External-Memory and Cache-Oblivious Algorithms: Theory and Experiments

Gerth Stølting Brodal

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

DANMARKS GRUNDFORSKNINGSFOND
DANISH NATIONAL RESEARCH FOUNDATION

AARHUS UNIVERSITET

Motivation

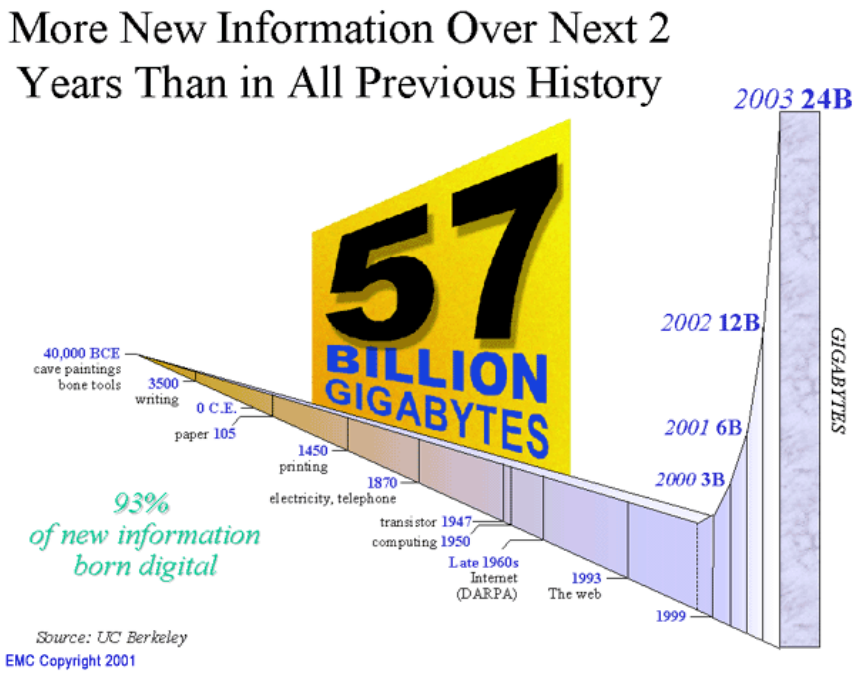
1 Datasets get **MASSIVE**

2 Computer memories are HIERARCHICAL

New Algorithmic Challenges 

....both theoretical and experimental

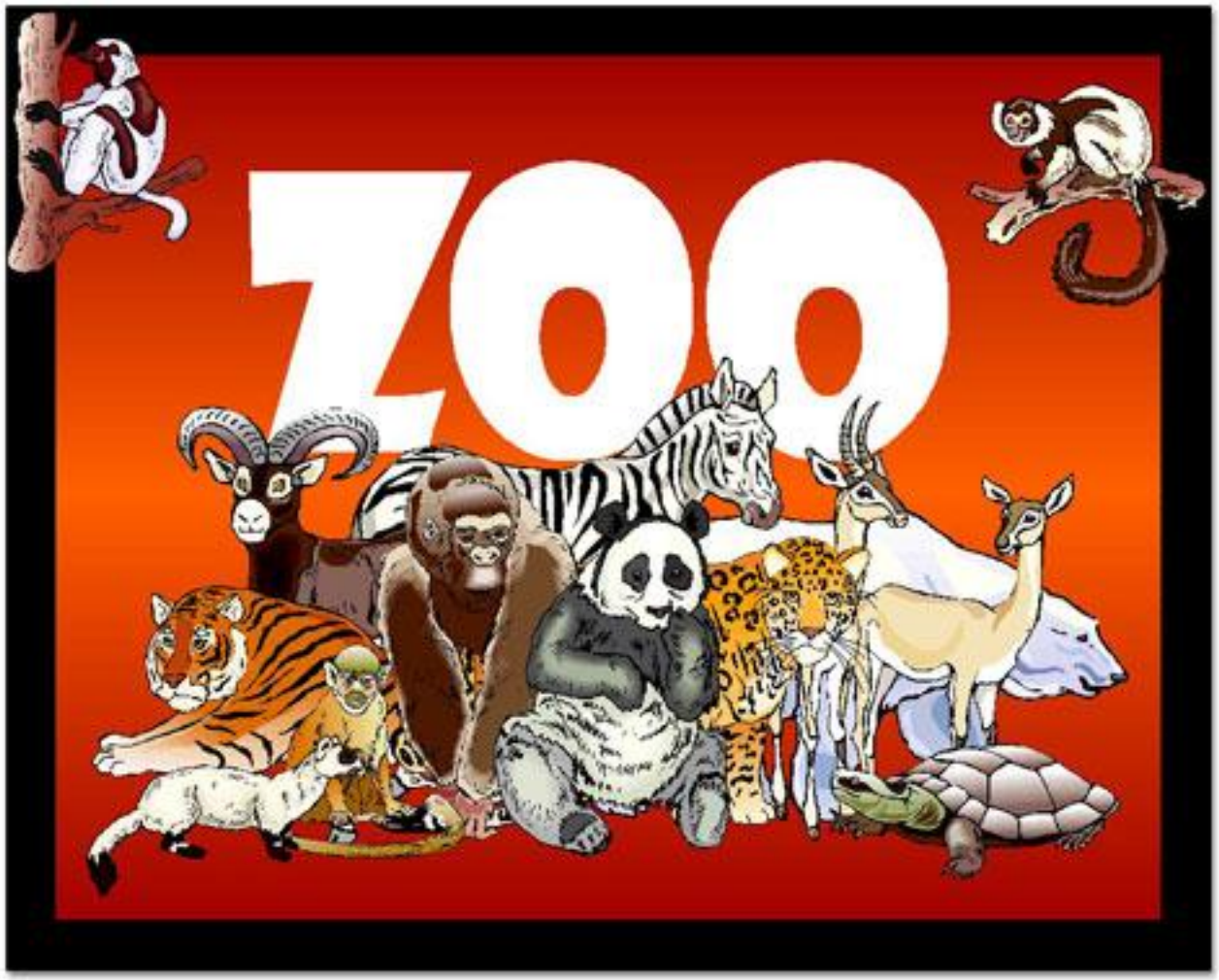
Massive Data



Examples (2002)

- Phone: AT&T 20TB phone call database, wireless tracking
- Consumer: WalMart 70TB database, buying patterns
- WEB/Network: Google index $8 \cdot 10^9$ pages, internet routers
- Geography: NASA satellites generate TB each day

Computer Hardware



A Typical Computer

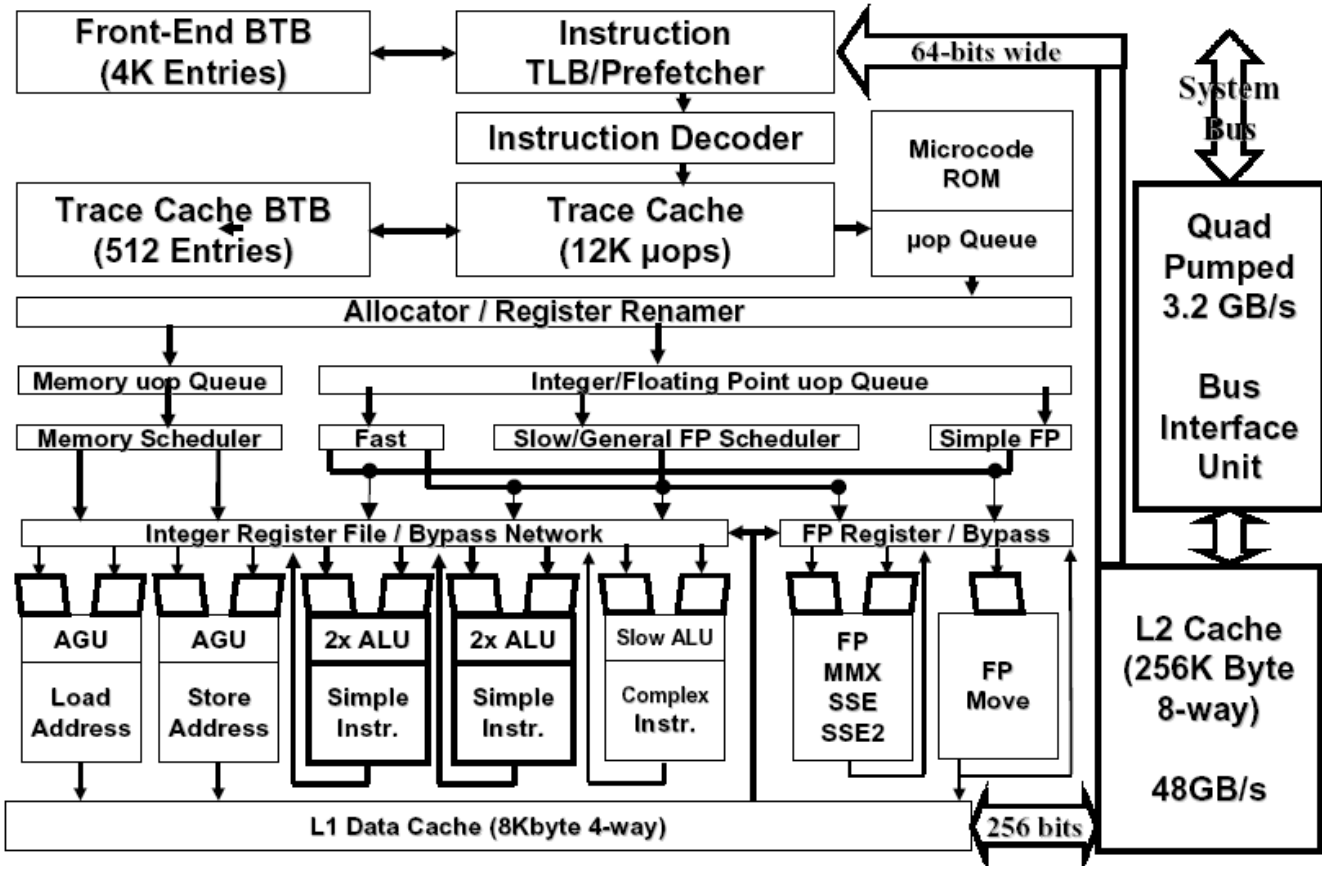
Customizing a Dell 650

www.dell.com

www.intel.com

Processor speed	2.4 – 3.2 GHz
L3 cache size	0.5 – 2 MB
Memory	1/4 – 4 GB
Hard Disk	36 GB– 146 GB
	7.200 – 15.000 RPM
CD/DVD	8 – 48x
L2 cache size	256 – 512 KB
L2 cache line size	128 Bytes
L1 cache line size	64 Bytes
L1 cache size	16 KB

Pentium® 4 Processor Microarchitecture



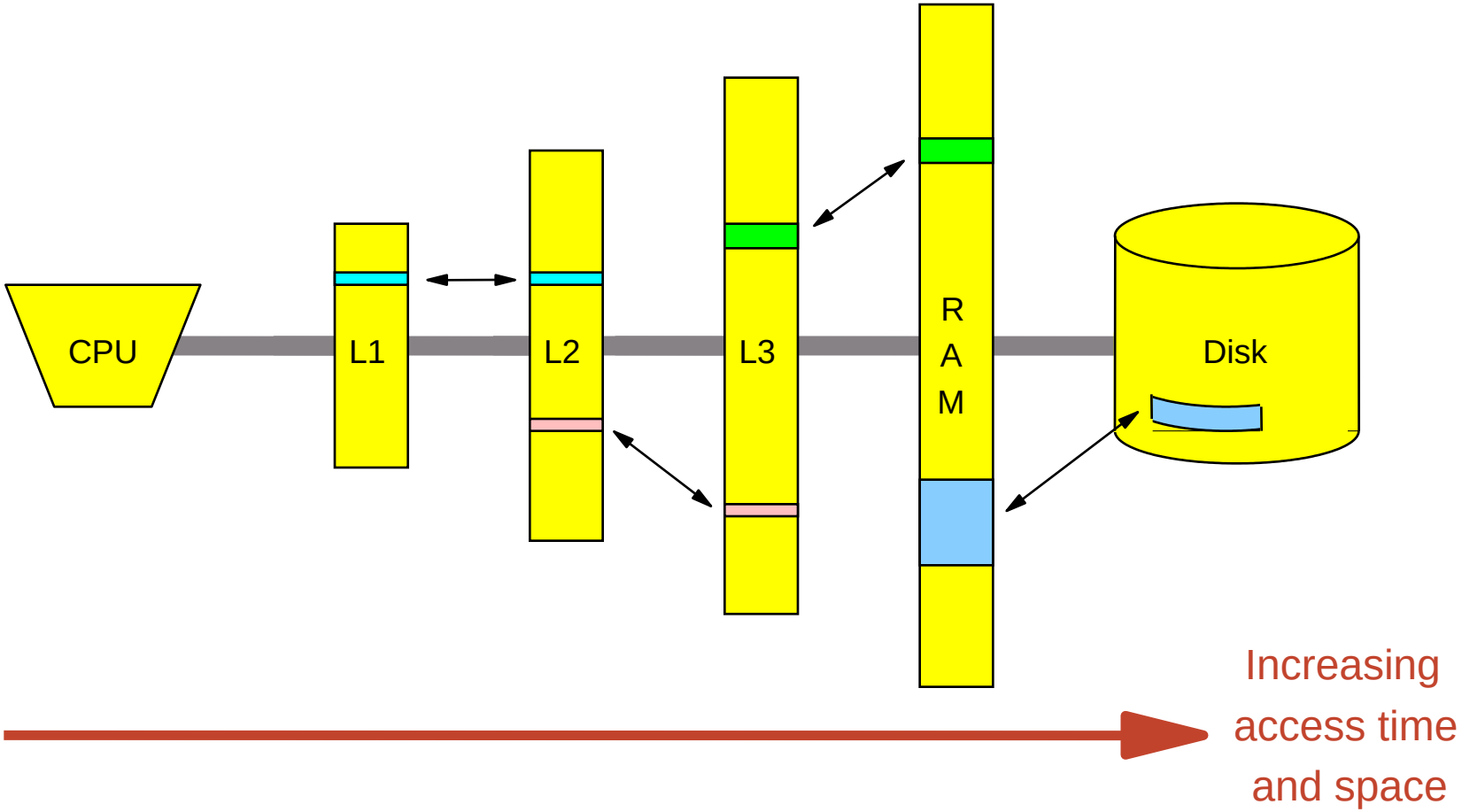
Intel Technology Journal, 2001

Memory Access Times

	Latency	Relative to CPU
Register	0.5 ns	1
L1 cache	0.5 ns	1-2
L2 cache	3 ns	2-7
DRAM	150 ns	80-200
TLB	500+ ns	200-2000
Disk	10 ms	10^7

