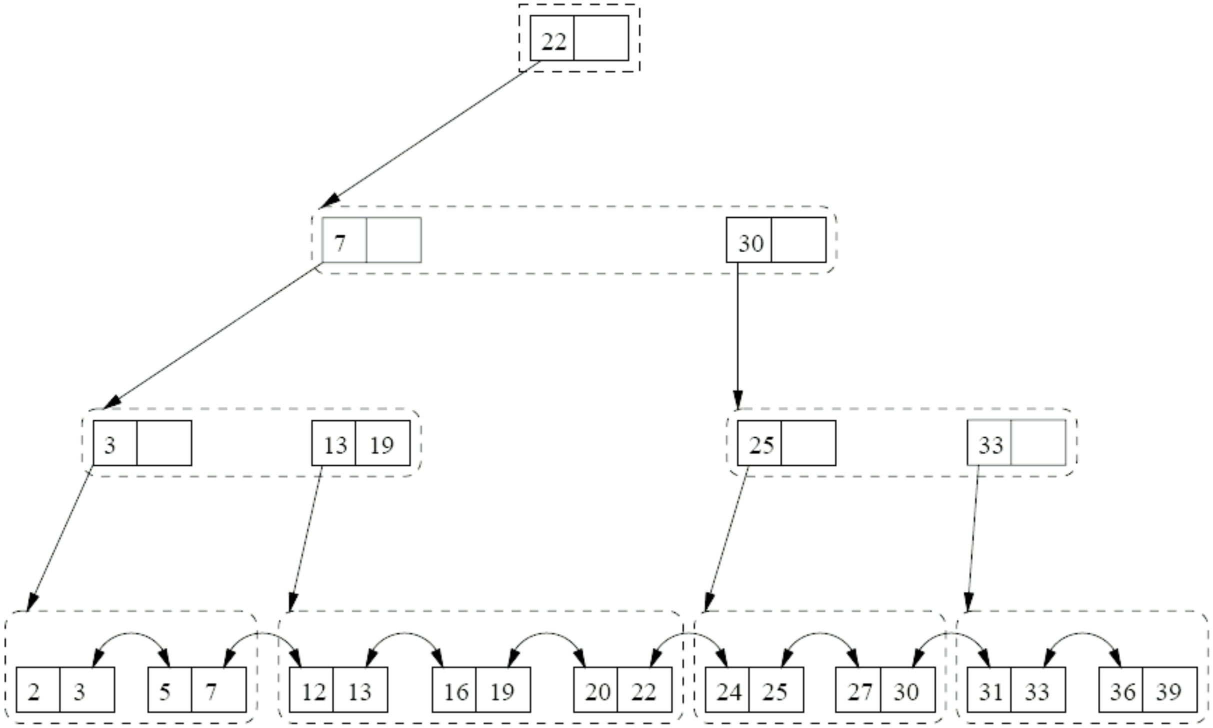
Making CSB+-Trees Processor Conscious

Michael L. Samuel
Anders U. Pedersen
Philippe Bonnet
University of Copenhagen

Goal

- Make CSB+-Trees processor conscious
- Incorporate our CSB+-Tree variant into MySQL



Related work

- Rao & Ross, 2000:
  - Making B+-Trees Cache Conscious in Main Memory
- Hankins & Patel, 2003:
  - Effect of Node Size on the Performance of CSB+-Trees
- Chen, Gibbons & Mowry, 2001:
  - Improving Index Performance through Prefetching

Approach

- Identify processor sensitive index-parameters
- Study performance impact of parameters
  - Isolated
  - Inside MySQL Memory storage engine
- Construct configuration table for each platform

CSB+-tree parameters

- Data structure
  - Node size
  - Fill factor
  - Tree height
  - Pointer size
- Operations
  - Searching in nodes
  - Compare method
  - Prefetching

