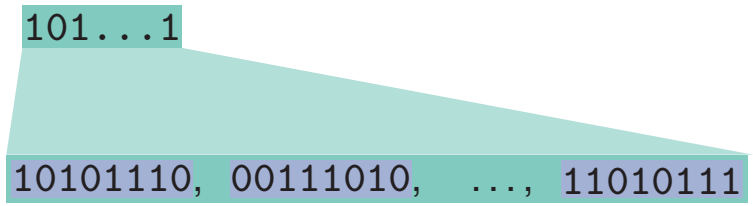
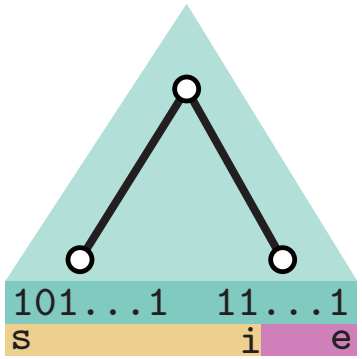
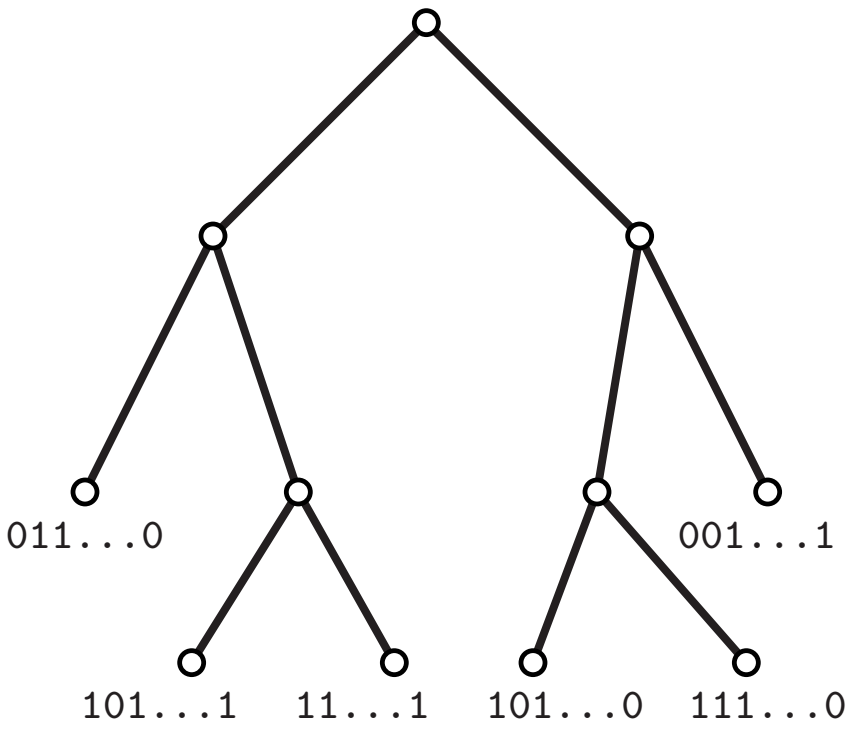
# Programmierprojekt
# Fortgeschrittene Datenstrukturen

