Waves of misery after index creation

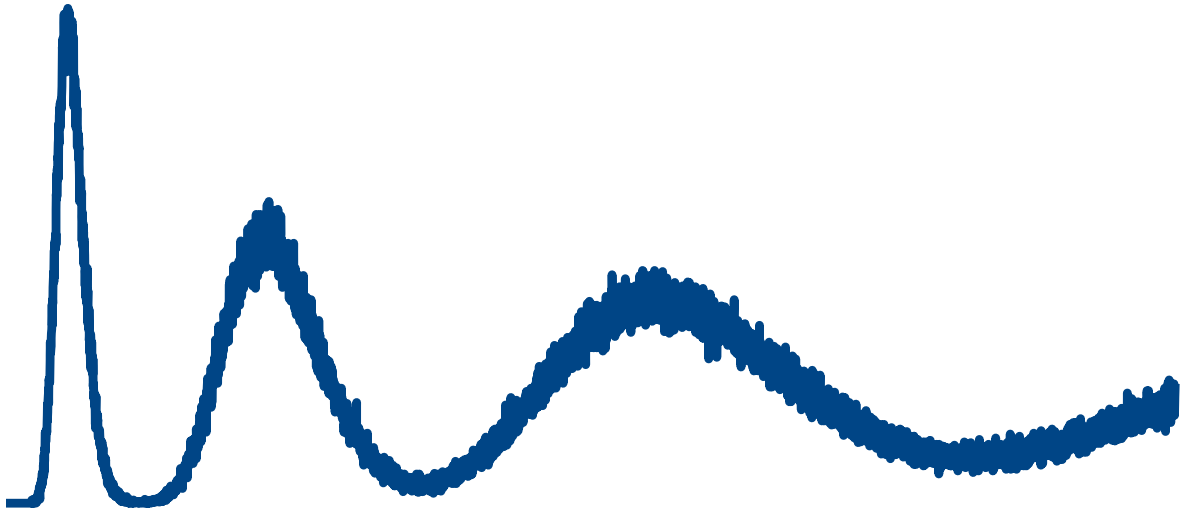
Nikolaus Glombiewski[1], Bernhard Seeger[1], Goetz Graefe[2]

[1]University of Marburg

[2]Google Inc.

Outline

- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion



Outline

- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

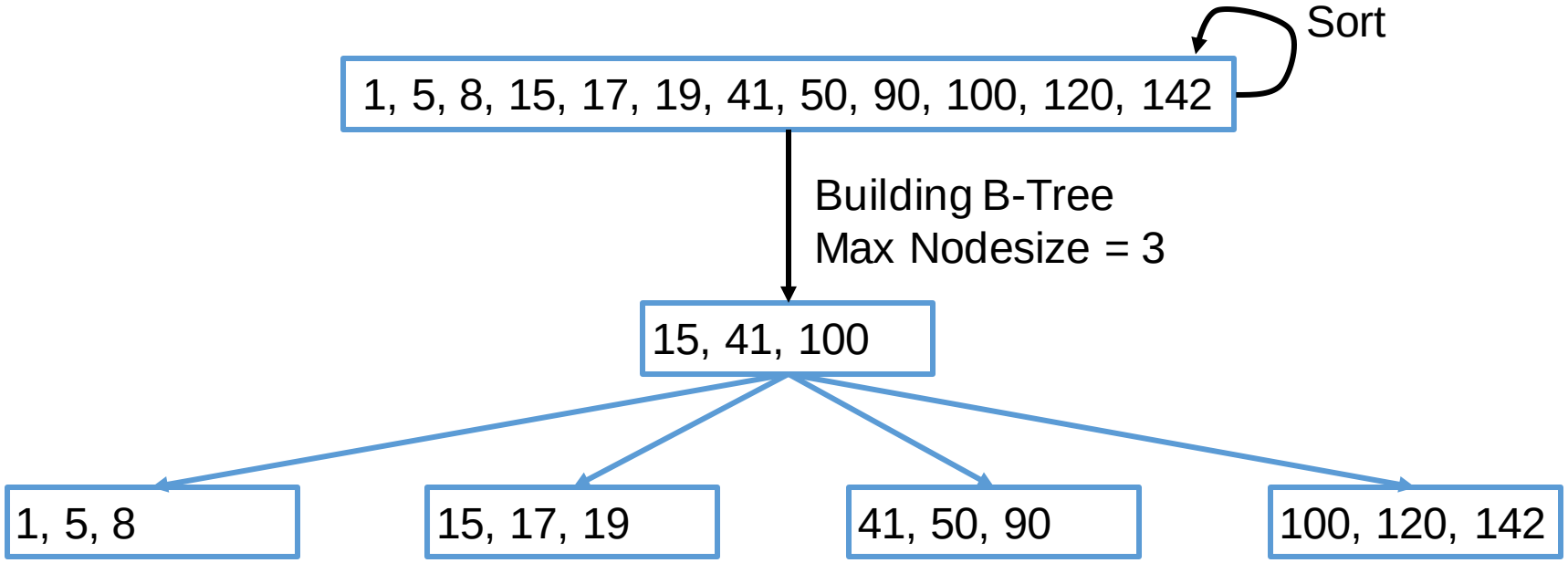
Indexes: Pros & Cons

- Pros
  - Fast lookups
  - Fast ordered range scans

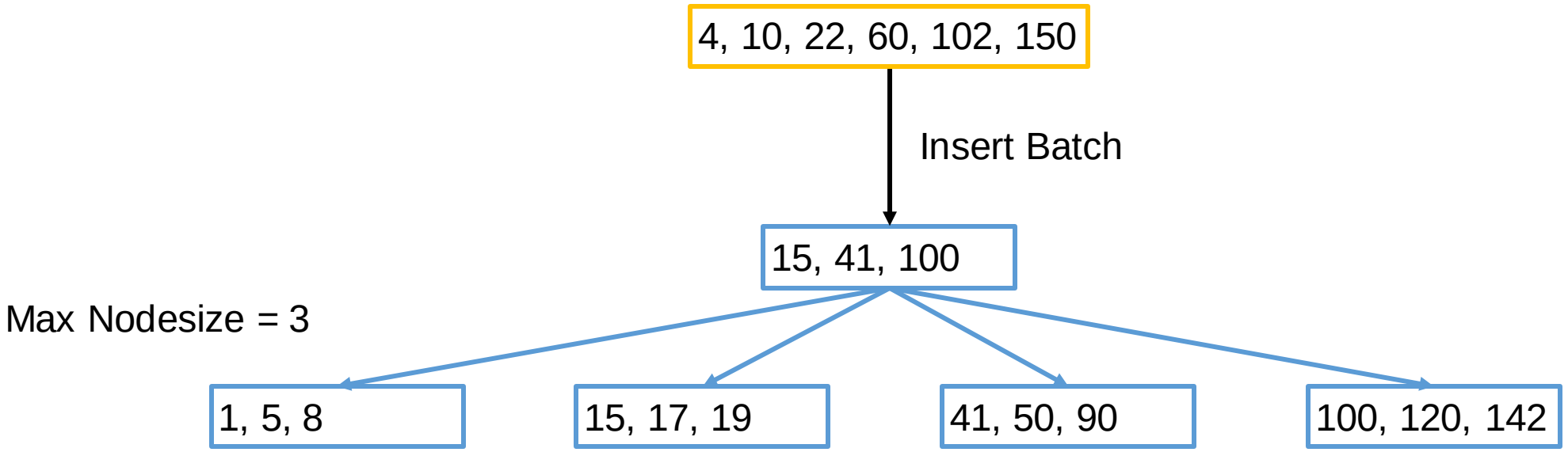
  ➔ Best supported by bulk loading a perfect secondary b-tree