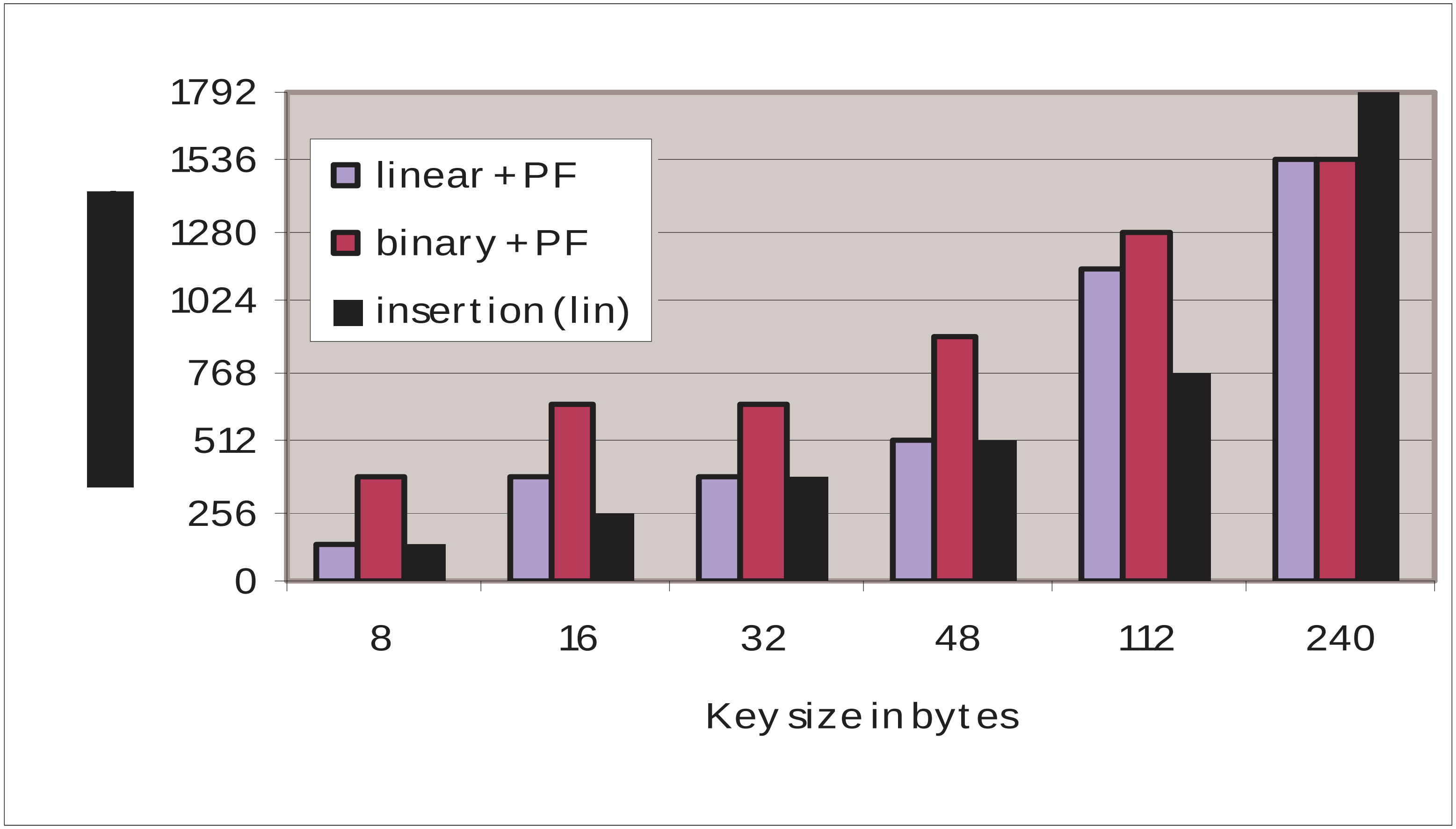
Benchmark workload

- Point query
- Range scan
- Insertion

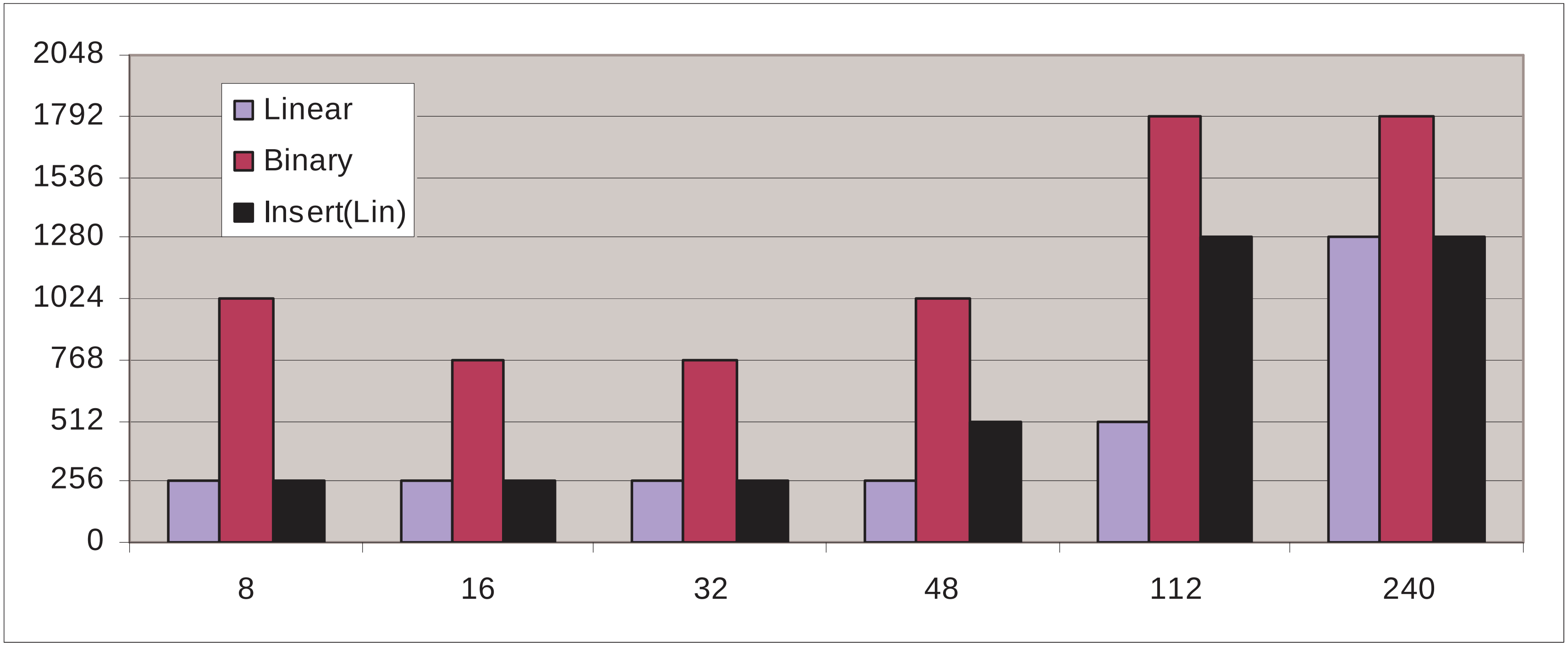
Benchmark parameters

- Node size
- Key size
- Node search method

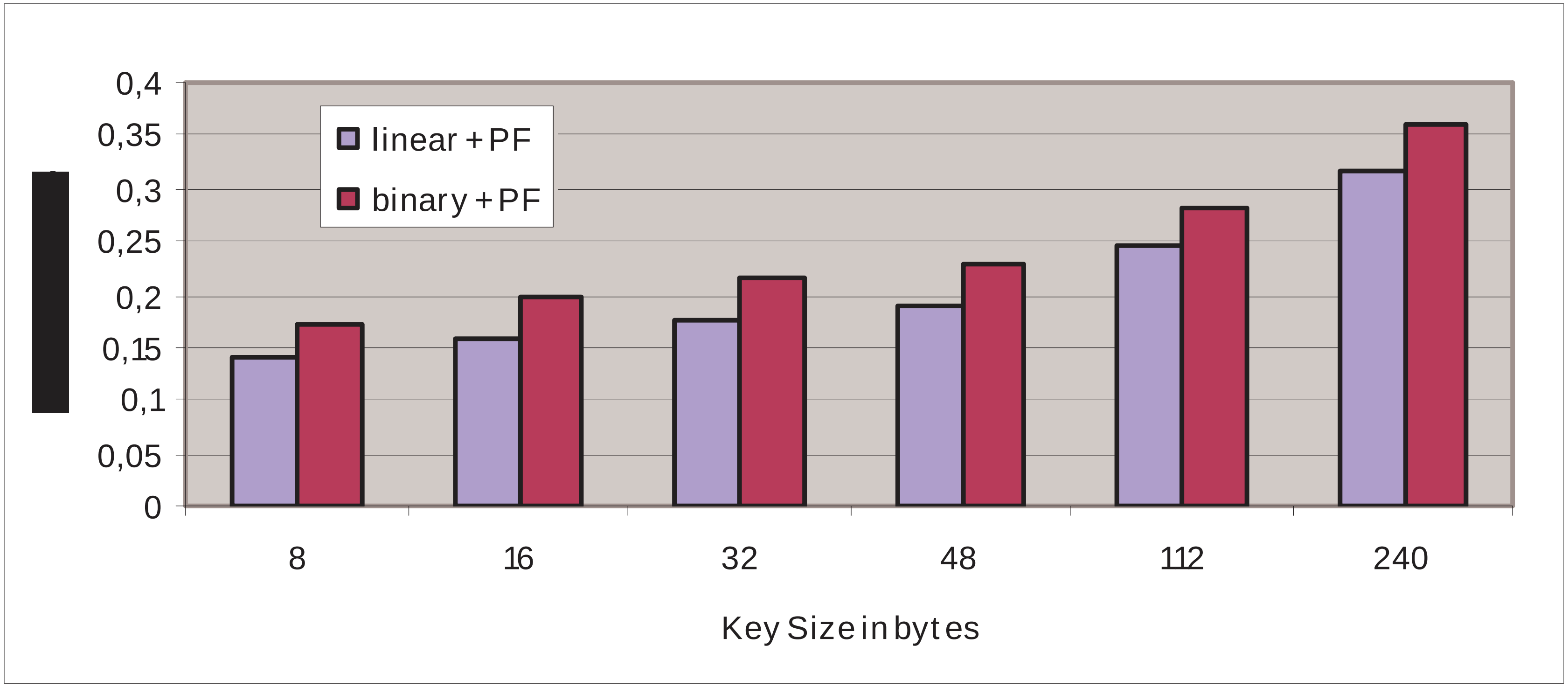
Optimal node sizes, Itanium 2



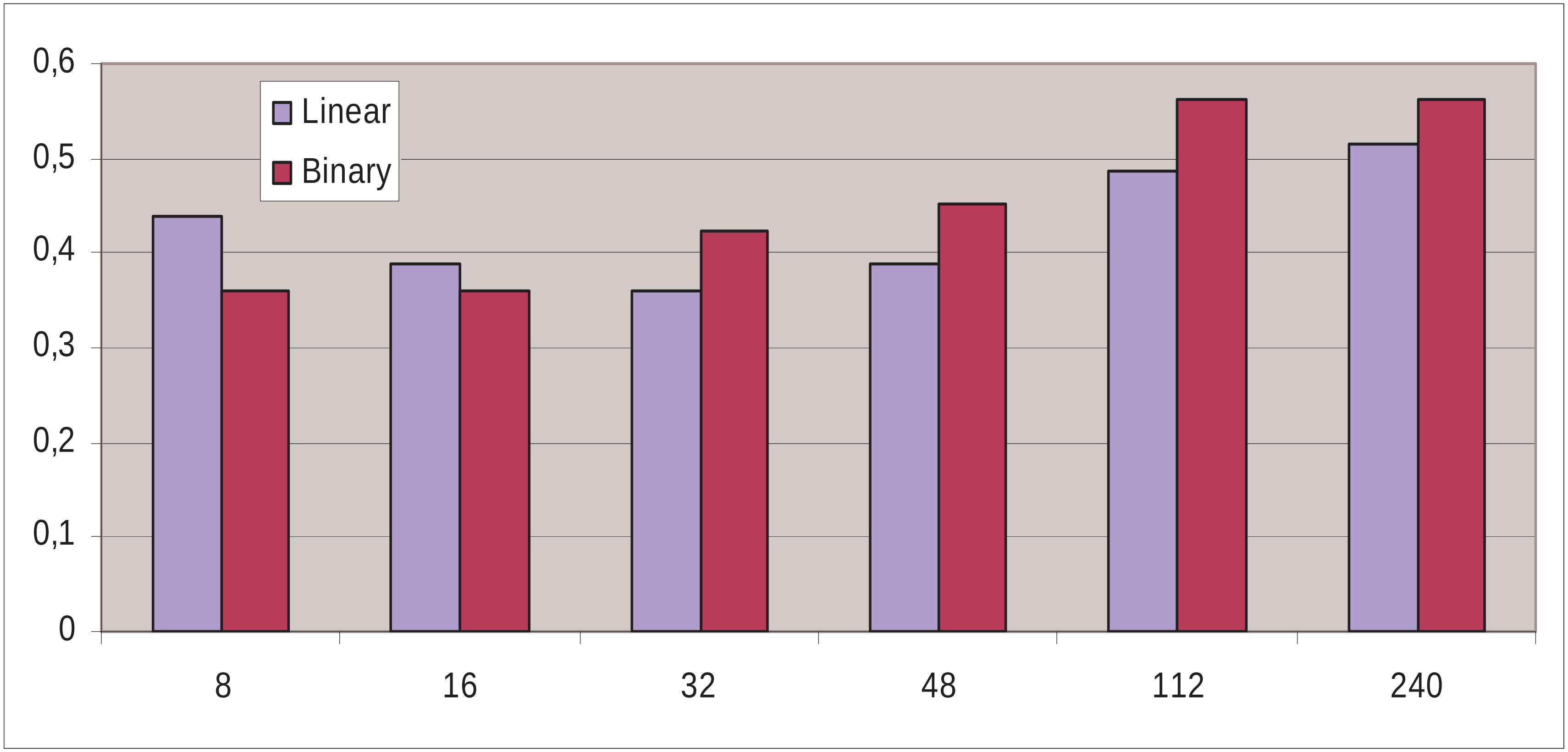
Optimal node sizes, P4



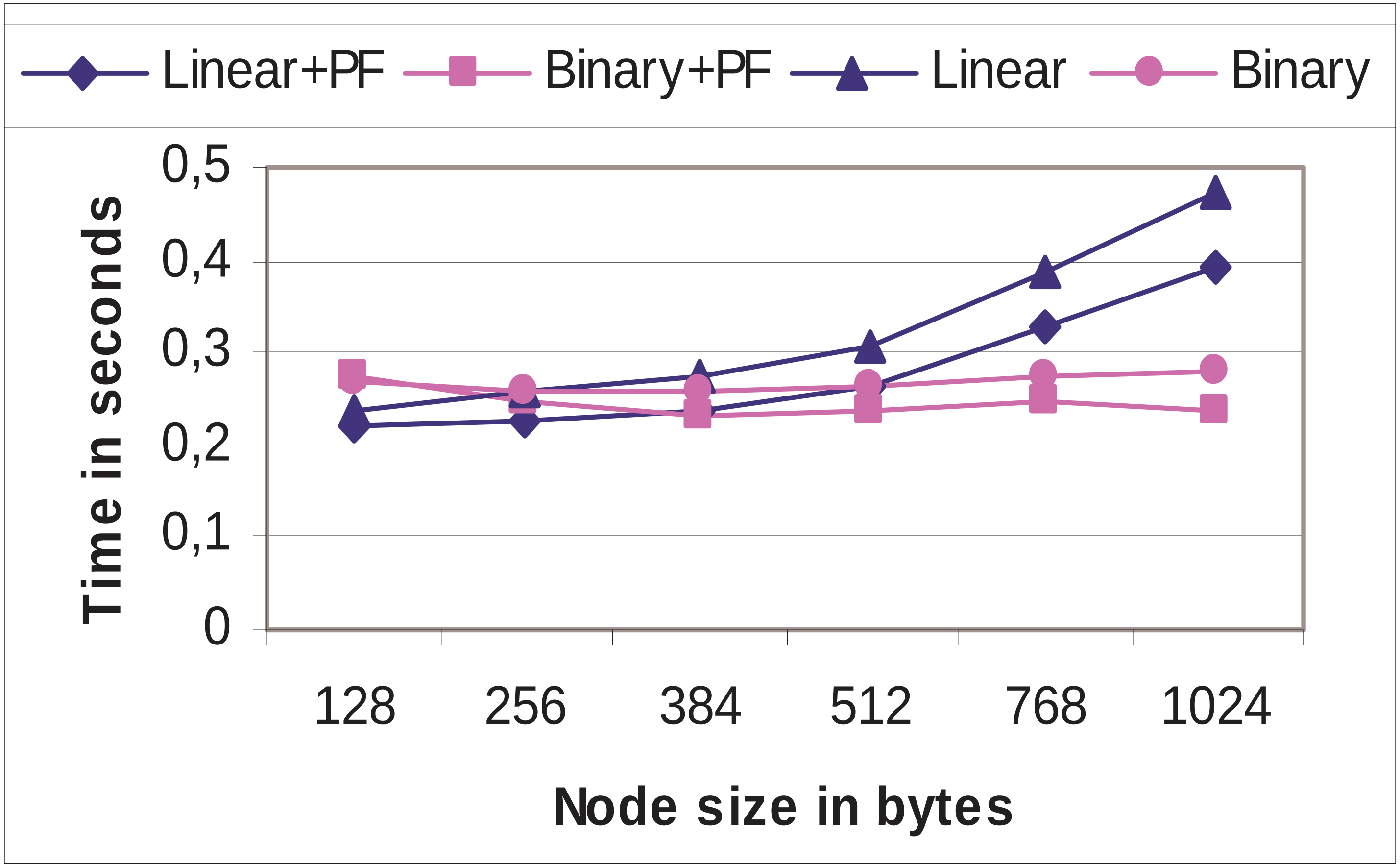