Abschlusspräsentation · 25.07.2022
Moritz Potthoff



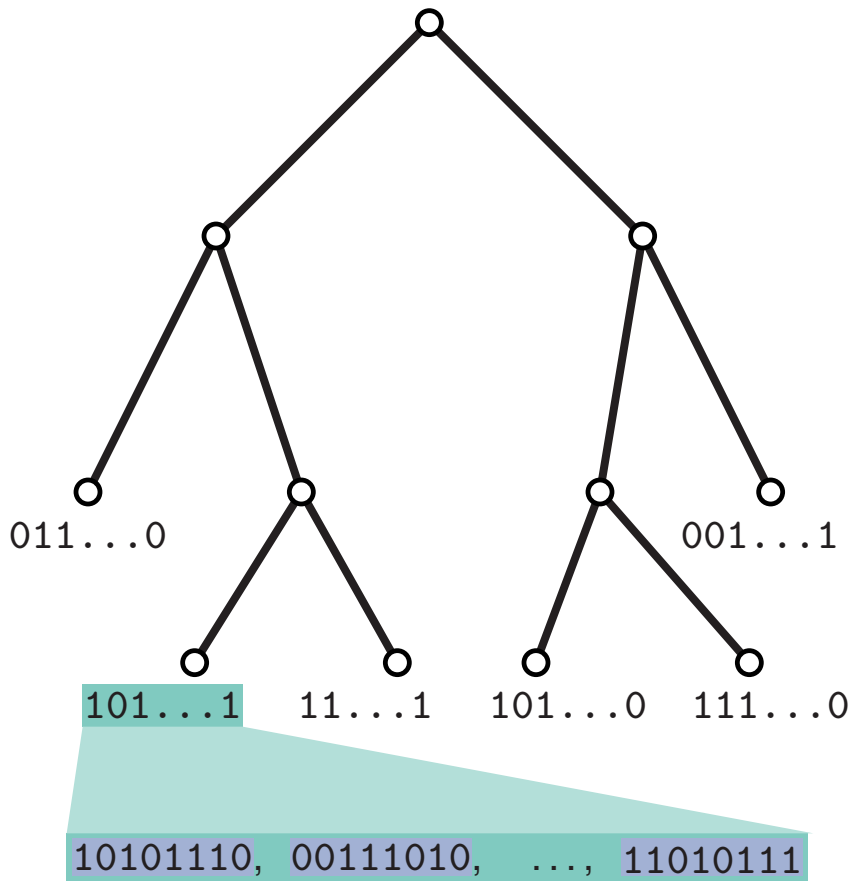



# BV: Platzeffizienter, schneller dynamischer Bitvektor



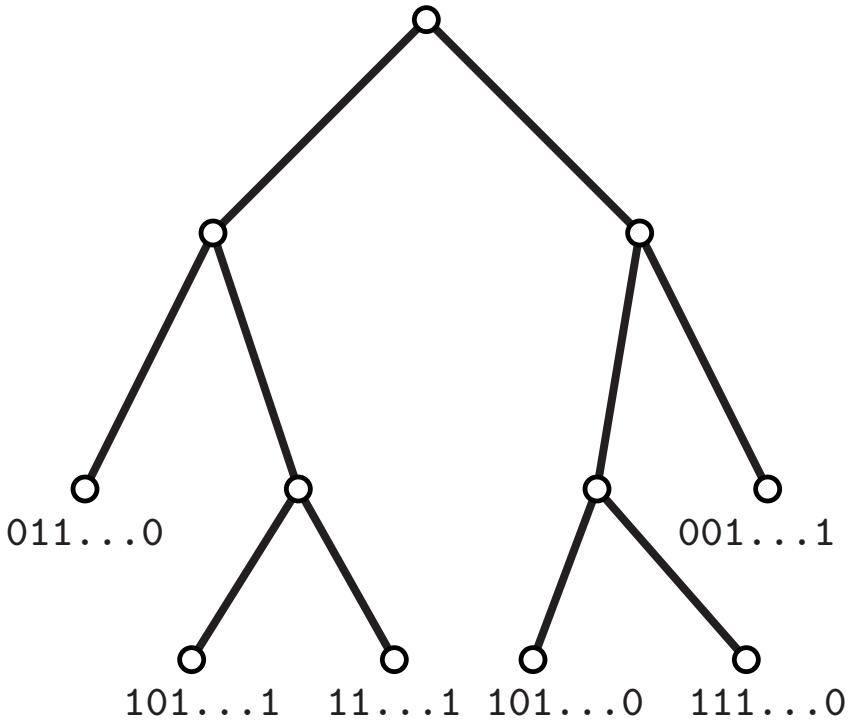

- Repräsentiere dynamischen Bitvektor als **AVL-Baum**
- Blätter speichern selbst dynamische Bitvektoren mit je $w^2/2$ bis $2w^2$ Bits
- Speichere Bits als Folge von `uint64_t`
  - Effizientere Anfragen durch popcount (verglichen mit `std::vector<bool>`, `boost::dynamic_bitset`)
  - Dynamische Größe, erlaube große Blätter ohne große Platzoverheads für leere Blätter (verglichen mit `std::bitset`)
