Wie schnell ist $O(n^2)$?

Algorithmen effizient implementieren

Tim Hegemann

29. Juni 2022

Sollten wir das nicht besser lassen?

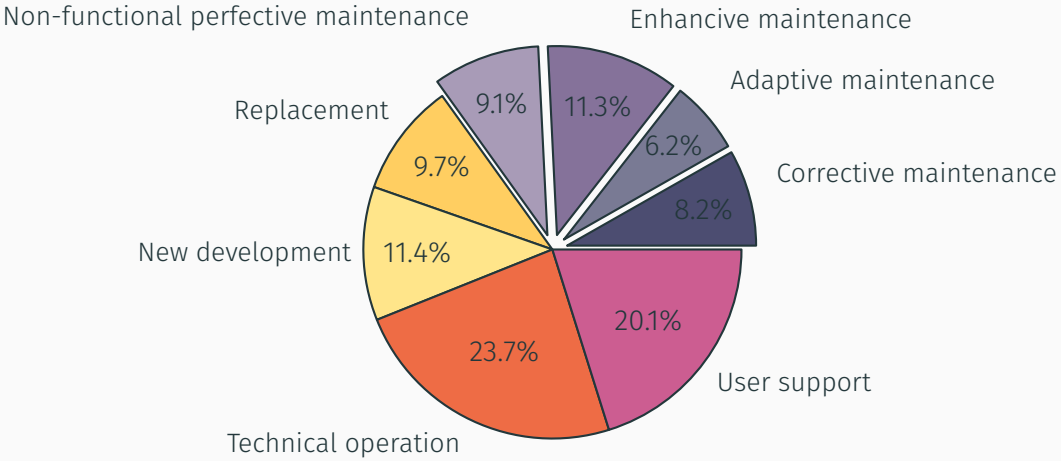
We *should* forget about small efficiencies, say about 97 % of the time:
premature optimization is the root of all evil.

Donald E. Knuth

We *should* forget about small efficiencies, say about 97 % of the time:
premature optimization is the root of all evil.

Donald E. Knuth

Structured Programming with `go to` Statements.
ACM Computing Surveys, 6(4), 1974, S. 261–301



Magne Kristoffer Davidsen und John Krogstie

A longitudinal study of development and maintenance.

Information and Software Technology, 52(7), Elsevier, 2010, S. 707–719.

Code-Quality Indicator

```
def veryFastSort(l: List[Int]): List[Int] =
  // very fast sorting algorithm
  // runs in O(1)
  // always returns a sorted list
  List(1, 2, 3, 4)
```

Code-Quality Indicator

```scala
def veryFastSort(l: List[Int]): List[Int] =
  // very fast sorting algorithm
  // runs in O(1)
  // always returns a sorted list
  List(1, 2, 3, 4)
```

The important point is to always remember Amdahl's law. This states that the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used.

Martin Reddy

API Design for C++. Elsevier, 2011, Kap. 7, S. 210.

The important point is to always remember Amdahl's law. This states that the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used.

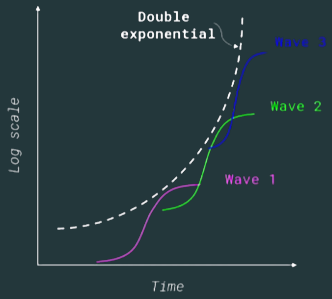
Martin Reddy

API Design for C++. Elsevier, 2011, Kap. 7, S. 210.

Gene M. Amdahl

Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.

AFIPS Conference Proceedings, 30, 1967, S. 483–485.



Law of Diminishing Returns

Individual technology S-curves

+

Law of Accelerating Returns

Sum of sequentially faster S-curves

Jim Keller

Moore's Law is Not Dead.

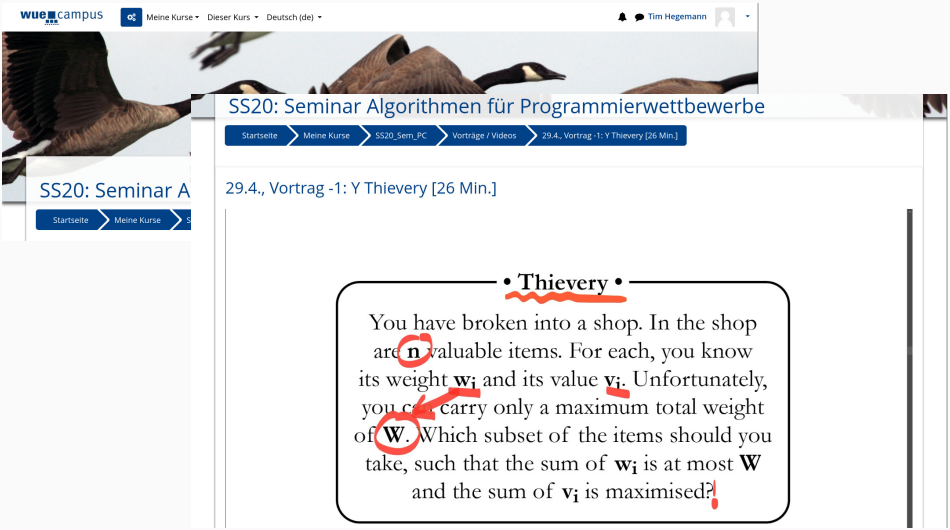
Vortrag im UC Berkeley EECS Colloquium, 18. September 2019.

Checkliste: Sollte ich mit Performance Optimierungen anfangen?

- ☐ Laufzeit ist ein Problem
- ☐ Erfüllt die funktionalen Anforderungen
- ☐ Hinreichender Anteil an Gesamtlaufzeit
- ☐ Algorithmisch optimal
- ☐ Bessere Hardware ist keine Option
- ☐ Bisher unoptimiert

Wie alles begann...


wue campus
Meine Kurse
Dieser Kurs
Deutsch (de)
Tim Hegemann
SS20: Seminar Algorithmen für Programmierwettbewerbe
Startseite
Meine Kurse
SS20_Sem_PC


wue campus
Meine Kurse
Dieser Kurs
Deutsch (de)
Tim Hegemann
SS20: Seminar A
Startseite
Meine Kurse
SS20: Seminar Algorithmen für Programmierwettbewerbe
Startseite
Meine Kurse
SS20_Sem_PC
Vorträge / Videos
29.4., Vortrag -1: Y Thievery [26 Min.]
29.4., Vortrag -1: Y Thievery [26 Min.]
• Thievery •
You have broken into a shop. In the shop are n valuable items. For each, you know its weight w_i and its value v_i. Unfortunately, you can carry only a maximum total weight of W. Which subset of the items should you take, such that the sum of w_i is at most W and the sum of v_i is maximised?

Ich wiederhole, dass alles <7 Sekunden reicht. Aber es könnte @Tim Hegemann & Co. interessieren zu hören, dass seine letzte Einreichung in Rust Laufzeit 0.4 Sekunden ergibt.

• Thievery •

You have broken into a shop. In the shop are n valuable items. For each, you know its weight w_i and its value v_i. Unfortunately, you can carry only a maximum total weight of W. Which subset of the items should you take, such that the sum of w_i is at most W and the sum of v_i is maximised?

Ich wiederhole, dass alles <7 Sekunden reicht. Aber es könnte @Tim Hegemann & Co. interessieren zu hören, dass seine letzte Einreichung in Rust Laufzeit 0.4 Sekunden ergibt.

Wenige Stunden später...

Here's my fastest C++ code

My original 0.6-second solution had a [SPOILER]. [SPOILER].
That took off some time [...]

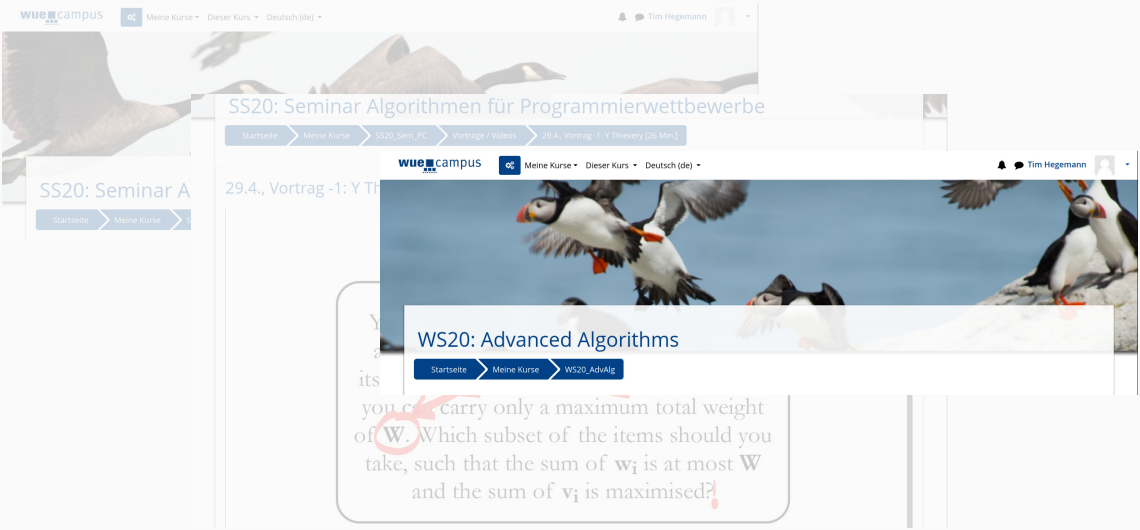
Ich wiederhole, dass alles <7 Sekunden reicht. Aber es könnte @Tim Hegemann & Co. interessieren zu hören, dass seine letzte Einreichung in Rust Laufzeit 0.4 Sekunden ergibt.

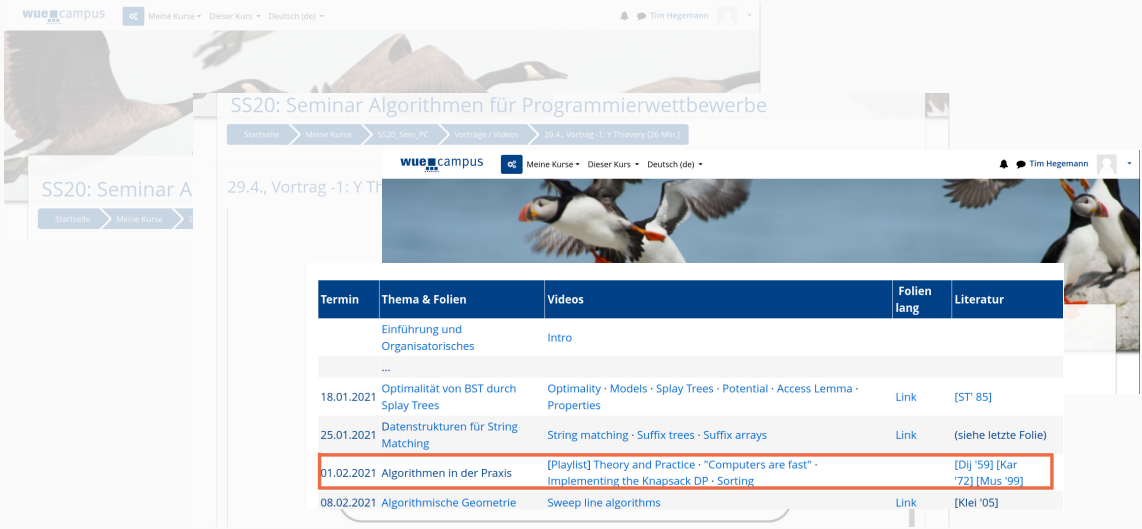
Wenige Stunden später...