Node search running times, Itanium 2



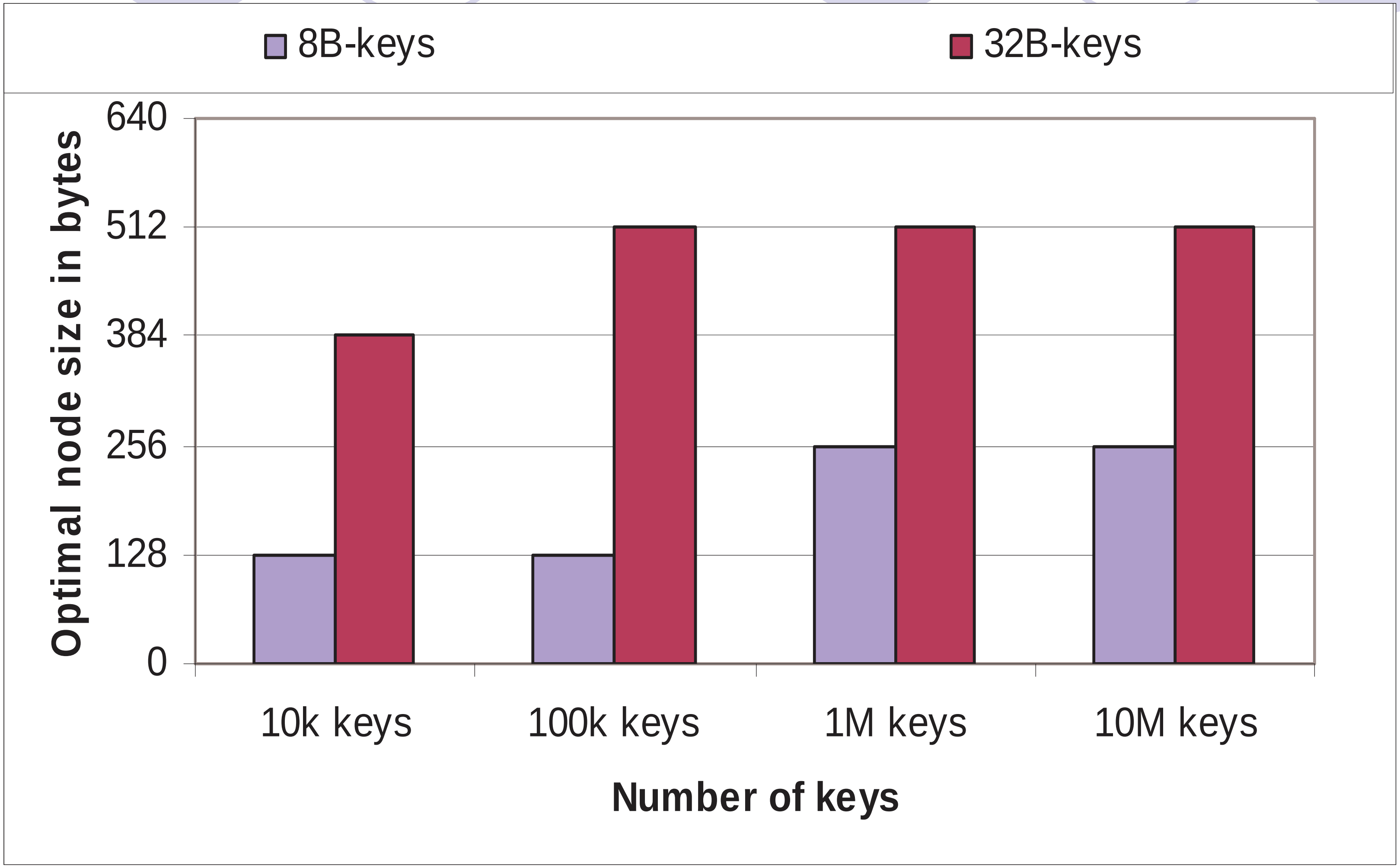
Node search running times, P4



Point query


Linear+PF
Binary+PF
Linear
Binary
Time in seconds
0,5
0,4
0,3
0,2
0,1
0
128
256
384
512
768
1024
Node size in bytes

Impact of number of keys



Future work

- Impact of other parameters
- Profile-guided optimization
- Benchmark on more platforms