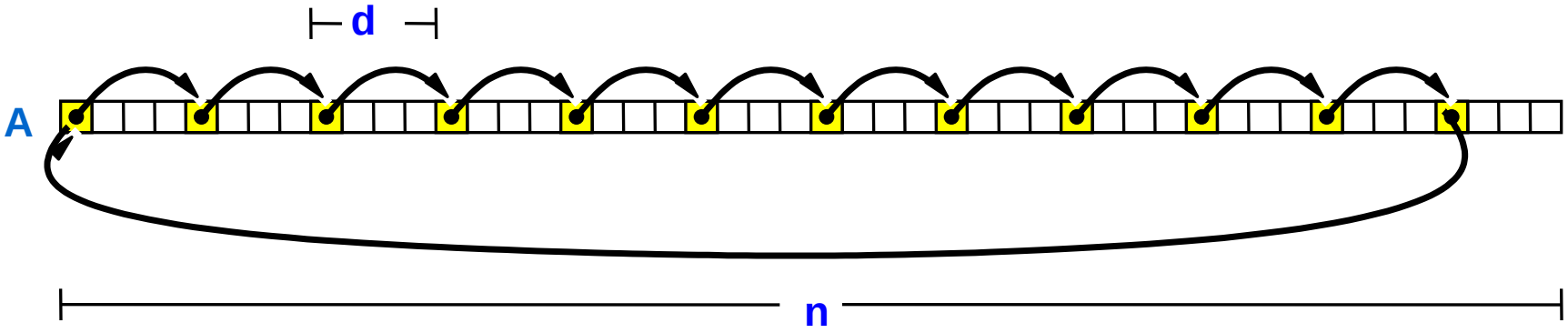
Increasing

Hierarchical Memory Basics

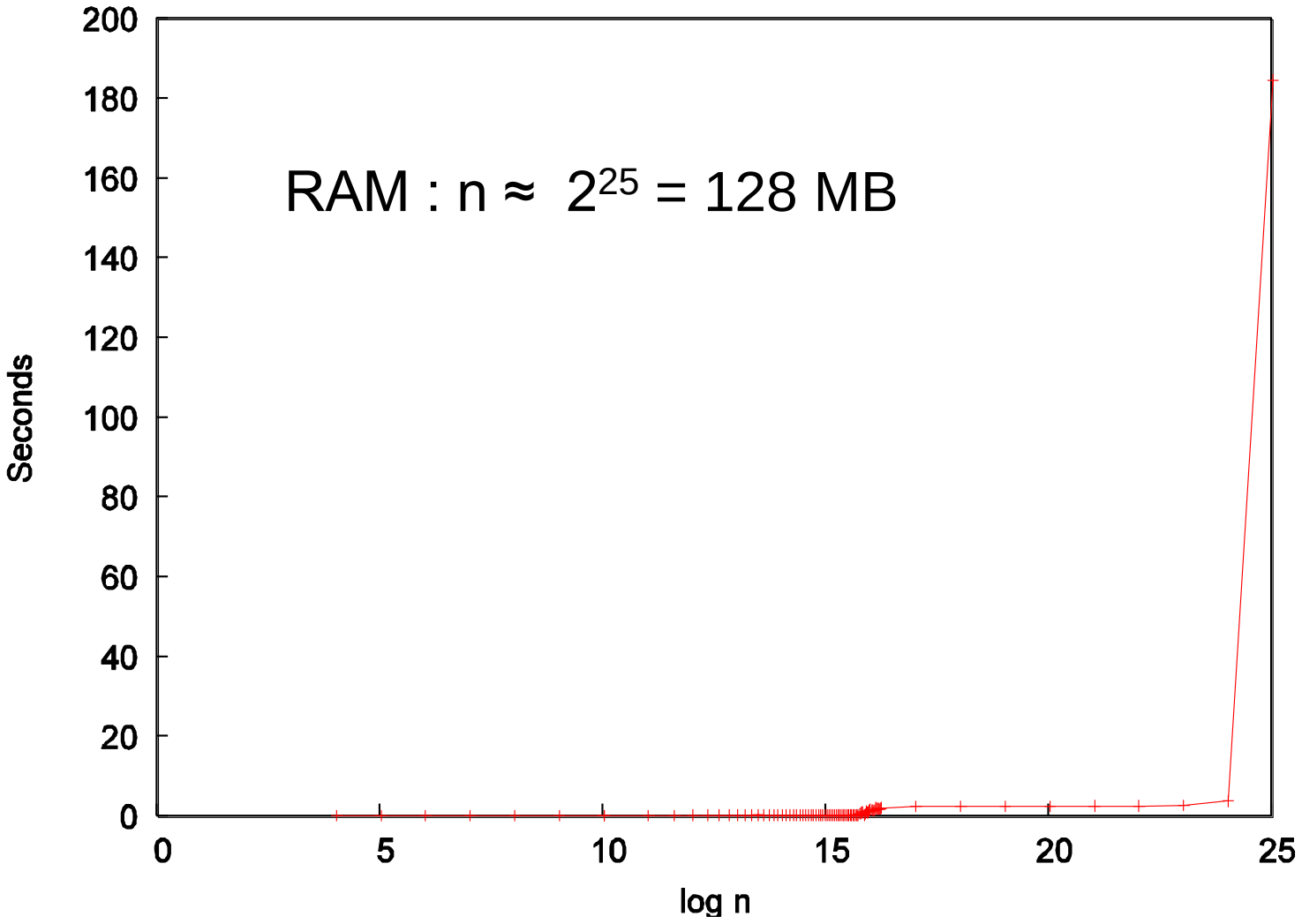


A Trivial Program

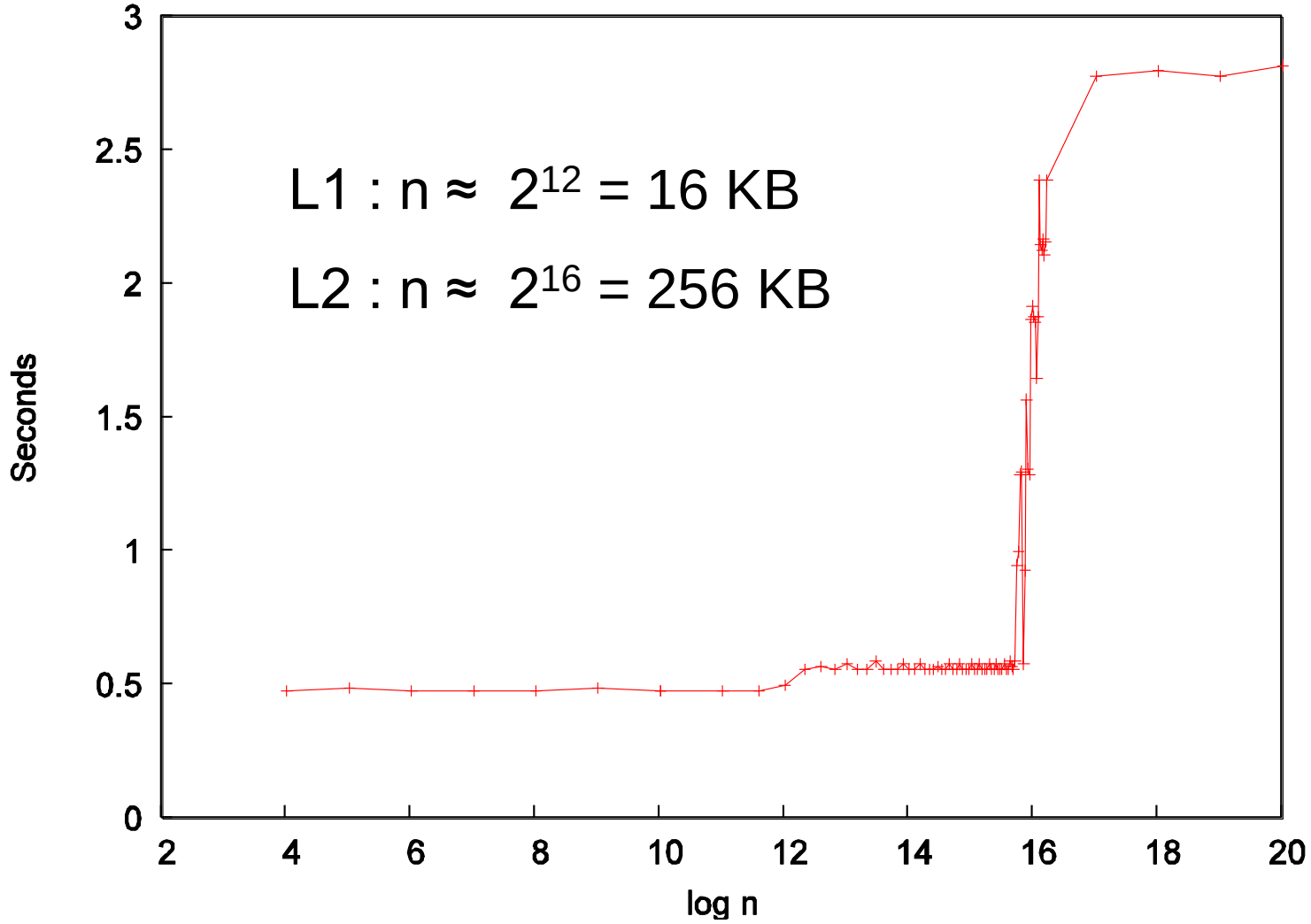
```
for (i=0; i+d<n; i+=d) A[i]=i+d;
A[i]=0;

for (i=0, j=0; j<8*1024*1024; j++) i = A[i];
```



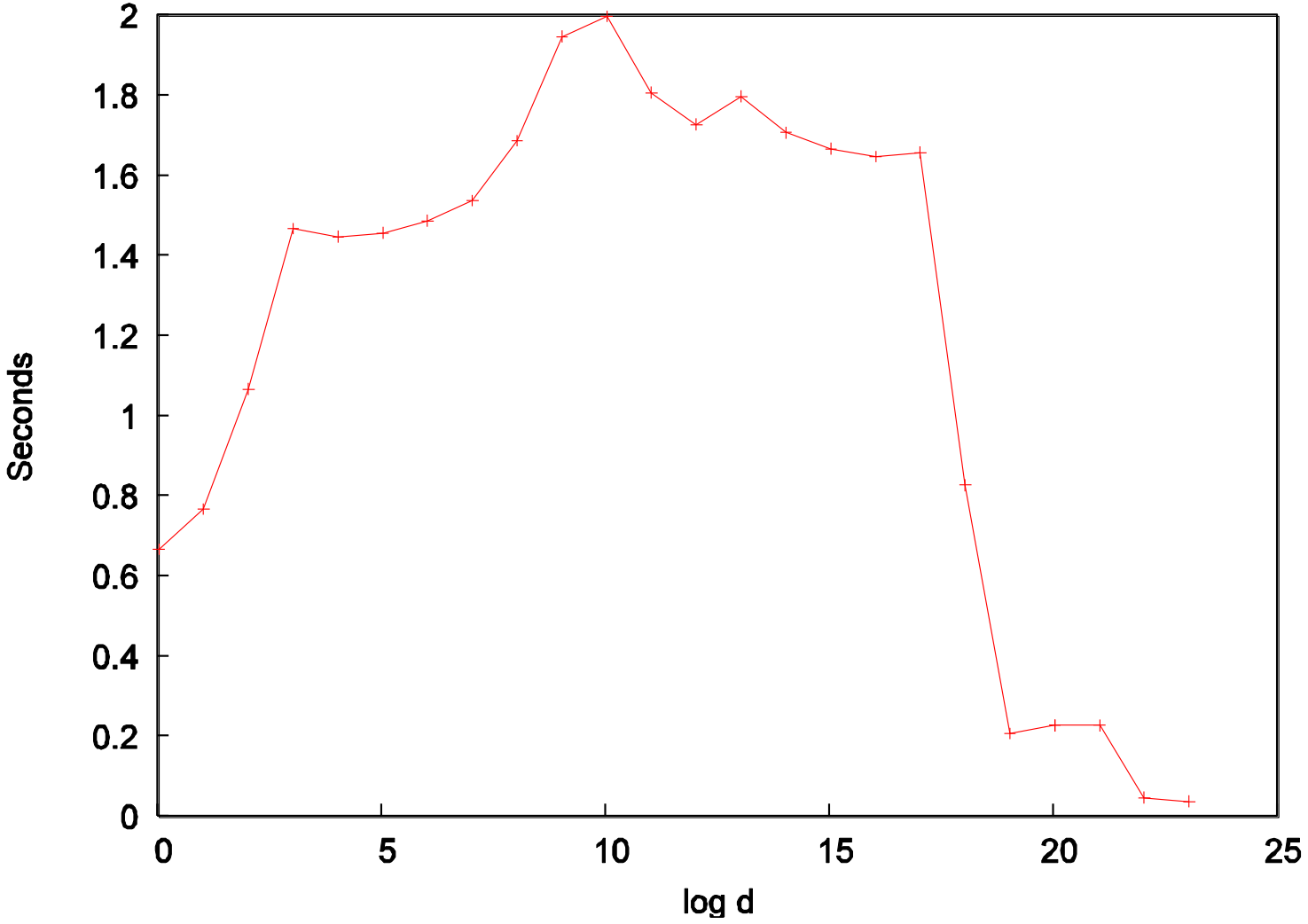
A Trivial Program — d=1



A Trivial Program — d=1



A Trivial Program — $n=2^{24}$



A Trivial Program

— hardward specification

Experiments were performed on a DELL 8000, PentiumIII, 850MHz, 128MB RAM, running Linux 2.4.2, and using gcc version 2.96 with optimization -O3