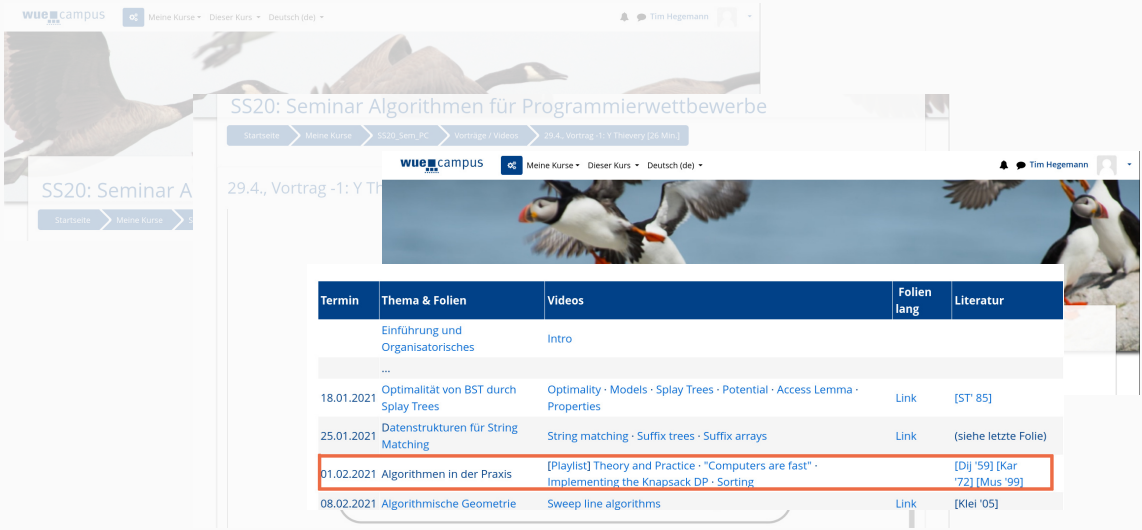
Here's my fastest C++ code

My original 0.6-second solution had a [SPOILER]. [SPOILER].
That took off some time [...]

...


wue campus
Meine Kurse
Dieser Kurs
Deutsch (de)
Tim Hegemann
SS20: Seminar Algorithmen für Programmierwettbewerbe
Startseite
Meine Kurse
SS20_Sem_PC
Vorträge / Videos
29.4., Vortrag -1: Y Thievery [26 Min.]
SS20: Seminar A
Startseite
Meine Kurse
29.4., Vortrag -1: Y Th
its
you c carry only a maximum total weight
of W. Which subset of the items should you
take, such that the sum of w_i is at most W
and the sum of v_i is maximised?
wue campus
Meine Kurse
Dieser Kurs
Deutsch (de)
Tim Hegemann
WS20: Advanced Algorithms
Startseite
Meine Kurse
WS20_AdvAlg


wue campus
Meine Kurse
Dieser Kurs
Deutsch (de)
Tim Hegemann
SS20: Seminar Algorithmen für Programmierwettbewerbe
Startseite
Meine Kurse
SS20_Sem_PC
Vorträge / Videos
29.4., Vortrag -1: Y Thievery [26 Min.]
SS20: Seminar A
29.4., Vortrag -1: Y Th
Termin
Thema & Folien
Videos
Folien lang
Literatur
Einführung und Organisatorisches
Intro
...
18.01.2021
Optimalität von BST durch Splay Trees
Optimality · Models · Splay Trees · Potential · Access Lemma · Properties
Link
[ST' 85]
25.01.2021
Datenstrukturen für String Matching
String matching · Suffix trees · Suffix arrays
Link
(siehe letzte Folie)
01.02.2021
Algorithmen in der Praxis
[Playlist] Theory and Practice · "Computers are fast" · Implementing the Knapsack DP · Sorting
[Dij '59] [Kar '72] [Mus '99]
08.02.2021
Algorithmische Geometrie
Sweep line algorithms
Link
[Klei '05]



Implementing the Knapsack DP

Einfach mal losimplementieren

Die DP-Formel als Code

$$f(1, w) = \begin{cases} v_1 & \text{if } w_1 \leq w \\ 0 & \text{otherwise} \end{cases}$$

$$f(i, w) = max \begin{cases} f(i-1, w) \\ v_i + f(i-1, w - w_i) & \text{if } w - w_i \geq 0 \end{cases}$$

Die DP-Formel als Code

```scala
case class Item(weight: Int, value: Int)

def solve(items: IndexedSeq[Item], weight: Int) =
  def go(i: Int, weight: Int): Int =
    val item = items(i)
    if i == 0 then
      if item.weight ⩽ weight then item.value else 0
    else if item.weight ⩽ weight then
      go(i - 1, weight) max
          (go(i - 1, weight - item.weight) + item.value)
    else go(i - 1, weight)

  go(items.size - 1, weight)
```

Die DP-Formel als Code

```scala
case class Item(weight: Int, value: Int)

def solve(items: IndexedSeq[Item], weight: Int) =
  def go(i: Int, weight: Int): Int =
    val item = items(i)
    if i == 0 then
      if item.weight ⩽ weight then item.value else 0
    else if item.weight ⩽ weight then
      go(i - 1, weight) max
          (go(i - 1, weight - item.weight) + item.value)
    else go(i - 1, weight)

  go(items.size - 1, weight)
```

Laufzeiten – Erwartungshaltungsmanagement

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Figure 2.4: Growth rates of common functions measured in nanoseconds

Steven S. Skiena

The Algorithm Design Manual. 2nd Edition, Springer, 2012, S. 38.

TODO: Baseline Runtimes + how to measure

Laufzeiten

Instanzgröße n	30
Scala naïve	

Laufzeiten

Instanzgröße n	30
Scala naïve	1.15 s

Laufzeiten

Instanzgröße n	30	100
Scala naïve	1.15 s	

Die DP-Formel als Code

```scala
case class Item(weight: Int, value: Int)

def solve(items: IndexedSeq[Item], weight: Int) =
  def go(i: Int, weight: Int): Int =
    val item = items(i)
    if i == 0 then
      if item.weight ⩽ weight then item.value else 0
    else if item.weight ⩽ weight then
      go(i - 1, weight) max
          (go(i - 1, weight - item.weight) + item.value)
    else go(i - 1, weight)

  go(items.size - 1, weight)
```

Naive Implementierung – Laufzeitanalyse

$$f(1) = 1$$

$$\begin{aligned} f(n) &= f(n-1) + f(n-1) \\ &= 2f(n-1) \\ &= \end{aligned}$$

Naive Implementierung – Laufzeitanalyse

$$f(1) = 1$$

$$\begin{aligned} f(n) &= f(n-1) + f(n-1) \\ &= 2f(n-1) \\ &= 2^n \end{aligned} \qquad \in \mathcal{O}(2^n)$$

Memoisation

```
def momoized[K, V](f: K ⇒ V): K ⇒ V =
  val dict = mutable.Map.empty[K, V]
  (key: K) ⇒ dict.getOrElseUpdate(key, f(key))
```

Die DP-Formel als Code – mit Memoisation

```
lazy val go: ((Int,Int)) ⇒ Int = momoized((args: (Int,Int)) ⇒
  val (i, weight) = args
  val item = items(i)
  if i == 0 then if item.weight ⩽ weight then item.value else 0
  else if item.weight ⩽ weight then
    go(i - 1, weight) max
        (go(i - 1, weight - item.weight) + item.value)
  else go(i - 1, weight)
)

go(items.size - 1, weight)
```

Die DP-Formel als Code – mit Memoisation

```
lazy val go: ((Int,Int)) ⇒ Int = momoized((args: (Int,Int)) ⇒
  val (i, weight) = args
  val item = items(i)
  if i == 0 then if item.weight ⩽ weight then item.value else 0
  else if item.weight ⩽ weight then
    go(i - 1, weight) max
        (go(i - 1, weight - item.weight) + item.value)
  else go(i - 1, weight)
)

go(items.size - 1, weight)
```

Laufzeiten

Instanzgröße n	30	100
Scala naïve	1.15 s	💀
Scala memoized	230 ms	

Laufzeiten

Instanzgröße n	30	100
Scala naïve	1.15 s	💀
Scala memoized	230 ms	267 ms

Laufzeiten

Instanzgröße n	30	100	1K
Scala naïve	1.15 s	💀	
Scala memoized	230 ms	267 ms	…

```
Exception in thread "main" java.lang.StackOverflowError
    at scala.runtime.BoxesRunTime.equalsNumNum(BoxesRunTime.java:148)
    at scala.runtime.BoxesRunTime.equalsNumObject(BoxesRunTime.java:138)
    at scala.runtime.BoxesRunTime.equals2(BoxesRunTime.java:127)
    at scala.runtime.BoxesRunTime.equals(BoxesRunTime.java:119)
    at scala.Tuple2.equals(Tuple2.scala:24)
    at scala.runtime.BoxesRunTime.equals2(BoxesRunTime.java:133)
    at scala.runtime.BoxesRunTime.equals(BoxesRunTime.java:119)
    at scala.collection.mutable.HashMap$Node.findNode(HashMap.scala:621)
    at scala.collection.mutable.HashMap.getOrElseUpdate(HashMap.scala:449)
    at y.Memoized$.momoized$$anonfun$1(Y.scala:64)
    at y.Memoized$.go$lzyINIT1$1$$anonfun$1(Y.scala:72)
    at y.Memoized$.momoized$$anonfun$1$$anonfun$1(Y.scala:64)
    at scala.collection.mutable.HashMap.getOrElseUpdate(HashMap.scala:454)
    at y.Memoized$.momoized$$anonfun$1(Y.scala:64)
    ...
```

Tail-Call Optimization (TCO)

Tail-Call Optimization (TCO)

```
enum StackItem:
  case DontTake(i: Int, weight: Int)
  case Take(dontTakeValue: Int, i: Int, weight: Int)
```

Tail-Call Optimization (TCO)

```
enum StackItem:
  case DontTake(i: Int, weight: Int)
  case Take(dontTakeValue: Int, i: Int, weight: Int)

val cache = mutable.Map.empty[(Int, Int), Int]
```