- Cons
  - Maintenance cost
  - Robustness of *performance* over time

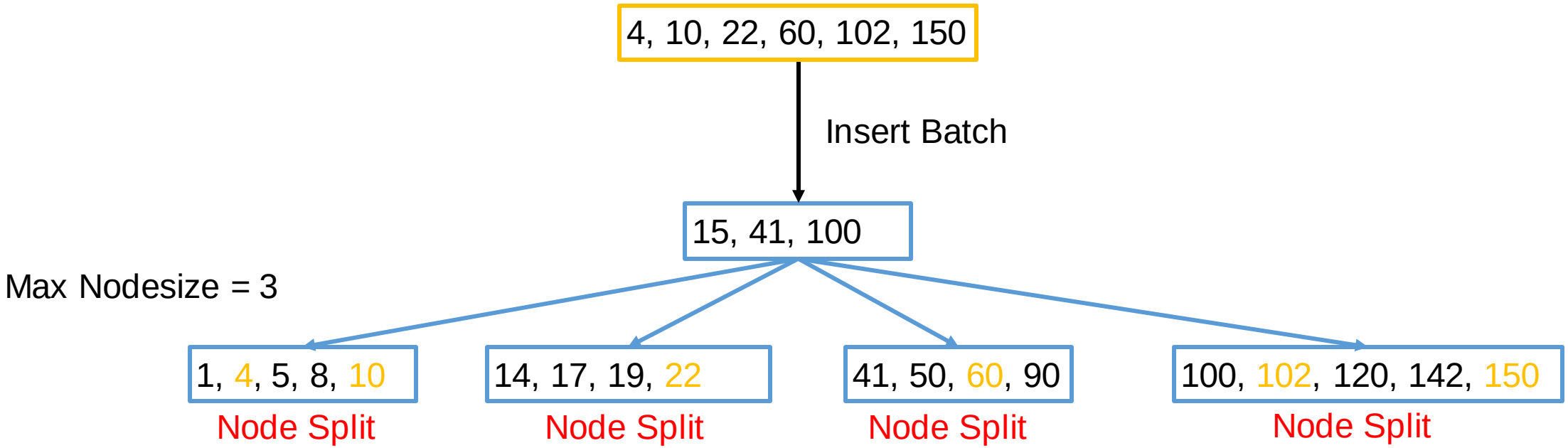
Creation of a Perfect B-tree



Subsequent Insertions on a Perfect B-tree



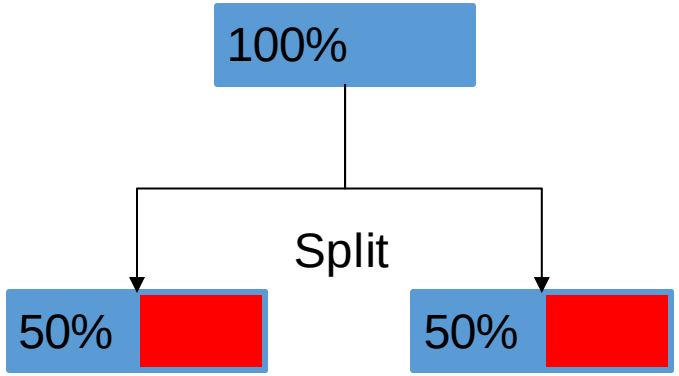
Subsequent Insertions on a Perfect B-tree



=> *Immediate*, widespread node splits after index creation

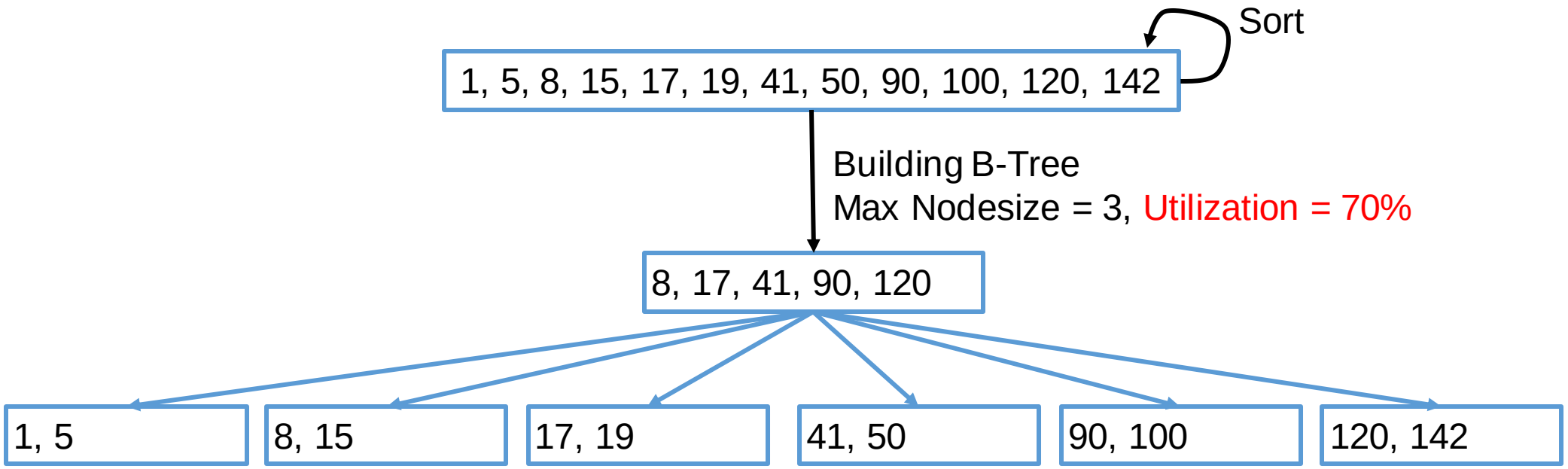
Problem of Subsequent Insertions

- Splits of almost all leaves within a short time period
  - high I/O load
  - low buffer utilization
  - low query performance due to contention

