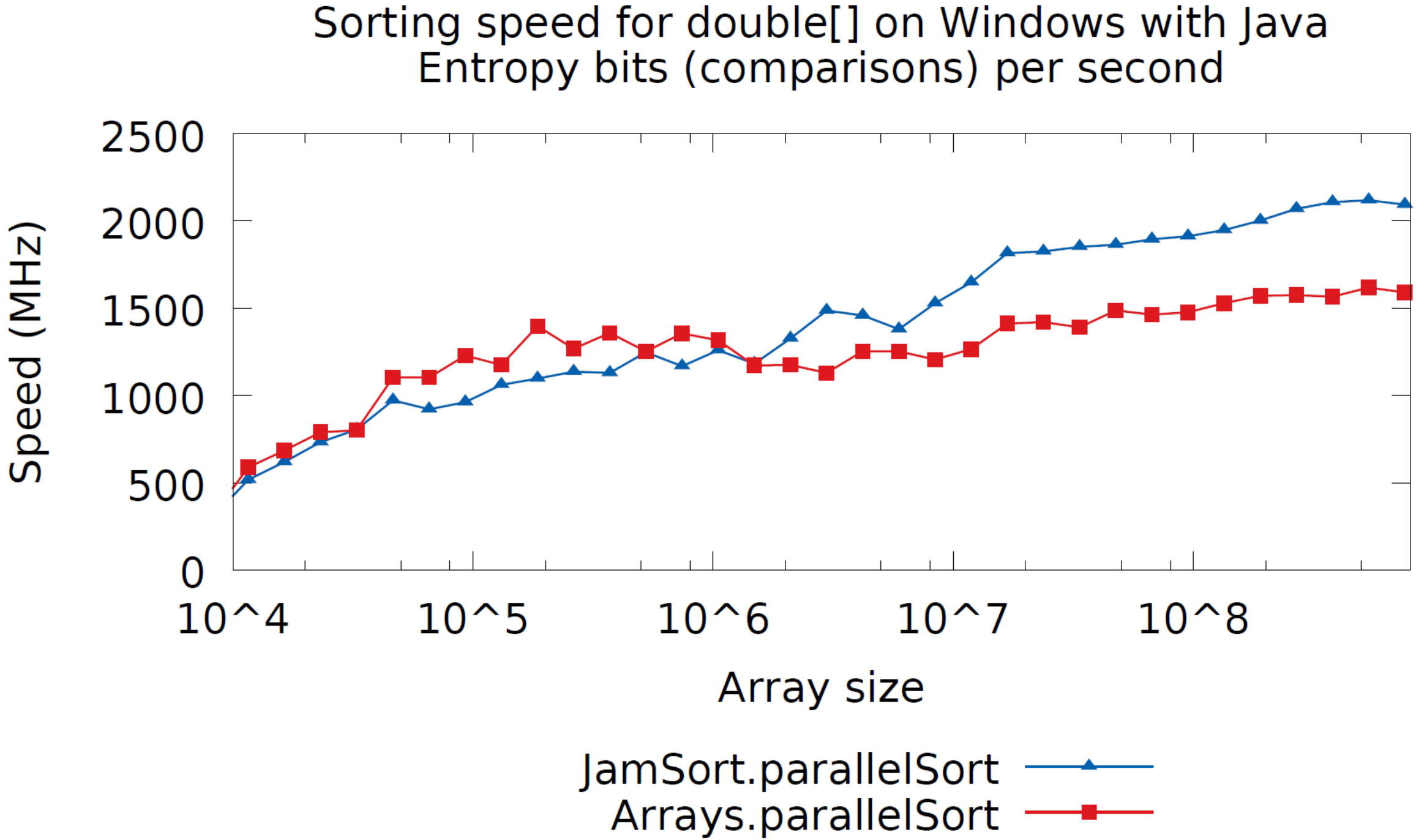
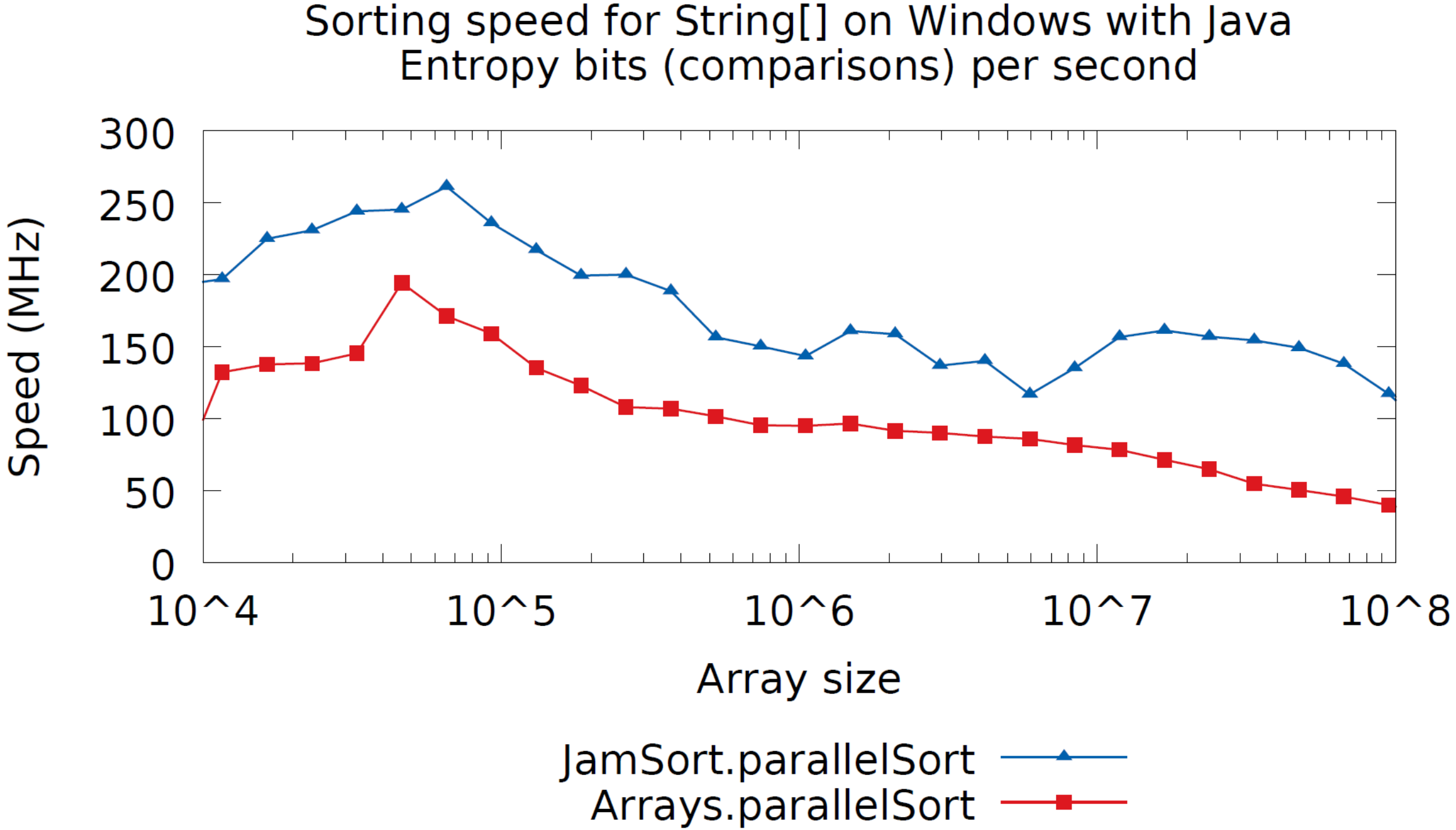
Samplesort

