technische universität dortmund

LS 2 / Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Stand der Dinge

Gierige Algorithmen

- Konstruiere Lösung Schritt für Schritt
- In jedem Schritt: Optimiere ein einfaches, lokales Kriterium

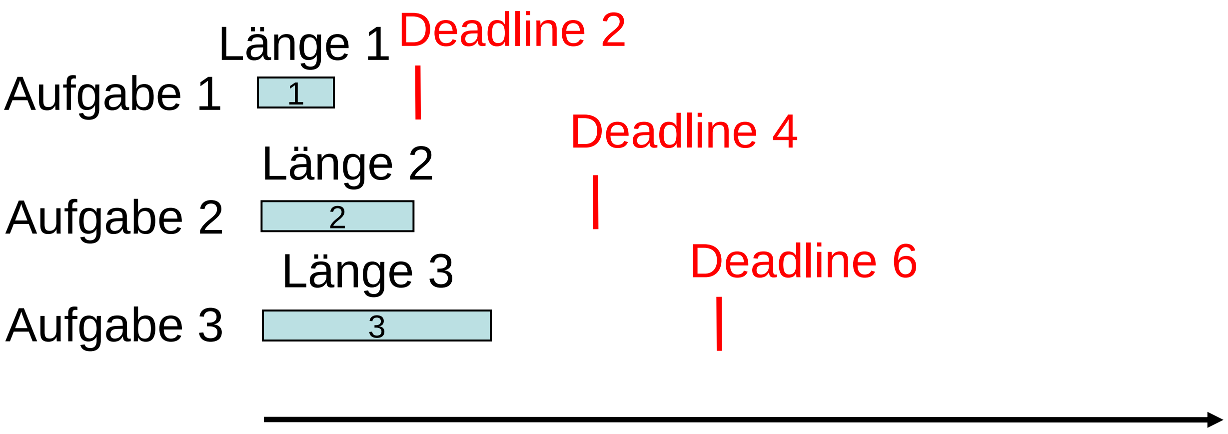
Beobachtung

- Man kann viele unterschiedliche gierige Algorithmen für ein Problem entwickeln
- Nicht jeder dieser Algorithmen löst das Problem korrekt

Gierige Algorithmen

Scheduling mit Deadlines

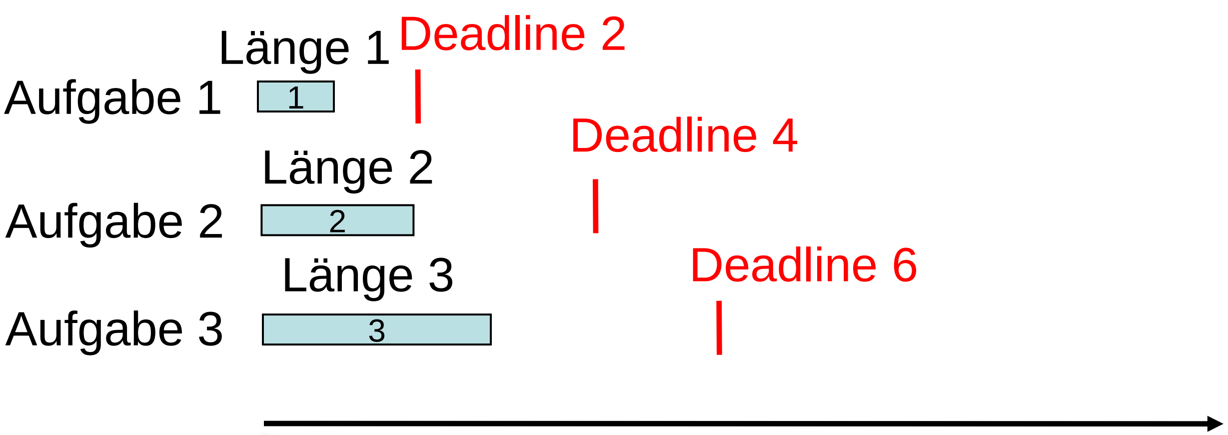
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

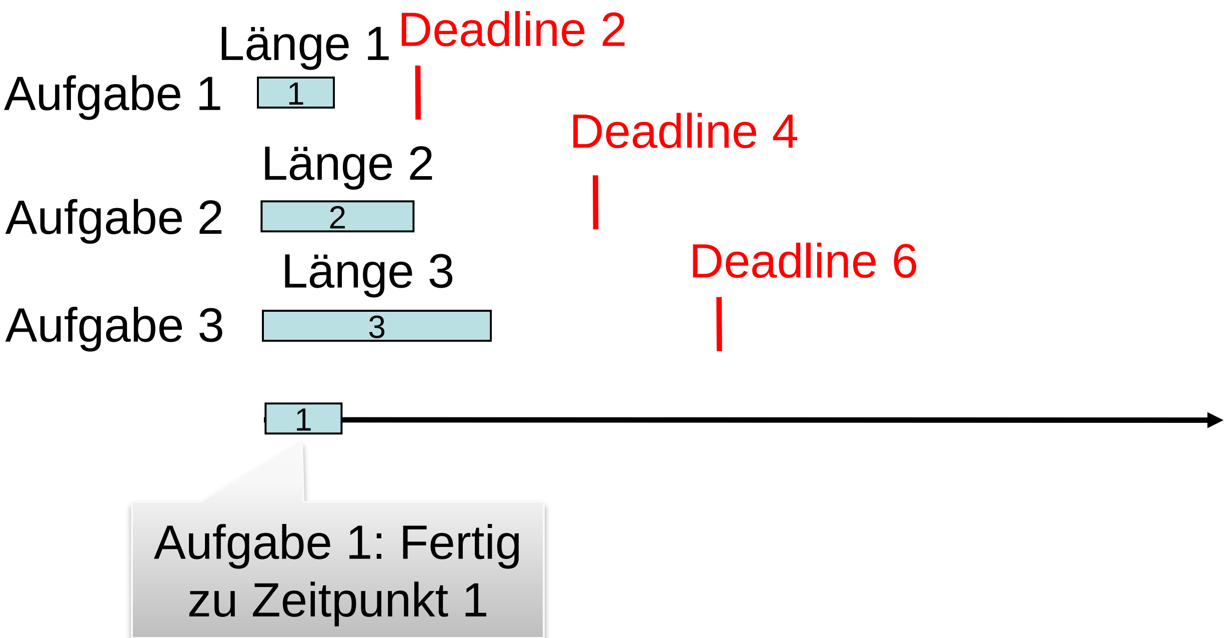


O.b.d.A. Resource steht ab Zeitpunkt 0 zur Verfügung

Gierige Algorithmen

Scheduling mit Deadlines

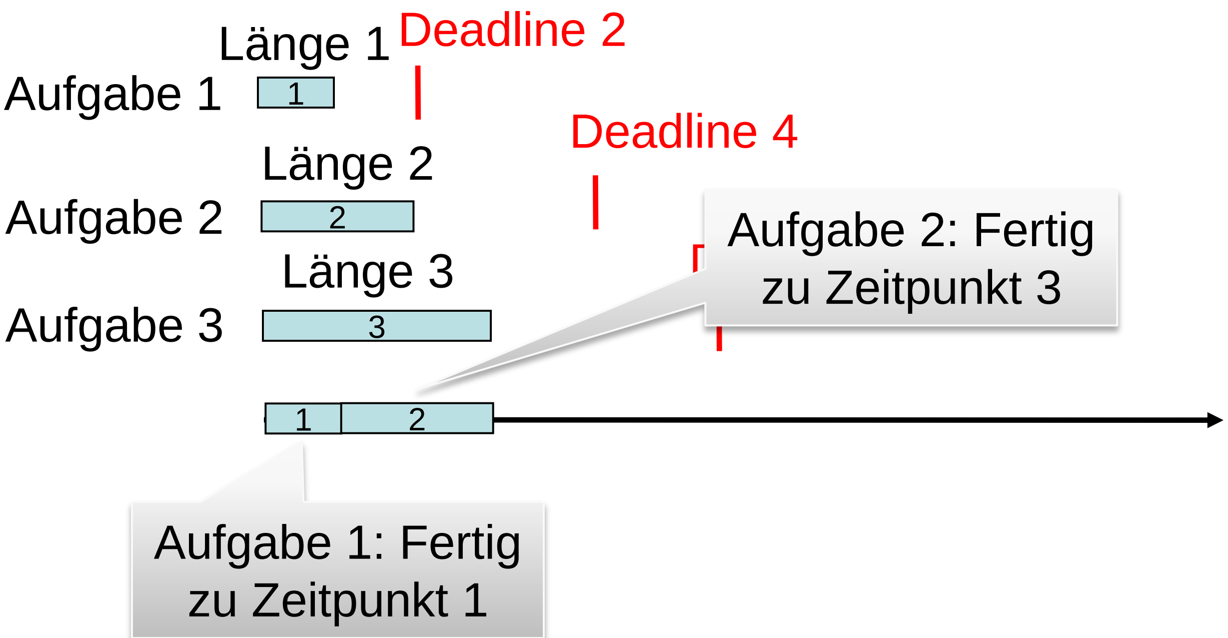
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

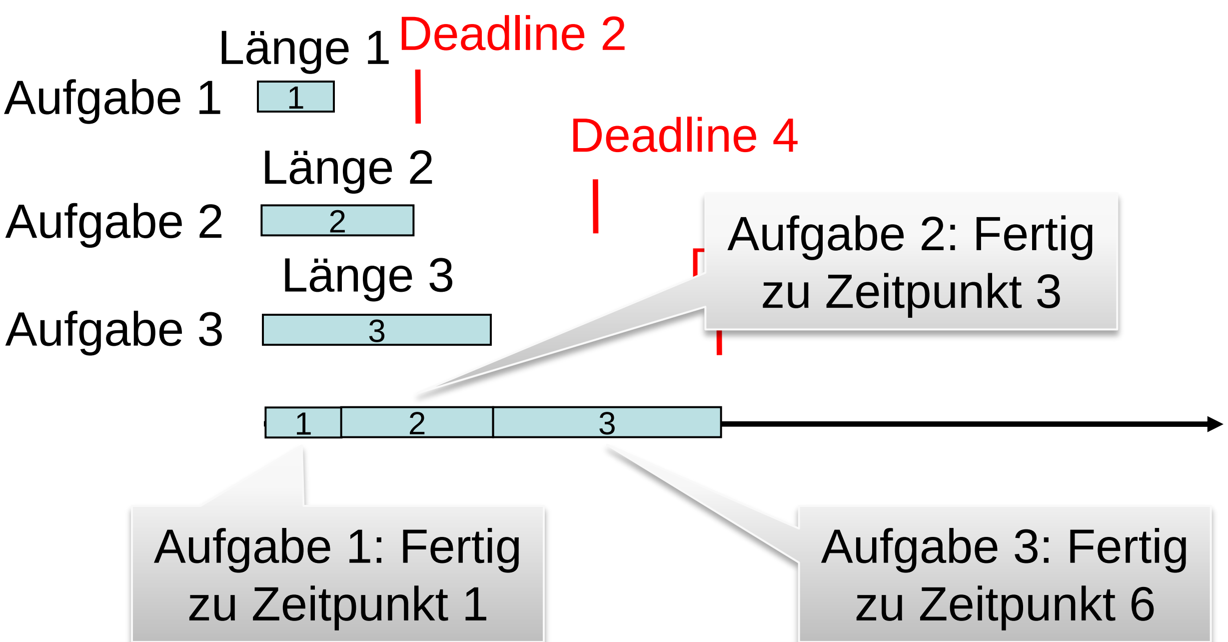
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

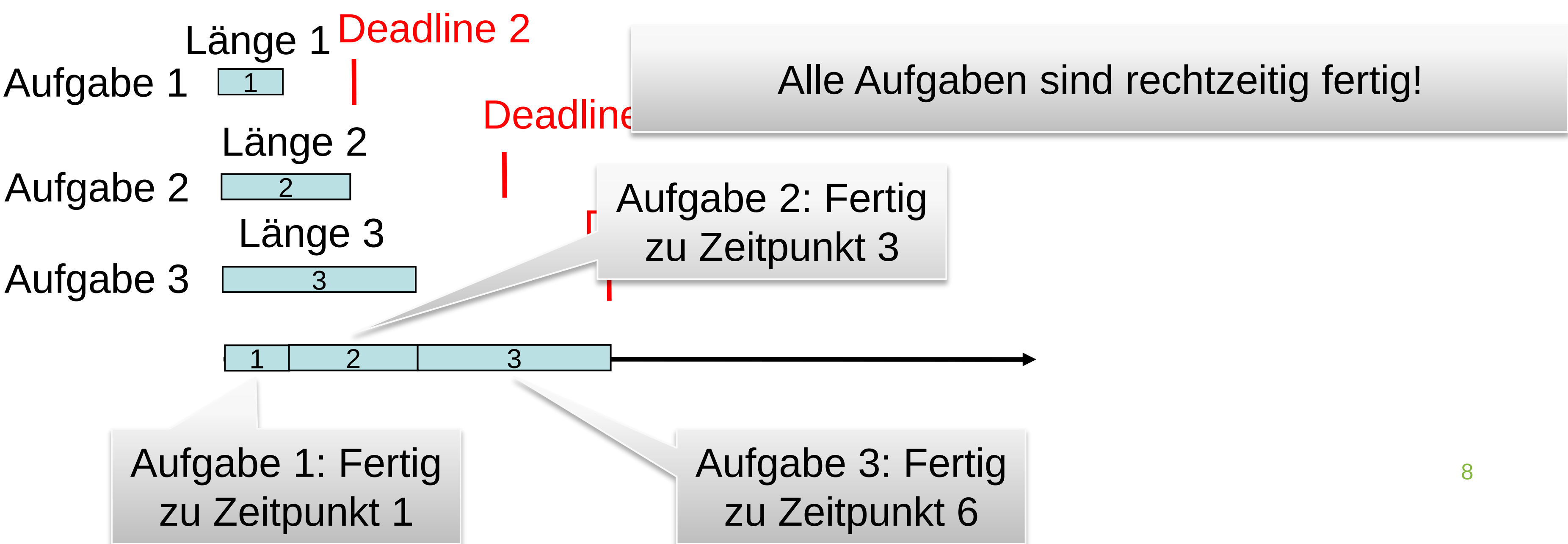
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

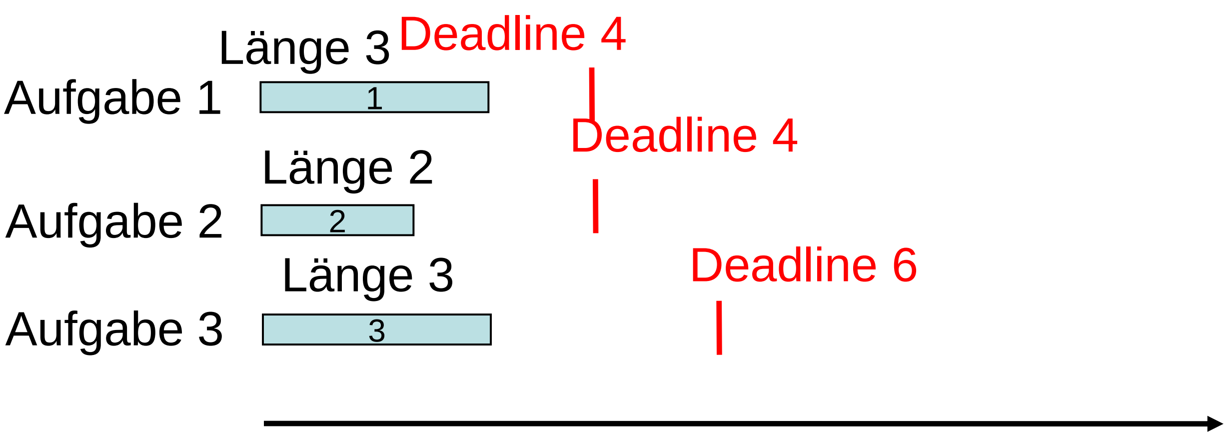
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d
  bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

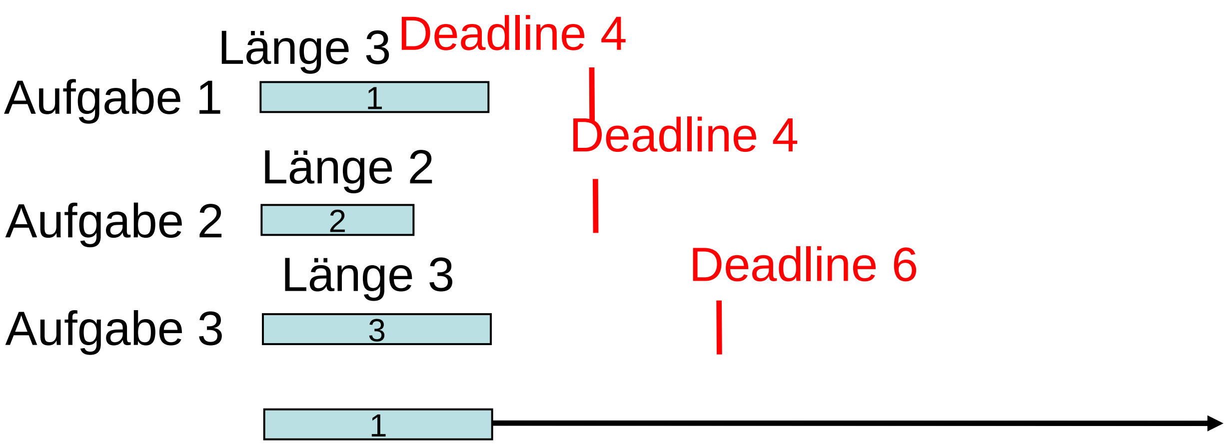
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

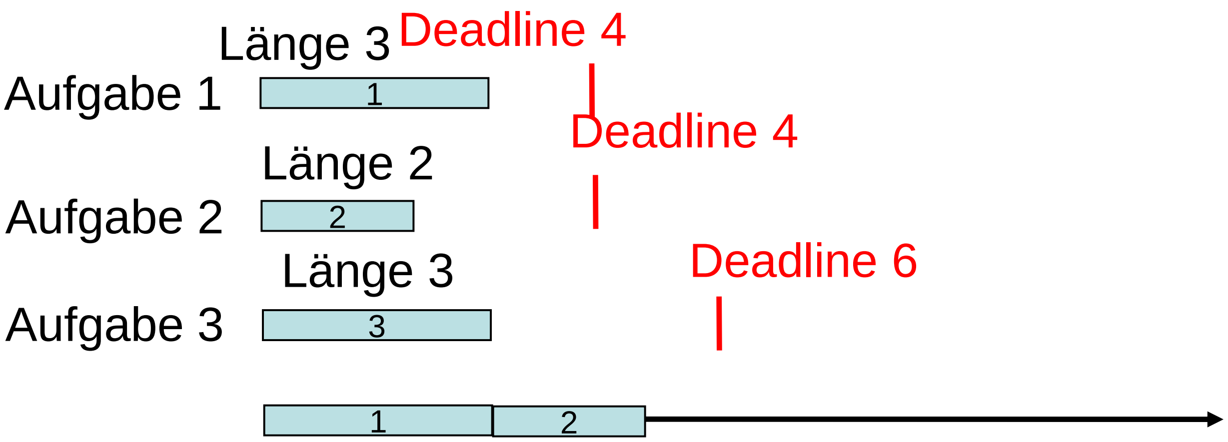
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Stand der Dinge

Scheduling mit Deadlines

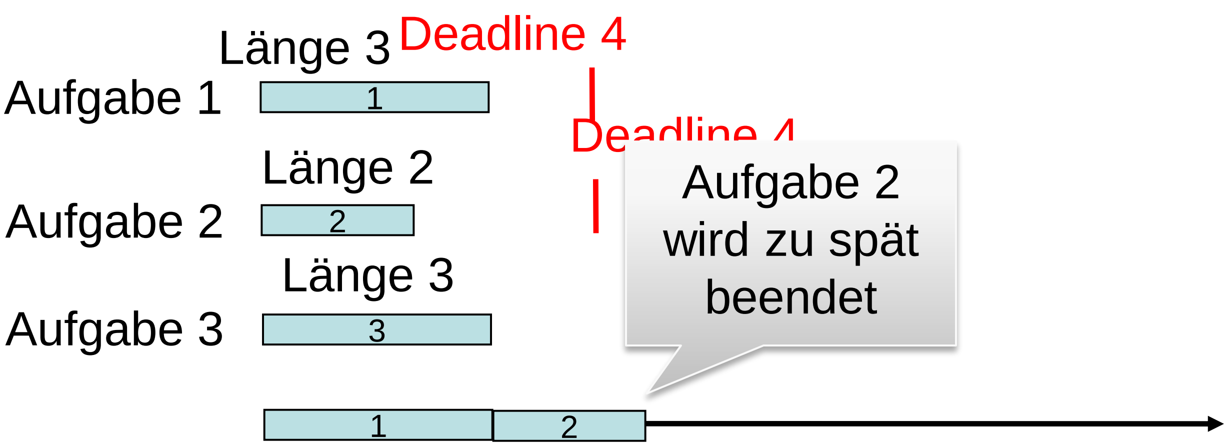
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Stand der Dinge