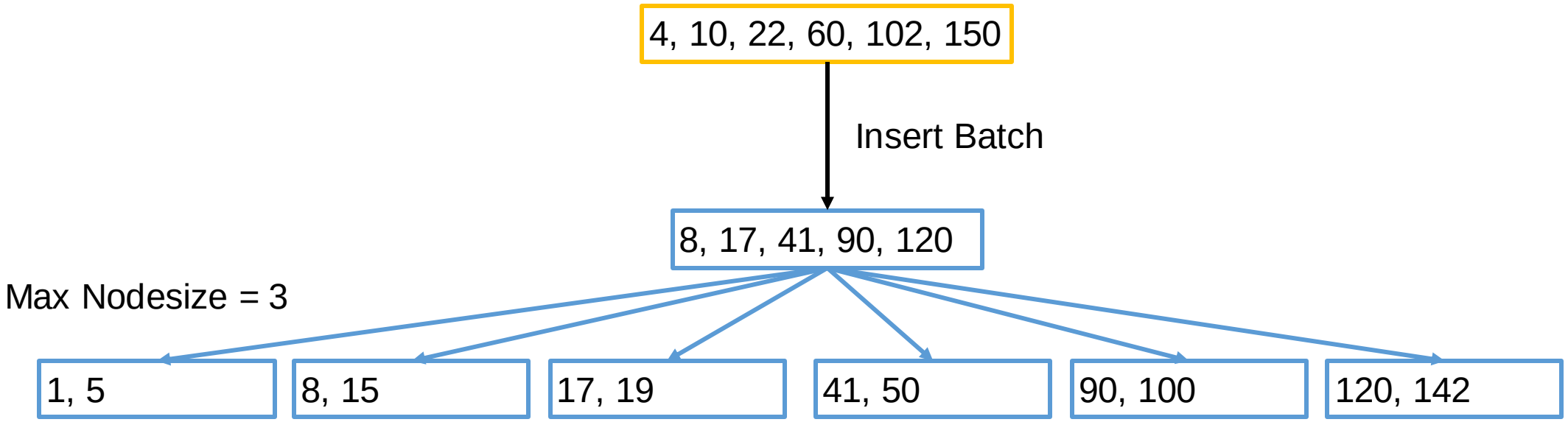


- Status quo database solution: Leave free space (e.g. 30%)
  - Oracle, SQL Server, DB2, …

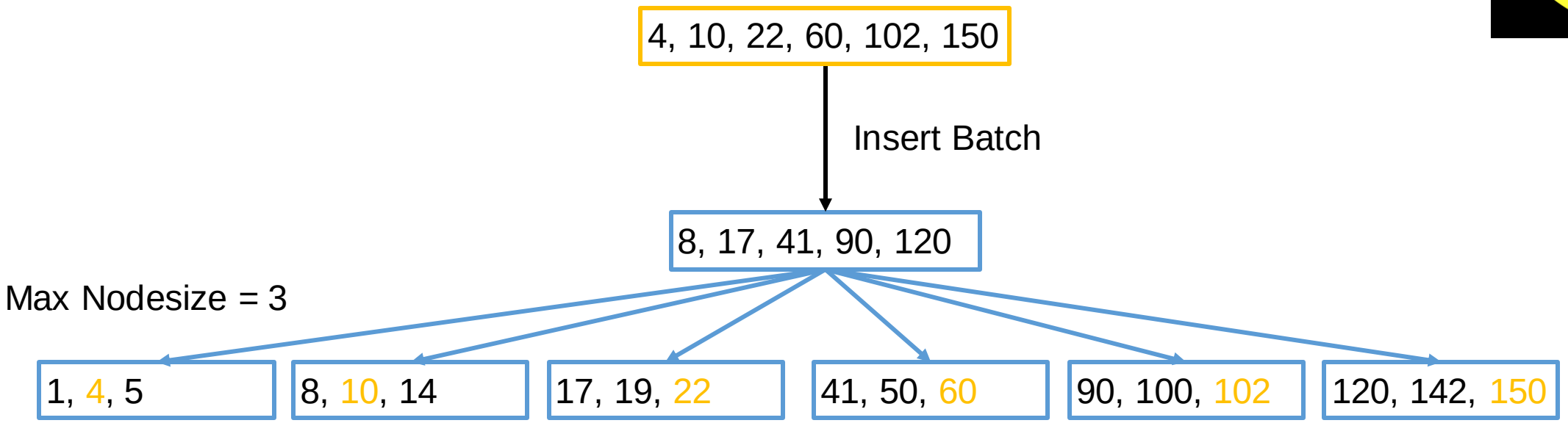
Creation of a Perfect B-tree with free space



