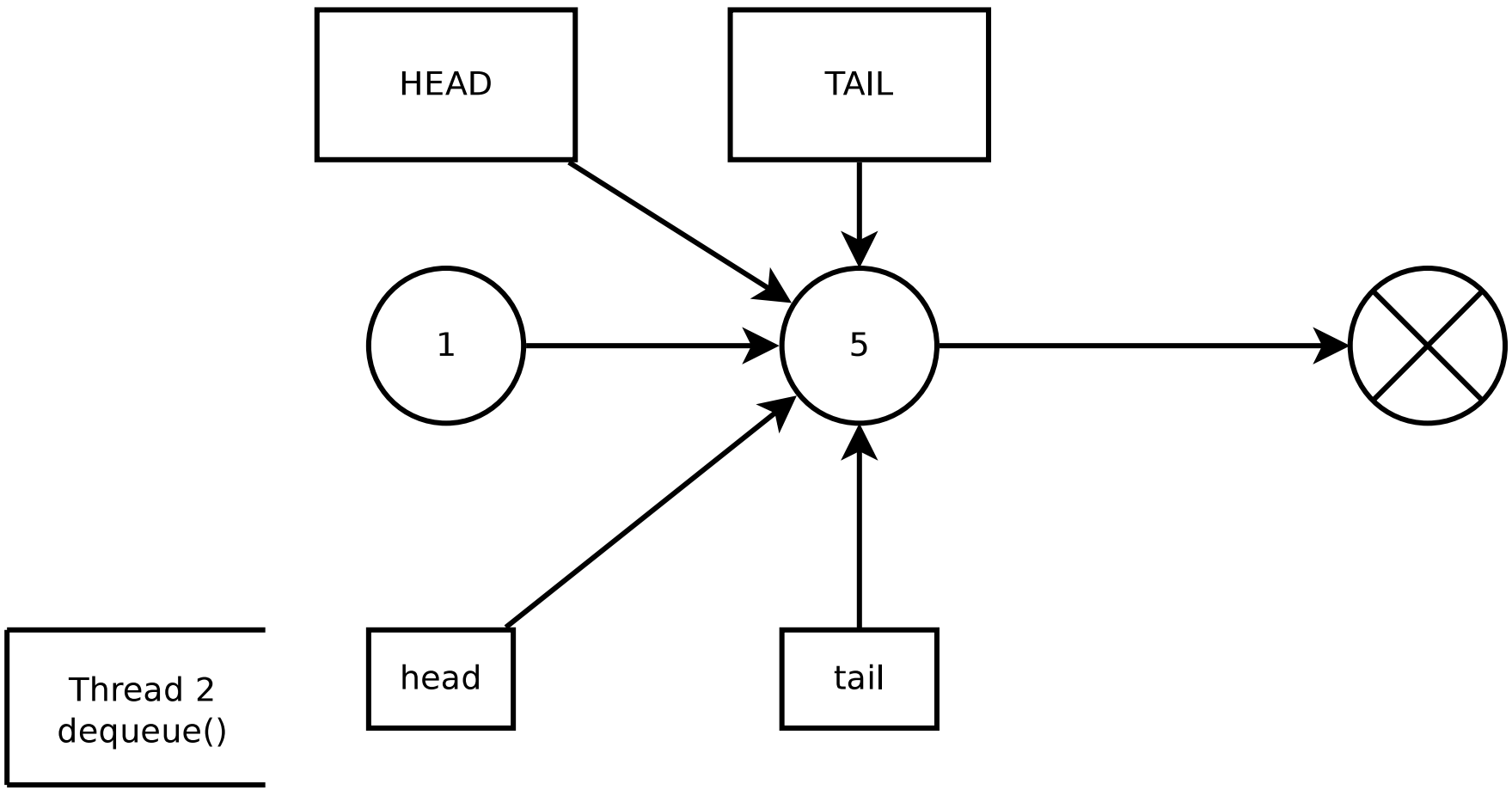
Beispiel eines Dequeues



Beispiel eines Dequeues



Beispiel eines Dequeues



Inhaltsverzeichnis

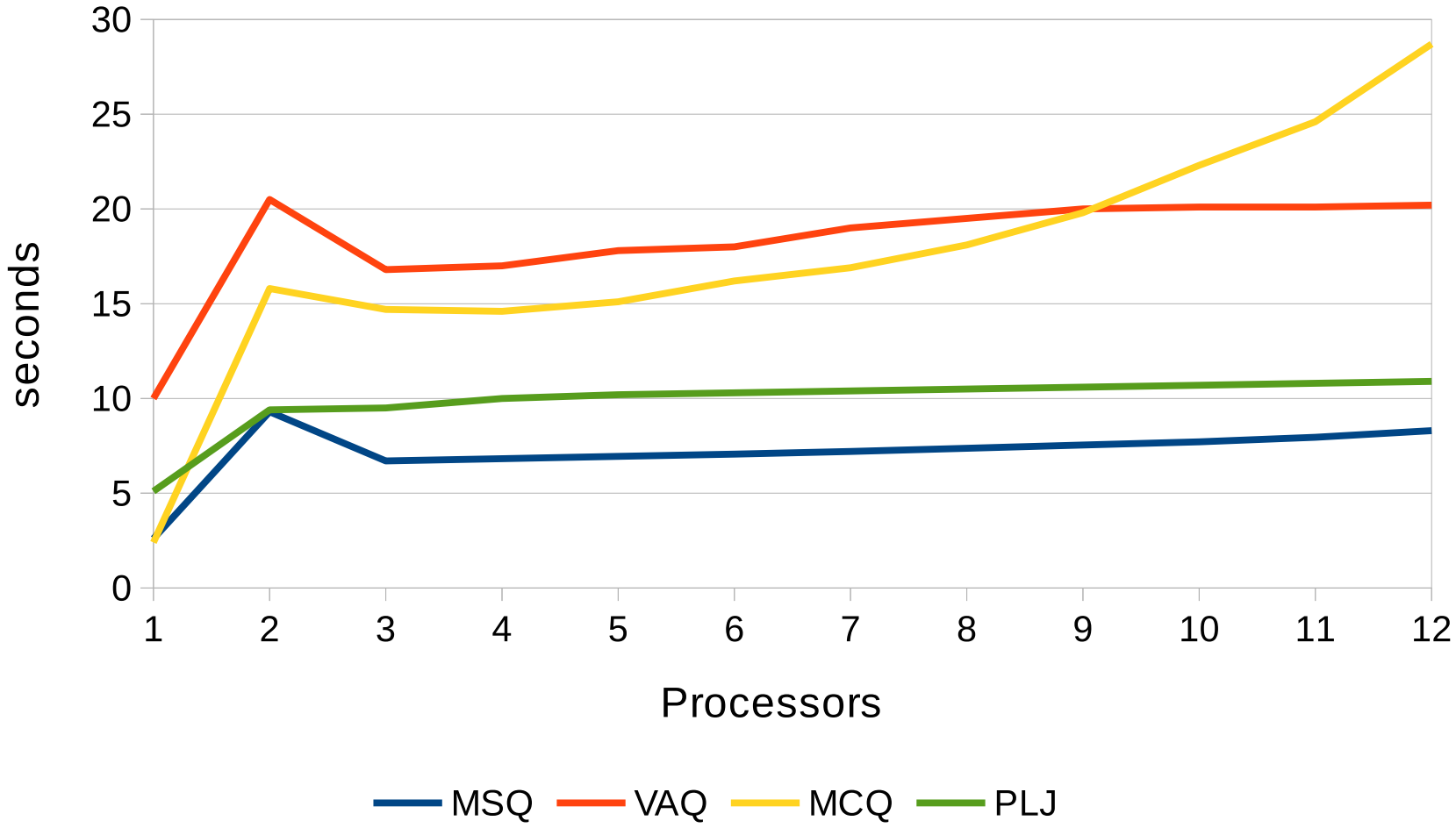
Methodik

- 20.000.000 Elemente
- Thread führt folgendes `num_items/thread_count` mal aus:
  - 1 enqueue
  - “andere” Arbeit
  - 1 dequeue
  - “andere” Arbeit
- “andere” Arbeit ist eine leere Schleife