Scheduling mit Deadlines

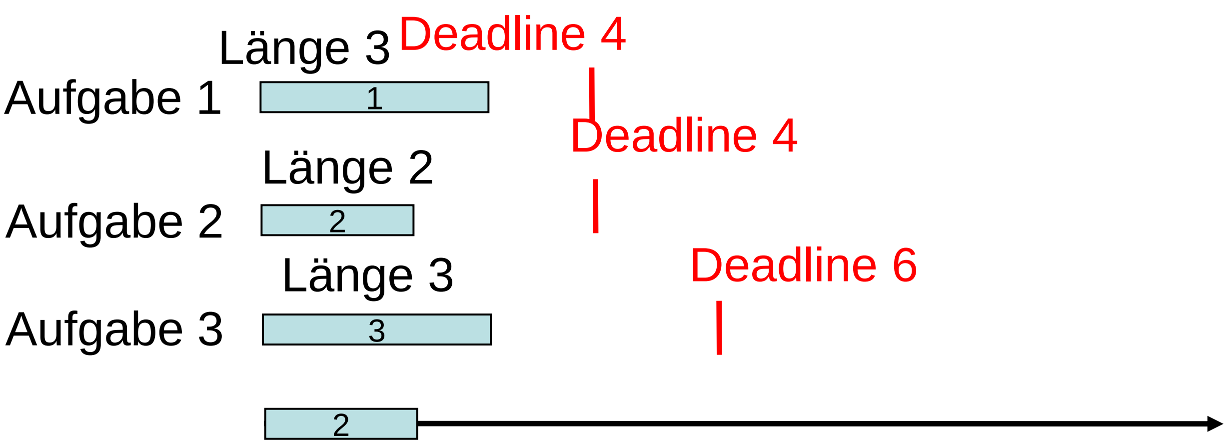
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

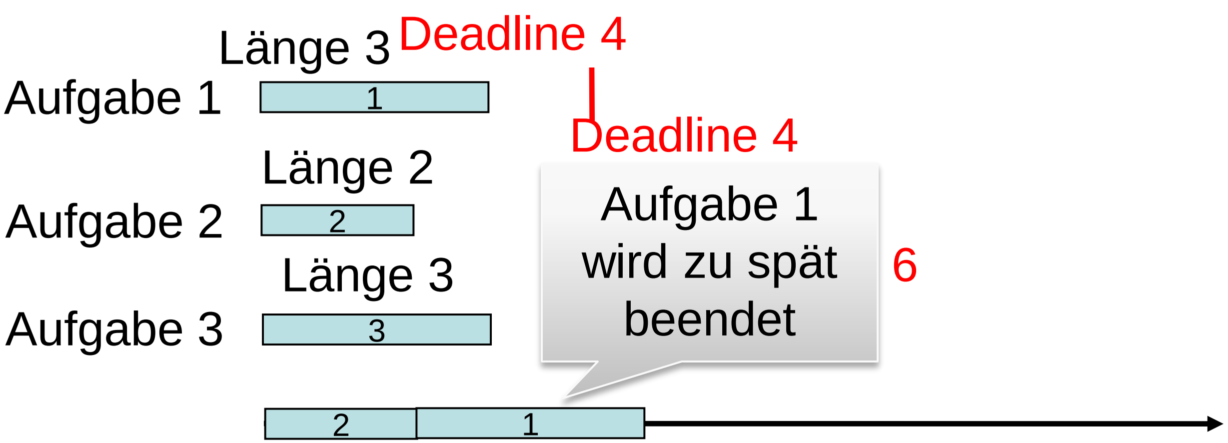
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

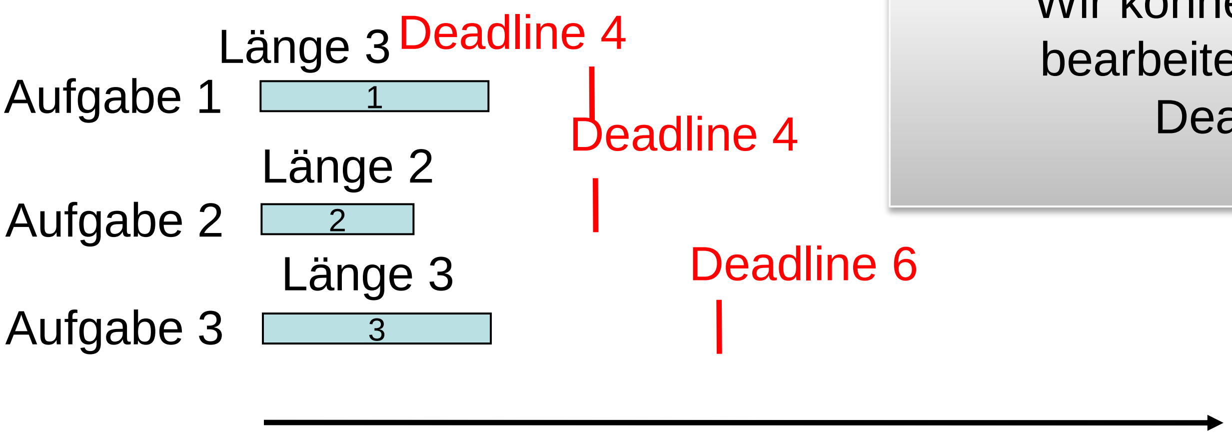
- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

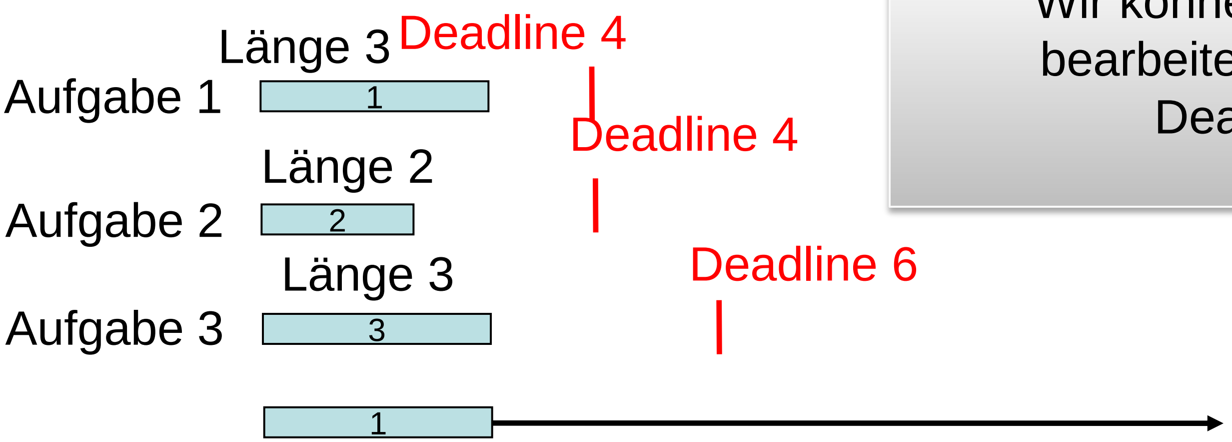


Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll



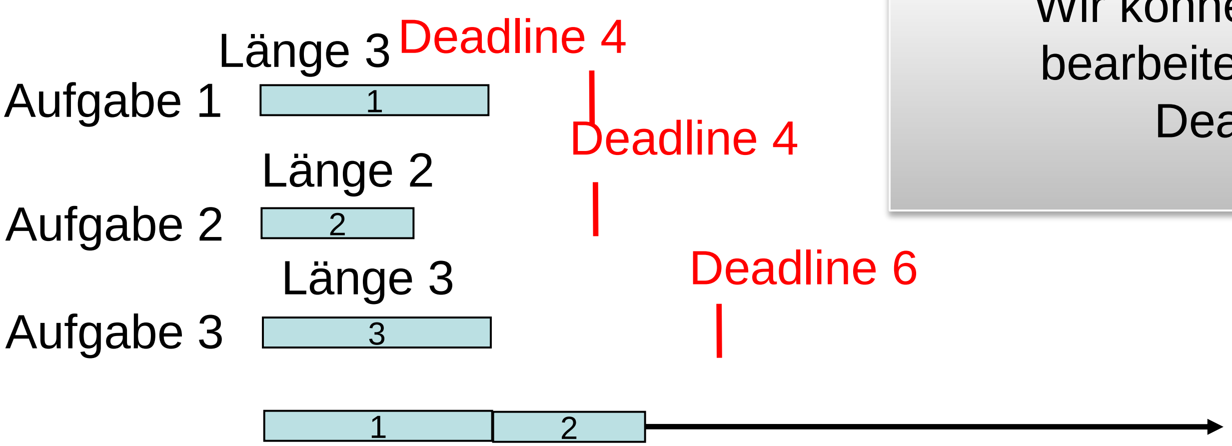
Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

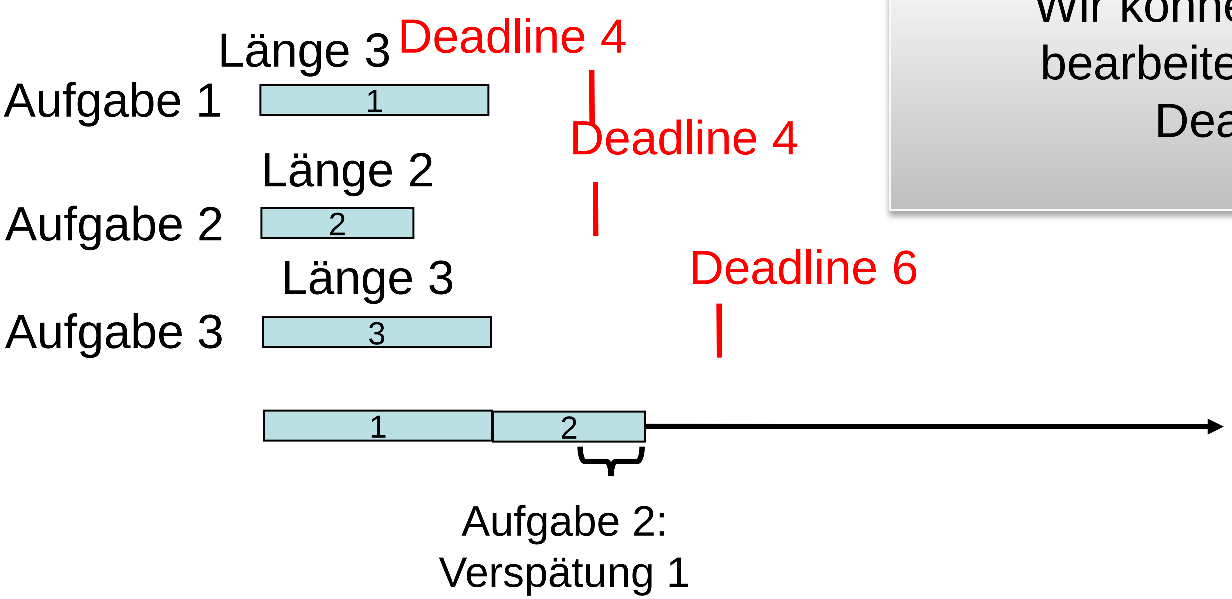


Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

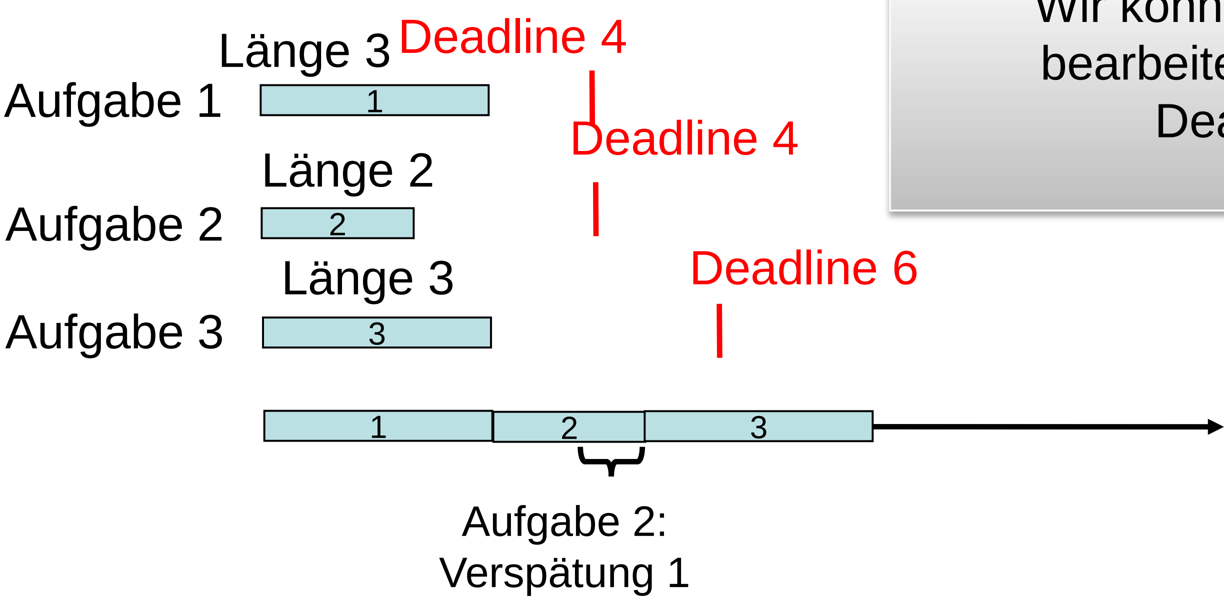


Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!

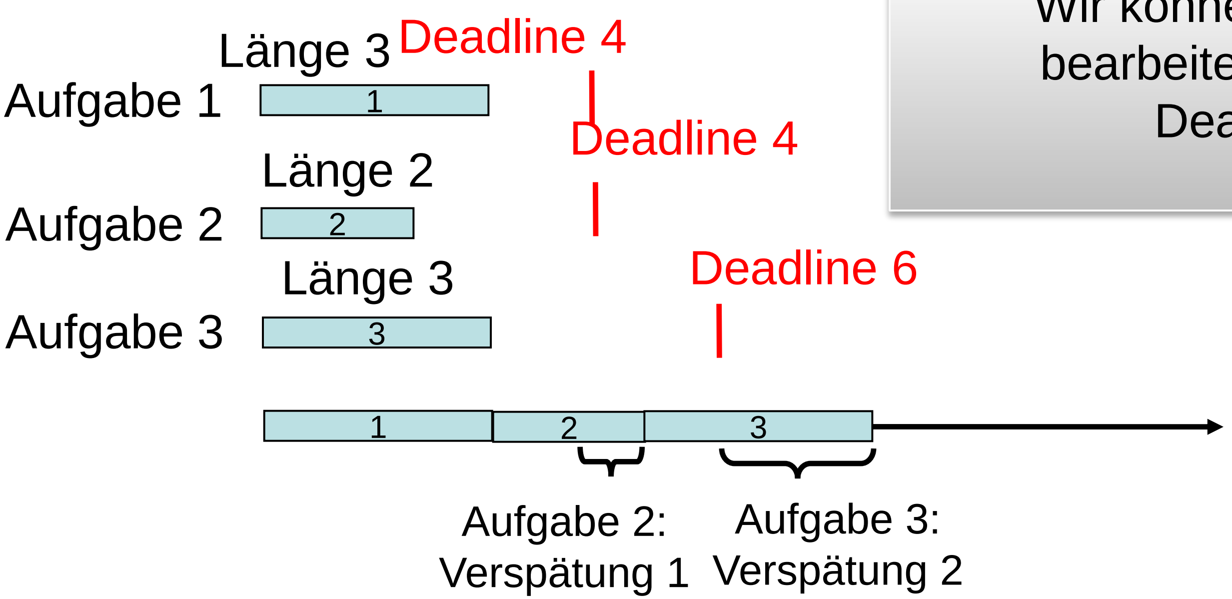


Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

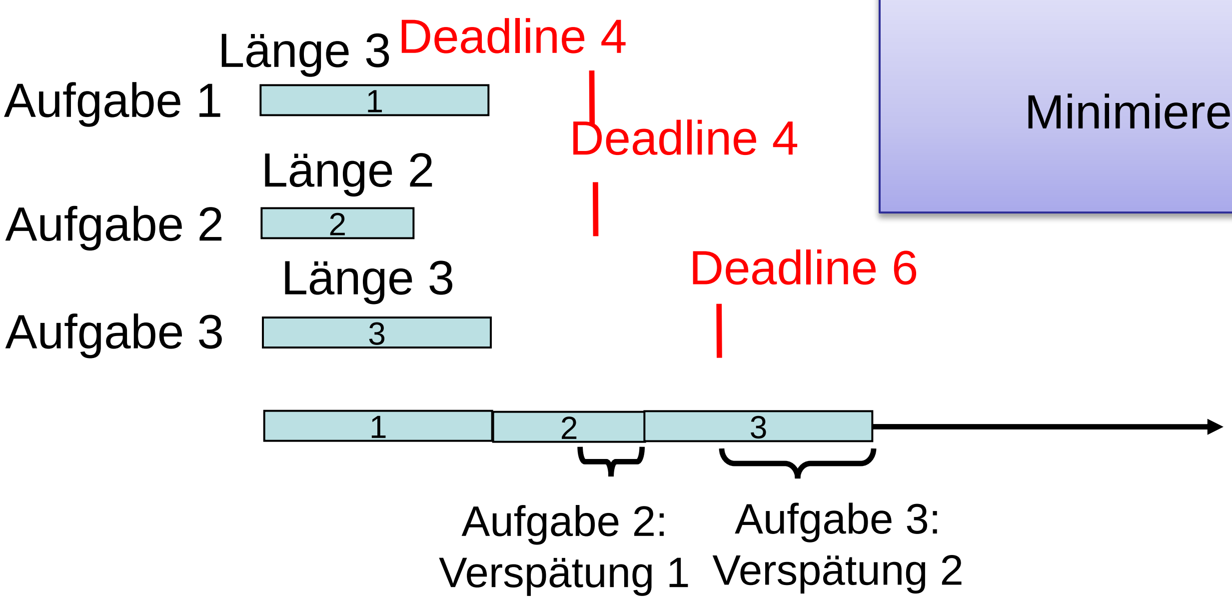
Wir können nicht alle Aufgaben bearbeiten und gleichzeitig die Deadlines einhalten!