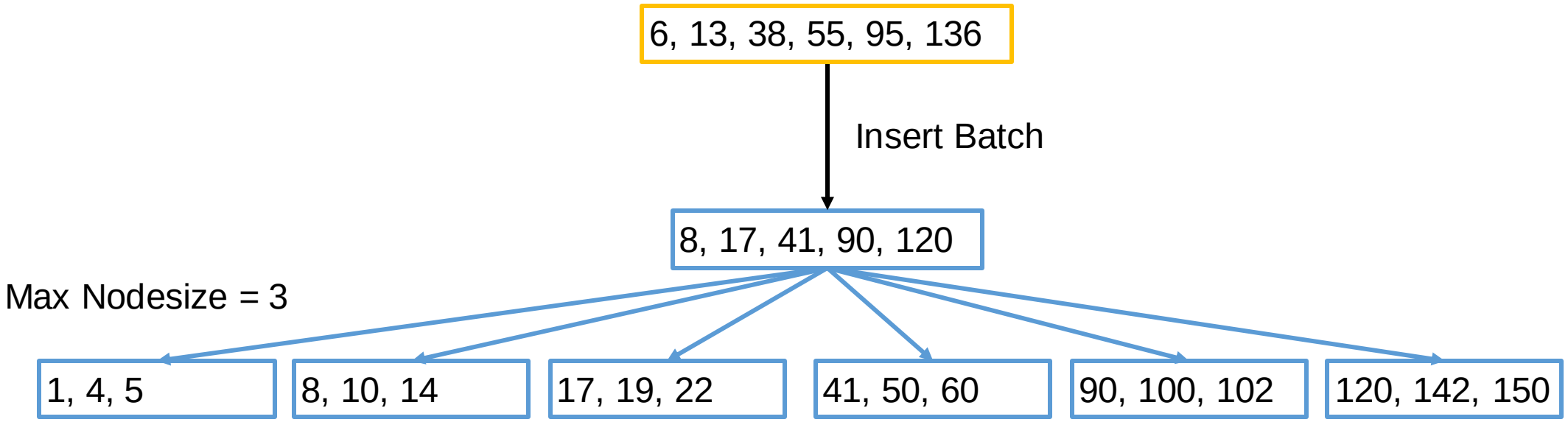
Subsequent insertions



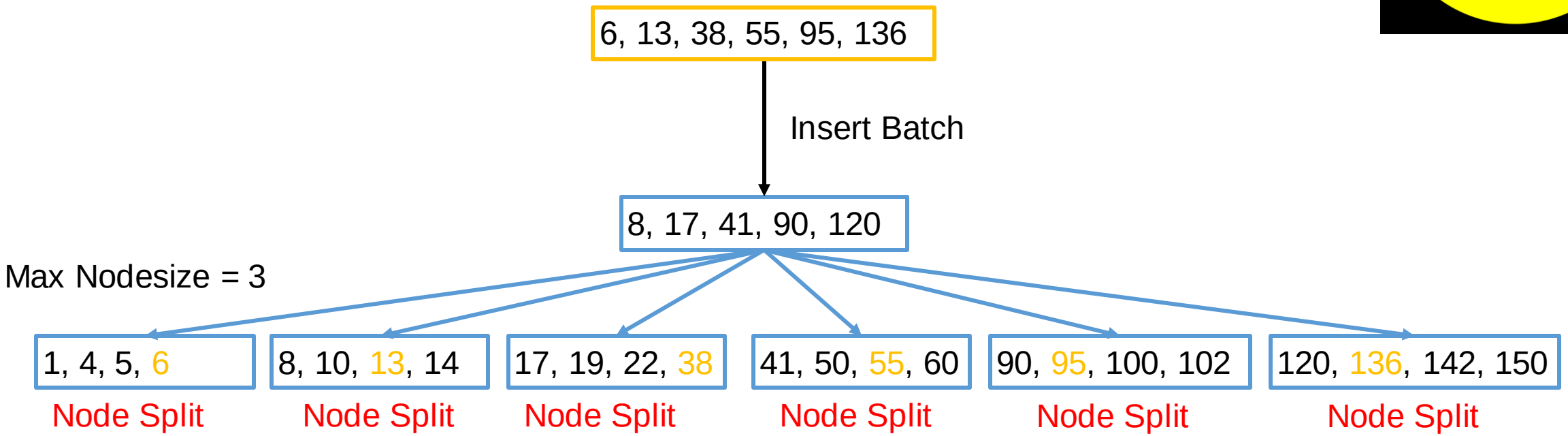
Subsequent insertions



Continuation of insertions



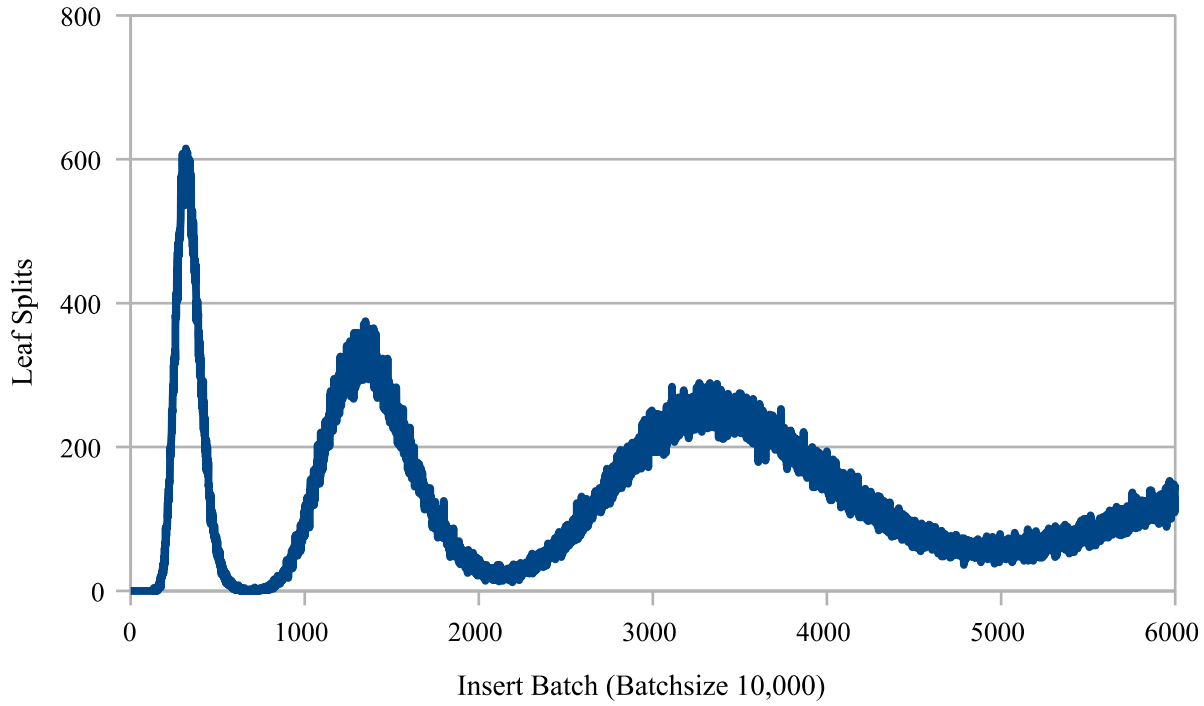
Continuation of insertions



=> *Delayed*, widespread node splits after index creation

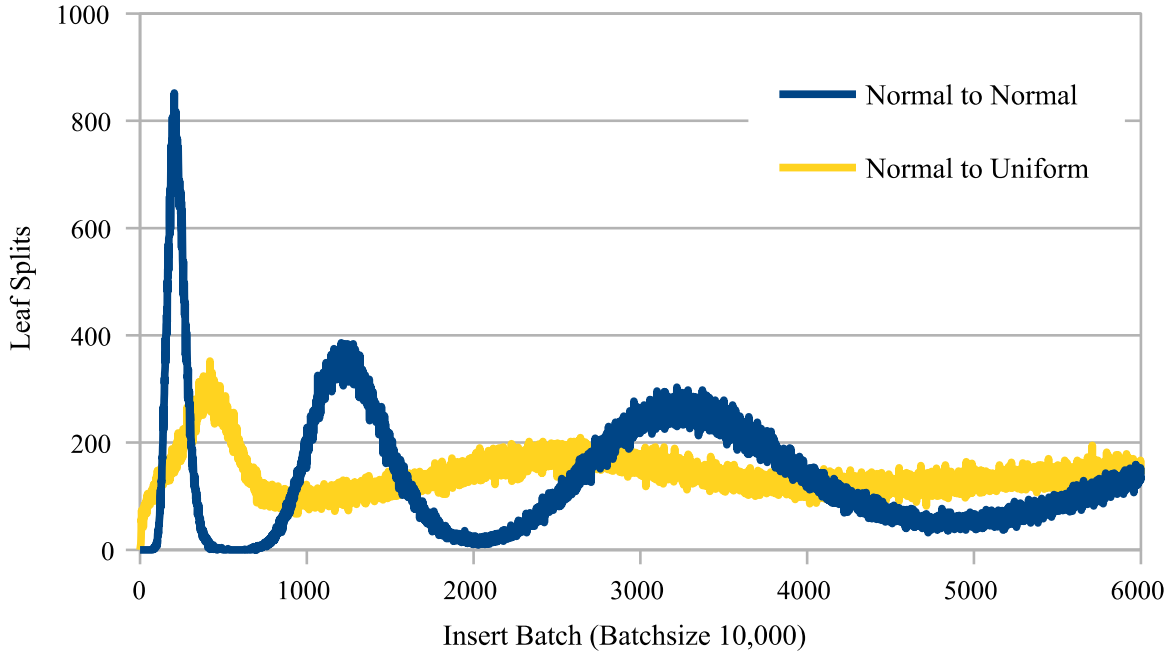
Limitations of the status quo

- The problem of splits is merely delayed
- Moreover, the problem occurs in *waves*



Problem Assessment – When does it occur?