Tail-Call Optimization (TCO)

```
enum StackItem:
  case DontTake(i: Int, weight: Int)
  case Take(dontTakeValue: Int, i: Int, weight: Int)

val cache = mutable.Map.empty[(Int, Int), Int]

@tailrec def go(stack: List[StackItem]): Unit = stack match
  case Nil                    ⇒ // finished \o/
  case DontTake(0, w) :: tail ⇒ go(Take(0, 0, w) :: tail)
  case Take(_, 0, w) :: tail ⇒
    cache += (0, w) → (
        if items(0).weight ⩽ w then items(0).value else 0
    )
    go(tail)
//  ...
```

Tail-Call Optimization (TCO)

```
enum StackItem:
  case DontTake(i: Int, weight: Int)
  case Take(dontTakeValue: Int, i: Int, weight: Int)

val cache = mutable.Map.empty[(Int, Int), Int]

@tailrec def go(stack: List[StackItem]): Unit = stack match
// ...
  case DontTake(i, w) :: tail ⇒
    cache.get(i - 1 → w) match
      case None    ⇒ go(DontTake(i - 1, w) :: stack)
      case Some(v) ⇒ go(Take(v, i, w) :: tail)
// ...
```

Tail-Call Optimization (TCO)

```
val cache = mutable.Map.empty[(Int, Int), Int]

@tailrec def go(stack: List[StackItem]): Unit = stack match
// ...
  case Take(dontTake, i, w) :: tail ⇒
    val item = items(i)
    if item.weight ⩽ w then
      cache.get(i - 1 → (w - item.weight)) match
        case None ⇒ go(DontTake(i-1, w-item.weight) :: stack)
        case Some(take) ⇒
          cache += (i, w) → (dontTake max (take + item.value))
          go(tail)
    else
      cache += (i, w) → dontTake
      go(tail)
```

Tail-Call Optimization (TCO)

```
val cache = mutable.Map.empty[(Int, Int), Int]

@tailrec def go(stack: List[StackItem]): Unit = // ...

go(DontTake(items.size - 1, weight) :: Nil)
cache(items.size - 1 → weight)
```

Tail-Call Optimization (TCO)

```
val cache = mutable.Map.empty[(Int, Int), Int]

@tailrec def go(stack: List[StackItem]): Unit = // ...

go(DontTake(items.size - 1, weight) :: Nil)
cache(items.size - 1 → weight)
```

Laufzeiten

Instanzgröße *n*	30	100	1K
Scala naïve	1.15 s	💀	
Scala memoized	230 ms	267 ms	⚡SO
Scala memoized TCO		273 ms	

Laufzeiten

Instanzgröße *n*	30	100	1K
Scala naïve	1.15 s	💀	
Scala memoized	230 ms	267 ms	⚡SO
Scala memoized TCO		273 ms	2.30 s

Laufzeiten

Instanzgröße *n*	30	100	1K	10K
Scala naïve	1.15 s	💀		
Scala memoized	230 ms	267 ms	⚡SO	
Scala memoized TCO		273 ms	2.30 s	…

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.Integer.valueOf(Integer.java:1077)
    at scala.runtime.BoxesRunTime.boxToInteger(BoxesRunTime.java:63)
    at y.TCO$.go$3(Y.scala:112)
    at y.TCO$.solve(Y.scala:121)
    at y.TCO$.runTco(Y.scala:82)
    at y.runTco.main(Y.scala:80)
```

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen...

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Die Map ist doof, da können wir einen 2D-Array nehmen stattdessen.

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Die Map ist doof, da können wir einen 2D-Array nehmen stattdessen.

Der 2D-Array ist doof, da können wir einen 1D-Array nehmen stattdessen.

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Die Map ist doof, da können wir einen 2D-Array nehmen stattdessen.

Der 2D-Array ist doof, da können wir einen 1D-Array nehmen stattdessen.

Scala ist doof, da können wir $SYSTEM_LANGUAGE nehmen stattdessen.

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Die Map ist doof, da können wir einen 2D-Array nehmen stattdessen.

Der 2D-Array ist doof, da können wir einen 1D-Array nehmen stattdessen.

Scala ist doof, da können wir $SYSTEM_LANGUAGE nehmen stattdessen.

Wenn euch das interessiert, schaut Thomas' Video!

Es ist nicht hoffnungslos!

Man kann hier jetzt weitermachen…

Rekursion ist doof, da können wir iterativ bottom-up vorgehen stattdessen.

Die Map ist doof, da können wir einen 2D-Array nehmen stattdessen.

Der 2D-Array ist doof, da können wir einen 1D-Array nehmen stattdessen.

Scala ist doof, da können wir $SYSTEM_LANGUAGE nehmen stattdessen.

Wenn euch das interessiert, schaut Thomas' Video!

Wir überspringen das aber…

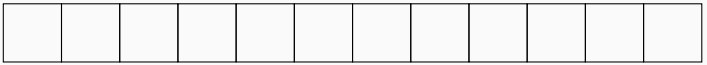
Nochmal Theorie

$$f(i,w) \;=\; max \begin{cases} f(\; i-1\;, w) \\ v_i + f(\; i-1\;, w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$

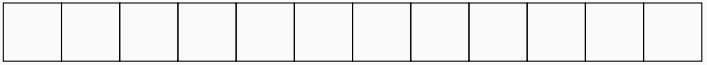
$$f(i,w) \;=\; max \begin{cases} f(\boxed{i-1}, w) \\ v_i + f(\boxed{i-1}, w - w_i) \quad \text{if } w - w_i \geq 0 \end{cases}$$

$$f(i,w) \;=\; max \begin{cases} f(\boxed{i-1}, w) \\ v_i + f(\boxed{i-1}, w - w_i) \quad \text{if } w - w_i \geq 0 \end{cases}$$

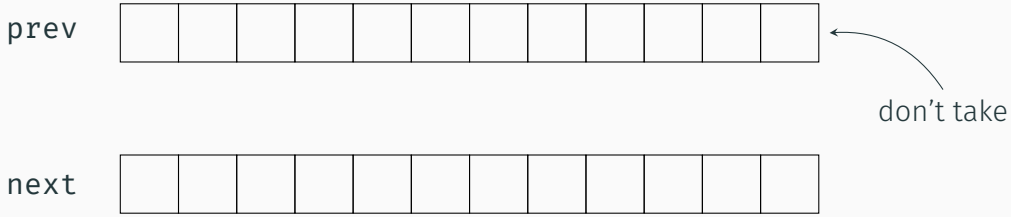
prev



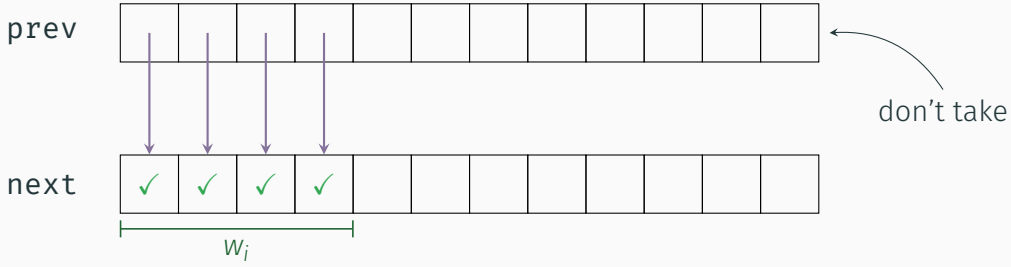
next



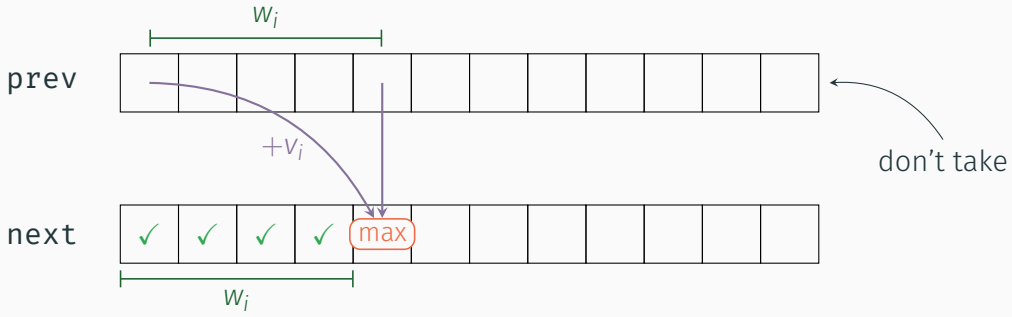
$$f(i,w) \;=\; max \begin{cases} f(\ i-1\ ,w) & \longleftarrow \text{dont't take} \\ v_i + f(\ i-1\ ,w-w_i) & \text{if } w-w_i \geq 0 \end{cases}$$



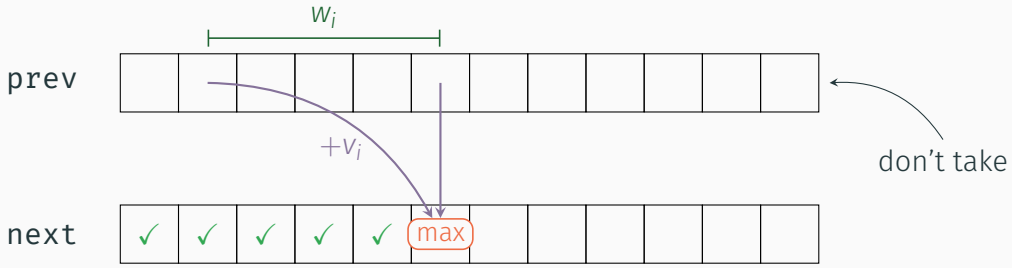
$$f(i,w) \;=\; max \begin{cases} f(\; i-1\;, w) \\ v_i + f(\; i-1\;, w-w_i) \quad \text{if } w - w_i \geq 0 \end{cases}$$



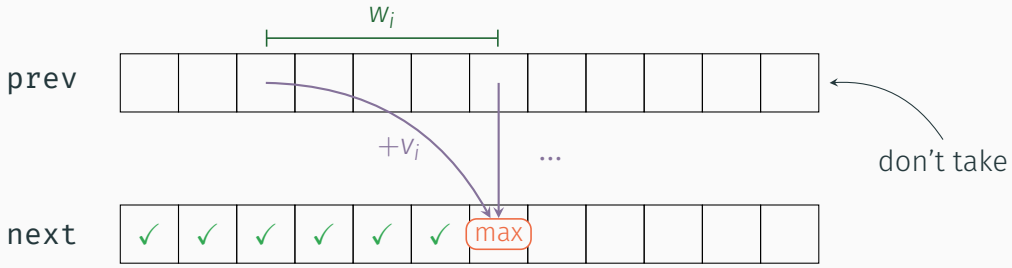
$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



Eine schnelle funktionale Lösung

