SIGMOD DaMoN 23.06.2014

Main Memory Adaptive Indexing for Multi-core Systems

Felix Martin Schuhknecht
Victor Alvarez
Jens Dittrich
Stefan Richter

Information Systems Group
Saarland University
https://infosys.uni-saarland.de/

Problem: Answer Range Queries

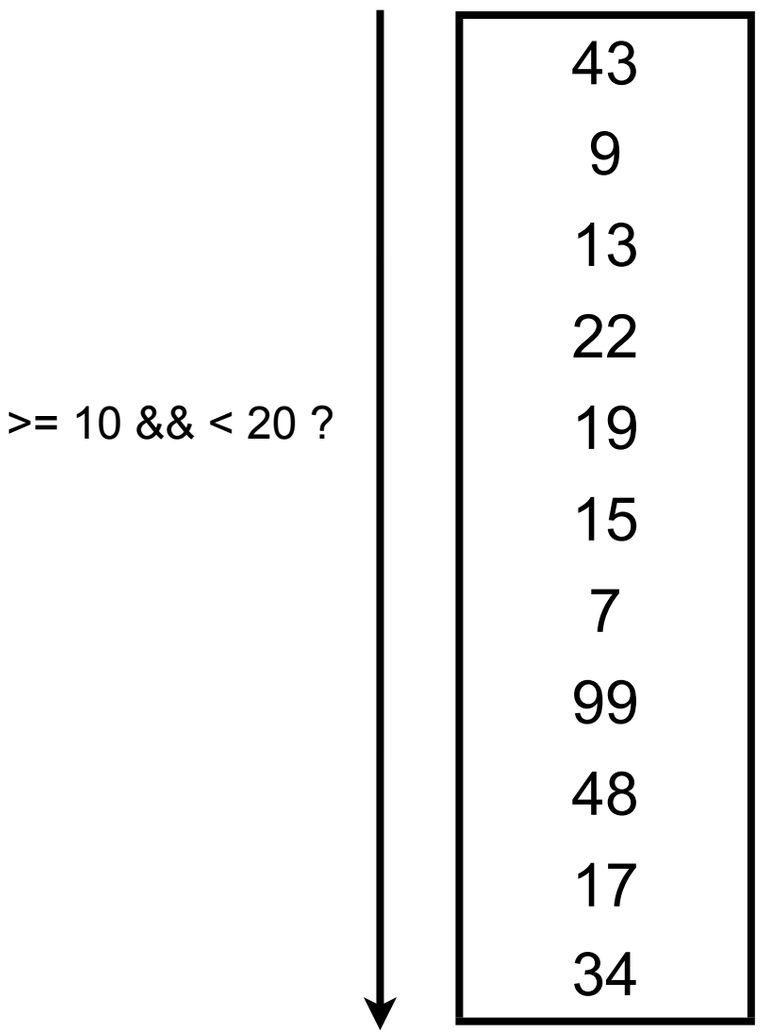
Problem: Answer Range Queries

```
select A
from   R
where  R.A >= 10 and R.A < 20
```

Problem: Answer Range Queries

```
select A
from   R
where  R.A >= 10 and R.A < 20
```

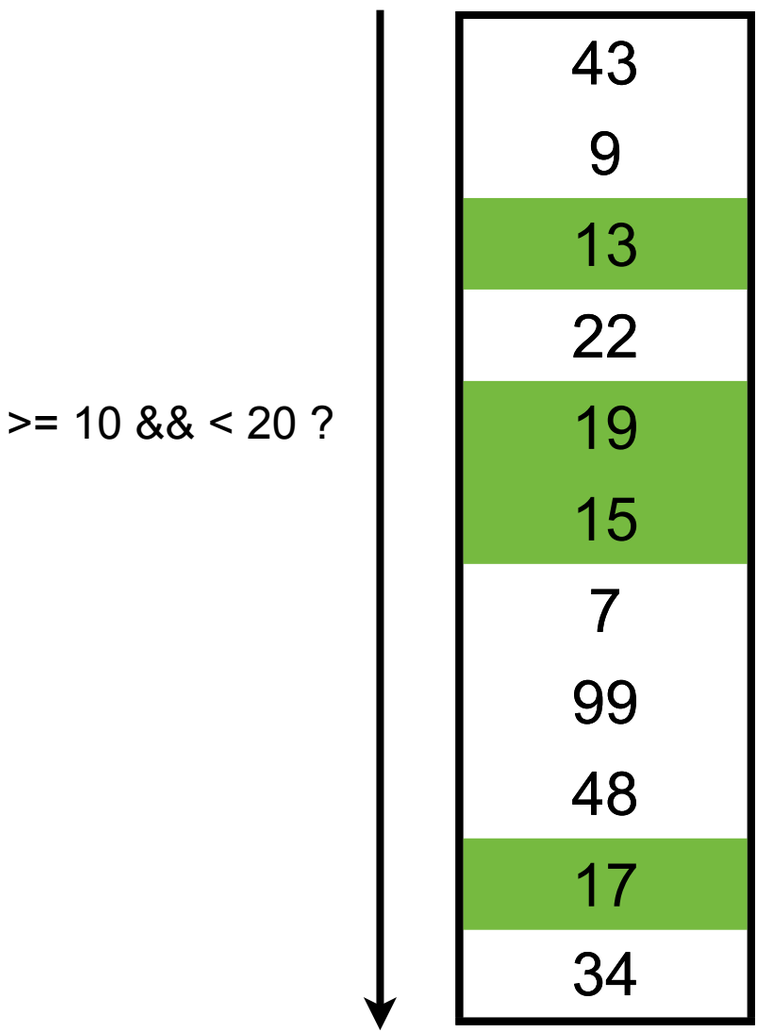
One extreme: Scan + Filter



Problem: Answer Range Queries

```
select A
from   R
where  R.A >= 10 and R.A < 20
```

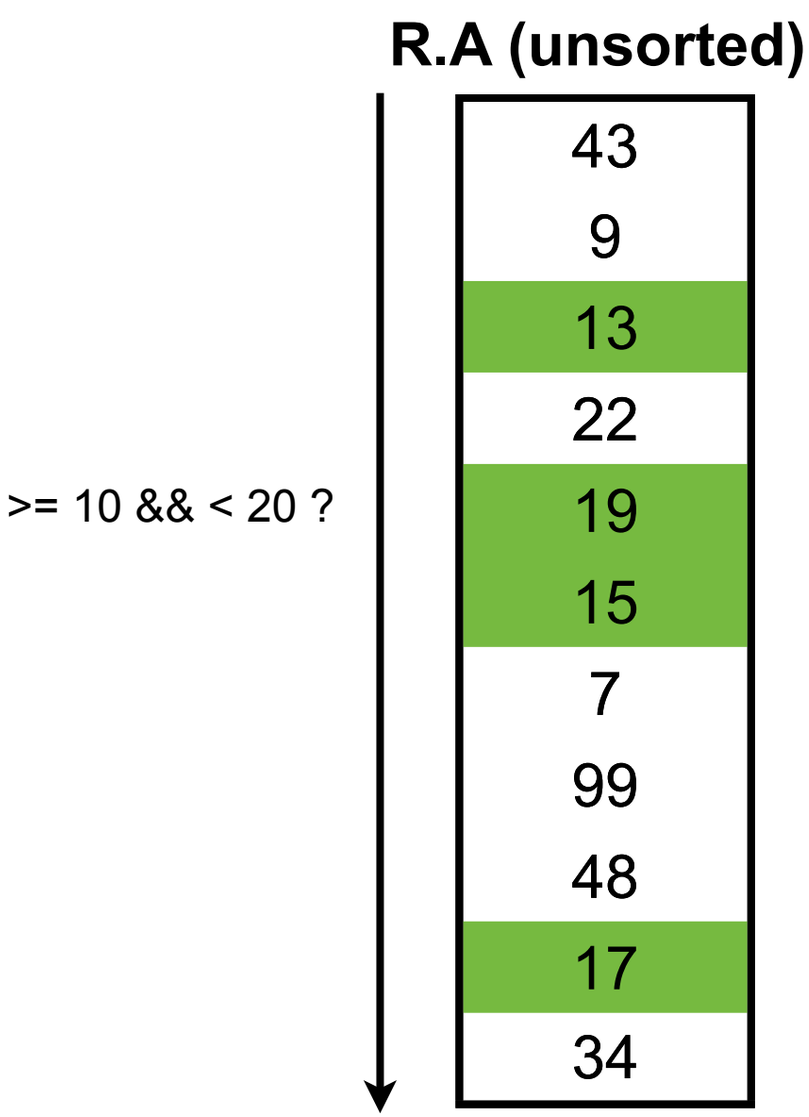
One extreme: Scan + Filter



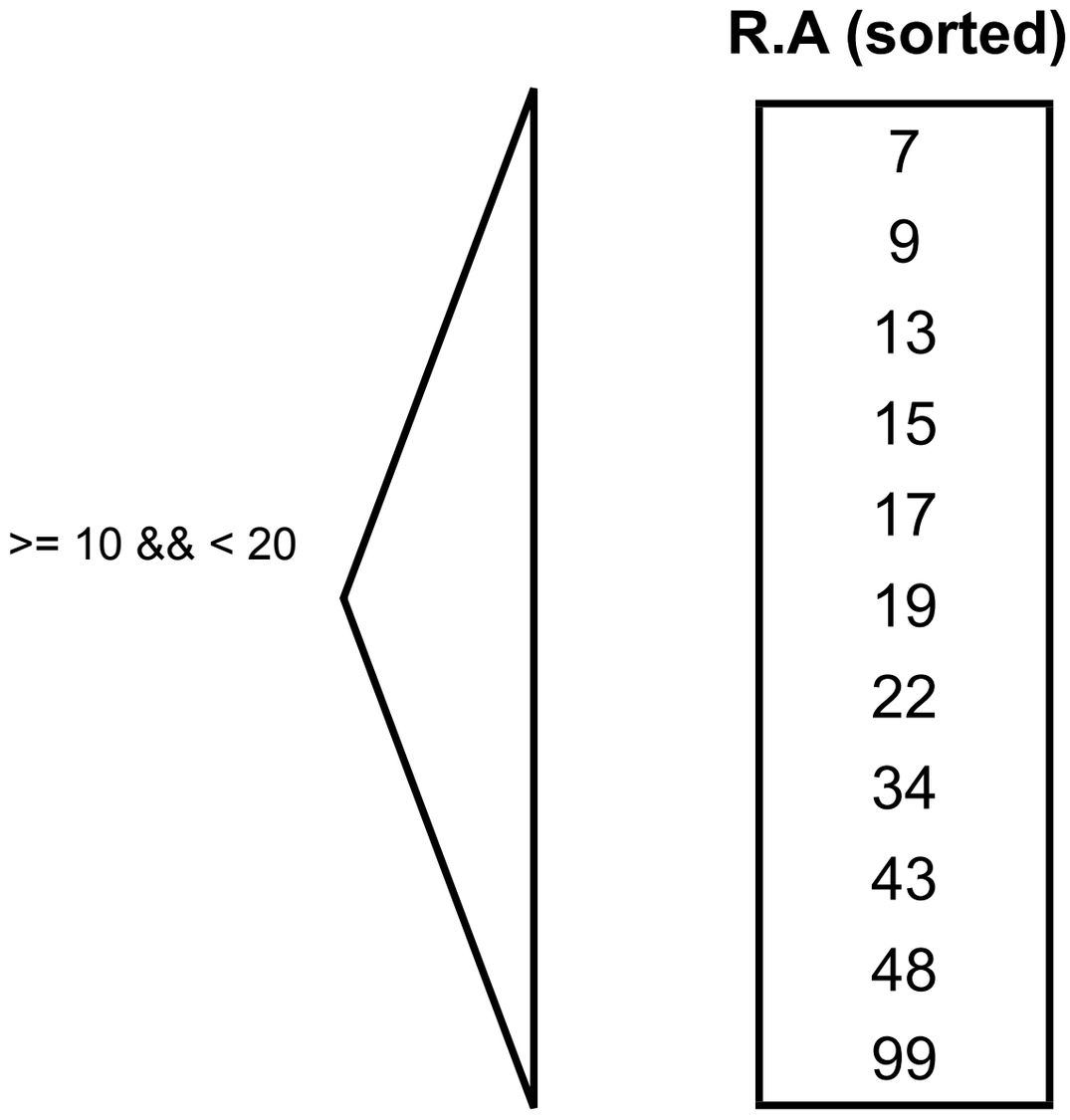
Problem: Answer Range Queries

```
select A
from   R
where  R.A >= 10 and R.A < 20
```

One extreme: Scan + Filter



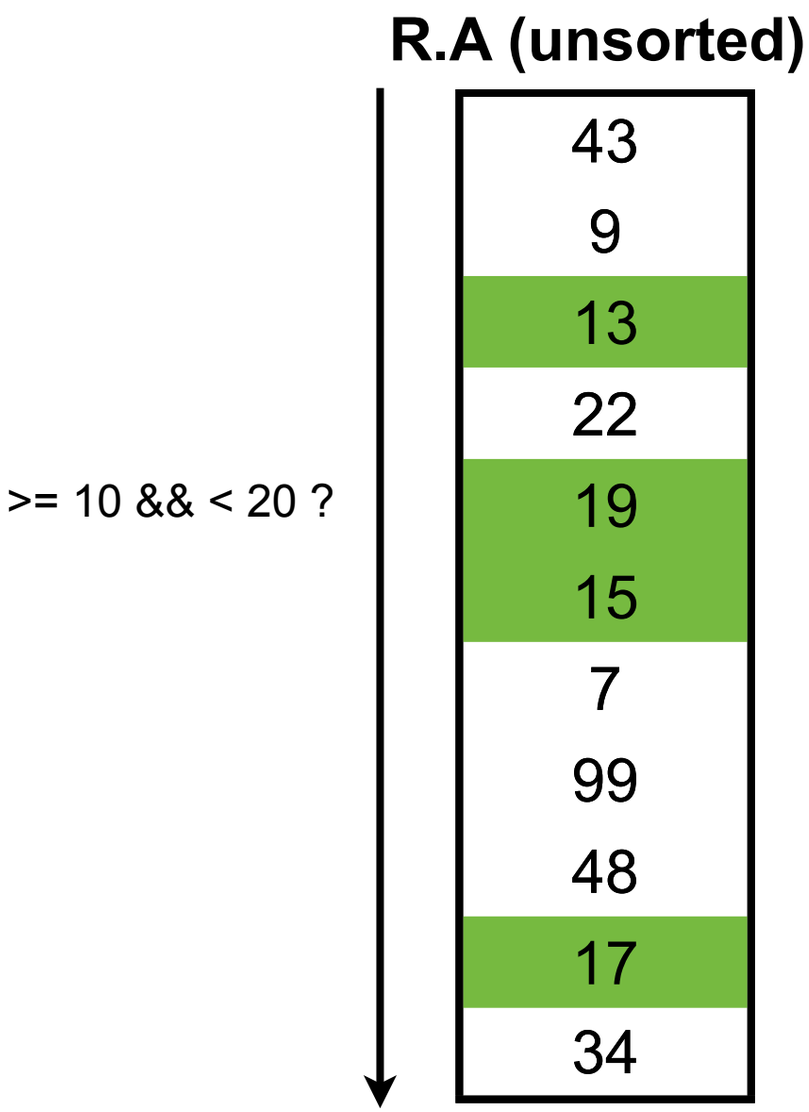
Other extreme: Index



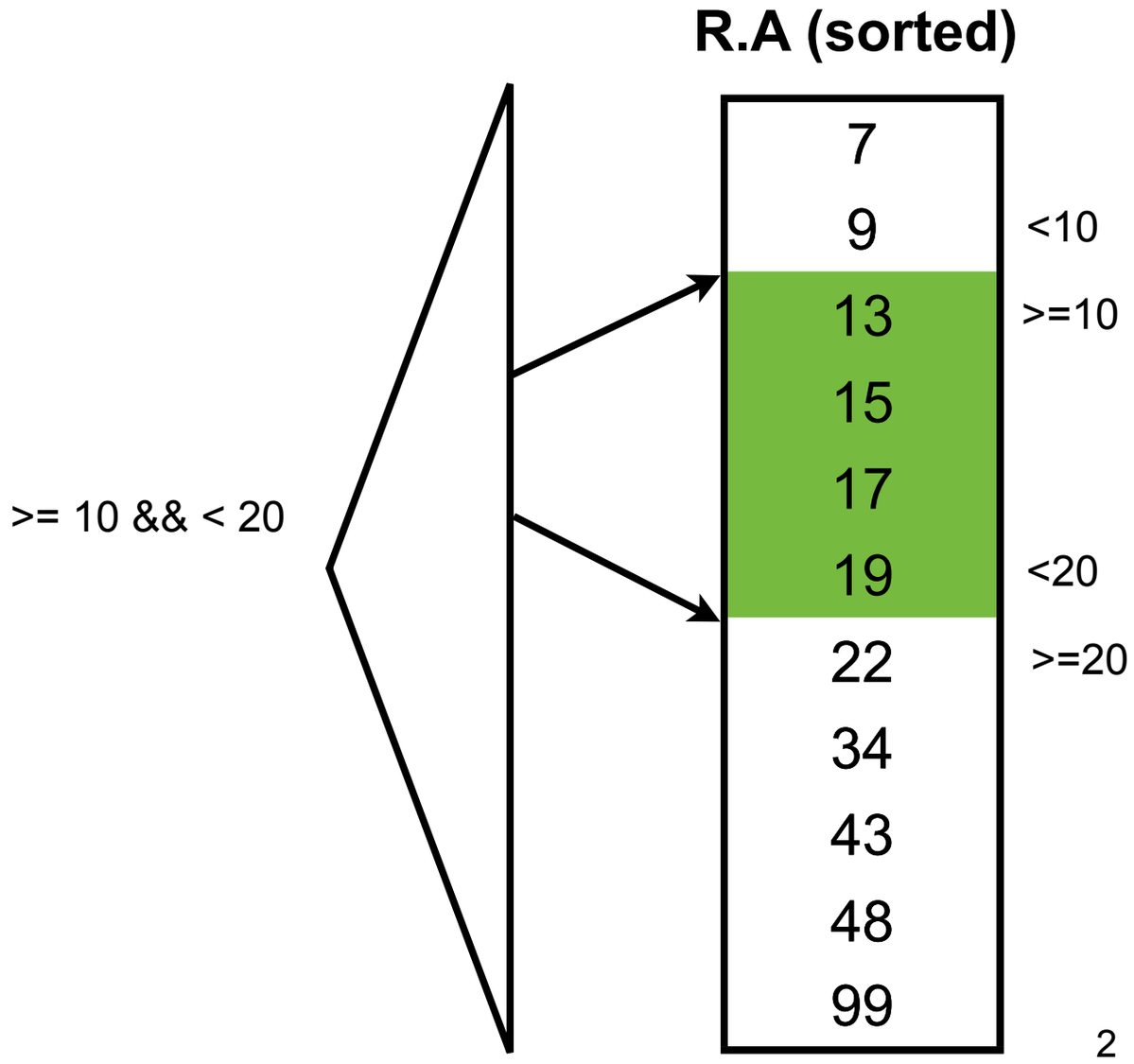
Problem: Answer Range Queries

```
select A
from   R
where  R.A >= 10 and R.A < 20
```

One extreme: Scan + Filter



Other extreme: Index



Index: When to build?

One extreme: At once (Traditional Indexing)



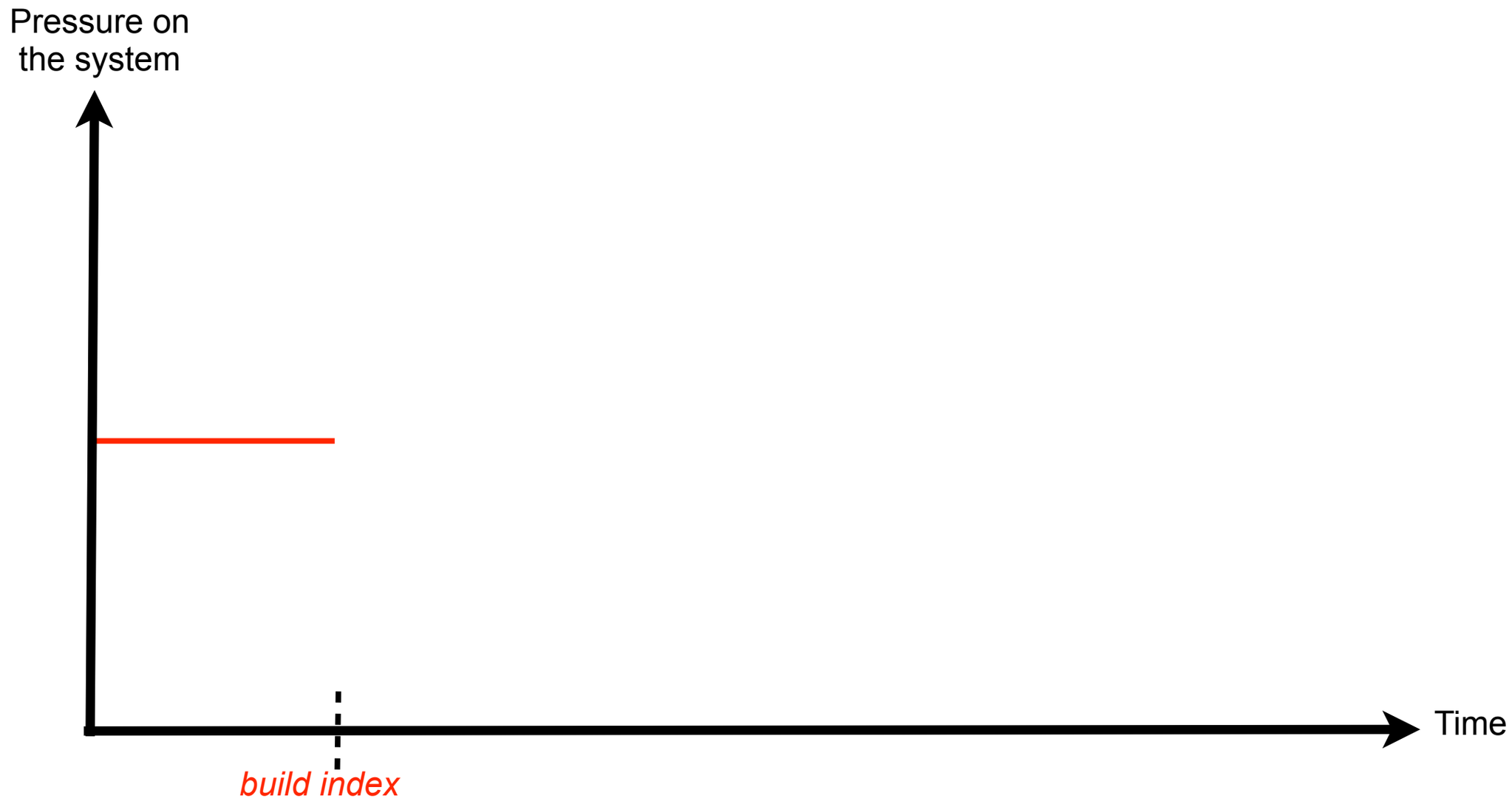
Index: When to build?

One extreme: At once (Traditional Indexing)



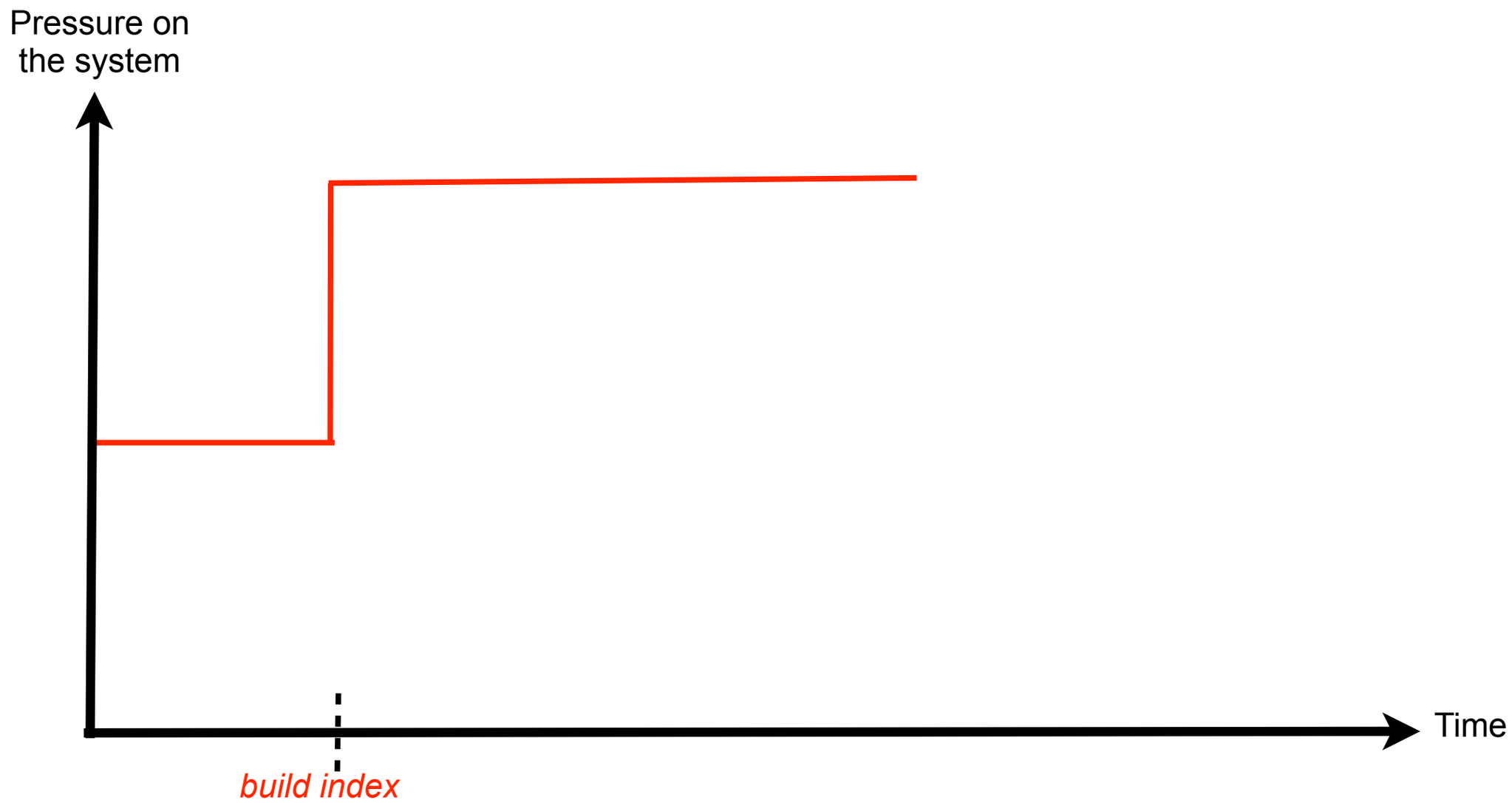
Index: When to build?

One extreme: At once (Traditional Indexing)



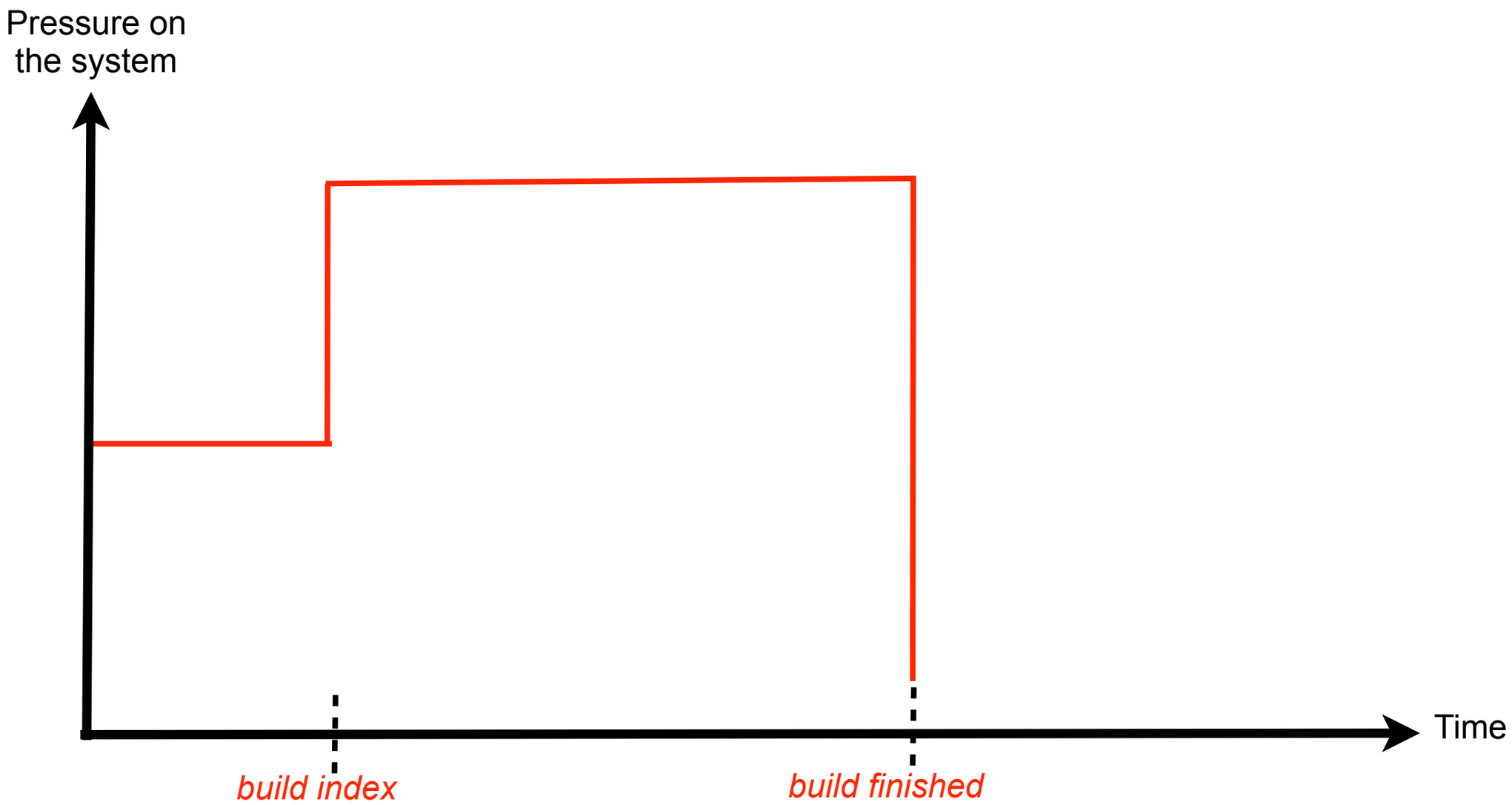
Index: When to build?

One extreme: At once (Traditional Indexing)



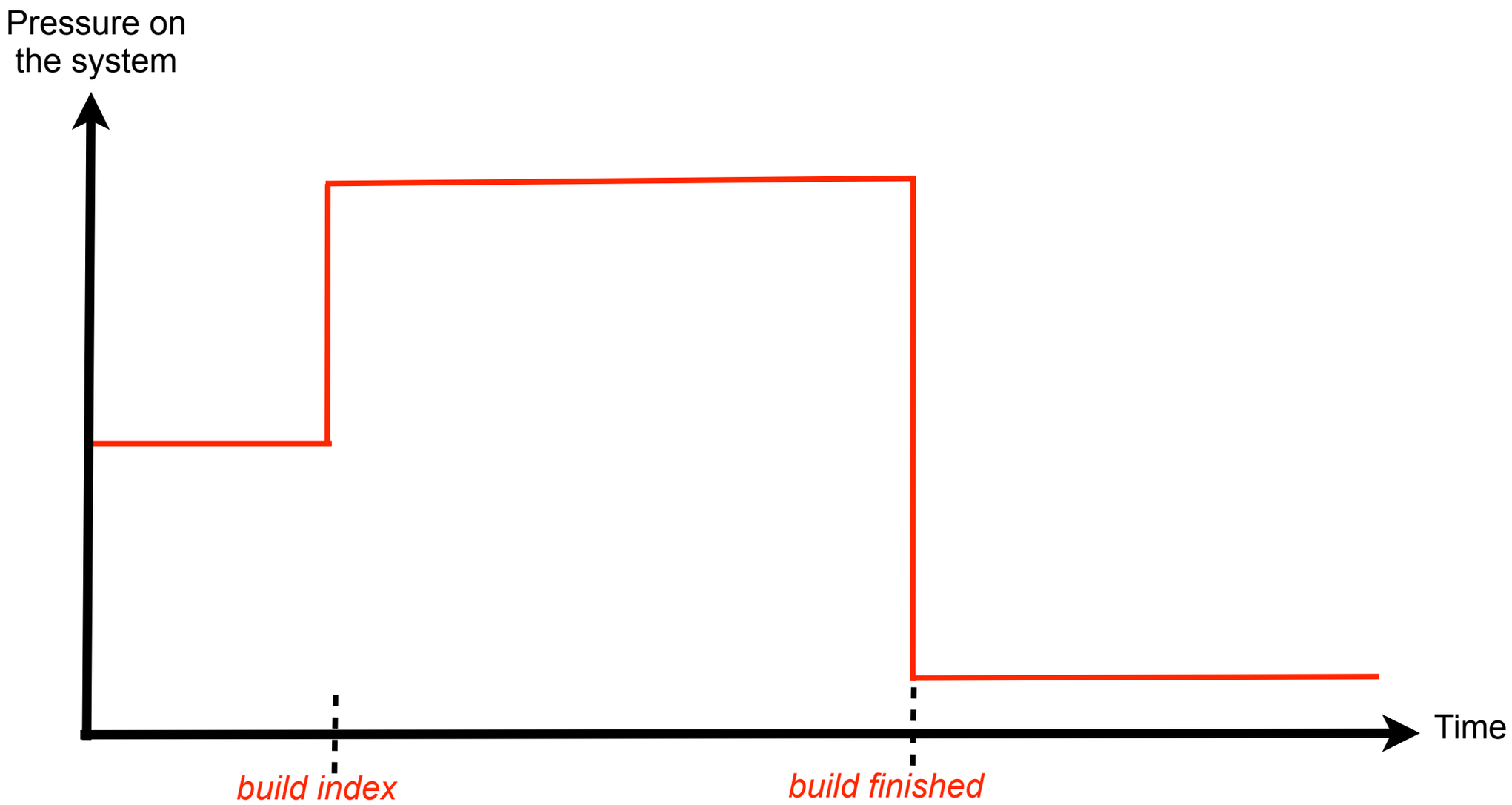
Index: When to build?