- Loading Distribution = Insert Distribution
  - E.g.: Hash-Keys

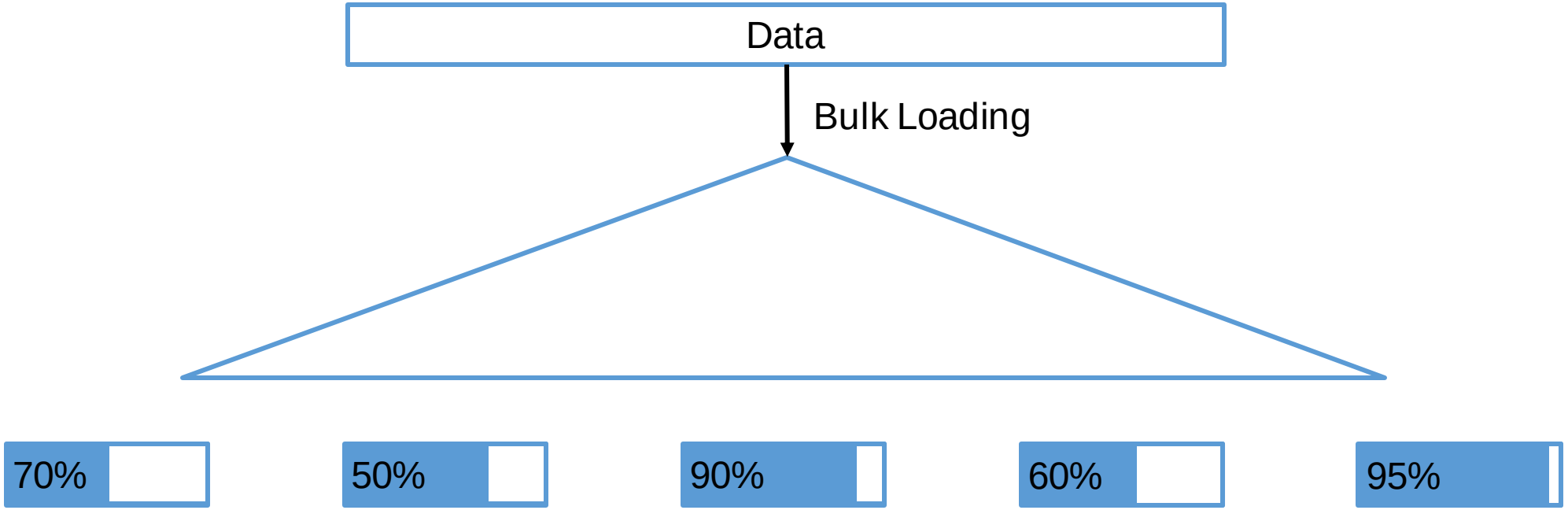


Outline

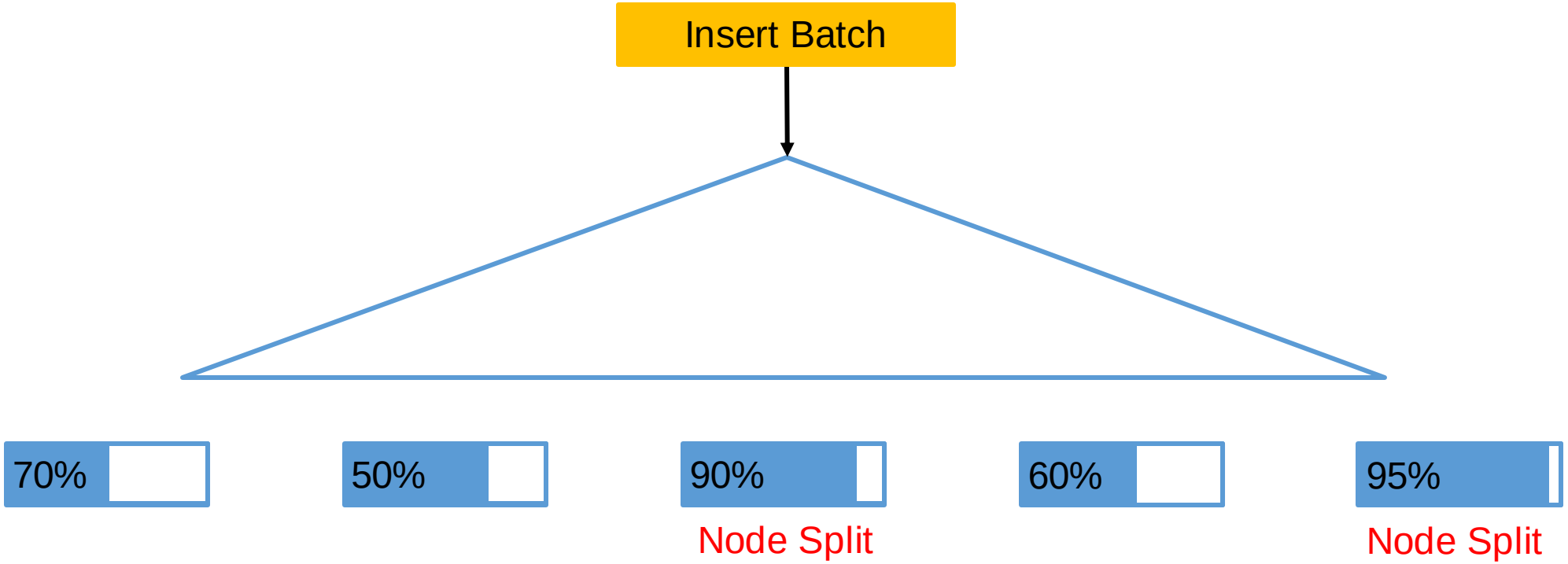
- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