- Tuning: 18 432 Bits für `bv`, 8 192 Bits für `bp`

# BV: Platzeffizienter, schneller dynamischer Bitvektor



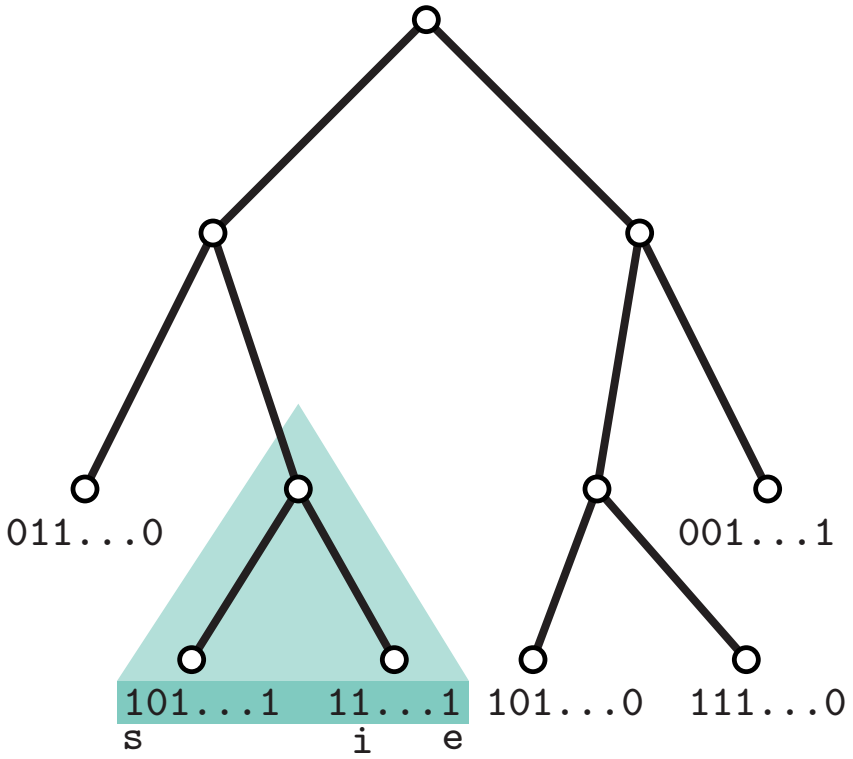

- Repräsentiere dynamischen Bitvektor als **AVL-Baum**
- Blätter speichern selbst dynamische Bitvektoren mit je $w^2/2$ bis $2w^2$ Bits
- Speichere Bits als Folge von `uint64_t`
  - Effizientere Anfragen durch popcount (verglichen mit `std::vector<bool>`, `boost::dynamic_bitset`)
  - Dynamische Größe, erlaube große Blätter ohne große Platzoverheads für leere Blätter (verglichen mit `std::bitset`)
- Tuning: 18 432 Bits für `bv`, 8 192 Bits für `bp`