One extreme: At once (Traditional Indexing)



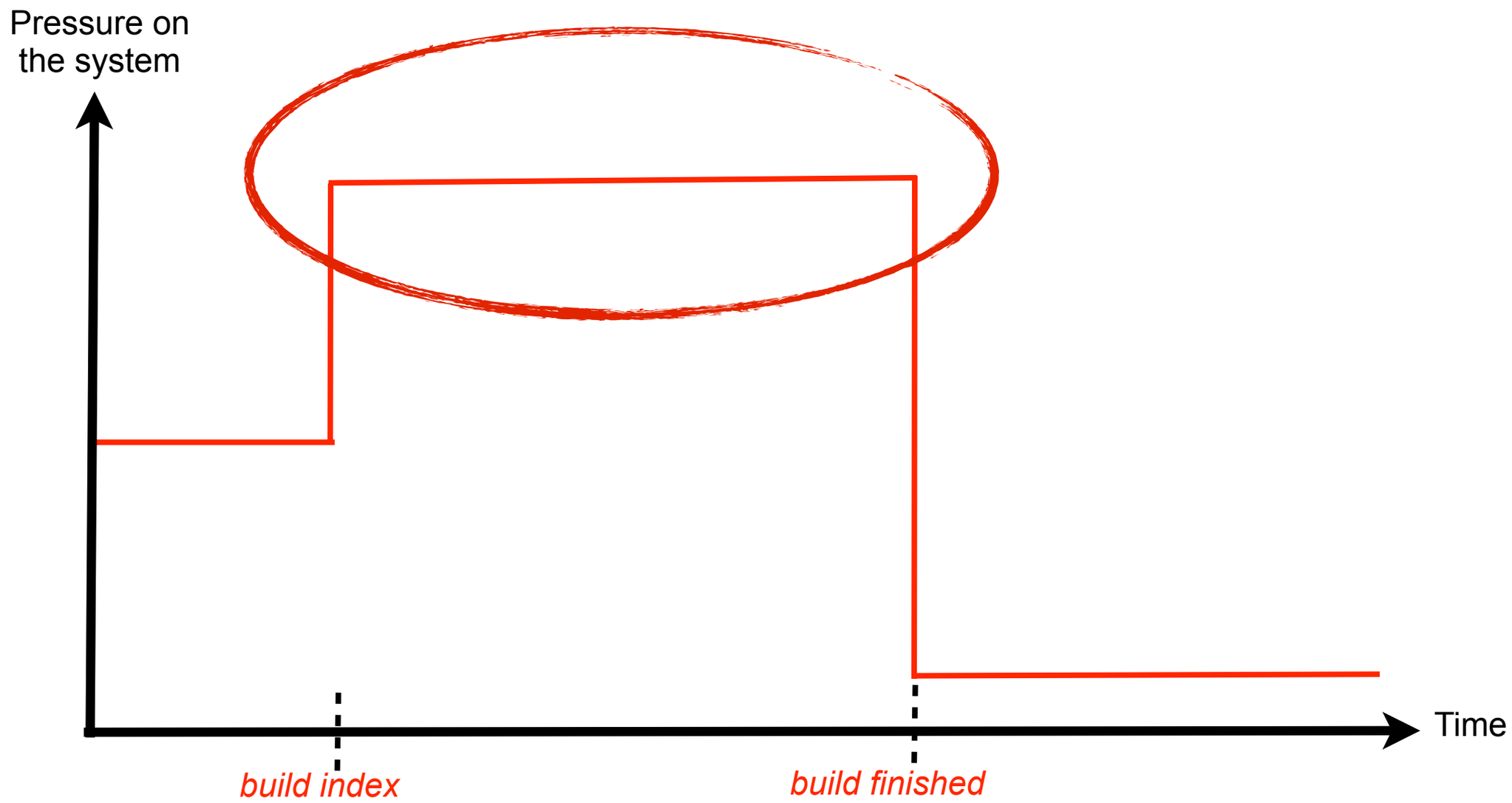
Index: When to build?

One extreme: At once (Traditional Indexing)



Index: When to build?

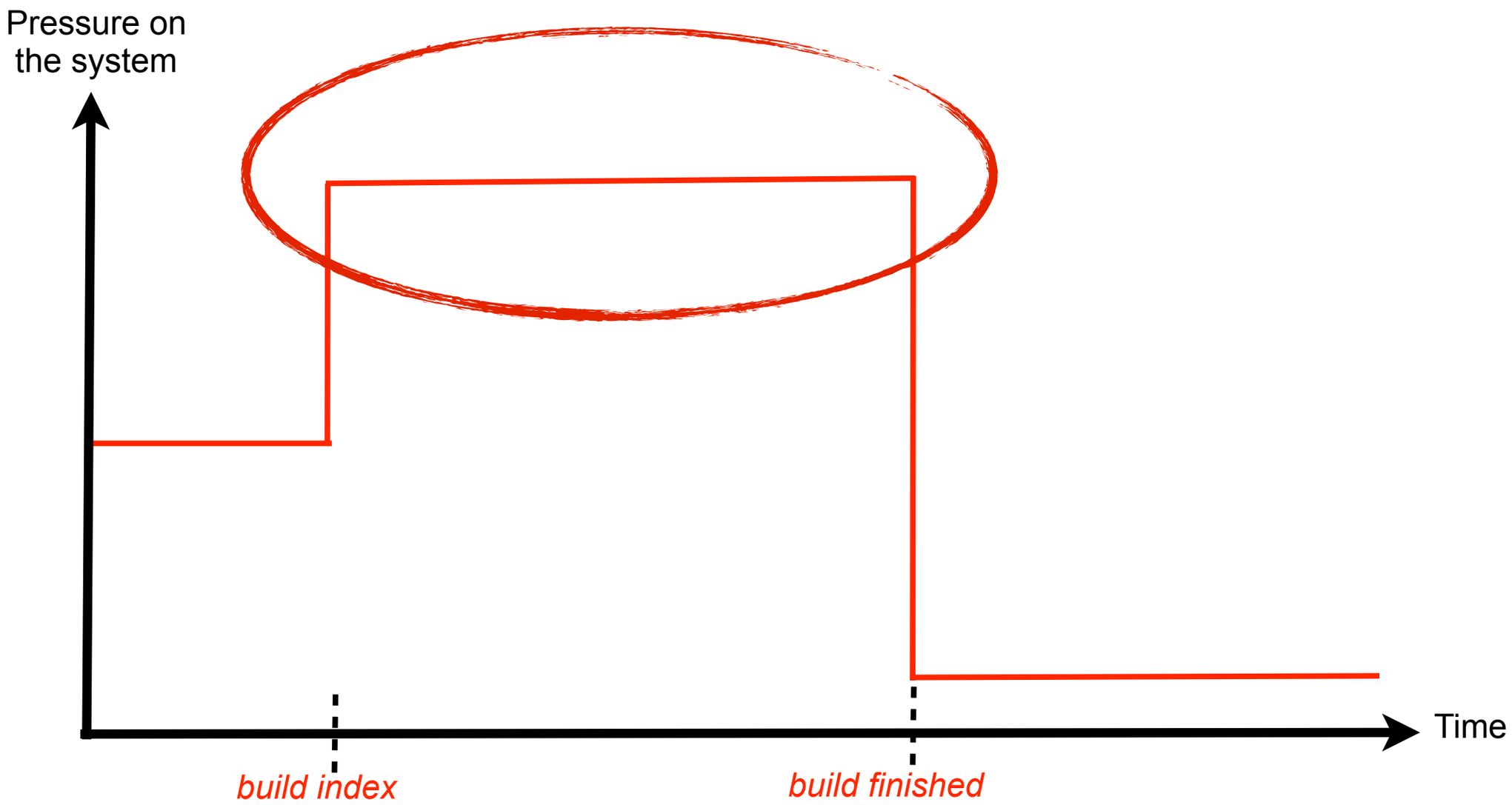
One extreme: At once (Traditional Indexing)



Index: When to build?

One extreme: At once (Traditional Indexing)

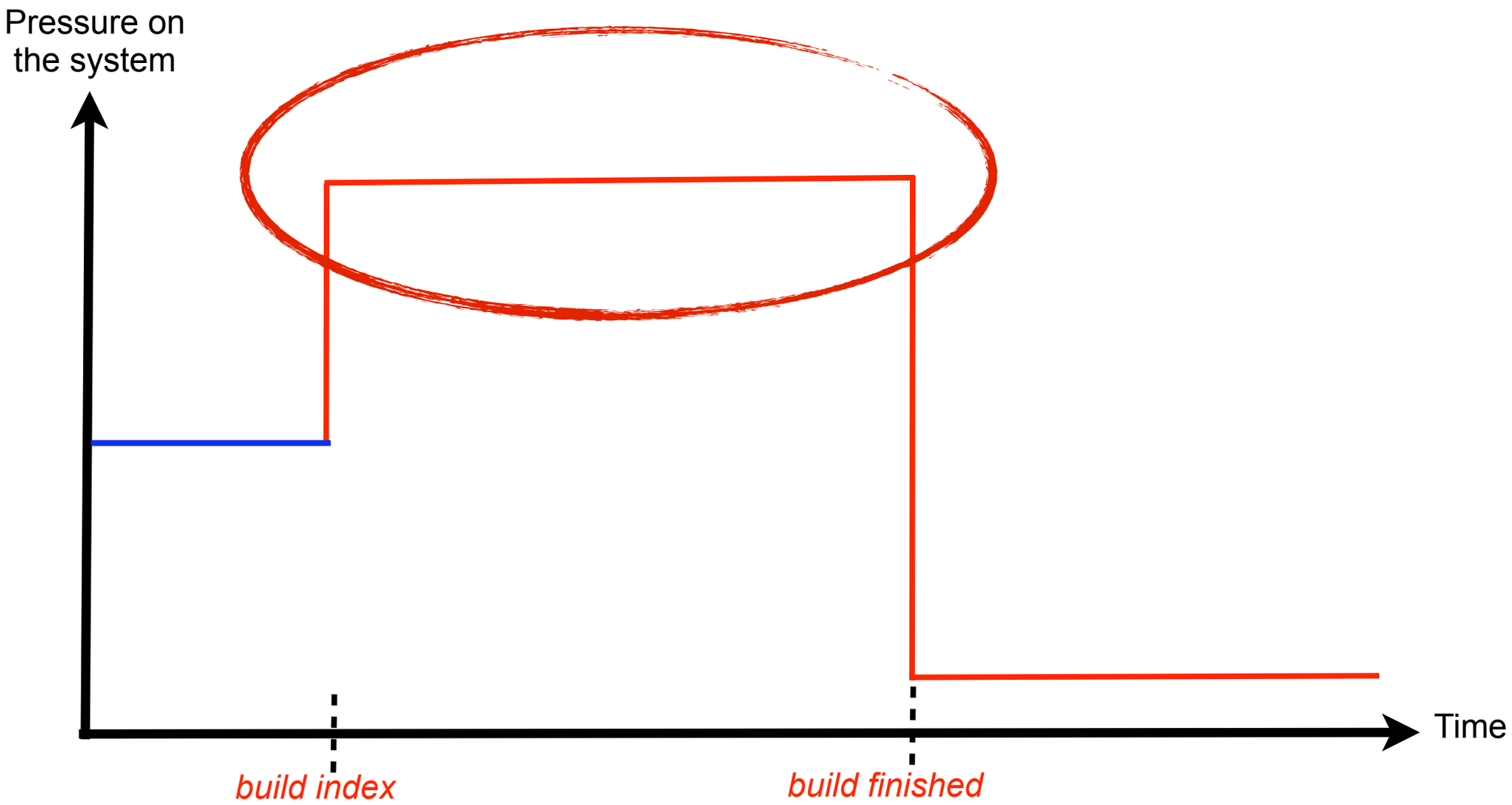
Other extreme: Incrementally at query time (Adaptive Indexing)



Index: When to build?

One extreme: At once (Traditional Indexing)

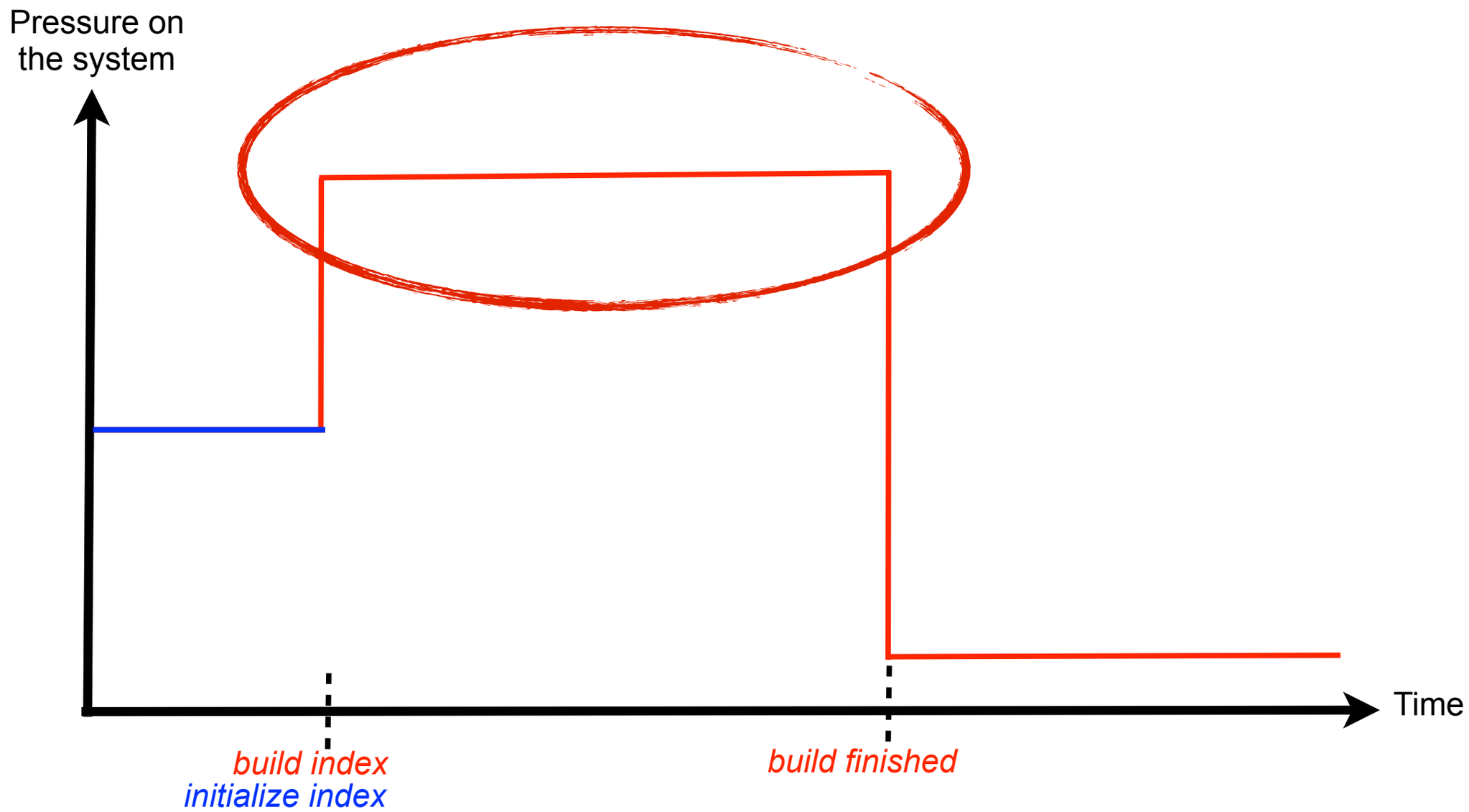
Other extreme: Incrementally at query time (Adaptive Indexing)



Index: When to build?

One extreme: At once (Traditional Indexing)

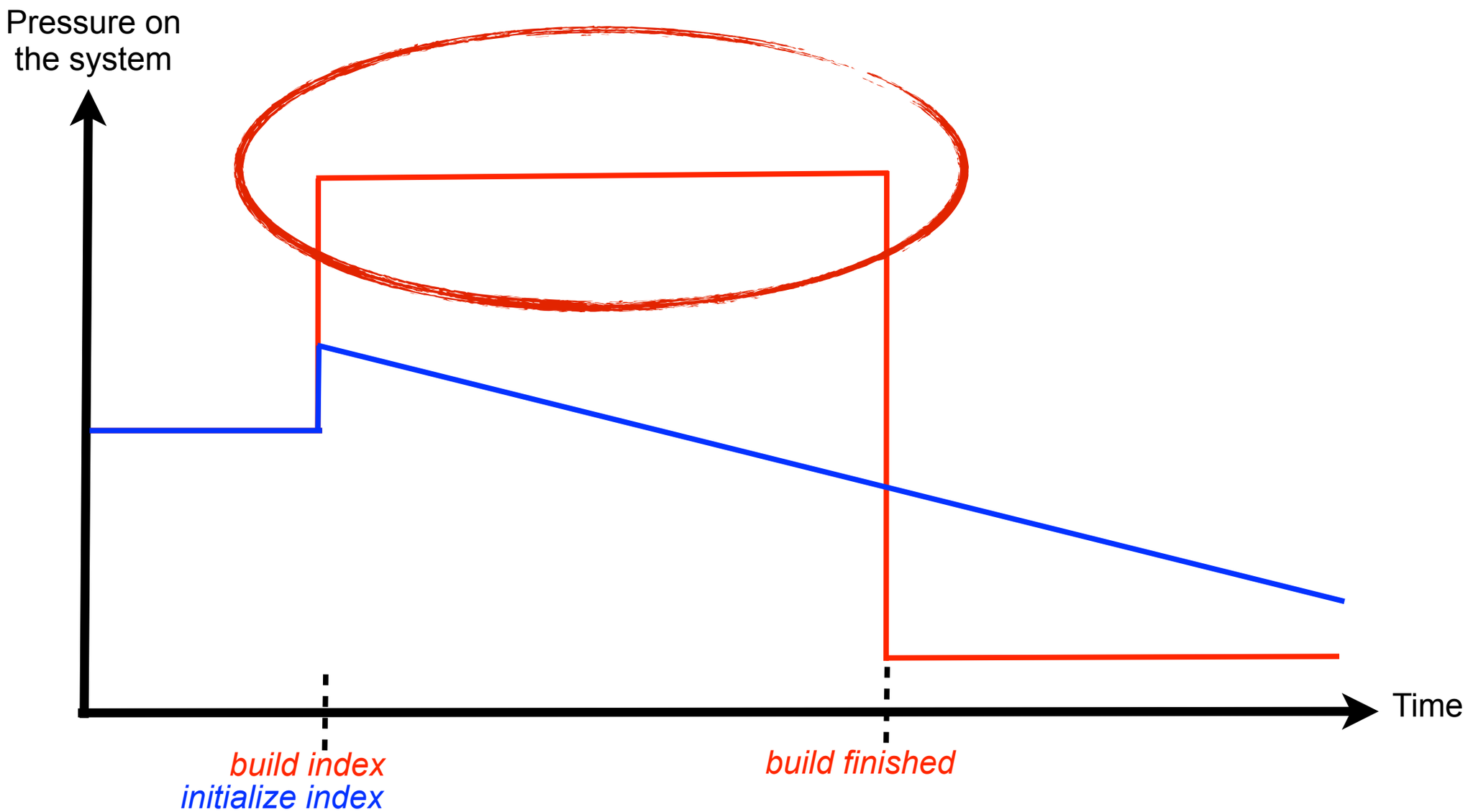
Other extreme: Incrementally at query time (Adaptive Indexing)



Index: When to build?

One extreme: At once (Traditional Indexing)

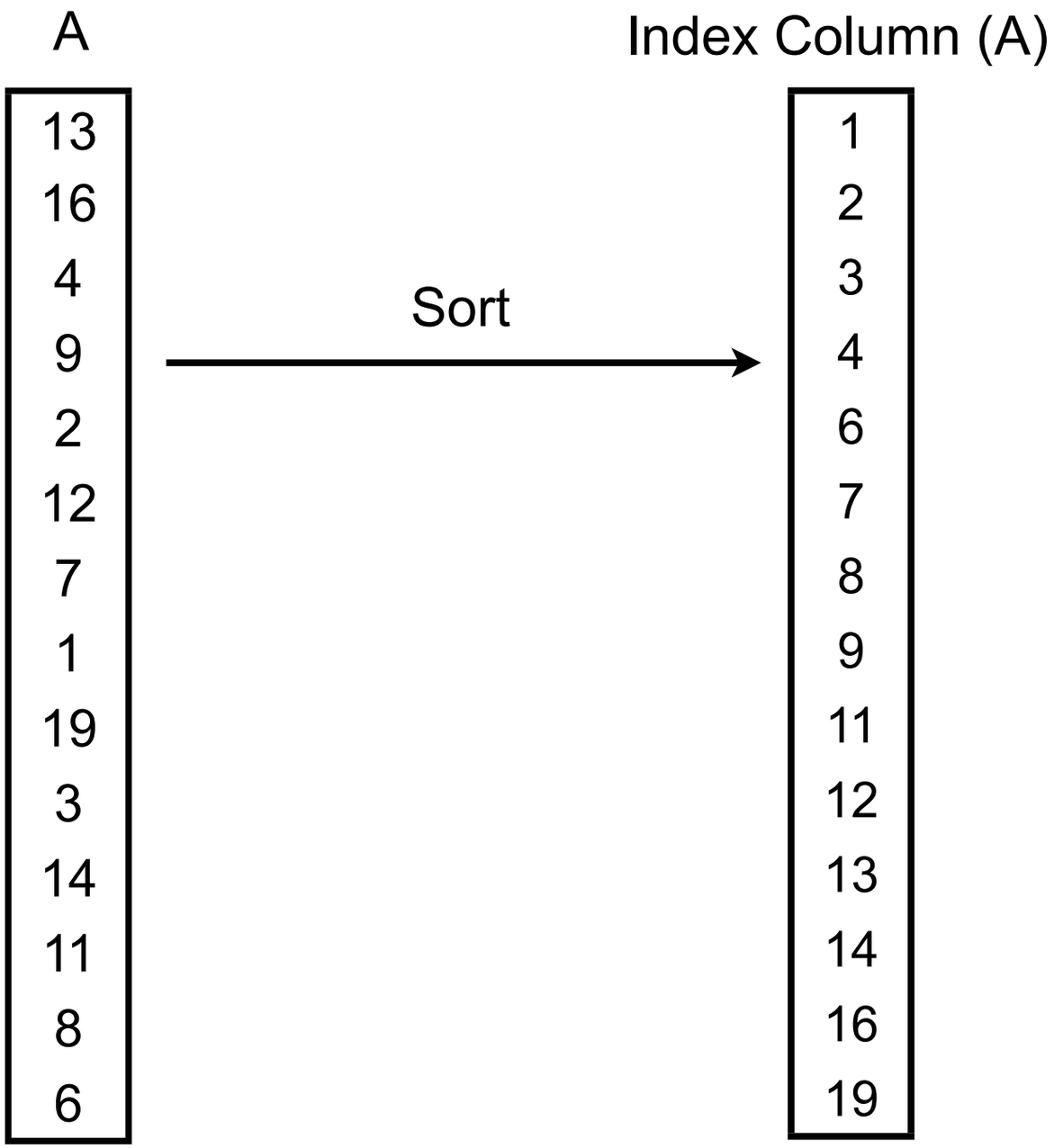
Other extreme: Incrementally at query time (Adaptive Indexing)



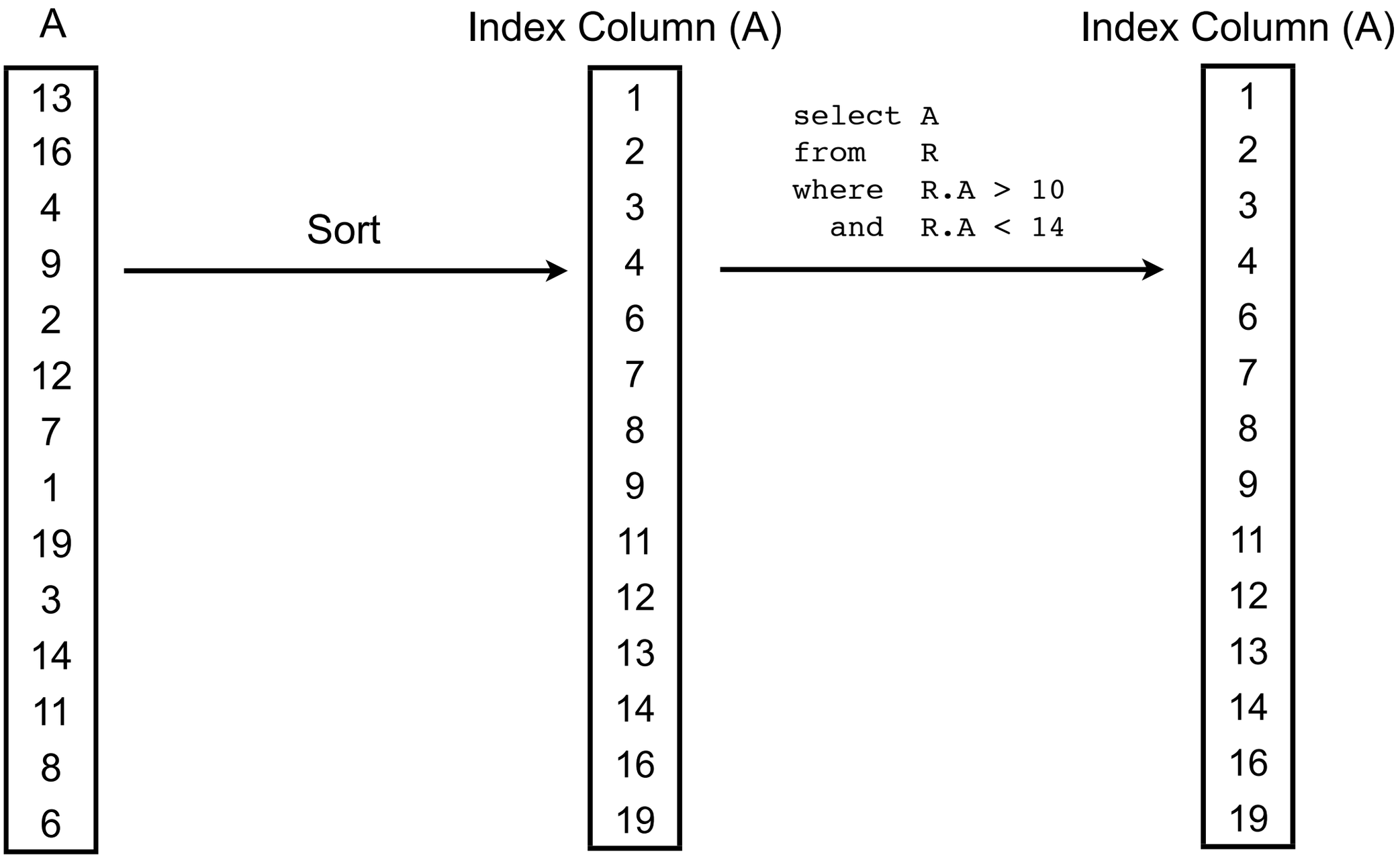
Traditional Indexing: Sort + Binary Search

A
13
16
4
9
2
12
7
1
19
3
14
11
8
6

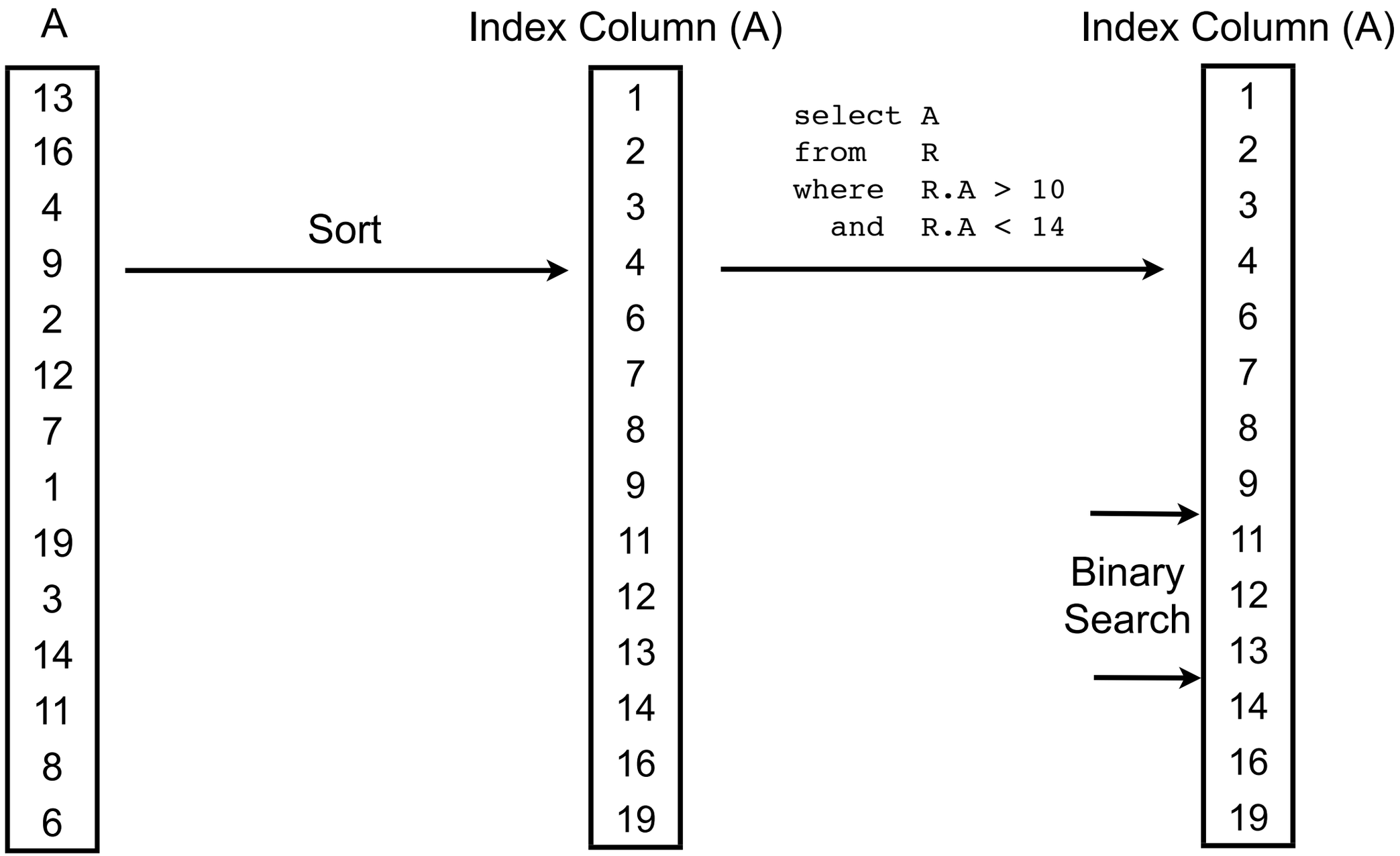
Traditional Indexing: Sort + Binary Search



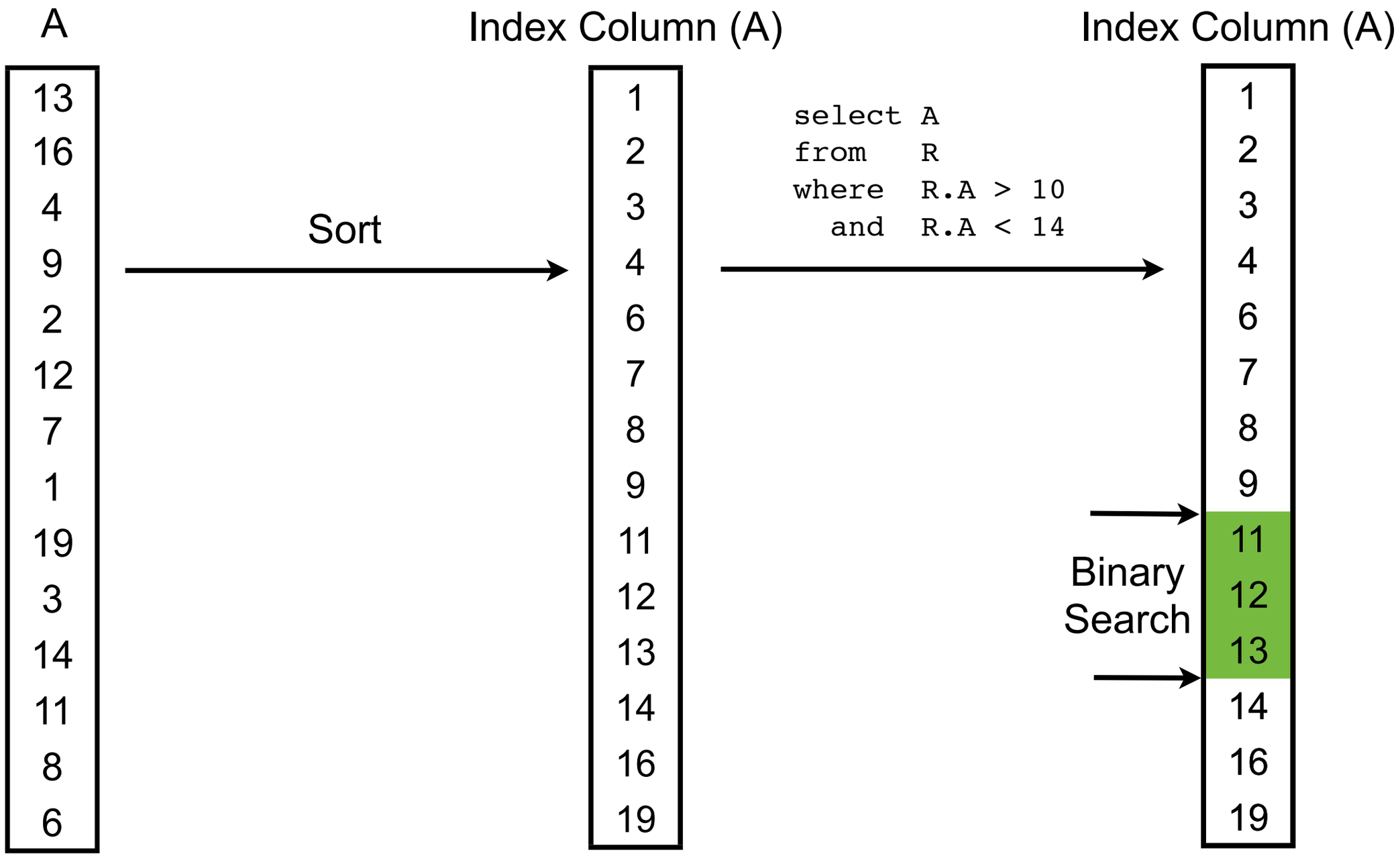
Traditional Indexing: Sort + Binary Search