- Indirect indexing
- Make it self-configuring and self-tuning

- Key comparisons and key types in MySQL
- Compare to existing access methods in MySQL

Wrap up

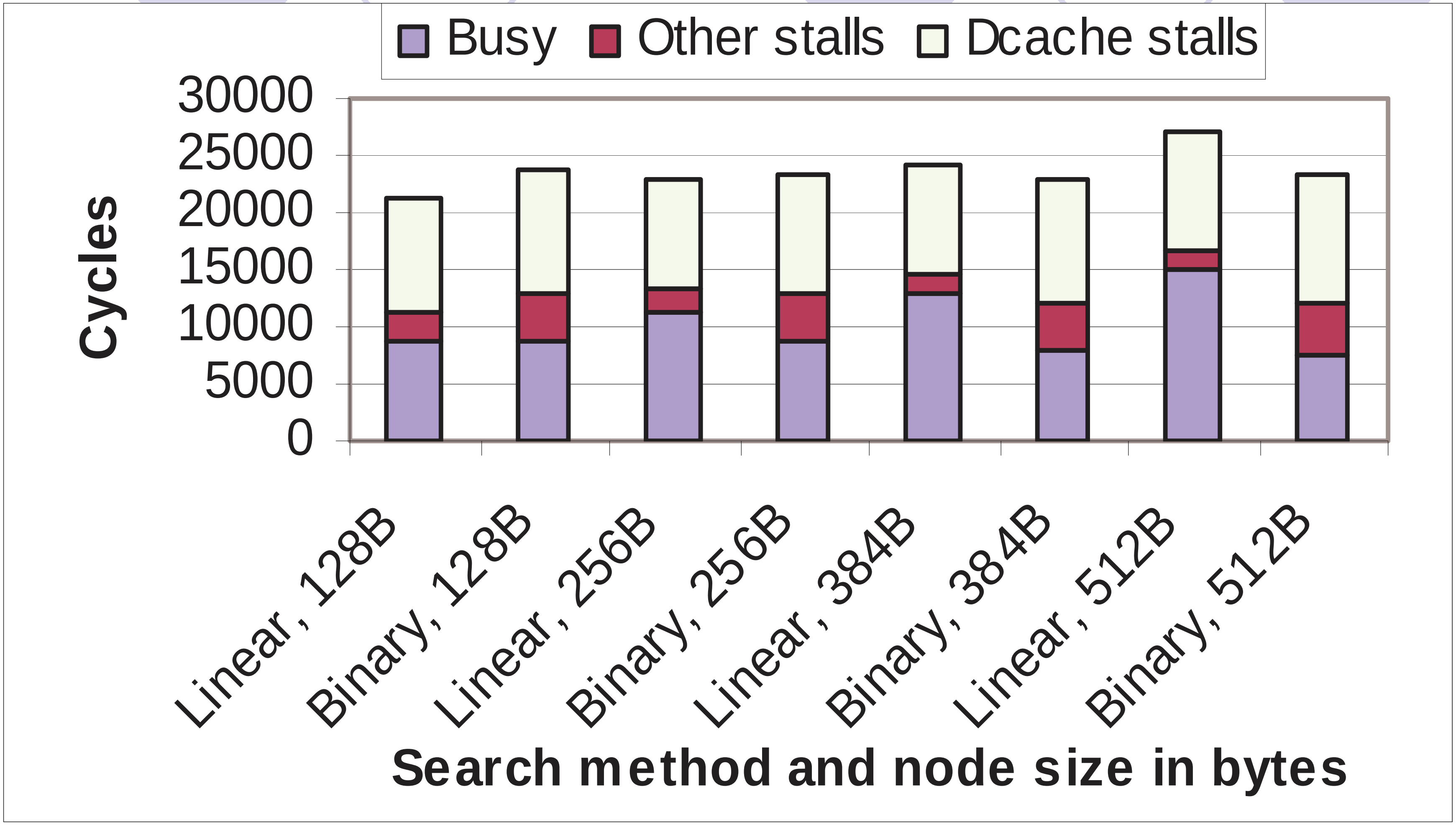
- Identified distinct differences between processors/architectures
- Portable implementation with several adjustable parameters
- Preliminary MySQL implementation