```
val table = items match
  case Nil ⇒ sys.error("cannot operate on empty input")
  case h :: tail ⇒
    tail.foldLeft(
      List.fill(h.weight)(0)
        ::: List.fill(weight - h.weight + 1)(h.value)
    )((prev, item) ⇒
      prev.take(item.weight)
        ::: (prev.drop(item.weight) zip prev)
          .map((dontTake, take) ⇒
            dontTake max (take + item.value))
    )

table.last
```

Eine schnelle funktionale Lösung

```
val table = items match
  case Nil ⇒ sys.error("cannot operate on empty input")
  case h :: tail ⇒
    tail.foldLeft(
      List.fill(h.weight)(0)
        ::: List.fill(weight - h.weight + 1)(h.value)
    )((prev, item) ⇒
      prev.take(item.weight)
        ::: (prev.drop(item.weight) zip prev)
          .map((dontTake, take) ⇒
            dontTake max (take + item.value))
    )

table.last
```

Laufzeiten

Instanzgröße *n*	30	100	1K	10K
Scala naïve	1.15 s	💀		
Scala memoized	230 ms	267 ms	⚡SO	
Scala memoized TCO		273 ms	2.30 s	⚡OOM
Scala pure			348 ms	

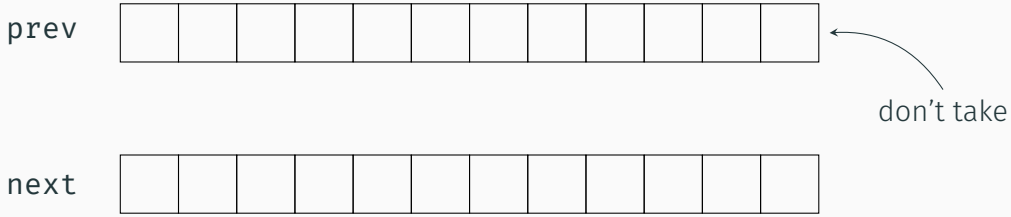
Laufzeiten

Instanzgröße *n*	30	100	1K	10K
Scala naïve	1.15 s	💀		
Scala memoized	230 ms	267 ms	⚡SO	
Scala memoized TCO		273 ms	2.30 s	⚡OOM
Scala pure			348 ms	7.89 s

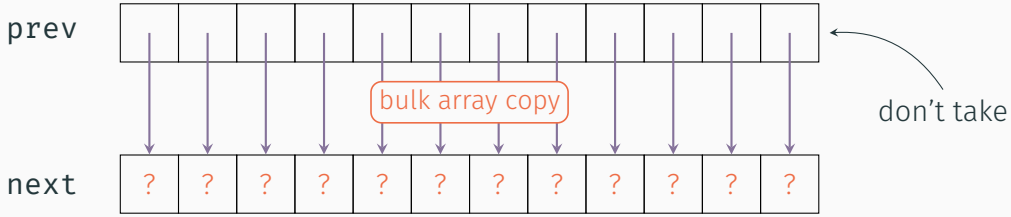
Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	

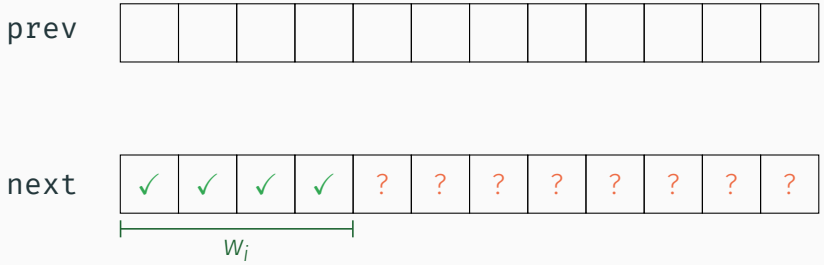
$$f(i,w) \;=\; max \begin{cases} f(\; i-1\;, w) \\ v_i + f(\; i-1\;, w-w_i) \quad \text{if } w - w_i \geq 0 \end{cases}$$



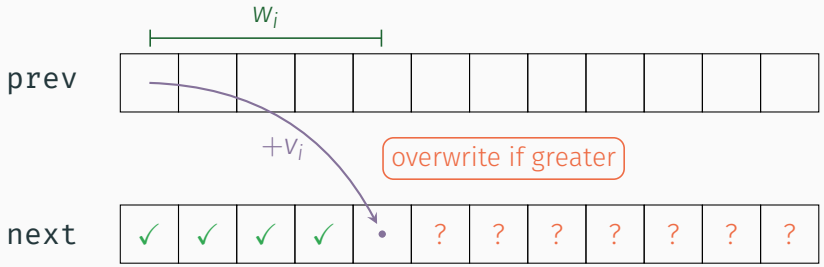
$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



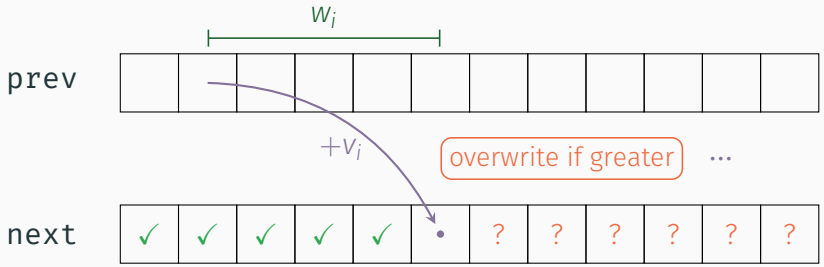
$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$

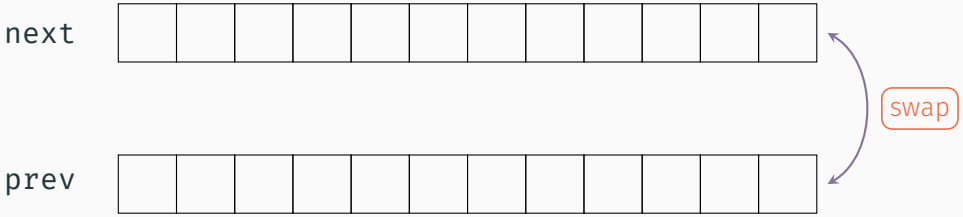


$$f(i,w) \;=\; max \begin{cases} f(\,i-1\,,w) \\ v_i + f(\,i-1\,,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



$$f(i,w) \;=\; max \begin{cases} f(\;i-1\;,w) \\ v_i + f(\;i-1\;,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$

$$f(i,w) \;=\; max \begin{cases} f(\;i-1\;,w) \\ v_i + f(\;i-1\;,w-w_i) \quad \text{if } w-w_i \geq 0 \end{cases}$$



Klassische imperative Programmierung

```
def solve(items: Seq[Item], weight: Int) =
  var prev = Array.fill(weight + 1)(0)
  var next = Array.fill(weight + 1)(0)
  val head = items.head
  Arrays.fill(prev, head.weight, weight + 1, head.value)
  for item ← items.iterator.drop(1) do
    Array.copy(prev, 0, next, 0, prev.length)
    var w = item.weight
    while w < next.length do
      val take = prev(w - item.weight) + item.value
      if next(w) < take then next(w) = take
      w += 1
    val tmp = next
    next = prev
    prev = tmp
  prev.last
```

Klassische imperative Programmierung

```
def solve(items: Seq[Item], weight: Int) =
  var prev = Array.fill(weight + 1)(0)
  var next = Array.fill(weight + 1)(0)
  val head = items.head
  Arrays.fill(prev, head.weight, weight + 1, head.value)
  for item ← items.iterator.drop(1) do
    Array.copy(prev, 0, next, 0, prev.length)
    var w = item.weight
    while w < next.length do
      val take = prev(w - item.weight) + item.value
      if next(w) < take then next(w) = take
      w += 1
    val tmp = next
    next = prev
    prev = tmp
  prev.last
```

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s

Battle of the Languages

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s

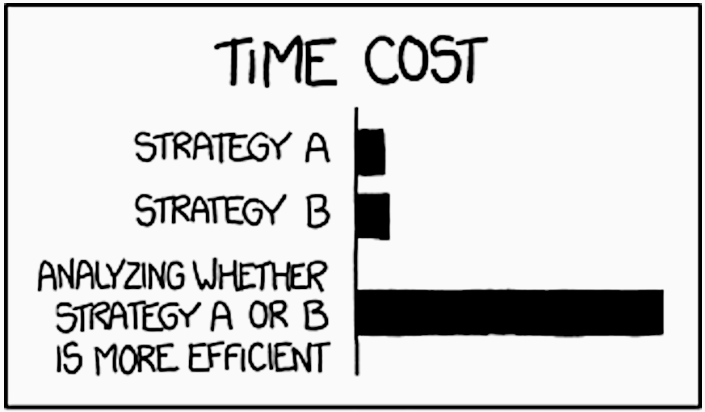
Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s

Performance Probleme Analysieren



Randall Munroe

 xkcd.com/1445

Das wichtigste haben wir schon hinter uns:
Laufzeit messen

Zwei Fragen bleiben

Zwei Fragen bleiben

Warum war Java langsamer als Scala?

Zwei Fragen bleiben

Warum war Java langsamer als Scala?

Warum war Rust (so viel) langsamer als C++?

Zwei Fragen bleiben

Warum war Java langsamer als Scala?

Warum war Rust (so viel) langsamer als C++?

```java
static int solve(List<Item> items, int weight) {
    var prev = new int[weight + 1];
    var next = new int[weight + 1];
    var head = items.get(0);
    Arrays.fill(prev, head.weight, weight + 1, head.value);
    for (var item : items) {
        if (item == head) continue; // just skip the first
        System.arraycopy(prev, 0, next, 0, prev.length);
        for (var w = item.weight; w < next.length; w++) {
            var v = prev[w - item.weight] + item.value;
            if (next[w] < v) next[w] = v;
        }
        var tmp = next;
        next = prev;
        prev = tmp;
    }
    return prev[weight];
}
```

```java
static int solve(List<Item> items, int weight) {
    var prev = new int[weight + 1];
    var next = new int[weight + 1];
    var head = items.get(0);
    Arrays.fill(prev, head.weight, weight + 1, head.value);
    for (var item : items) {
        if (item == head) continue; // just skip the first
        System.arraycopy(prev, 0, next, 0, prev.length);
        for (var w = item.weight; w < next.length; w++) {
            var v = prev[w - item.weight] + item.value;
            if (next[w] < v) next[w] = v;
        }
        var tmp = next;
        next = prev;
        prev = tmp;
    }
    return prev[weight];
}
```

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

```
static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

```
static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

...

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

```
static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

...

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

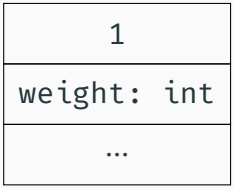
```
static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

weight: int
…

Was passiert da eigentlich genau?

```
$> javap -p -v Y

static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```



Was passiert da eigentlich genau?

```
$> javap -p -v Y

static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

weight + 1
…

Was passiert da eigentlich genau?