L1 instruction and data caches

- 4-way set associative, 32-byte line size
- 16KB instruction cache and 16KB write-back data cache

L2 level cache

- 8-way set associative, 32-byte line size
- 256KB

www.Intel.com

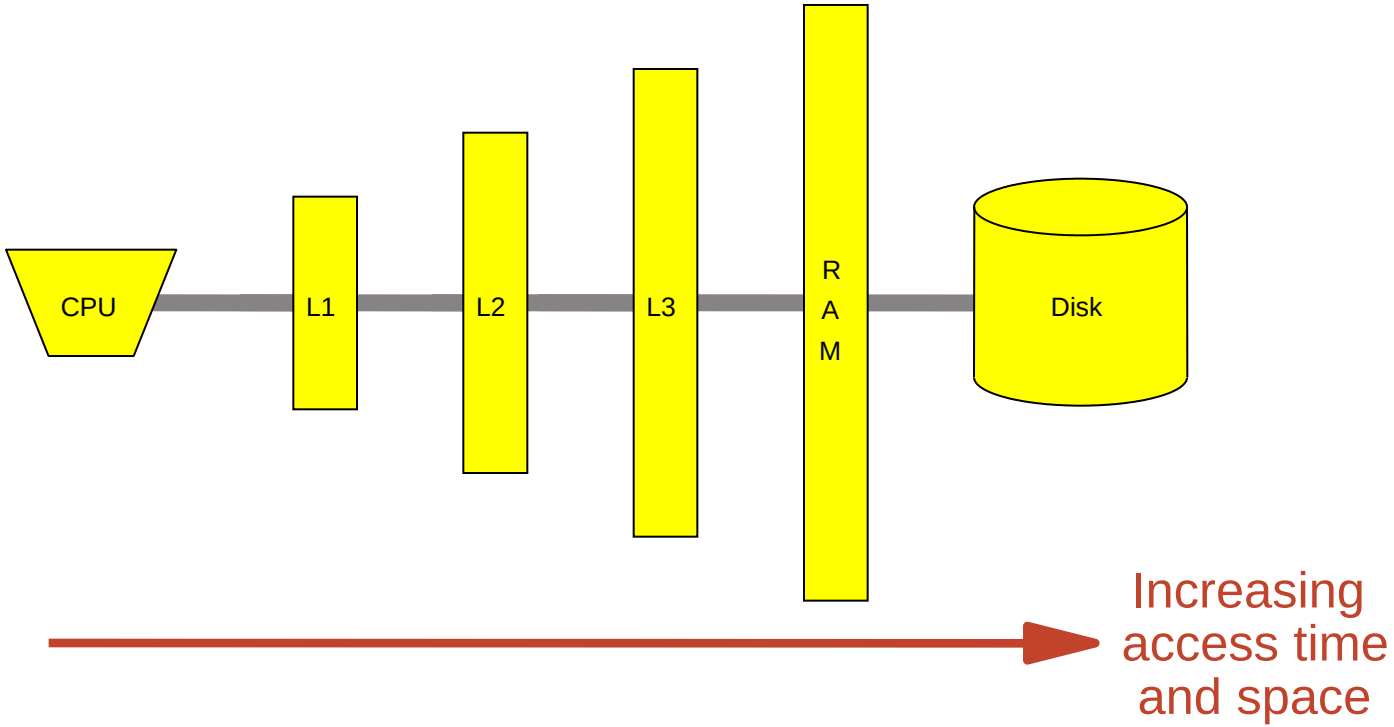
Algorithmic Problem

- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters
  - by knowing many of the parameters at runtime, or
  - by knowing few essential parameters, or
  - ignoring the memory hierarchies ⟵ **Practice**
- Programs are executed on unpredictable configurations
  - Generic portable and scalable software libraries
  - Code downloaded from the Internet, e.g. Java applets
  - Dynamic environments, e.g. multiple processes

Memory Models

Hierarchical Memory Model

— many parameters

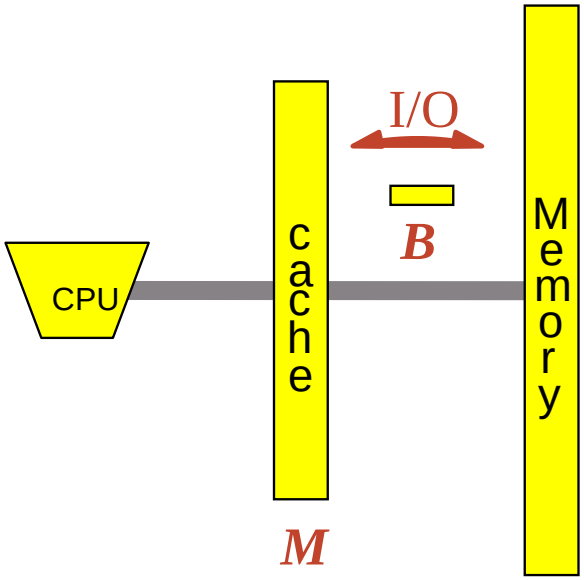


Limited success because **to complicated**

External Memory Model— two parameters

Aggarwal and Vitter 1988

- Measure number of block transfers between two memory levels
- Bottleneck in many computations
- Very successful (simplicity)



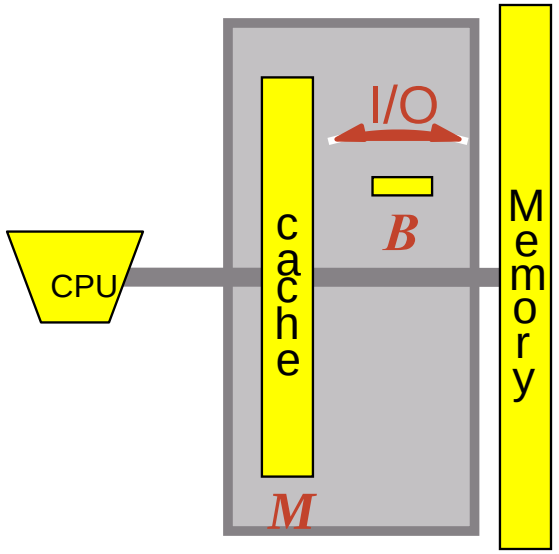
Limitations

- Parameters B and M must be known
- Does not handle multiple memory levels
- Does not handle dynamic M

Ideal Cache Model— no parameters !?

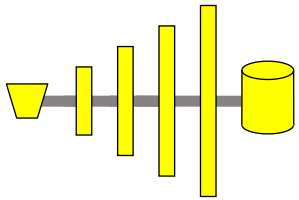
Frigo, Leiserson, Prokop, Ramachandran 1999

- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement
- strategy arbitrary B and M



Advantages

- Optimal on arbitrary level → optimal on all levels
- Portability, B and M not hard-wired into algorithm
- Dynamic changing parameters



Justification of the Ideal-Cache Model

Frigo, Leiserson, Prokop, Ramachandran 1999

Optimal replacement LRU + 2 × cache size → at most 2 × cache misses

Sleator and Tarjan, 1985

Corollary

$T_{M,B}(N) = O(T_{2M,B}(N))$) #cache misses using LRU is $O(T_{M,B}(N))$

Two memory levels

Optimal cache-oblivious algorithm satisfying $T_{M,B}(N) = O(T_{2M,B}(N))$
→ optimal #cache misses on each level of a multilevel LRU cache

Fully associativity cache

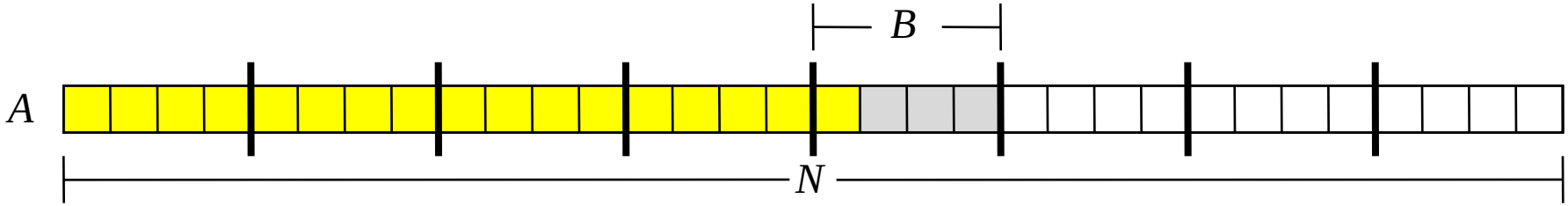
Simulation of LRU

- Direct mapped cache
- Explicit memory management
- Dictionary (2-universal hash functions) of cache lines in memory
- Expected O(1) access time to a cache line in memory

Basic External-Memory and Cache-Oblivious Results

Scanning

```
sum = 0
for i = 1 to N do sum = sum + A[i]
```

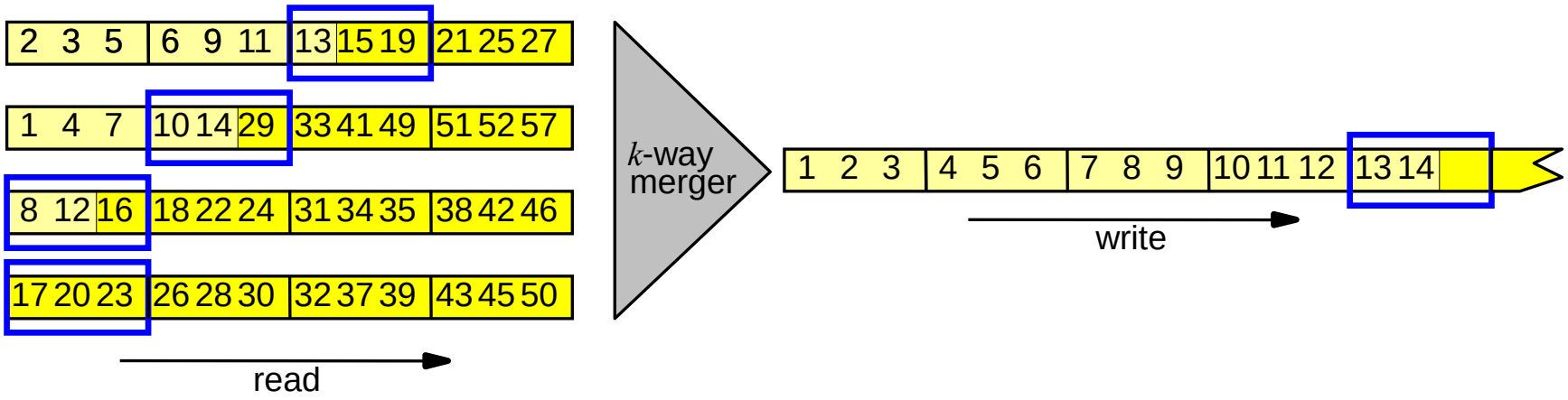


$O(N/B)$ I/Os

Corollary External/Cache-oblivious selection requires $O(N/B)$ I/Os

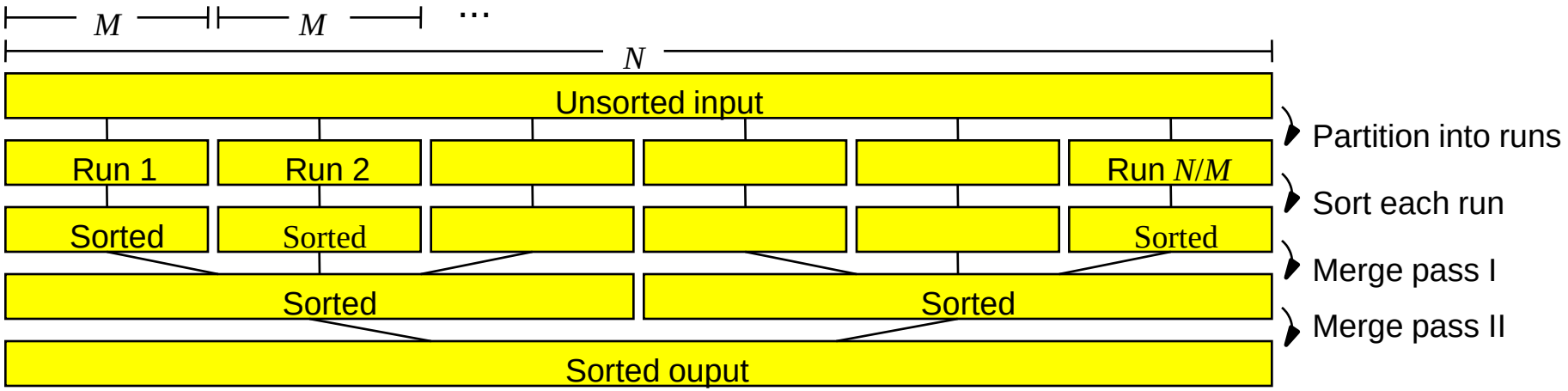
Hoare 1961 / Blum et al. 1973

External-Memory Merging



Merging k sequences with N elements requires $O(N/B)$ IOs provided $k \leq M/B - 1$