Benchmark platform: Itanium 2

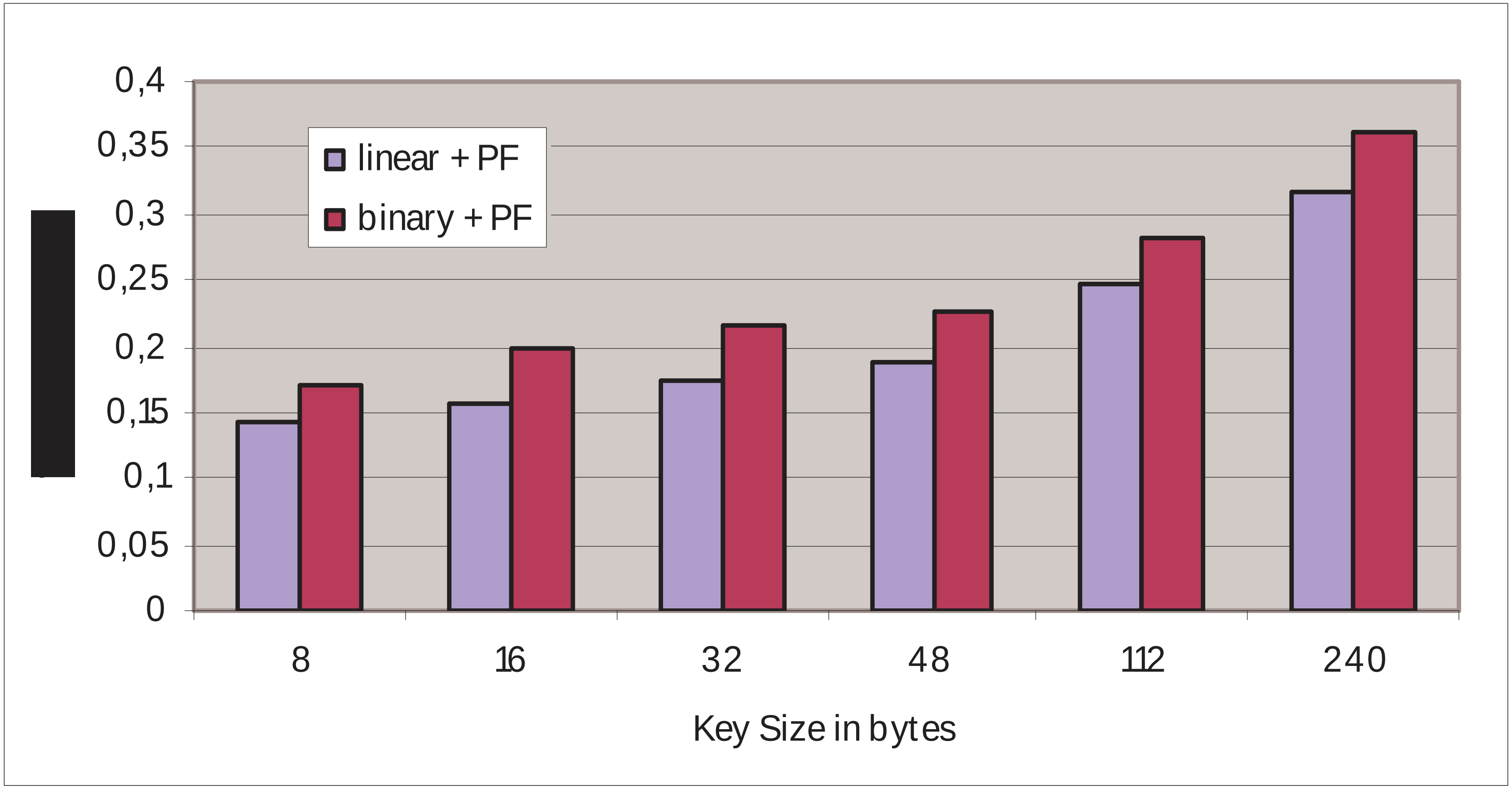
Addressing	64 bits
Clock frequency	900MHz
Number of processors	2
RAM	4GB
L1/L2/L3 (total size/line size)	16KB/64B; 256KB/128B; 1,5MB/128B
OS	Debian GNU/Linux 2.4.25
Compiler	Intel C/C++ 8.0.066