Úrtaksröðun: Röðun byggð á flokkun á grunni slembiúrtaks


Sorting speed for double[] on Windows with Java
Entropy bits (comparisons) per second
Speed (MHz)
2500
2000
1500
1000
500
0
10^4
10^5
10^6
10^7
10^8
Array size
JamSort.parallelSort
Arrays.parallelSort

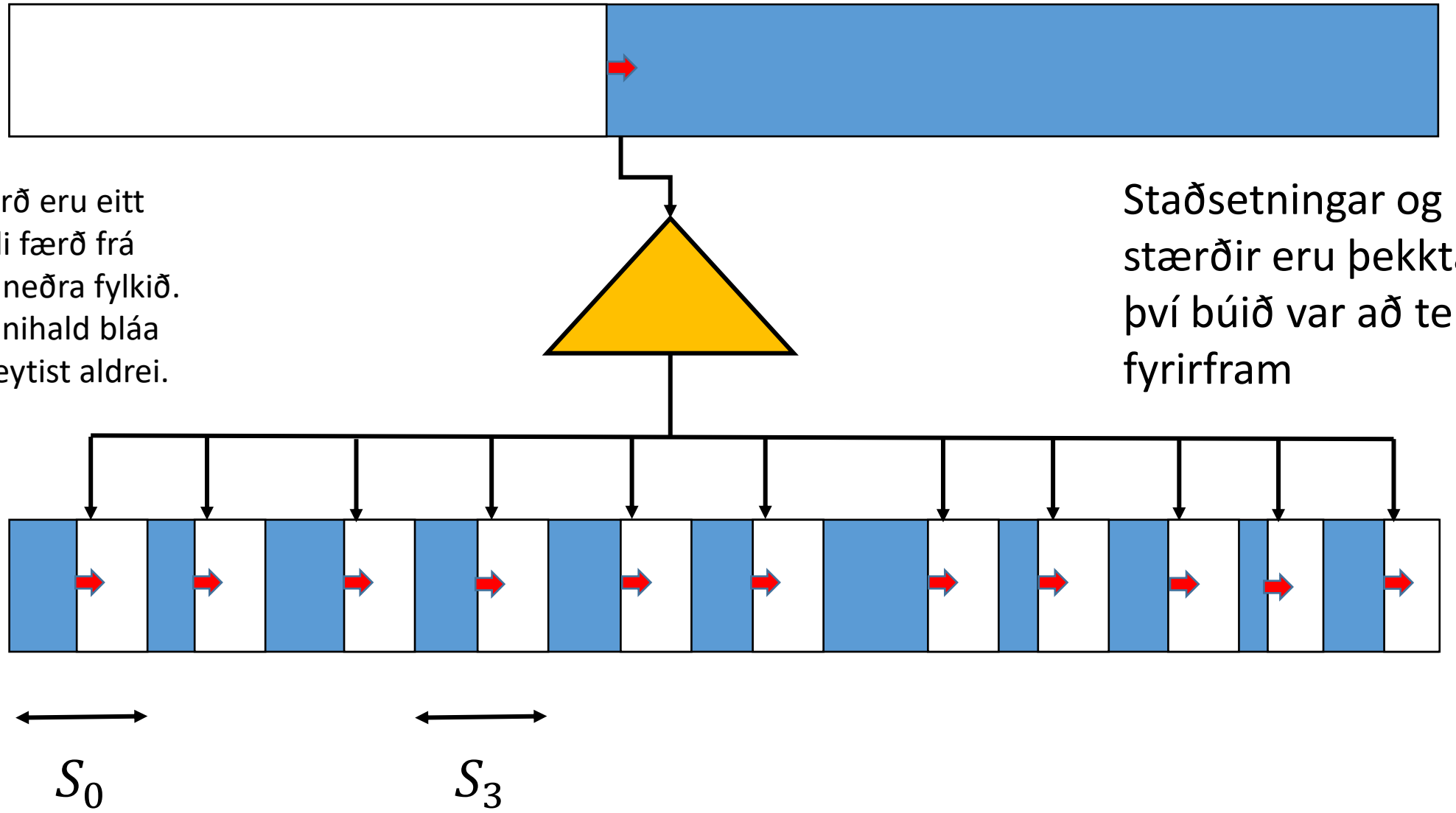

Sorting speed for String[] on Windows with Java
Entropy bits (comparisons) per second
Speed (MHz)
300
250
200
150
100
50
0
10^4
10^5
10^6
10^7
10^8
Array size
JamSort.parallelSort
Arrays.parallelSort

Grunnhugmynd samplesort

- Tökum (slembi)úrtak úr inntakspoka þeirra gilda sem raða skal
- Finnum vendigildi $p_1 \leq p_2 \leq \cdots \leq p_k$ í úrtakinu
- Flokkum gildin í inntakspokanum í poka $S_0, S_1, \ldots, S_k$ þannig að fyrir öll gildi $x_i \in S_i$ gildir $x_0 \leq p_1 \leq x_1 \leq p_2 \leq \cdots \leq p_k \leq x_k$
  - Þ.e. gildi í fremri poka eru $\leq$ gildi í aftari poka
- Röðum hverjum poka S_i og fáum runu R_i
- Skeytum saman rununum og fáum raðaða heildarrunu