External-Memory Sorting



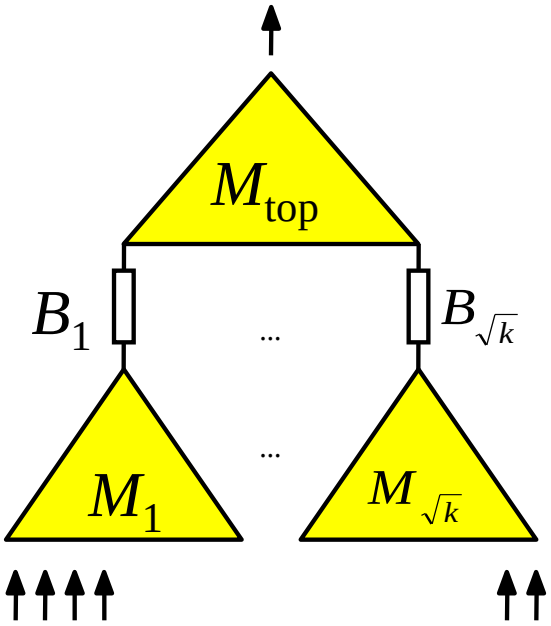
- θ(M/B)-way MergeSort achieves optimal $O(\text{Sort}(N)=O(N/B \cdot \log_{M/B}(N/B))$ I/Os

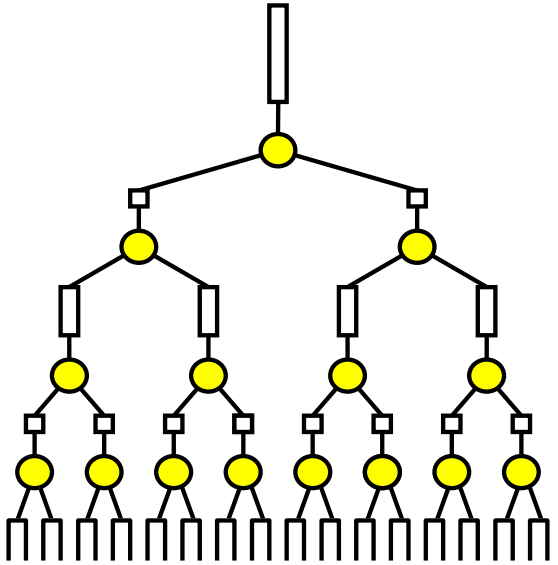
Aggarwal and Vitter 1988

Cache-Oblivious Merging

Frigo, Leiserson, Prokop, Ramachandran 1999

- k-way merging (lazy) using binary merging with buffers
- tall cache assumption $M \geq B^2$
- $O(N/B \cdot \log_{M/B} k)$ IOs

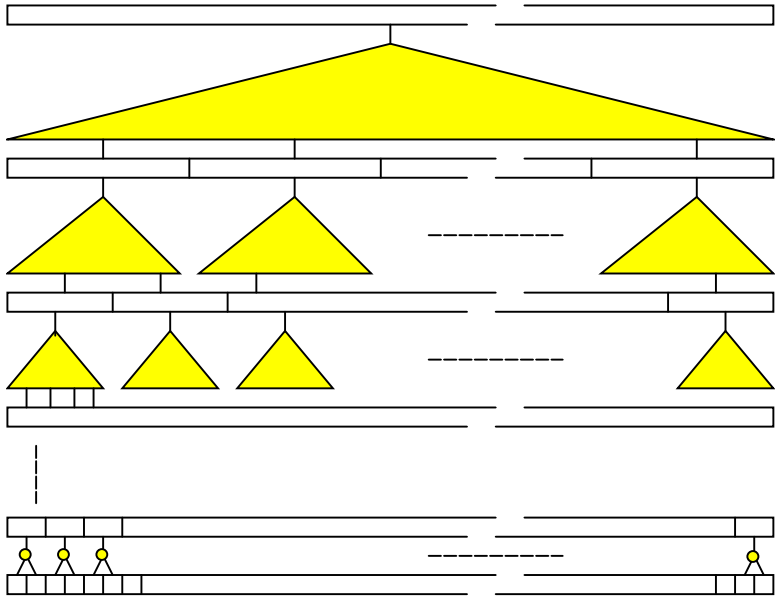




Cache-Oblivious Sorting – FunnelSort

Frigo, Leiserson, Prokop and Ramachandran 1999

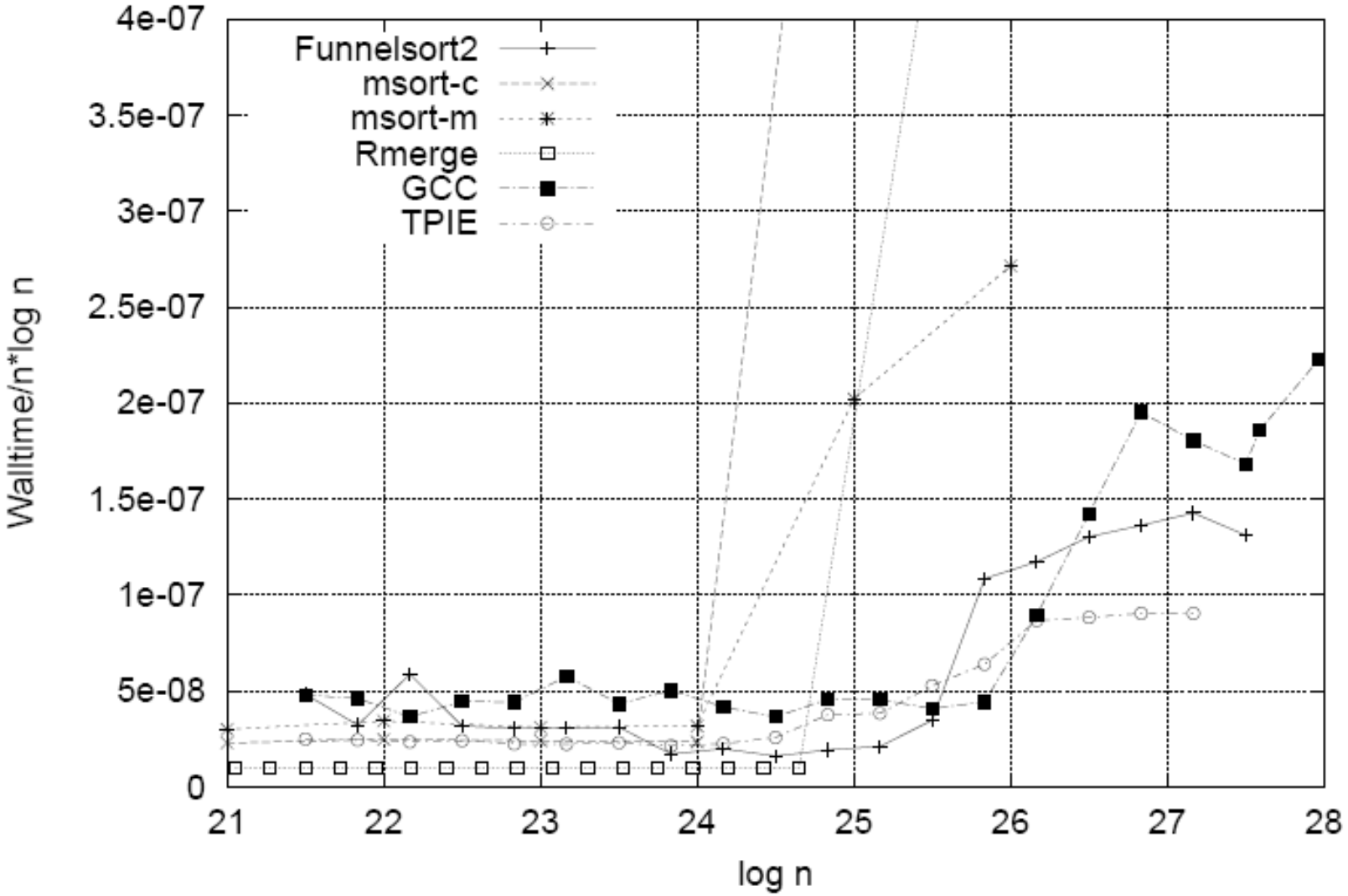
- Divide input in $N^{1/3}$ segments of size $N^{2/3}$
- Recursively **FunnelSort** each segment
- Merge sorted segments by an **$N^{1/3}$-merger**



- **Theorem** Provided $M \geq B^2$ performs optimal $O(\text{Sort}(N))$ I/Os

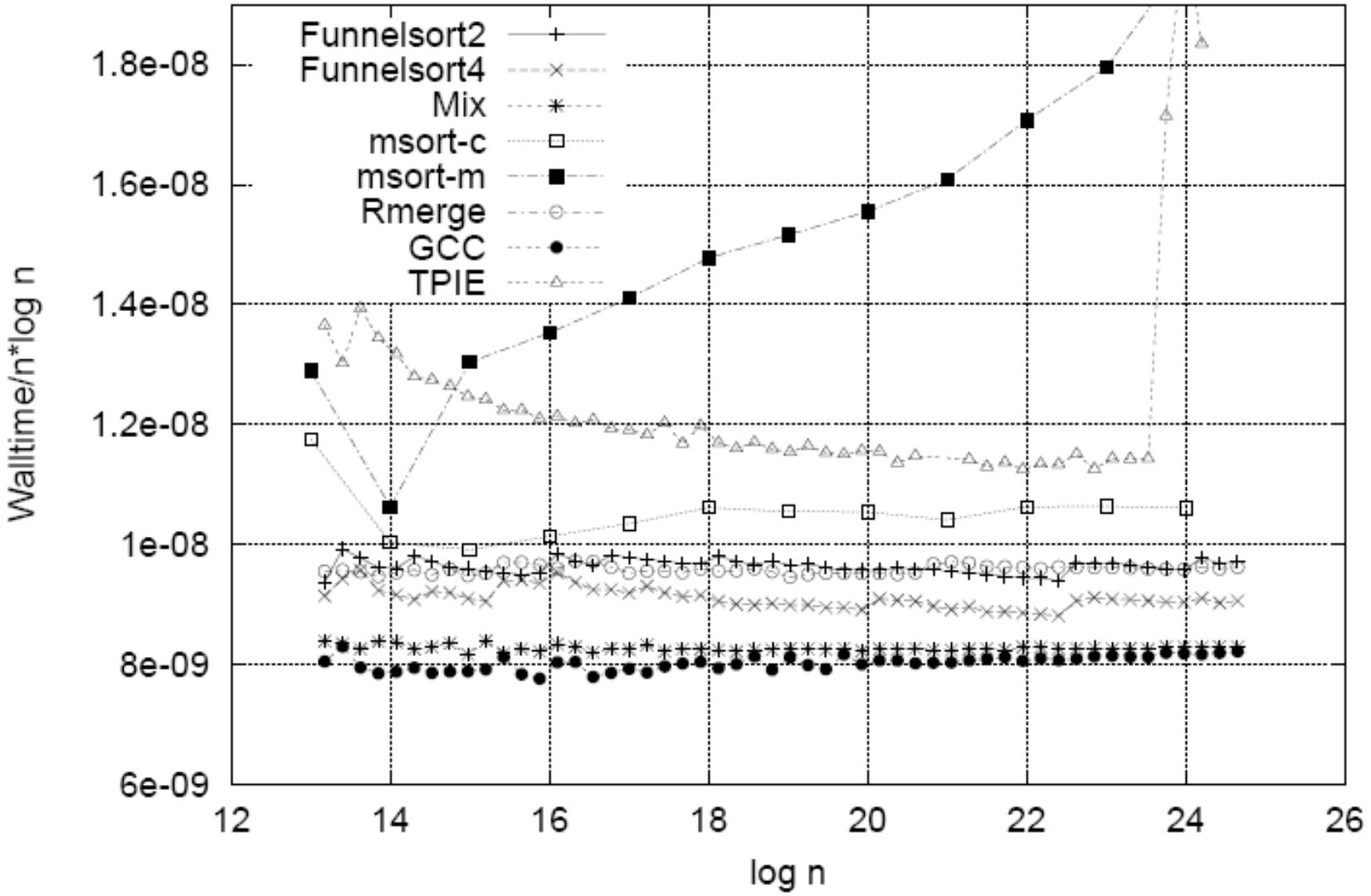
Sorting (Disk)