- EPIC
- No rearranging of instructions at run-time
- High-level prefetching

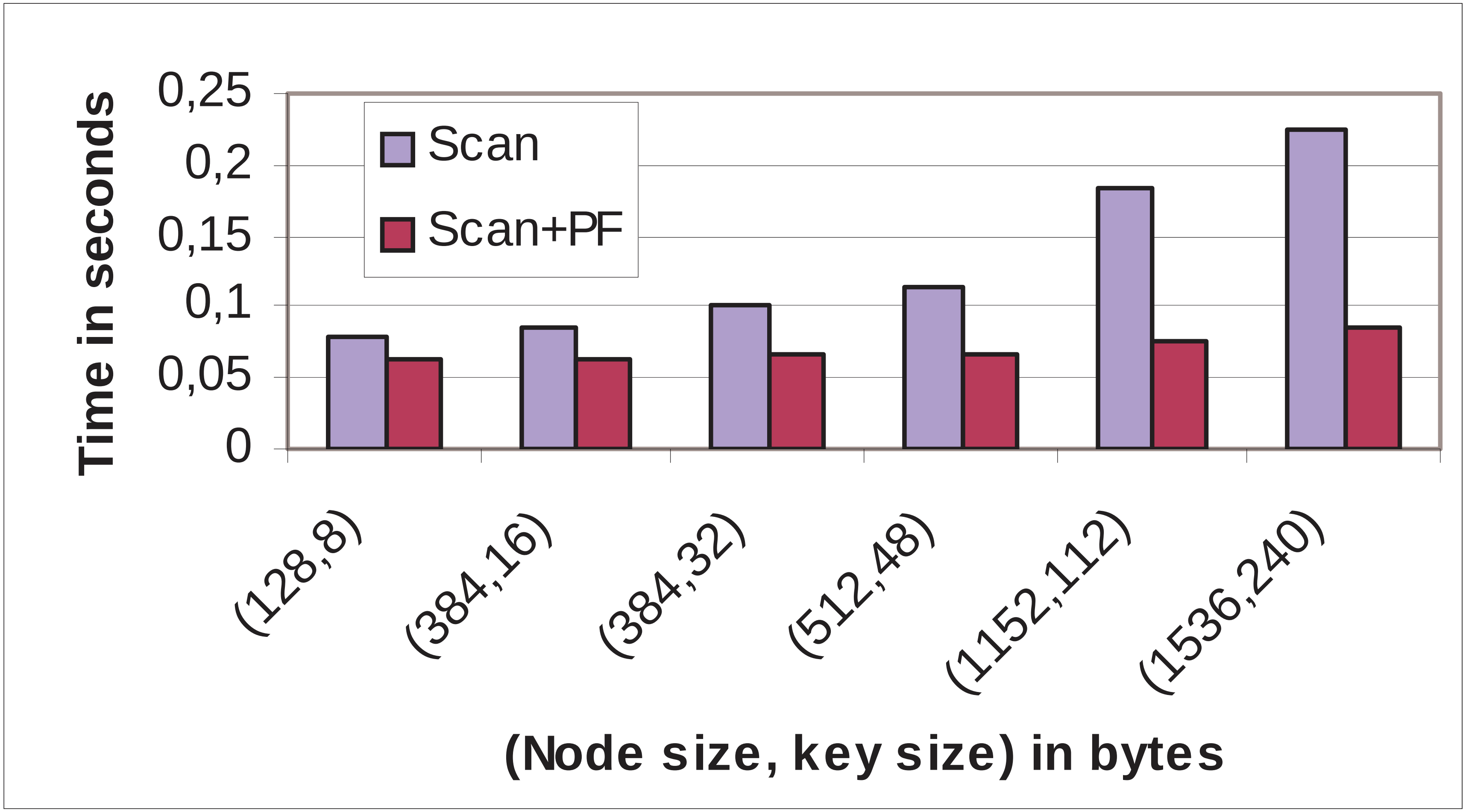
Point query - decomposed



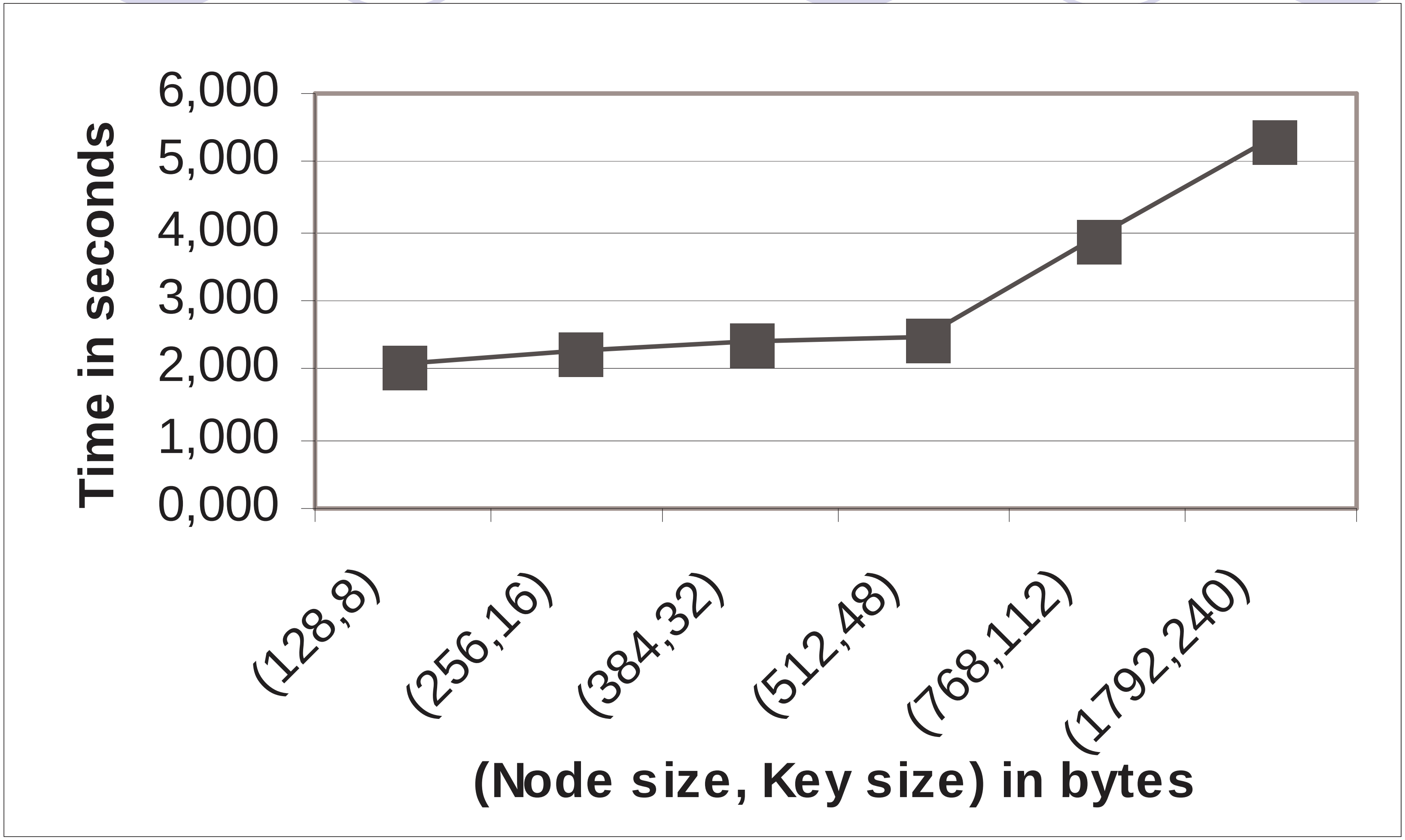
Point query: response time Itanium 2



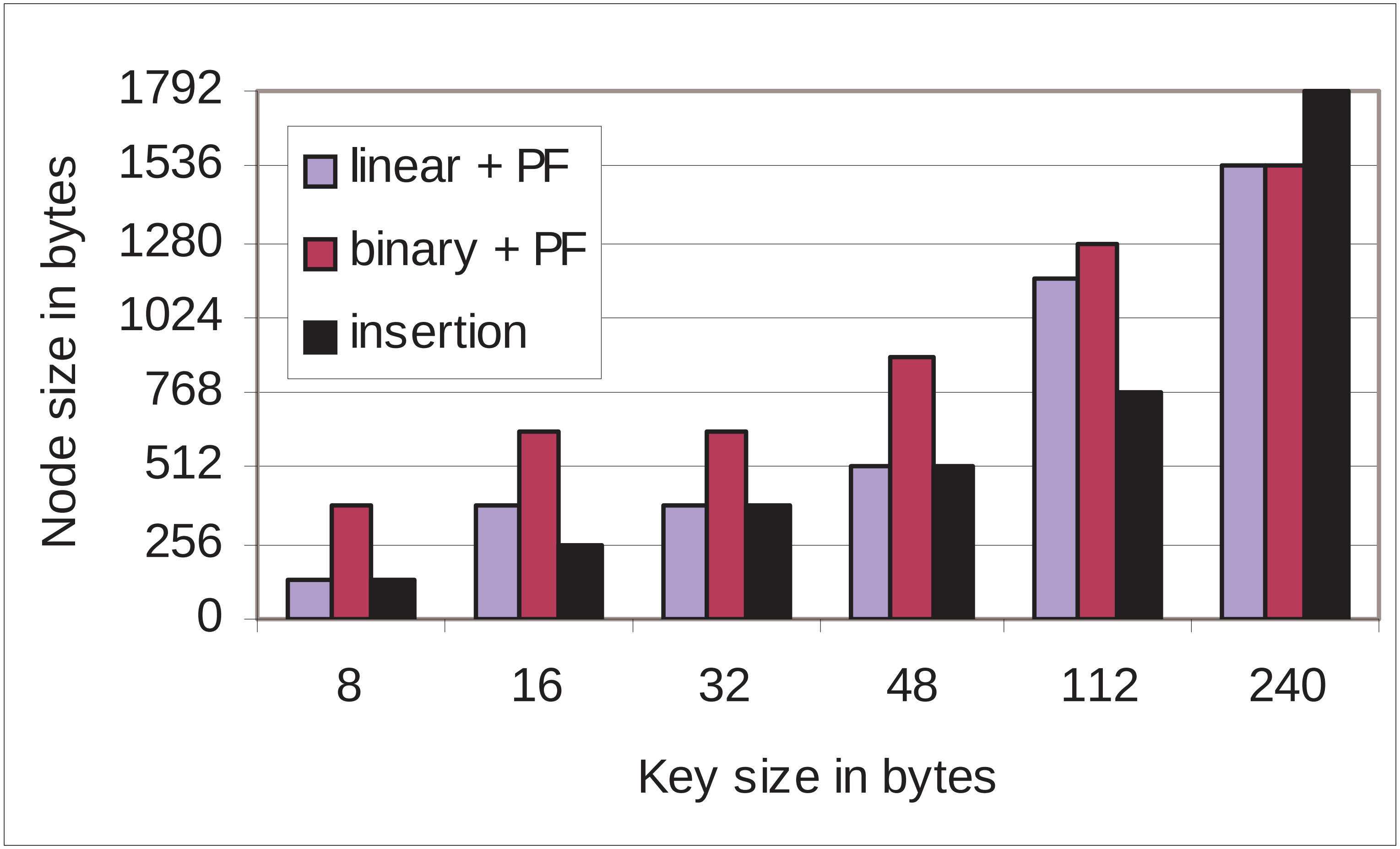
Range scan: response time



Insertion: response time