```
$> javap -p -v Y
```

```
static int solve(java.util.List<Y$Item>, int);
  descriptor: (Ljava/util/List;I)I
  flags: (0×0008) ACC_STATIC
  Code:
    stack=5, locals=9, args_size=2
       0: iload_1
       1: iconst_1
       2: iadd
       3: newarray       int
[ ... ]
```

prev: [int]
…

```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

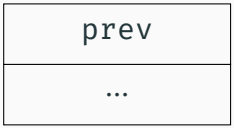
```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

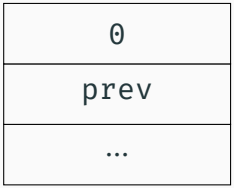
```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
[ ... ]
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
```

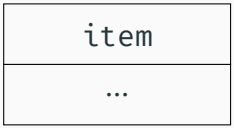
```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
 92: aload_2
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
111: arraylength
112: if_icmpge     154
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

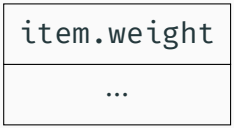
```
 93: iconst_0
 94: aload_3
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arraycopy
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
111: arraylength
112: if_icmpge     154
115: aload_2
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
 95: iconst_0
 96: aload_2
 97: arraylength
 98: invokestatic  #65 // System.arra
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
111: arraylength
112: if_icmpge     154
115: aload_2
116: iload         7
118: aload         6
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

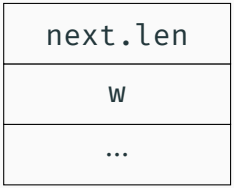


```
 97: arraylength
 98: invokestatic  #65 // System.arra
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
111: arraylength
112: if_icmpge     154
115: aload_2
116: iload         7
118: aload         6
120: getfield      #47 // Item.weight
123: isub
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
 98: invokestatic  #65 // System.arra
101: aload         6
103: getfield      #47 // Item.weight
106: istore        7
108: iload         7
110: aload_3
111: arraylength
112: if_icmpge     154
115: aload_2
116: iload         7
118: aload         6
120: getfield      #47 // Item.weight
123: isub
124: iaload
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
108: iload          7
110: aload_3
111: arraylength
112: if_icmpge      154
115: aload_2
116: iload          7
118: aload          6
120: getfield       #47 // Item.weight
123: isub
124: iaload
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

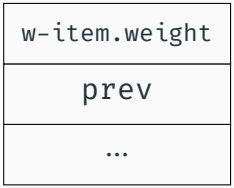
...

```
110: aload_3
111: arraylength
112: if_icmpge      154
115: aload_2
116: iload          7
118: aload          6
120: getfield       #47 // Item.weight
123: isub
124: iaload
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
133: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
111: arraylength
112: if_icmpge     154
115: aload_2
116: iload         7
118: aload         6
120: getfield      #47 // Item.weight
123: isub
124: iaload
125: aload         6
127: getfield      #48 // Item.value
130: iadd
131: istore        8
133: aload_3
134: iload         7
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

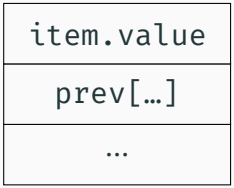


```
115: aload_2
116: iload          7
118: aload          6
120: getfield       #47 // Item.weight
123: isub
124: iaload
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
133: aload_3
134: iload          7
136: iaload
137: iload          8
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
118: aload          6
120: getfield       #47 // Item.weigh
123: isub
124: iaload
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
133: aload_3
134: iload          7
136: iaload
137: iload          8
139: if_icmpge      148
142: aload_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

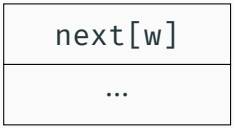


```
124: iaload
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
133: aload_3
134: iload          7
136: iaload
137: iload          8
139: if_icmpge      148
142: aload_3
143: iload          7
145: iload          8
147: iastore
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

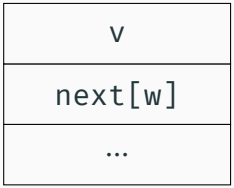
```
125: aload          6
127: getfield       #48 // Item.value
130: iadd
131: istore         8
133: aload_3
134: iload          7
136: iaload
137: iload          8
139: if_icmpge      148
142: aload_3
143: iload          7
145: iload          8
147: iastore
148: iinc           7, 1
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
127: getfield      #48 // Item.value
130: iadd
131: istore        8
133: aload_3
134: iload         7
136: iaload
137: iload         8
139: if_icmpge     148
142: aload_3
143: iload         7
145: iload         8
147: iastore
148: iinc          7, 1
151: goto          108
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



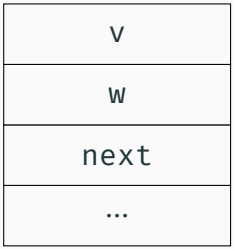
```
133: aload_3
134: iload          7
136: iaload
137: iload          8
139: if_icmpge      148
142: aload_3
143: iload          7
145: iload          8
147: iastore
148: iinc           7, 1
151: goto           108
154: aload_3
155: astore         8
157: aload_2
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

...

```
134: iload          7
136: iaload
137: iload          8
139: if_icmpge      148
142: aload_3
143: iload          7
145: iload          8
147: iastore
148: iinc           7, 1
151: goto           108
154: aload_3
155: astore         8
157: aload_2
158: astore_3
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```



```
137: iload          8
139: if_icmpge      148
142: aload_3
143: iload          7
145: iload          8
147: iastore
148: iinc           7, 1
151: goto           108
154: aload_3
155: astore         8
157: aload_2
158: astore_3
159: aload          8
161: astore_2
```

```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

...

```
139: if_icmpge     148
142: aload_3
143: iload         7
145: iload         8
147: iastore
148: iinc          7, 1
151: goto          108
154: aload_3
155: astore        8
157: aload_2
158: astore_3
159: aload         8
161: astore_2
[ ... ]
```

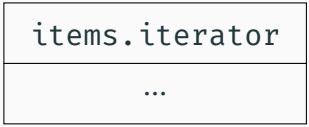
```
System.arraycopy(prev, 0, next, 0, prev.length);
for (var w = item.weight; w < next.length; w++) {
    var v = prev[w - item.weight] + item.value;
    if (next[w] < v) next[w] = v;
}
var tmp = next;
next = prev;
prev = tmp;
```

```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload         4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

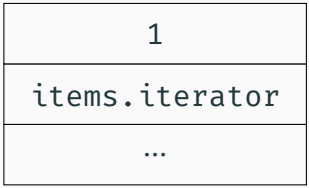
```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload         4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

...

```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload          4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

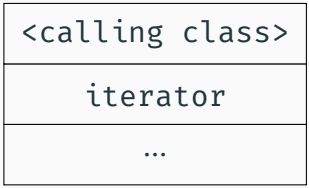


```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload          4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

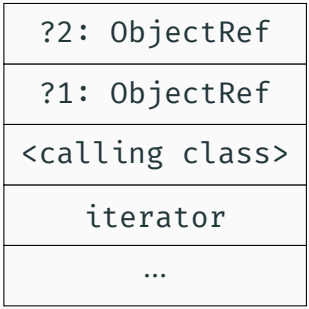


```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload         4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

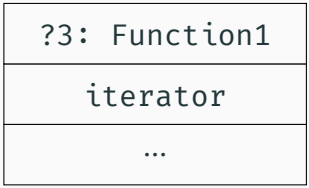
```
public int solve(scala.collection.immutable.Seq<y.Item>, int);
[ ... ]
 97: aload_1
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload         4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```



```
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload          4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```



```
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload          4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```



```
 98: invokeinterface #177,  1 // Seq.iterator
103: iconst_1
104: invokeinterface #183,  2 // Iterator.drop
109: aload_0
110: aload_3
111: aload         4
113: invokedynamic #195,  0   // #2
118: invokeinterface #199,  2 // Iterator.foreach
[ ... ]
```

```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

...

```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

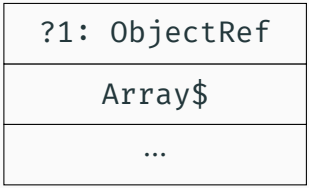
...

```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

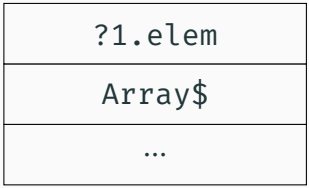
```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```



```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```



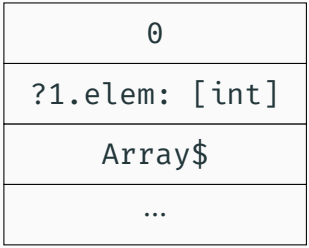
```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

?1.elem: [int]
Array$
…

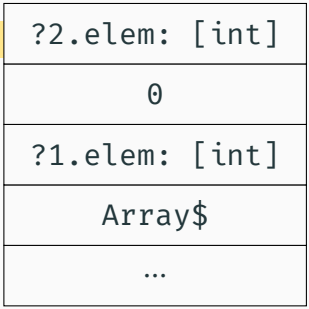
```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```



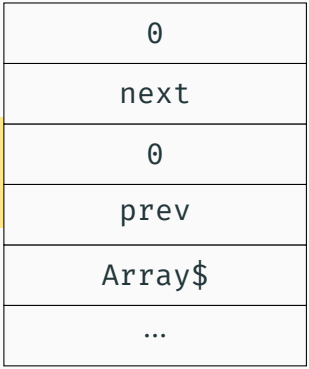
```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```



```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

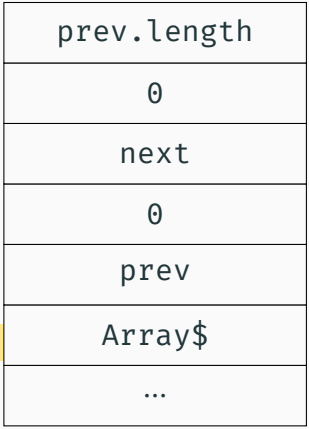


```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```



```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

```
private final void solve$$anonfun$1
 0: getstatic     #115 // Array$.MODULE$
 3: aload_1
 4: getfield      #166 // ObjectRef.elem
 7: checkcast     #146 // class "[I"
10: iconst_0
11: aload_2
12: getfield      #166 // ObjectRef.elem
15: checkcast     #146 // class "[I"
18: iconst_0
19: aload_1
20: getfield      #166 // ObjectRef.elem
23: checkcast     #146 // class "[I"
26: arraylength
27: invokevirtual #224 // Array$.copy
[ ... ]
```

```
Array.copy(prev, 0, next, 0, prev.length)
var w = item.weight
while w < next.length do
  val v = prev(w - item.weight) + item.value
  if next(w) < v then next(w) = v
  w += 1
val tmp = next
next = prev
prev = tmp
```

...