$$R_0 + R_1 + \cdots + R_k$$

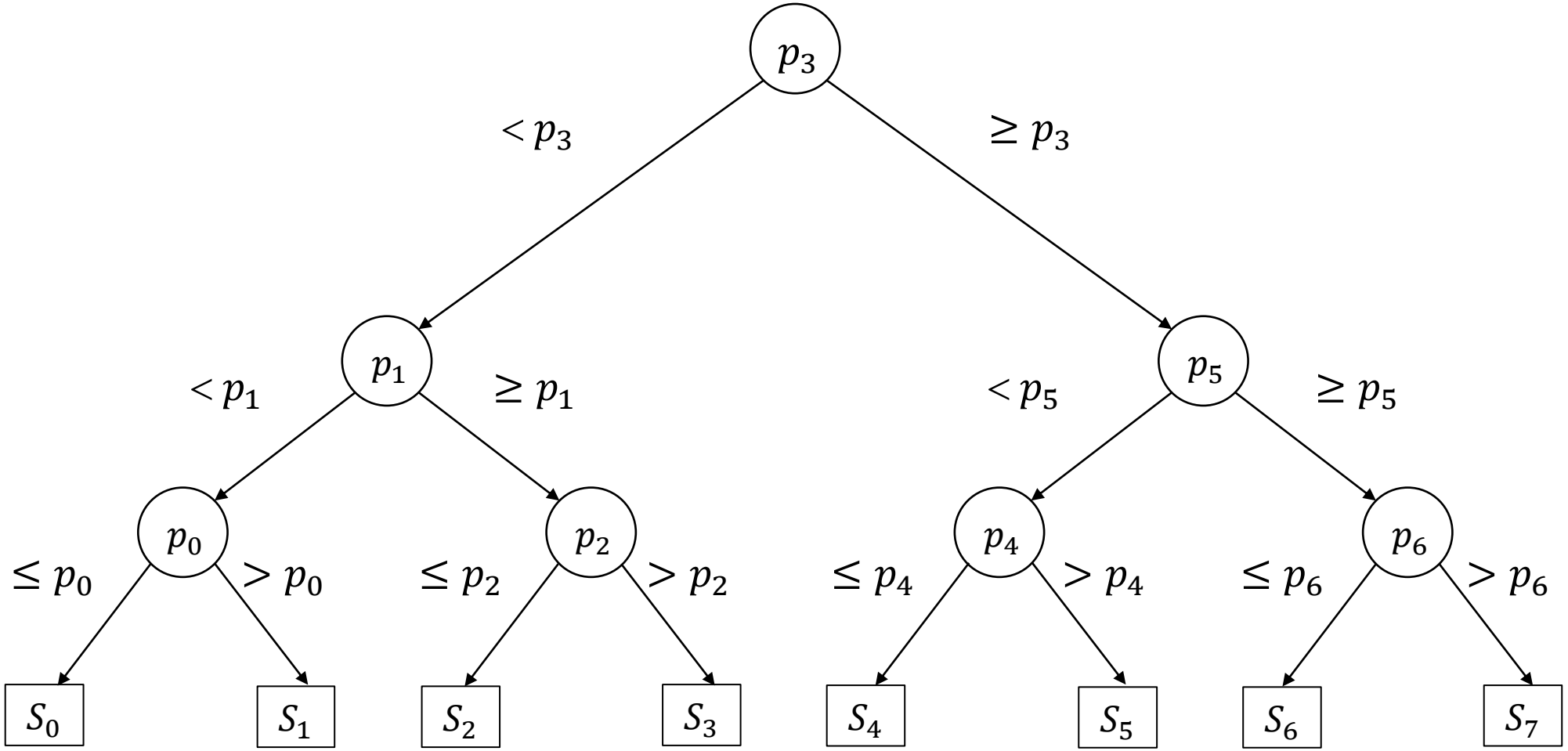
Samplesort flokkunarlykkja



Í hverri umferð eru eitt eða fleiri gildi færð frá efra fylkinu í neðra fylkið. Samanlagt innihald bláa svæðisins breytist aldrei.

Staðsetningar og stærðir eru þekktar því búið var að telja fyrirfram

Flokkun gilda með mörgum vendigildum



Fastayrðing flokkunarlykkju í Dafny

```
decreases x';
invariant r'[0] + r'[1] + r'[2] + r'[3] +
          r'[4] + r'[5] + r'[6] + r'[7] +
          x' == x;
invariant forall z | z in r'[0] :: z < p[0];
invariant forall z | z in r'[1] :: p[0] <= z <= p[1];
invariant forall z | z in r'[2] :: p[1] < z < p[2];
invariant forall z | z in r'[3] :: p[2] <= z <= p[3];
invariant forall z | z in r'[4] :: p[3] < z < p[4];
invariant forall z | z in r'[5] :: p[4] <= z <= p[5];
invariant forall z | z in r'[6] :: p[5] < z < p[6];
invariant forall z | z in r'[7] :: p[6] <= z;
```

Stofn flokkunarlykkjunnar í Dafny

```
var z :| z in x';
x' := x'-multiset{z};
var i := 0;
if p[3] < z    { i := i+4; }
if p[i+1] < z  { i := i+2; }
if p[i] <= z   { i := i+1; }
r'[i] := r'[i]+multiset{z};
```

Samskeiða samplesort: Röðun spilastokks

- Fjórir spilarar við sama borð vilja raða spilastokknum og skipta vinnunni jafnt
- Hver spilari fær 13 spil og flokkar þau í lauf, tigla, hjörtu og spaða
- Hver spilari fær síðan bunka af spilum í sama lit og raðar þeim
- Útkomunum er síðan staflað saman sem gefur raðaðan stokk

Kostir og gallar samplesort

Kostir

- Hraðvirkt fyrir mjög stór fylki
- Viðheldur fyrri röð (stable)
- Getur verið samskeiða (parallel)
- Getur verið „superscalar“
- Nýtir vel skyndiminni (cache)
- Ódýrt ef mörg gildi eru jöfn
- Nálgast bestu mögulega meðaltímaflækju
- Til þess þarf úrtakið að vera slembið og margfeldi af fjölda vendigilda

Gallar

- Hægvirkt fyrir lítil fylki
- Græðir ekki á fyrri röð
- Krefst hjálparfylkis
- Hugsanleg óheppileg vendigildi
  - Mjög ólíklegt ef úrtakið er stórt

Hvað er „superscalar“?