Testsystem und Queues

- Testsystem:
  - Ubuntu 16.10
  - Intel® Xeon® E3-1230 v2
  - g++ 6.2 (-03)
- Queues (libcds) [Khi17]:
  - Michael Scott Queue (MSQ)
  - Moir Queue (MOQ)
  - Basket Queue (BAQ)
  - Optimistic Queue (OPQ)
  - Lock based Queue (LBQ)

Originale Benchmark



Eigener Benchmark

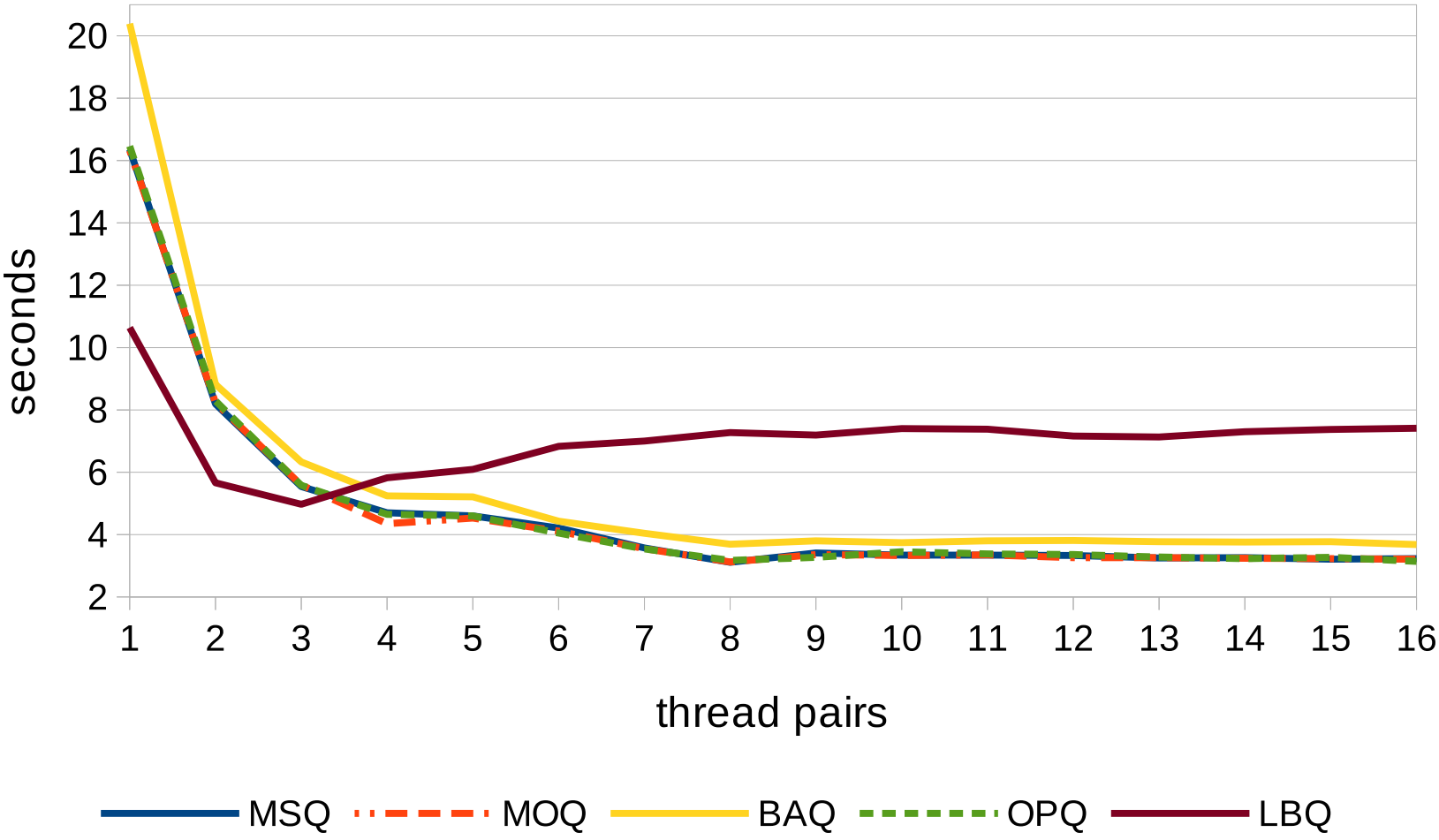


Figure: `iteration_count = 150`

Eigener Benchmark

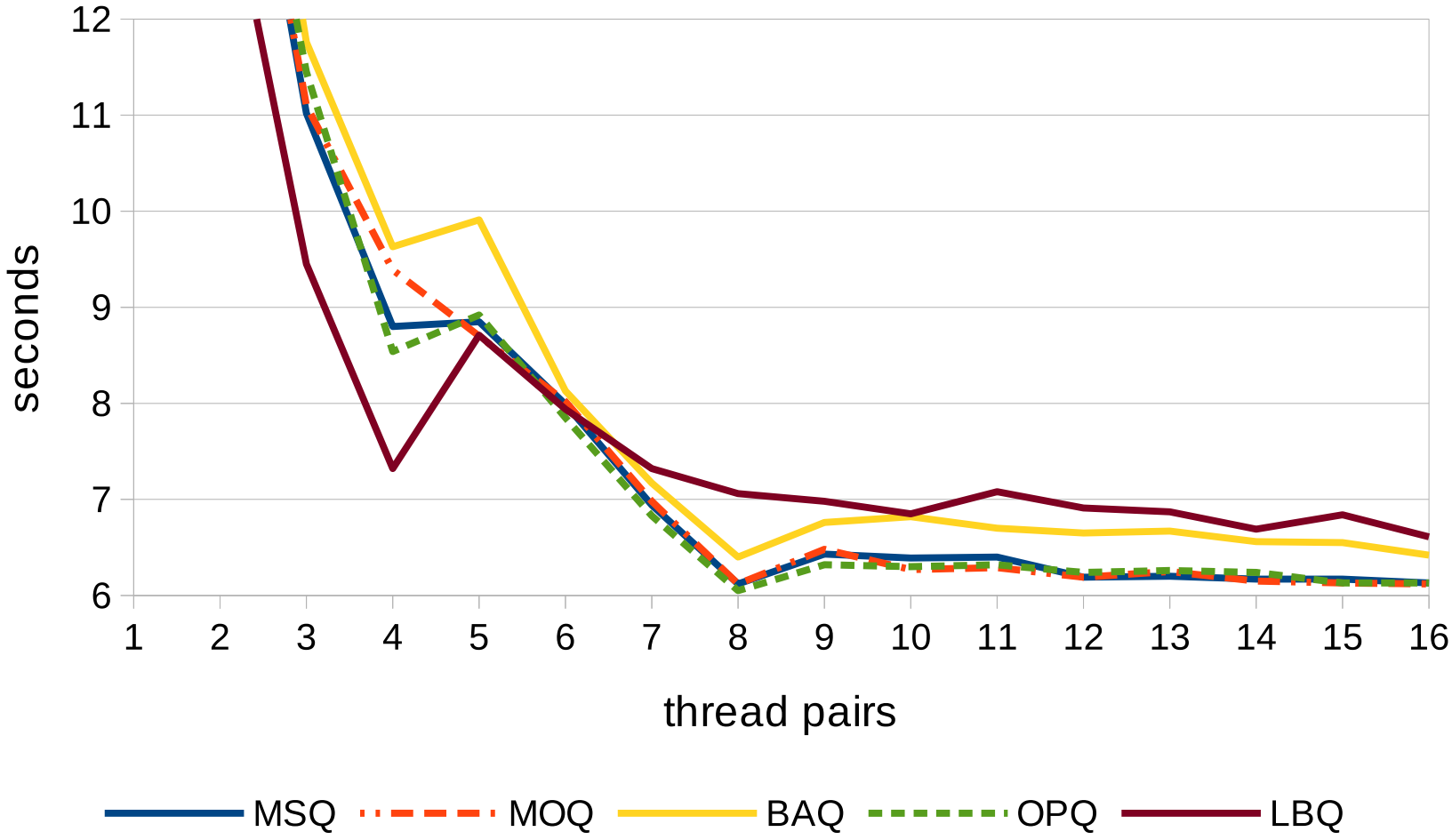
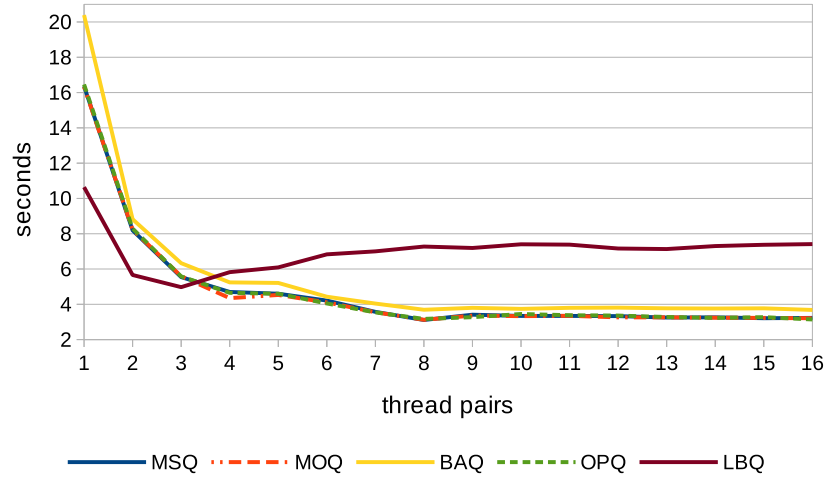