Traditional Indexing: Sort + Binary Search



Traditional Indexing: Sort + Binary Search



Adaptive Indexing: Standard Cracking

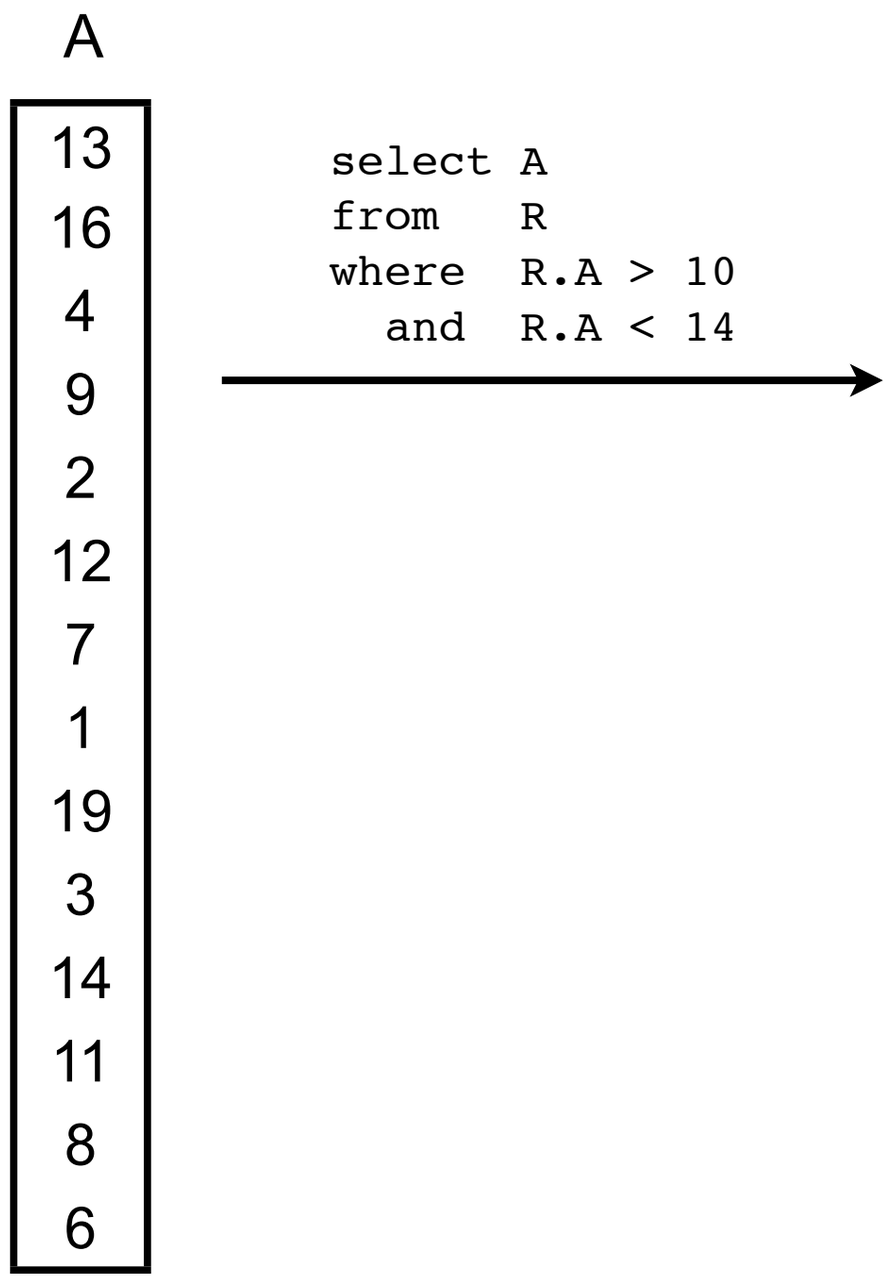
[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking

A
13
16
4
9
2
12
7
1
19
3
14
11
8
6

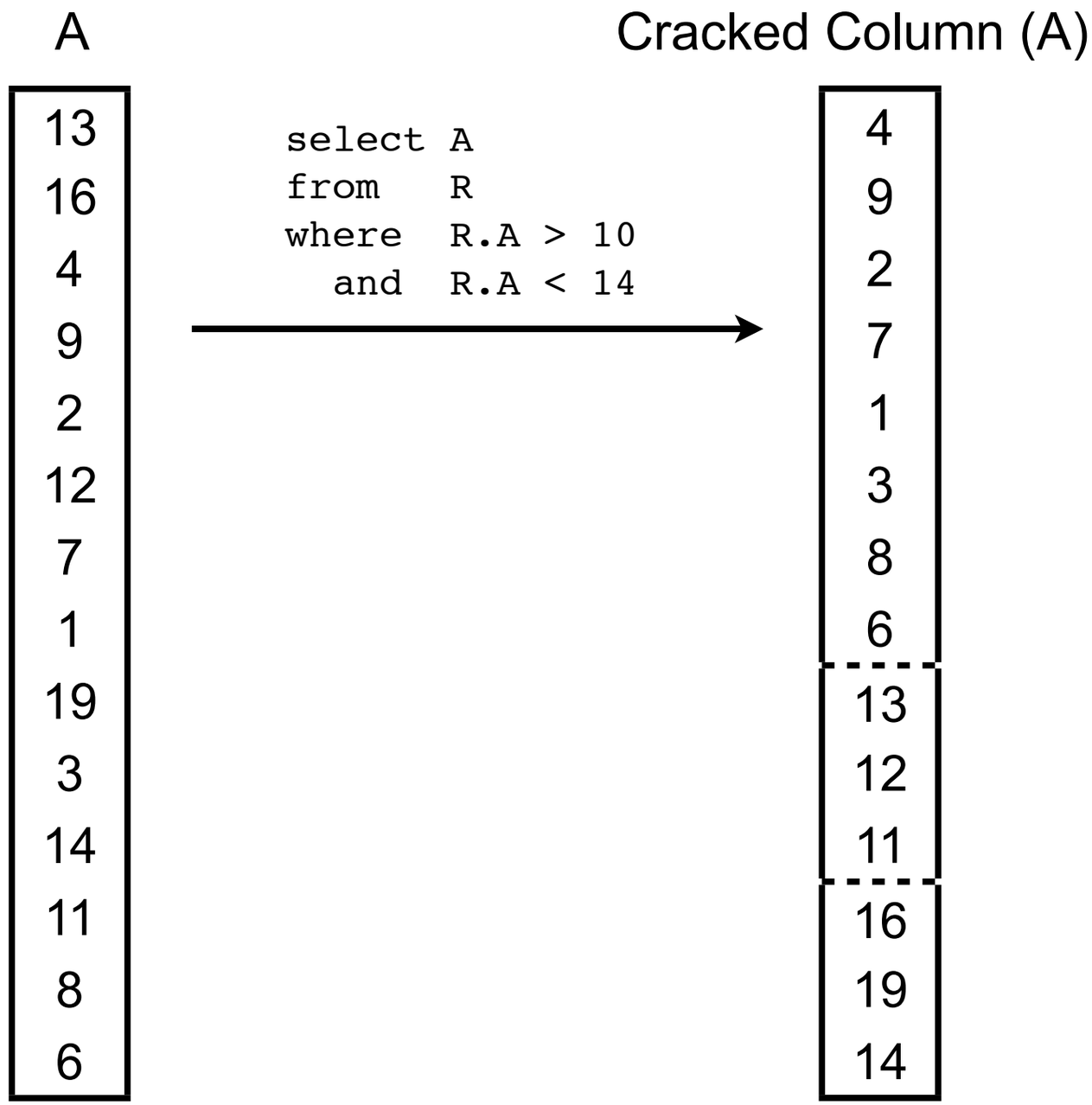
[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking



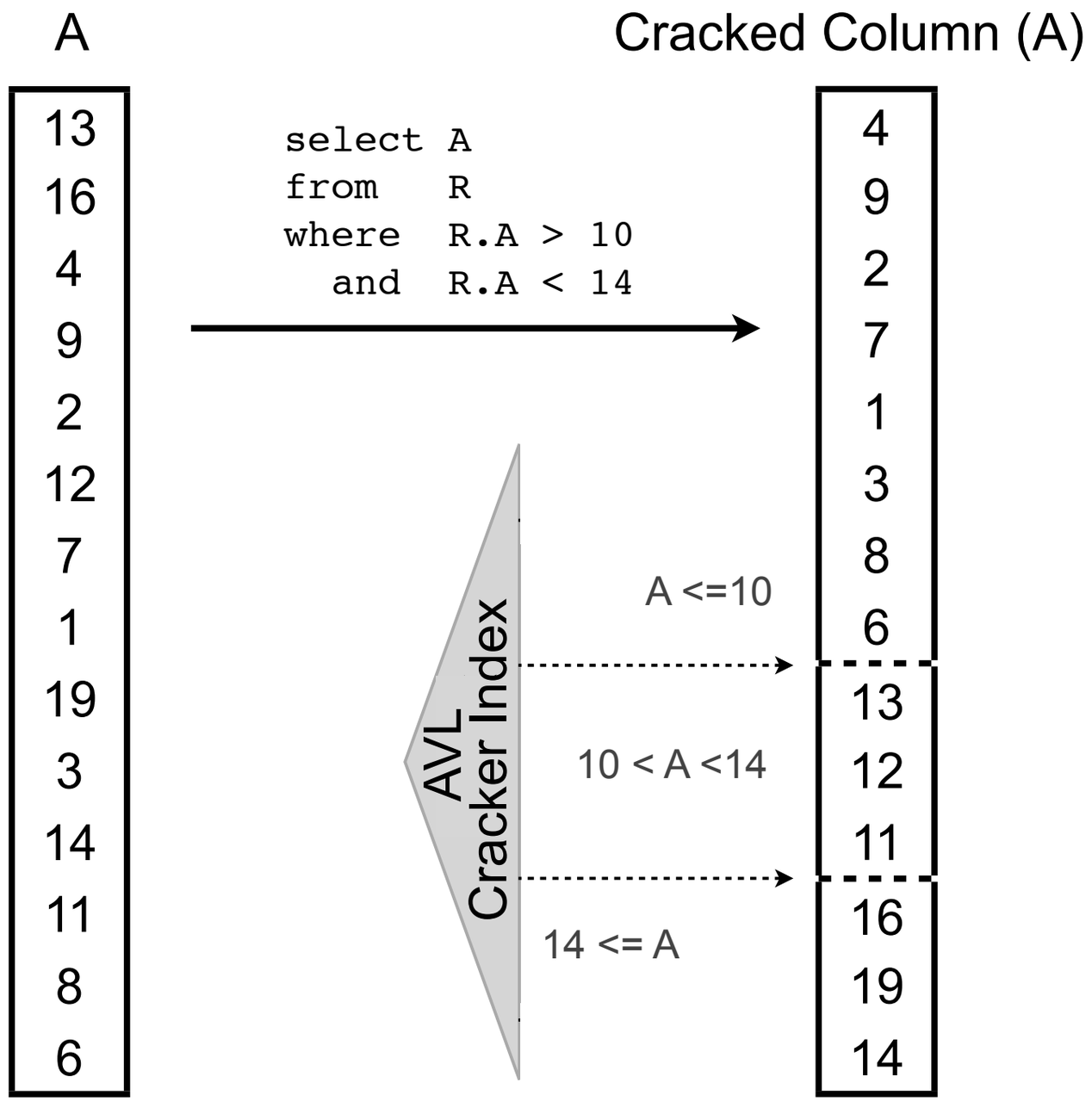
[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking



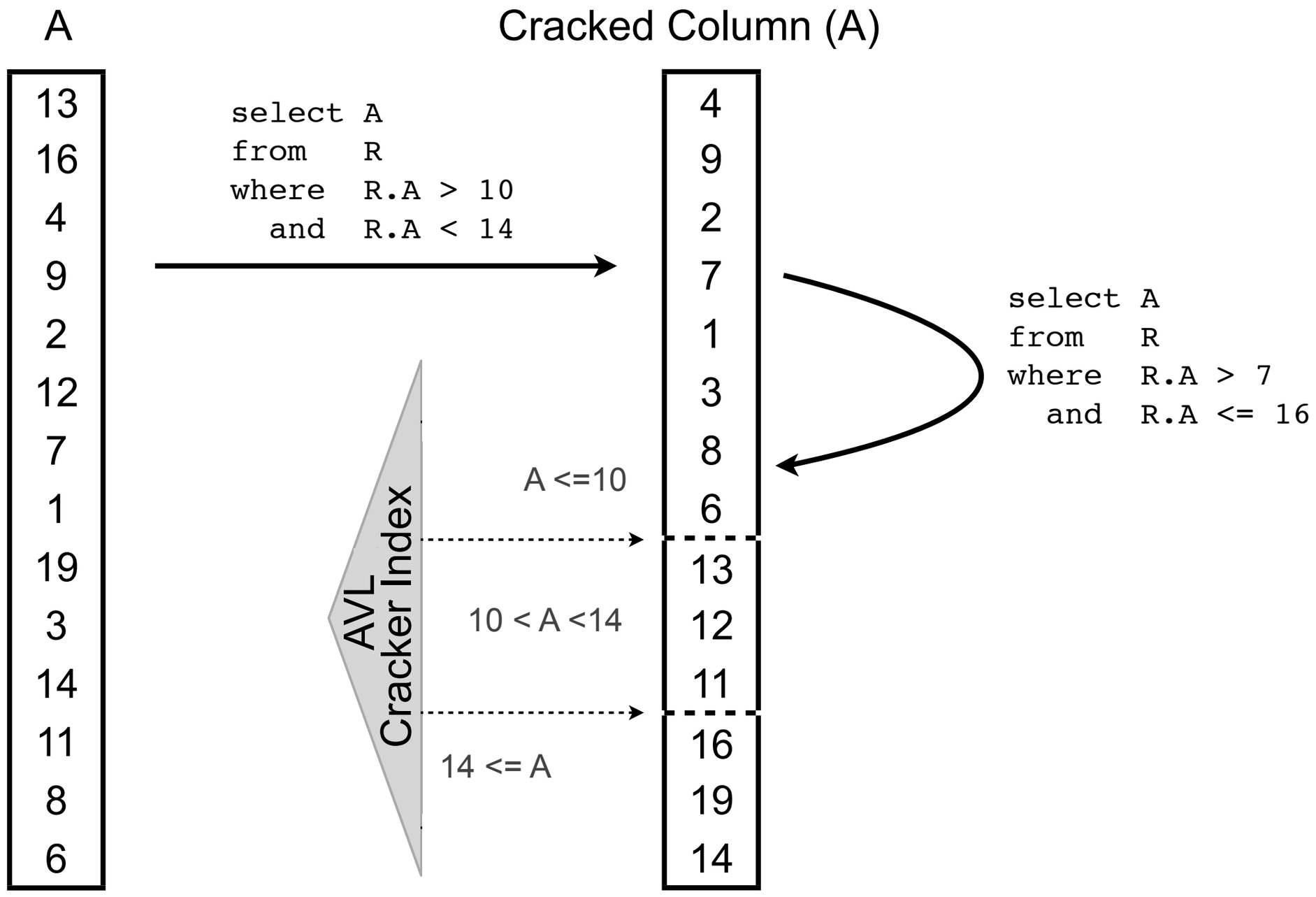
[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking



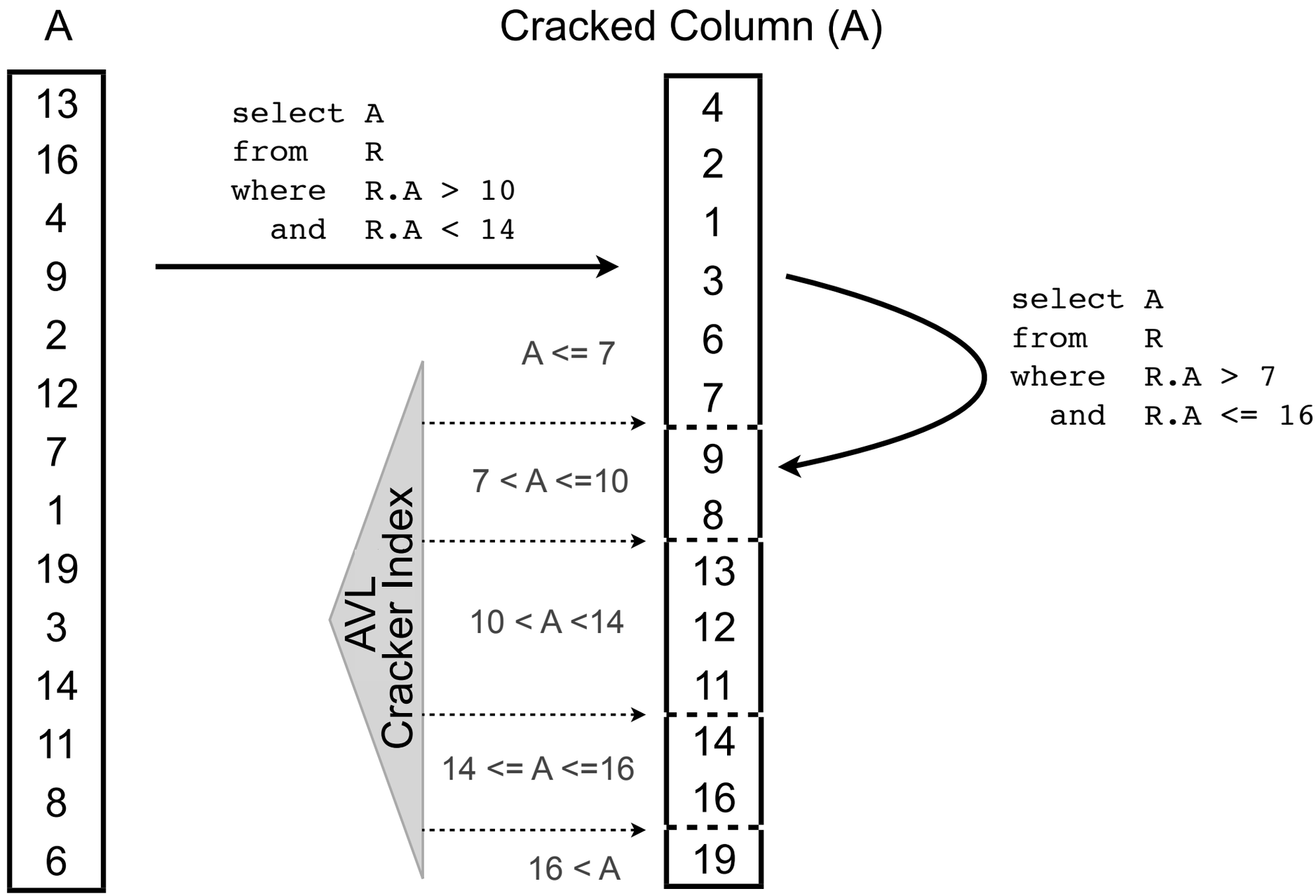
[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking



[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Adaptive Indexing: Standard Cracking



[Database Cracking. S. Idreos, M. Kersten, S. Manegold. In CIDR 2007.]

Motivation

Standard Cracking

Motivation

Standard Cracking

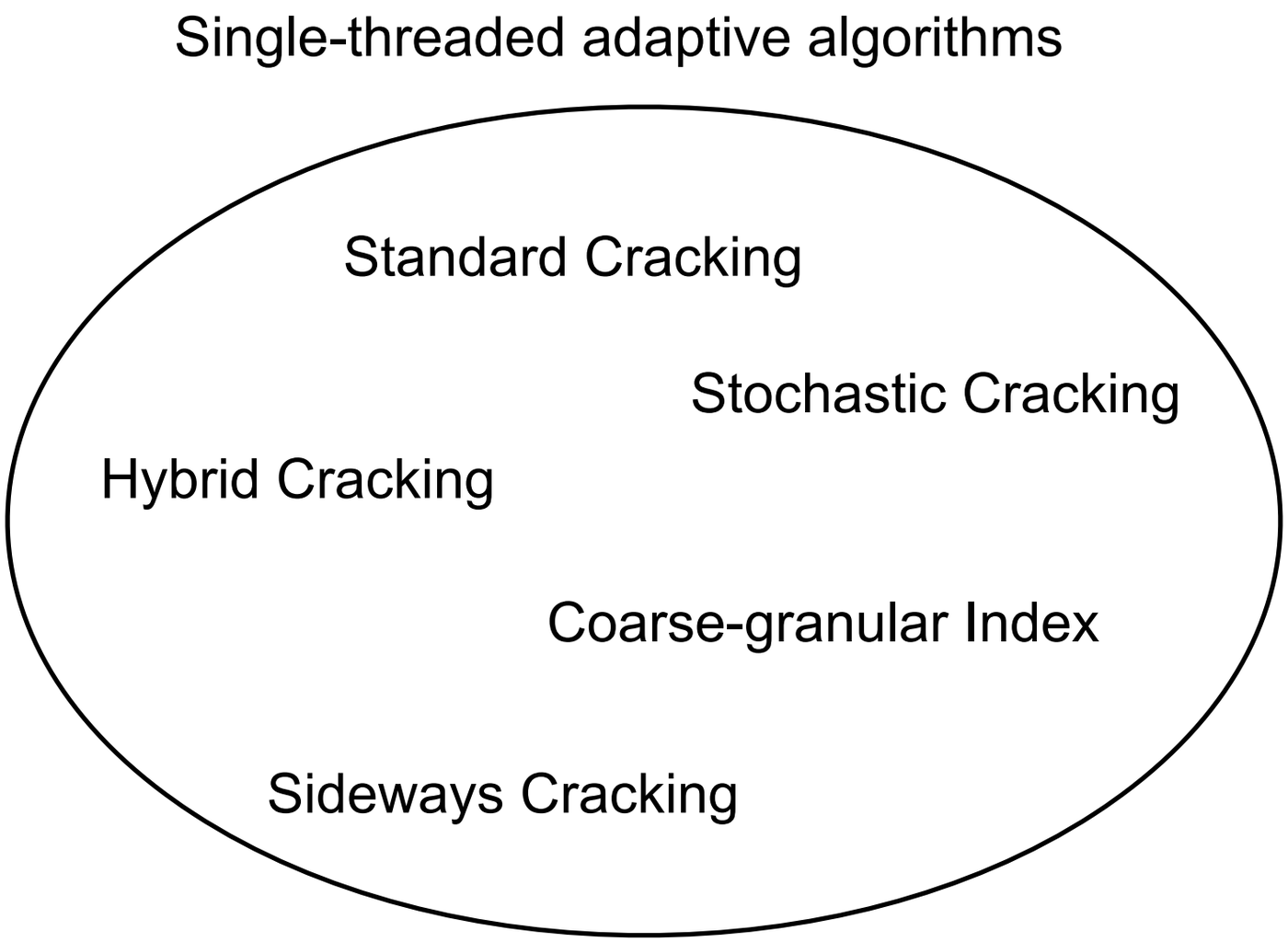
Stochastic Cracking

Hybrid Cracking

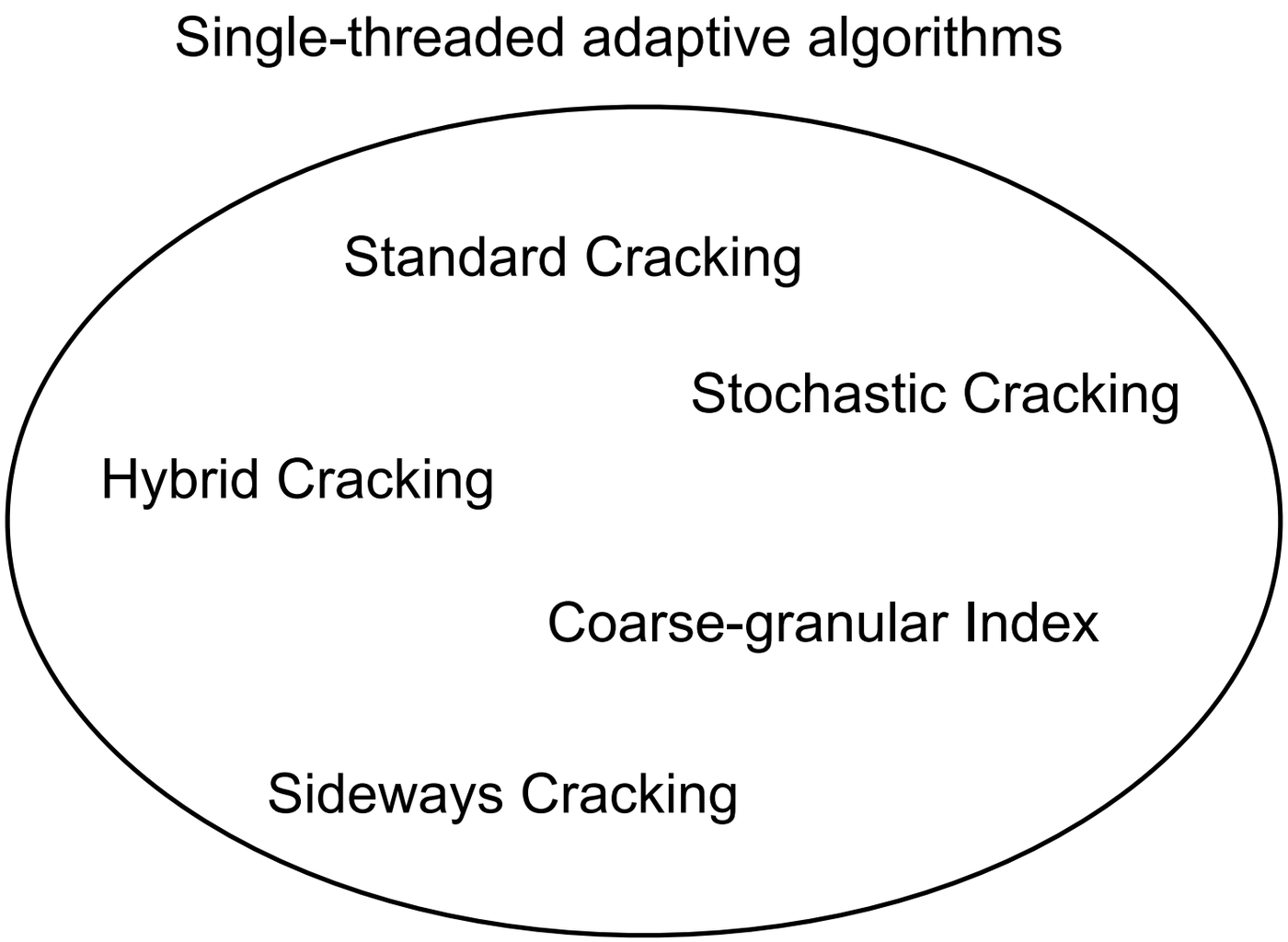
Coarse-granular Index

Sideways Cracking

Motivation


Single-threaded adaptive algorithms
Standard Cracking
Stochastic Cracking
Hybrid Cracking
Coarse-granular Index
Sideways Cracking

Motivation

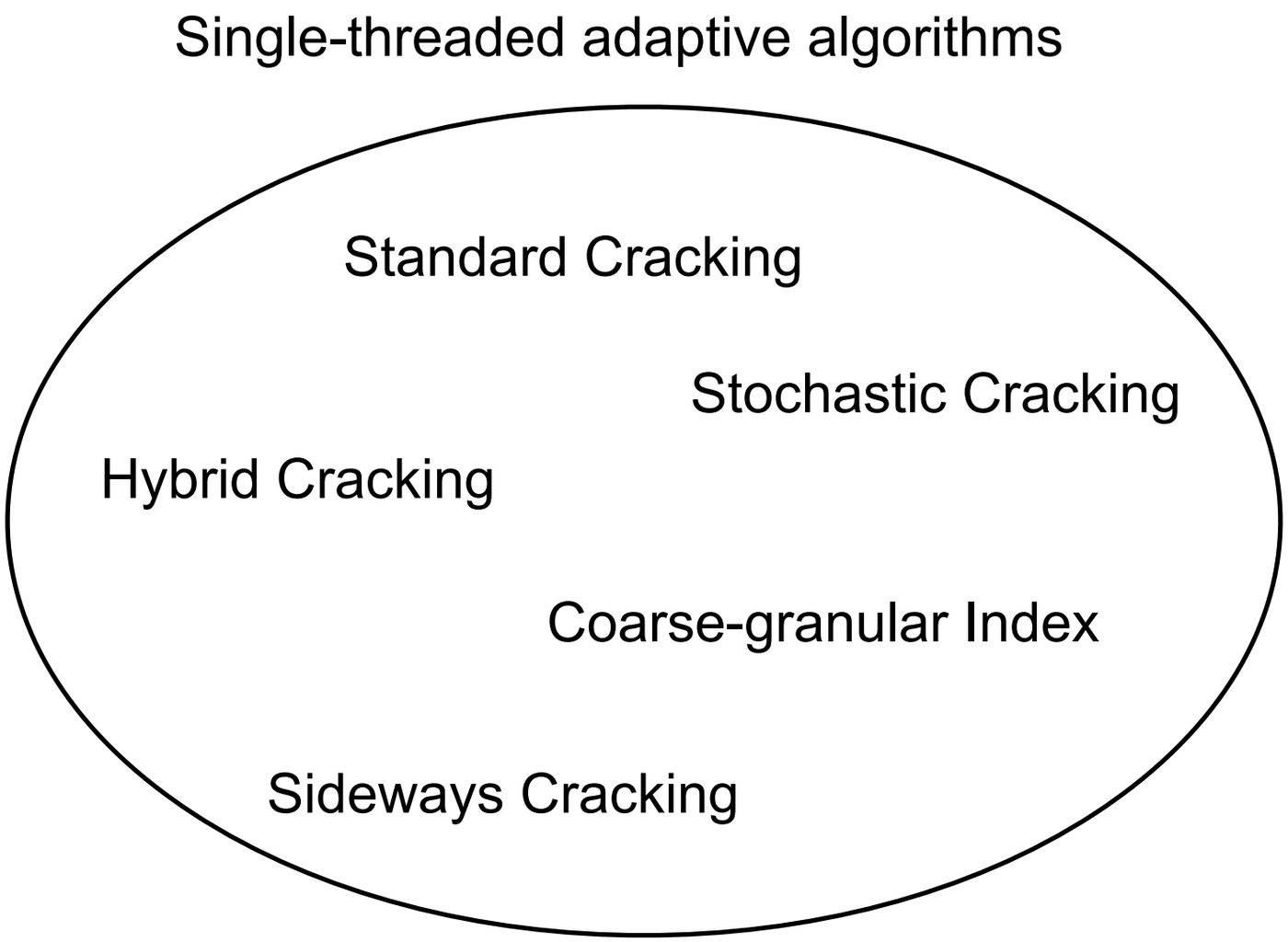


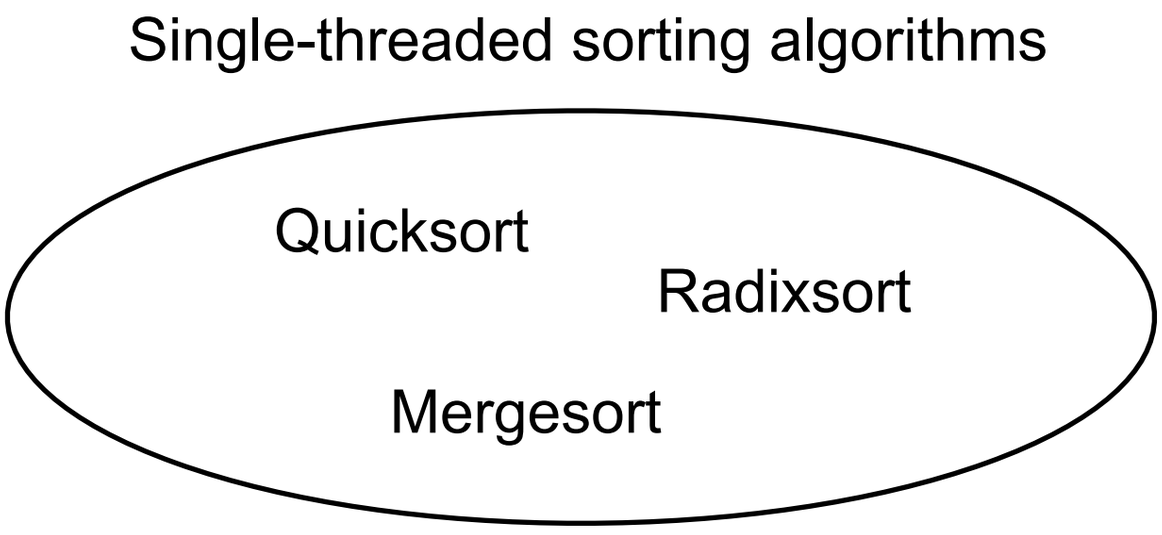
Quicksort

Radixsort

Mergesort

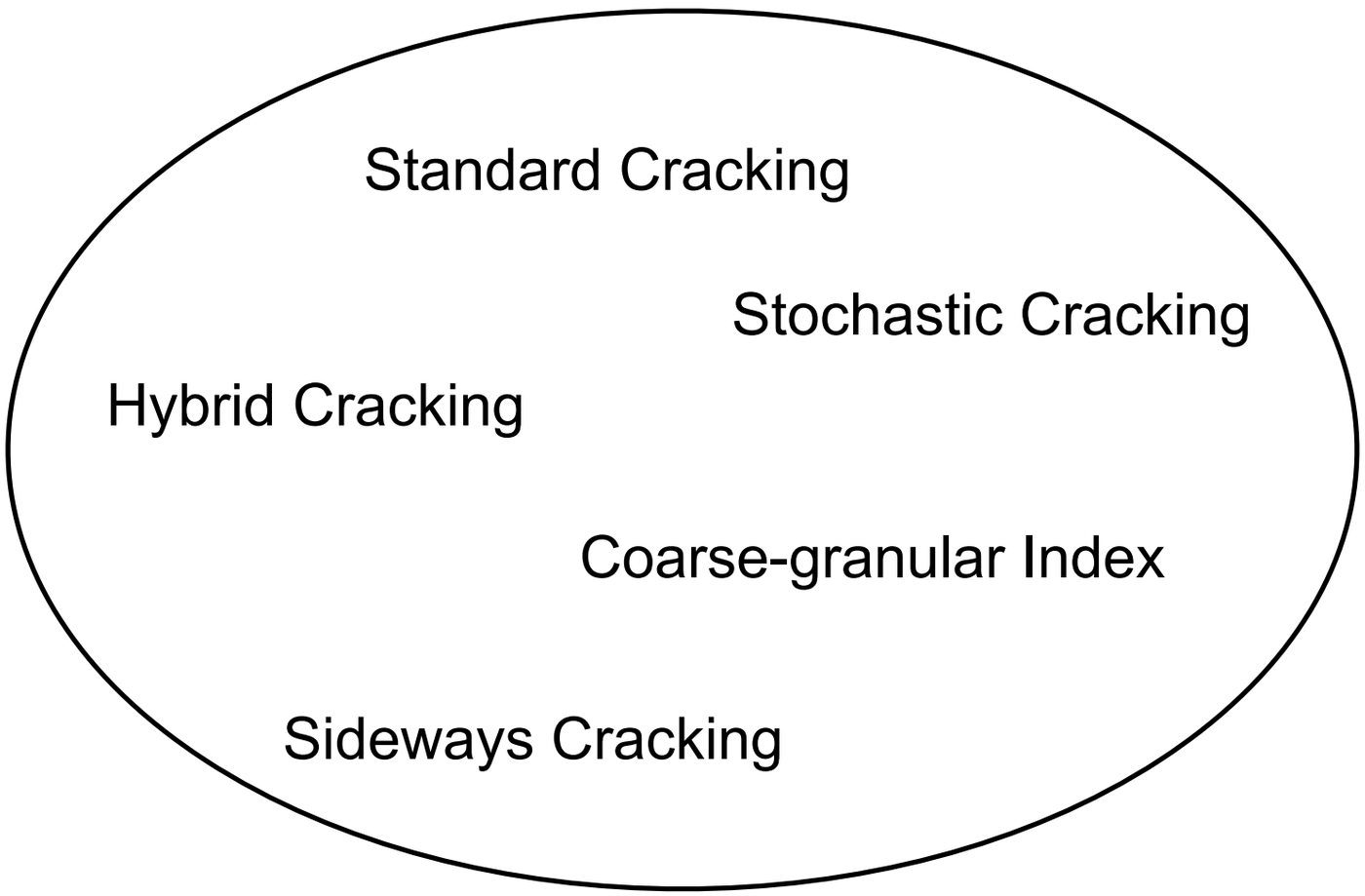
Motivation


Single-threaded adaptive algorithms
Standard Cracking
Stochastic Cracking
Hybrid Cracking
Coarse-granular Index
Sideways Cracking