- Nútíma örgjörvar geta framkvæmt fleiri en eina vélarmálsskipun samtímis í sama þræði
- Til þess að það gerist þarf kjarninn að sannreyna að engin skipananna þurfi úttak úr annari skipun
- Í samplesort er hægt að flokka fleiri en eitt gildi samtímis því niðurstöður samanburða eru óháðar
- Þá er í raun verið að keyra margar óháðar helmingunarleitir samtímis
- Þetta hraðar samplesort verulega þegar unnið er með frumstæð gildi svo sem int og double

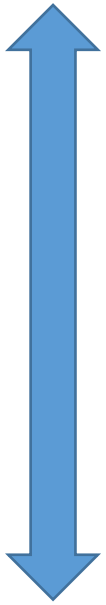
Dæmi um „superscalar“ flokkunarlykkju

```
while( n!=1 )
{
    // | <xK | ?? | >=xK |
    //          ^    ^
    //          iK   iK+n
    // n>0 is of form 2^k-1
    n >>= 1;
    np = n+1;
```

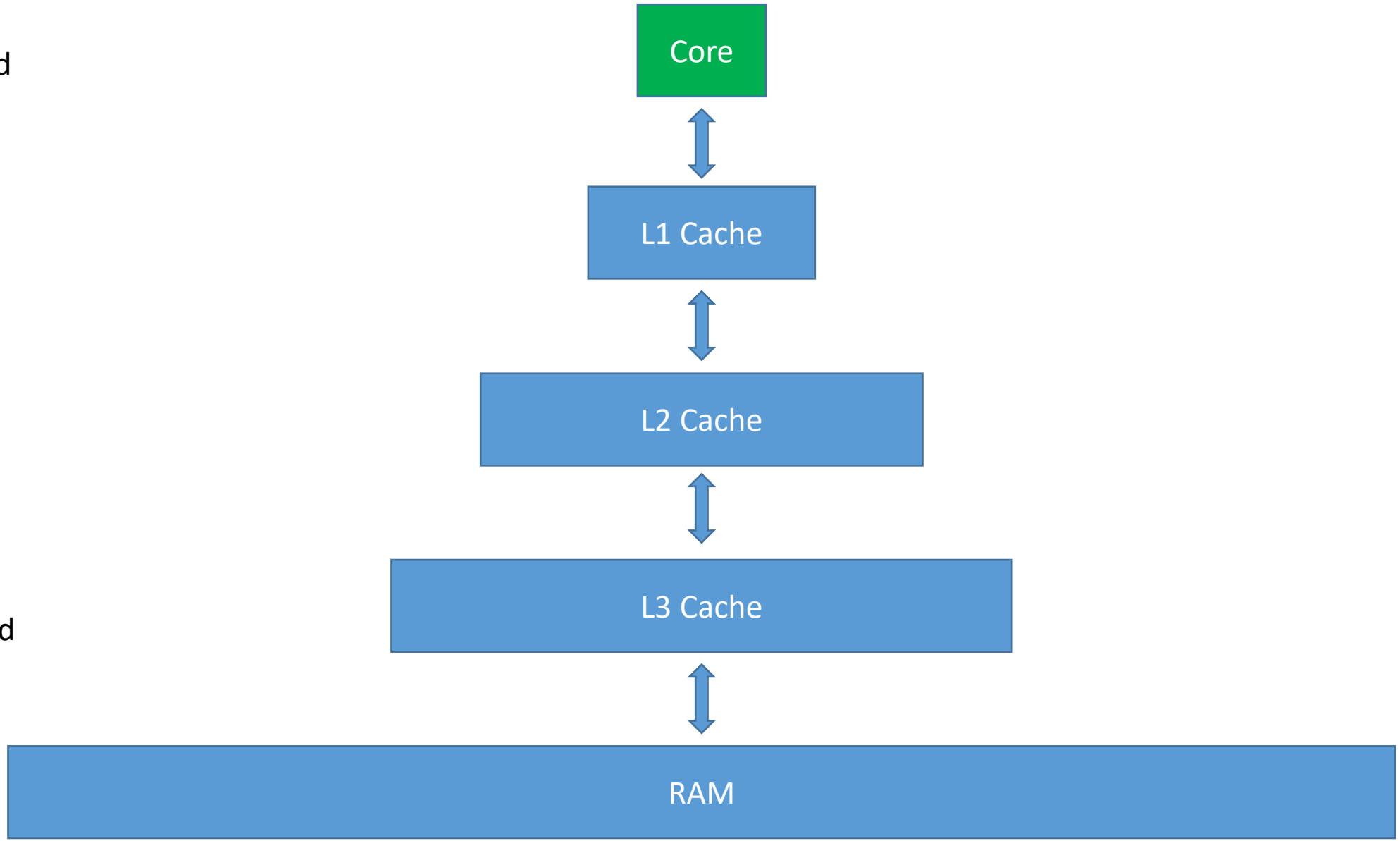
```
    i0 += p[i0+n]<x0?np:0;
    i1 += p[i1+n]<x1?np:0;
    i2 += p[i2+n]<x2?np:0;
    i3 += p[i3+n]<x3?np:0;
    i4 += p[i4+n]<x4?np:0;
    i5 += p[i5+n]<x5?np:0;
    i6 += p[i6+n]<x6?np:0;
    i7 += p[i7+n]<x7?np:0;
}
```

Skyndiminni (cache)