Brodal, Fagerberg, Vinther 2004



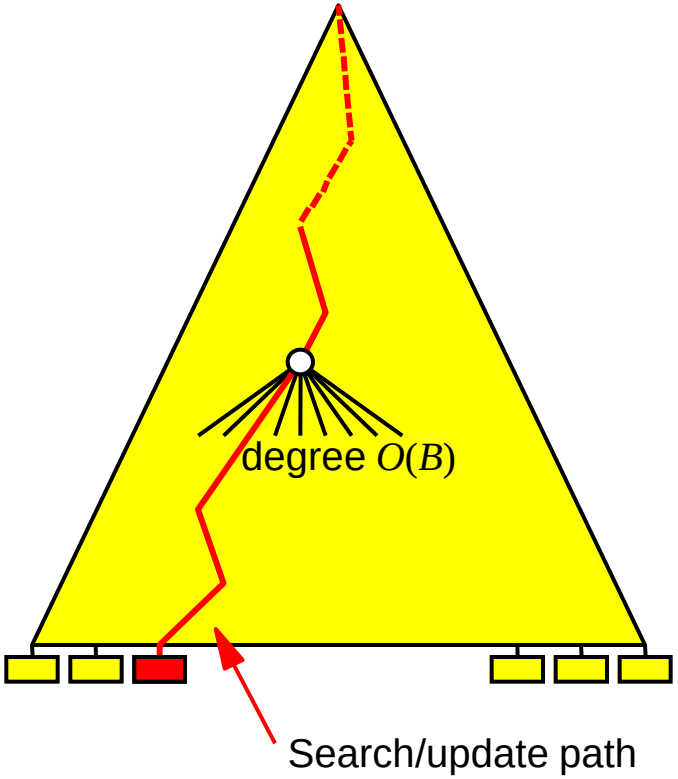
Sorting (RAM)

Brodal, Fagerberg, Vinther 2004



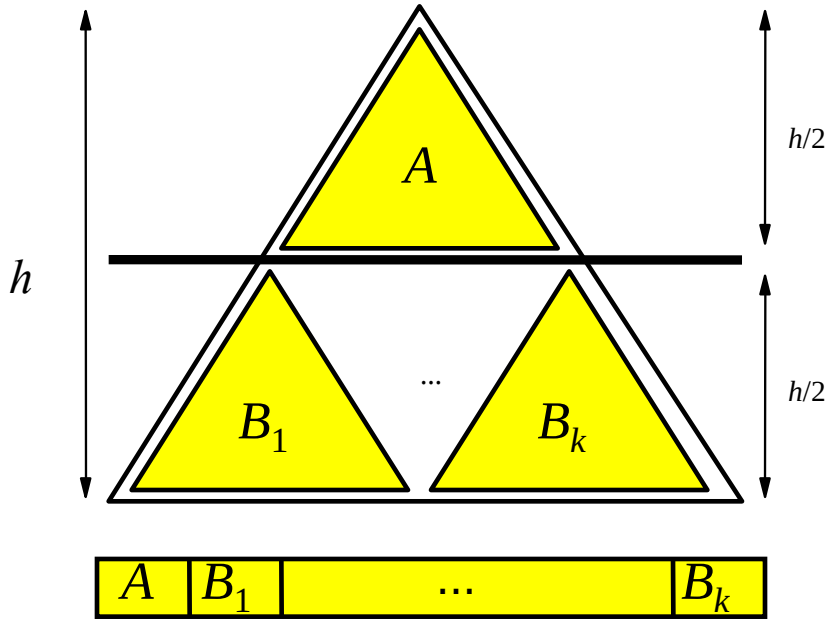
External Memory Search Trees

- B-trees Bayer and McCreight 1972
- Searches and updates use $O(\log_B N)$ I/Os

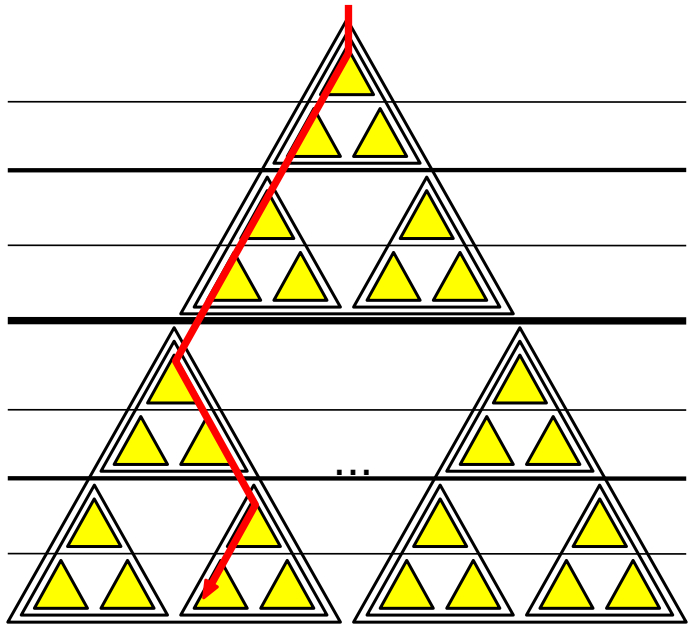


Cache-Oblivious Search Trees

- Recursive memory layout (van Emde Boas) Prokop 1999



Binary tree

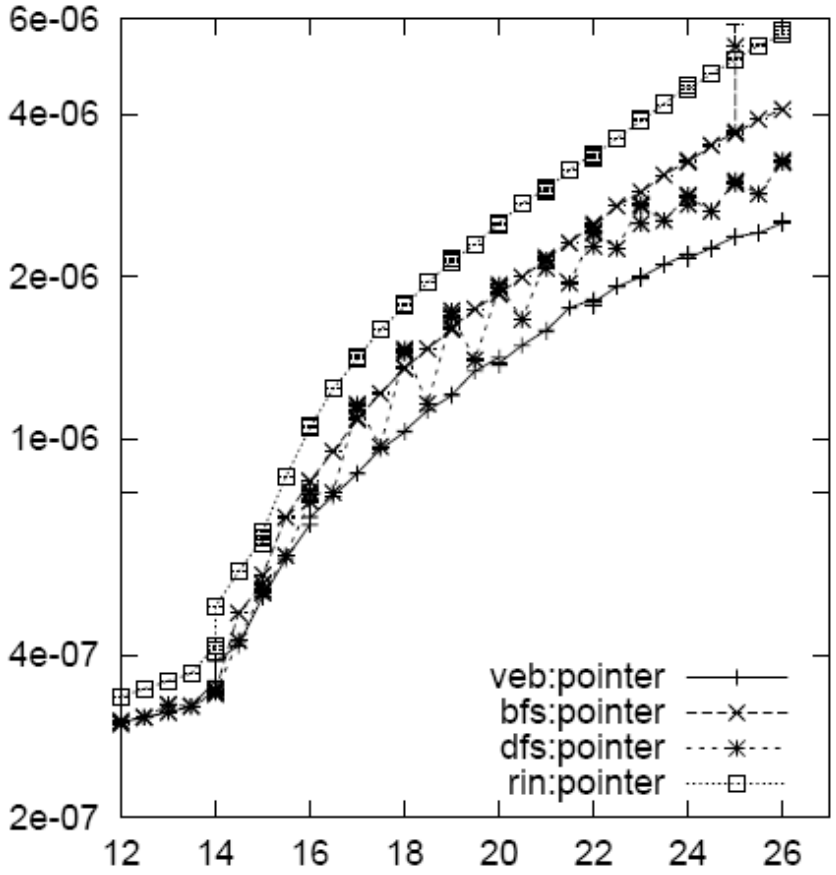


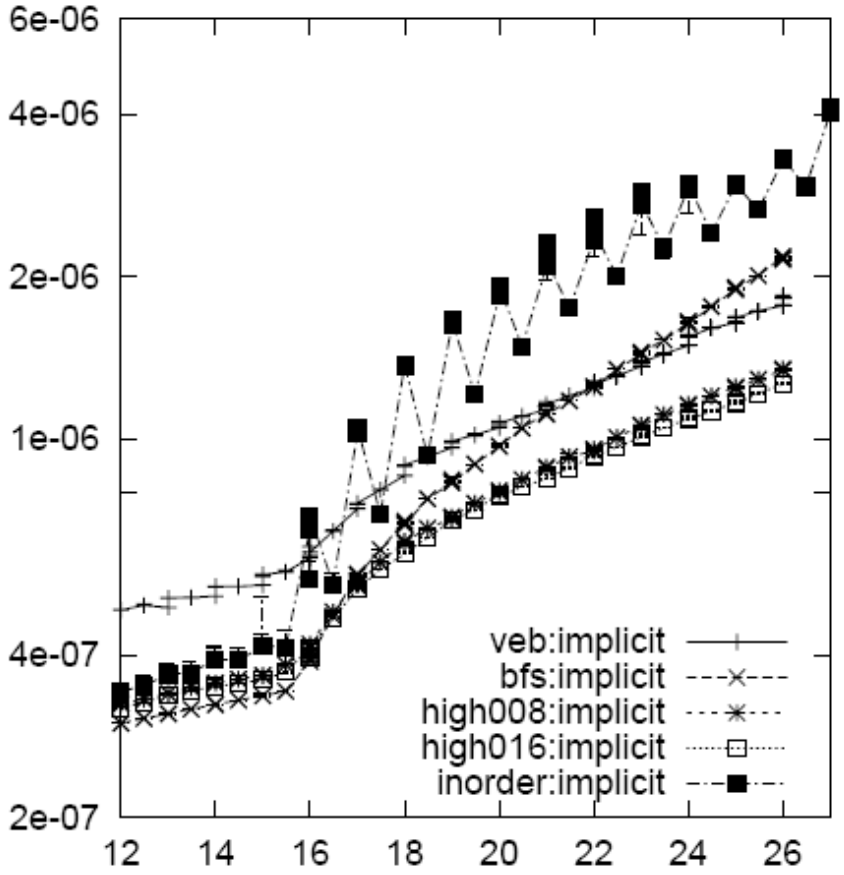
Searches use $O(\log_B N)$ I/Os

- Dynamization (several papers) – ”reduction to static layout”

