**Practical Performance Target (circa 1992)**
- Sort one billion large keys in one minute on one thousand processors.
- Good sort on a workstation can do 1 million keys in about 10 seconds
  - just fits in memory
  - 16 bit Radix Sort
- Performance unit: \( \mu s \) per key per processor
  - \( s \approx 10 \) for single Sparc 2

---

**Performance Targets in 2001 (Jim Gray)**
- Datamation (1 April, 1985)
  - Sort a million hundred-byte records
  - tests file system, I/O system, utility access
- Minute Sort:
  - Sort as many records as you can in a minute
  - Report rate and price (\$/cost/1e6)
  - Indy-MinuteSort: a Formula-1 sort where price is no object.
  - Daytona-MinuteSort: a stock sort where price is no object.
- Penny Sort:
  - Sort as much as you can for a penny.
  - Proposed Change to PennySort: Performance/Price Sort
  - Compute GB/$ of a two pass sort: performance per dollar.
  - Indy-PennySort: a Formula-1 biggest-bang-for-the buck sort.
  - Daytona-PennySort: a stock sort giving the biggest bang-for-the buck

---

**Datamation Results**
- Was one hour, now less \( \frac{1}{2} \) second
- Benchmark retired in 2001

---

**2004 Results**

<table>
<thead>
<tr>
<th></th>
<th>Daytona</th>
<th>Indy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Penny</td>
<td>10GB, 105M recs</td>
<td>42 GB, 433 M rec</td>
</tr>
<tr>
<td></td>
<td>1198 secs on a 3857 Linux/AMD</td>
<td>1541 seconds on 6145 Linux/AMD</td>
</tr>
<tr>
<td>Minute</td>
<td>34 B records</td>
<td>Same as Daytona</td>
</tr>
<tr>
<td></td>
<td>32 x Itanium2 NEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 B records</td>
<td>Ordinal Tech.</td>
</tr>
<tr>
<td></td>
<td>Tandem</td>
<td></td>
</tr>
<tr>
<td>Terabyte</td>
<td>48 min</td>
<td>1057 secs</td>
</tr>
<tr>
<td></td>
<td>68x2 Compaq</td>
<td>5P</td>
</tr>
<tr>
<td></td>
<td>Tandem</td>
<td>cluster 2168</td>
</tr>
<tr>
<td></td>
<td>Sandia Labs</td>
<td>disks</td>
</tr>
<tr>
<td></td>
<td>Jim Wyllie</td>
<td></td>
</tr>
</tbody>
</table>

---

**Local Sort Performance**

(11 bit radix sort of 32 bit numbers)

- \( \text{Entropy in Key Values} \)
- \( \text{Entropy} = \sum p_i \log_2 p_i \)
- \( p_i = \text{Probability of key } i \)
**Bitonic Sort**

- A bitonic sequence is one that is:
  1. Monotonically increasing and then monotonically decreasing
  2. Or can be circularly shifted to satisfy 1

- A half-cleaner takes a bitonic sequence and produces
  1. First half is smaller than smallest element in 2nd
  2. Both halves are bitonic

- A full sort is a series of half-cleaners starting with depth 1 and increasing to depth of (log n)

**Column Sort**

1. Sort
2. Transpose - block to cyclic
3. Sort
4. Transpose - cyclic to block w/o scatter
5. Sort
6. Shift
7. Merge
8. Unshift

Work efficient

**Bitonic Sort**

1. Sort
2. Transpose - block to cyclic
3. Sort
4. Transpose - cyclic to block
5. Sort
6. Shift
7. Merge
8. Unshift

**Histo-radix sort**

1. Compute local histogram
2. Compute position of 1st member of each bucket in global array
   - \(2^r\) scans with end-around
3. Distribute all keys

Only \(r = 8, 11, 16\) make sense for sorting 32 bit numbers

**Histo-Radix Sort**

- Local Data
- Local Histograms

Each Pass
- Form local histograms
- Form global histogram
- Globally distribute data
Radix: Breakdown

Radix: Stream Broadcast Problem

Sample Sort

1. compute P-1 values of keys that would split the input into roughly equal pieces.
   - take S~64 samples per processor
   - sort P S keys
   - take key S, 2S, . . . (P-1)S
   - broadcast splitters

2. Distribute keys based on splitters

3. Local sort

[4.] possibly reshift

Sample Breakdown

Comparison

- Good fit between predicted and measured (10%)
- Different sorts for different sorts
- scaling by processor, input size, sensitivity
- All are global / local hybrids
- the local part is hard to implement and model