Gierige Algorithmen

Scheduling mit Deadlines

- Resource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Aufgabe, die Zeit t benötigt und bis Zeitpunkt d bearbeitet sein soll

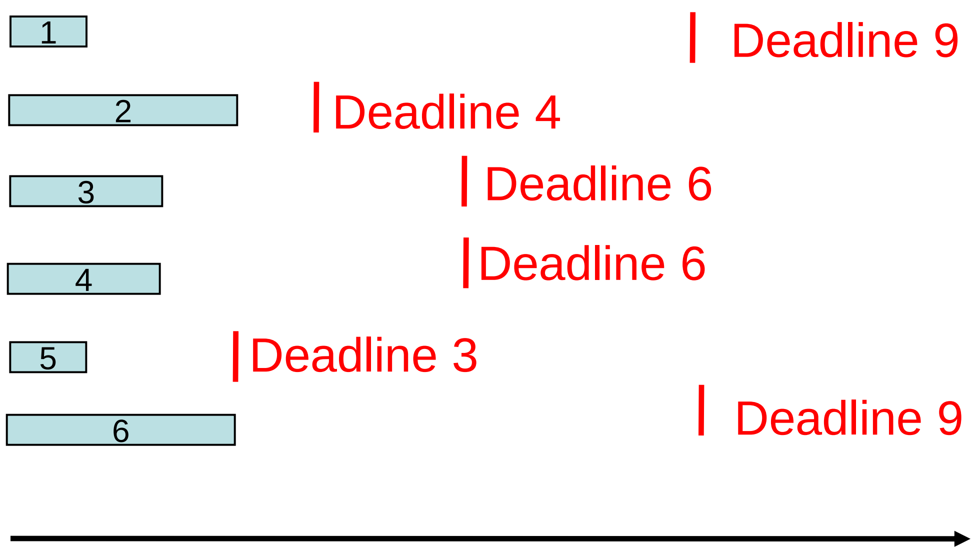


Ziel:
Minimiere maximale Verspätung

Gierige Algorithmen

Strategie 1

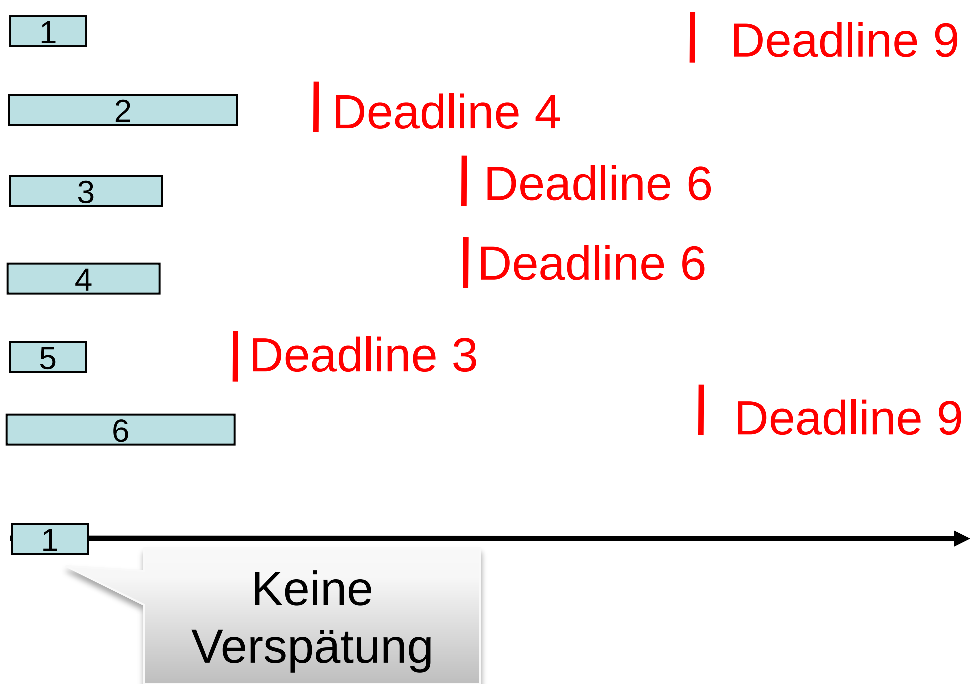
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

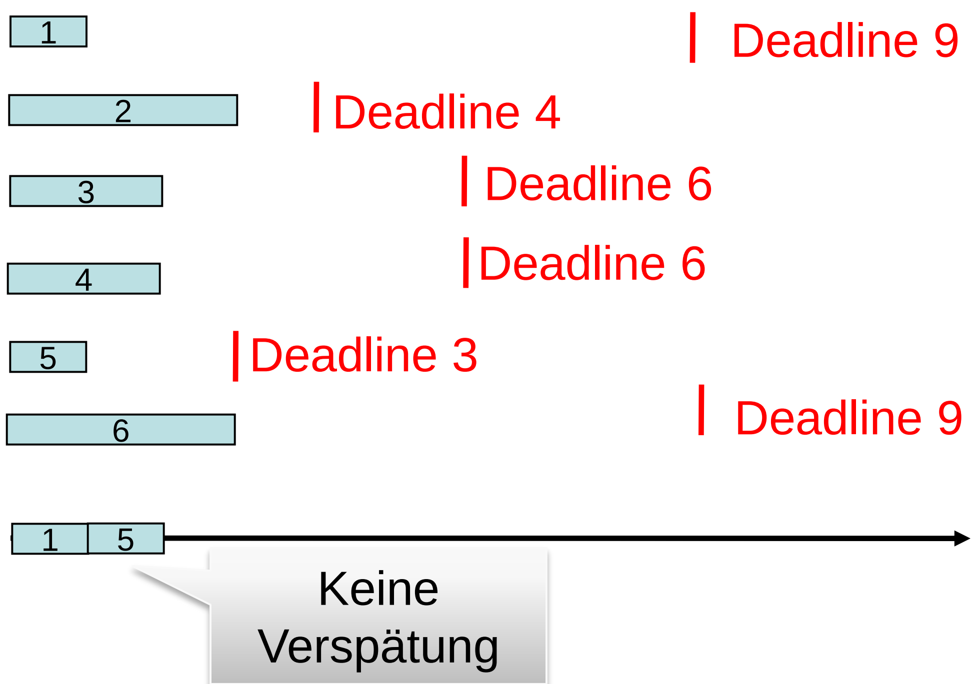
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

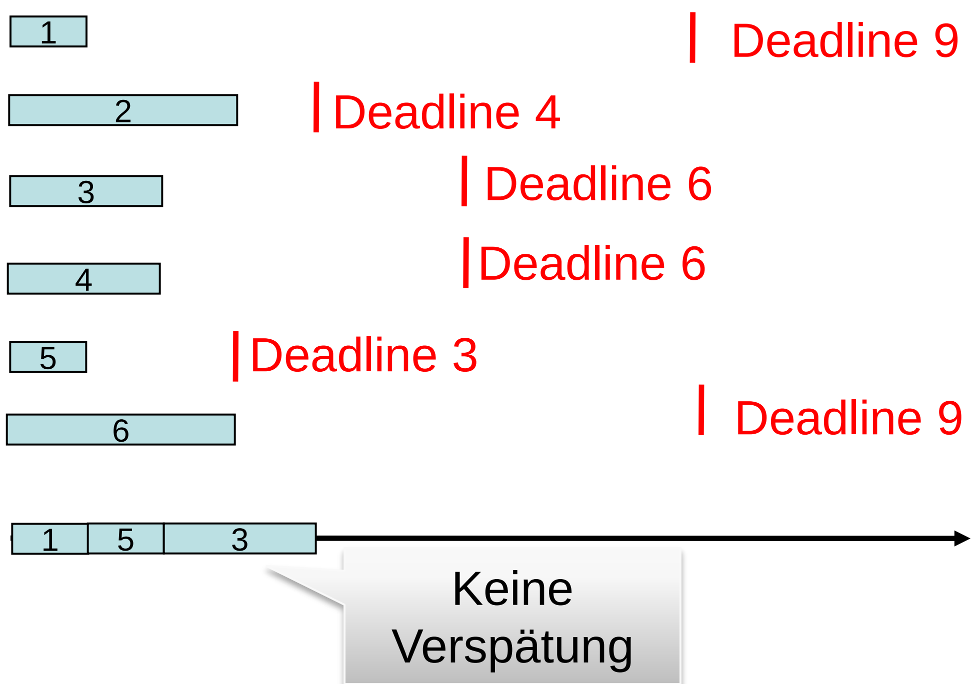
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

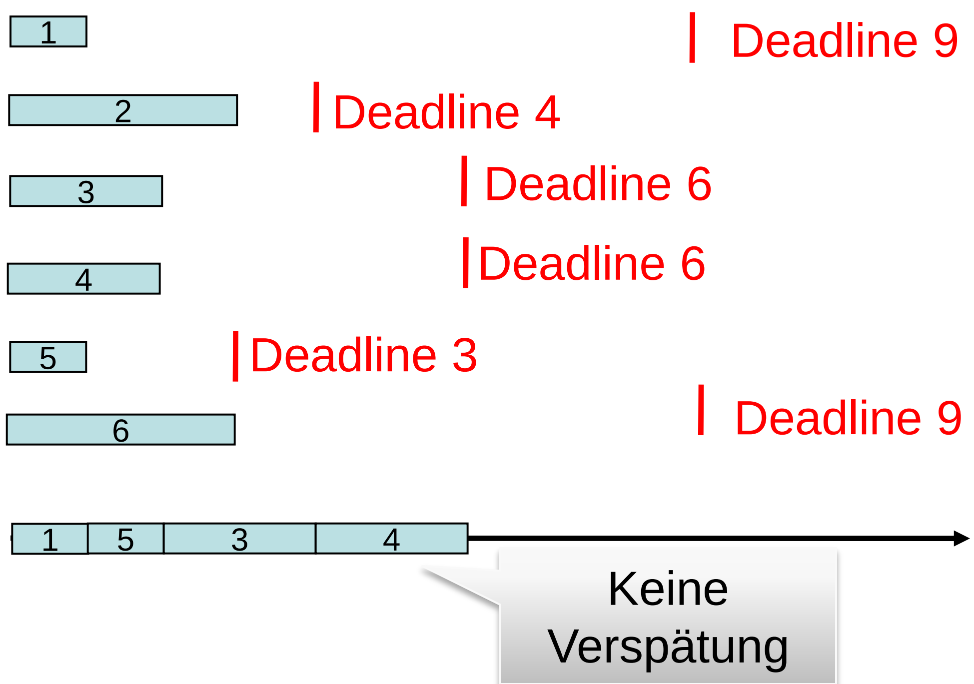
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

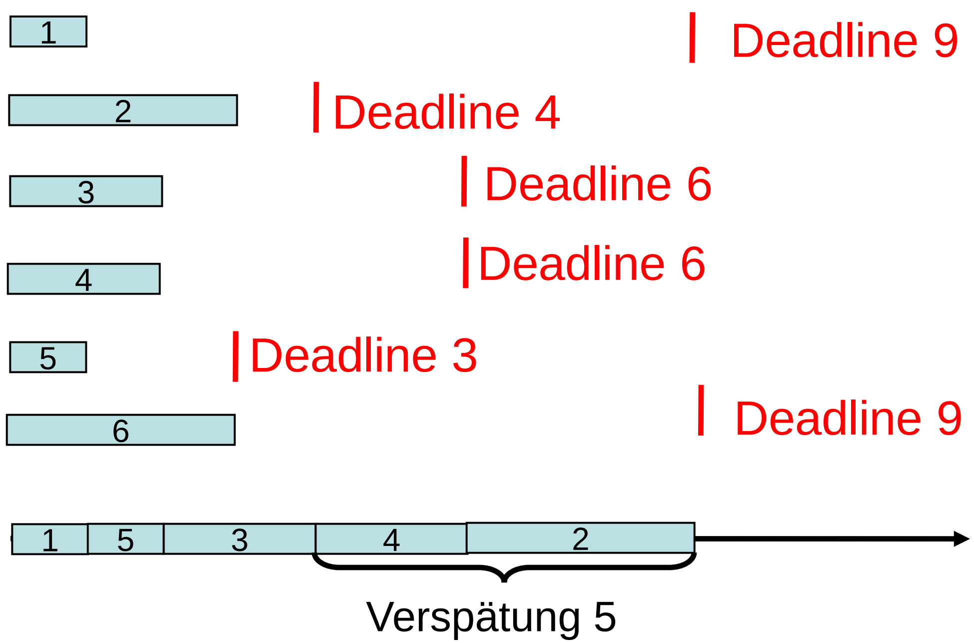
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

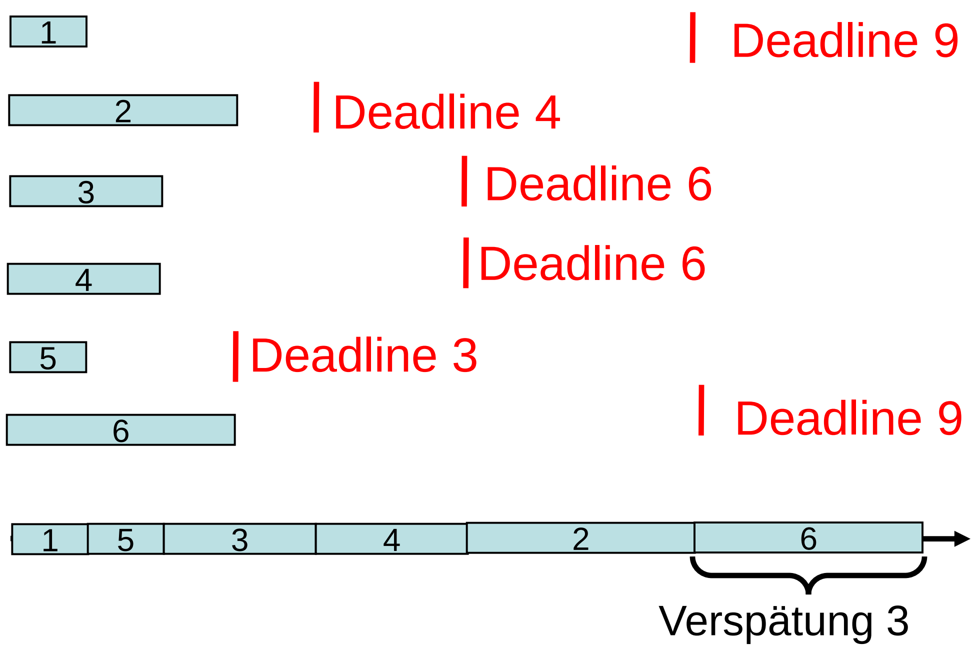
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

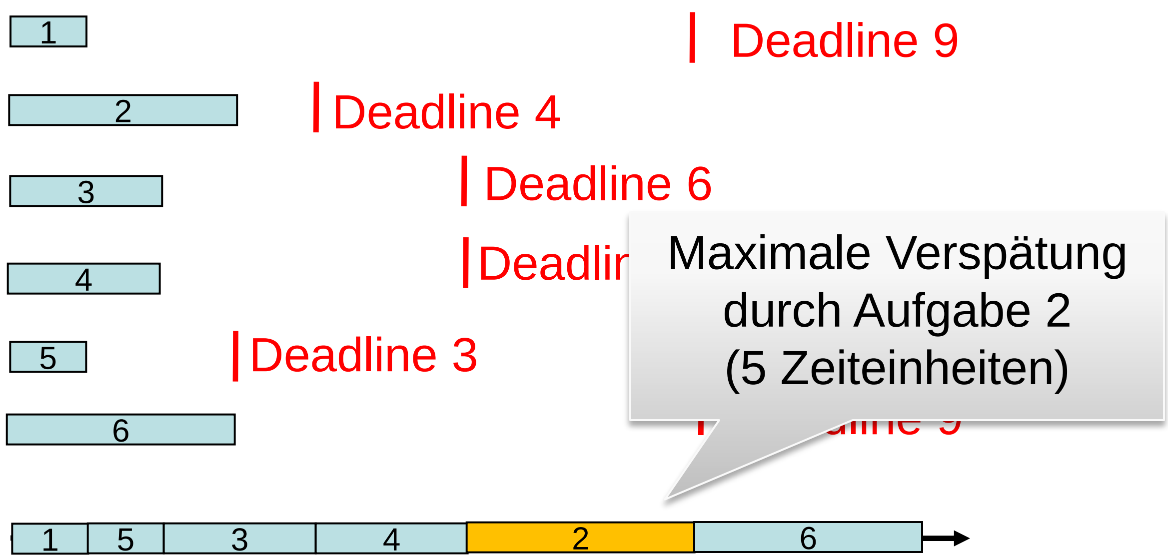
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

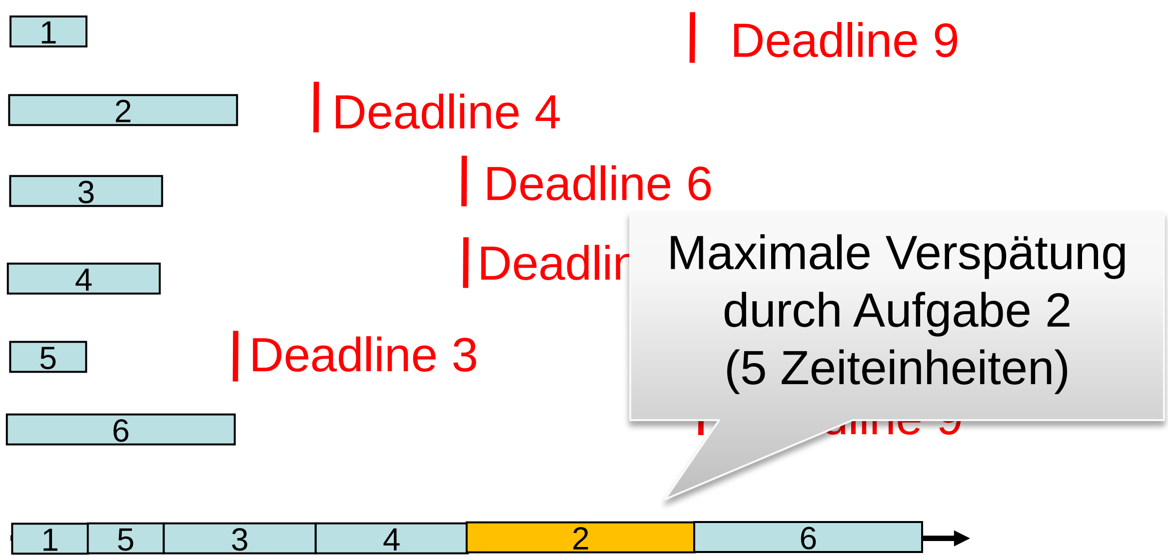
- Bearbeite die Jobs nach ansteigender Länge



Gierige Algorithmen

Strategie 1

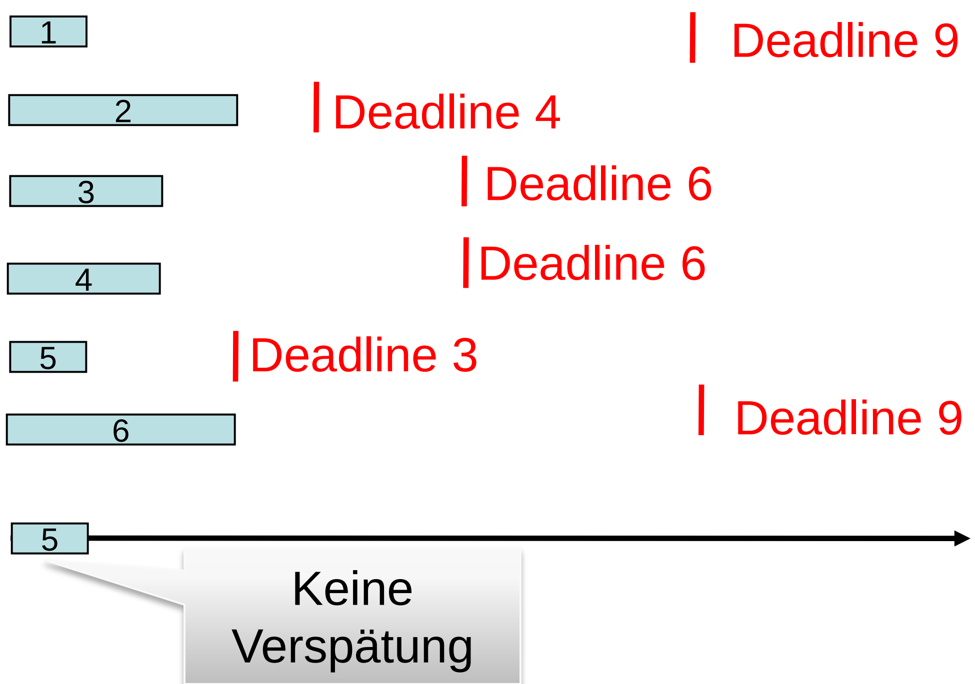
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