# BP: Pruning der Backward Search



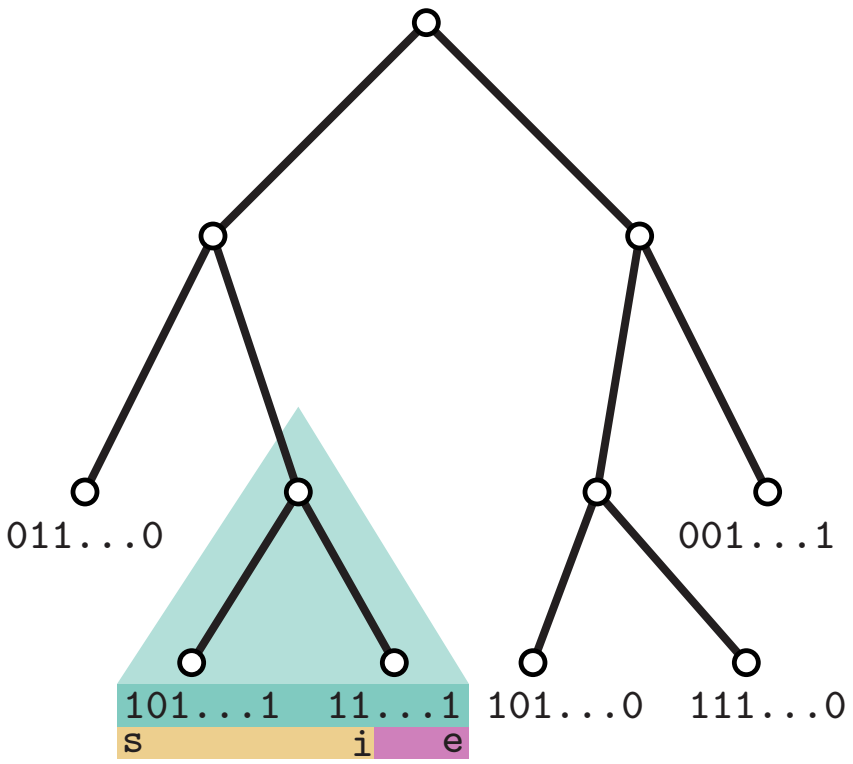

- Backward search berücksichtigt Excess rückwärts
- Forward search prunt mit minimalem vorwarts-Excess
- Anstatt maximalen rückwärts-Excess zu speichern:
  - `excess(s, i) + excess(i+1, e) = excess(s, e) = totalExcess`
  - `min fwd excess(s, e)` at `i` $\rightarrow$ `max bwd excess(s, e)` at `i+1`
- Prune somit auch backward search mit minimalem vorwärts-Excess

# BP: Pruning der Backward Search




- Backward search berücksichtigt Excess rückwärts
- Forward search prunt mit minimalem vorwarts-Excess
- Anstatt maximalen rückwärts-Excess zu speichern:
  - `excess(s, i) + excess(i+1, e) = excess(s, e) = totalExcess`
  - `min fwd excess(s, e)` at `i` $\rightarrow$ `max bwd excess(s, e)` at `i+1`
- Prune somit auch backward search mit minimalem vorwärts-Excess