Meiri bandbreidd
Styttri biðtími
Minna pláss



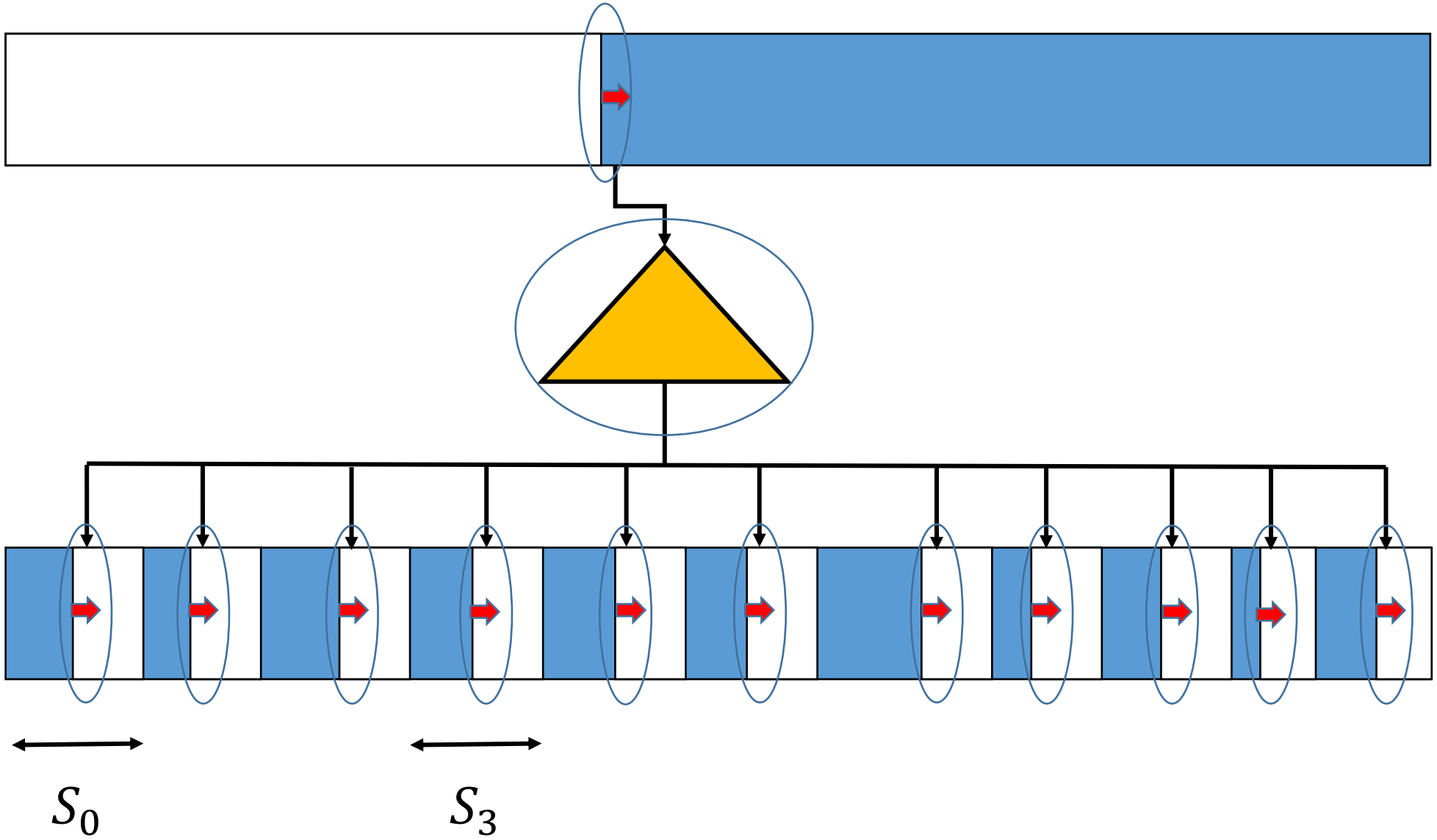
Minni bandbreidd
Lengri biðtími
Meira pláss



Skyndiminniskær forrit (cache friendly)

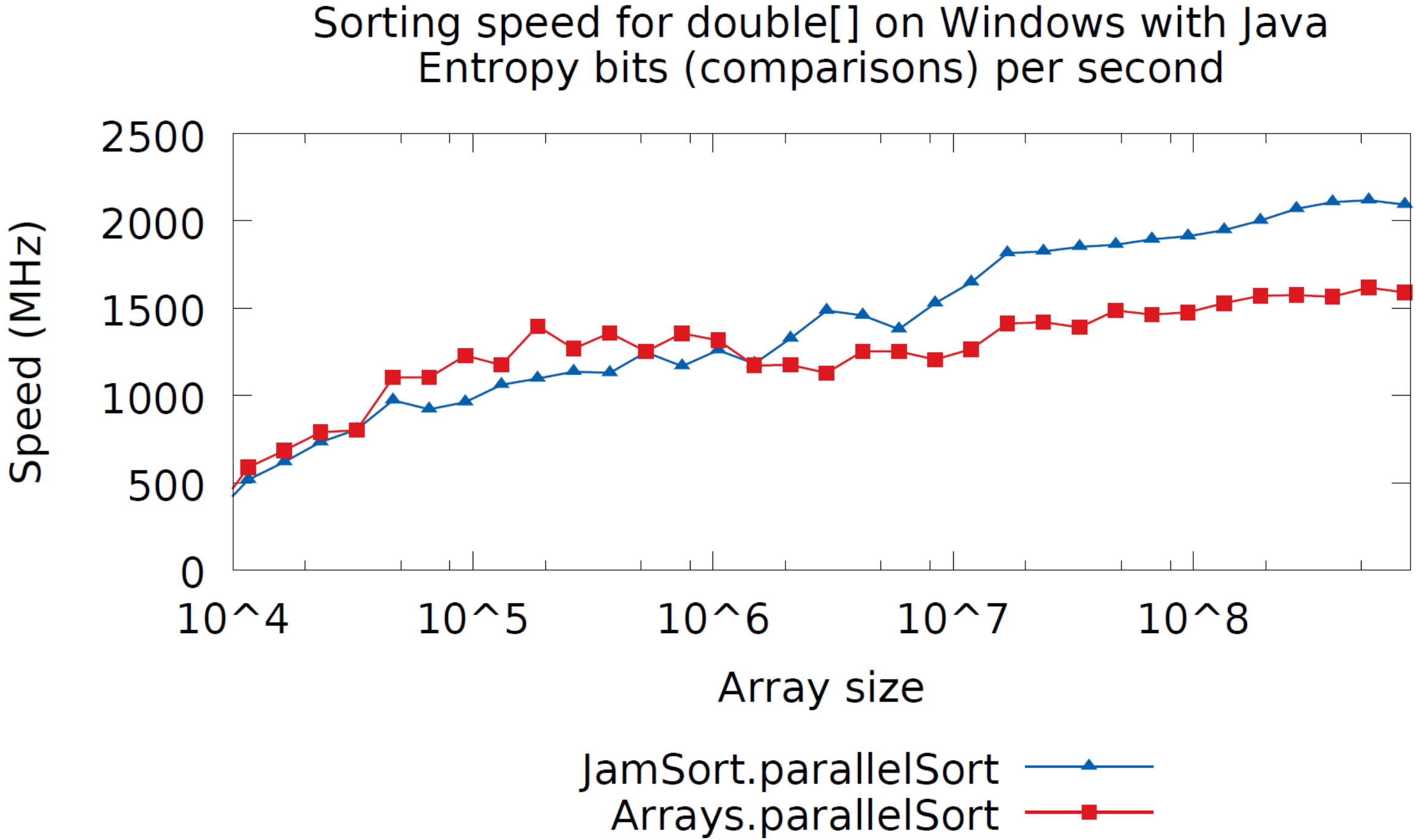
- Skyndiminnisfótsporið skal vera lítið
  - Fótsporið (cache footprint) er sá hluti RAM sem kjarninn þarf aðgang að á hverjum tíma til að halda útreikningum áfram
- Skal vera fyrirsjáanlegt (predictable) hvernig fótsporið breytist
- Forðast skal að potast í sama svæði í RAM oft með löngu millibili