Wir probieren da mal was aus...

```
static class Box<T> {
    public T elem;
    public Box(T elem) {
        this.elem = elem;
    }
}
```

Wir probieren da mal was aus...

```
static class Box<T> {
    public T elem;
    public Box(T elem) {
        this.elem = elem;
    }
}

var prev = new Box<>(new int[weight + 1]);
var next = new Box<>(new int[weight + 1]);
[ ... ] // initialization
var iter = items.iterator();
iter.next();
iter.forEachRemaining(item → {
  [ ... ] // inner loop
});
return prev.elem[weight];
```

Wir probieren da mal was aus...

```
static class Box<T> {
    public T elem;
    public Box(T elem) {
        this.elem = elem;
    }
}

var prev = new Box<>(new int[weight + 1]);
var next = new Box<>(new int[weight + 1]);
[ ... ] // initialization
var iter = items.iterator();
iter.next();
iter.forEachRemaining(item → {
  [ ... ] // inner loop
});
return prev.elem[weight];
```

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick					
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s

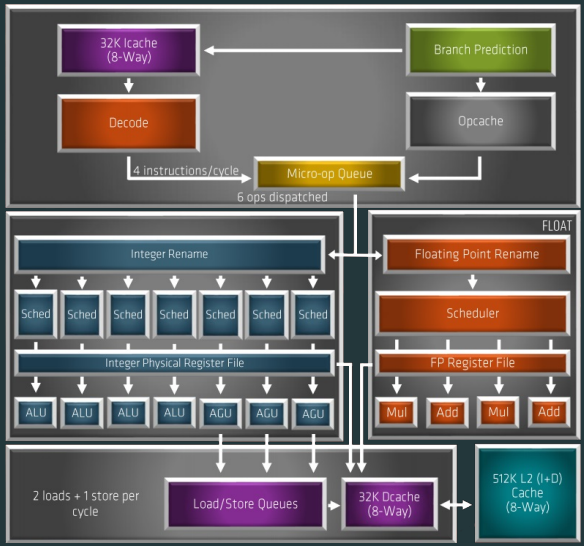
Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s

Zwei Fragen bleiben

Warum war Java langsamer als Scala?

Warum war Rust (so viel) langsamer als C++?


32K Icache
(8-Way)
Branch Prediction
Decode
Opcache
4 instructions/cycle
Micro-op Queue
6 ops dispatched
FLOAT
Integer Rename
Floating Point Rename
Sched
Sched
Sched
Sched
Sched
Sched
Sched
Scheduler
Integer Physical Register File
FP Register File
ALU
ALU
ALU
ALU
AGU
AGU
AGU
Mul
Add
Mul
Add
2 loads + 1 store per
cycle
Load/Store Queues
32K Dcache
(8-Way)
512K L2 (I+D)
Cache
(8-Way)

Mike Clark

Performance Counter

```
#> perf stat ./a.out < ../sample-n50k.in
```

Performance Counter

```
#> perf stat ./a.out < ../sample-n50k.in


      5.524,94 msec task-clock                #    1,000 CPUs utilized
            18      context-switches          #    3,258 /sec
             4      cpu-migrations            #    0,724 /sec
           715      page-faults               #  129,413 /sec
24.777.555.856      cycles                    #    4,485 GHz
   127.093.786      stalled-cycles-frontend   #    0,51% idle
20.309.710.125      stalled-cycles-backend    #   81,97% idle
18.390.846.417      instructions              #    0,74  insn per cycle
 3.560.017.361      branches                  #  644,354 M/sec
       796.256      branch-misses             #    0,02% of all branches

   5,525497441 seconds time elapsed
```

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}
Rust	1.87×10^{10}	0.58	0.08 %	87.47 %	1.98×10^{9}	5.47×10^{5}

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}
Rust	1.87×10^{10}	0.58	0.08 %	87.47 %	1.98×10^{9}	5.47×10^{5}
Scala	1.03×10^{11}	3.58	0.64 %	34.98 %	1.84×10^{10}	2.98×10^{7}

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}
Rust	1.87×10^{10}	0.58	0.08 %	87.47 %	1.98×10^{9}	5.47×10^{5}
Scala	1.03×10^{11}	3.58	0.64 %	34.98 %	1.84×10^{10}	2.98×10^{7}
Rust[1]	1.63×10^{11}	4.20	0.02 %	67.35 %	3.75×10^{10}	5.16×10^{5}

[1] ohne den `memcpy`-Trick

Acht auf einen Streich

Problem: Wir verwenden zu viel Zeit darauf, Daten zu laden und Befehle zu dekodieren.

Problem: Wir verwenden zu viel Zeit darauf, Daten zu laden und Befehle zu dekodieren.

Lösung: Wir laden viele Daten auf einmal und rechnen auf Blöcken statt auf einzelnen Einträgen.

Spielregeln

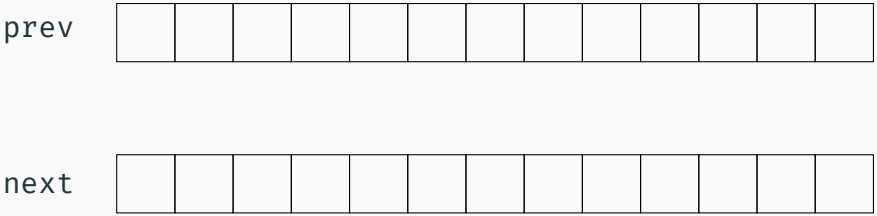
- 32-bit integers pro Tabelleneintrag

Spielregeln

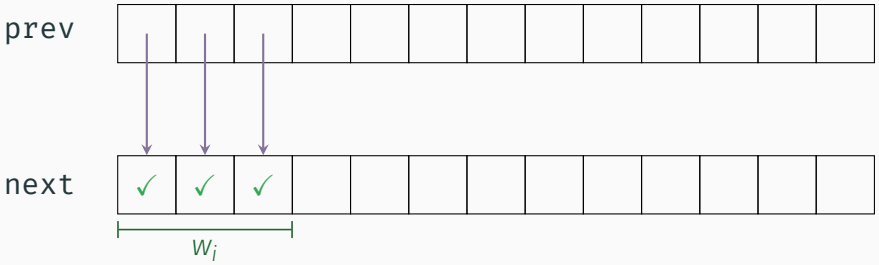
- 32-bit integers pro Tabelleneintrag
- `-march=native` oder äquivalent

Algorithmus für einen 4× Vektorprozessor

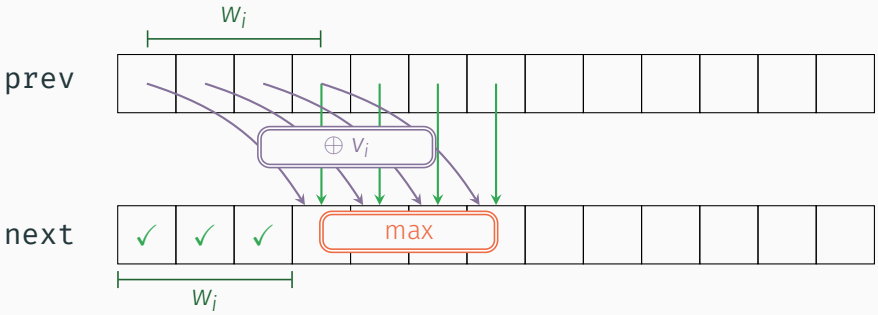
Initialisierung wie bei der klassichen Variante



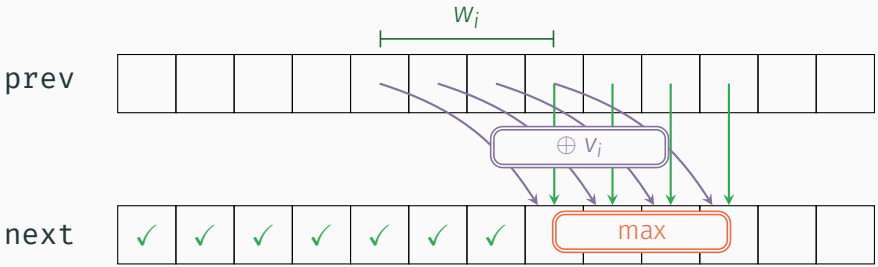
Algorithmus für einen 4× Vektorprozessor



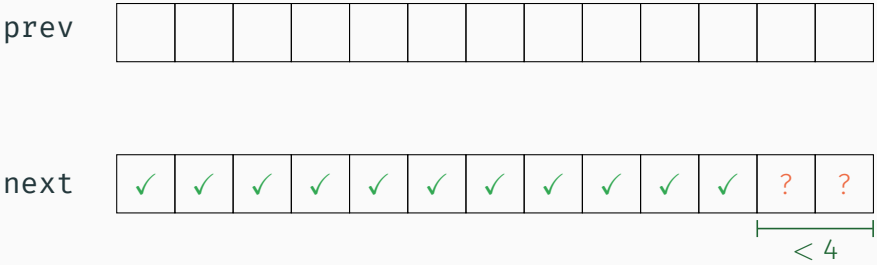
Algorithmus für einen 4× Vektorprozessor



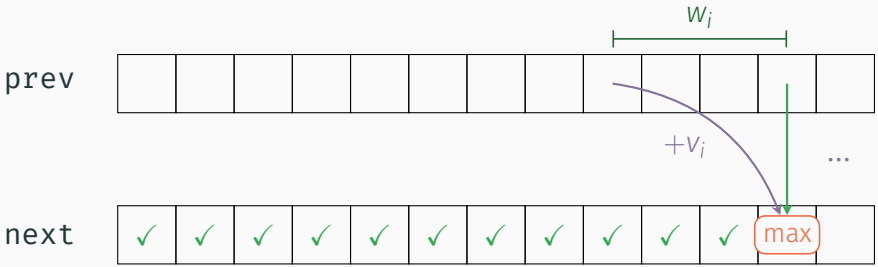
Algorithmus für einen 4× Vektorprozessor