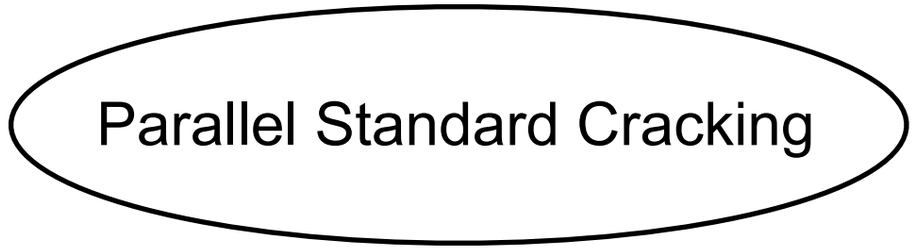

Single-threaded sorting algorithms
Quicksort
Radixsort
Mergesort

Motivation

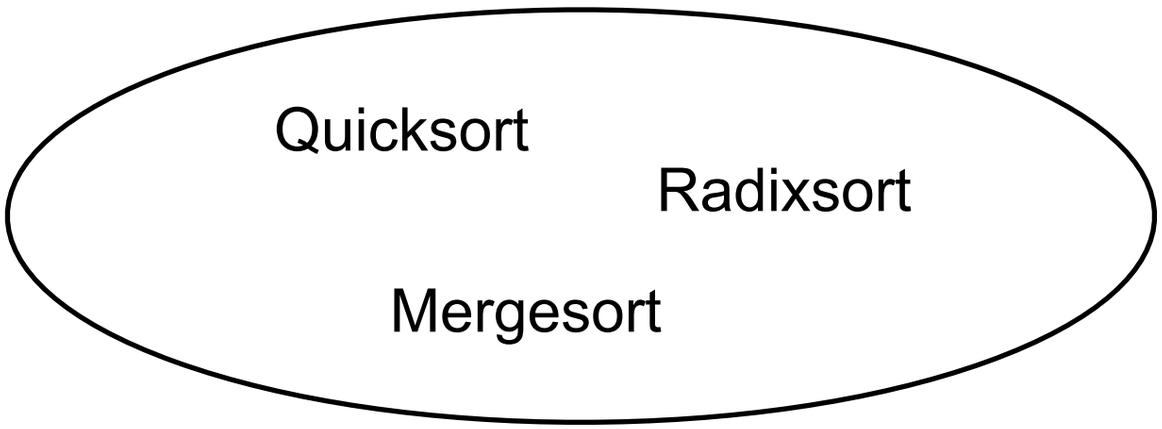
Single-threaded adaptive algorithms



Multi-threaded adaptive algorithms



Single-threaded sorting algorithms



Motivation

Single-threaded adaptive algorithms

Standard Cracking

Stochastic Cracking

Hybrid Cracking

Coarse-granular Index

Sideways Cracking

Single-threaded sorting algorithms

Quicksort

Radixsort

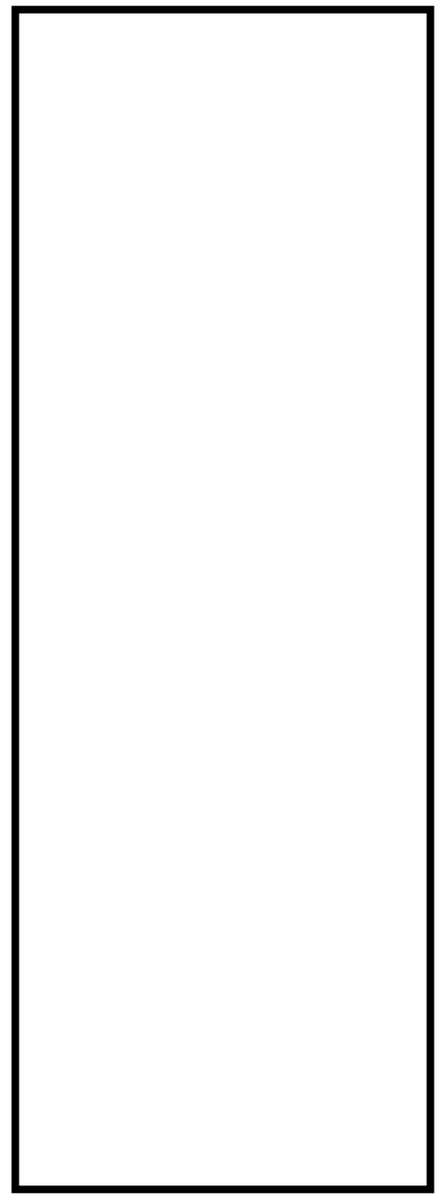
Mergesort

Multi-threaded adaptive algorithms

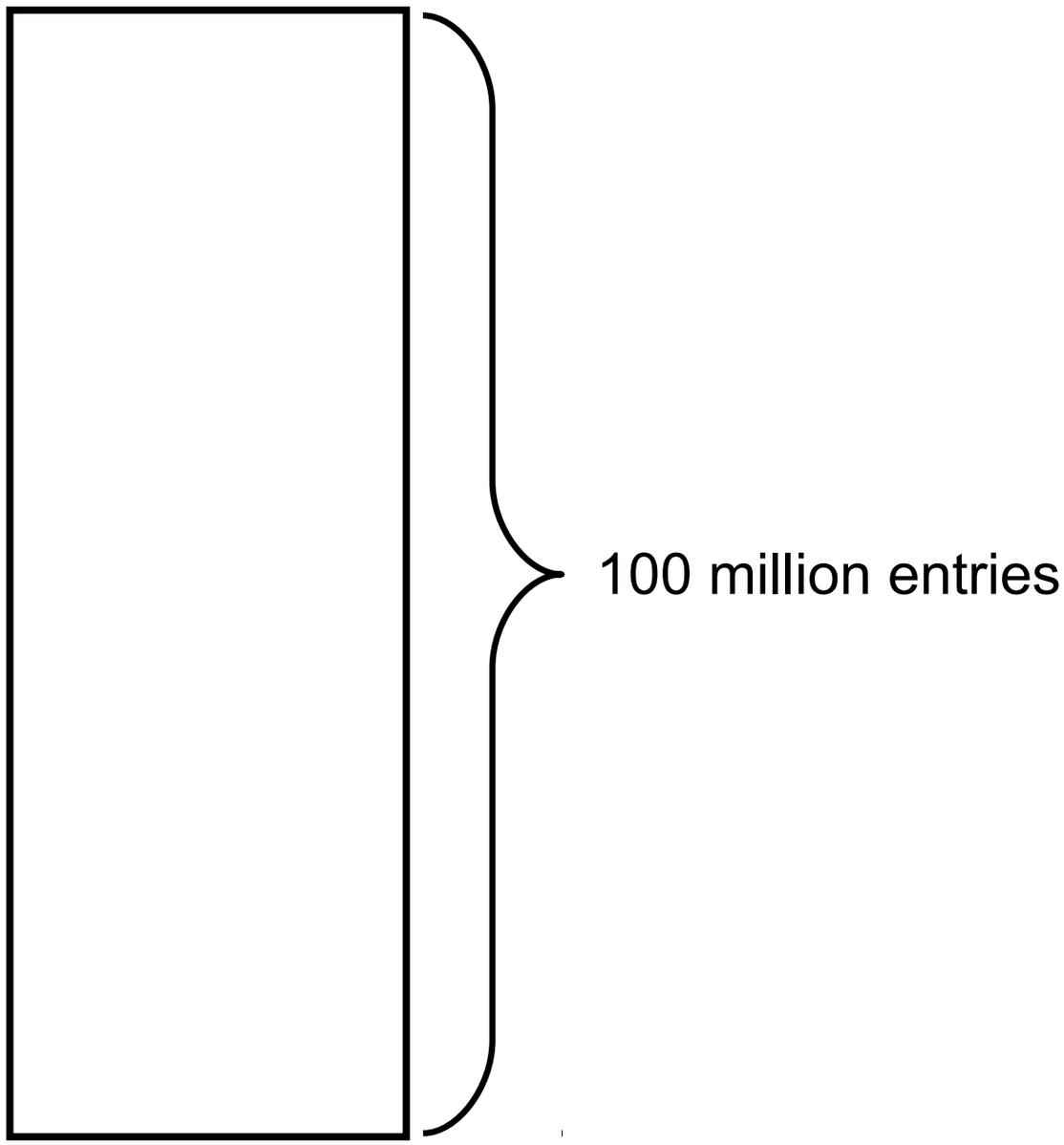
Parallel Standard Cracking

?