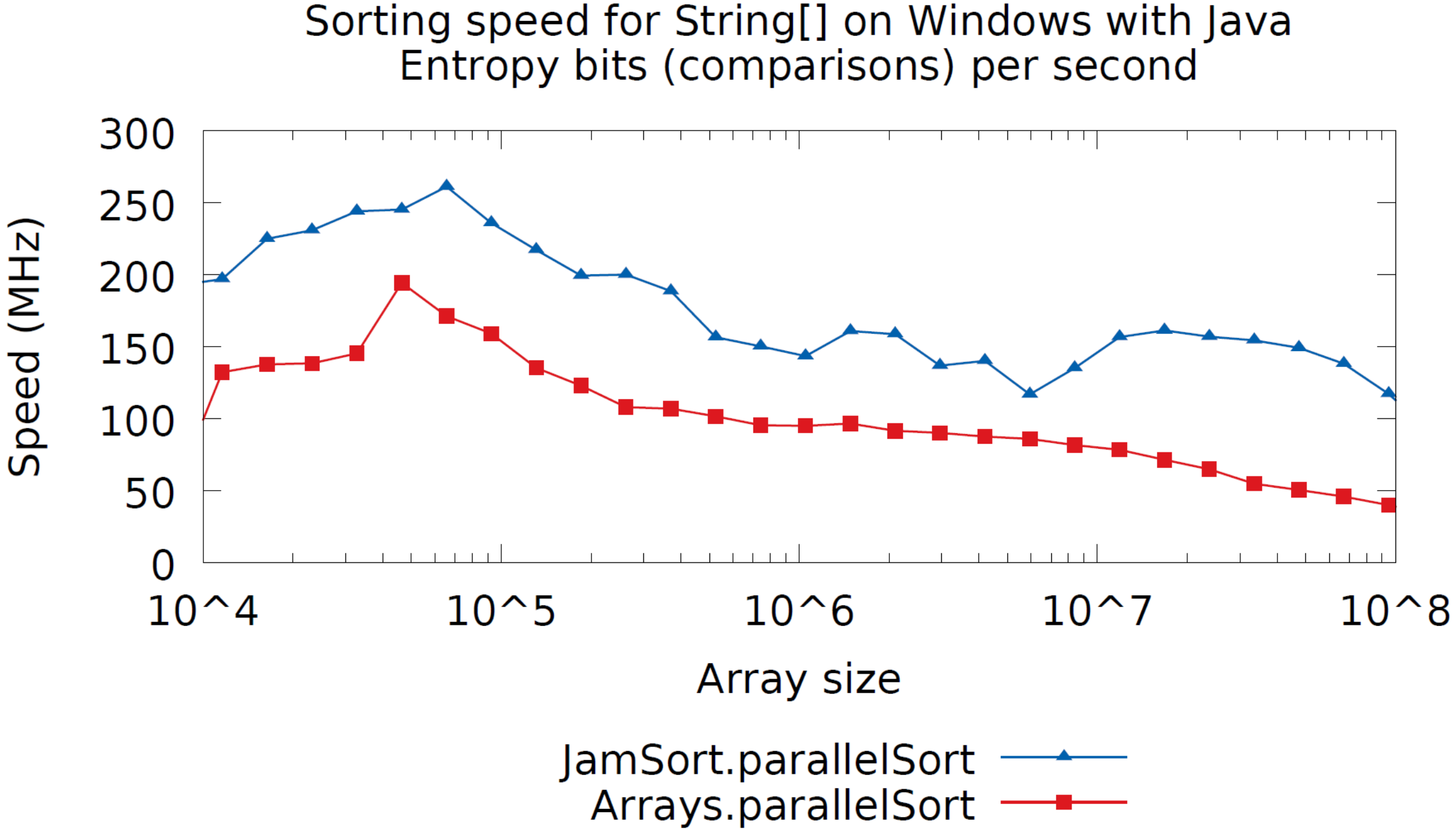
Fótspor samplesort í skyndiminni
Cache Footprint



Samplesort

A sorting method based on classifying values based on a random sample


Sorting speed for double[] on Windows with Java
Entropy bits (comparisons) per second
Speed (MHz)
2500
2000
1500
1000
500
0
10^4
10^5
10^6
10^7
10^8
Array size
JamSort.parallelSort
Arrays.parallelSort