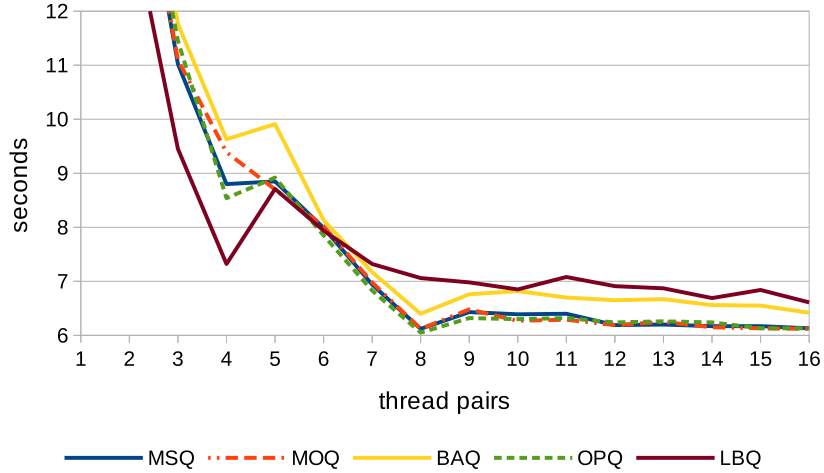
Figure: `iteration_count = 400`

Generelle Erkenntnisse

- Skalieren anfangs fast perfekt
- Lock-free Queues nah beieinander
- Je mehr Arbeit, desto weniger Unterschiede