Cache-Oblivious Search Trees

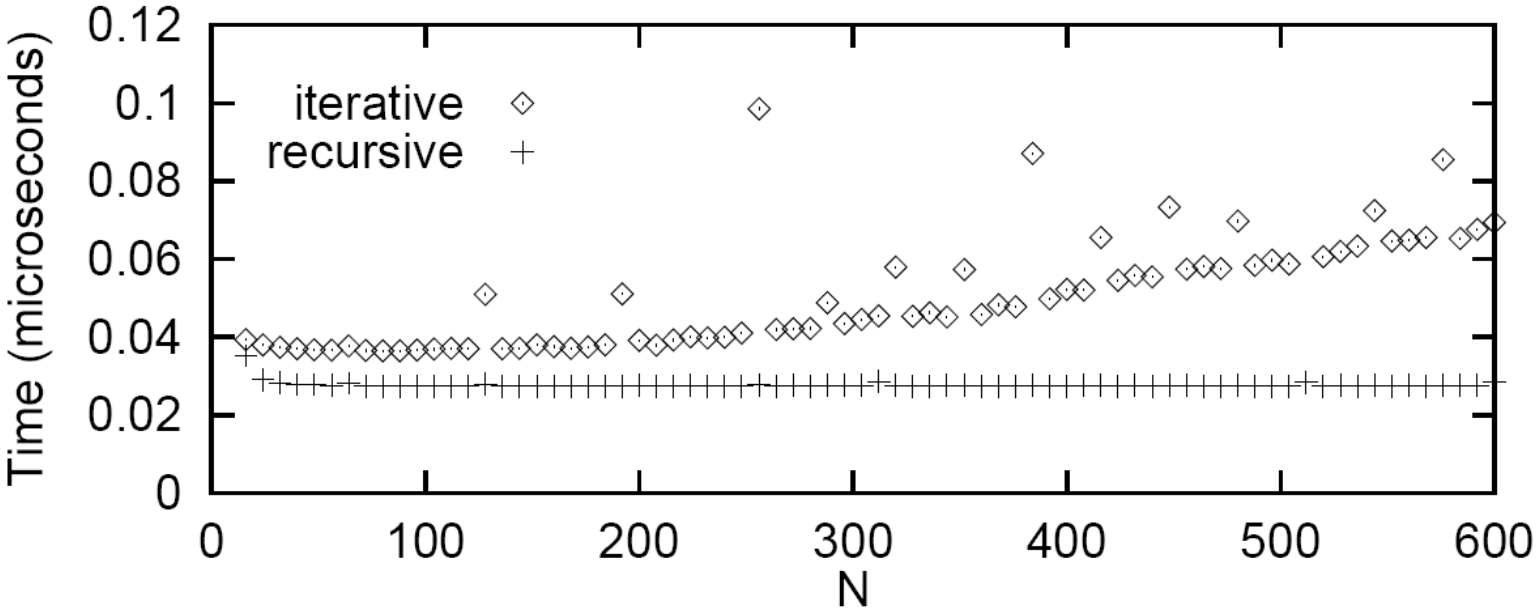
Brodal, Jacob, Fagerberg 1999





Matrix Multiplication

Frigo, Leiserson, Prokop and Ramachandran 1999



Average time taken to multiply two N x N matrics divided by N^3

Iterative: $O(N^3)$ I/O

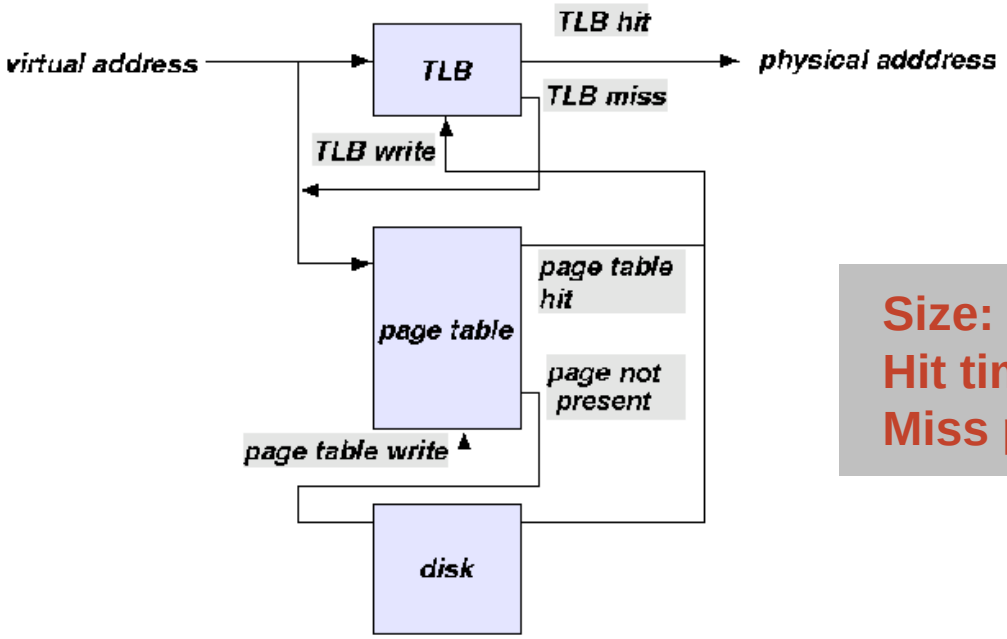
Recursive: $O\left(\frac{N^3}{B\sqrt{M}}\right)$ I/Os

The Influence of other Chip Technologies…

….why some experiments do not turn out as expected

Translation Lookaside Buffer (TLB)

- translate virtual addresses into physical addresses
- small table (full associative)
- TLB miss requires lookup to the page table

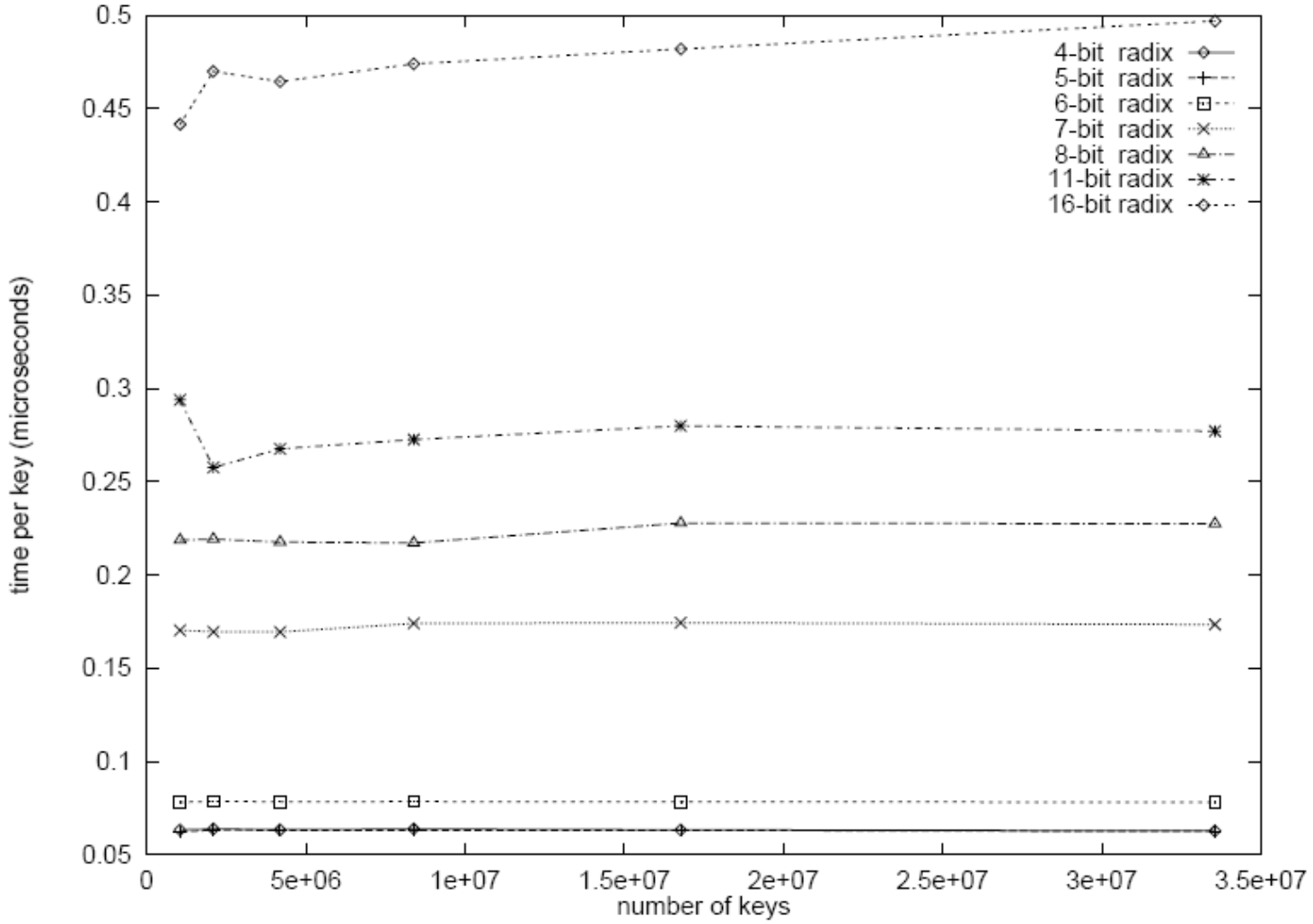


wikipedia.org

Size: 8 - 4,096 entries
Hit time: 0.5 - 1 clock cycle
Miss penalty: 10 - 30 clock cycles

TLB and Radix Sort

Rahman and Raman 1999



Time for one permutation phase

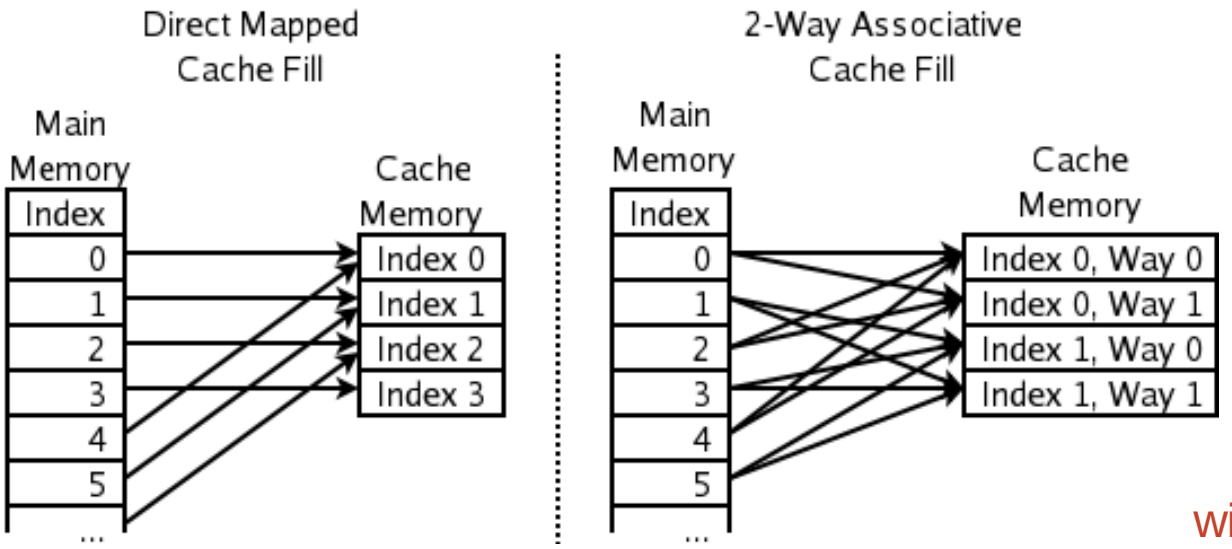
TLB and Radix Sort

Rahman and Raman 1999

Timings(sec)								
n	EPLSB11	PLSB11	EBT11	LSB556	LSB6	FLSB11	QSort	MSort
1M	**0.46**	*0.47*	0.54	0.57	0.64	0.90	0.70	1.02
2M	**0.89**	*0.92*	1.09	1.12	1.28	1.86	1.50	2.22
4M	**1.74**	*1.82*	2.20	2.20	2.56	3.86	3.24	4.47
8M	**3.53**	*3.64*	4.49	4.35	5.09	7.68	6.89	9.71
16M	**7.48**	*7.85*	8.81	8.57	10.22	15.23	14.65	19.47
32M	**14.96**	*15.66*	17.55	17.52	20.45	31.71	31.69	41.89

TLB optimized

Cache Associativity



wikipedia.org

Execution times for scanning k sequences of total length $N=2^{24}$ in round-robin fashion (SUN-Sparc Ultra, direct mapped cache)

k	1	2	4	8	16	32	64	128	256	512	1024
T	0.52	4.03	3.99	4.02	4.04	4.01	5.6	5.58	5.6	5.53	5.55

Cache associativity

TLB misses

Sanders 1999

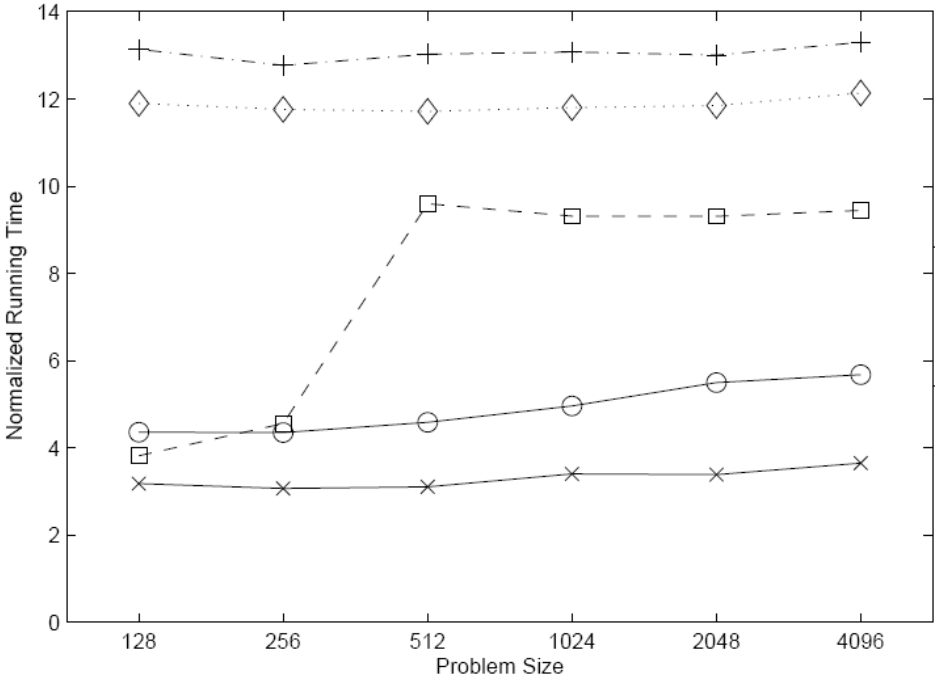
Prefetching vs. Caching

Pan, Cherng, Dick, Ladner 2007

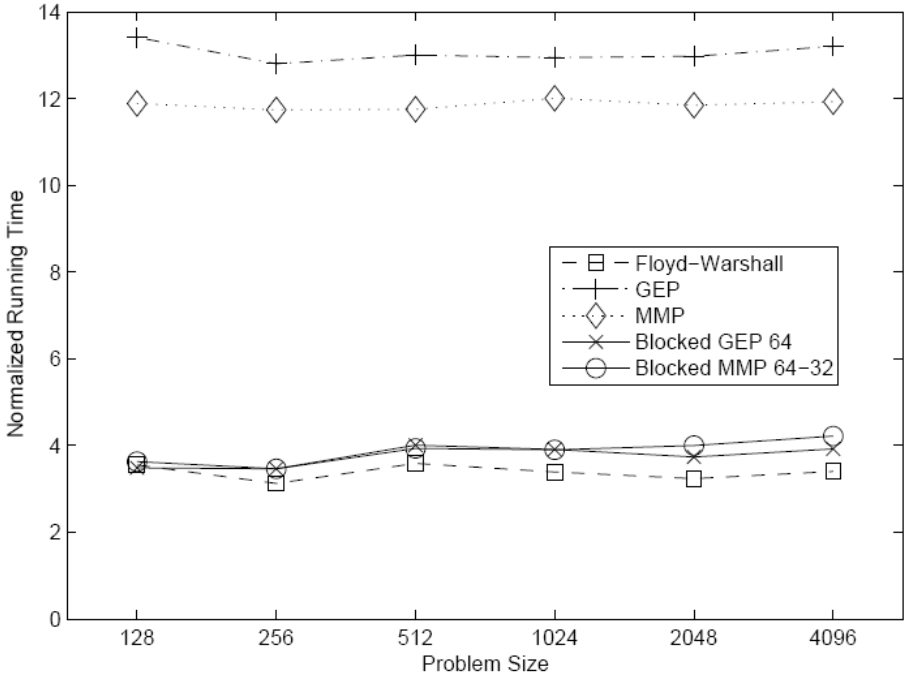
All Pairs Shortest Paths (APSP)