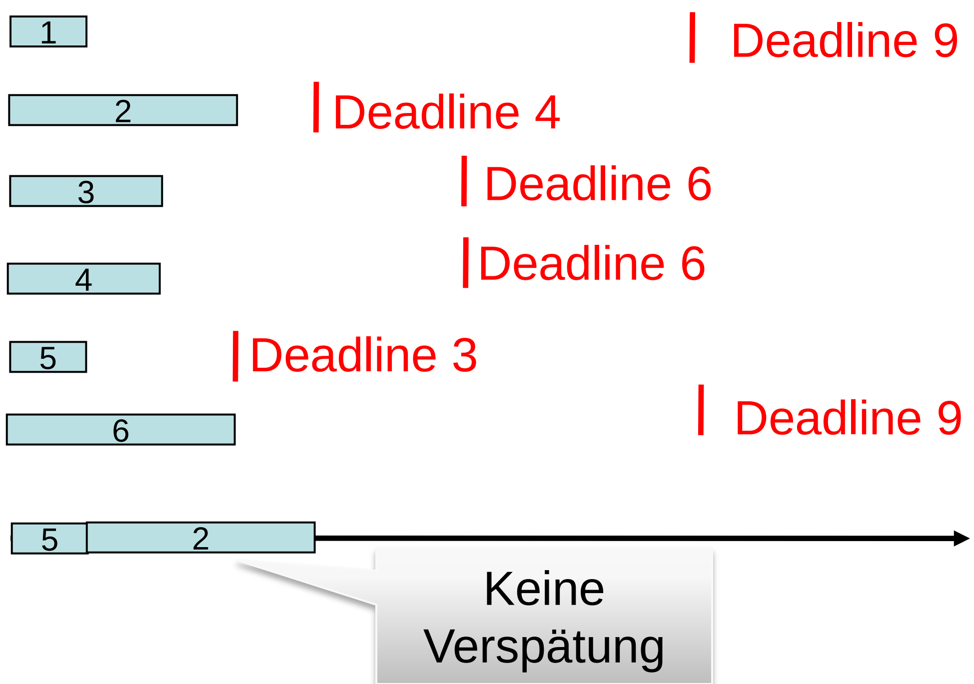
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

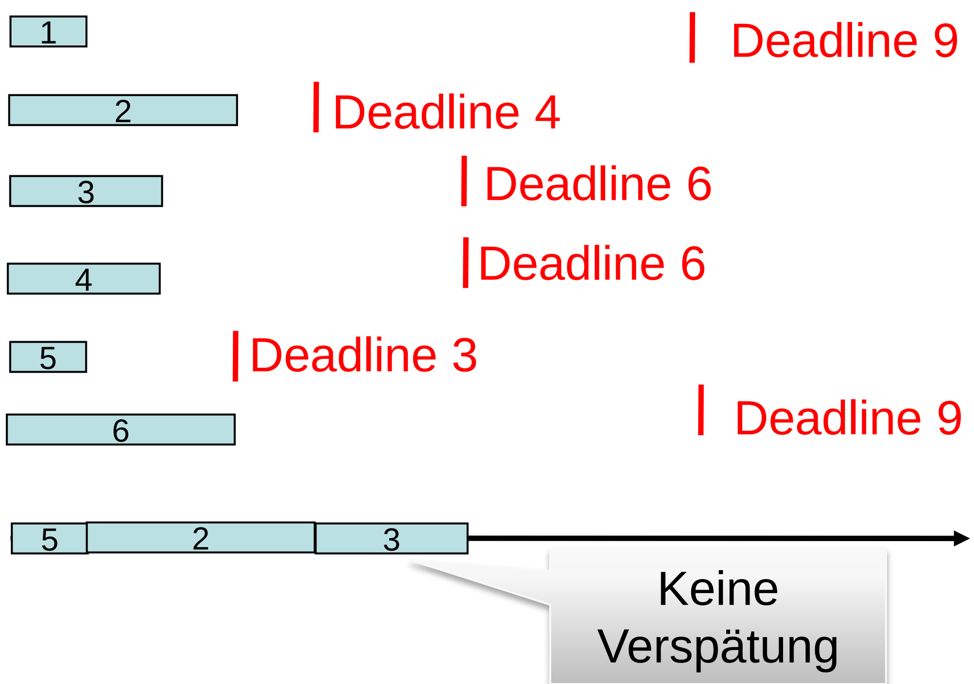
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

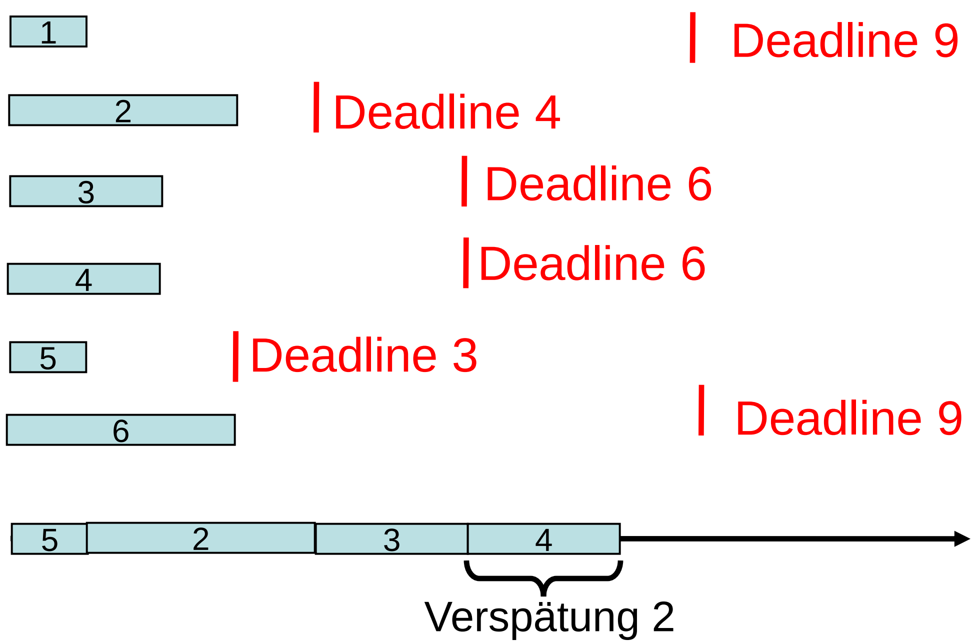
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

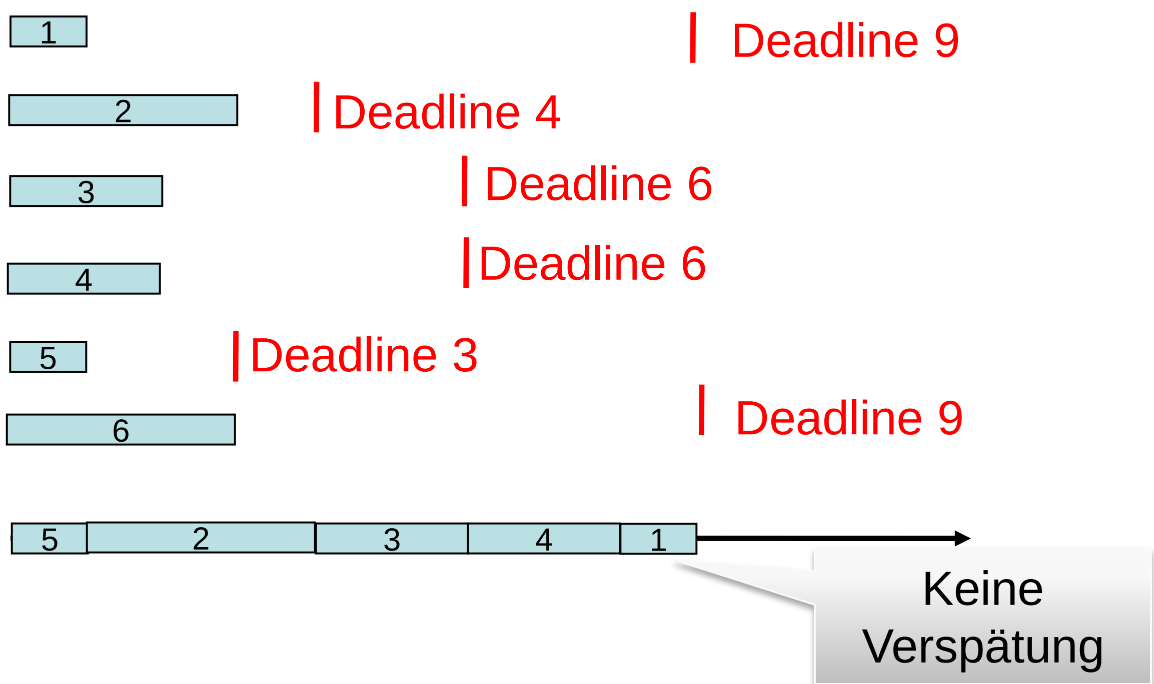
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

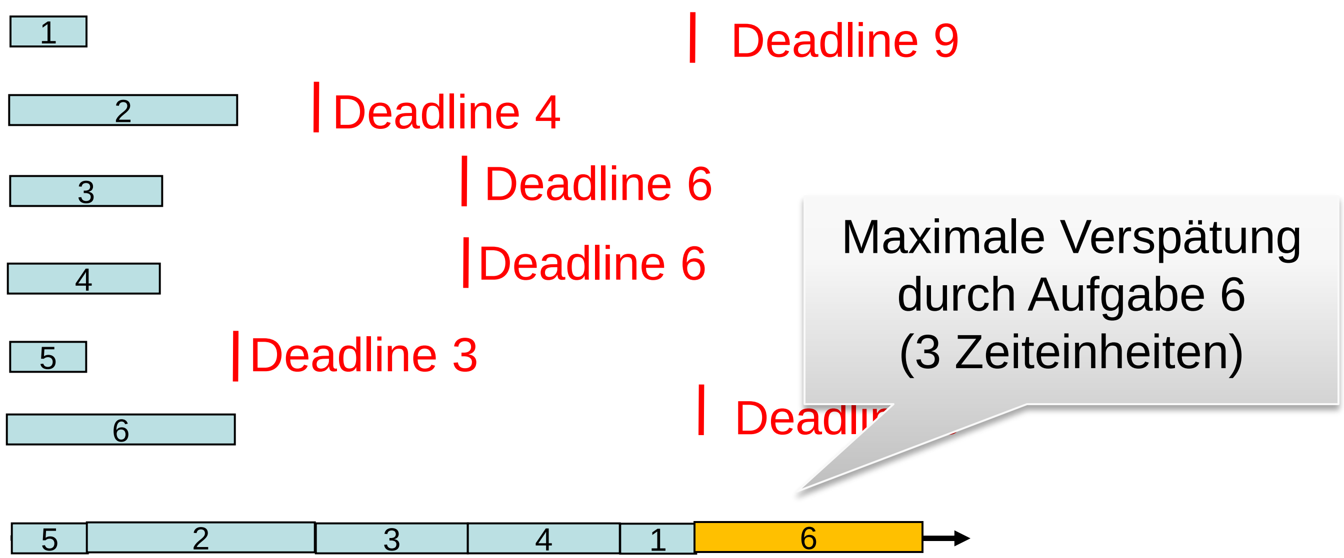
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

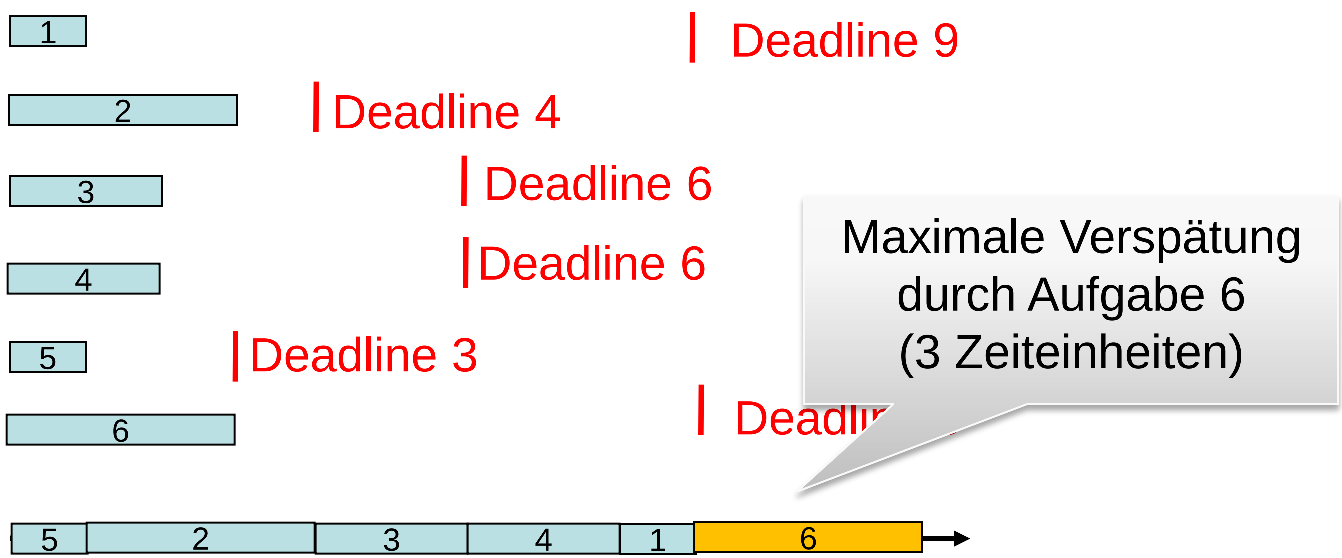
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?



Gierige Algorithmen

Strategie 1

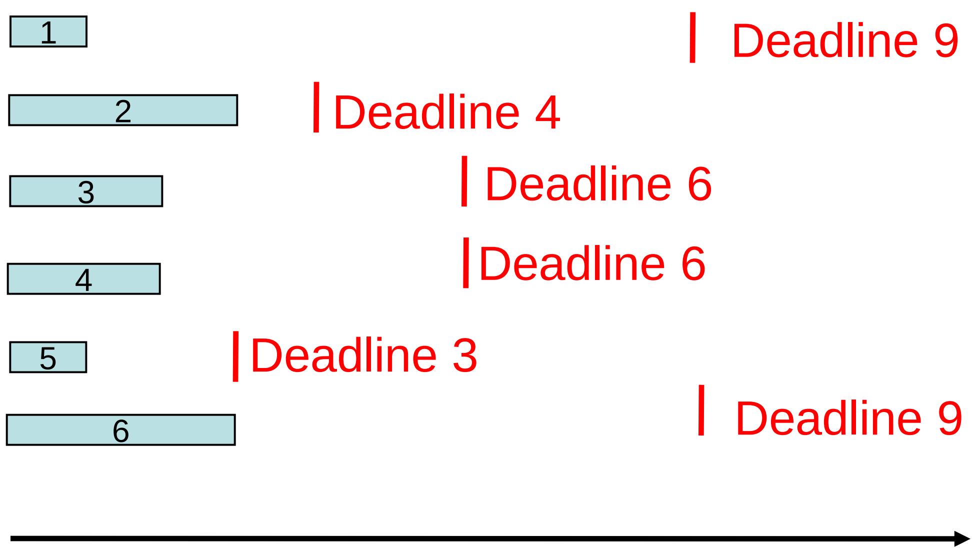
- Bearbeite die Jobs nach ansteigender Länge
- Optimalität?
- Problem: Ignoriert Deadlines völlig



Gierige Algorithmen

Strategie 2

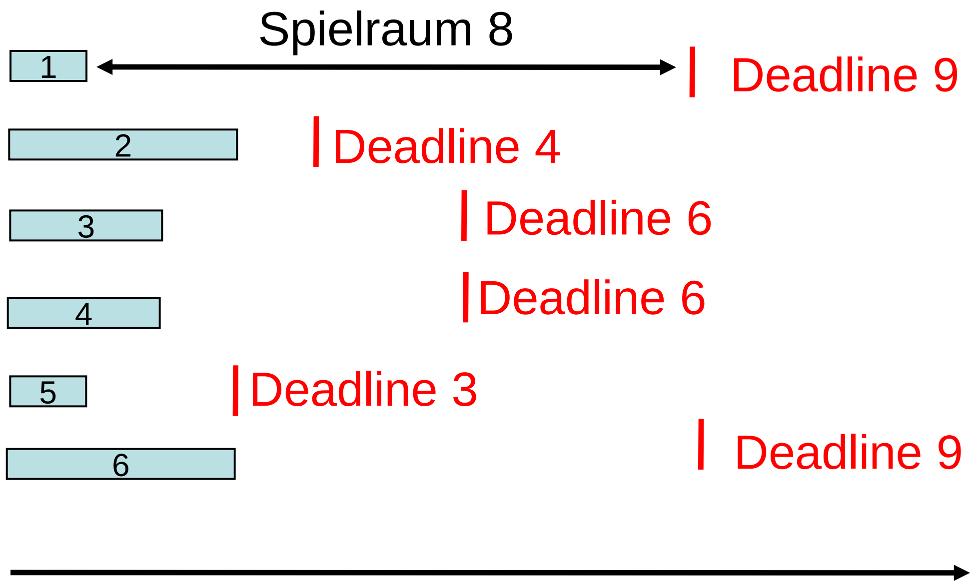
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

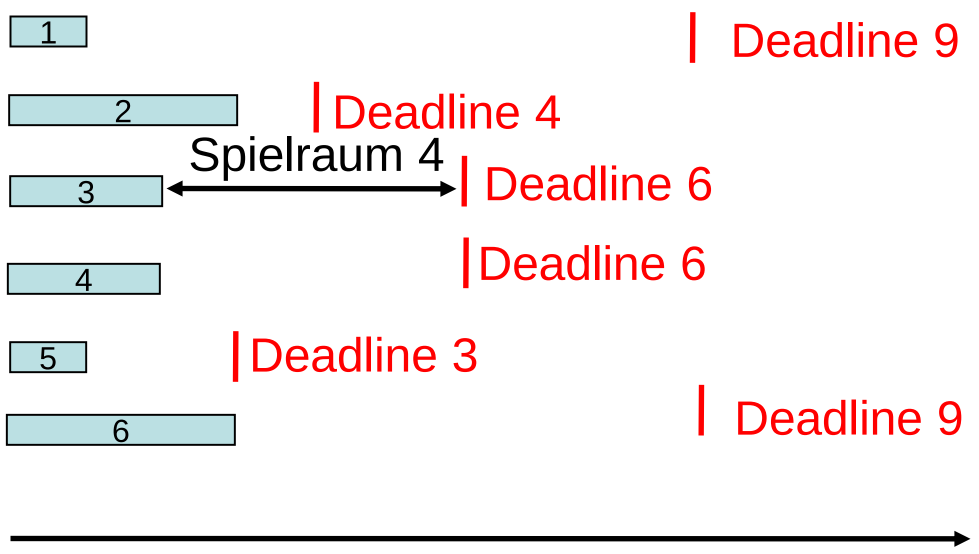
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

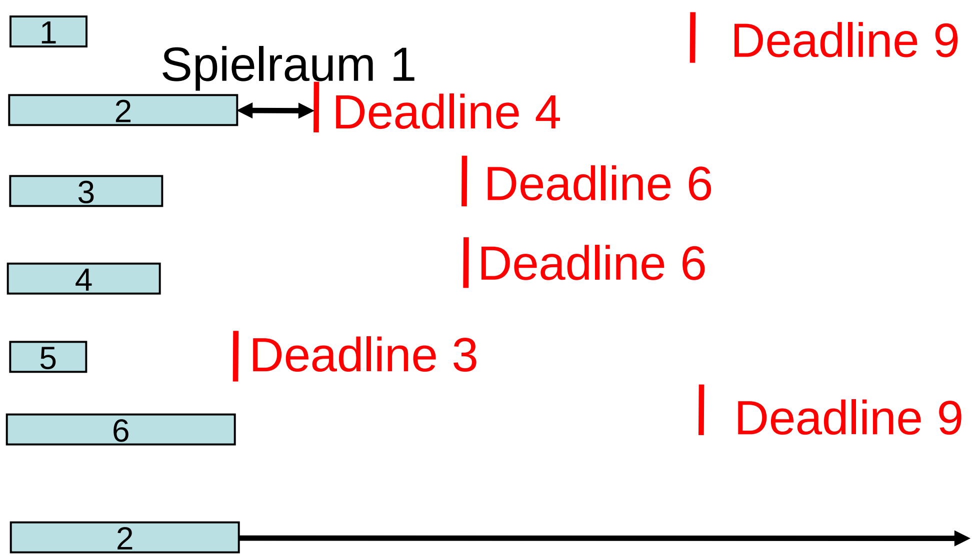
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