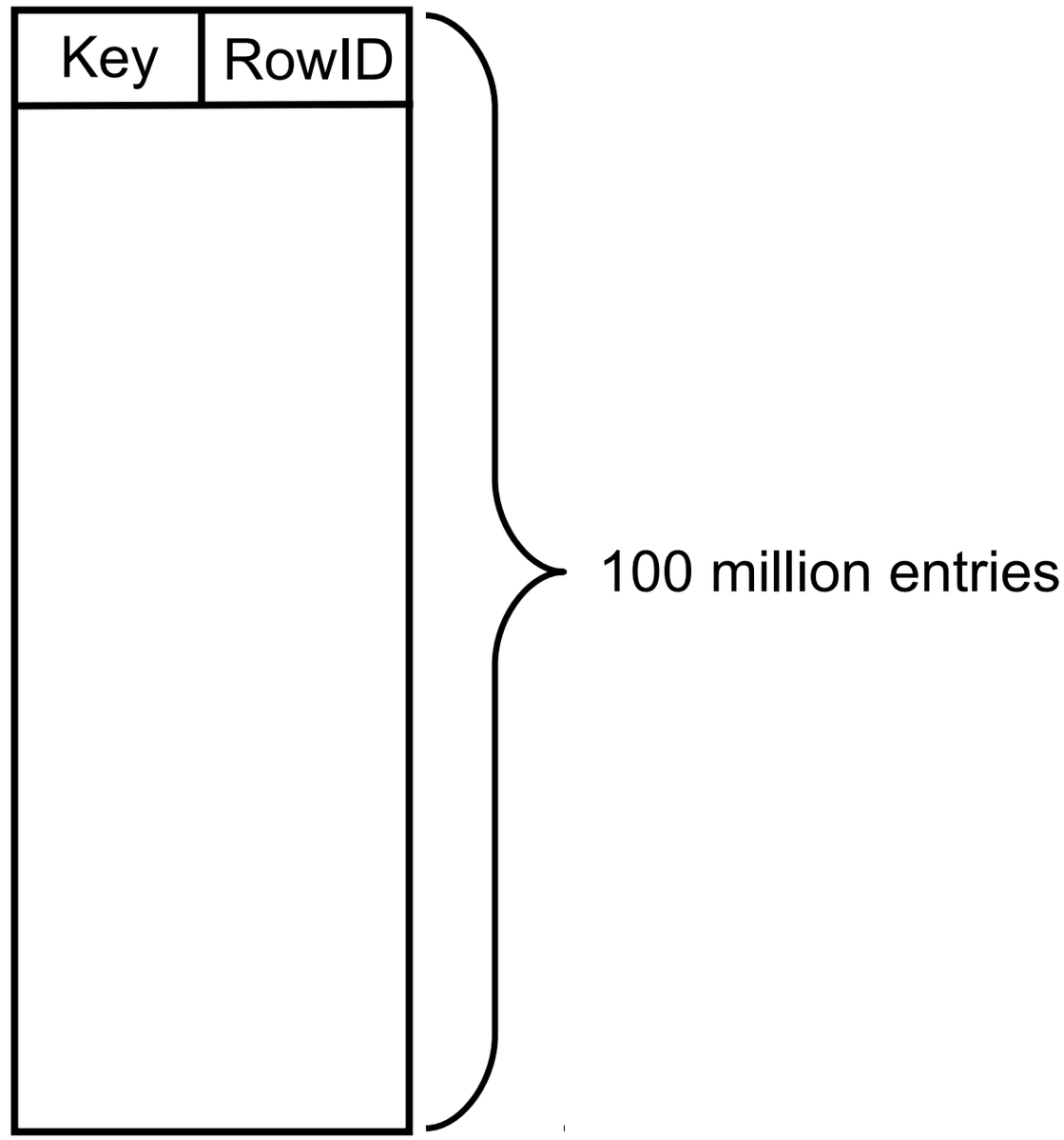
Setup



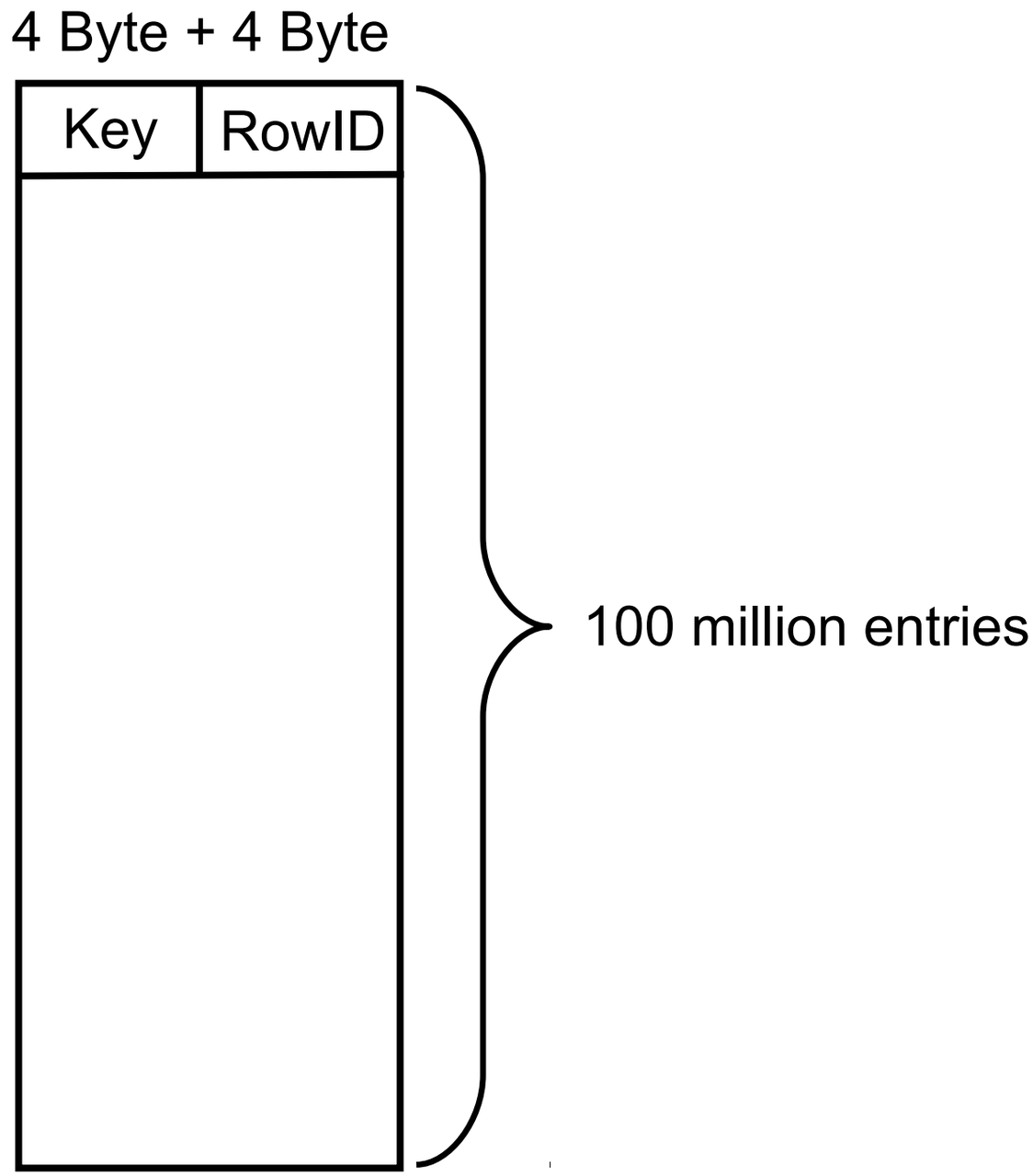
Setup



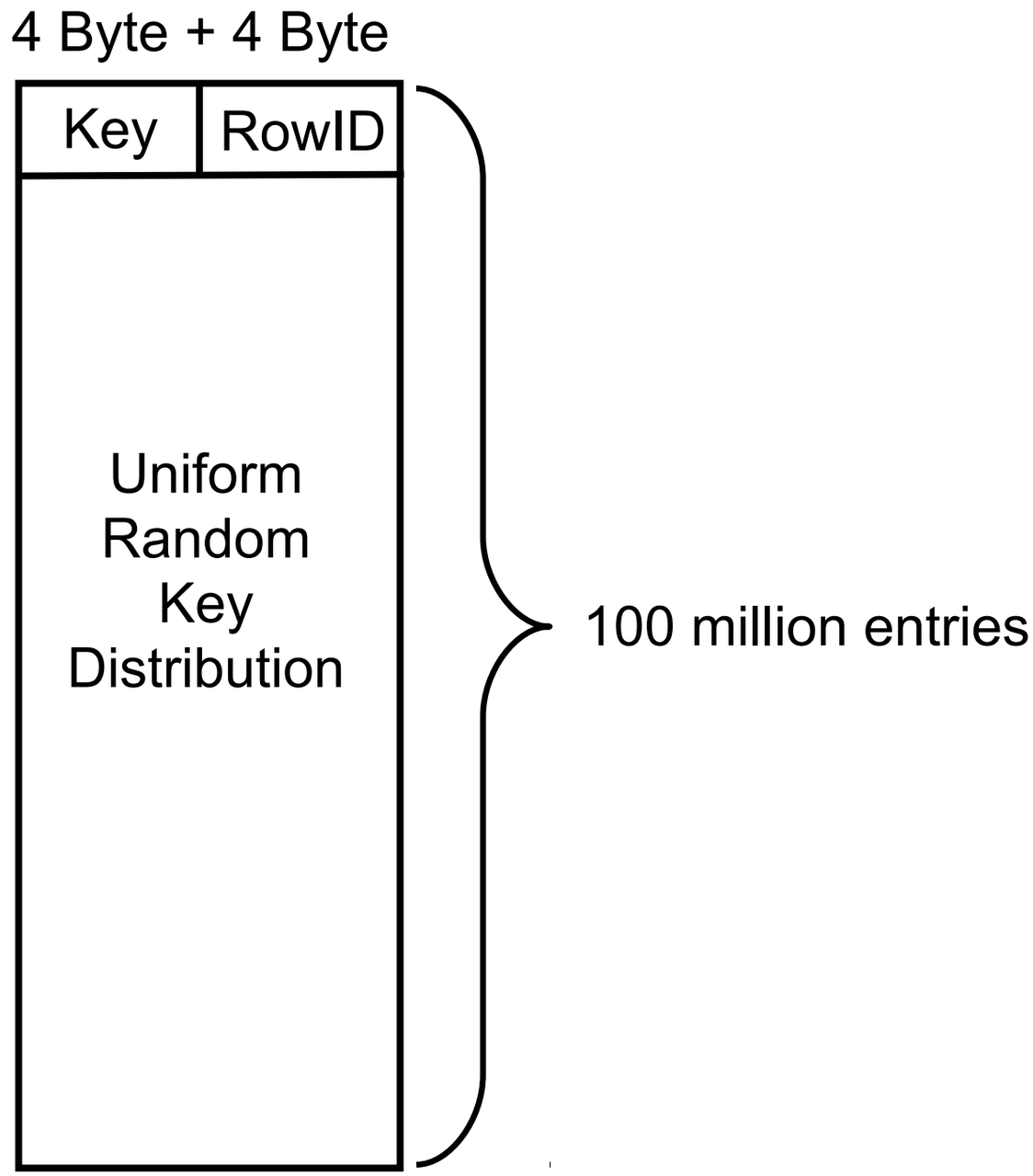
Setup



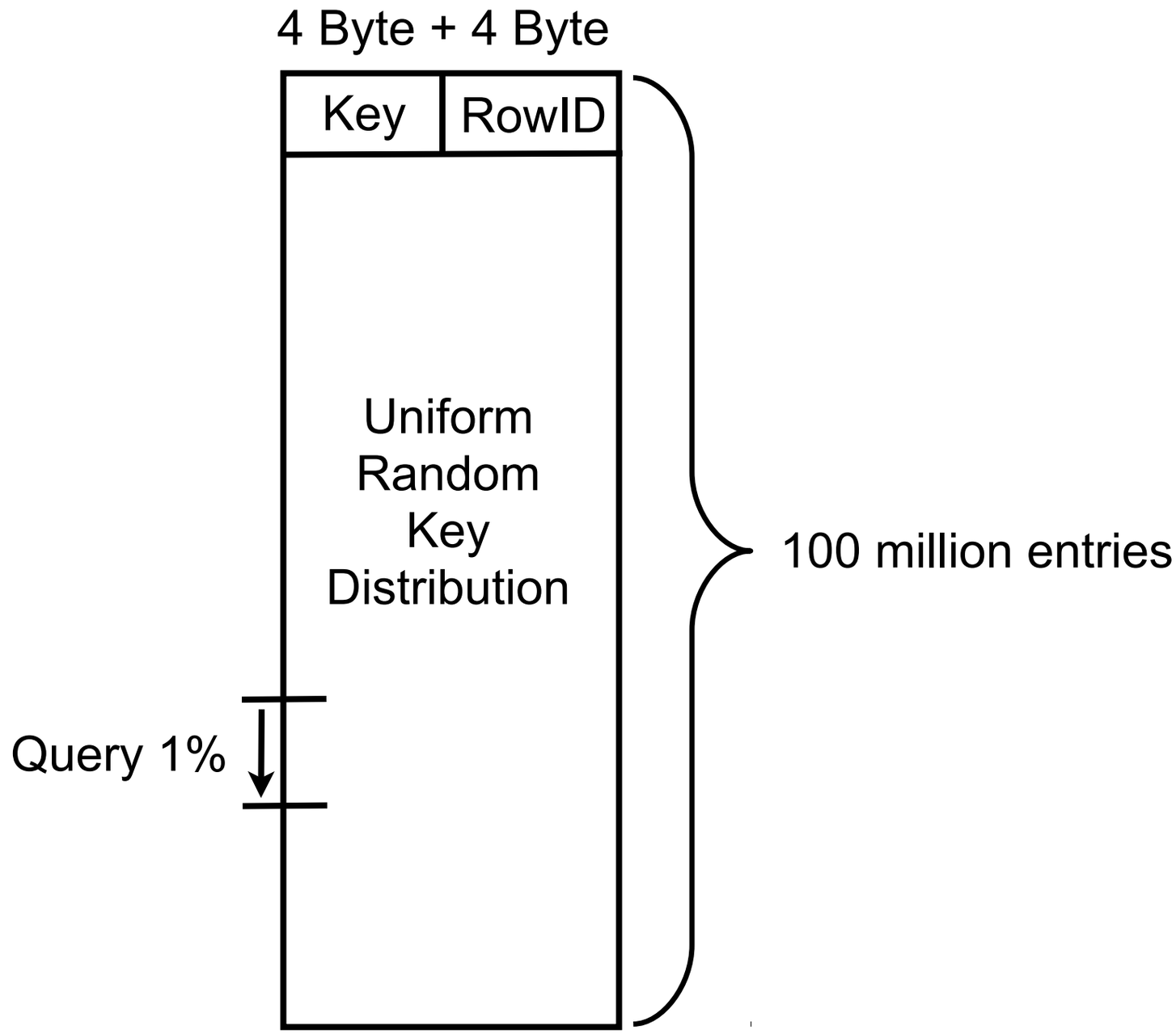
Setup



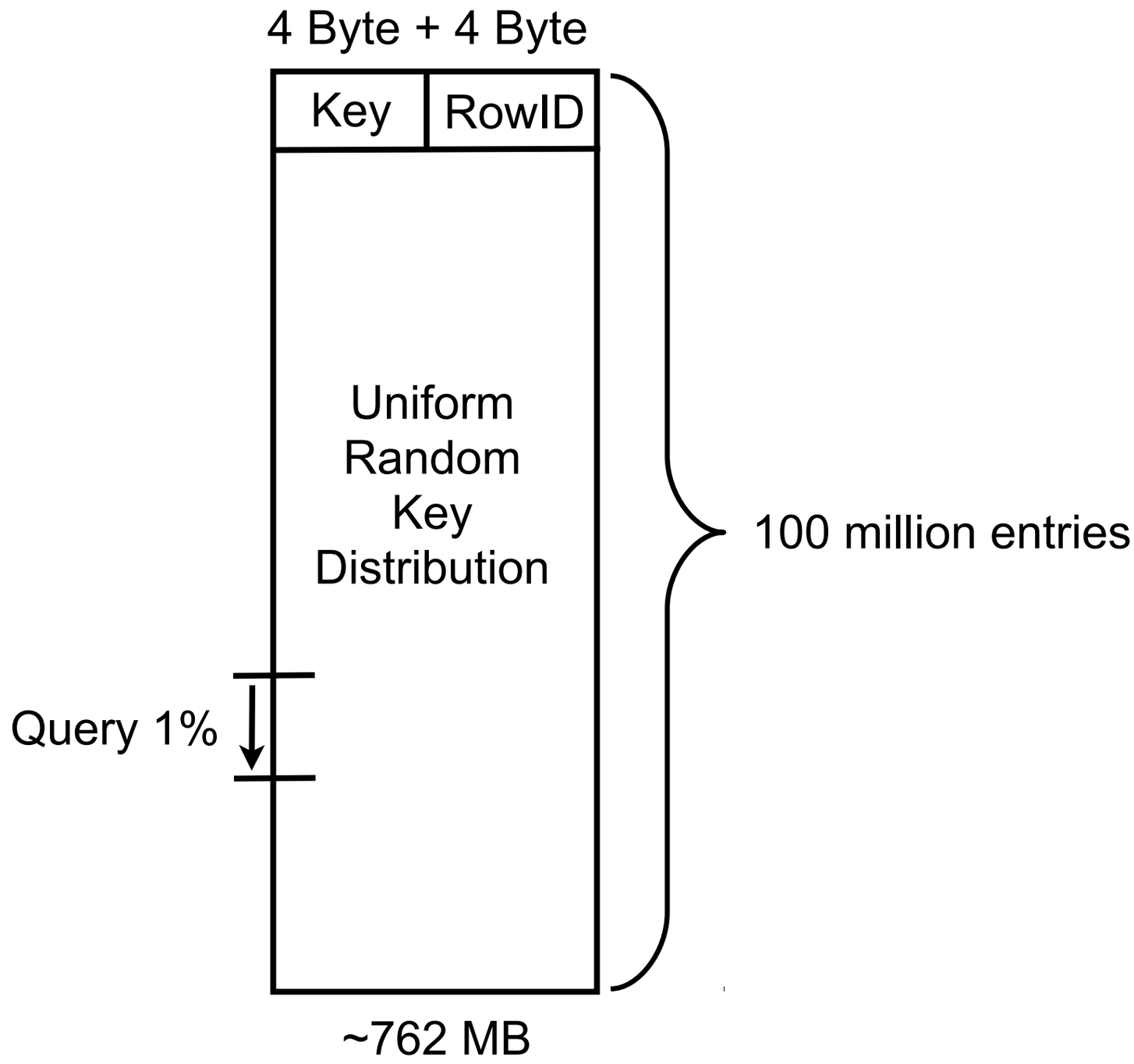
Setup


4 Byte + 4 Byte
Key
RowID
Uniform
Random
Key
Distribution
100 million entries

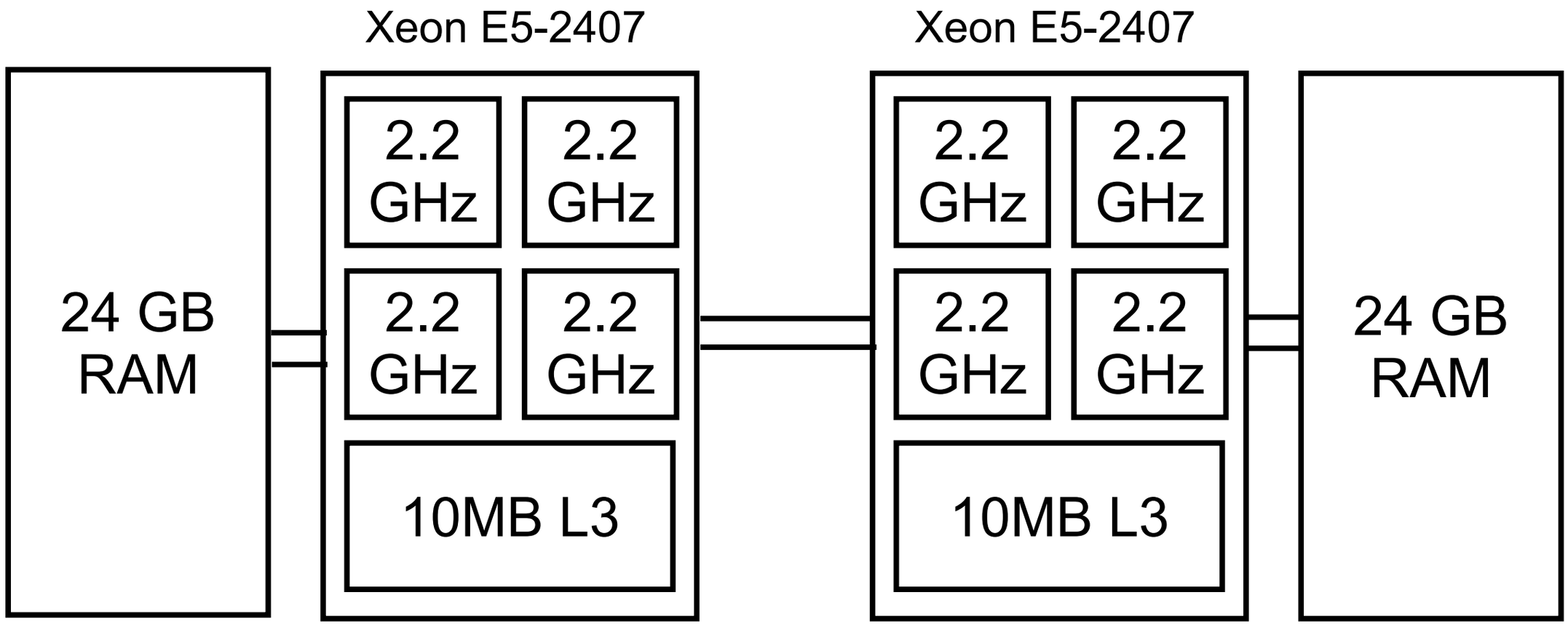
Setup


4 Byte + 4 Byte
Key
RowID
Uniform
Random
Key
Distribution
100 million entries
Query 1%

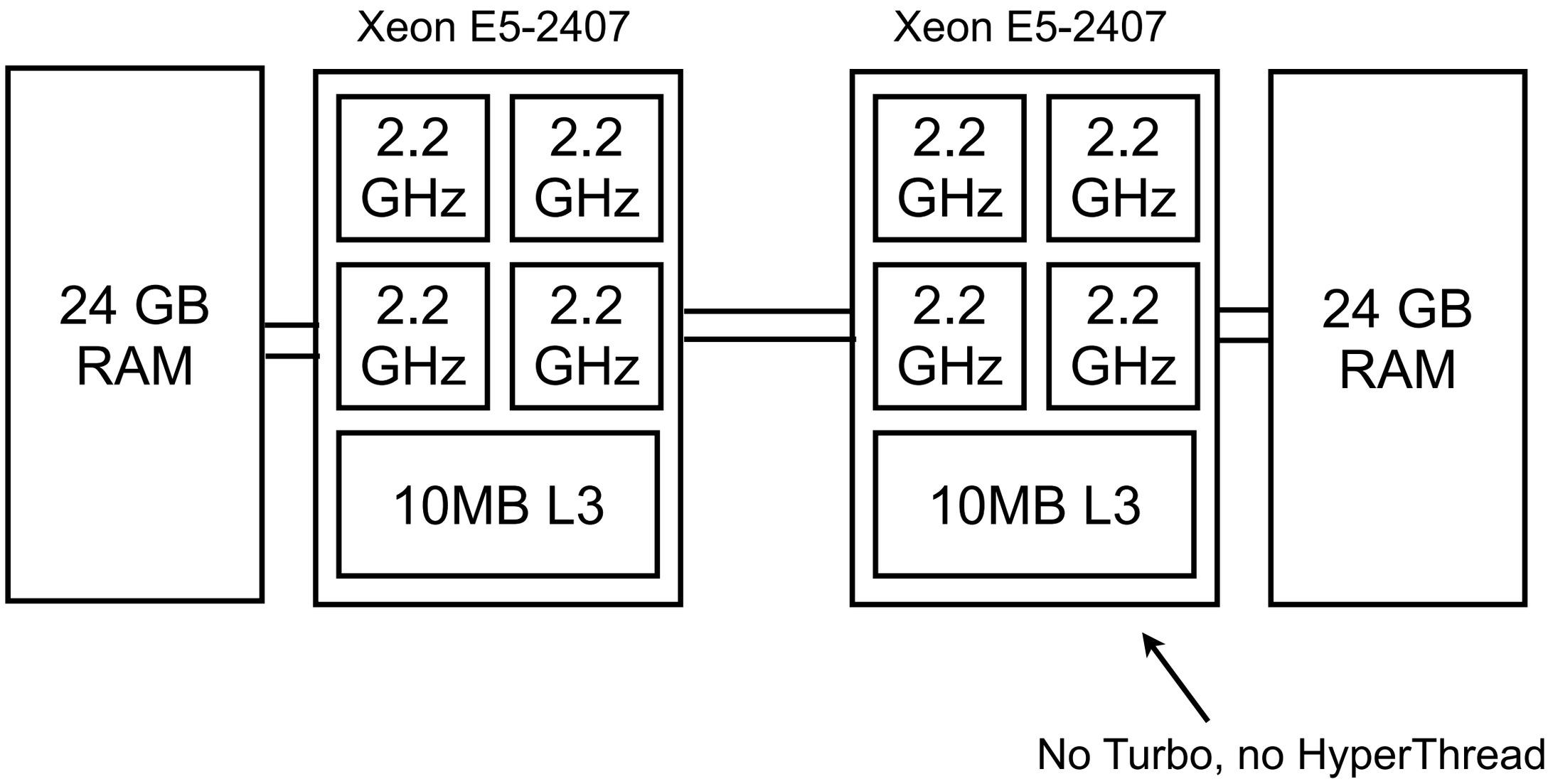
Setup


4 Byte + 4 Byte
Key
RowID
Uniform
Random
Key
Distribution
100 million entries
Query 1%
~762 MB

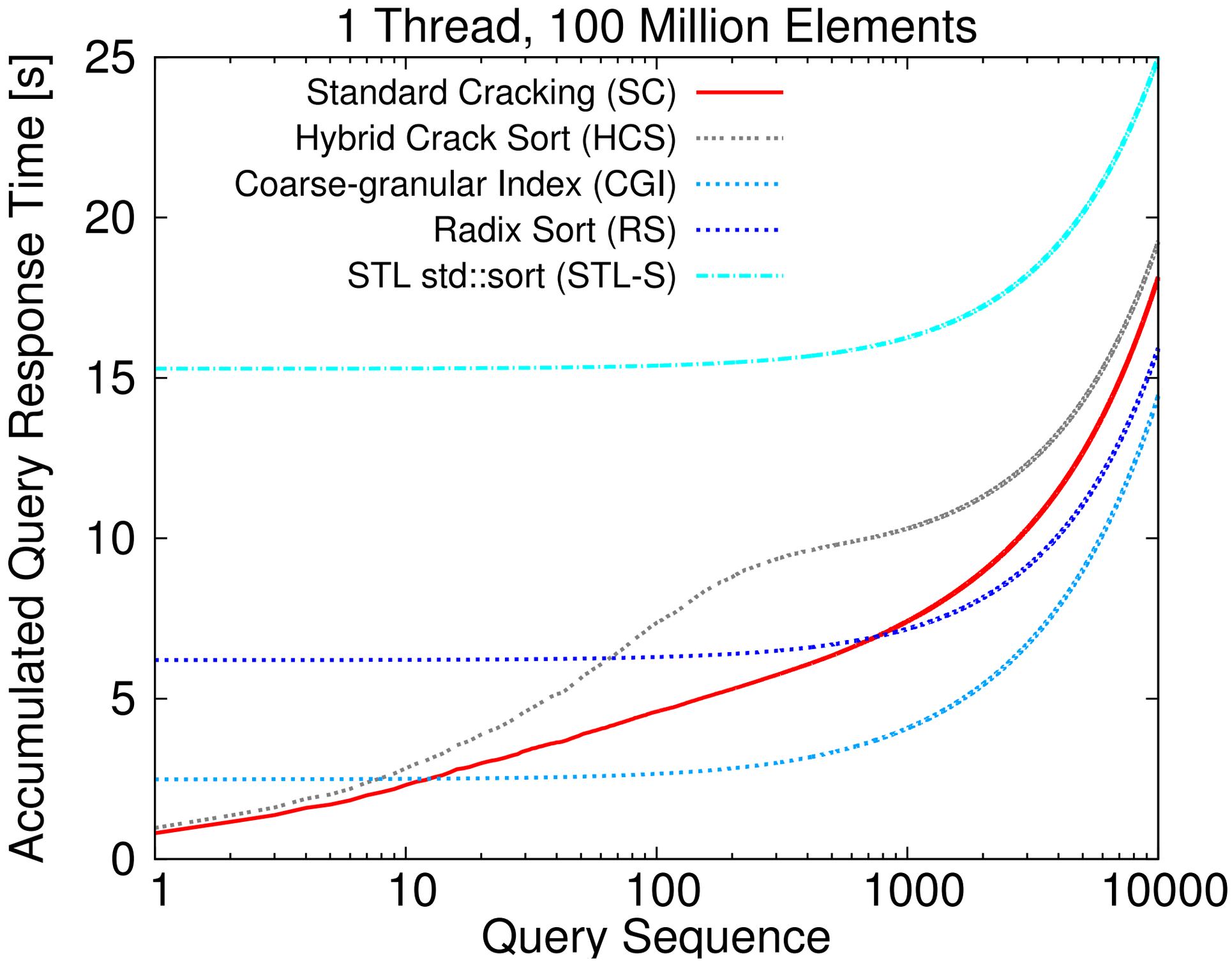
Setup


Xeon E5-2407
Xeon E5-2407
24 GB RAM
2.2 GHz
2.2 GHz
2.2 GHz
2.2 GHz
10MB L3
2.2 GHz
2.2 GHz
2.2 GHz
2.2 GHz
10MB L3
24 GB RAM

Setup

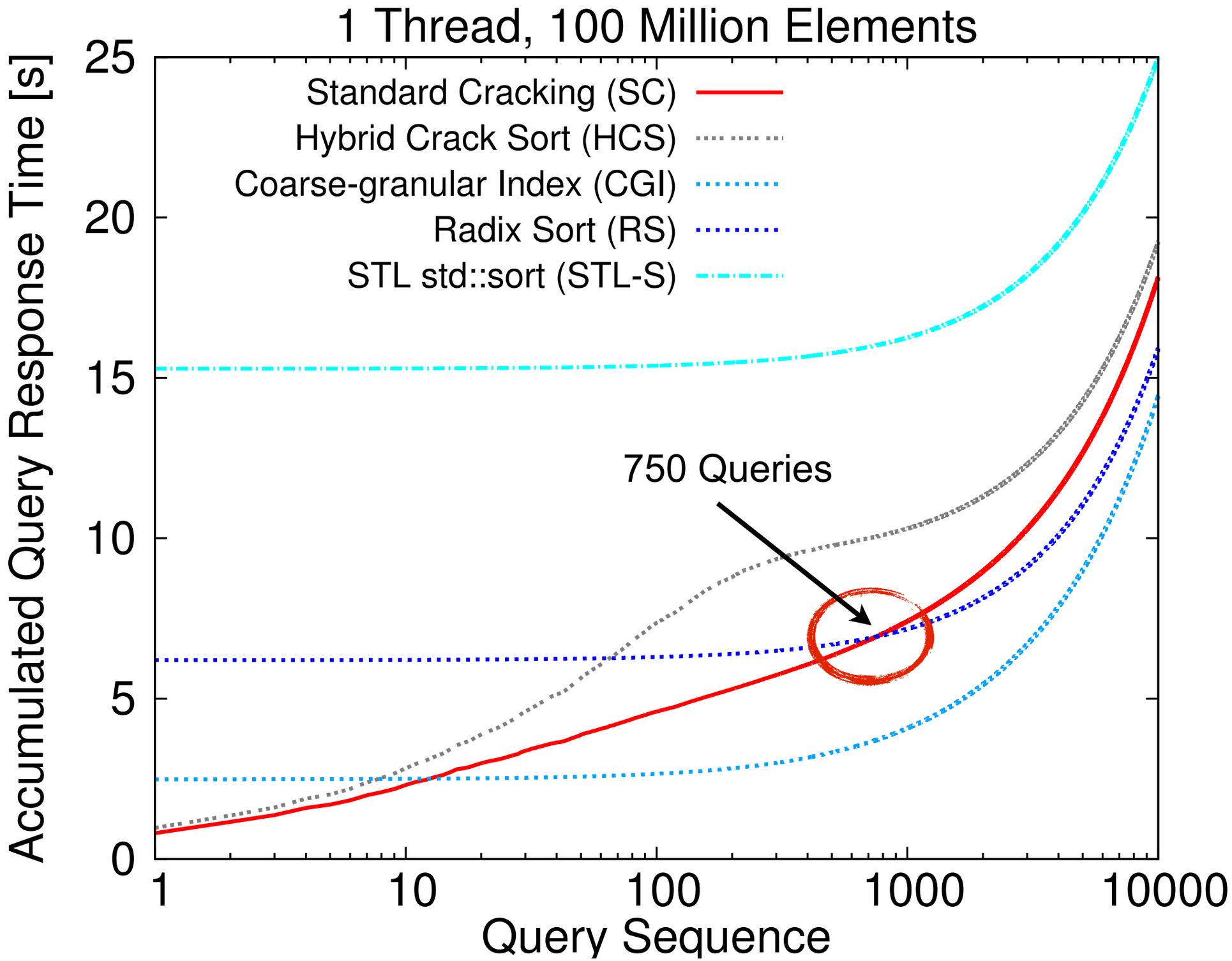

Xeon E5-2407
Xeon E5-2407
24 GB RAM
2.2 GHz
2.2 GHz
2.2 GHz
2.2 GHz
10MB L3
2.2 GHz
2.2 GHz
2.2 GHz
2.2 GHz
10MB L3
24 GB RAM
No Turbo, no HyperThreading

Single-threaded algorithms



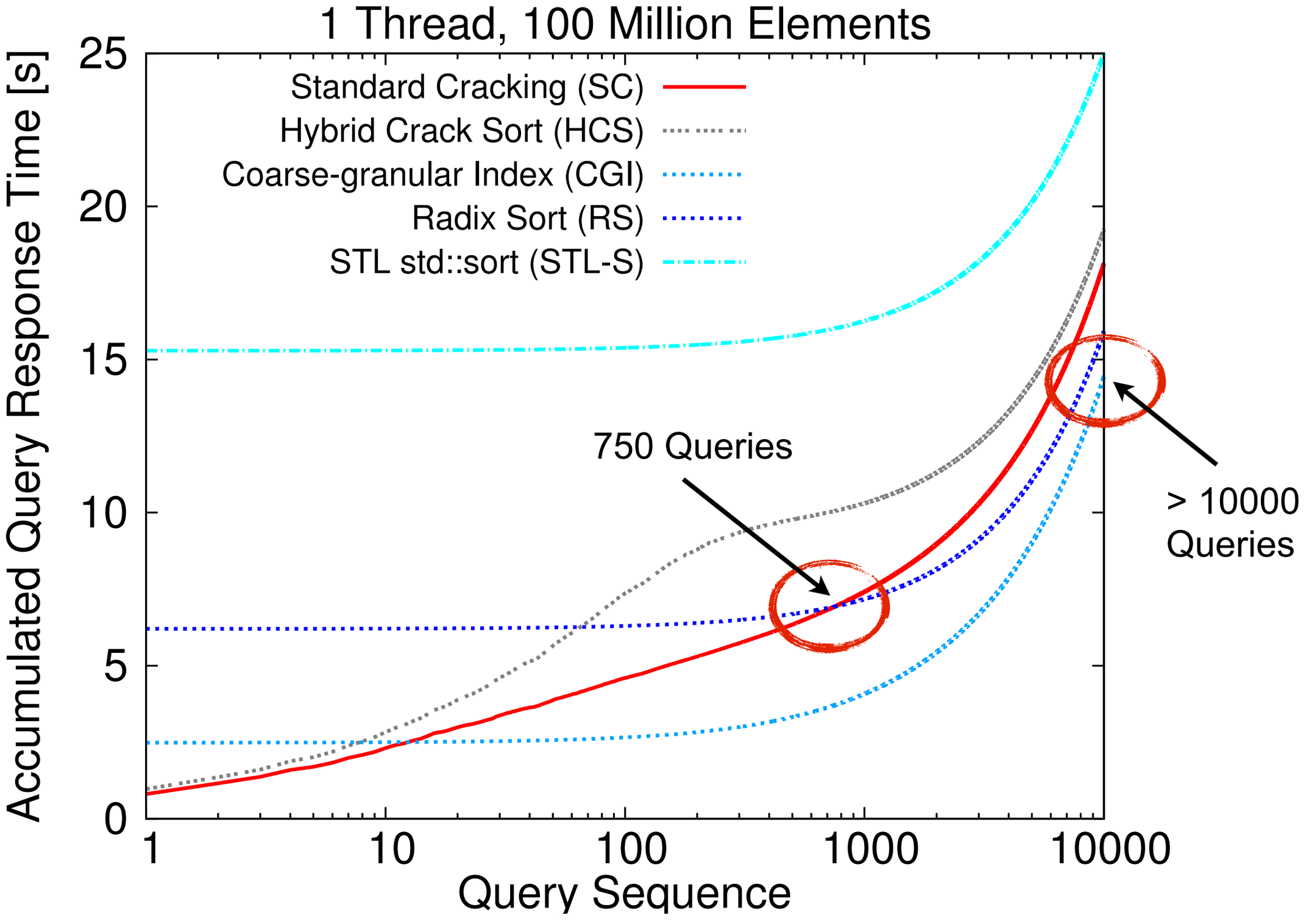
[The Uncracked Pieces in Database Cracking. F. M. Schuhknecht, A. Jindal, J. Dittrich. In PVLDB 2013]

Single-threaded algorithms



[The Uncracked Pieces in Database Cracking. F. M. Schuhknecht, A. Jindal, J. Dittrich. In PVLDB 2013]

Single-threaded algorithms



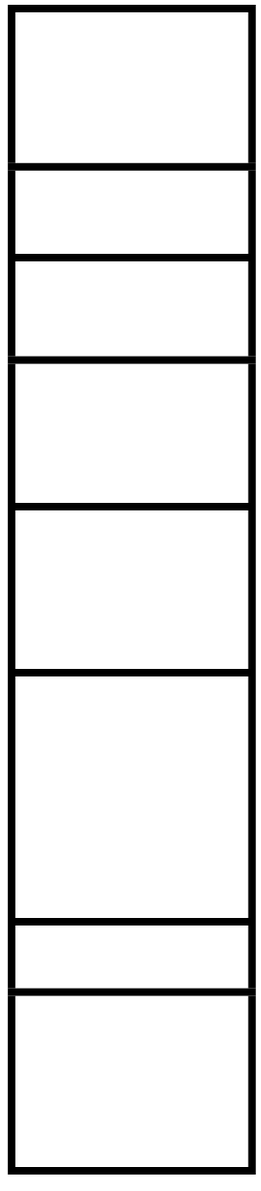
[The Uncracked Pieces in Database Cracking. F. M. Schuhknecht, A. Jindal, J. Dittrich. In PVLDB 2013]

Multi-threaded environments?



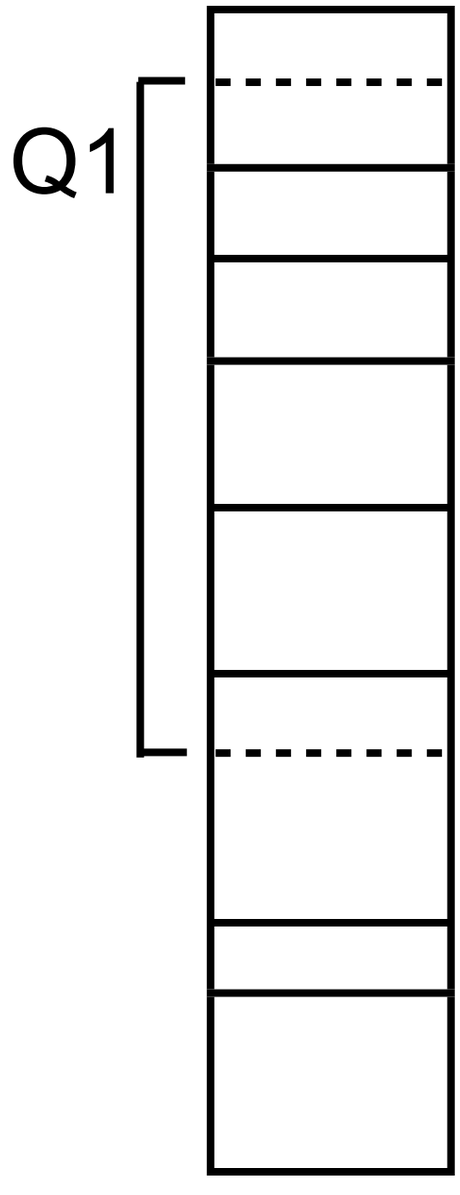
Multi-threaded algorithms!

Multi-threaded algorithms:
Parallel Standard Cracking (P-SC)



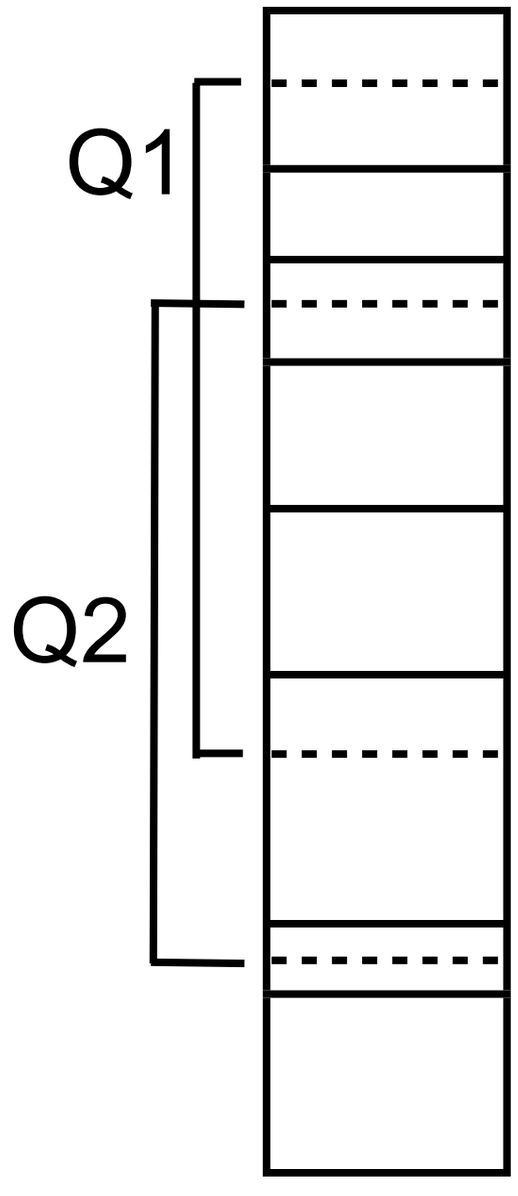
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms:
Parallel Standard Cracking (P-SC)



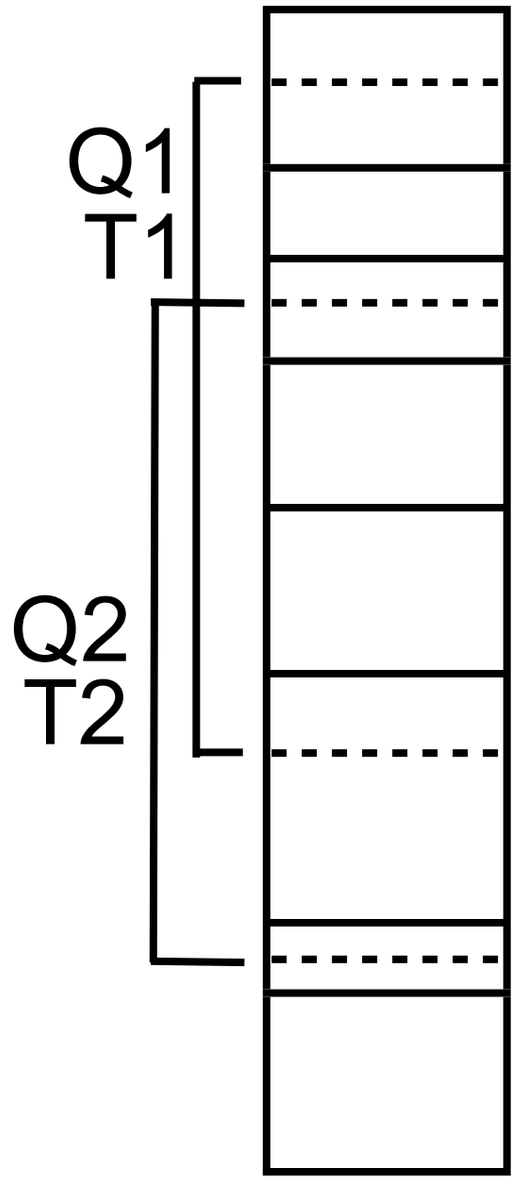
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms:
Parallel Standard Cracking (P-SC)



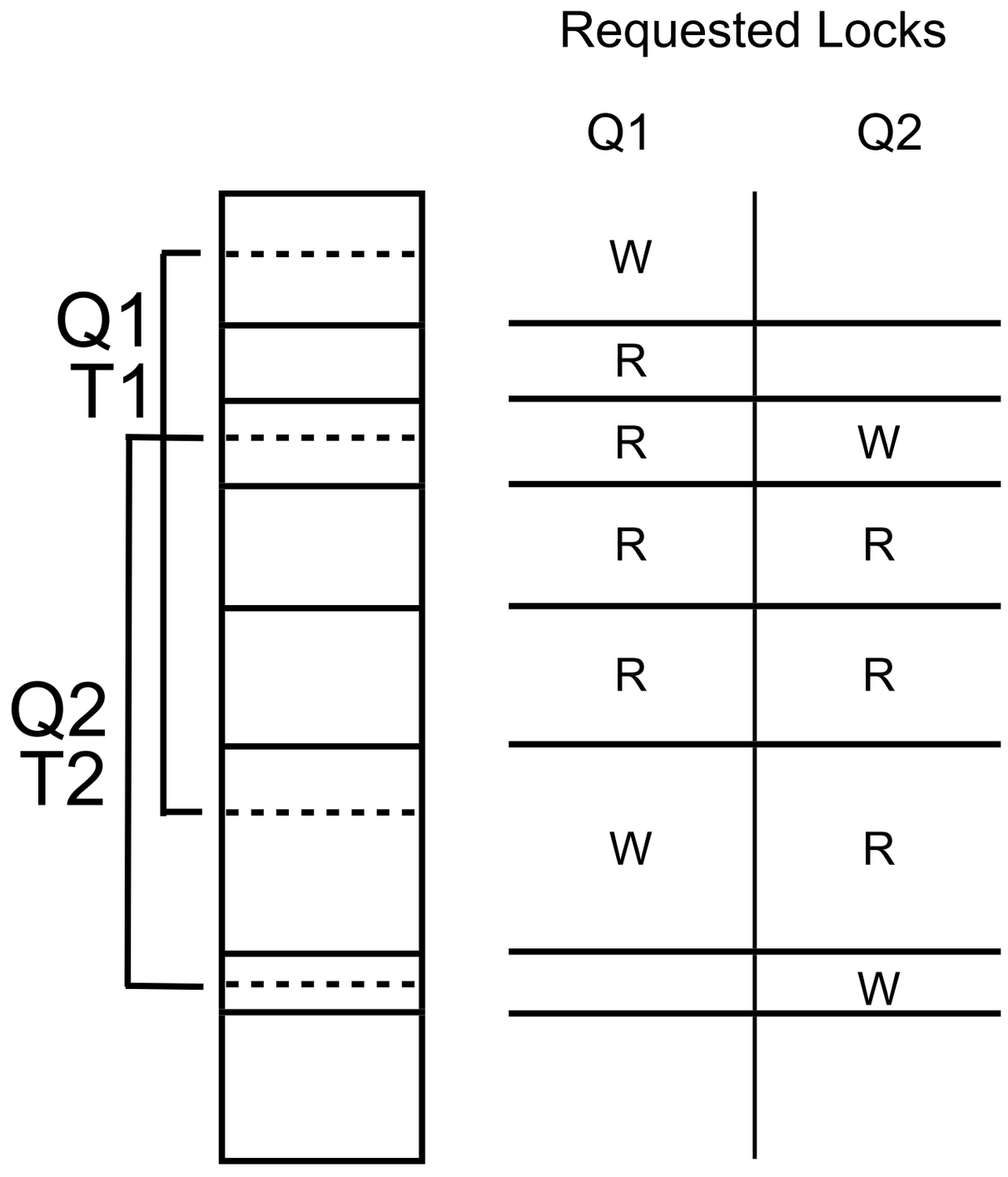
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms:
Parallel Standard Cracking (P-SC)



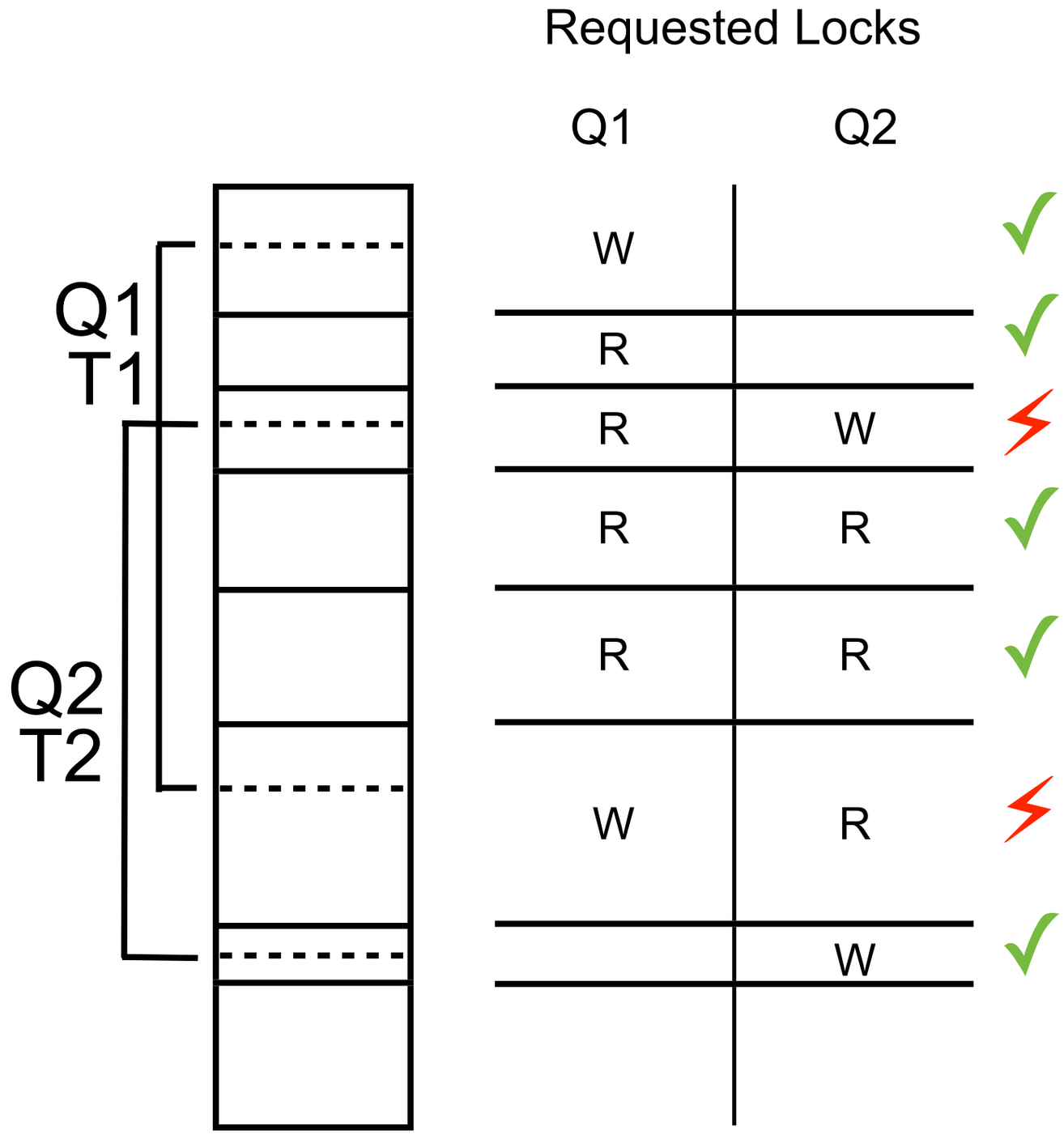
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms: Parallel Standard Cracking (P-SC)