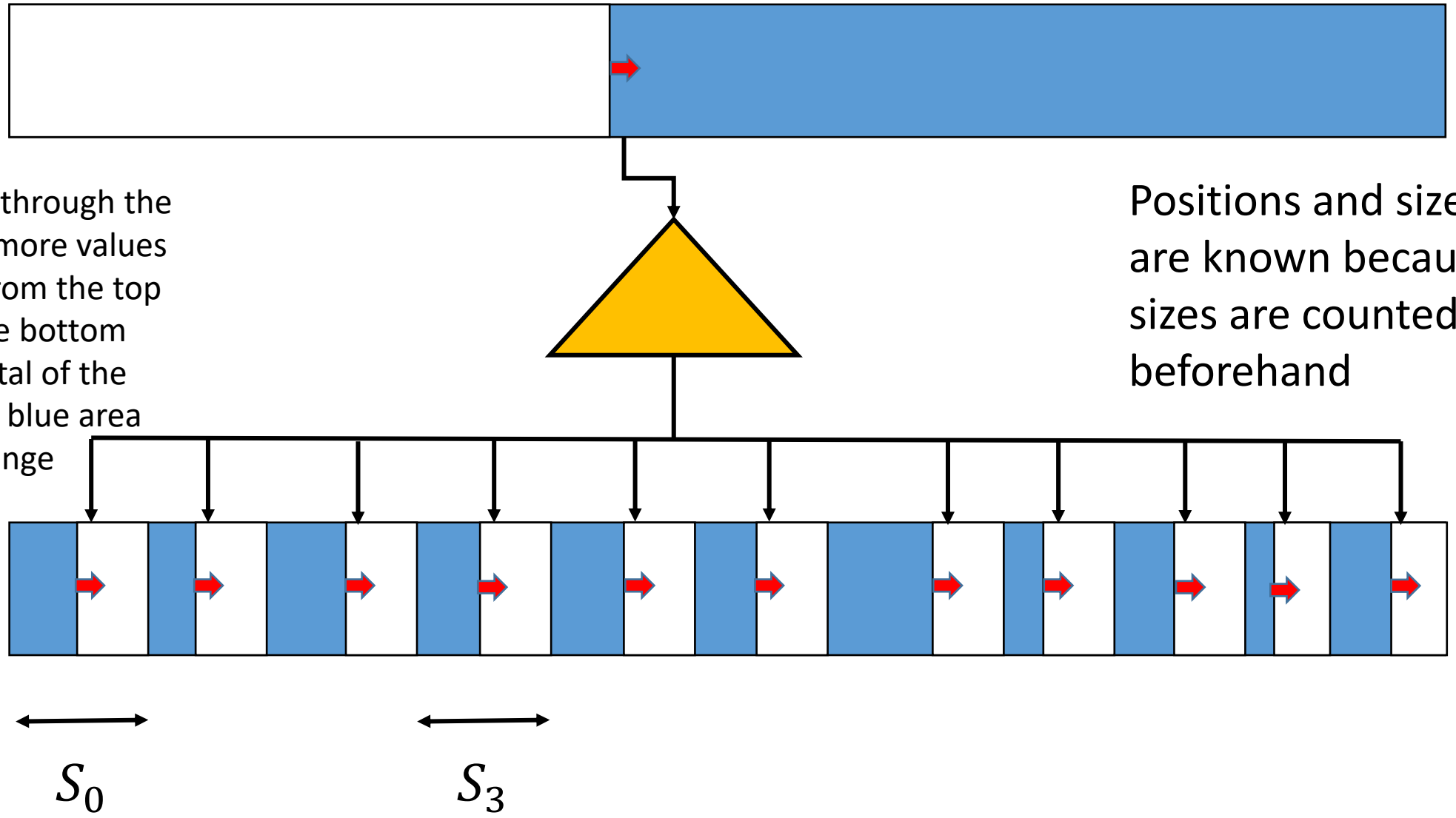

Sorting speed for String[] on Windows with Java
Entropy bits (comparisons) per second
Speed (MHz)
300
250
200
150
100
50
0
10^4
10^5
10^6
10^7
10^8
Array size
JamSort.parallelSort
Arrays.parallelSort

The basic idea of samplesort

- Take a random sample from the bag of values being sorted
- Find pivot values $p_1 \leq p_2 \leq \cdots \leq p_k$ in the sample
- Classify the values in the input bag into bags $S_0, S_1, \ldots, S_k$ such that for all values $x_i \in S_i$ we have $x_0 \leq p_1 \leq x_1 \leq p_2 \leq \cdots \leq p_k \leq x_k$
  - I.e. values in lower indexed bags are $\leq$ values in subsequent bags
- Sort each bag S_i yielding sequence R_i
- Concatenate the sequences and get a sorted total sequence

$$R_0 + R_1 + \cdots + R_k$$

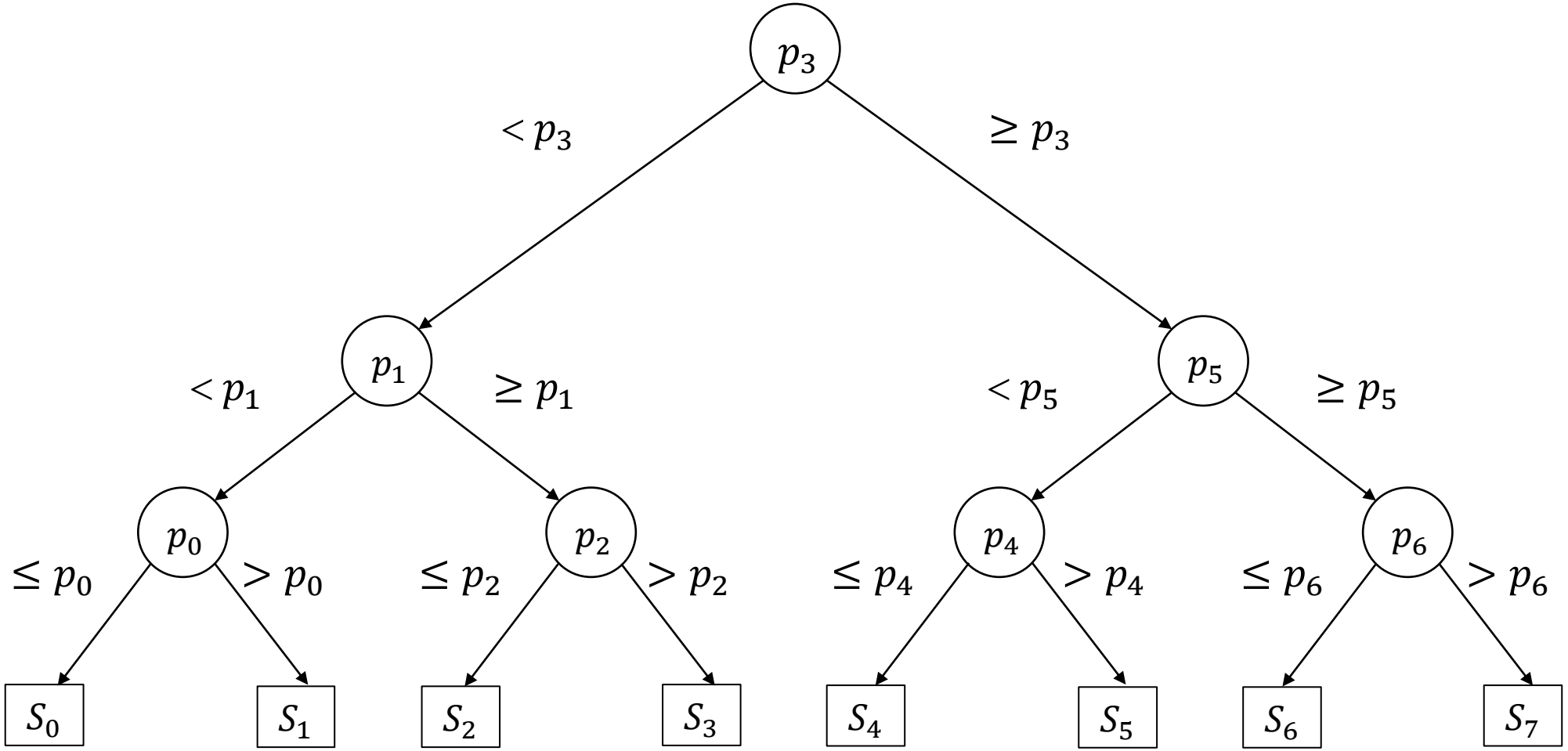
Samplesort classification loop



In each pass through the loop one or more values are moved from the top array into the bottom array. The total of the values in the blue area does not change

Positions and sizes are known because sizes are counted beforehand

Classification of values with multiple pivots



Invariant of classification loop in Dafny

```
decreases x';
invariant r'[0] + r'[1] + r'[2] + r'[3] +
          r'[4] + r'[5] + r'[6] + r'[7] +
          x' == x;
invariant forall z | z in r'[0] :: z < p[0];
invariant forall z | z in r'[1] :: p[0] <= z <= p[1];
invariant forall z | z in r'[2] :: p[1] < z < p[2];
invariant forall z | z in r'[3] :: p[2] <= z <= p[3];
invariant forall z | z in r'[4] :: p[3] < z < p[4];
invariant forall z | z in r'[5] :: p[4] <= z <= p[5];
invariant forall z | z in r'[6] :: p[5] < z < p[6];
invariant forall z | z in r'[7] :: p[6] <= z;
```