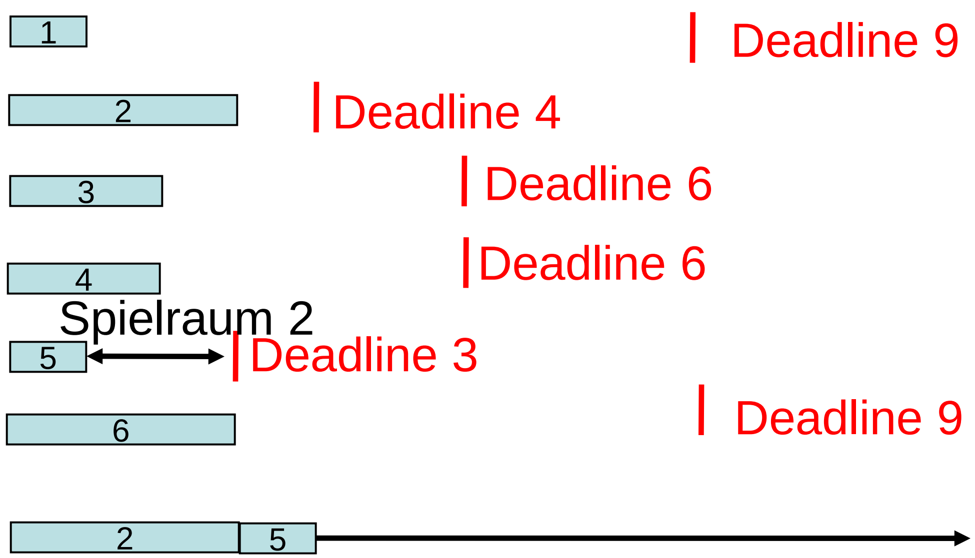
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

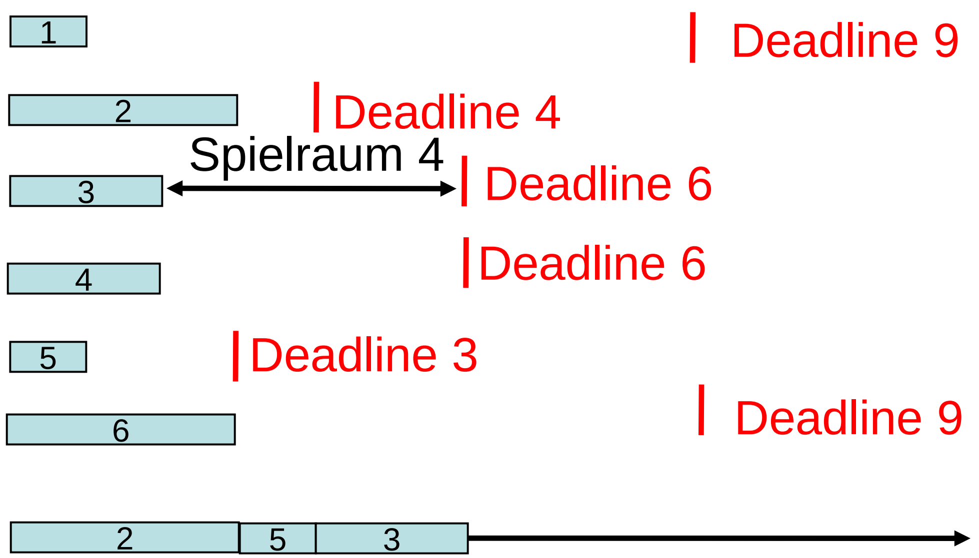
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

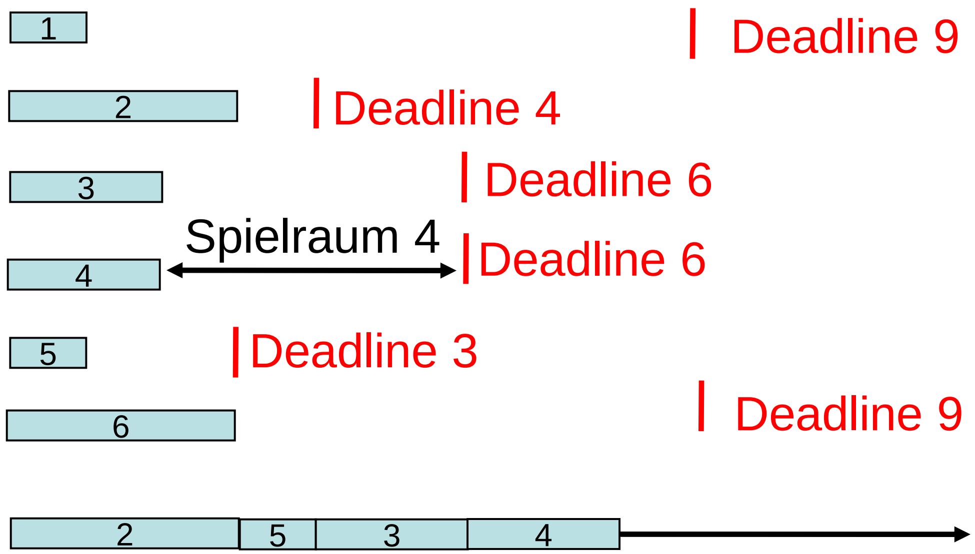
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

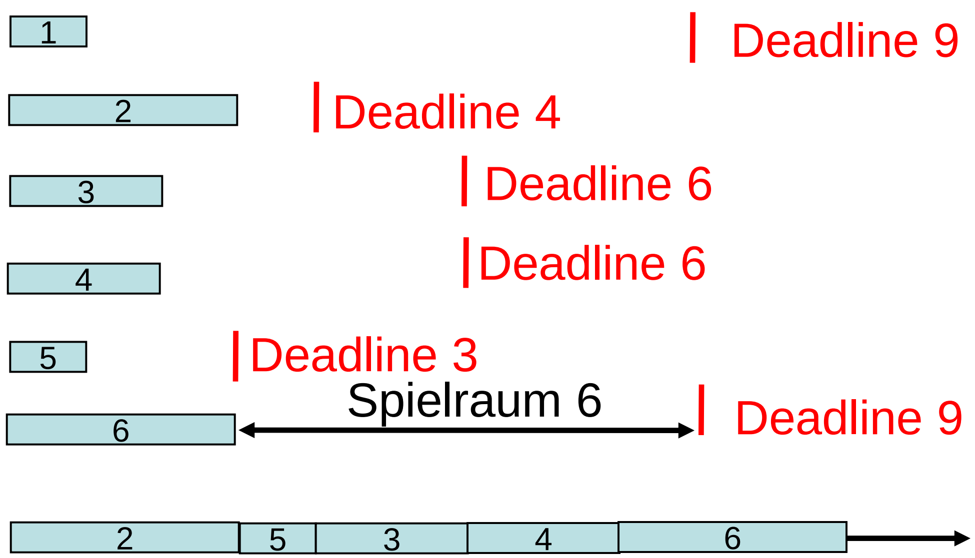
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

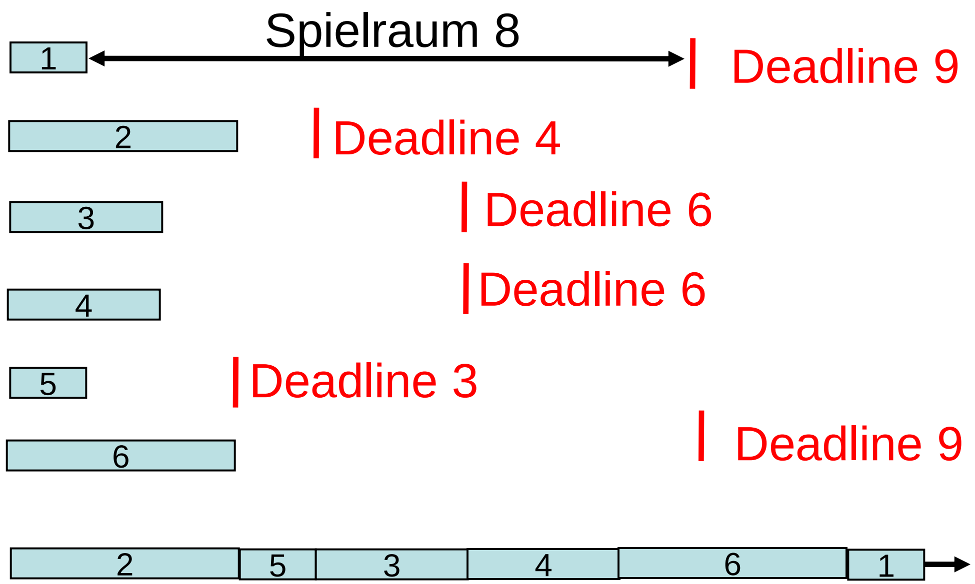
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

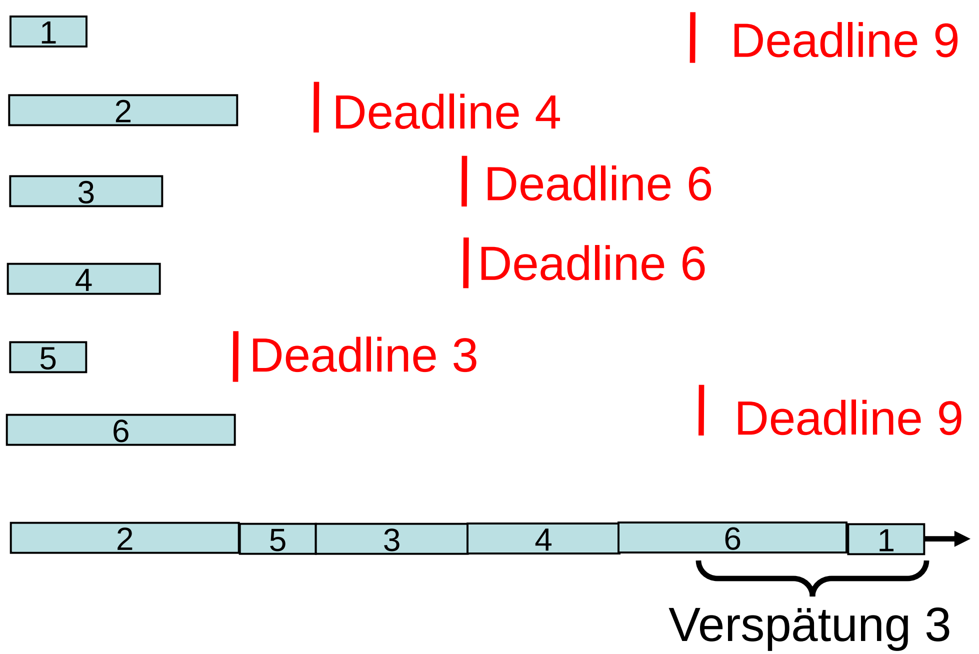
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

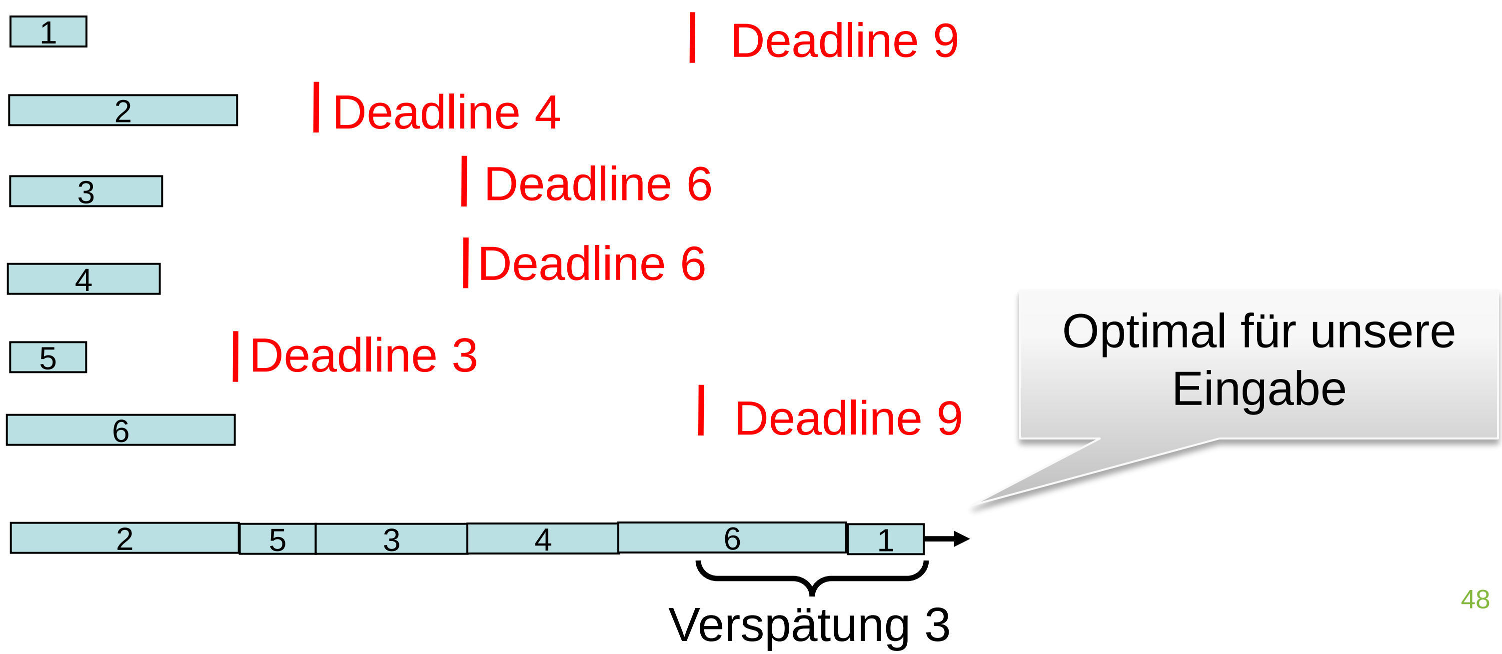
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

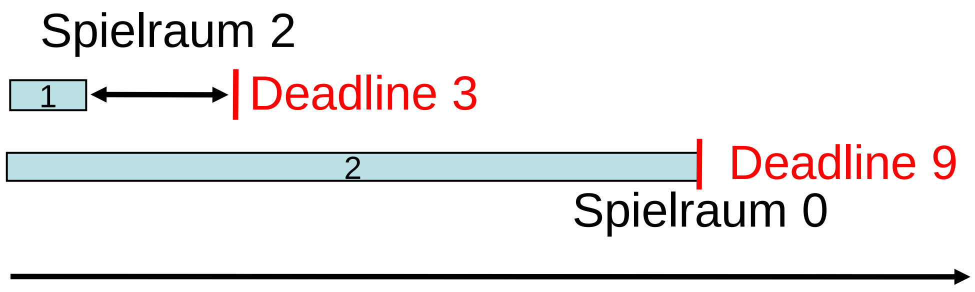
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t



Gierige Algorithmen

Strategie 2

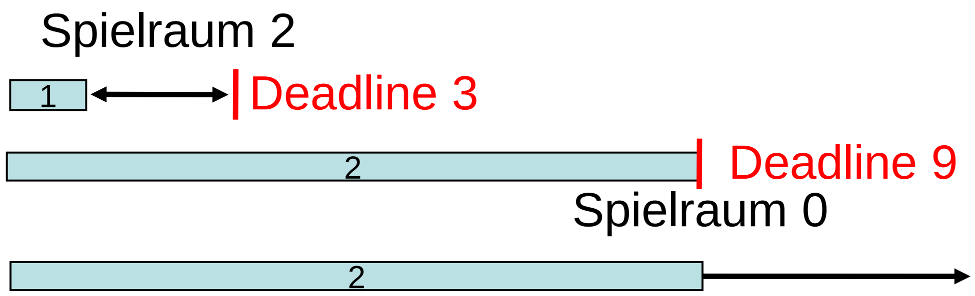
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t
- Optimalität?



Gierige Algorithmen

Strategie 2

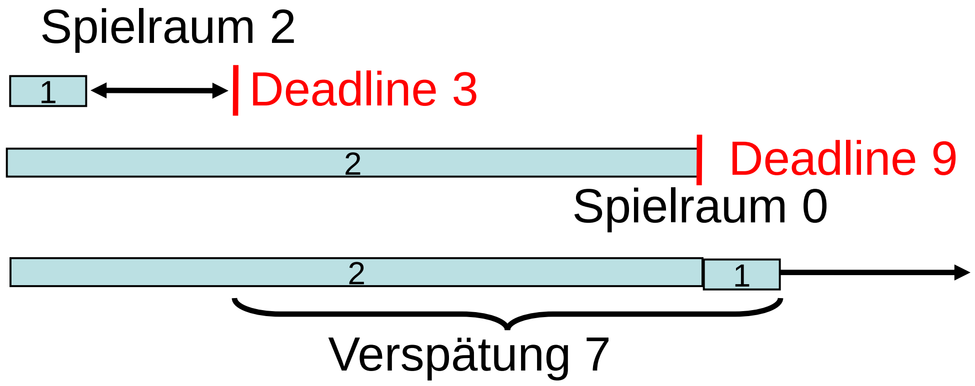
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t
- Optimalität?



Gierige Algorithmen

Strategie 2

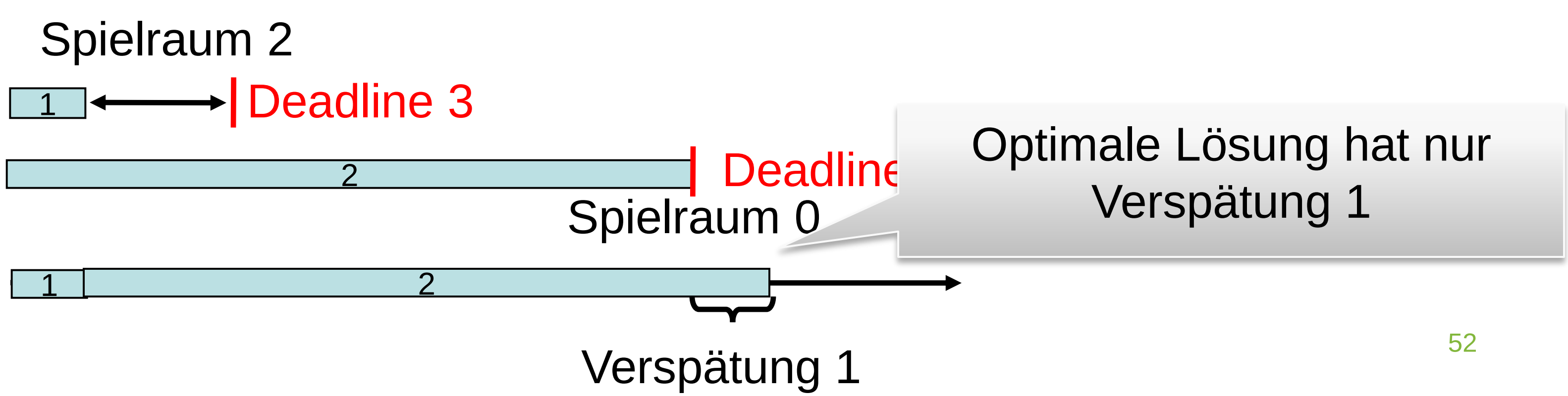
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t
- Optimalität?



Gierige Algorithmen

Strategie 2

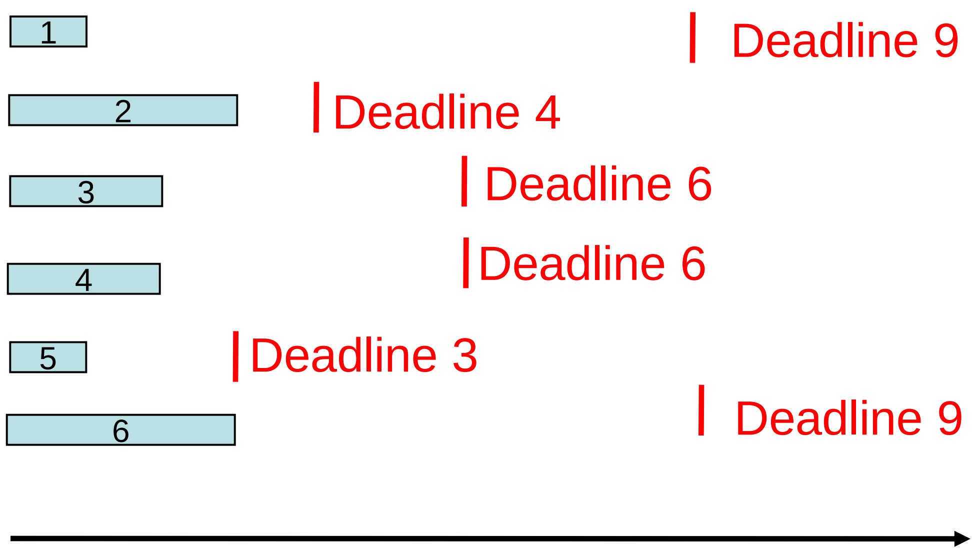
- Bearbeite zunächst die Aufgaben mit geringstem Spielraum d-t
- Optimalität?