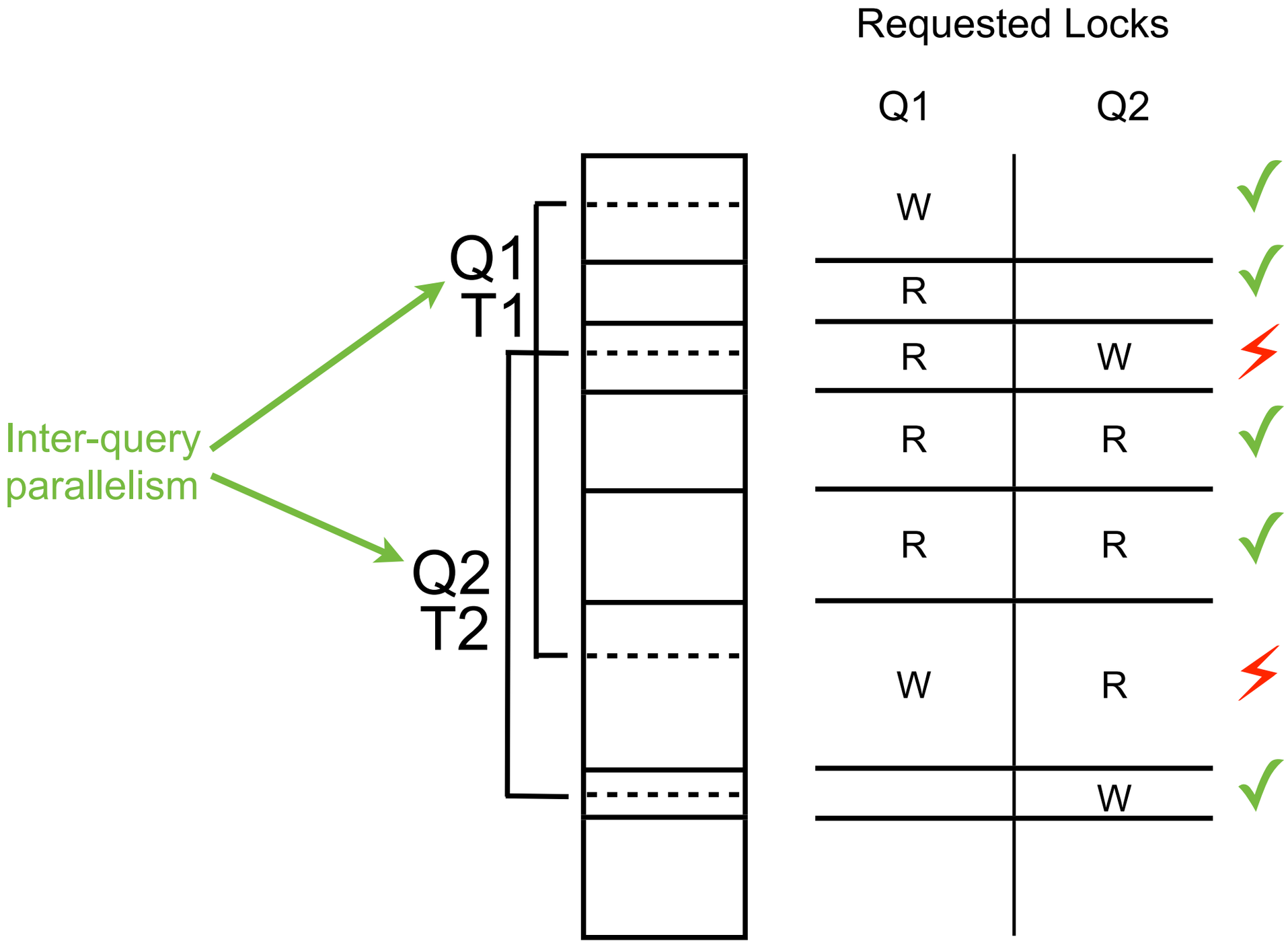
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms: Parallel Standard Cracking (P-SC)



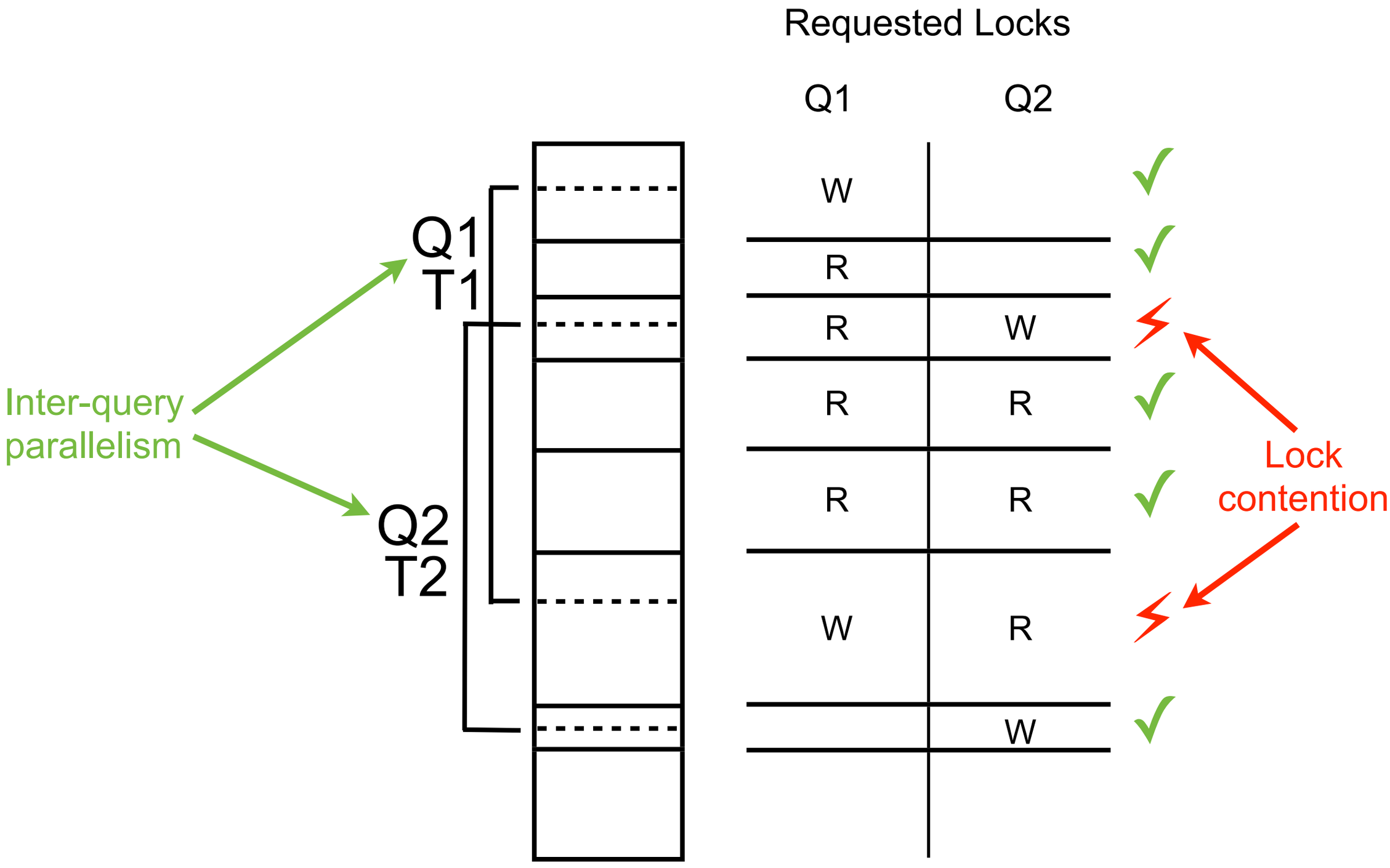
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms: Parallel Standard Cracking (P-SC)



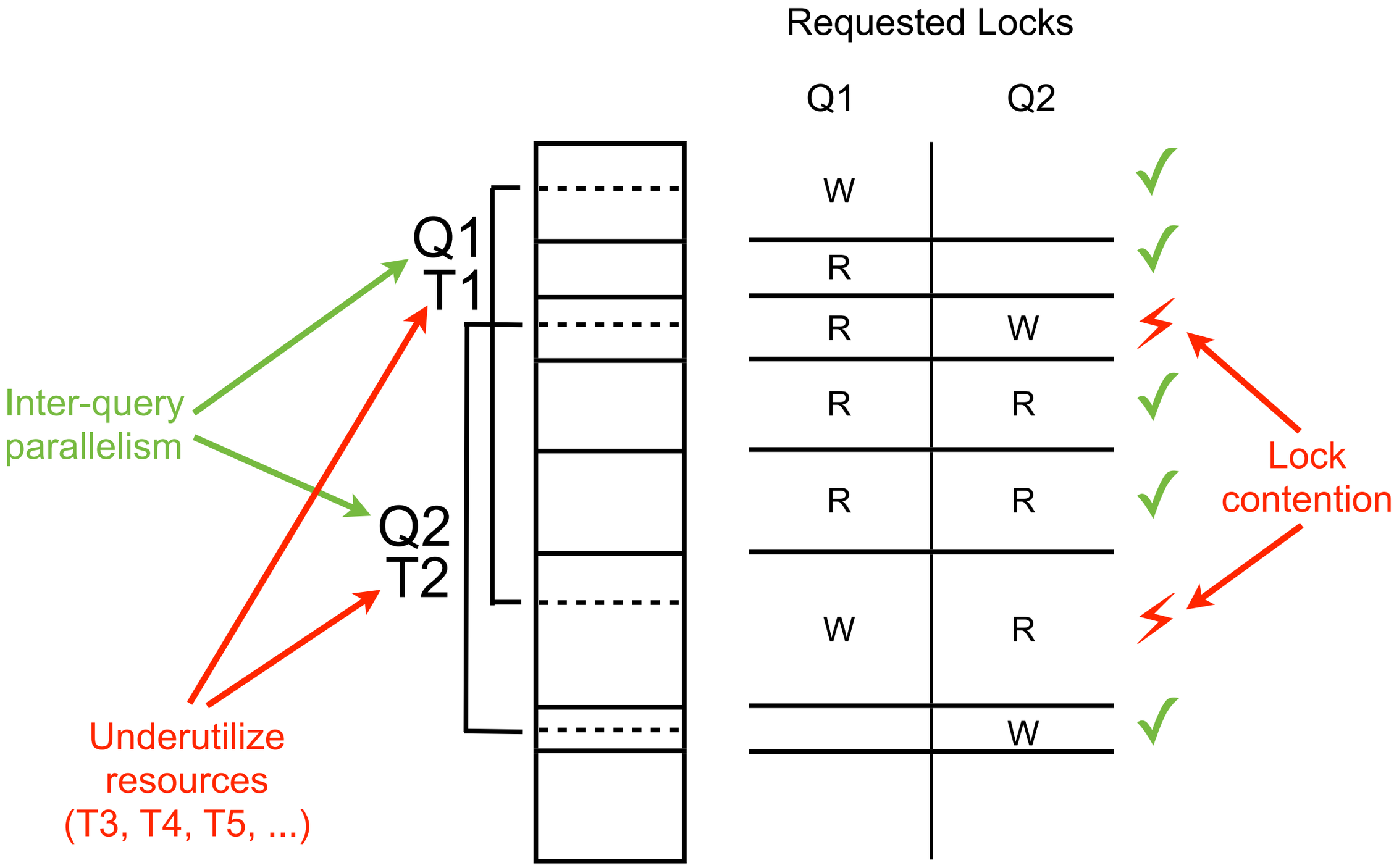
[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

Multi-threaded algorithms: Parallel Standard Cracking (P-SC)



[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

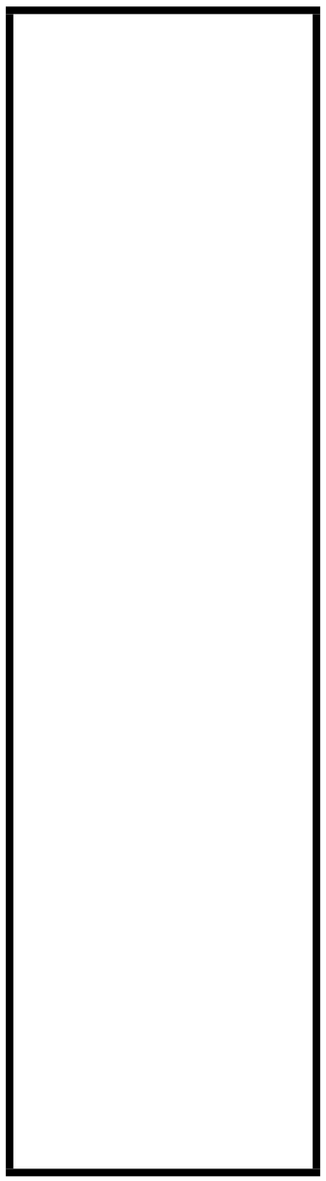
Multi-threaded algorithms: Parallel Standard Cracking (P-SC)



[Concurrency control for adaptive indexing. G.Graefe, F.Halim, S.Idreos, H.Kuno, S.Manegold. In PVLDB 2013]

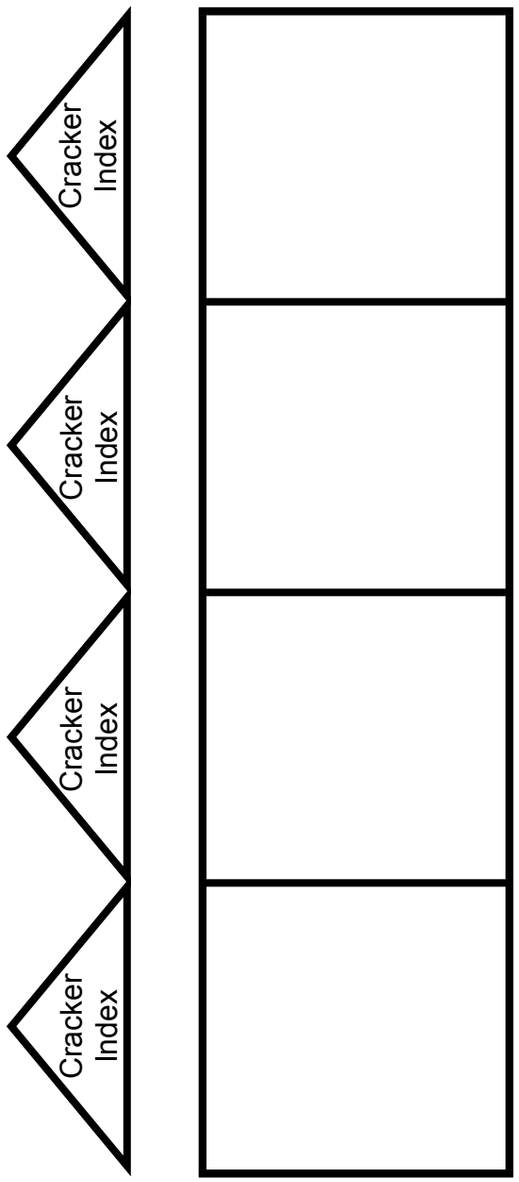
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)

Query



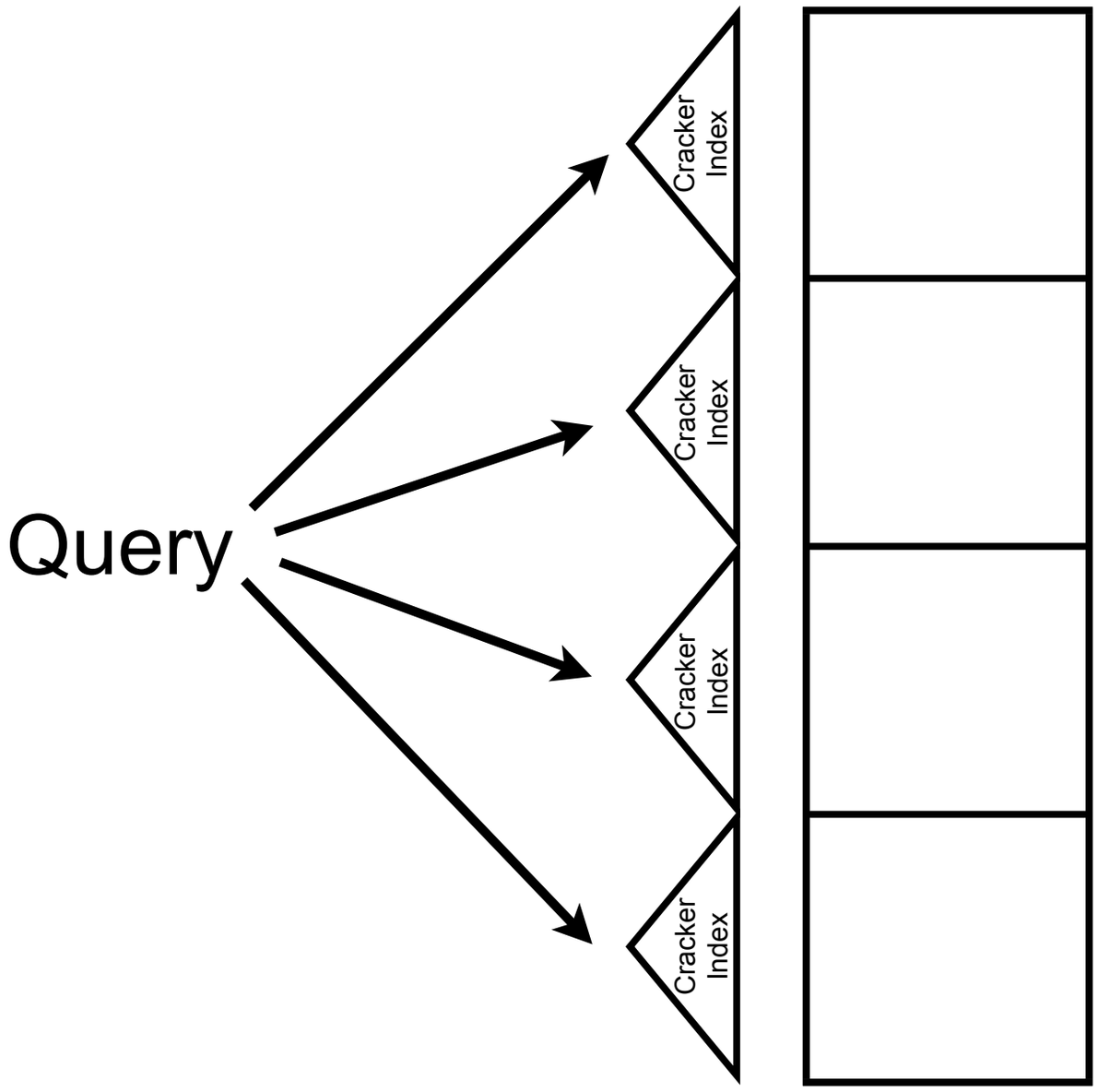
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)

Query

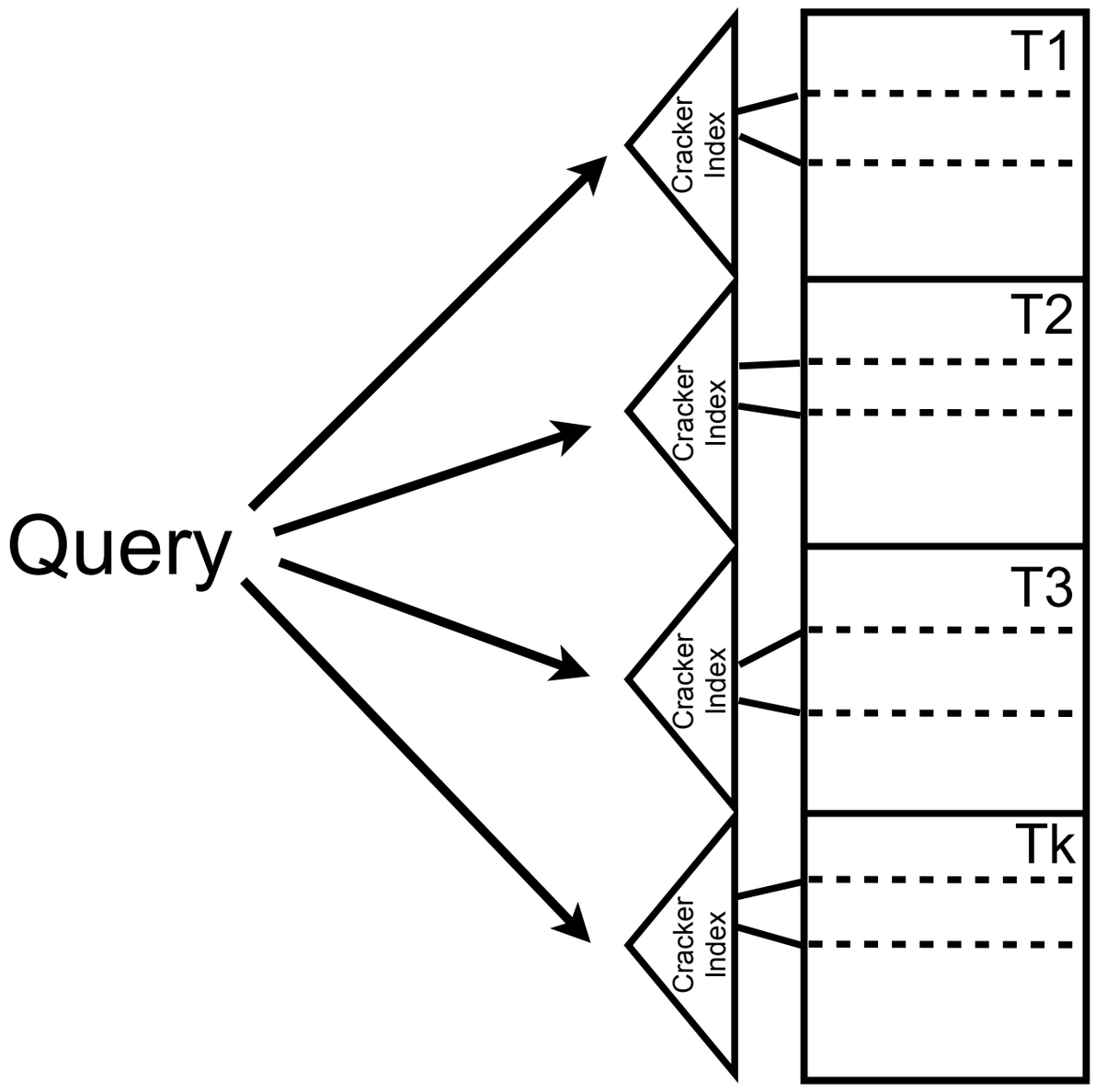


k Chunks

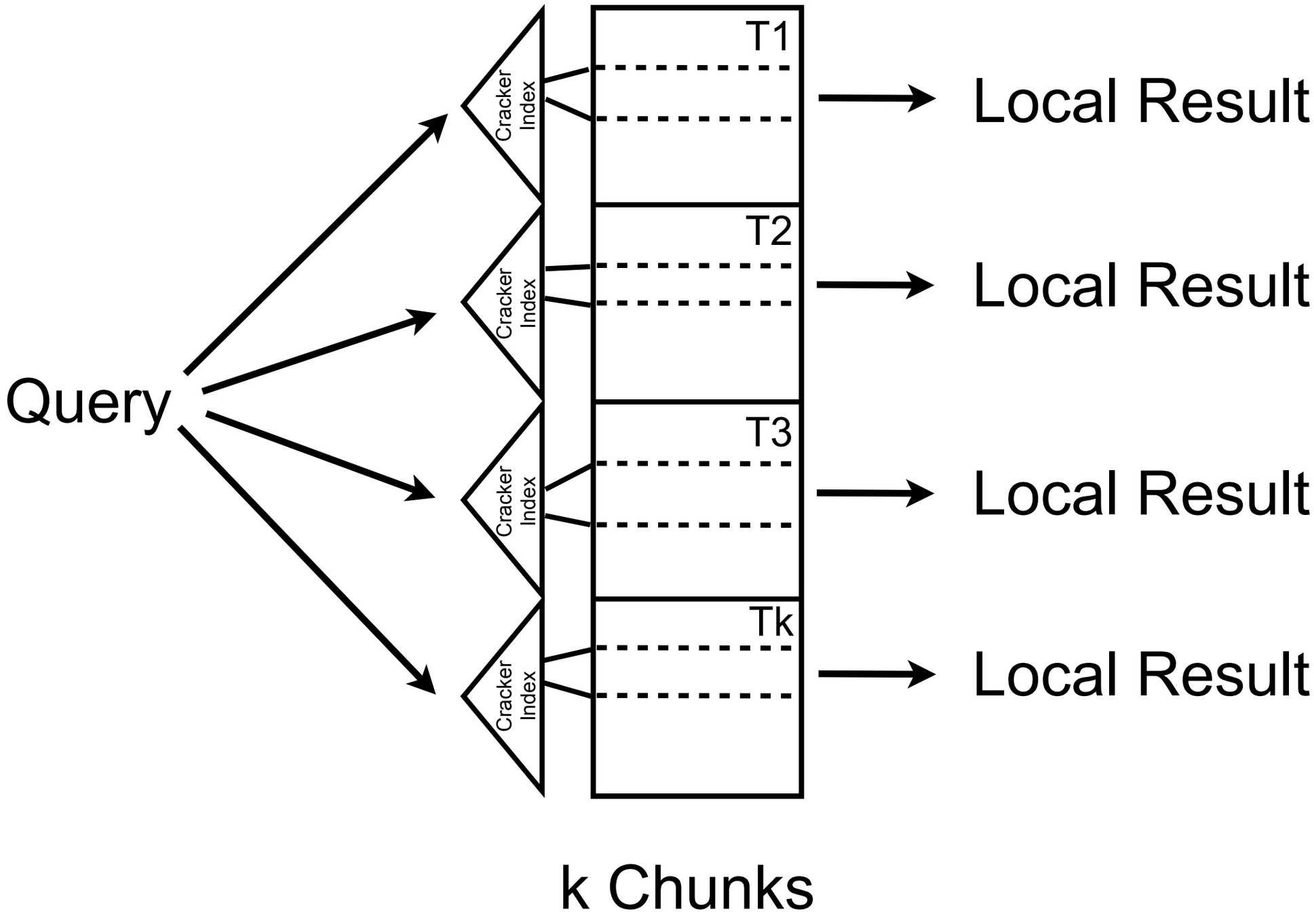
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



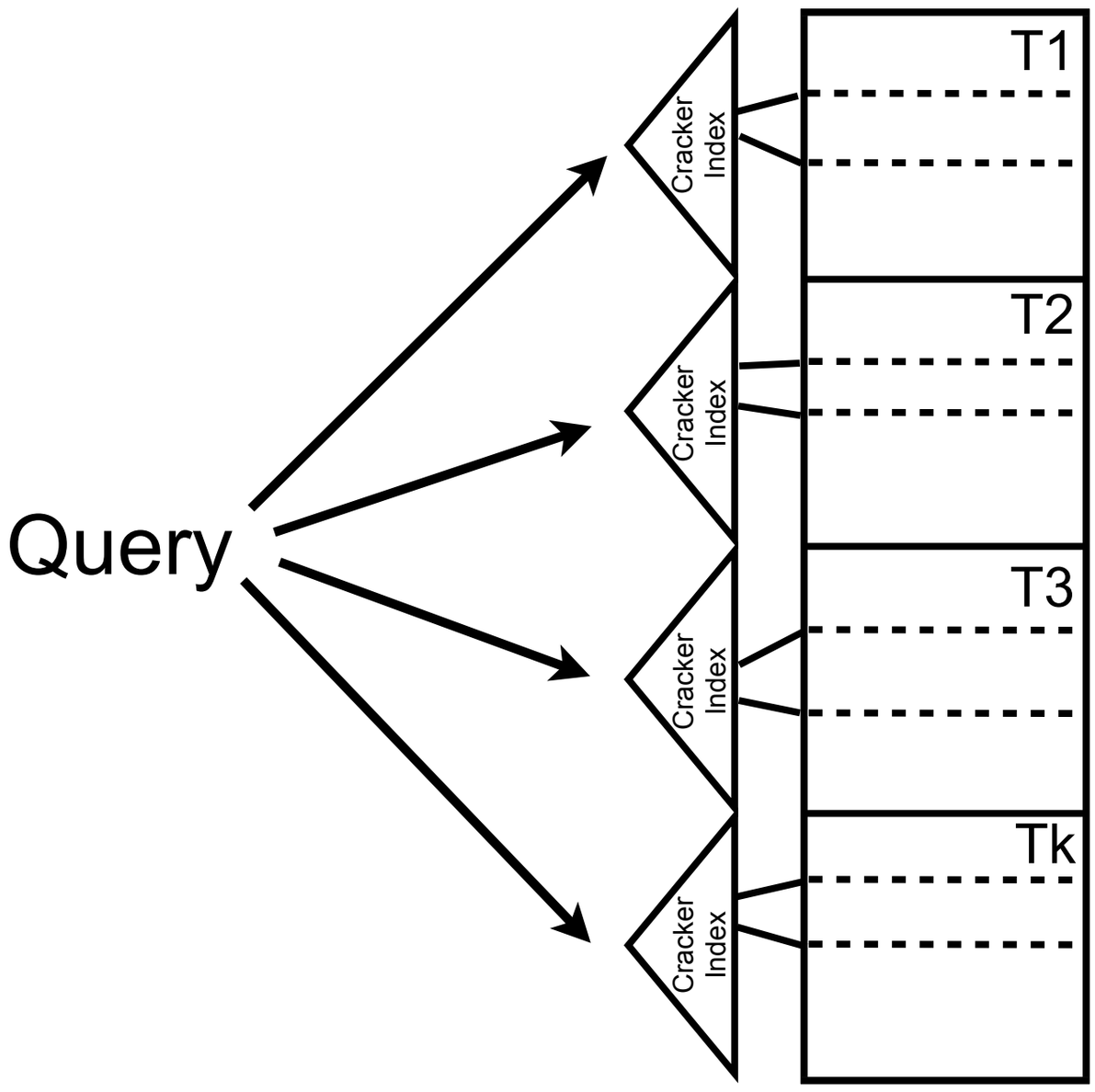
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



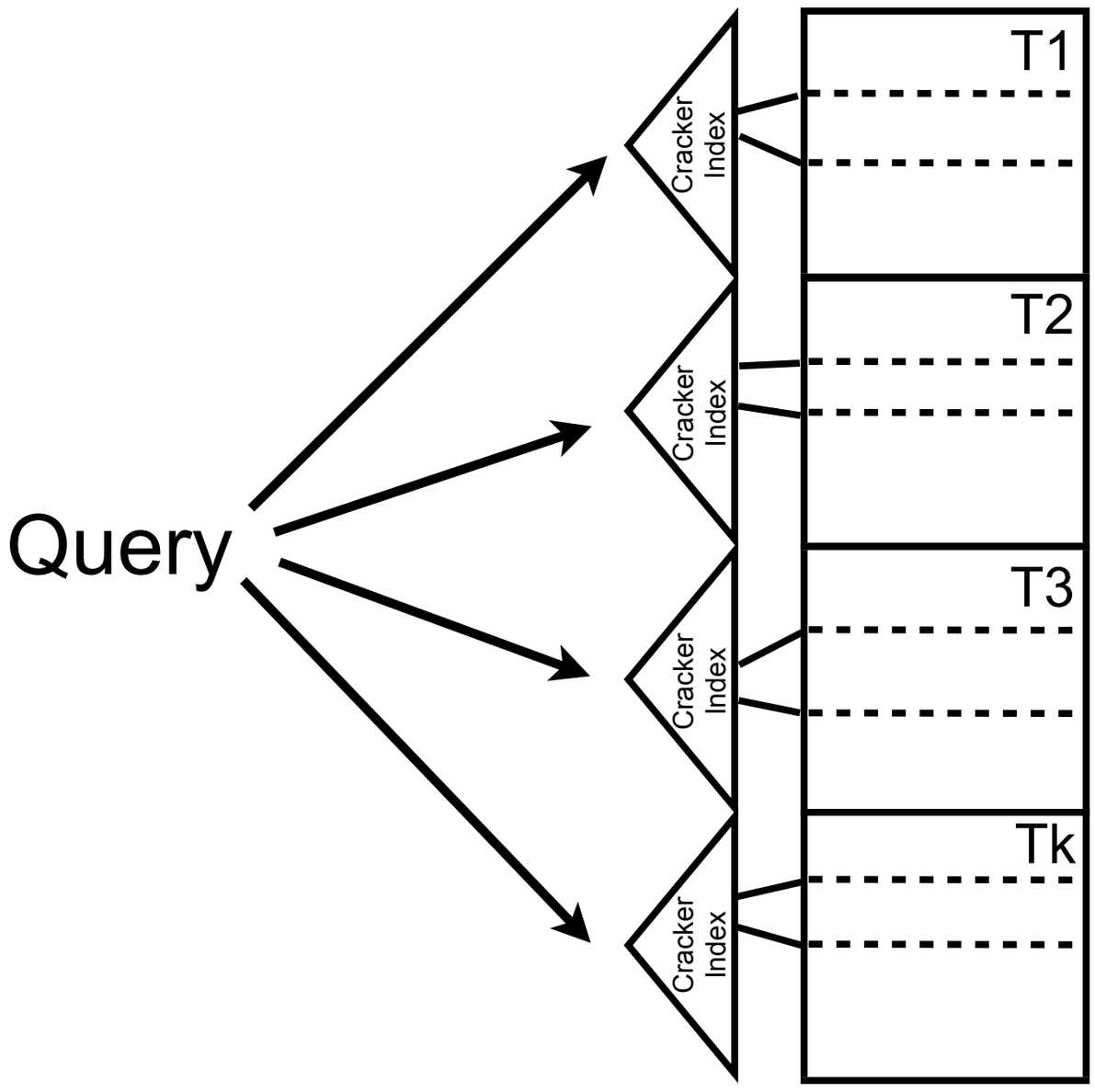
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