Algorithmus für einen 4× Vektorprozessor

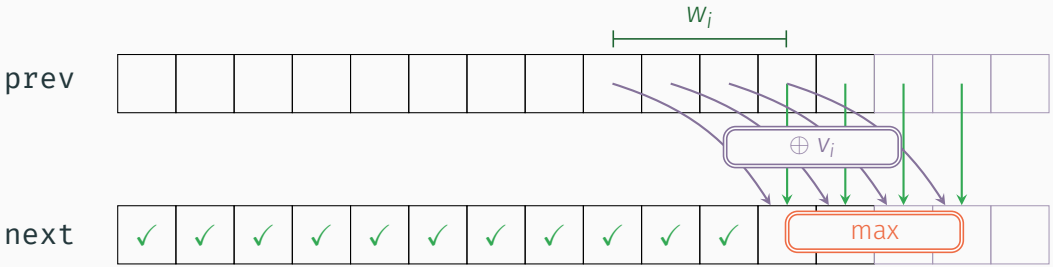


Algorithmus für einen 4× Vektorprozessor

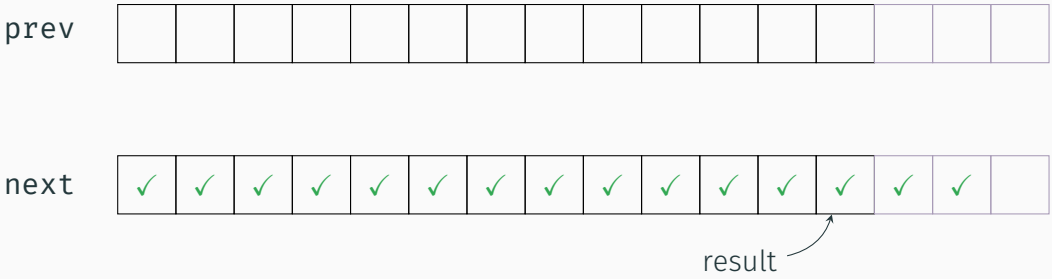


Algorithmus für einen 4× Vektorprozessor

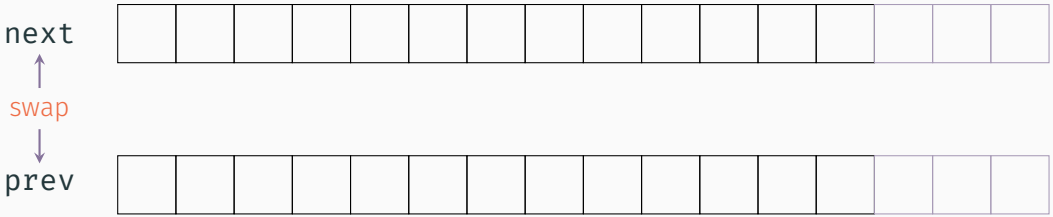
Algorithmus für einen 4× Vektorprozessor



Algorithmus für einen 4× Vektorprozessor



Algorithmus für einen 4× Vektorprozessor



Umsetzung

Umsetzung

Umsetzung


```
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i
```

Umsetzung


```
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i

pub unsafe fn _mm256_add_epi32(a: __m256i, b: __m256i) → __m256i
```

Umsetzung


```
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i
```

```
pub unsafe fn _mm256_add_epi32(a: __m256i, b: __m256i) → __m256i
```

max

Umsetzung


```rust
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i
```

```rust
pub unsafe fn _mm256_add_epi32(a: __m256i, b: __m256i) → __m256i
```

max

```rust
pub unsafe fn _mm256_max_epu32(a: __m256i, b: __m256i) → __m256i
```

Umsetzung


```rust
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i
```

```rust
pub unsafe fn _mm256_add_epi32(a: __m256i, b: __m256i) → __m256i
```

max

```rust
pub unsafe fn _mm256_max_epu32(a: __m256i, b: __m256i) → __m256i
```

```rust
pub unsafe fn _mm256_loadu_si256(mem_addr: *const __m256i) → __m256i
```

Umsetzung


```
pub unsafe fn _mm256_set1_epi32(a: i32) → __m256i
pub unsafe fn _mm256_add_epi32(a: __m256i, b: __m256i) → __m256i
```

max

```
pub unsafe fn _mm256_max_epu32(a: __m256i, b: __m256i) → __m256i

pub unsafe fn _mm256_loadu_si256(mem_addr: *const __m256i) → __m256i
pub unsafe fn _mm256_storeu_si256(mem_addr: *mut __m256i, a: __m256i)
```

Alignment?

AVX Befehle, die mit dem Arbeitsspeicher interagieren, haben eine `aligned` und eine `unaligned` Variante. Erstere ist meist deutlich schneller.

Alignment?

AVX Befehle, die mit dem Arbeitsspeicher interagieren, haben eine `aligned` und eine `unaligned` Variante. Erstere ist meist deutlich schneller.

```
aligned = pointer % sizeof(vec_t) == 0;
```

Alignment?

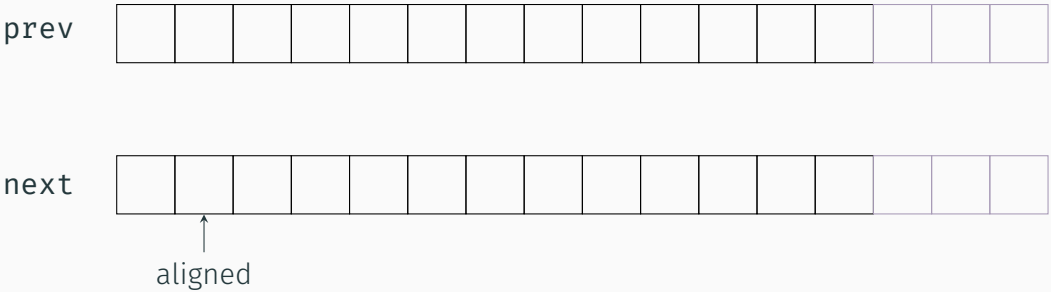
AVX Befehle, die mit dem Arbeitsspeicher interagieren, haben eine `aligned` und eine `unaligned` Variante. Erstere ist meist deutlich schneller.

```
aligned = pointer % sizeof(vec_t) == 0;
```

In modernen Sprachen kann man das auch ohne Pointer-Arithmetik rausfinden.

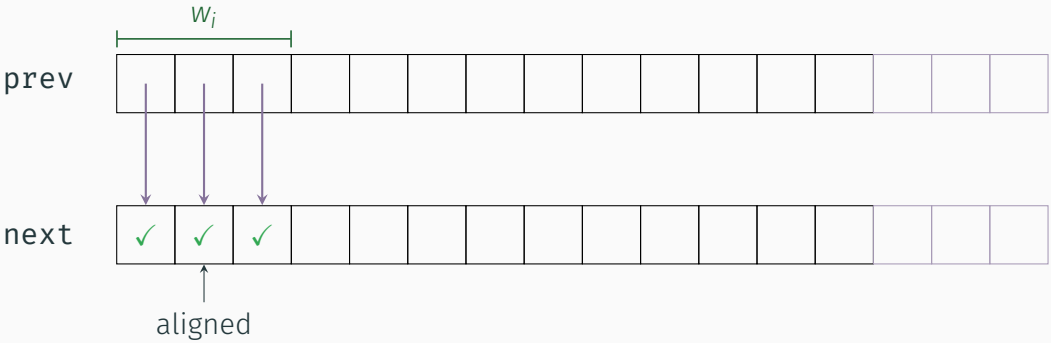
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



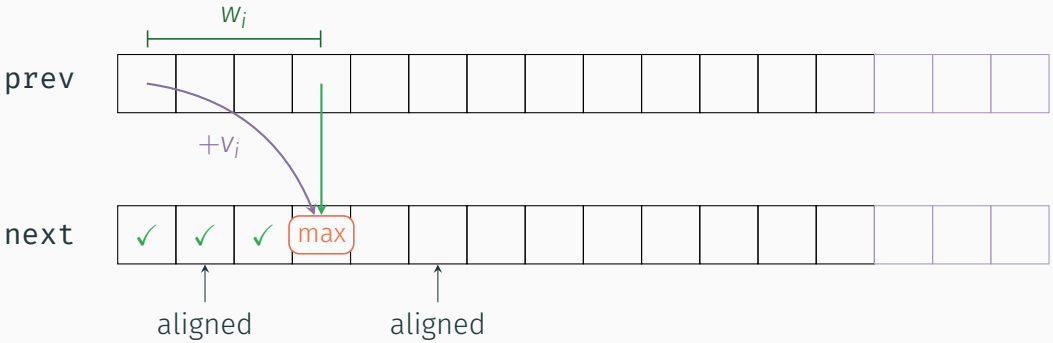
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



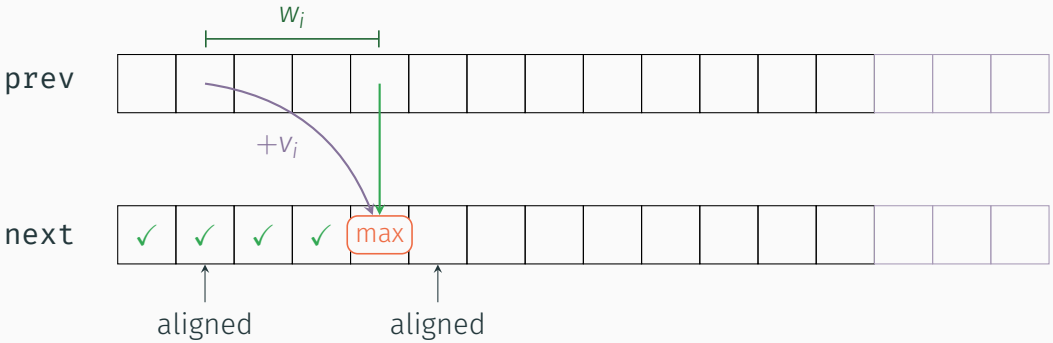
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



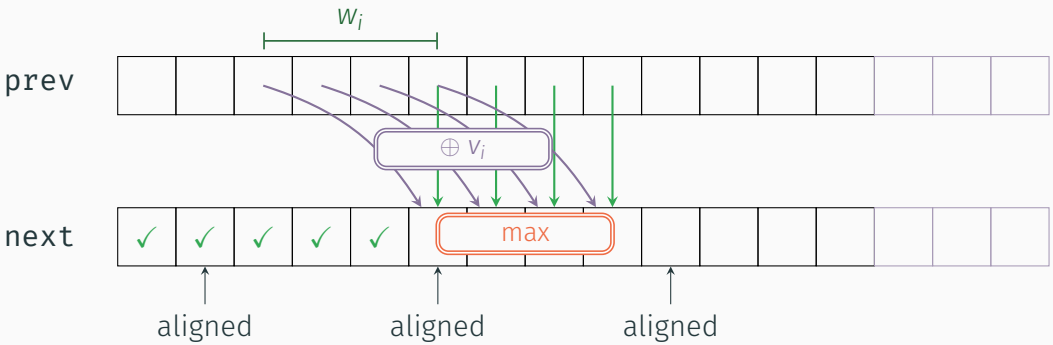
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



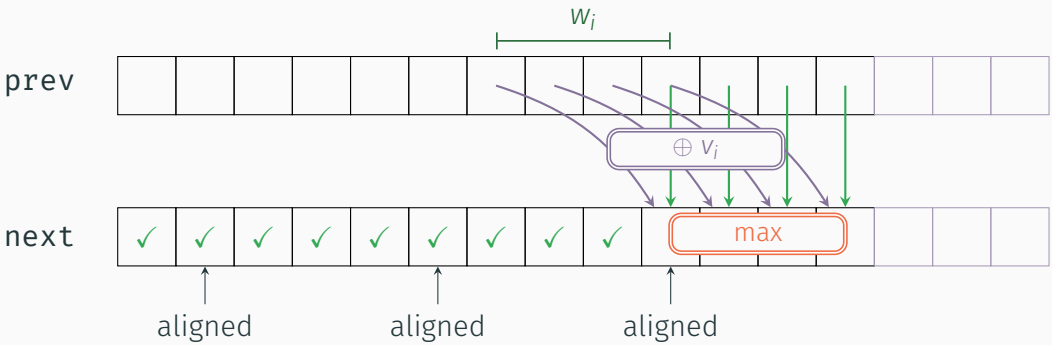
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