Basic Idea

- Do not leave constant free space while loading



Basic Idea – Insert Batch

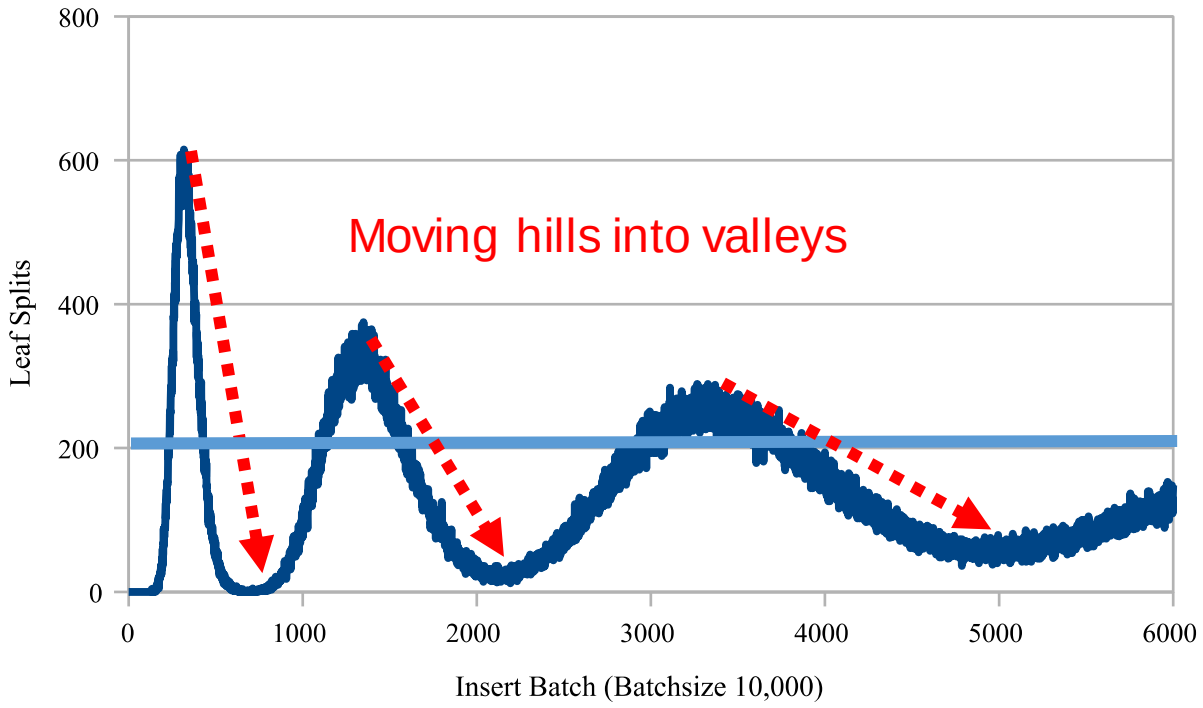


=> *Distributing* node splits over time

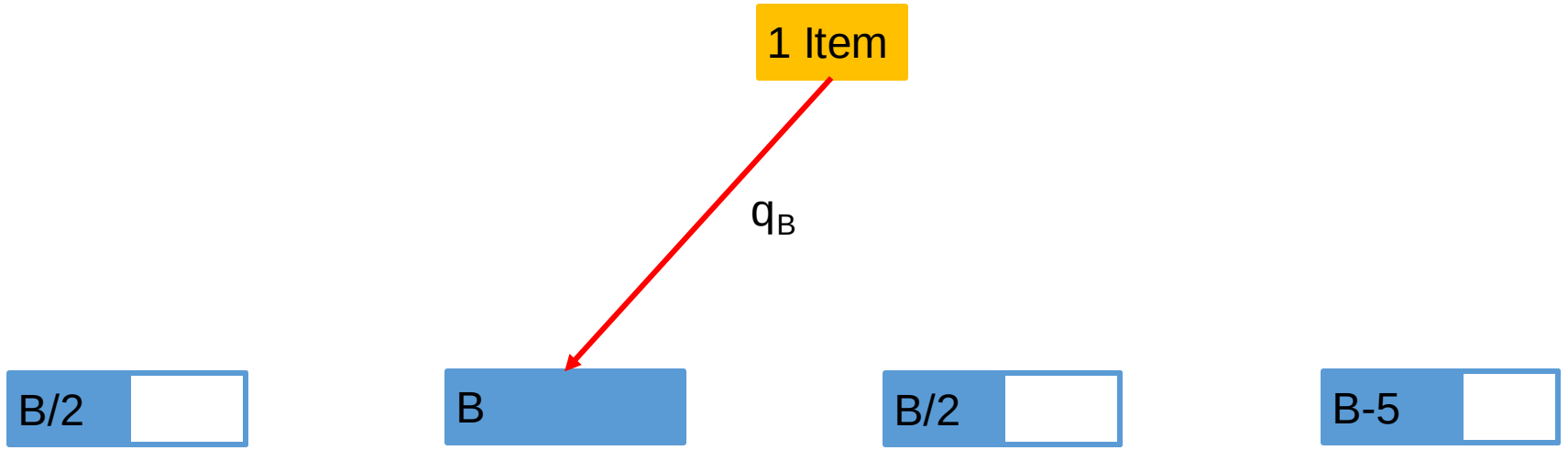
Outline

- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

Ideal solution for predictable splits



Ideal solution (Leaf Nodes)



- q_B = Probability of a split after insertion

Ideal solution (Leaf Nodes)

- Fringe Analysis: $\begin{pmatrix} q_{\frac{B}{2}} \\ \dots \\ q_B \end{pmatrix} = \vec{q}(n)$

- Insert-Operation: $\vec{q}(n) * \left(I + \frac{1}{n+1} T\right) = \vec{q}(n+1)$

- Goal: $\vec{q}(n) = \vec{q}(n+k)$ => A stable state

Ideal solution (Leaf Nodes)

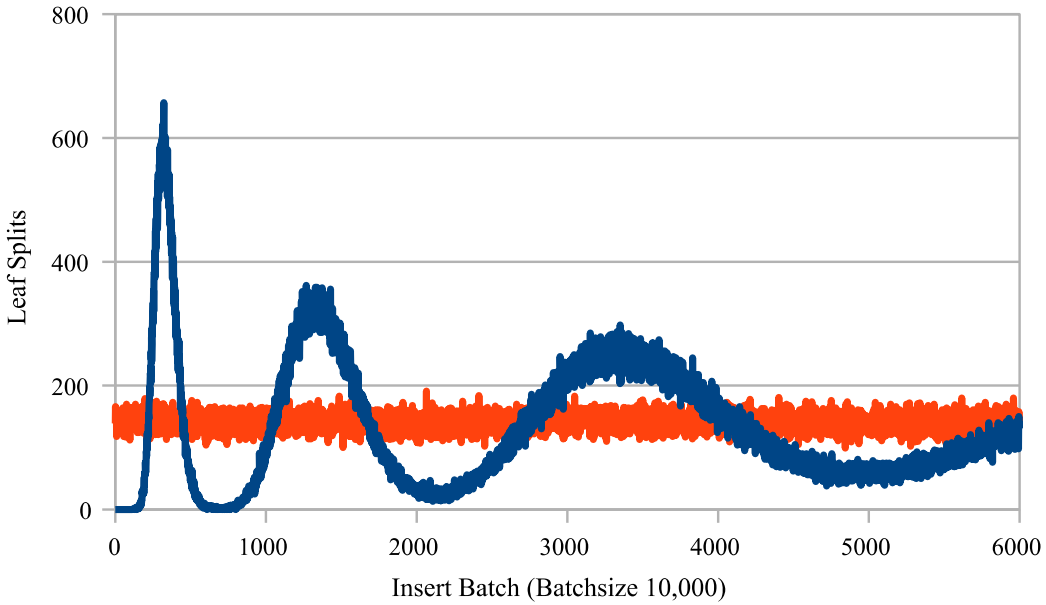
- Goal: $\vec{q}(n) = \vec{q}(n + k)$ => A stable state