Body of classification loop in Dafny

```
var z :| z in x';
x' := x'-multiset{z};
var i := 0;
if p[3] < z    { i := i+4; }
if p[i+1] < z  { i := i+2; }
if p[i] <= z   { i := i+1; }
r'[i] := r'[i]+multiset{z};
```

Parallel samplesort: sorting a deck of cards

- Four players at the same table wish to sort the deck of cards and split the work evenly between them
- Each player gets 13 cards and classifies them into leaves, diamonds, hearts and spades
- Each player then gets a deck of cards of the same color and sorts them
- The resulting decks are then stacked together yielding a sorted deck

Advantages and disadvantages of samplesort

Advantages

- Fast for huge arrays
- Stable sorting method
- Can be parallelized
- Can be superscalar
- Cache friendly
- Fast if many values are equal
- Asymptotically optimal on average for random inputs
- For that to be true the sample must be random and a multiple of the pivot count

Disadvantages

- Slow for small arrays
- Is not faster for already sorted input
- Needs a helper array
- Conceivable bad pivot values
  - Very unlikely for large random samples

What is superscalar?

- Modern microprocessors can execute more than one machine instructions at the same time in the same thread
- For this to happen, the core must verify that none of the instructions needs an output from one of the other instructions
- In samplesort more than one value can be classified simultaneously because results from comparisons are independent
- We are then in effect running multiple independent binary searches simultaneously
- This speeds up samplesort considerably when working with primitive values such as int and double

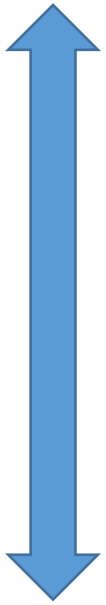
Example of a superscalar classification loop

```
while( n!=1 )
{
    // | <xK | ?? | >=xK |
    //          ^    ^
    //          iK   iK+n
    // n>0 is of form 2^k-1
    n >>= 1;
    np = n+1;
```

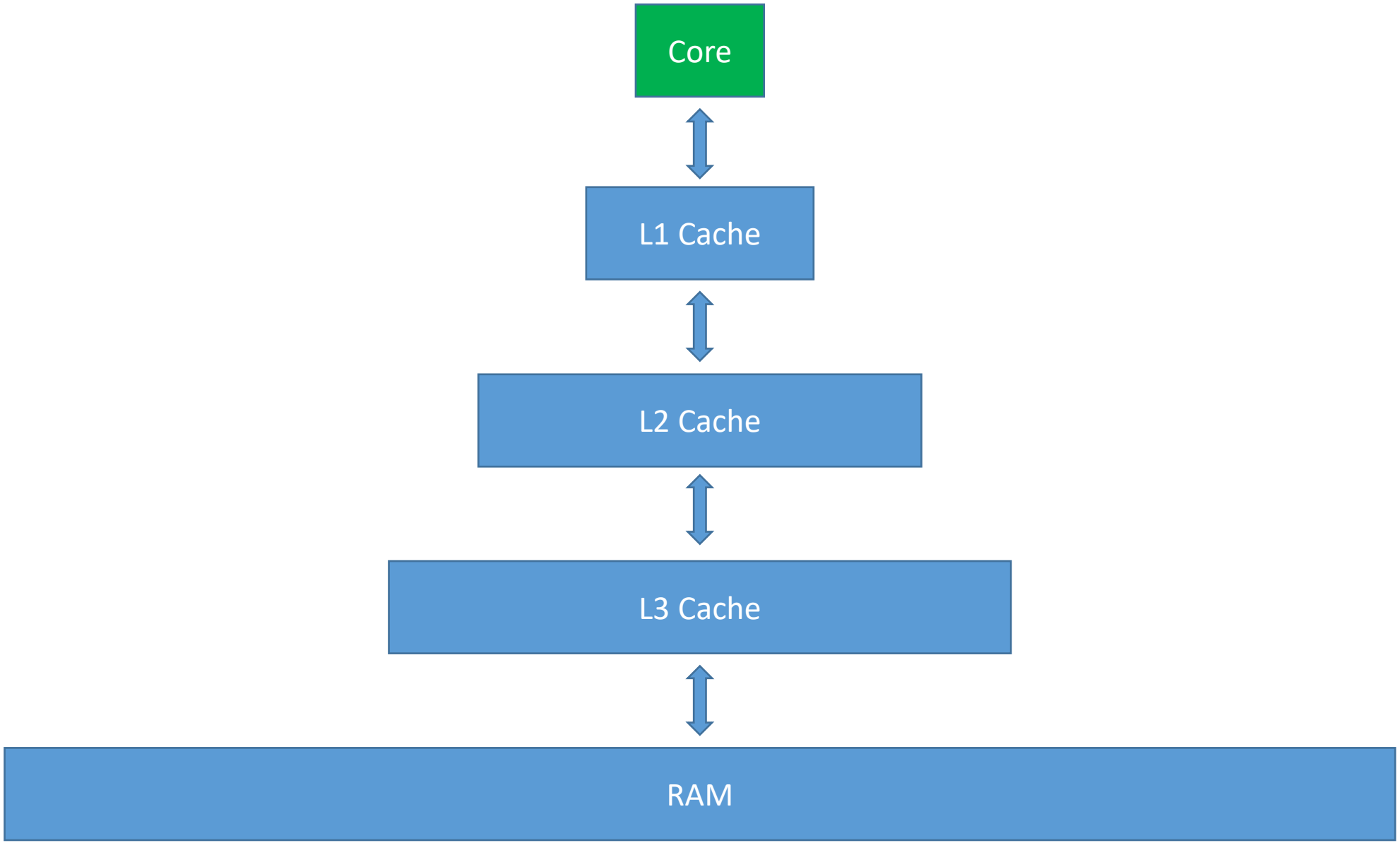
```
    i0 += p[i0+n]<x0?np:0;
    i1 += p[i1+n]<x1?np:0;
    i2 += p[i2+n]<x2?np:0;
    i3 += p[i3+n]<x3?np:0;
    i4 += p[i4+n]<x4?np:0;
    i5 += p[i5+n]<x5?np:0;
    i6 += p[i6+n]<x6?np:0;
    i7 += p[i7+n]<x7?np:0;
}
```

Cache

More bandwidth
Shorter latency
Less space



Less bandwidth
Higher latency
More space



Cache friendly programs

- The cache footprint should be small
  - The cache footprint is that part of the RAM that the core needs to access at each time in order to continue with the computations
- How the cache footprint changes should be predictable
- Accessing the same area in RAM often with a long period between accesses should be avoided

Samplesort Cache Footprint