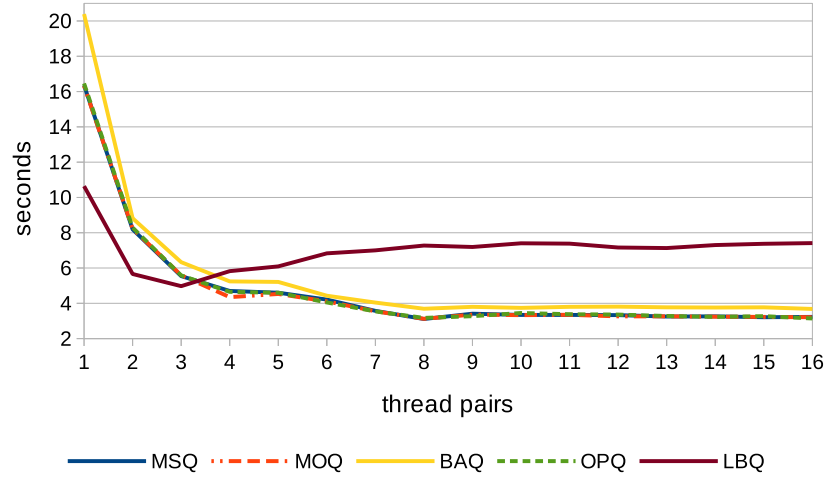
`iteration_count = 150`



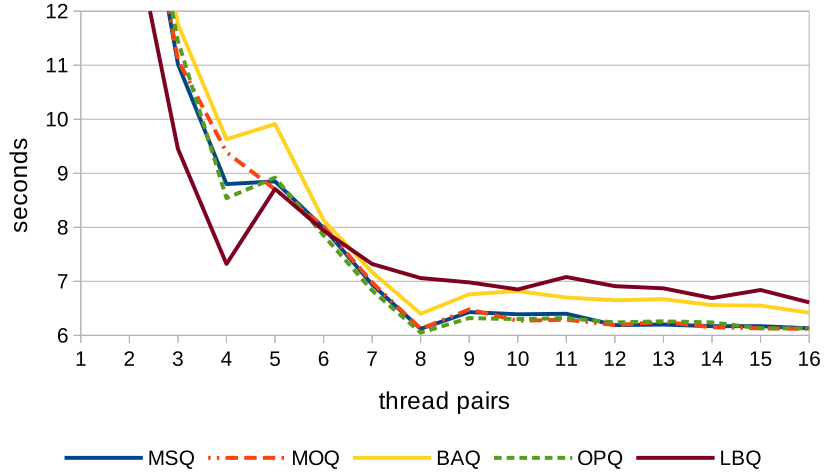
`iteration_count = 400`

Queue-spezifische Erkenntnisse

- MSQ einer der Schnellsten



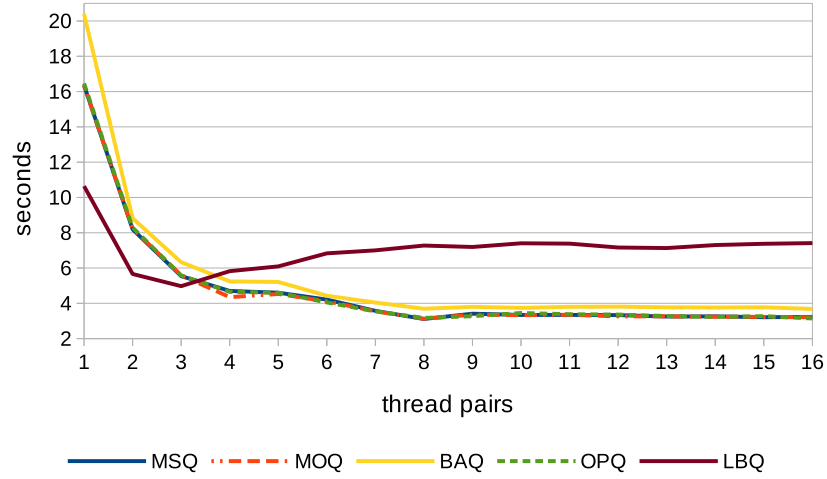
iteration_count = 150



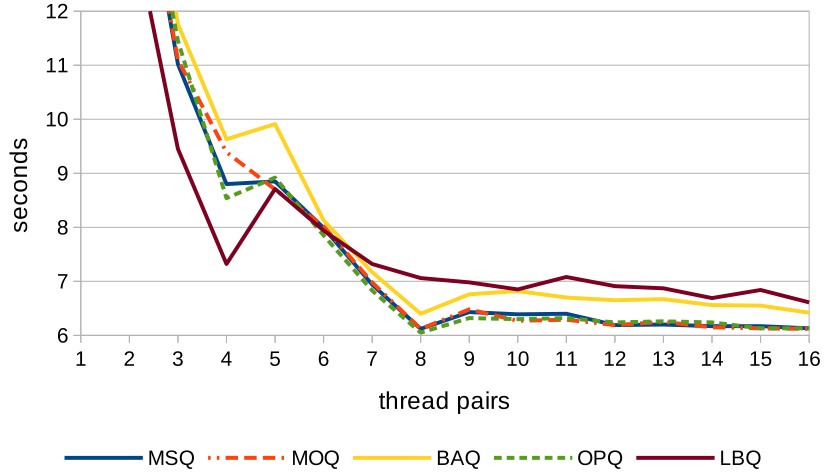
iteration_count = 400

Queue-spezifische Erkenntnisse