Optimal node sizes

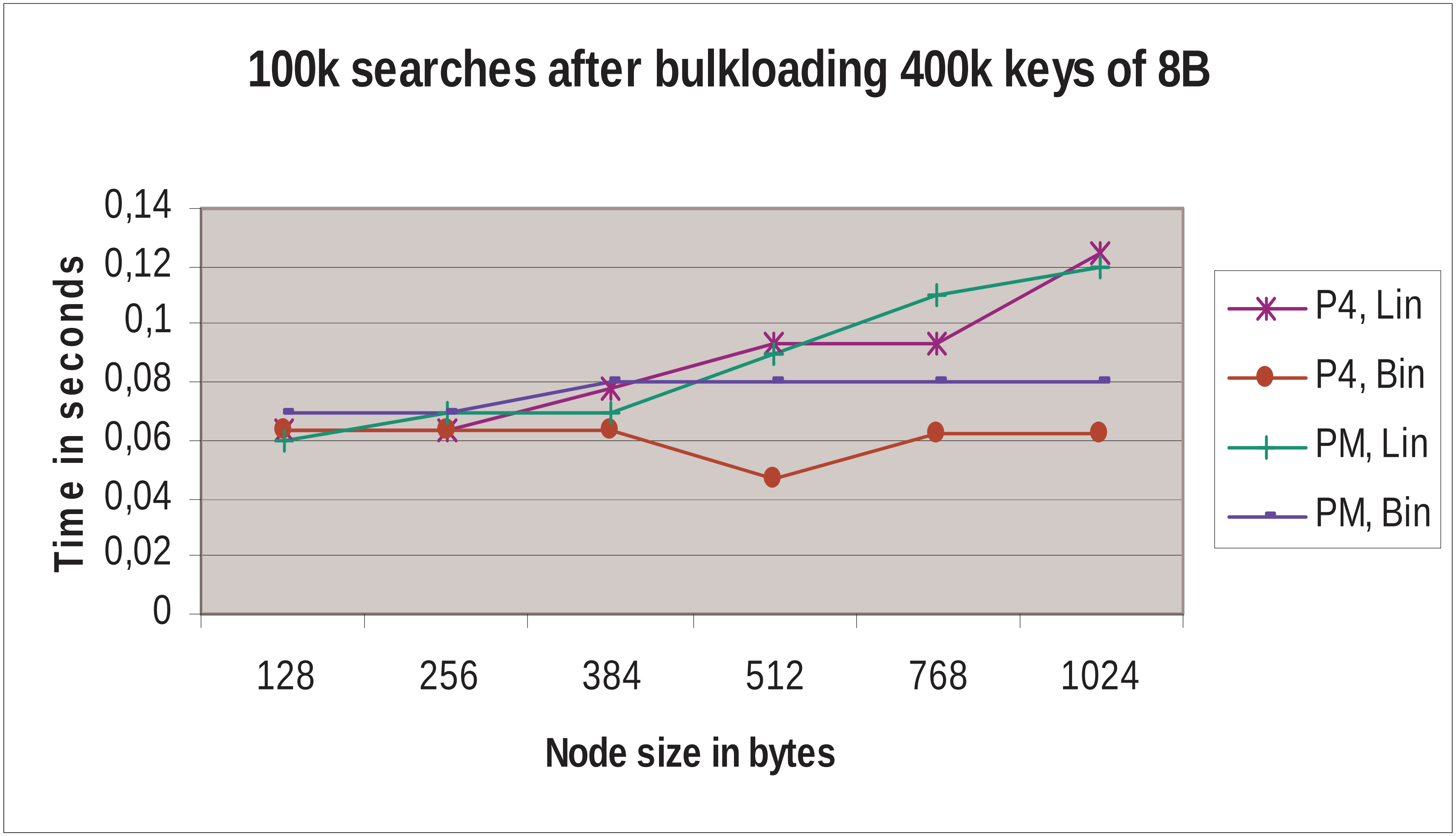


Benchmark platform: Pentium

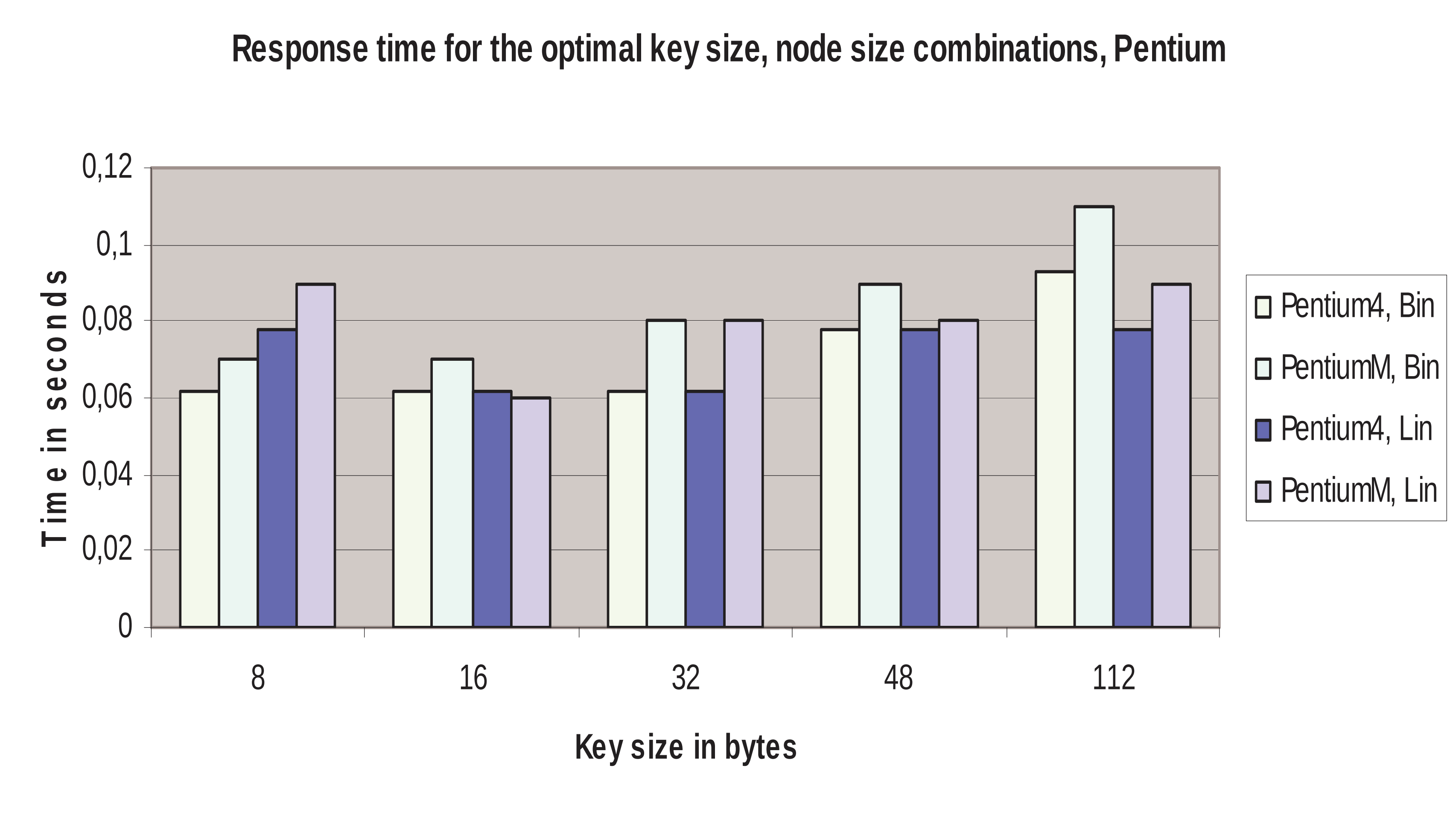
Processor	Pentium 4	Pentium M
Clock frequency	3 Ghz	1,3 Ghz
RAM	1024MB	512MB
Addressing	32	
Number of processors	1	
OS	Windows XP, SP2	
Cache	1Mb	
Compiler	MS VC++ 6.0	