Gierige Algorithmen

Strategie 3

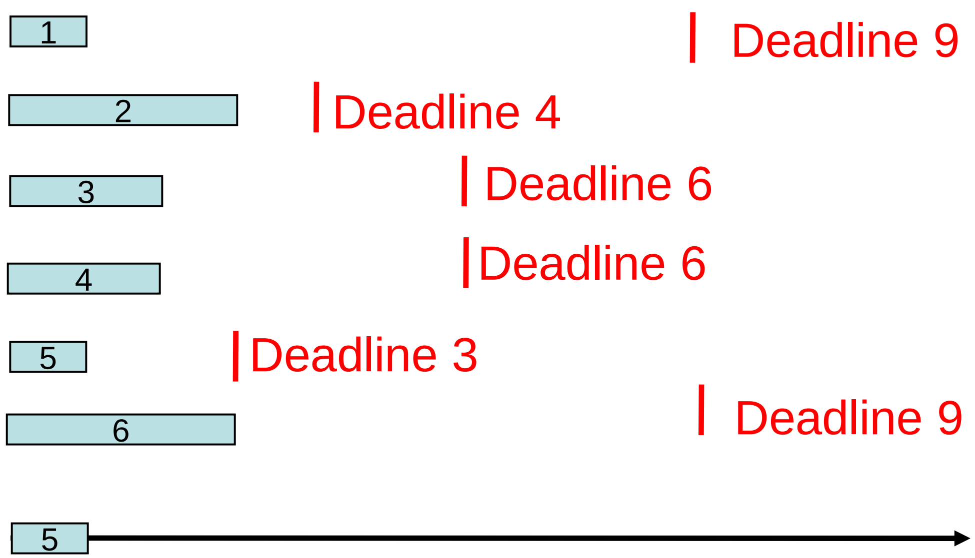
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

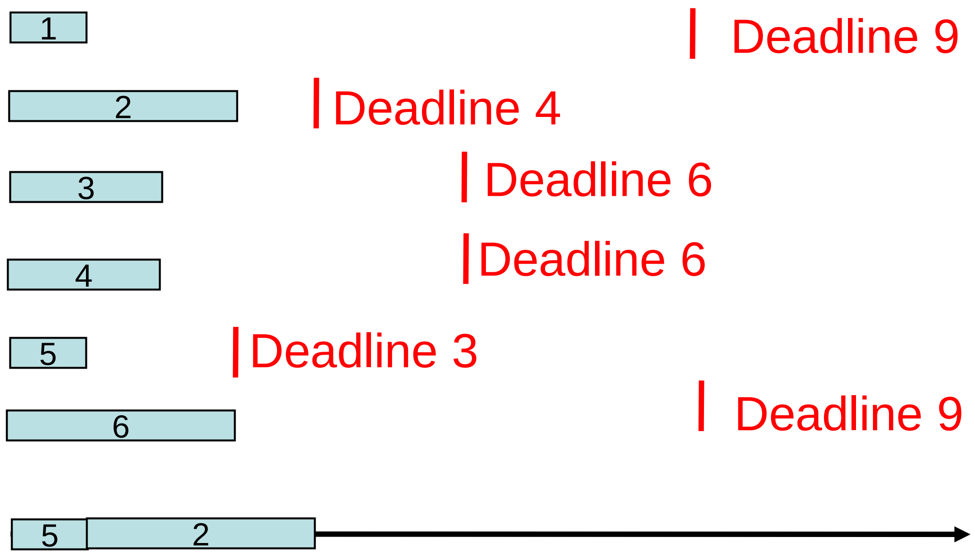
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

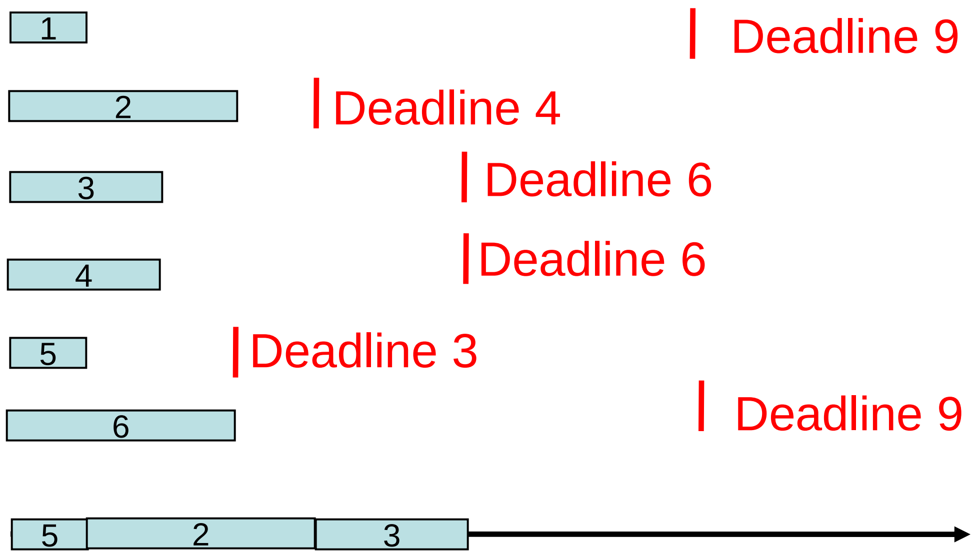
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

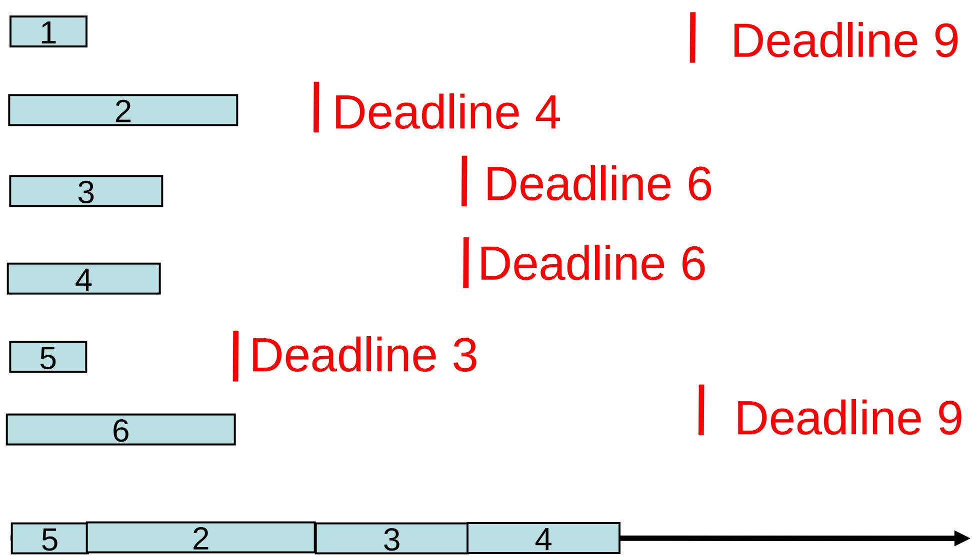
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

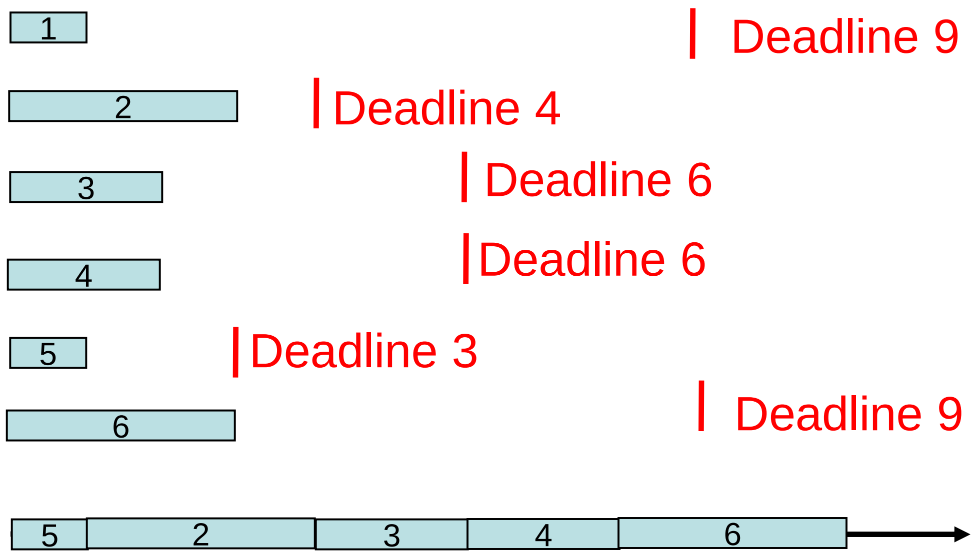
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

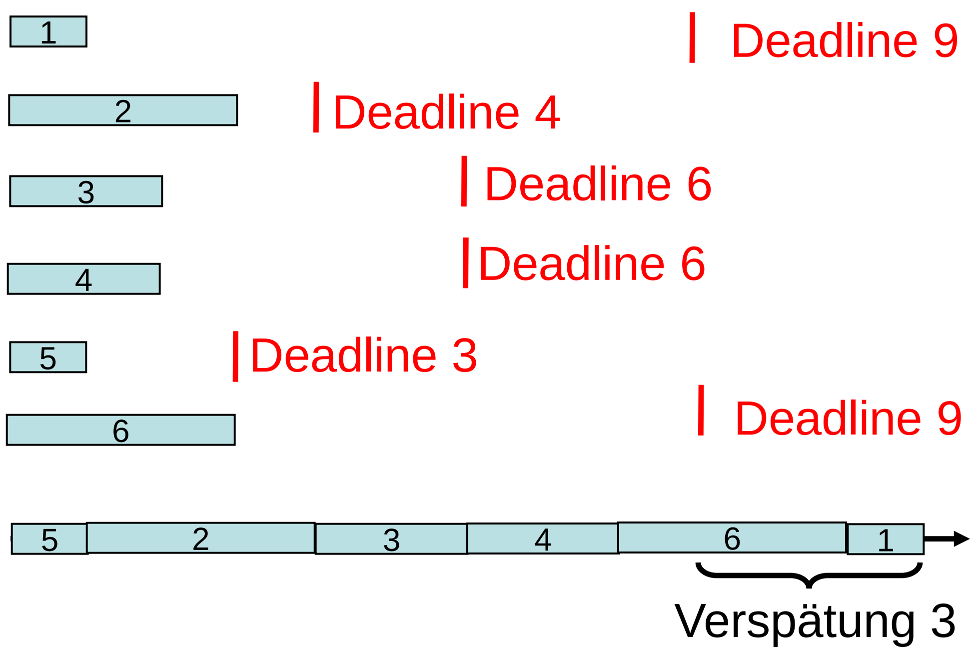
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

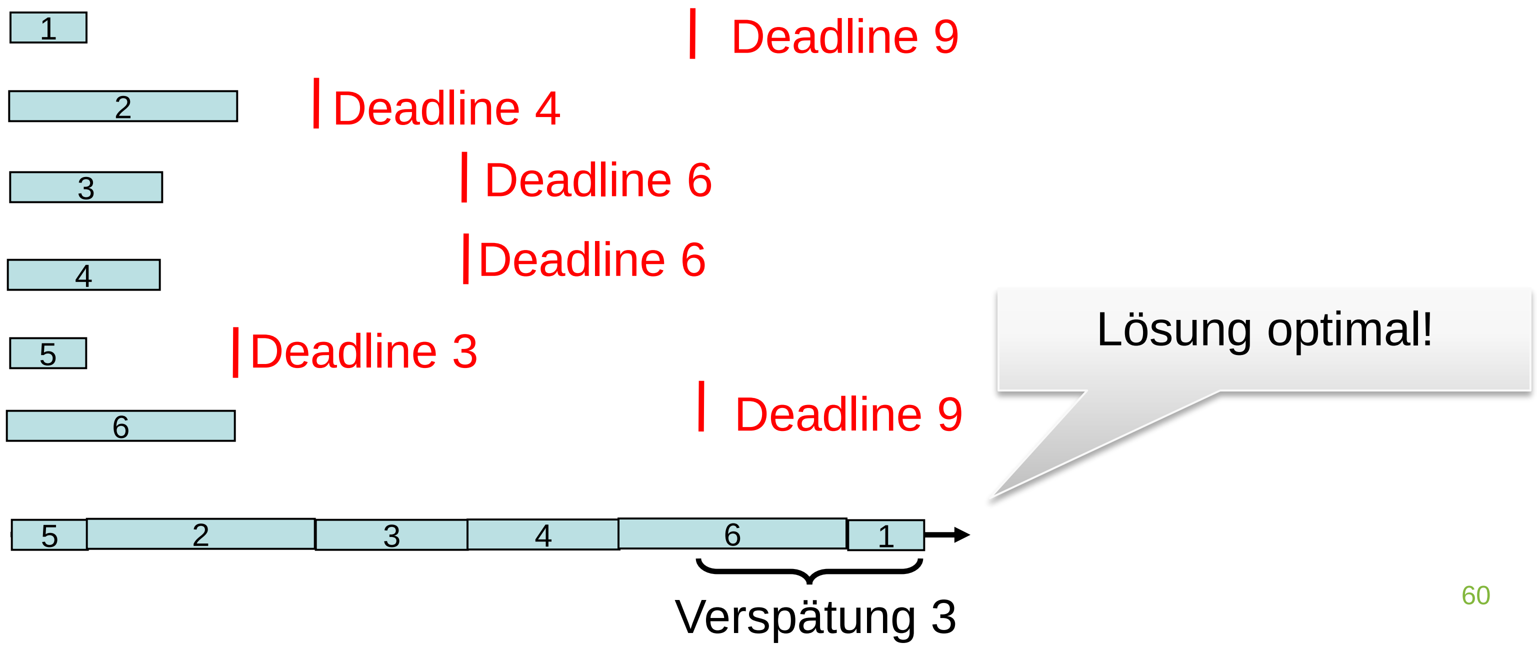
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

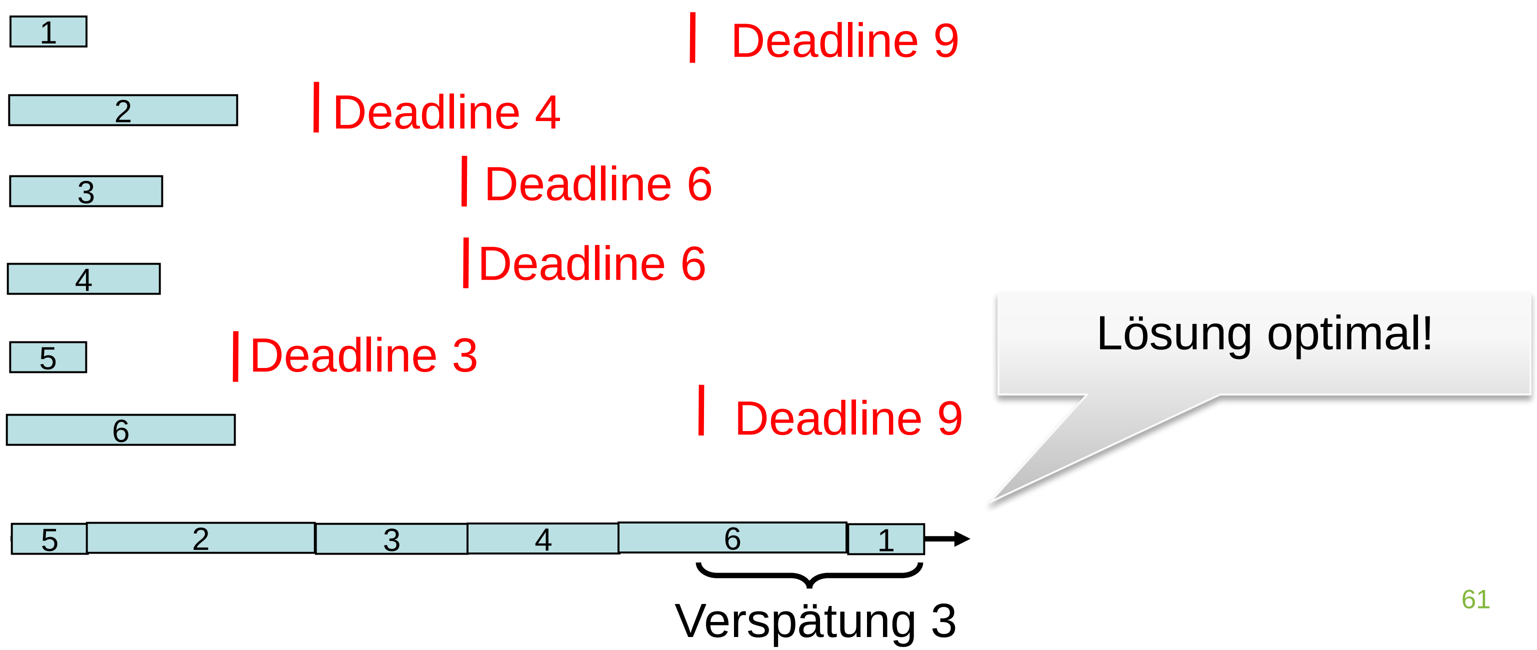
- Bearbeite zunächst die Aufgabe mit der frühesten Deadline



Gierige Algorithmen

Strategie 3

- Bearbeite zunächst die Aufgabe mit der frühesten Deadline
- Algorithmus ist optimal!

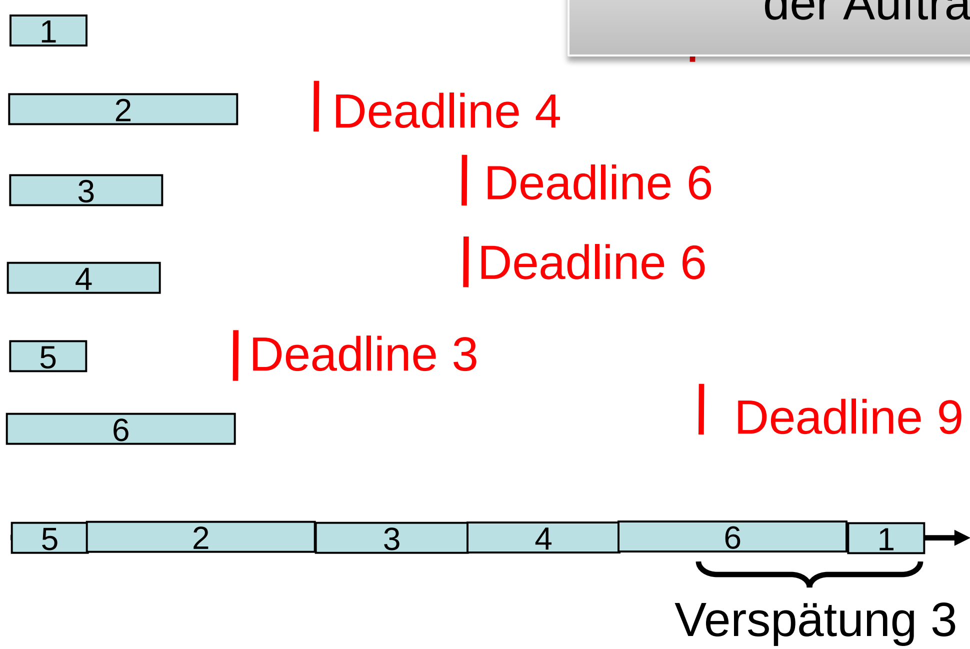


Gierige Algorithmen

Strategie 3

- Bearbeite zunächst die Aufgabe mit der frühesten Deadline
- Algorithmus ist optimal!