- MSQ einer der Schnellsten
- MOQ enthält emuliertes LL/SC



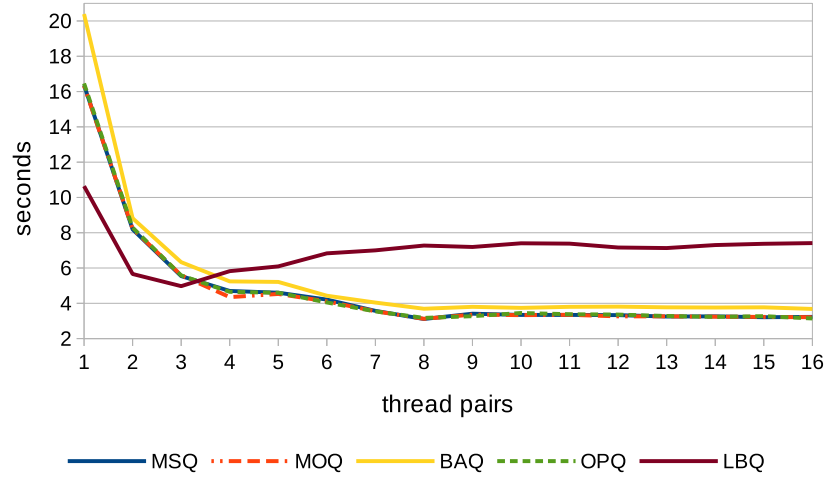
`iteration_count = 150`



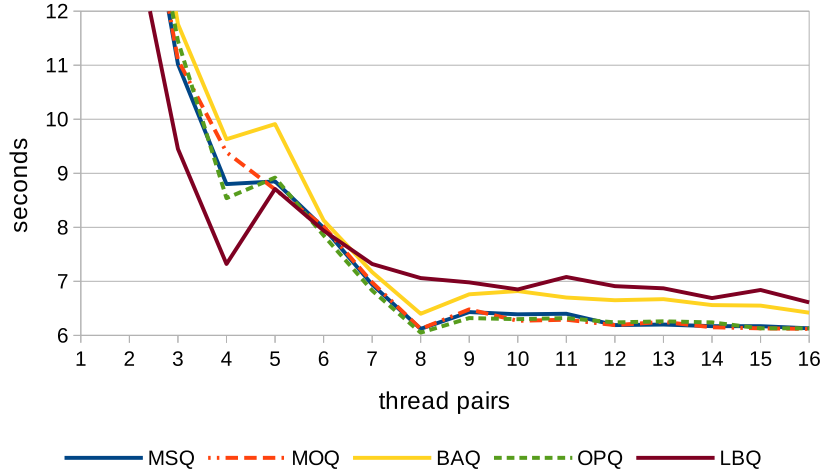
`iteration_count = 400`

Queue-spezifische Erkenntnisse

- MSQ einer der Schnellsten
- MOQ enthält emuliertes LL/SC
- BAQ
  - Verwaltet Buckets mit Elementen
  - Fügt Buckets hinzu im Falle eines Wettstreits