# BP: Pruning der Backward Search

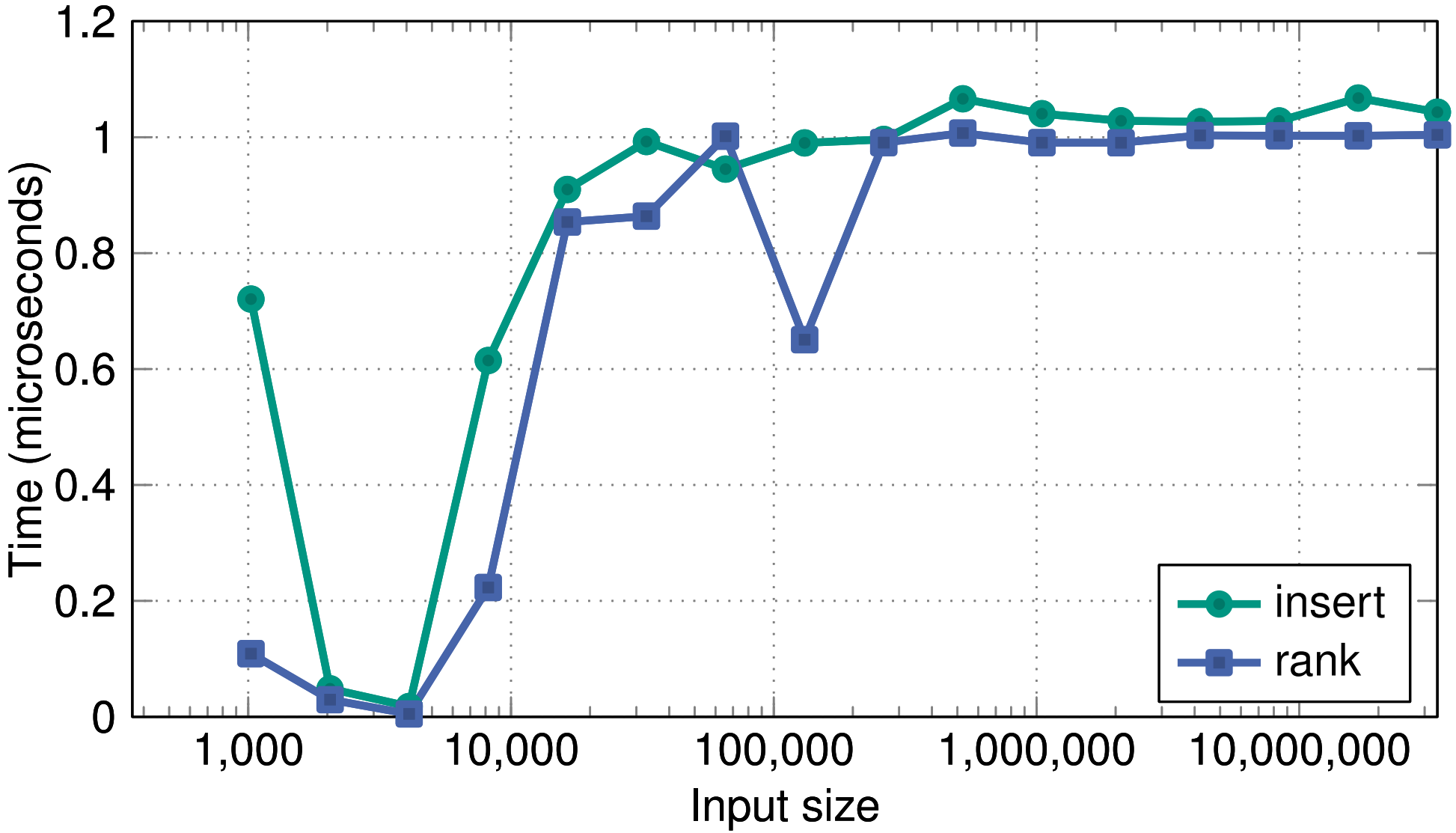



- Backward search berücksichtigt Excess rückwärts
- Forward search prunt mit minimalem vorwarts-Excess
- Anstatt maximalen rückwärts-Excess zu speichern:
  - `excess(s, i)` + `excess(i+1, e)` = `excess(s, e)` = `totalExcess`
  - `min fwd excess(s, e)` at `i` $\rightarrow$ `max bwd excess(s, e)` at `i+1`
- Prune somit auch backward search mit minimalem vorwärts-Excess