Komisch, da Strategie unabhängig von der Länge der Aufträge



Gierige Algorithmen

Formale Problemformulierung

- Problem: Scheduling mit Deadline
- Eingabe: Felder t und d
  t[i] enthält Länge des i-ten Intervals
  d[i] enthält Deadline
- Ausgabe: Startzeitpunkte der Intervalle

Wichtige Annahme

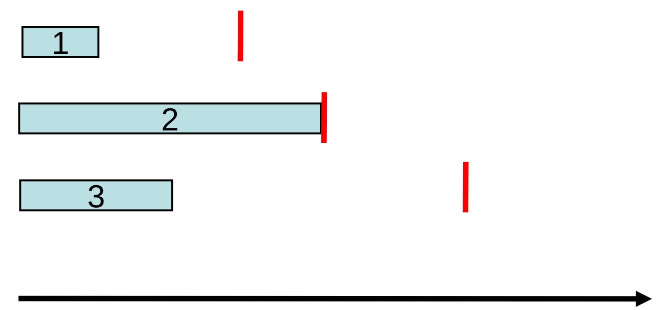
- Eingabe sortiert nach Deadlines
- $d[1] \leq d[2] \leq \ldots \leq d[n]$

Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

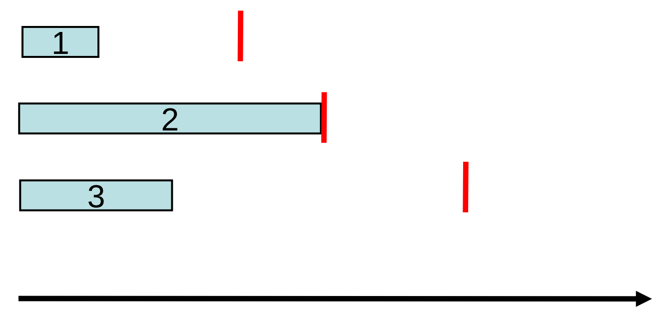


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5.   A[i] ← z
6.   z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

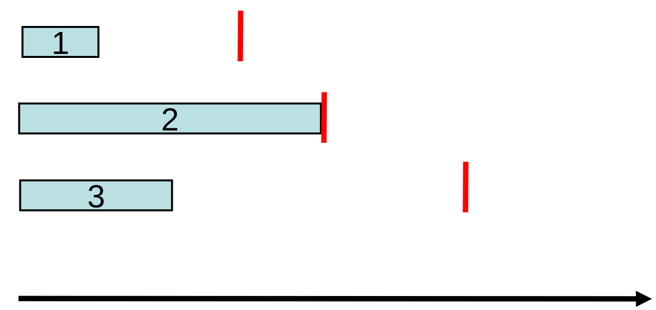


Gierige Algorithmen

LatenessScheduling(t,d)

```
n ← length[t]
new array A[1..n]
z ← 0
for i ← 1 to n do
    A[i] ← z
    z ← z + t[i]
return A
```

t	1	4	2
d	3	4	6

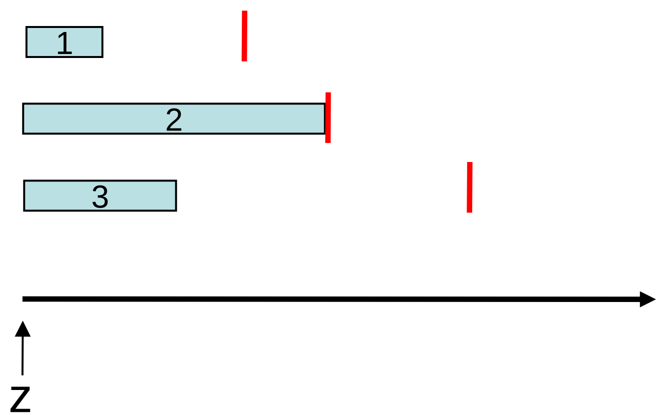


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5.   A[i] ← z
6.   z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

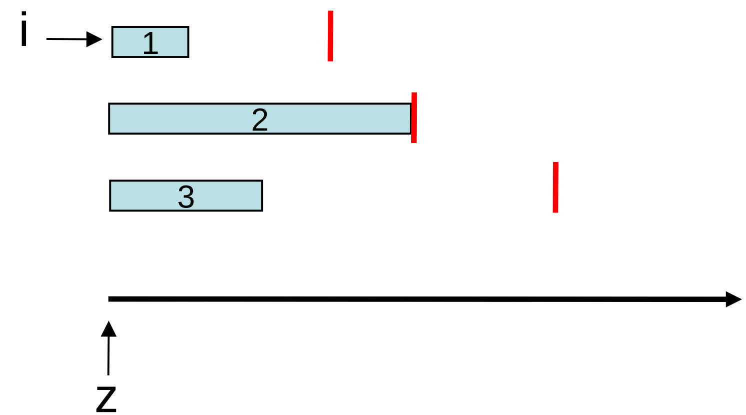


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5.   A[i] ← z
6.   z ← z + t[i]
7. **return** A

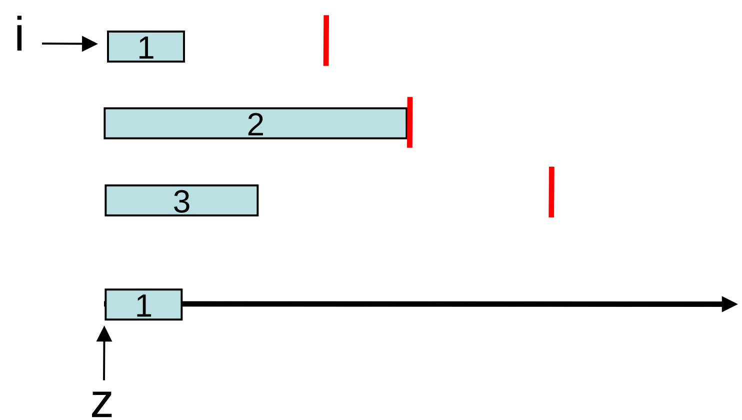
t	1	4	2
d	3	4	6



Gierige Algorithmen

```
LatenessScheduling(t,d)
n ← length[t]
new array A[1..n]
z ← 0
for i ← 1 to n do
    A[i] ← z
    z ← z + t[i]
return A
```

t	1	4	2
d	3	4	6

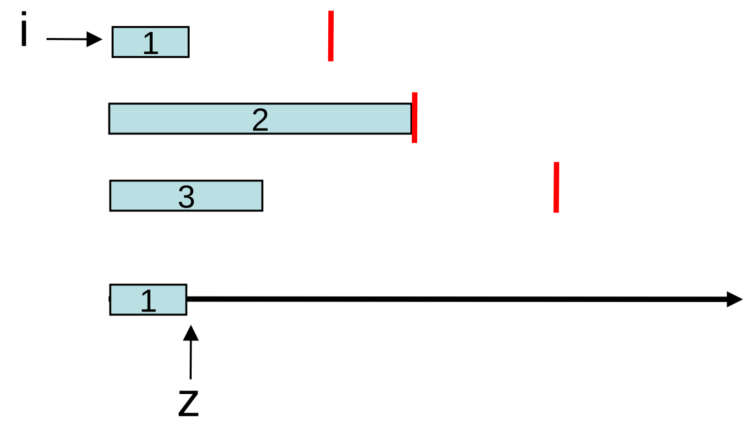


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. $\boxed{z \leftarrow z + t[i]}$
7. **return** A

t	1	4	2
d	3	4	6

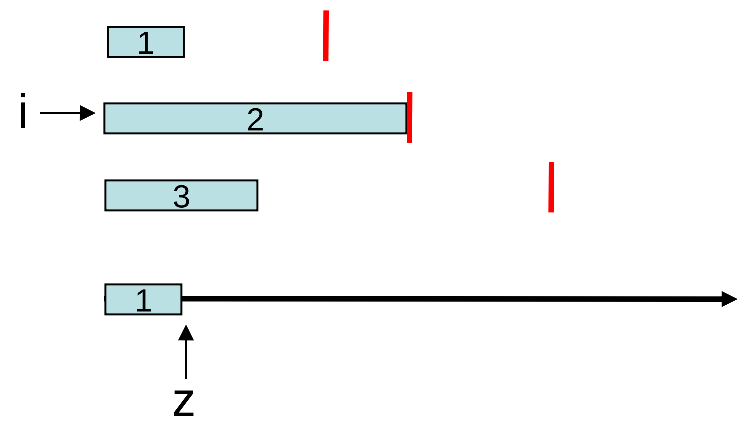


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

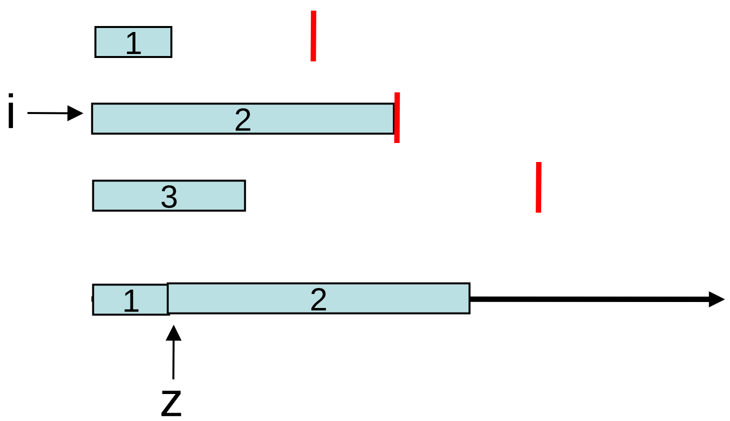


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. $\boxed{\text{A[i]} \leftarrow \text{z}}$
6. z ← z + t[i]
7. **return** A

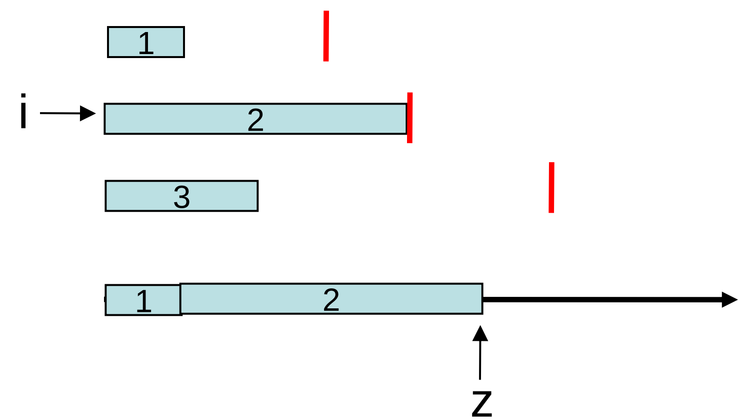
t	1	4	2
d	3	4	6



Gierige Algorithmen

```
LatenessScheduling(t,d)
n ← length[t]
new array A[1..n]
z ← 0
for i ← 1 to n do
    A[i] ← z
    z ← z + t[i]
return A
```

t	1	4	2
d	3	4	6

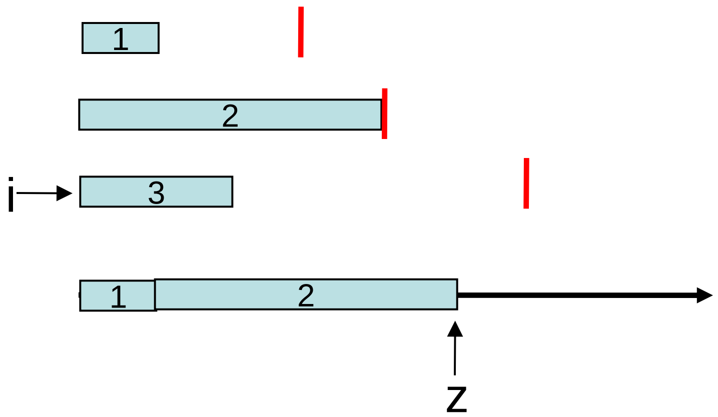


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

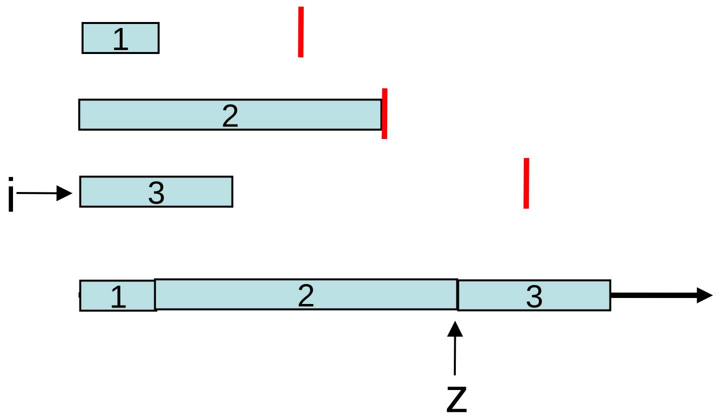


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

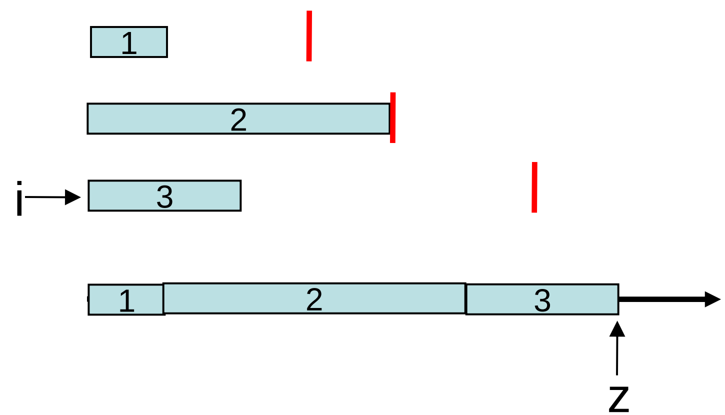


Gierige Algorithmen

LatenessScheduling(t,d)

```
n ← length[t]
new array A[1..n]
z ← 0
for i ← 1 to n do
    A[i] ← z
    z ← z + t[i]
return A
```

t	1	4	2
d	3	4	6

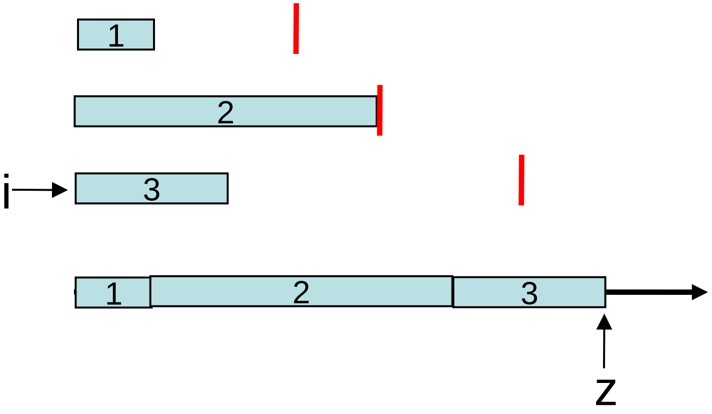


Gierige Algorithmen

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5. A[i] ← z
6. z ← z + t[i]
7. **return** A

t	1	4	2
d	3	4	6

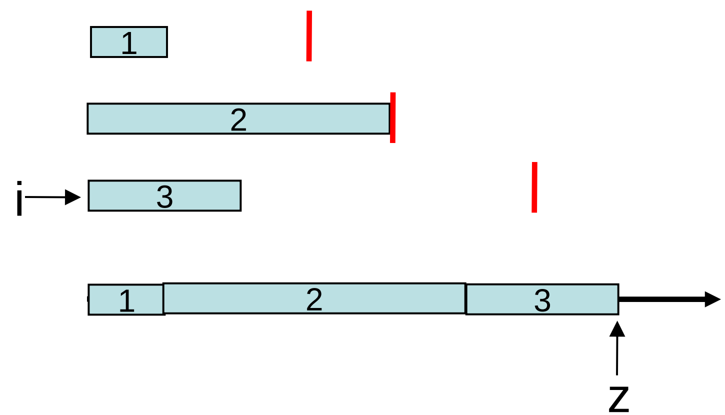


Gierige Algorithmen

t	1	4	2
d	3	4	6

LatenessScheduling(t,d)

1. n ← length[t]
2. new array A[1..n]
3. z ← 0
4. **for** i ← 1 **to** n **do**
5.   A[i] ← z
6.   z ← z + t[i]
7. **return** A

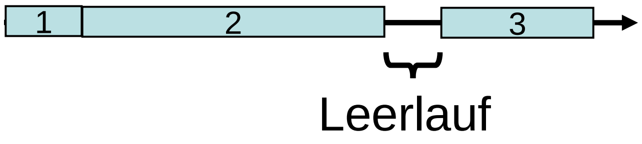


- Laufzeit des Algorithmus: O(n)
- Wichtige Konvention: Erzeugen von Feldern (Zeile 2) braucht Zeit proportional zur Größe des Feldes (also hier O(n))

Gierige Algorithmen

Beobachtung

Es gibt eine optimale Lösung ohne Leerlaufzeit.



Gierige Algorithmen

Lemma 15

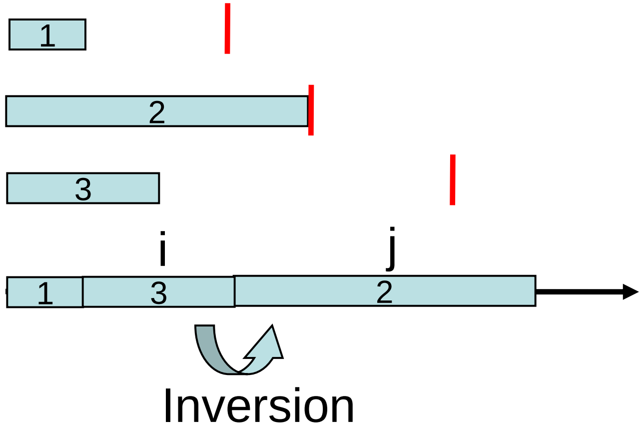
Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

Definition

Lösung hat Inversion, wenn Aufgabe i

Mit Deadline d_i vor Aufgabe j mit

Deadline $d_j < d_i$ bearbeitet wird.



Gierige Algorithmen

Lemma 15

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

Beweis

- Haben zwei Schedules weder Inversionen noch Leerlaufzeiten, so haben sie zwar nicht notwendigerweise dieselbe Ordnung, aber sie können sich nur in der Ordnung der Aufgaben mit identischer Deadline unterscheiden. Betrachten wir eine solche Deadline d.

Gierige Algorithmen

Lemma 15

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

Beweis

- Haben zwei Schedules weder Inversionen noch Leerlaufzeiten, so haben sie zwar nicht notwendigerweise dieselbe Ordnung, aber sie können sich nur in der Ordnung der Aufgaben mit identischer Deadline unterscheiden. Betrachten wir eine solche Deadline d. In beiden Schedules werden alle Aufgaben mit Deadline d nacheinander ausgeführt.

Gierige Algorithmen

Lemma 15

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

Beweis

- Haben zwei Schedules weder Inversionen noch Leerlaufzeiten, so haben sie zwar nicht notwendigerweise dieselbe Ordnung, aber sie können sich nur in der Ordnung der Aufgaben mit identischer Deadline unterscheiden. Betrachten wir eine solche Deadline d. In beiden Schedules werden alle Aufgaben mit Deadline d nacheinander ausgeführt. Unter den Aufgaben mit Deadline d hat die letzte die größte Verzögerung und diese hängt nicht von der Reihenfolge der Aufgaben ab.

Gierige Algorithmen

Lemma 15

Alle Lösungen ohne Inversionen und Leerlaufzeit haben dieselbe maximale Verzögerung.

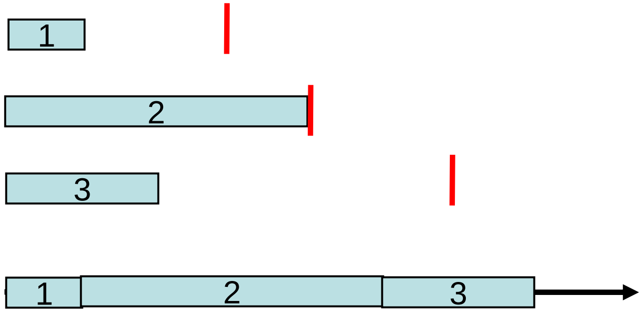
Beweis

- Haben zwei Schedules weder Inversionen noch Leerlaufzeiten, so haben sie zwar nicht notwendigerweise dieselbe Ordnung, aber sie können sich nur in der Ordnung der Aufgaben mit identischer Deadline unterscheiden. Betrachten wir eine solche Deadline d. In beiden Schedules werden alle Aufgaben mit Deadline d nacheinander ausgeführt. Unter den Aufgaben mit Deadline d hat die letzte die größte Verzögerung und diese hängt nicht von der Reihenfolge der Aufgaben ab.

Gierige Algorithmen

Lemma 16

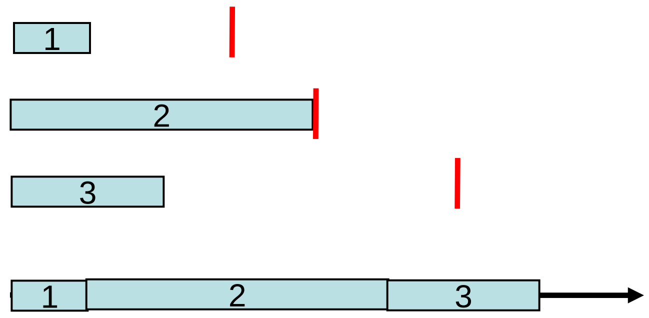
Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.



Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.



Ohne Inversionen und Leerlauf: Also optimal!

Gierige Algorithmen

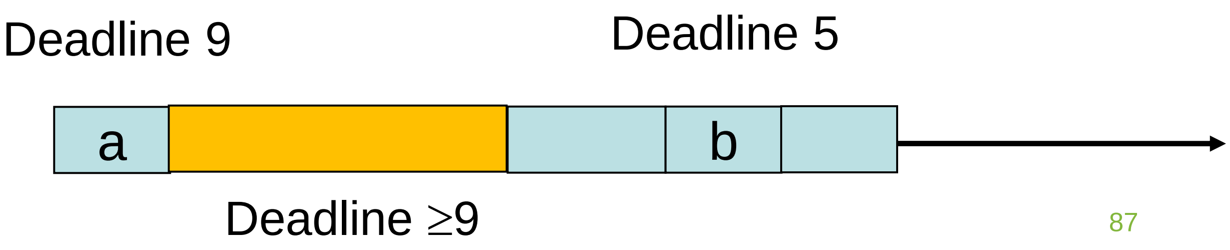
Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst

- (a) Wenn O eine Inversion hat, dann gibt es ein Paar Aufgaben i und j, so dass j direkt nach i auftritt und $d_j < d_i$ ist.
(D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



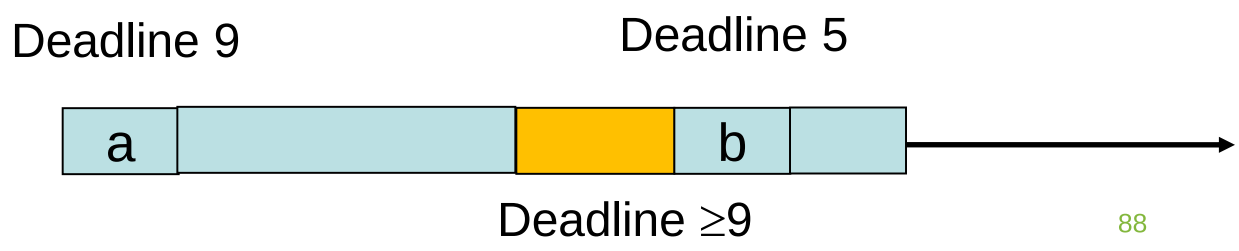
Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (a) Wenn O eine Inversion hat, dann gibt es ein Paar Aufgaben i und j, so dass j direkt nach i auftritt und $d_j < d_i$ ist.
  (D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



Gierige Algorithmen

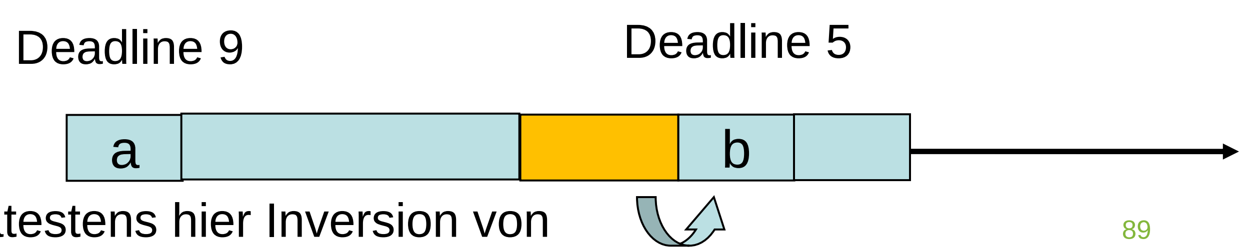
Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst

- (a) Wenn O eine Inversion hat, dann gibt es ein Paar Aufgaben i und j, so dass j direkt nach i auftritt und $d_j < d_i$ ist.
(D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



Spätestens hier Inversion von
aufeinander folgenden Aufgaben

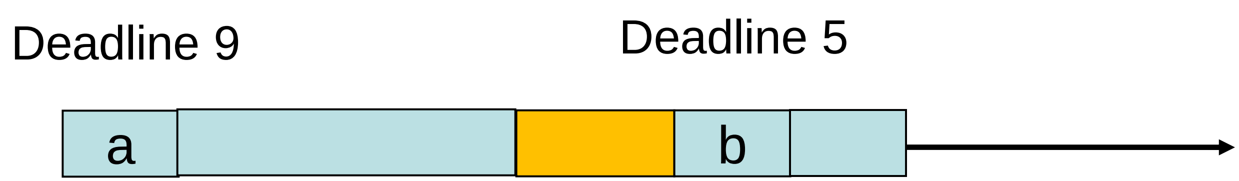
Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (a) Wenn O eine Inversion hat, dann gibt es ein Paar Aufgaben i und j, so dass j direkt nach i auftritt und $d_j < d_i$ ist.
  (D.h. eine Inversion von aufeinanderfolgenden Aufgaben)



Spätestens hier Inversion von aufeinander folgenden Aufgaben

Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (b) Nach dem Austauschen von einer benachbarten Inversion i und j erhalten wir ein Schedule mit einer Inversion weniger.
- Es wird die Inversion von i und j durch das Vertauschen aufgehoben und es wird keine neue Inversion wird erzeugt.

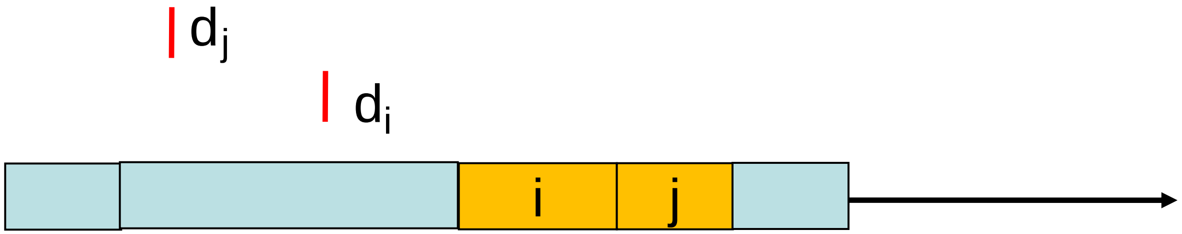
Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.



Aufeinander folgende Inversion (i,j)

Gierige Algorithmen

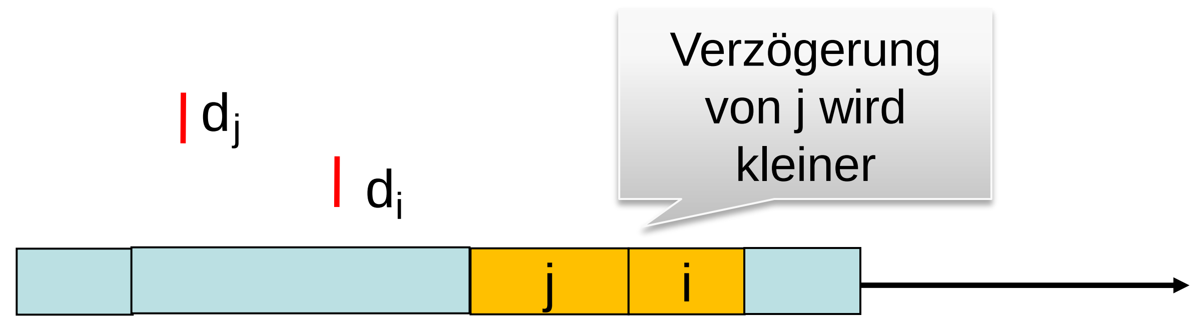
Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst

- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.



Vertausche i und j

Gierige Algorithmen

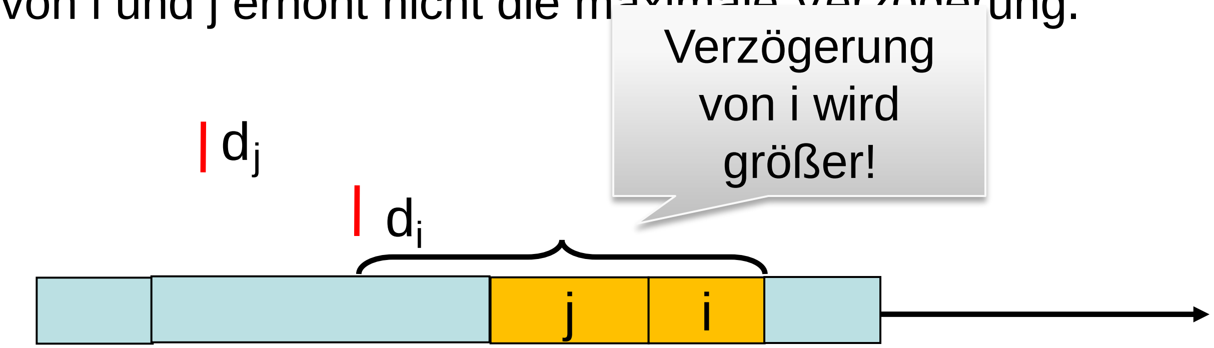
Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst

- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.



Vertausche i und j

Gierige Algorithmen

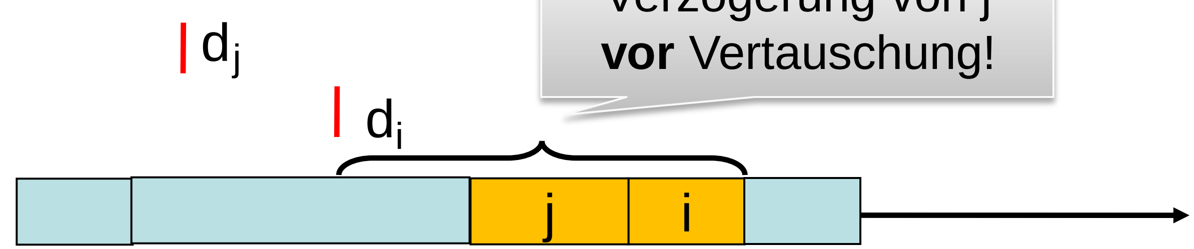
Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst
- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.

Ist aber kleiner als Verzögerung von j **vor** Vertauschung!



Vertausche i und j

Gierige Algorithmen

Lemma 16

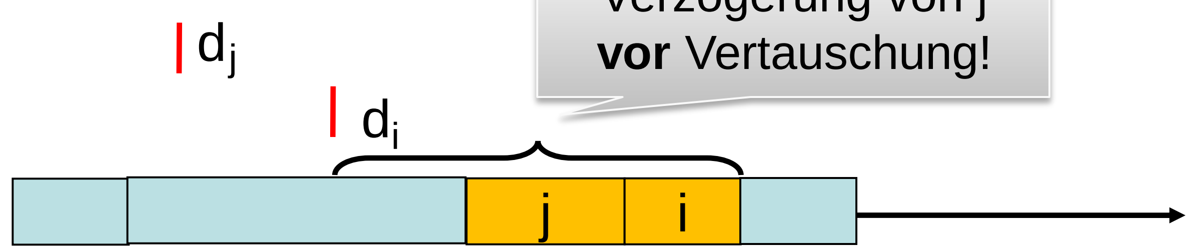
Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- Sei O ein optimales Schedule ohne Leerlauf. Wir zeigen zunächst

- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.

Ist aber kleiner als Verzögerung von j **vor** Vertauschung!

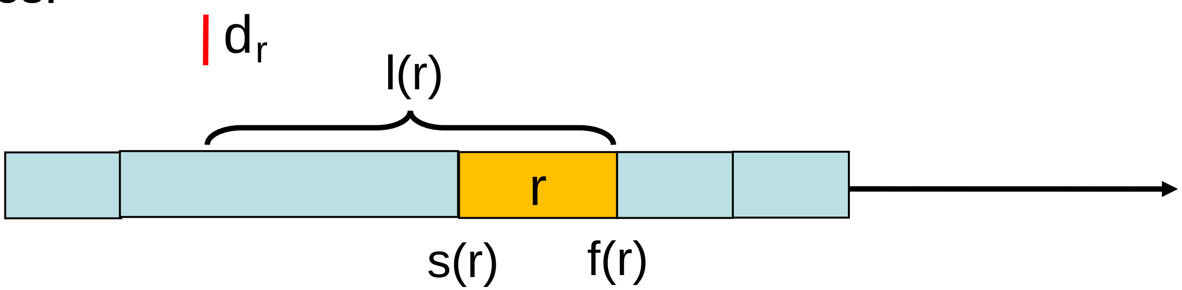


Vertausche i und j

Gierige Algorithmen

Formaler Beweis von (c)

- Notation für O: Aufgabe r wird im Invervall [s(r),f(r)] ausgeführt und hat Verzögerung l(r). Sei L = max l(r) die maximale Verzögerung dieses Schedules.

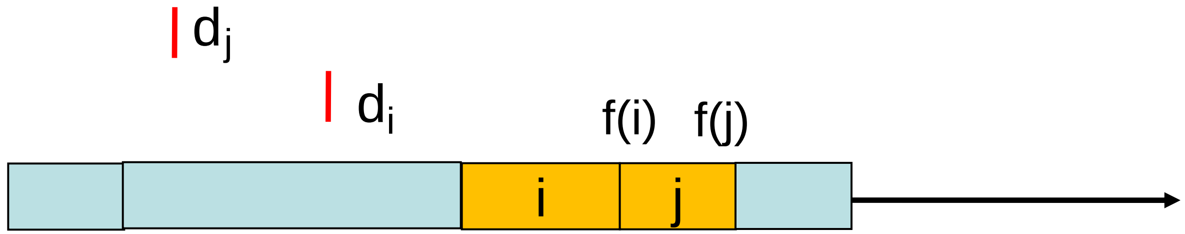


- Notation für das Schedule O* nach Austauschen: s*(r), f*(r), l*(r) und L* mit der entsprechenden Bedeutung wie oben.
- s(r), s*(r) heißt Startzeit
- f(r), f*(r) heißt Abarbeitungszeit

Gierige Algorithmen

Formaler Beweis von (c)

- Betrachten wir nun die benachbarte Inversion von i und j. Die Abarbeitungszeit f(j) von j vor dem Austauschen ist gleich der Abarbeitungszeit f*(i) von i nach dem Austauschen. Daher haben alle anderen Aufgaben vor und nach dem Tauschen dieselbe Abarbeitungszeit.

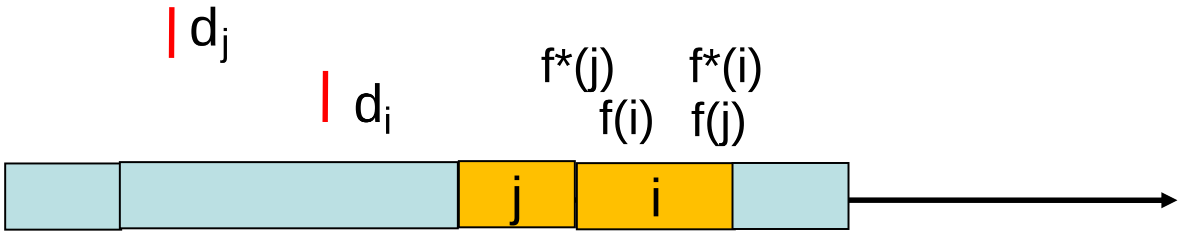


Aufeinander folgende Inversion (i,j)

Gierige Algorithmen

Formaler Beweis von (c)

- Betrachten wir nun die benachbarte Inversion von i und j. Die Abarbeitungszeit f(j) von j vor dem Austauschen ist gleich der Abarbeitungszeit f*(i) von i nach dem Austauschen. Daher haben alle anderen Aufgaben vor und nach dem Tauschen dieselbe Abarbeitungszeit.
- Für Aufgabe j ist das neue Schedule besser, d.h. f*(j)<f(j).

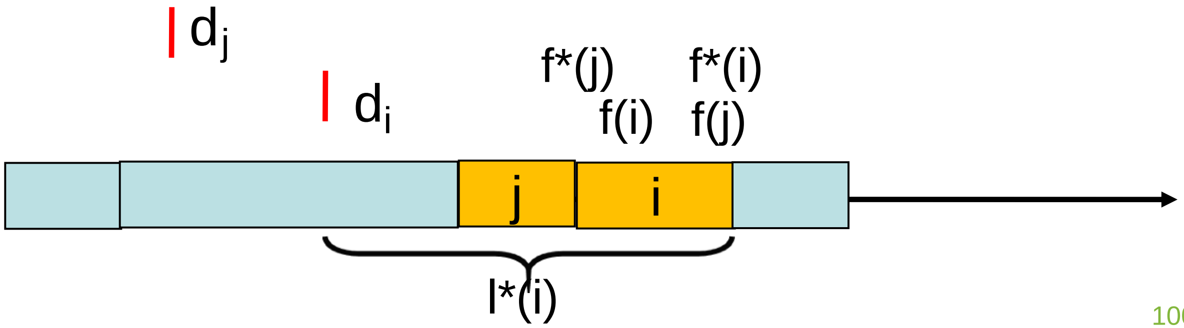


Aufeinander folgende Inversion (i,j)

Gierige Algorithmen

Formaler Beweis von (c)

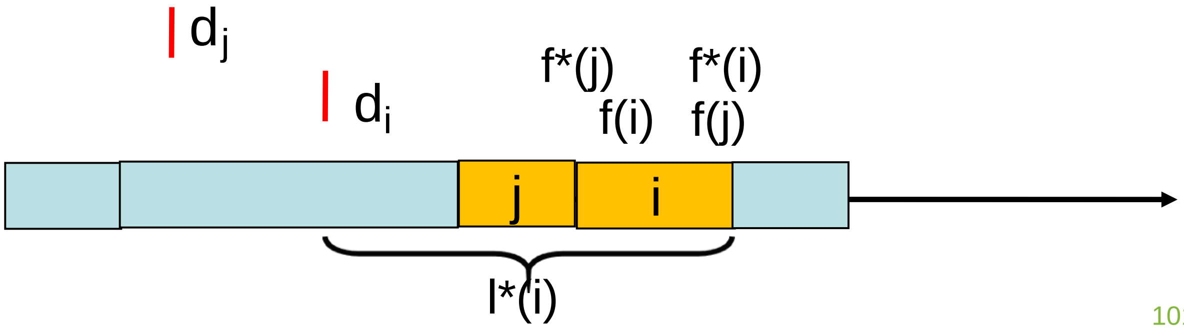
- Betrachte nur Aufgabe i: Nach dem Tauschen ist die Verzögerung $l^*(i) = f^*(i) - d_i$.



Gierige Algorithmen

Formaler Beweis von (c)

- Betrachte nur Aufgabe i: Nach dem Tauschen ist die Verzögerung $l^*(i) = f^*(i)-d_i$.
- Wegen $d_i > d_j$ folgt $l^*(i) = f(j) - d_i < f(j) - d_j = l(j)$.
- Damit wird die maximale Verzögerung nicht erhöht.



Gierige Algorithmen

Lemma 16

Es gibt eine optimale Lösung ohne Inversionen und Leerlaufzeit.

Beweis

- (a) Wenn O eine Inversion hat, dann gibt es ein Paar Aufgaben i und j, so dass j direkt nach i auftritt und $d_j < d_i$ ist.
- (b) Nach dem Austauschen von einer benachbarten Inversion i und j erhalten wir ein Schedule mit einer Inversion weniger.
- (c) Das Tauschen von i und j erhöht nicht die maximale Verzögerung.

- Die Anzahl Inversionen ist zu Beginn höchstens $\binom{n}{2}$.Wir können (a)-(c) solange anwenden, bis keine Inversionen mehr vorhanden sind.

Gierige Algorithmen

Satz 17

Die Lösung A, die von Algorithmus LatenessScheduling berechnet wird, hat optimale (d.h. minimale) maximale Verzögerung.

Beweis

Aus dem zweiten Lemma folgt, dass es ein optimales Schedule ohne Inversionen gibt. Aus dem ersten Lemma folgt, dass alle Schedules ohne Inversionen dieselbe maximale Verzögerung haben. Damit ist jedes Schedule ohne Inversionen optimal. Unser gieriger Algorithmus berechnet aber eine Lösung ohne Inversionen.

Gierige Algorithmen

Zusammenfassung

- Löse globales Optimierungsproblem durch lokale Optimierungsstrategie
- Liefert häufig recht einfache Algorithmen
- Funktioniert leider nicht immer und es ist manchmal nicht ganz einfach, die ‚richtige' Strategie zu finden

Algorithmische Entwurfsmethoden

- Teile & Herrsche
- Dynamische Programmierung
- Gierige Algorithmen