- Analyze Transition: $T * \begin{pmatrix} q_B \\ \frac{}{2} \\ \dots \\ q_B \end{pmatrix} = \vec{0}$

- Formula holds for $q_j = 1/(j+1)$

Ideal solution (Leaf Nodes)

- Intuition:
  - Few full pages split *immediately*
  - Many half full pages *eventually*



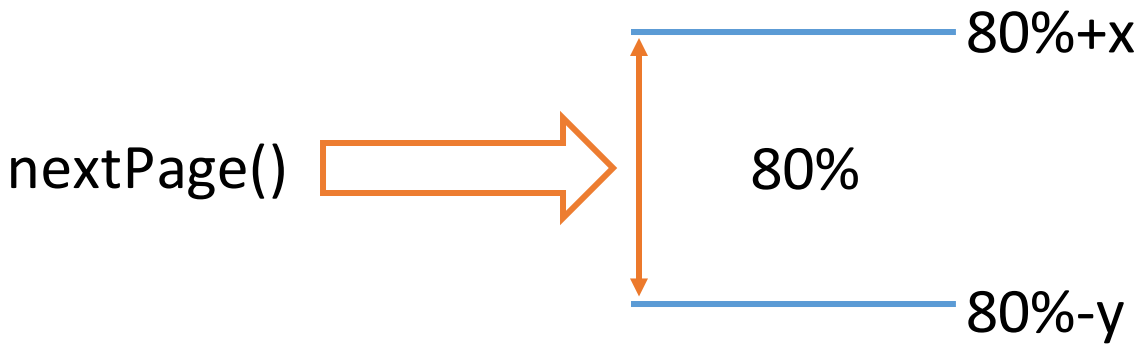
- Ideal solution 
  - …for expected B-tree utilization
  - …i.e., for Utilization of ln(2) = 69% 

Outline

- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

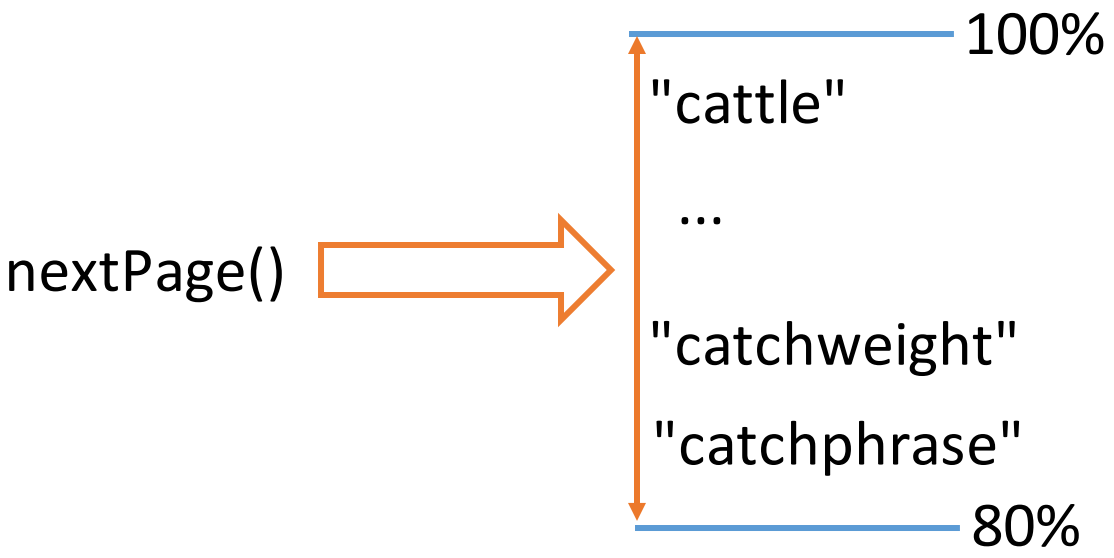
Practical Remedies – Random

- While loading: Randomly pick around target utilization



Practical Remedies – Suffix Truncation

- While loading: Search for shortest key within range



- Added compression effect

Outline

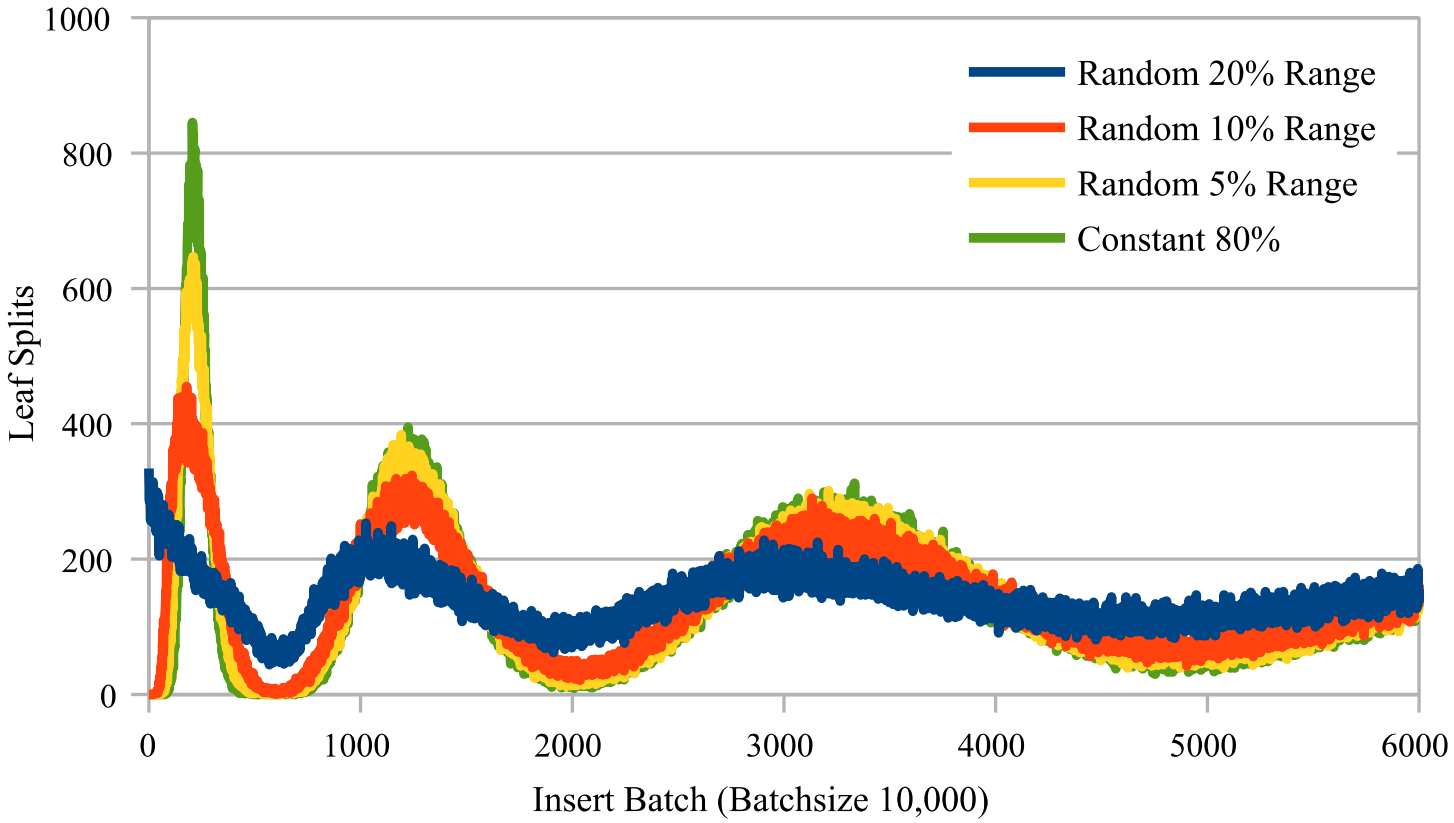
- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