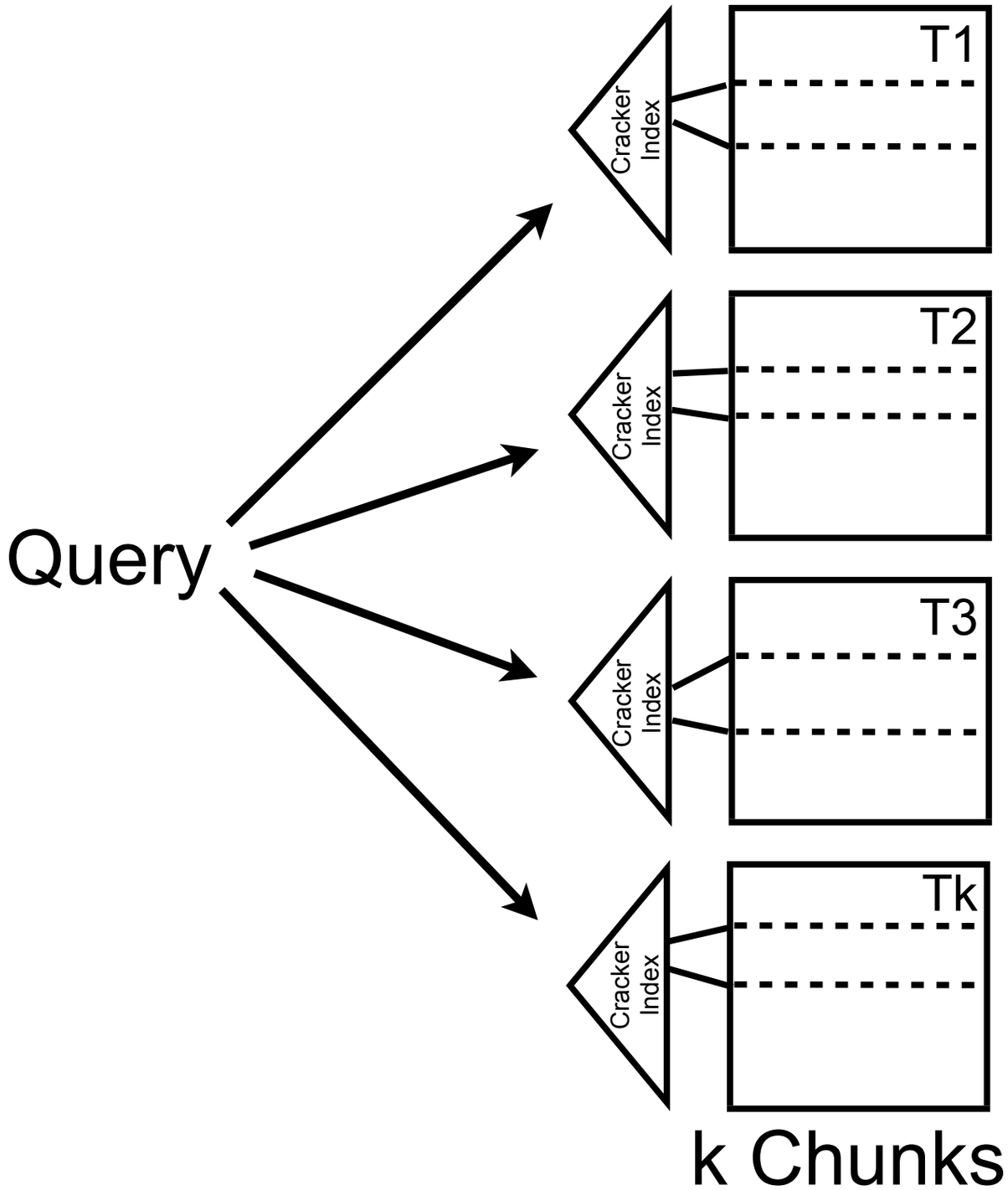
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



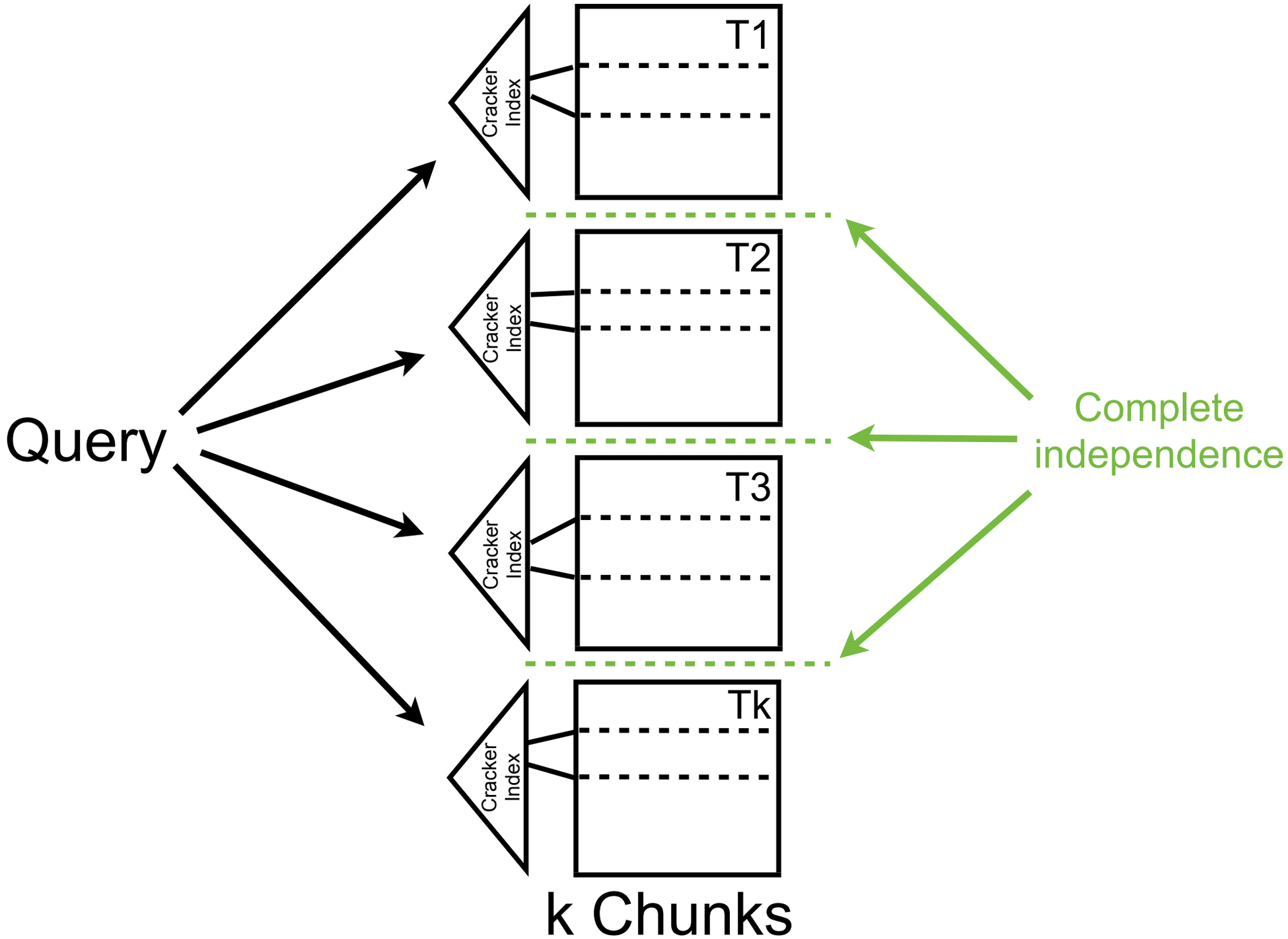
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



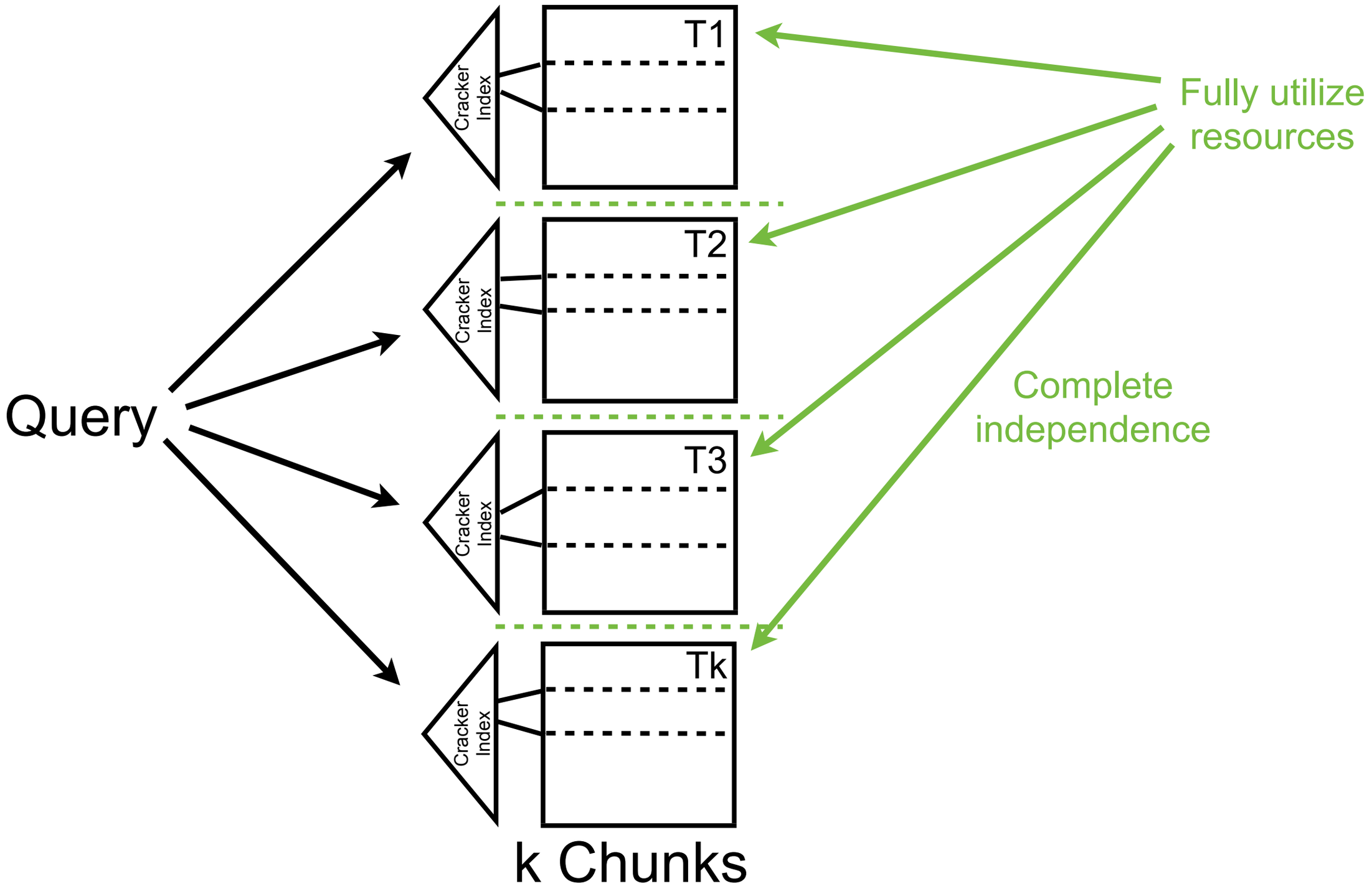
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



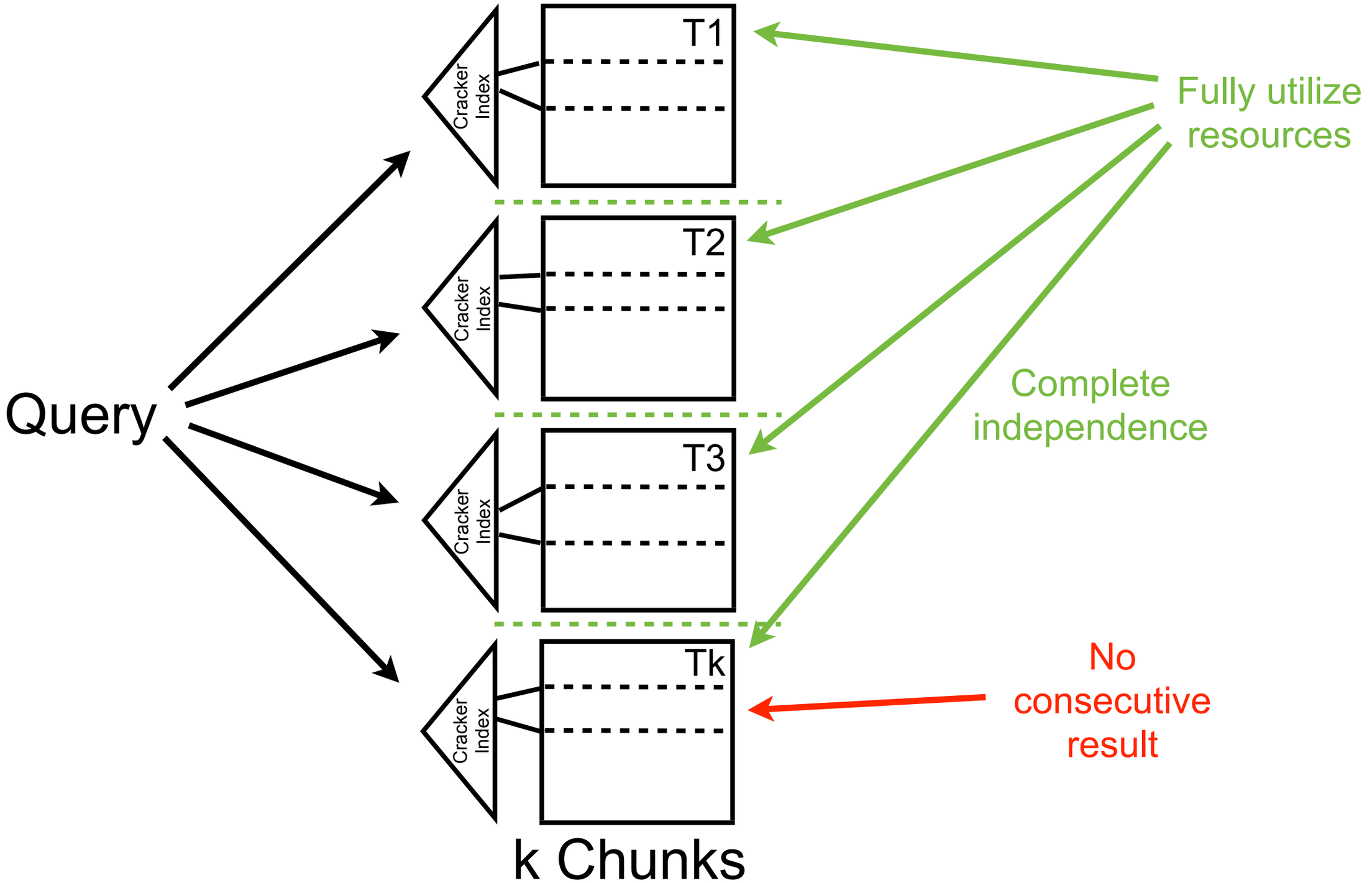
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



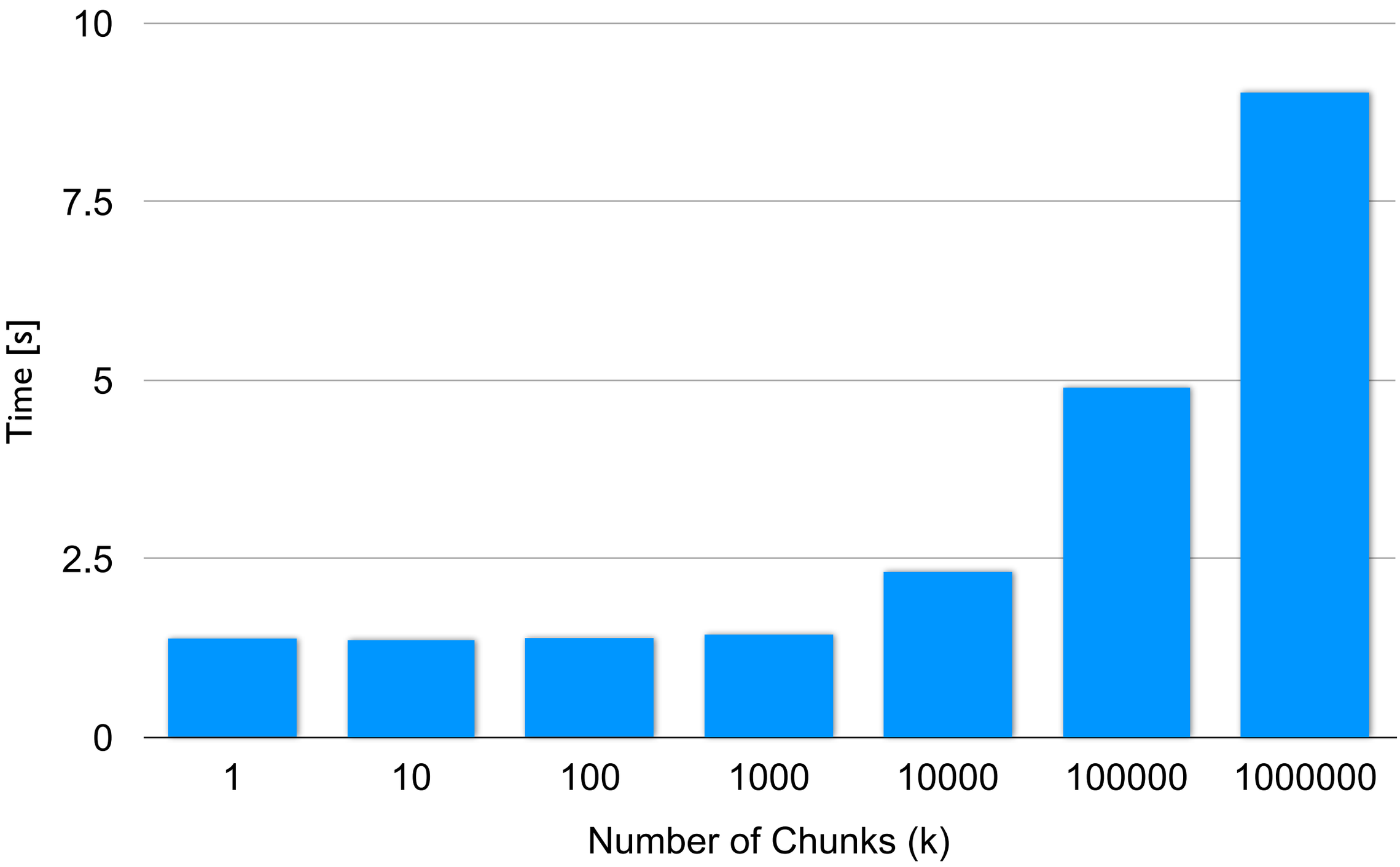
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



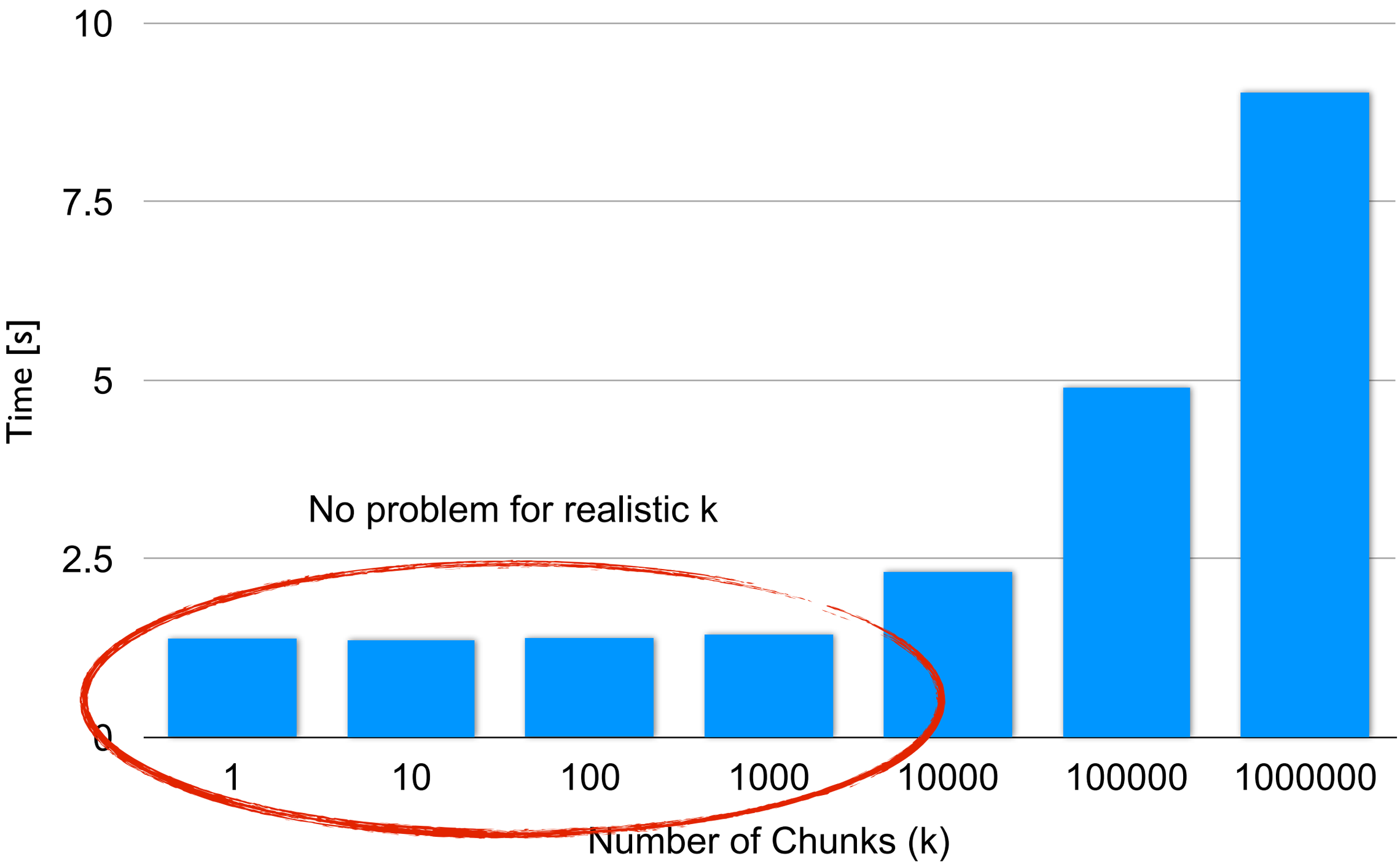
Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



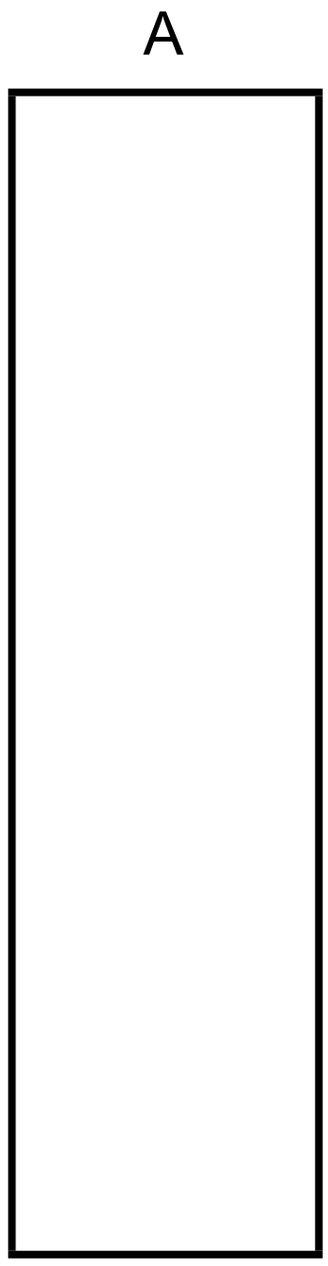
Micro Benchmark
Reading 1% from k locations using one thread



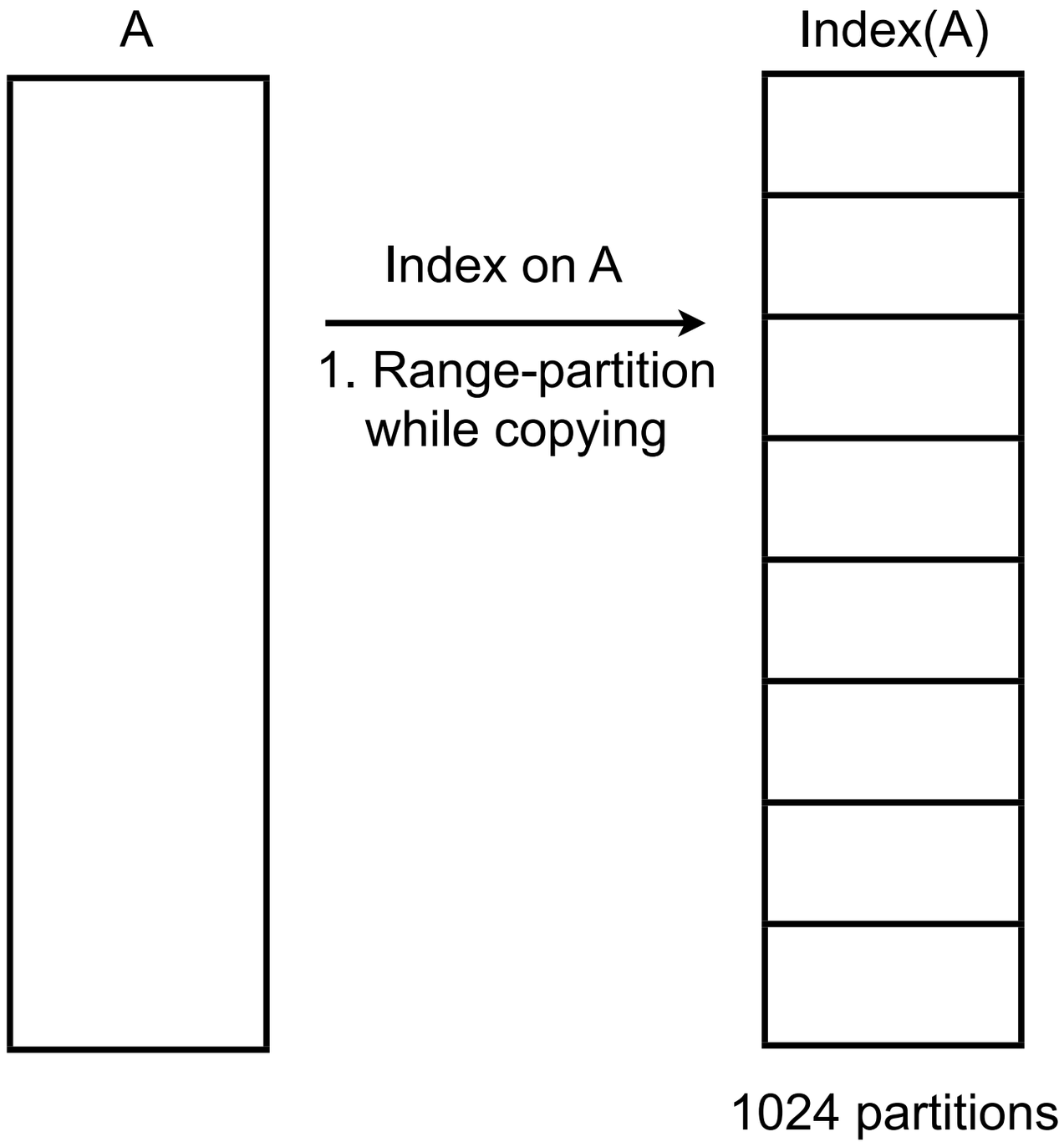
Micro Benchmark
Reading 1% from k locations using one thread



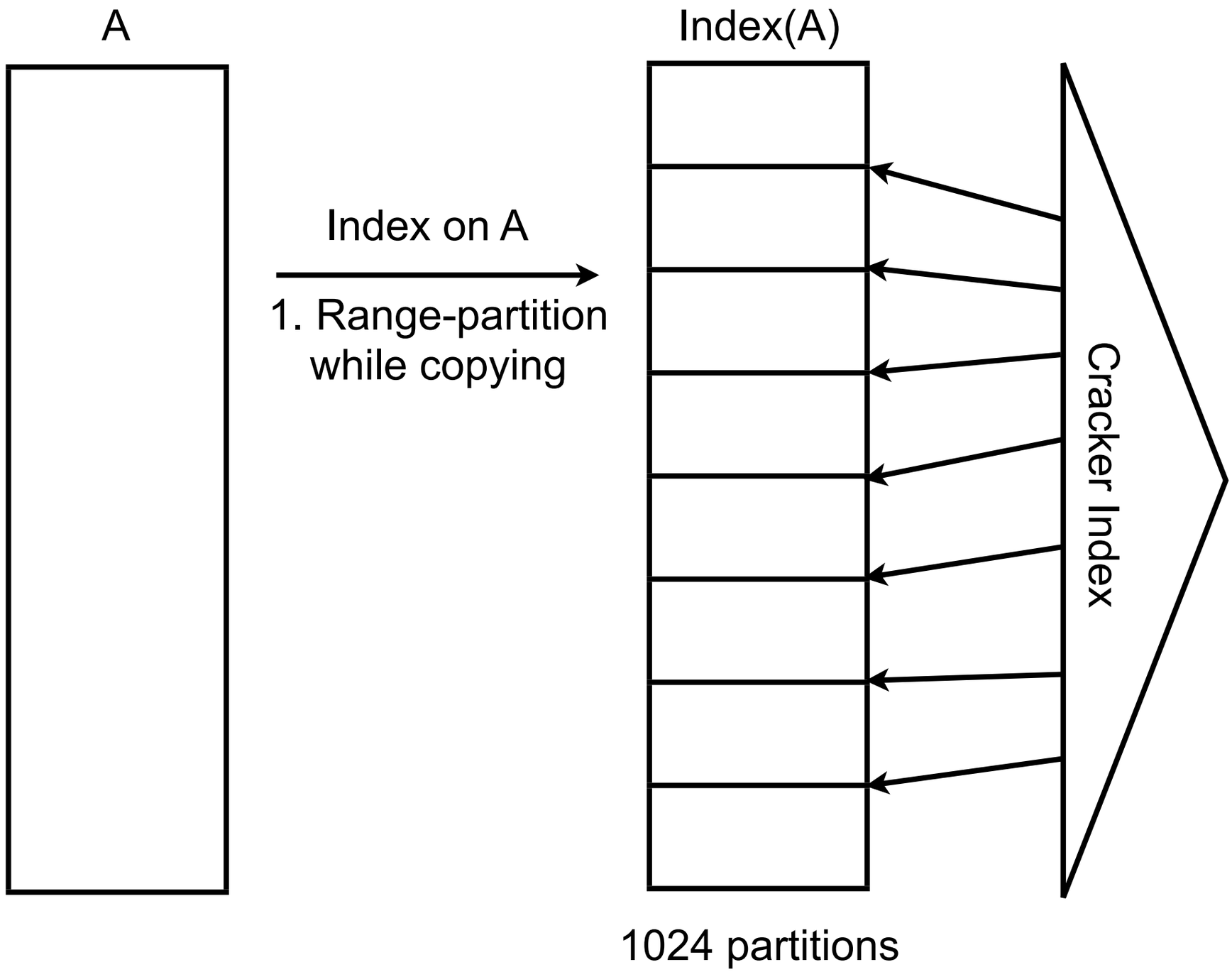
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



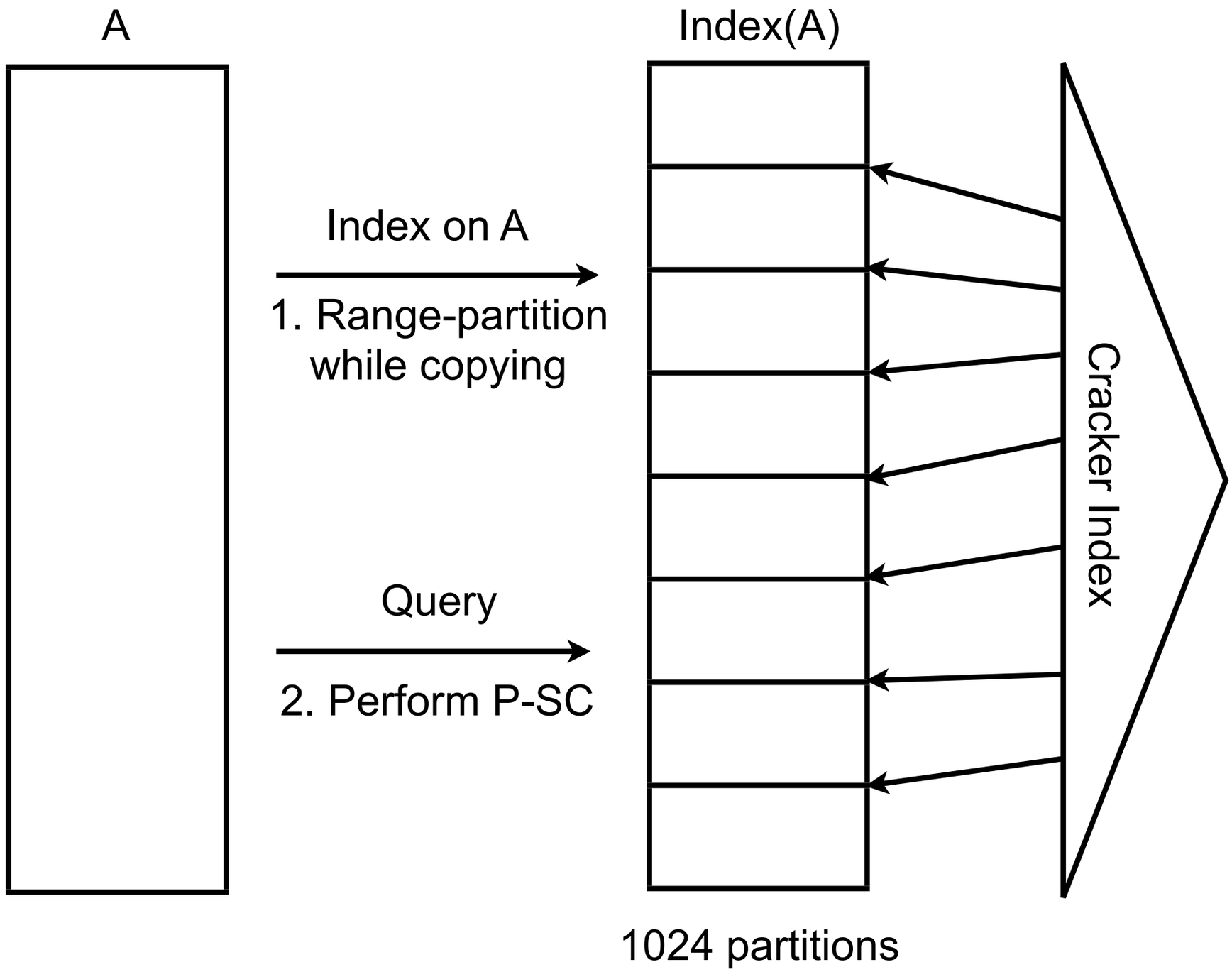
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