- Out-of-order execution logic

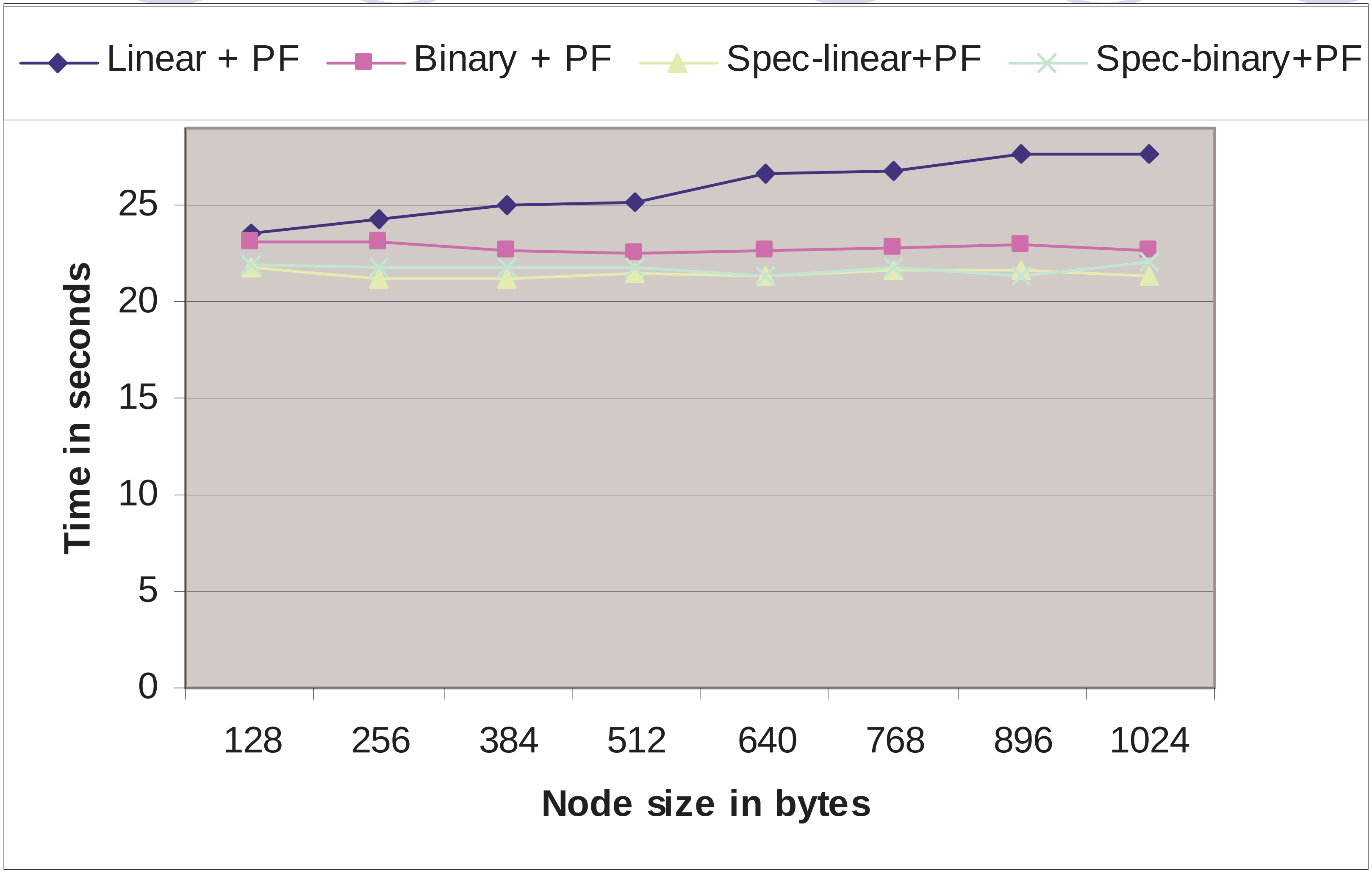
Point query


100k searches after bulkloading 400k keys of 8B
Time in seconds
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
128
256
384
512
768
1024
Node size in bytes
P4, Lin
P4, Bin
PM, Lin
PM, Bin

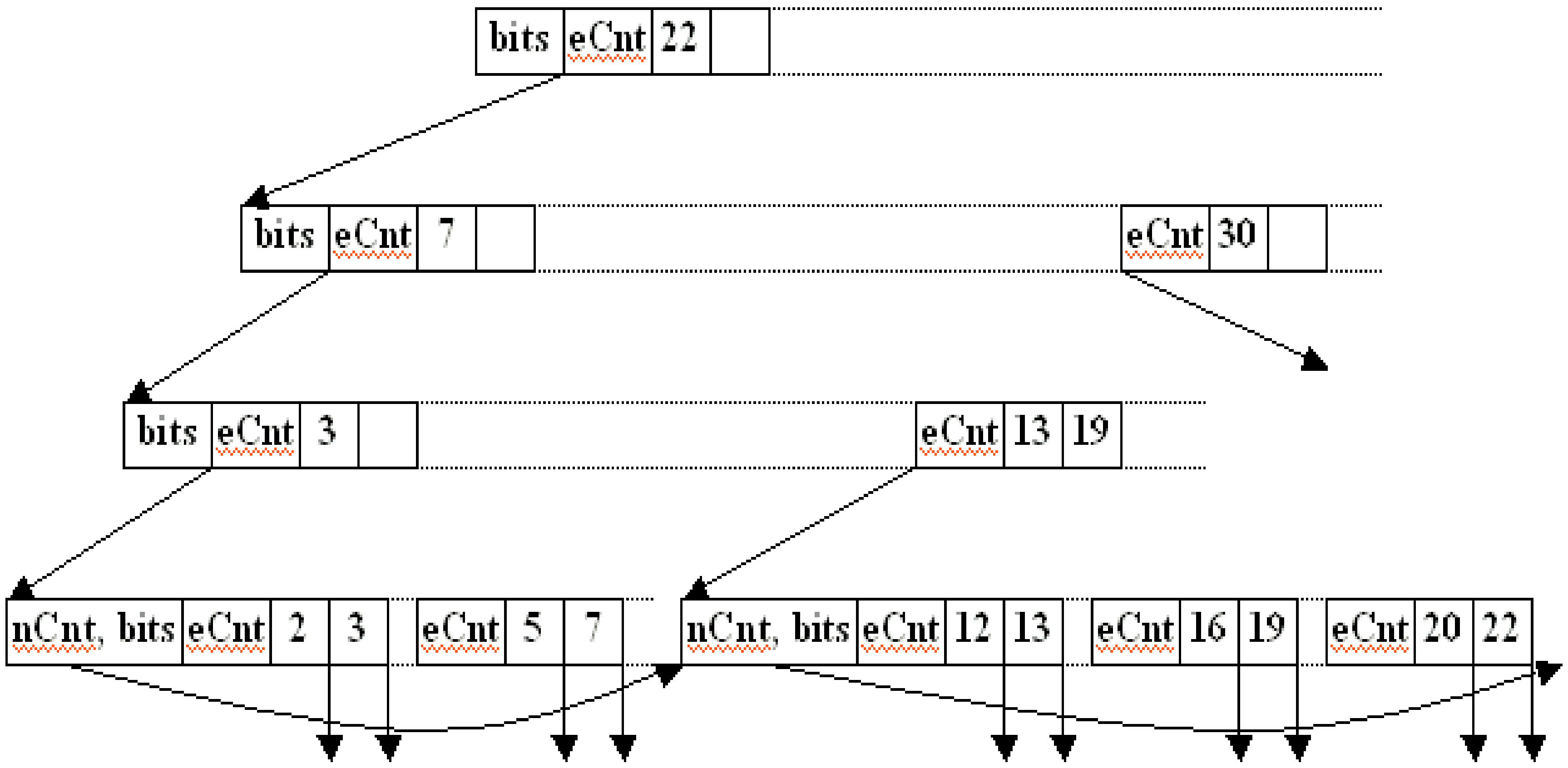
Point query: Response time



Initial MySQL results



Index structure



bits
eCnt
22
bits
eCnt
7
eCnt
30
bits
eCnt
3
eCnt
13
19
nCnt, bits
eCnt
2
3
eCnt
5
7
nCnt, bits
eCnt
12
13
eCnt
16
19
eCnt
20
22