Experimental Evaluation – Setup

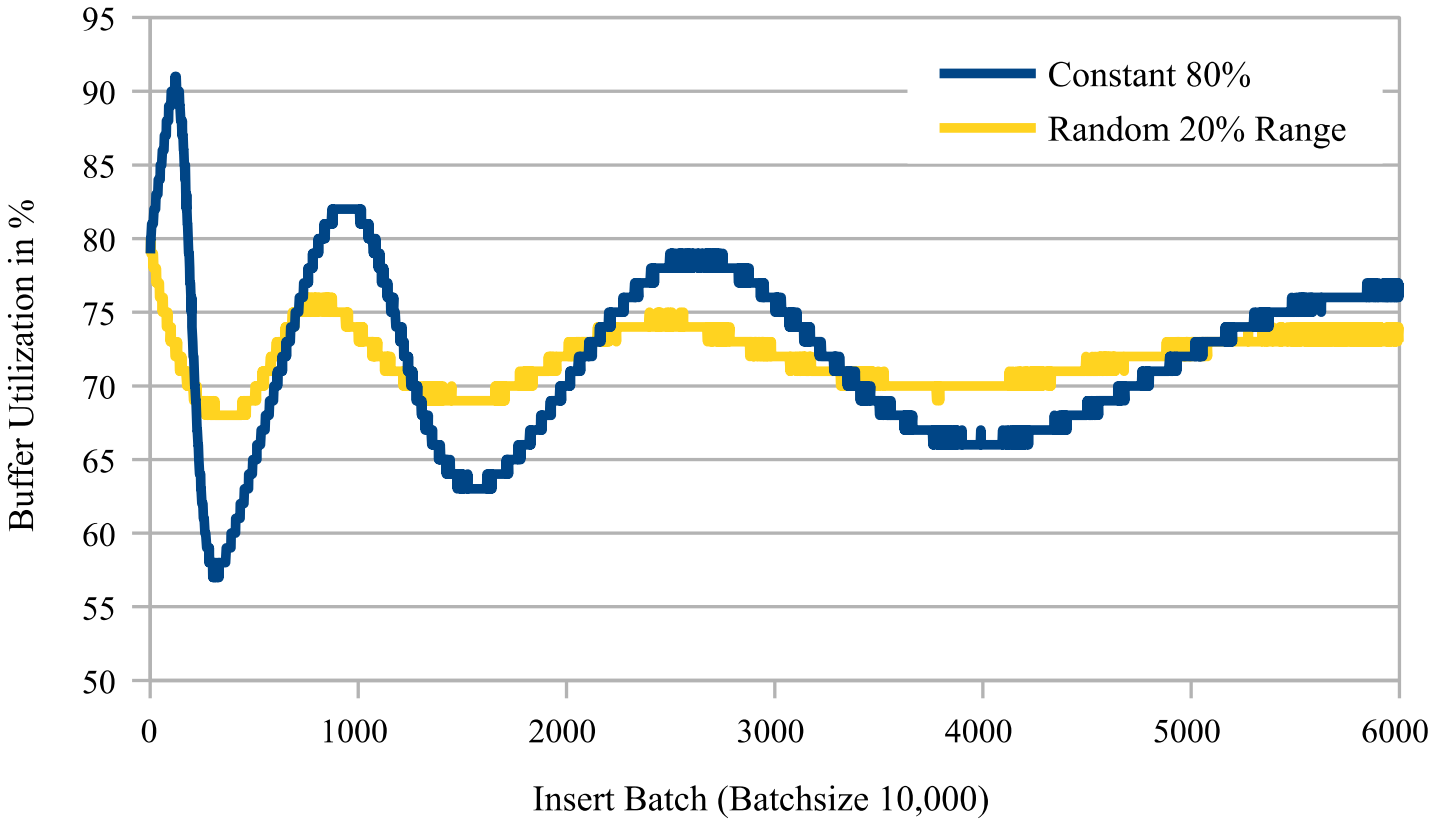
- Procedure:
  - Records: 21 integers (84 bytes), normal distribution
  - Loading b-tree with 100,000 pages of 8KB
  - Inserting batches of 10,000 records

- Workstation:
  - AMD Ryzen7 2700X
  - 16GB memory
  - Java indexing library XXL

Experimental Evaluation – Random



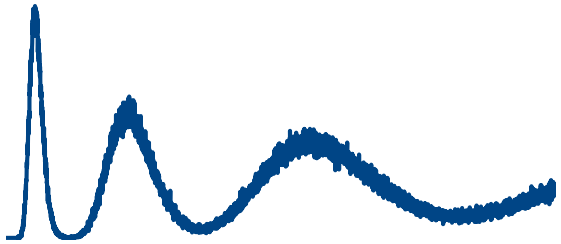
Experimental Evaluation – Buffer Utilization



Outline

- Problem Assessment
- Basic Solution
- Ideal Solution
- Practical Remedies
- Experimental Evaluation
- Conclusion

Waves of Misery after index creation



- Loading secondary b-tree index in…
  - Write-intensive workloads
  - Loading distribution = Insert distribution
- Want to achieve predictable split performance:

Don't just leave *constant* free space in your tree nodes!

Work towards starting in the steady state of the b-tree.

Thank you for your attention!