Organize data so that the CPU can prefetch the data

→ computation (can) dominate cache effects



Prefetching disabled



Prefetching enabled

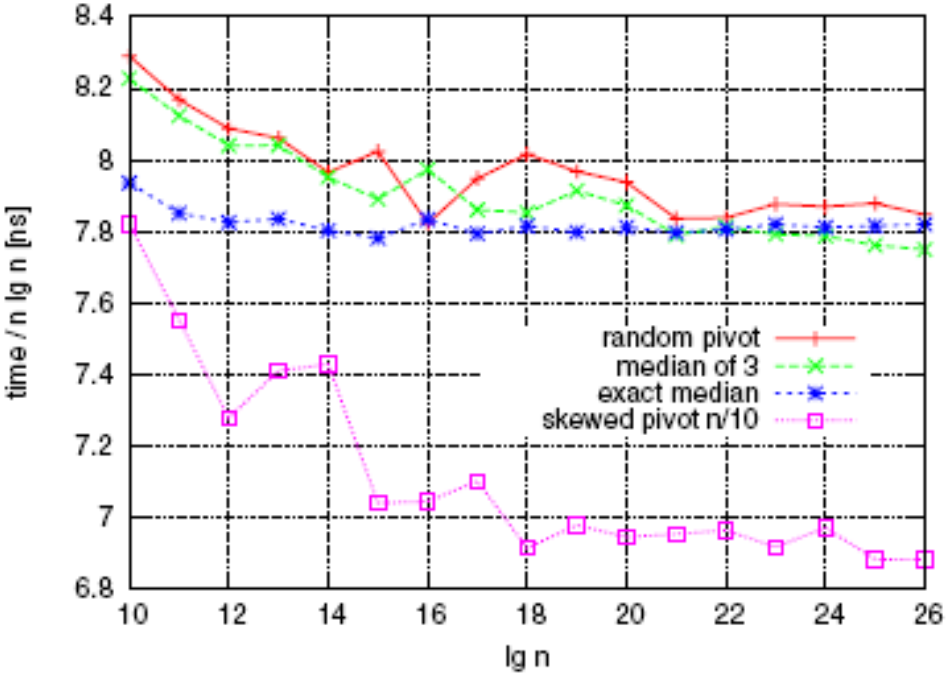
Branch Prediction vs. Caching

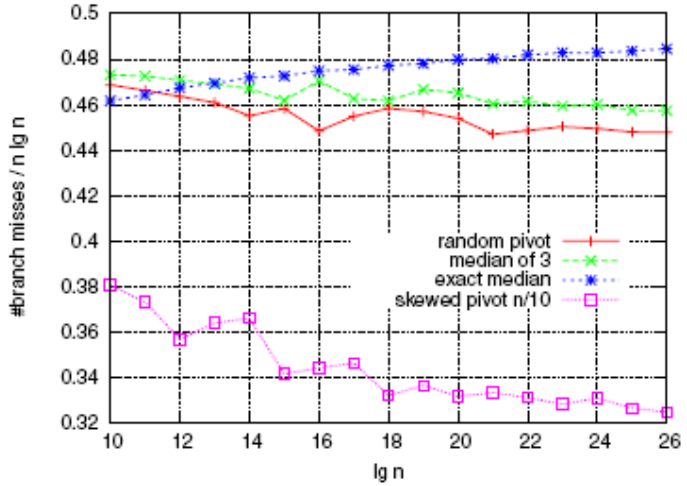
QuickSort Select the pivot biased to achieve subproblems of size α and 1-α

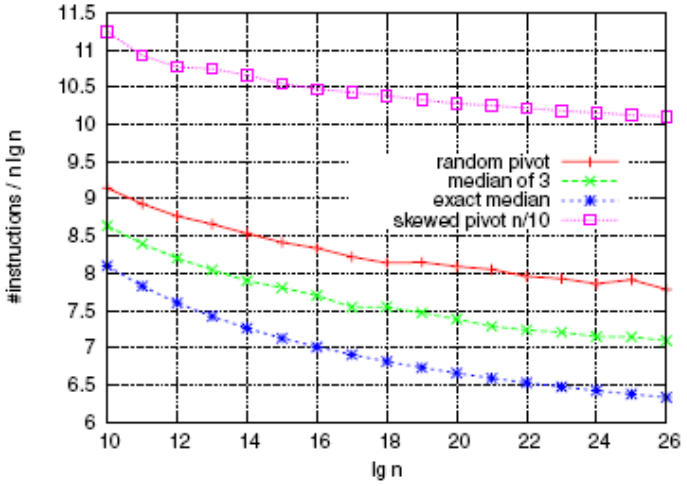
+ reduces # branch mispredictions

- increases # instructions and # cache faults

Kaligosi and Sanders 2006





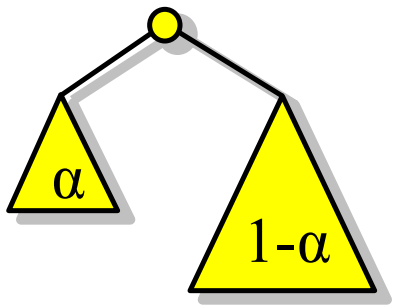


Branch Prediction vs. Caching

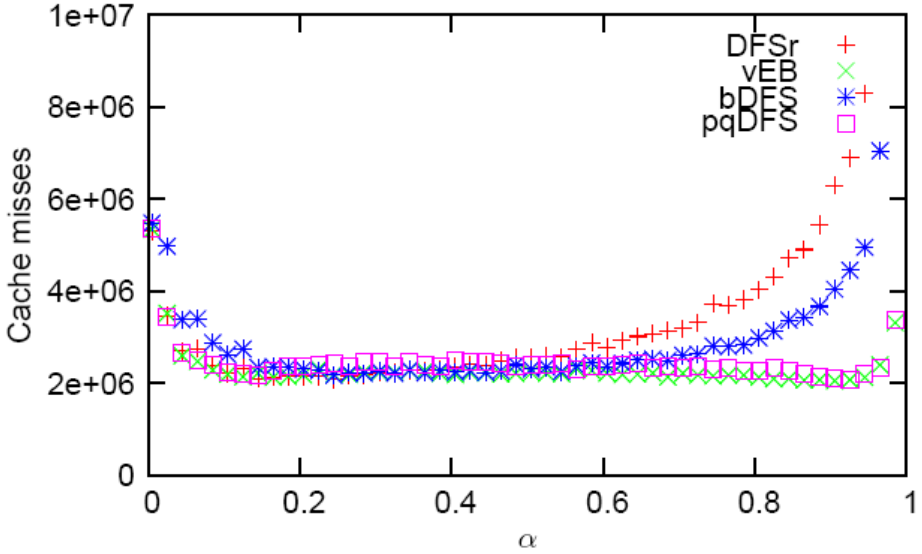
Skewed Search Trees Subtrees have size α and 1-α

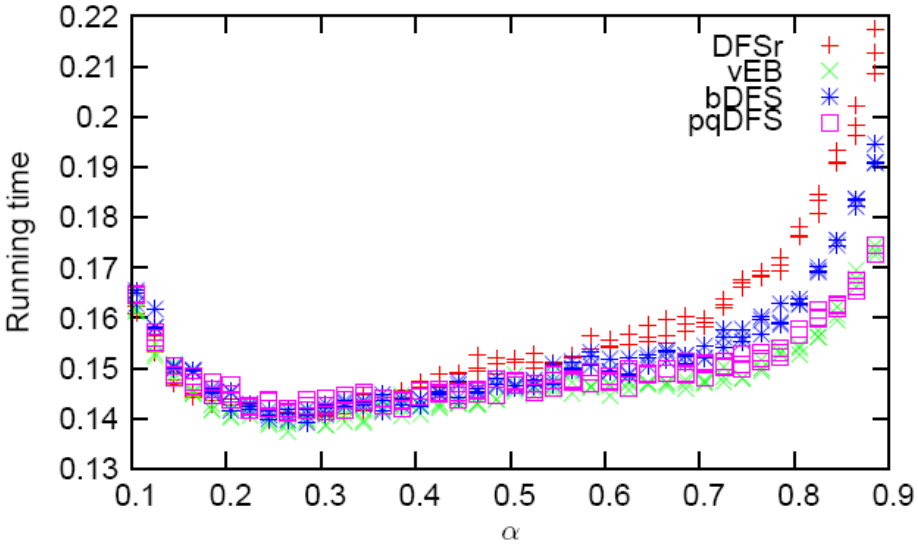
+ reduces # branch mispredictions

- increases # instructions and # cache faults



Brodal and Moruz 2006

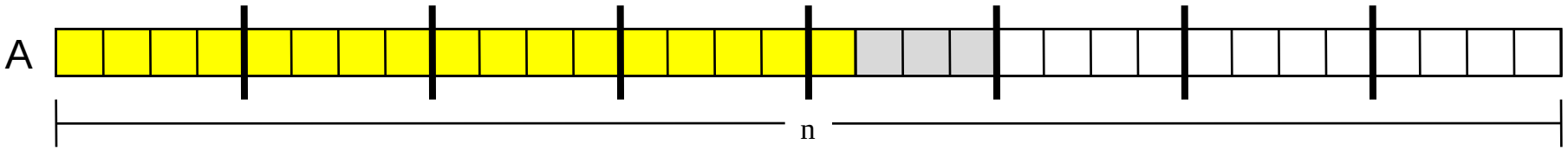




Another Trivial Program

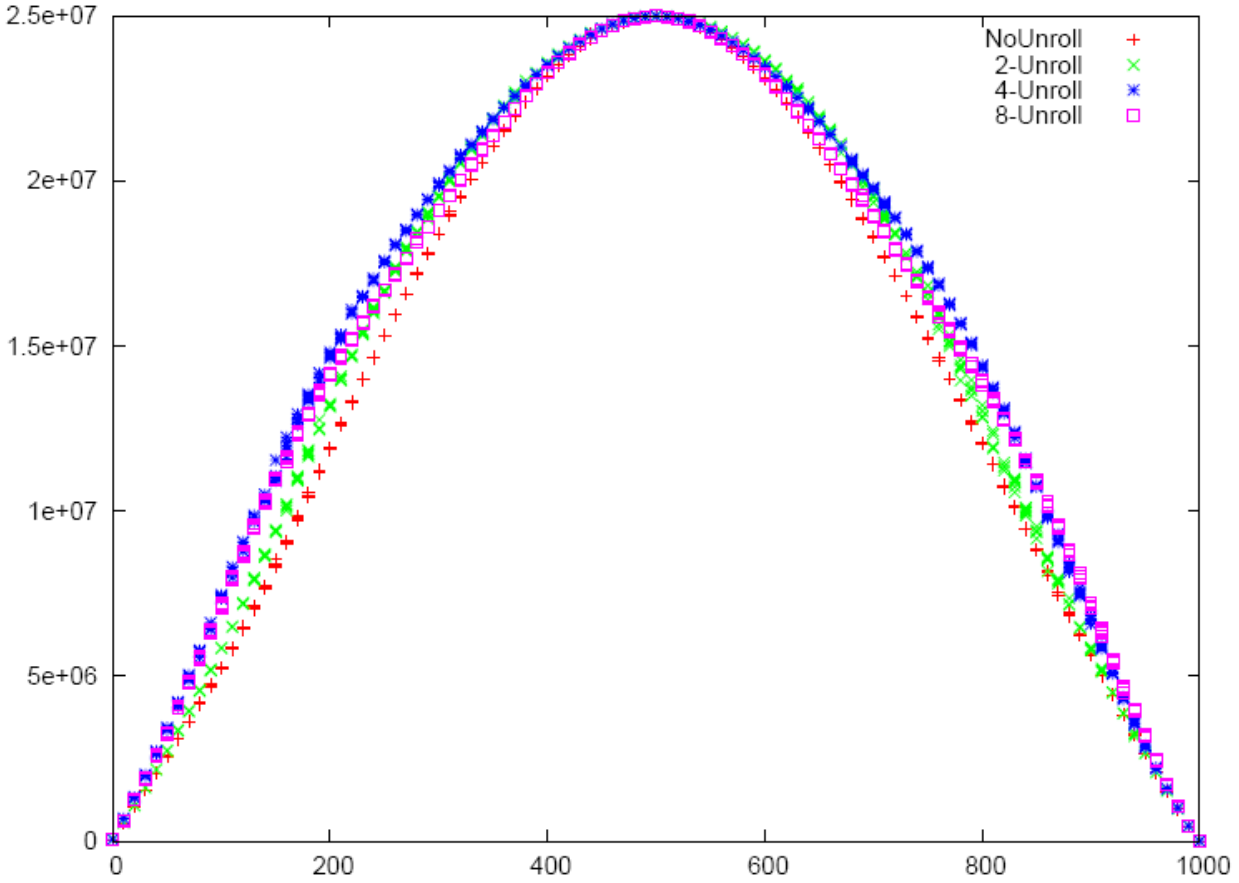
— the influence of branch predictions

```
for(i=0;i<n;i++)
  A[i]=rand()% 1000;

for(i=0;i<n;i++)
  if(A[i]>threshold)
    g++;
  else
    s++;
```