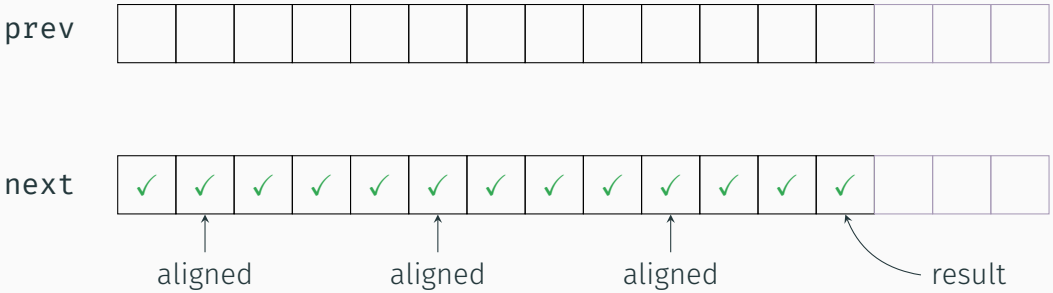
Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



Algorithmus für einen 4× Vektorprozessor

Mit korrektem Alignment (zumindest für die Schreibzugriffe)



Initialisierung mit Padding:

```
let (mut next, mut prev) = (vec![0; weight + 8], vec![0; weight + 8]);
let head = &items[0];
if head.weight ⩽ weight {
    prev[head.weight..].fill(head.value);
}
```

Initialisierung mit Padding:

```
let (mut next, mut prev) = (vec![0; weight + 8], vec![0; weight + 8]);
let head = &items[0];
if head.weight ⩽ weight {
    prev[head.weight..].fill(head.value);
}
```

Für die Einträge zwischen w_i und dem ersten Eintrag mit korrektem Alignment brauchen wir die innere Schleife einmal in nicht-vektorisiert:

```
fn fallback(mut w: usize, stop: usize, item: &Item,
            prev: &[u32], next: &mut [u32]) {
    while w < stop {
        let take = prev[w - item.weight] + item.value;
        next[w] = take.max(prev[w]);
        w += 1;
    }
}
```

Äußere Schleife: Ein Durchlauf pro Item, neues `next` ausrechnen, `prev` und `next` tauschen

Äußere Schleife: Ein Durchlauf pro Item, neues `next` ausrechnen, `prev` und `next` tauschen

```
for item in items.iter().skip(1) {
    let w = item.weight;
    if w > weight {
        continue;
    }
    next[..w].copy_from_slice(&prev[..w]);
```

Äußere Schleife: Ein Durchlauf pro Item, neues `next` ausrechnen, `prev` und `next` tauschen

```
for item in items.iter().skip(1) {
    let w = item.weight;
    if w > weight {
        continue;
    }
    next[..w].copy_from_slice(&prev[..w]);
    unsafe {
        let mut prev_ptr: *const u32 = prev.as_ptr();
        let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);

        let offset = next_ptr.align_offset(align_of::<__m256i>());
```

Äußere Schleife: Ein Durchlauf pro Item, neues `next` ausrechnen, `prev` und `next` tauschen

```
for item in items.iter().skip(1) {
    let w = item.weight;
    if w > weight {
        continue;
    }
    next[..w].copy_from_slice(&prev[..w]);
    unsafe {
        let mut prev_ptr: *const u32 = prev.as_ptr();
        let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);

        let offset = next_ptr.align_offset(align_of::<__m256i>());
        // ...
    }

    std::mem::swap(&mut next, &mut prev);
}
prev[weight]
```

```rust
let mut prev_ptr: *const u32 = prev.as_ptr();
let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);
let offset = next_ptr.align_offset(align_of::<__m256i>());
```

```rust
let mut prev_ptr: *const u32 = prev.as_ptr();
let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);
let offset = next_ptr.align_offset(align_of::<__m256i>());
if w + offset > next.len() {
    fallback(w, next.len(), item, &prev, &mut next);
} else {
    fallback(w, w + offset + 1, item, &prev, &mut next);
    prev_ptr = prev_ptr.add(offset);
    next_ptr = next_ptr.add(offset);
```

```
let mut prev_ptr: *const u32 = prev.as_ptr();
let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);
let offset = next_ptr.align_offset(align_of::<__m256i>());
if w + offset > next.len() {
    fallback(w, next.len(), item, &prev, &mut next);
} else {
    fallback(w, w + offset + 1, item, &prev, &mut next);
    prev_ptr = prev_ptr.add(offset);
    next_ptr = next_ptr.add(offset);
    for _ in 0..(weight - w - offset + 8) / 8 {
        let dont_take = _mm256_loadu_si256(prev_ptr.add(w).cast());
        let take = _mm256_add_epi32(
            _mm256_loadu_si256(prev_ptr.cast()),
            _mm256_set1_epi32(item.value as i32),
        );
        _mm256_store_si256(next_ptr.cast(),
                           _mm256_max_epu32(dont_take, take));
        prev_ptr = prev_ptr.add(8);
        next_ptr = next_ptr.add(8);
    }
}
```

```rust
let mut prev_ptr: *const u32 = prev.as_ptr();
let mut next_ptr: *mut u32 = next.as_mut_ptr().add(w);
let offset = next_ptr.align_offset(align_of::<__m256i>());
if w + offset > next.len() {
    fallback(w, next.len(), item, &prev, &mut next);
} else {
    fallback(w, w + offset + 1, item, &prev, &mut next);
    prev_ptr = prev_ptr.add(offset);
    next_ptr = next_ptr.add(offset);
    for _ in 0..(weight - w - offset + 8) / 8 {
        let dont_take = _mm256_loadu_si256(prev_ptr.add(w).cast());
        let take = _mm256_add_epi32(
            _mm256_loadu_si256(prev_ptr.cast()),
            _mm256_set1_epi32(item.value as i32),
        );
        _mm256_store_si256(next_ptr.cast(),
                           _mm256_max_epu32(dont_take, take));
        prev_ptr = prev_ptr.add(8);
        next_ptr = next_ptr.add(8);
    }
}
```

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned					

Laufzeiten

Instanzgröße *n*	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}
Rust	1.87×10^{10}	0.58	0.08 %	87.47 %	1.98×10^{9}	5.47×10^{5}
Scala	1.03×10^{11}	3.58	0.64 %	34.98 %	1.84×10^{10}	2.98×10^{7}
Rust[1]	1.63×10^{11}	4.20	0.02 %	67.35 %	3.75×10^{10}	5.16×10^{5}
Rust[2]	1.25×10^{10}	2.60	0.04 %	61.51 %	1.57×10^{9}	2.48×10^{5}

[1] ohne den `memcpy`-Trick
[2] mit AVX

safe and fast – pick two!

```
fn solve(items: &[Item], weight: usize) → u32 {
    // initialization ...
    for item in items.iter().skip(1) {
        if item.weight < next.len() {
            inner(&mut prev, &mut next, item)
        }
    }
    prev[weight]
}
```

safe and fast – pick two!

```rust
fn solve(items: &[Item], weight: usize) -> u32 {
    // initialization ...
    for item in items.iter().skip(1) {
        if item.weight < next.len() {
            inner(&mut prev, &mut next, item)
        }
    }
    prev[weight]
}
fn inner<'a>(prev: &mut &'a mut [u32], next: &mut &'a mut [u32], item: &Item) {
    let mut w = item.weight;
    next[..w].copy_from_slice(&prev[..w]);
    while w < next.len() {
        let take = prev[w - item.weight] + item.value;
        next[w] = take.max(prev[w]);
        w += 1;
    }
    std::mem::swap(next, prev);
}
```

safe and fast – pick two!

```
$> RUSTFLAGS="-C target-cpu=native" cargo build --release
```

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

safe and fast – pick two!

```rust
fn solve(items: &[Item], weight: usize) -> u32 {
    // initialization ...
    for item in items.iter().skip(1) {
        if item.weight < next.len() {
            inner(&mut prev, &mut next, item)
        }
    }
    prev[weight]
}
fn inner<'a>(prev: &mut &'a mut [u32], next: &mut &'a mut [u32], item: &Item) {
    let mut w = item.weight;
    next[..w].copy_from_slice(&prev[..w]);
    while w < next.len() {
        let take = prev[w - item.weight] + item.value;
        next[w] = take.max(prev[w]);
        w += 1;
    }
    std::mem::swap(next, prev);
}
```

	instructions	ipc	f/e idle	b/e idle	branches	misses
C++	1.84×10^{10}	0.74	0.51 %	81.97 %	3.56×10^{9}	7.96×10^{5}
Rust	1.87×10^{10}	0.58	0.08 %	87.47 %	1.98×10^{9}	5.47×10^{5}
Scala	1.03×10^{11}	3.58	0.64 %	34.98 %	1.84×10^{10}	2.98×10^{7}
Rust[1]	1.63×10^{11}	4.20	0.02 %	67.35 %	3.75×10^{10}	5.16×10^{5}
Rust[2]	1.25×10^{10}	2.60	0.04 %	61.51 %	1.57×10^{9}	2.48×10^{5}
Rust[3]	9.42×10^{9}	1.89	1.61 %	60.94 %	1.57×10^{9}	2.65×10^{5}

[1] ohne den `memcpy`-Trick
[2] mit AVX
[3] mit Auto-Vektorisierung

Bonus: Latenz vs. Durchsatz

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure			348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative				437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick				335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js				957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js	86 ms			957 ms	83.2 s
Python3				96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js	86 ms			957 ms	83.2 s
Python3	12 ms			96.8 s	💩
Pypy3				2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js	86 ms			957 ms	83.2 s
Python3	12 ms			96.8 s	💩
Pypy3	30 ms			2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++				212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js	86 ms			957 ms	83.2 s
Python3	12 ms			96.8 s	💩
Pypy3	30 ms			2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++	<1 ms			212 ms	14.2 s
Rust AVX aligned				43 ms	4.36 s
Rust autovectorized				44 ms	4.55 s

Laufzeiten

Instanzgröße n	30	100	1K	10K	100K
Scala naïve	1.15 s	💀			
Scala memoized	230 ms	267 ms	⚡SO		
Scala memoized TCO		273 ms	2.30 s	⚡OOM	
Scala pure	153 ms		348 ms	7.89 s	≈13 min
Scala imperative	142 ms			437 ms	24.0 s
Java				636 ms	58.8 s
Java w/ box trick	32 ms			335 ms	24.0 s
Node.js	86 ms			957 ms	83.2 s
Python3	12 ms			96.8 s	💩
Pypy3	30 ms			2.48 s	≈6 min
Rust (safe)				300 ms	28.1 s
Thomas' C++	<1 ms			212 ms	14.2 s
Rust AVX aligned	<1 ms			43 ms	4.36 s
Rust autovectorized	<1 ms			44 ms	4.55 s

Latenz vs. Durchsatz