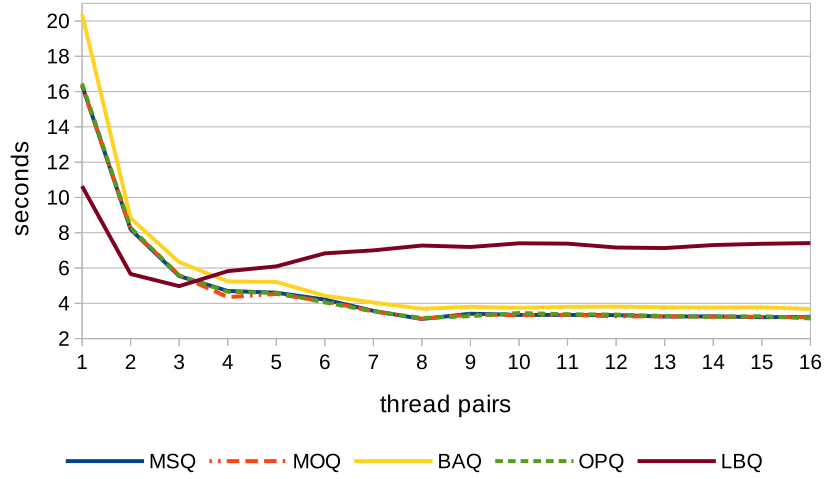
`iteration_count = 150`



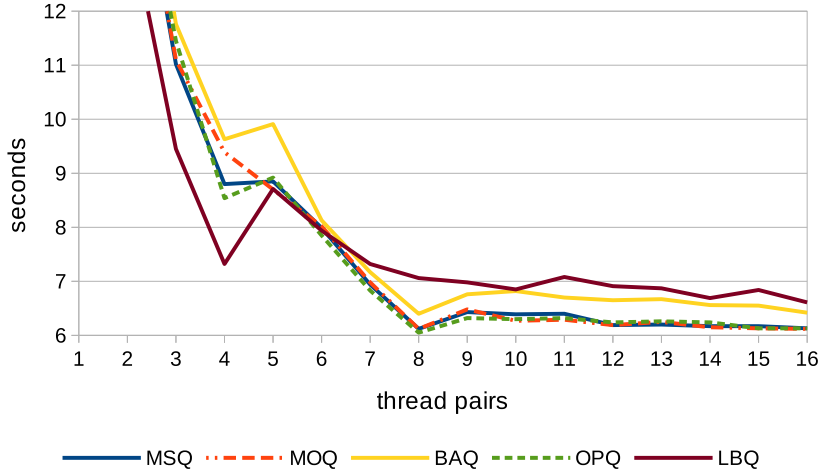
`iteration_count = 400`

Queue-spezifische Erkenntnisse

- MSQ einer der Schnellsten
- MOQ enthält emuliertes LL/SC
- BAQ
  - Verwaltet Buckets mit Elementen
  - Fügt Buckets hinzu im Falle eines Wettstreits
- OPQ
  - Fügt vorne hinzu, entfernt hinten
  - Doppelt verkettete Liste
  - `prev` Pointer wird ohne Synchronisations-Primitive gesetzt
  - Falls Pointer inkonsistent
    → `fixList()`



`iteration_count = 150`



`iteration_count = 400`

Zusammenfassung

- Nicht in jeder Situation ist lock-free nötig
- MSQ schnell, angesichts des Alters
- Jede Queue hat ihre eigenen Vor- und Nachteile

Quellen I

ARM Architecture Reference Manual. https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf. Accessed: 2017-01-16.

C++ International Standard. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4296.pdf. Accessed: 2017-01-16.

Lock-Free Code: A False Sense of Security. http://www.drdobbs.com/cpp/lock-free-code-a-false-sense-of-security/210600279. Accessed: 2017-01-09.

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2 (2A, 2B, 2C & 2D): Instruction Set Reference, A-Z. http://www.intel.de/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf. Accessed: 2017-01-16.

Max Khizhinsky. *A C++ library of Concurrent Data Structures. commit-id: 03601c4b049873992f30f269b131e0f0f19742e5*. https://github.com/khizmax/libcds. 2017.