Branch Mispredictions

```
for(i=0;i<n;i++)
  a[i]=rand()% 1000;

for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```

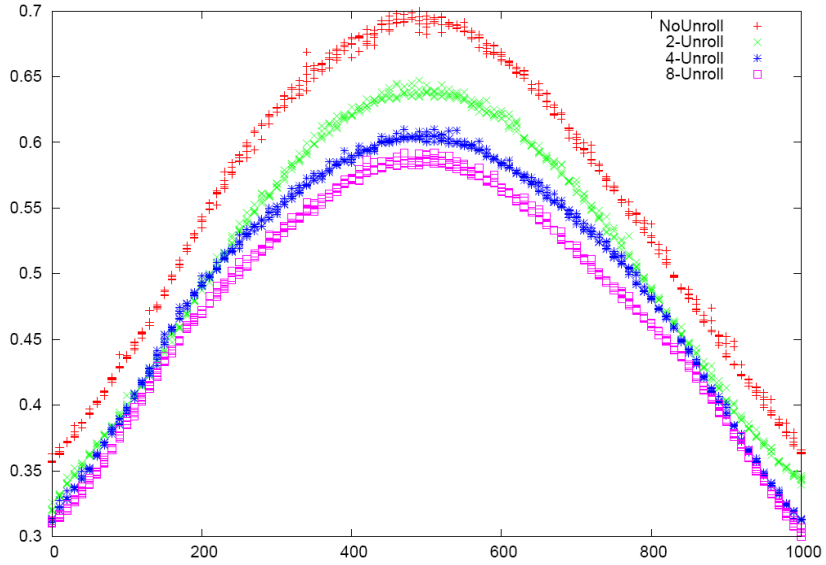


- Worst-case number of mispredictions for threshold=500

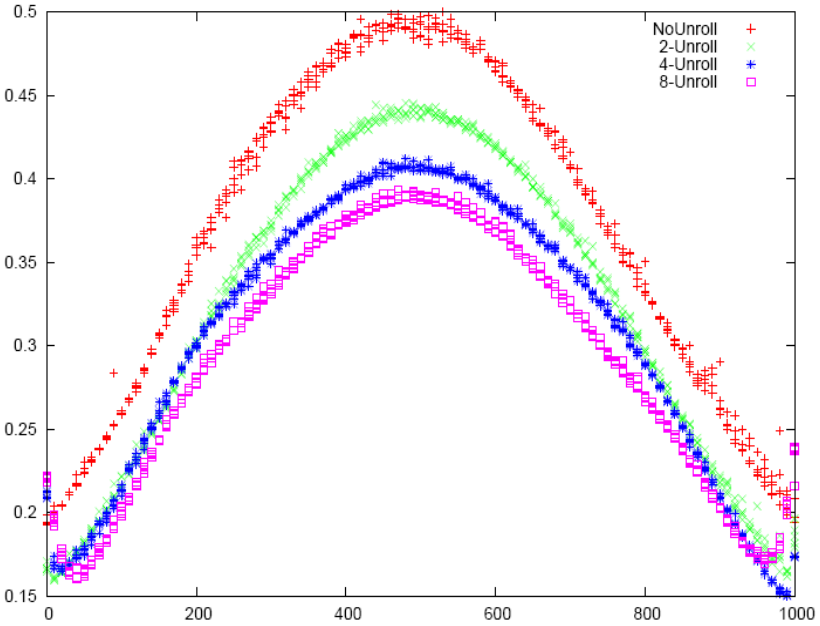
Running Time

```
for(i=0;i<n;i++)
  a[i]=rand()% 1000;

for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```

- Prefetching disabled
  → 0.3 - 0.7 sec
- Prefetching enabled
  → 0.15 – 0.5 sec



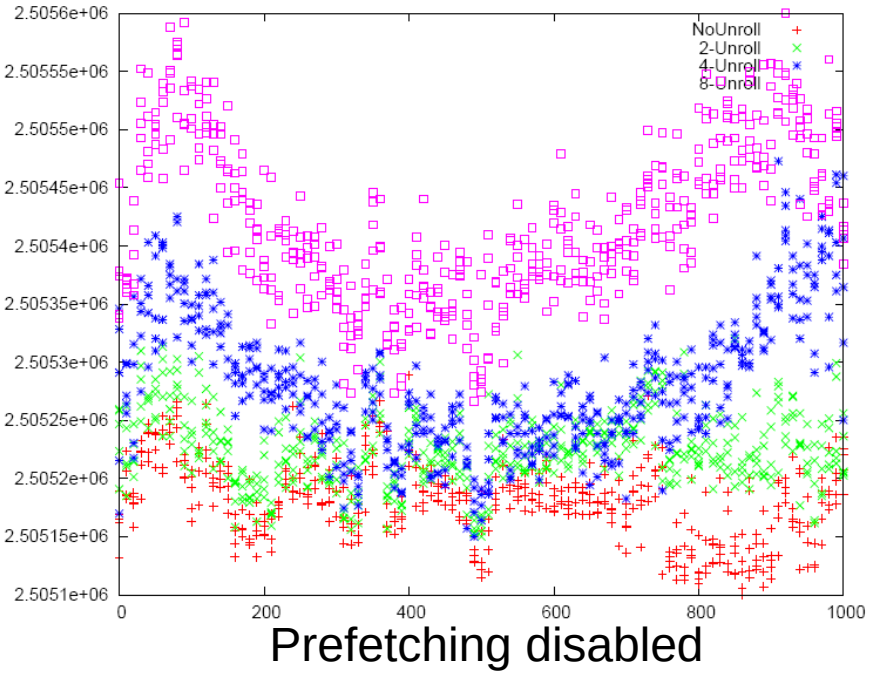
Prefetching disabled



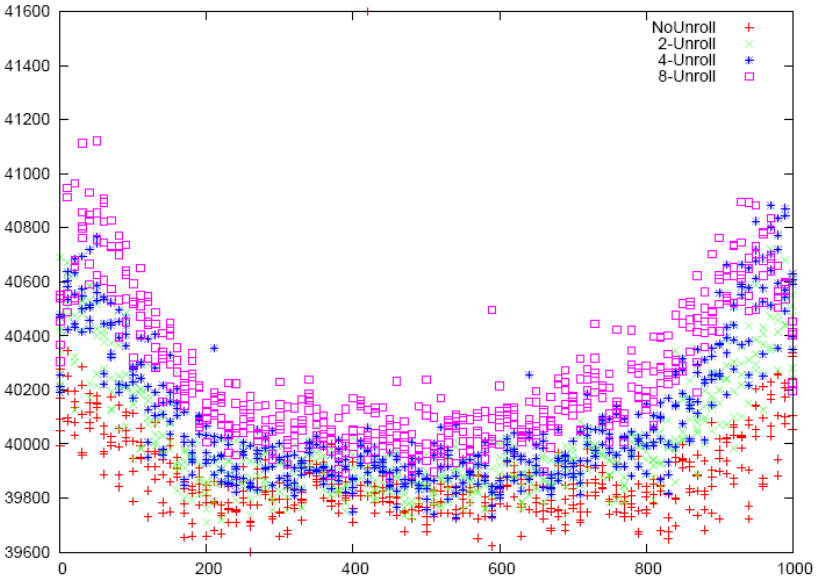
Prefetching enabled

L2 Cache Misses

```
for(i=0;i<n;i++)
  a[i]=rand()% 1000;

for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```



Prefetching disabled



Prefetching enabled

- Prefetching disabled
  → 2.500.000 cache misses
- Prefetching enabled
  → 40.000 cache misses

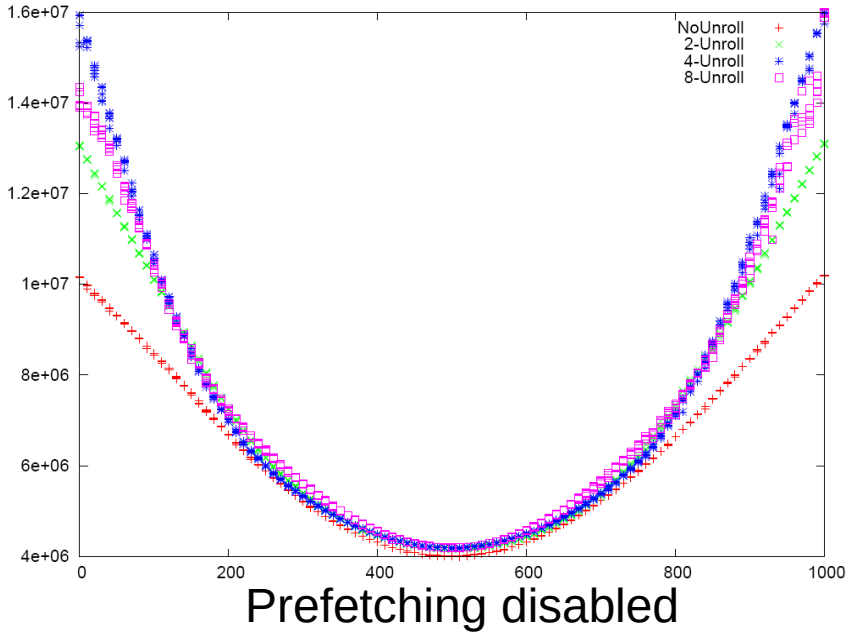
L1 Cache Misses

```
for(i=0;i<n;i++)
  a[i]=rand()% 1000;

for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```

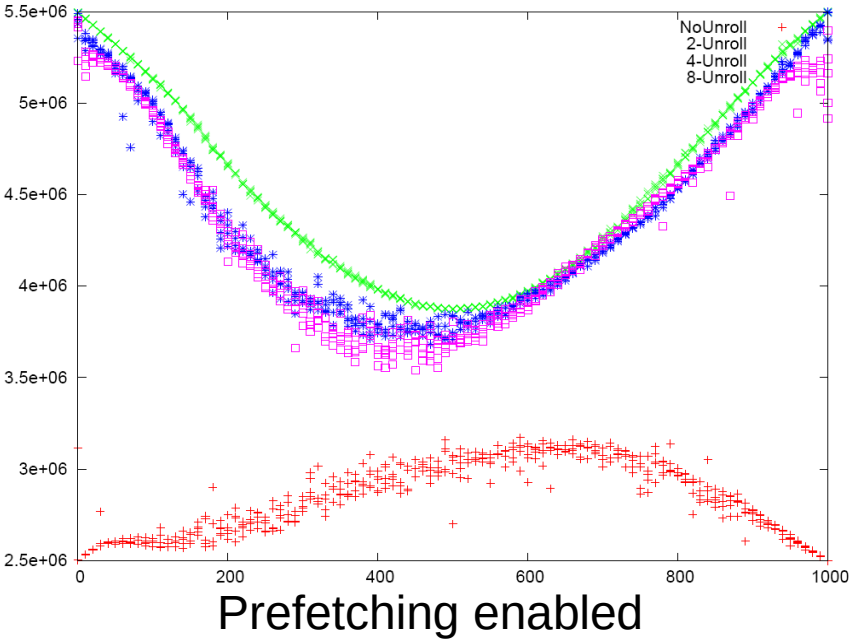
- Prefetching disabled

  → 4 – 16 x 10^6 cache misses
- Prefetching enabled

  → 2.5 – 5.5 x 10^6 cache misses



Prefetching disabled



Prefetching enabled

Summary

- **Be conscious about the presence of memory hierarcies when designing algorithms**
- Experimental results not often quite as expected due to neglected hardware features in algorithm design
- External memory model and cache-oblivious model adequate models for capturing disk/cache buttlenecks

What did I not talk about...

- non-uniform memory
- parallel disks
- parallel/distributed algorithms
- graph algorithms
- computational geometry
- string algorithms
- and a lot more...

THE END