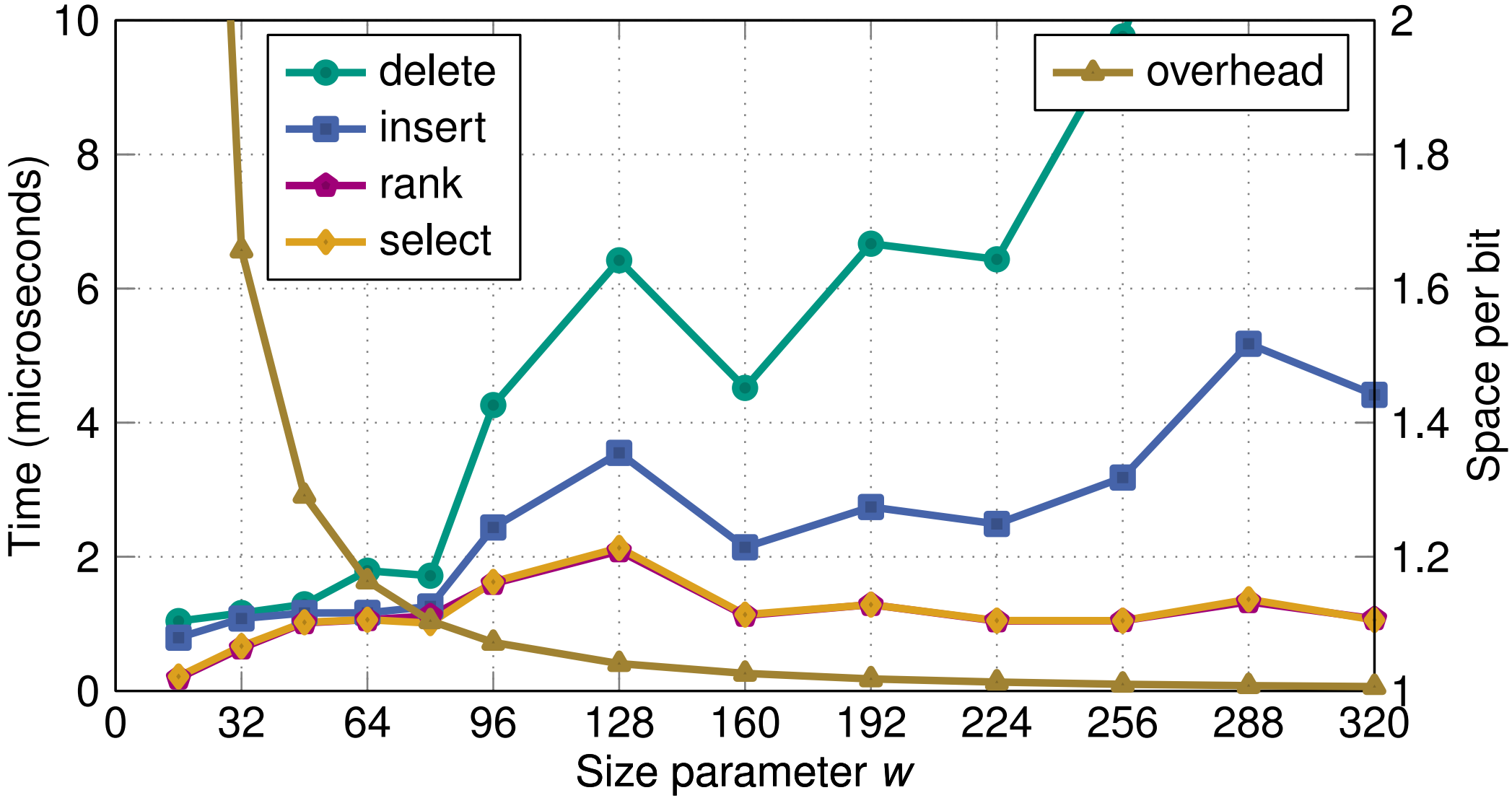
# Evaluation BV

Keine Zeit für BP

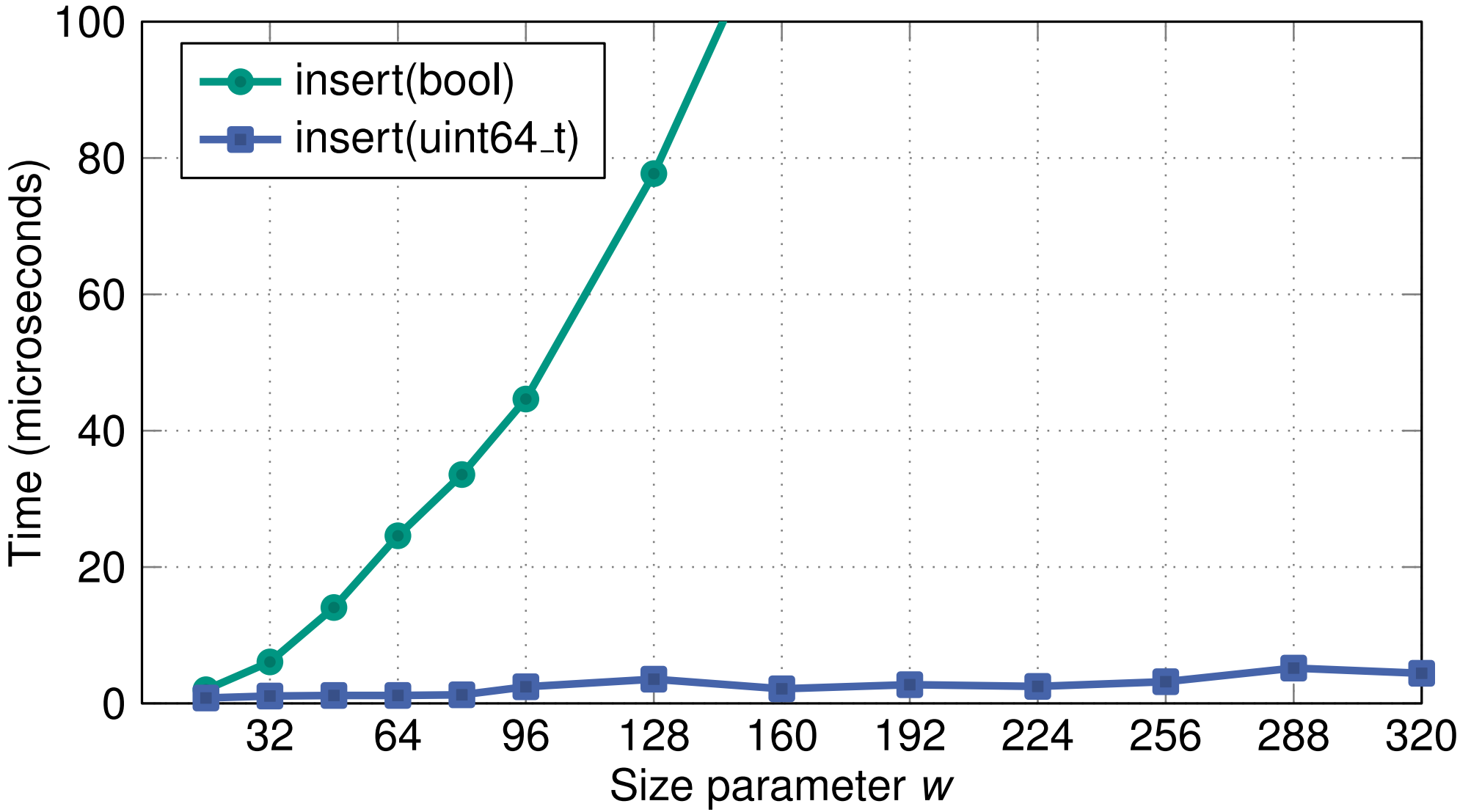
# Laufzeiten für Bitvektoren



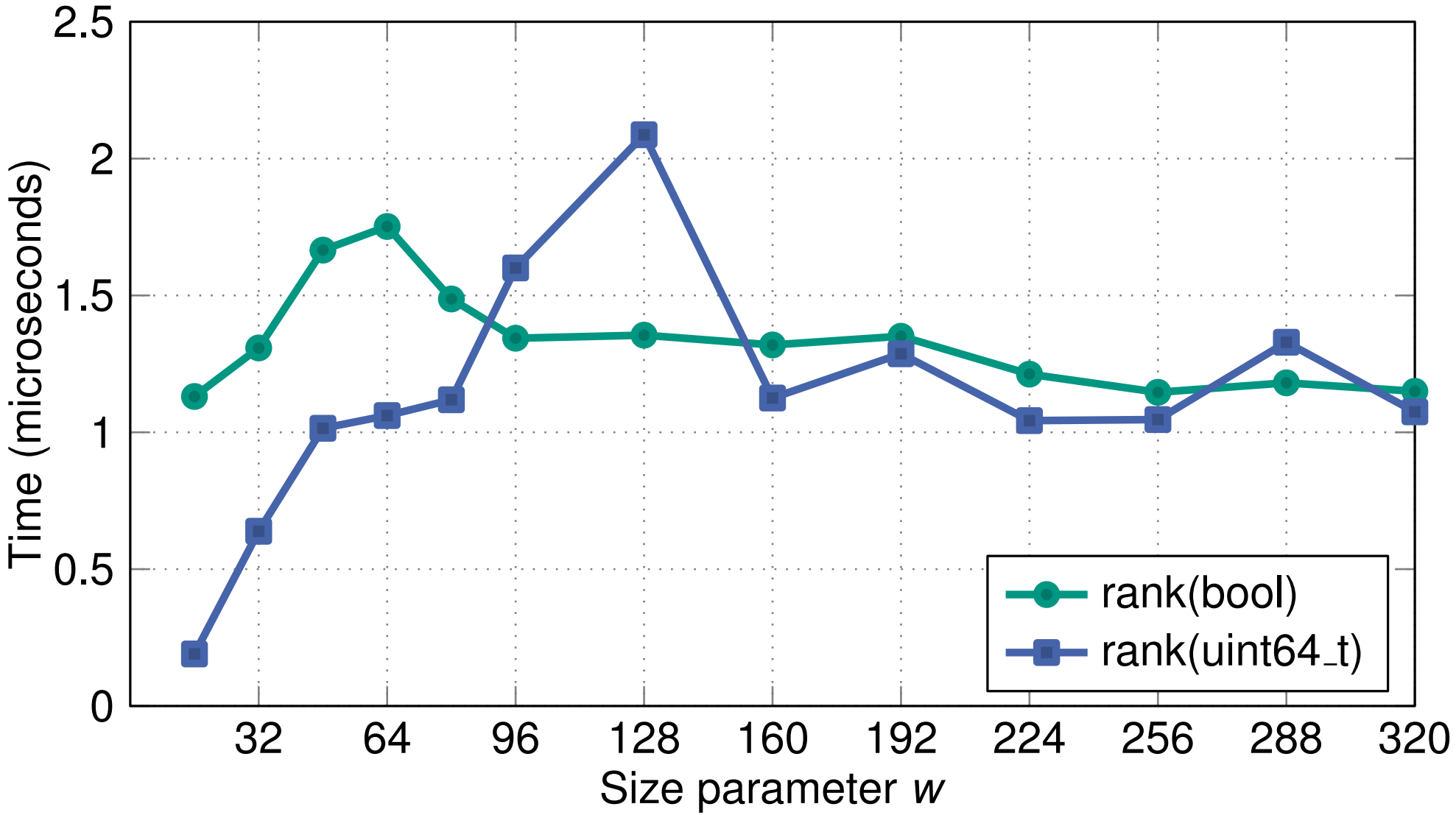
# Bitvektor Performance nach Blattgröße



# Schnellere Inserts durch `uint64_t`-basierte BV



# (Leicht) schnellere Queries durch `uint64_t`-basierte BV



# Danke!