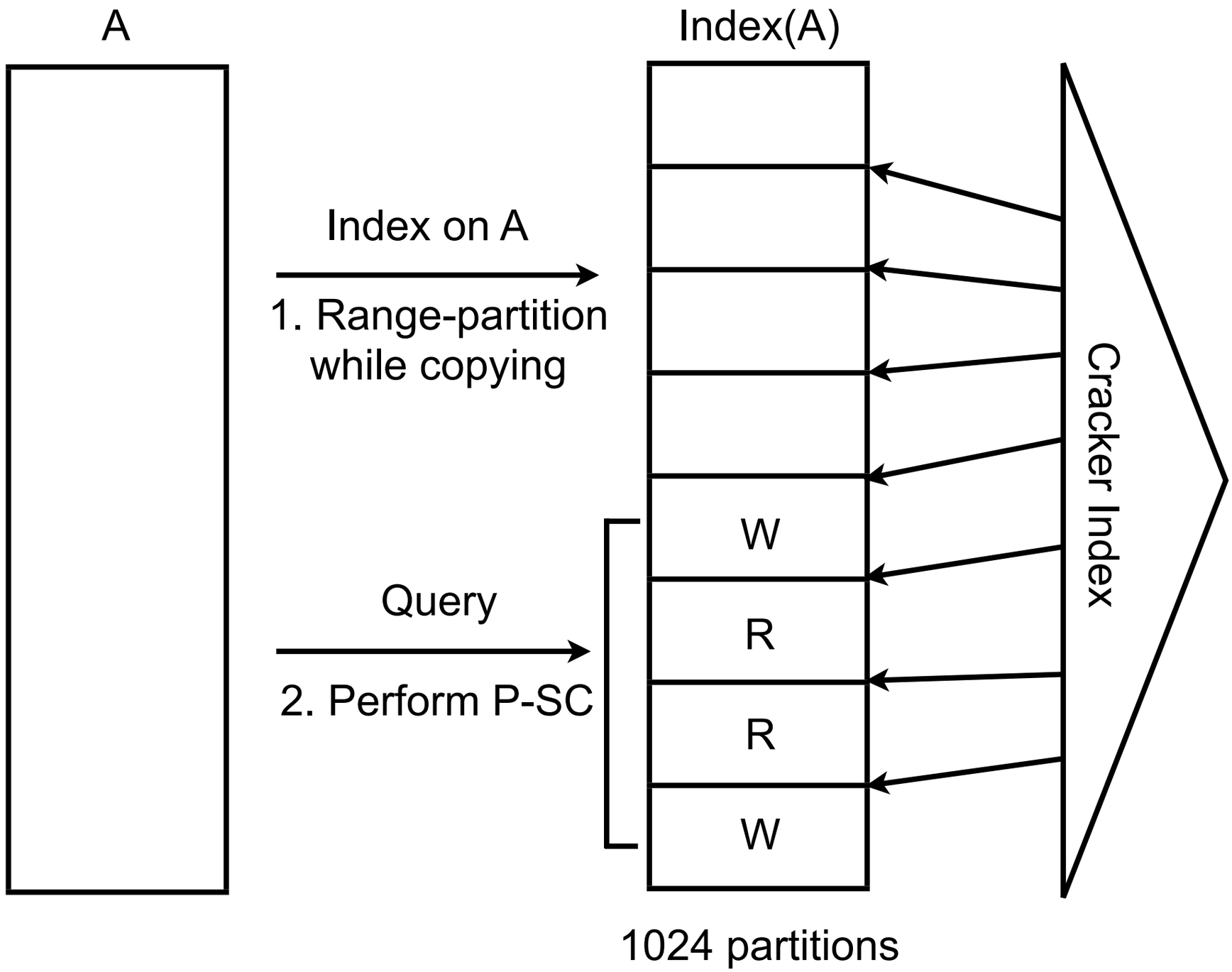
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



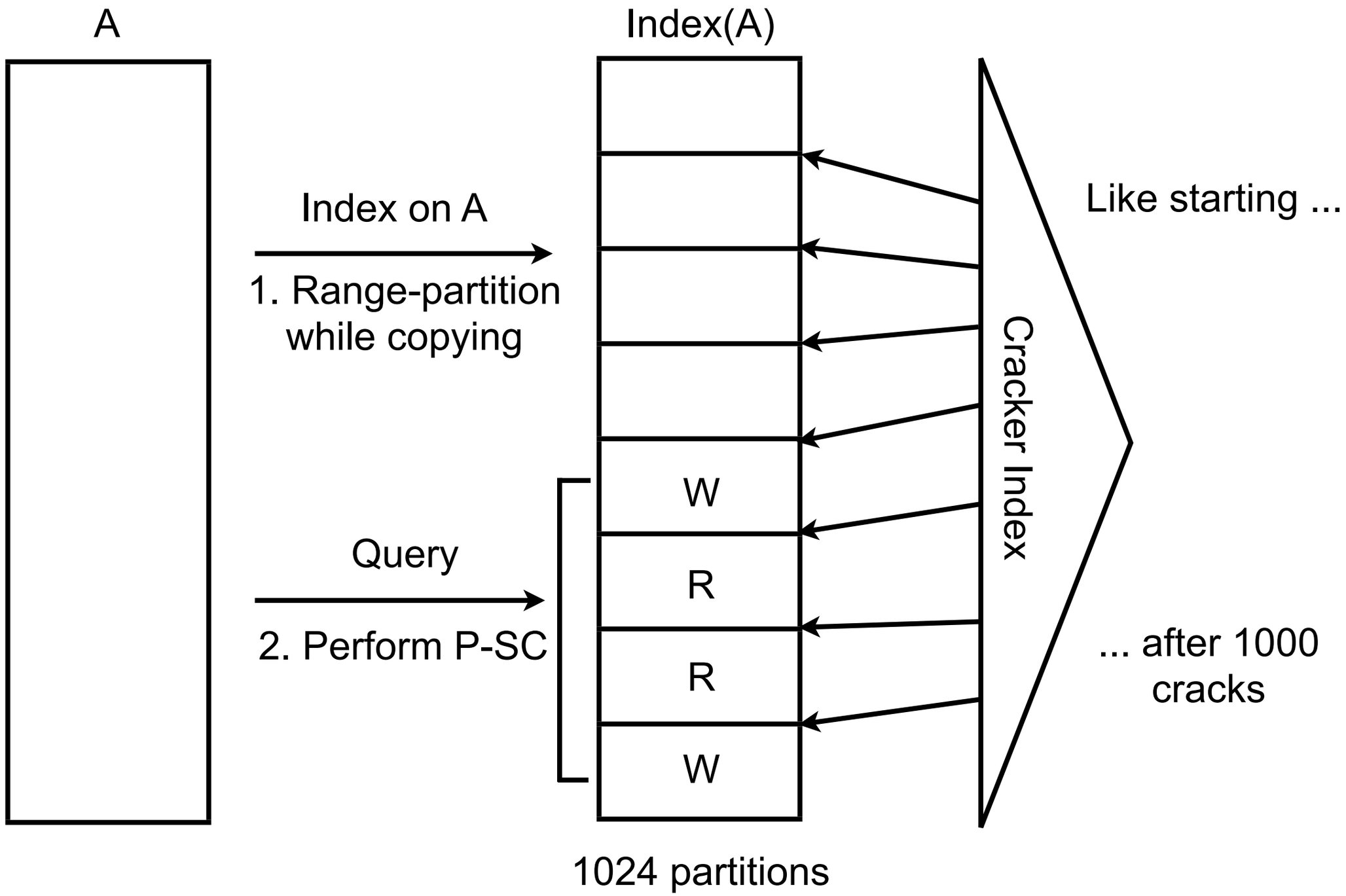
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



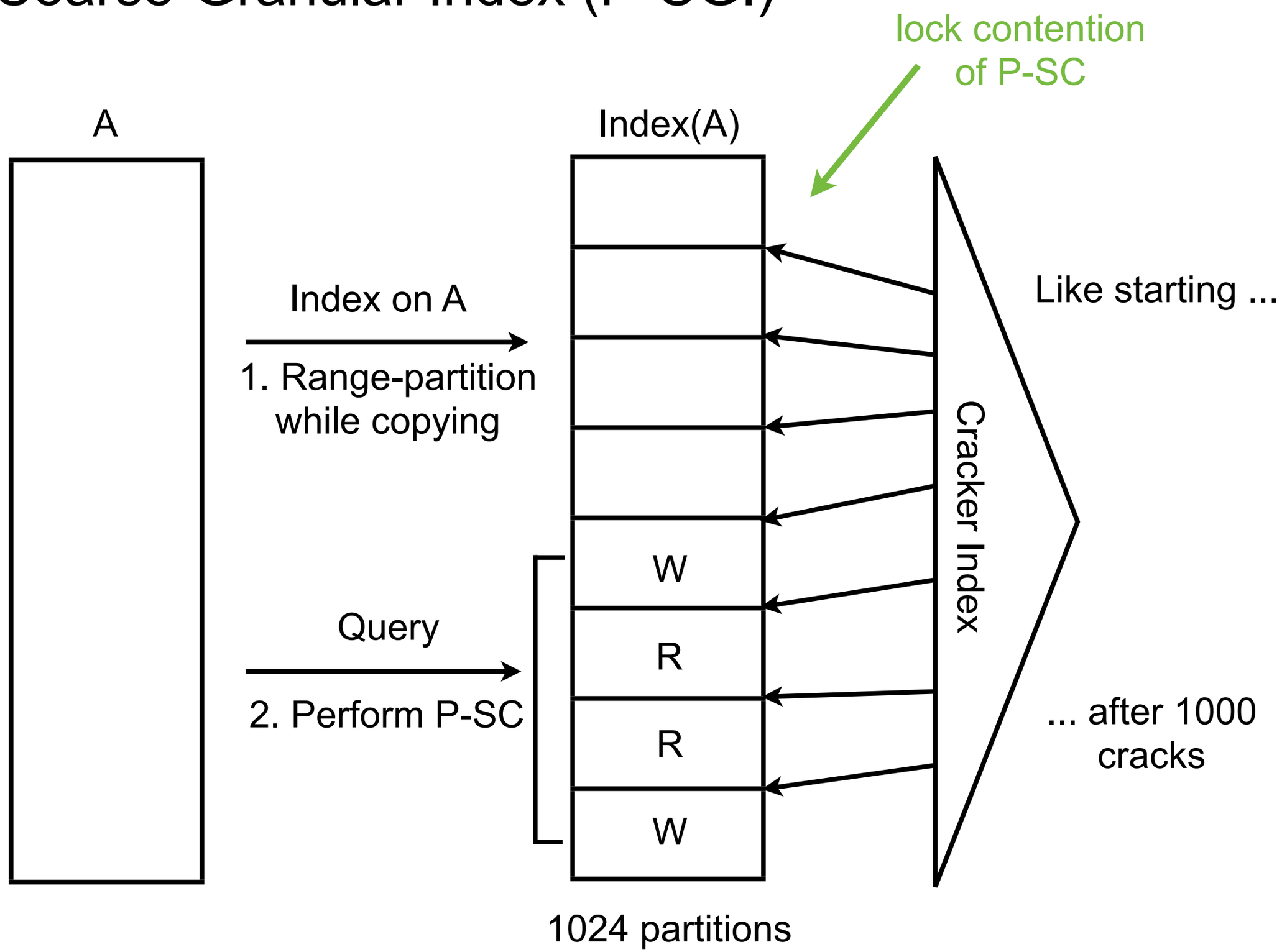
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



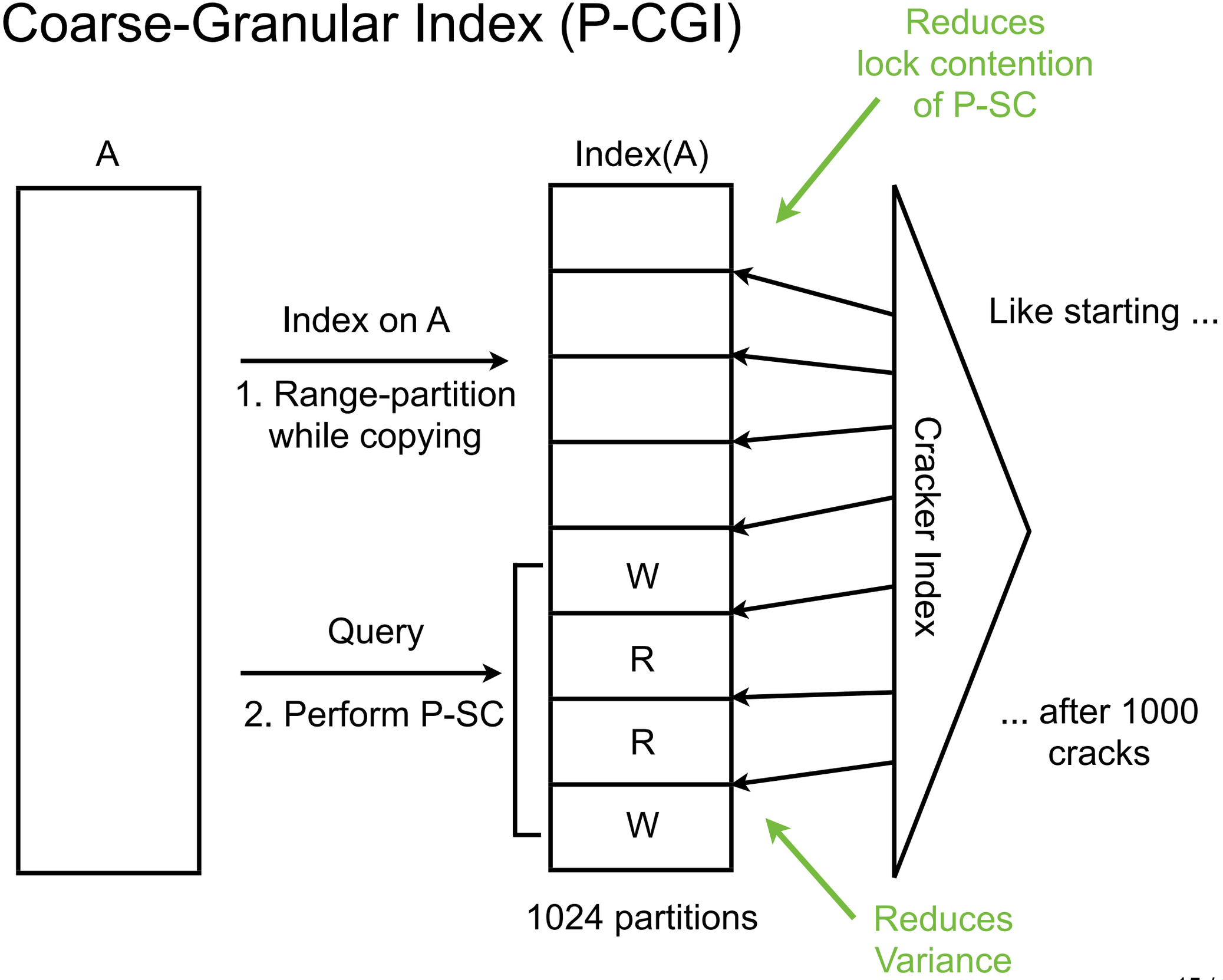
Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI)



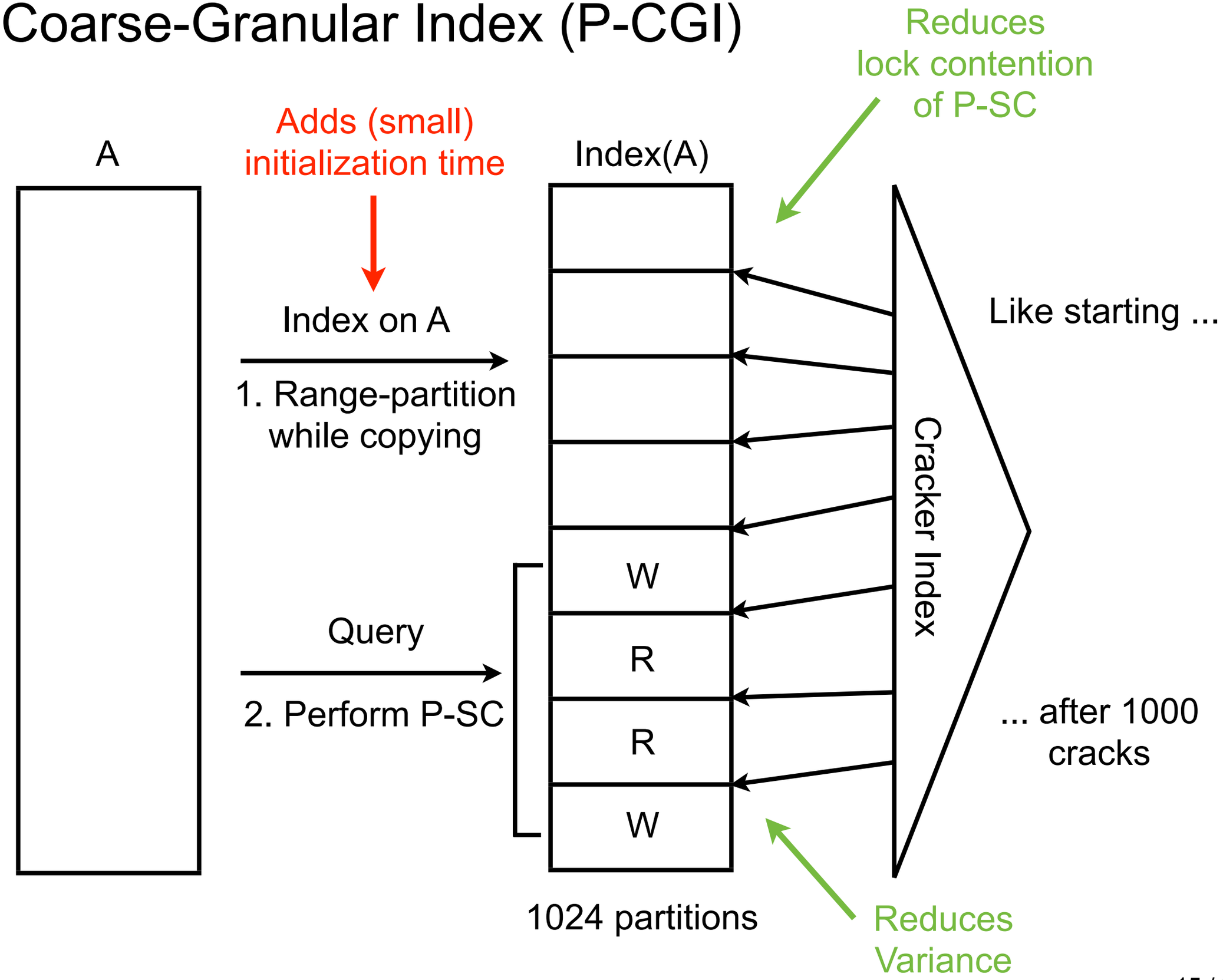
Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI)



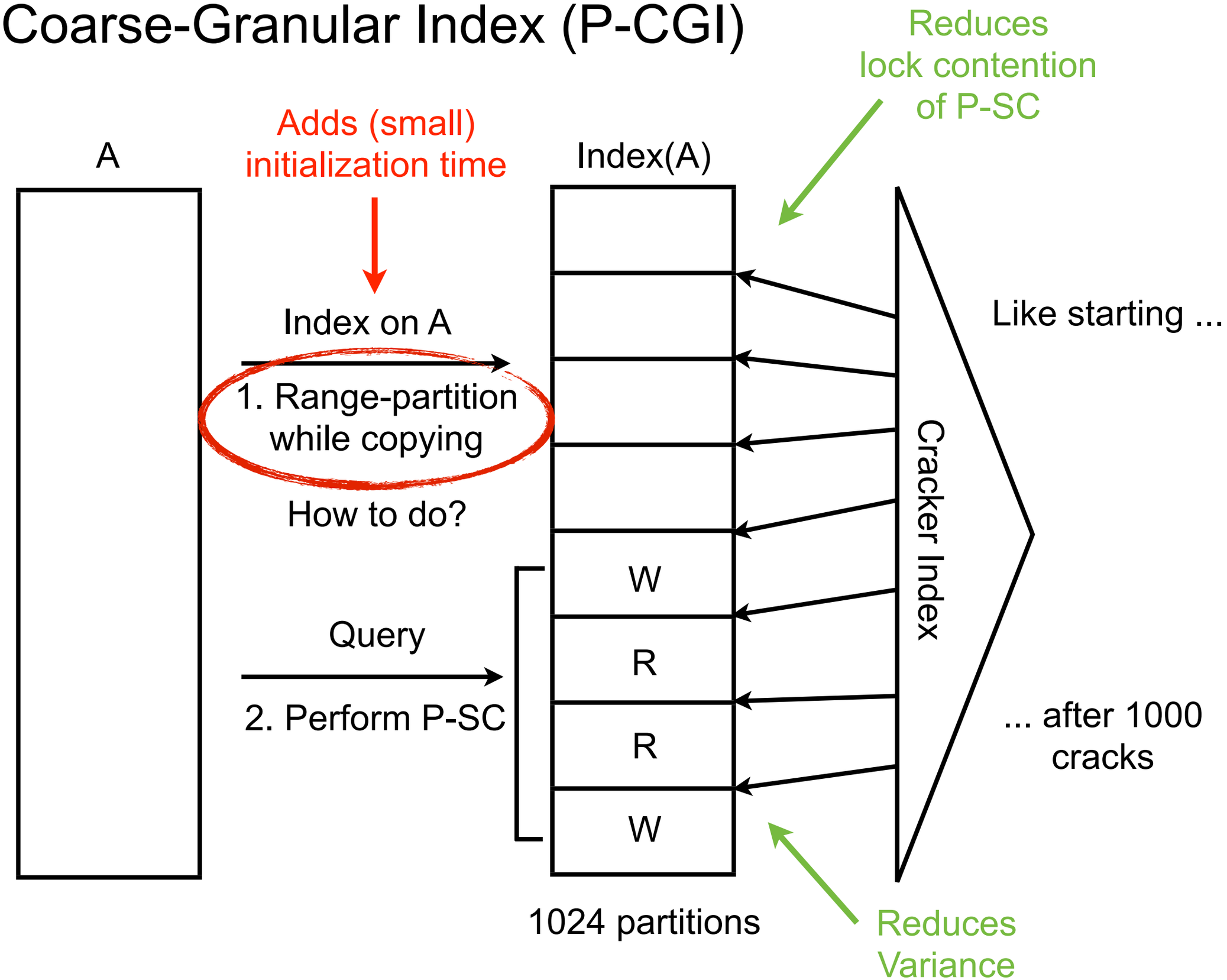
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



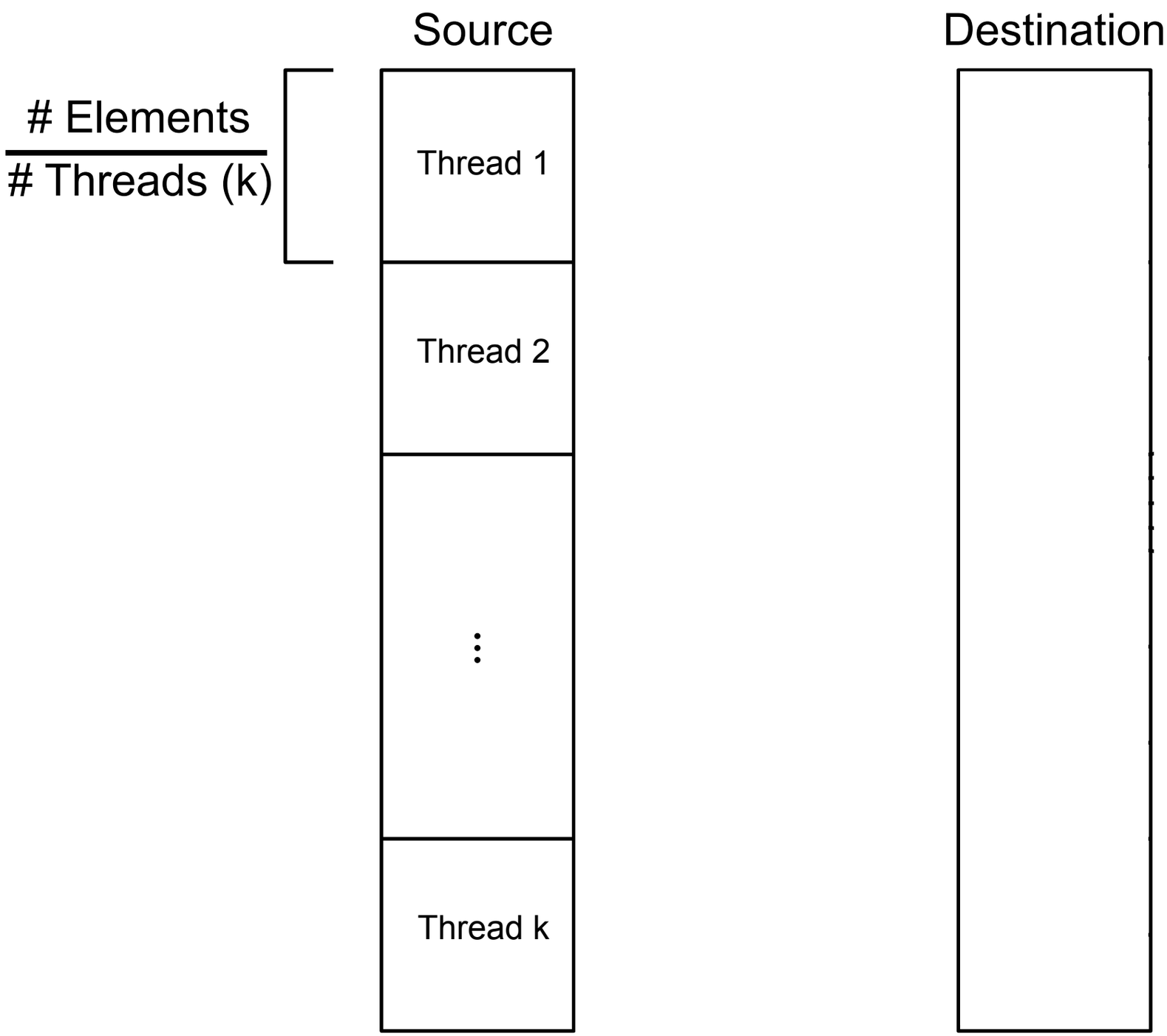
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)



Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI)

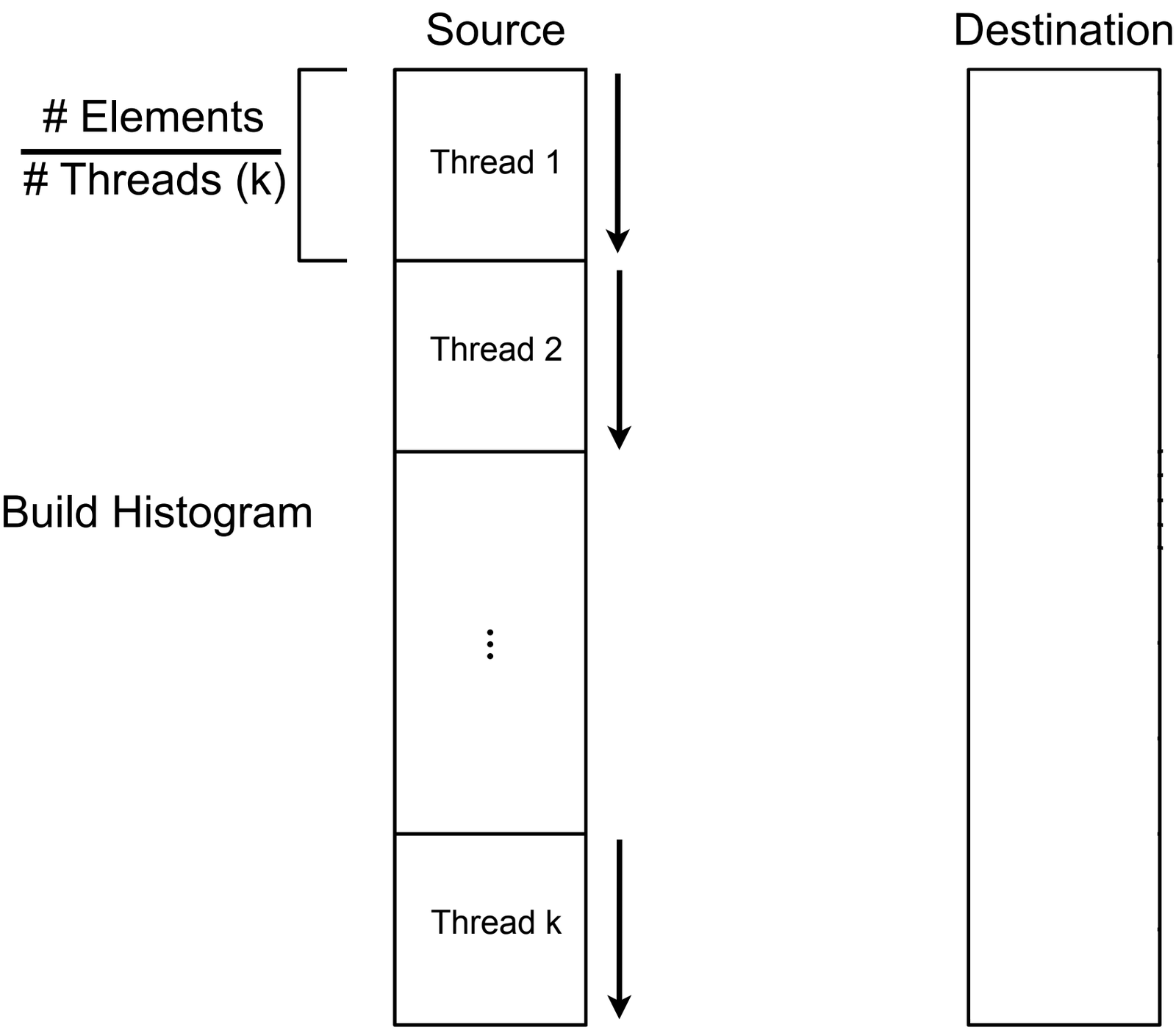


Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning



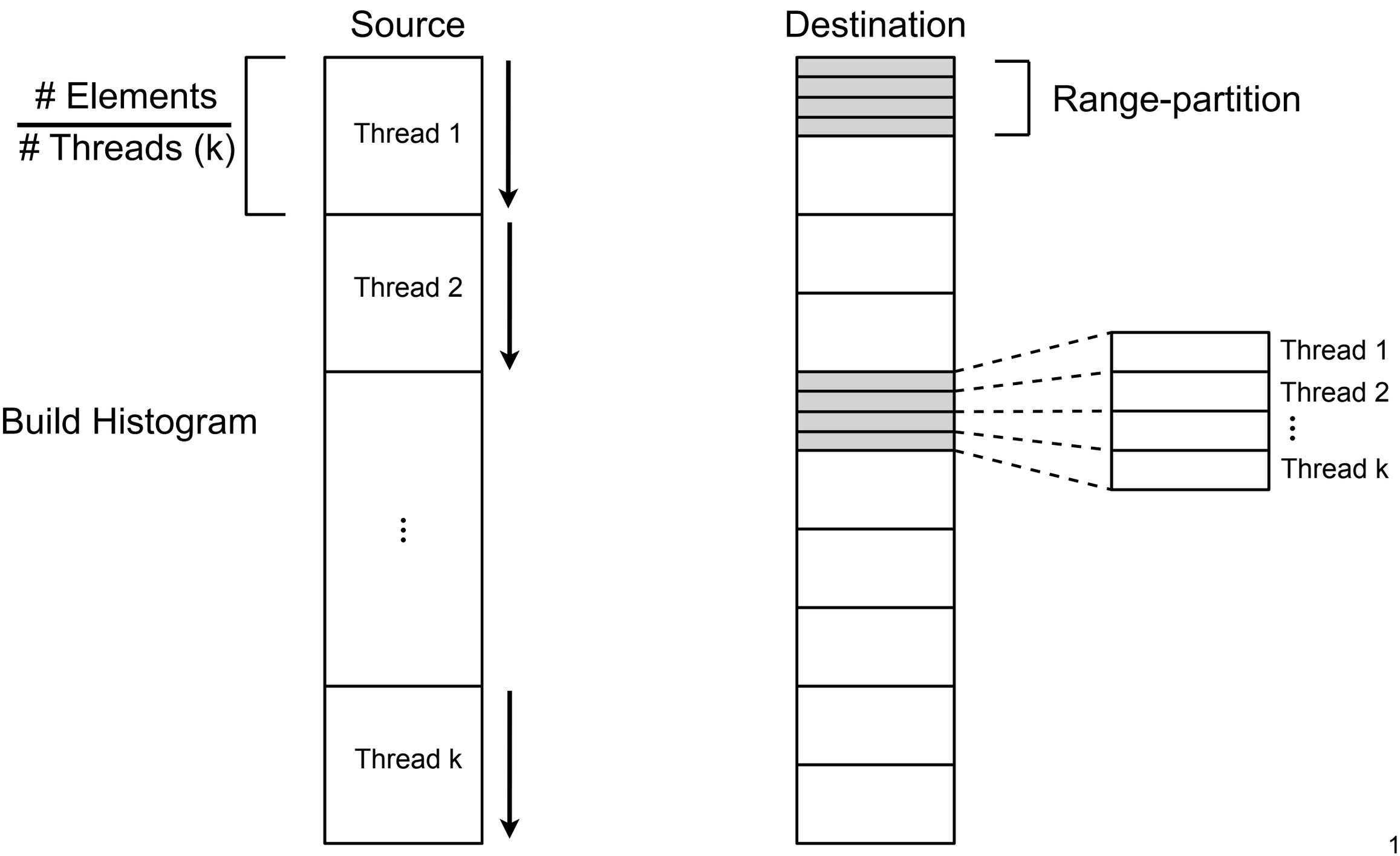
Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning

1. Build Histogram



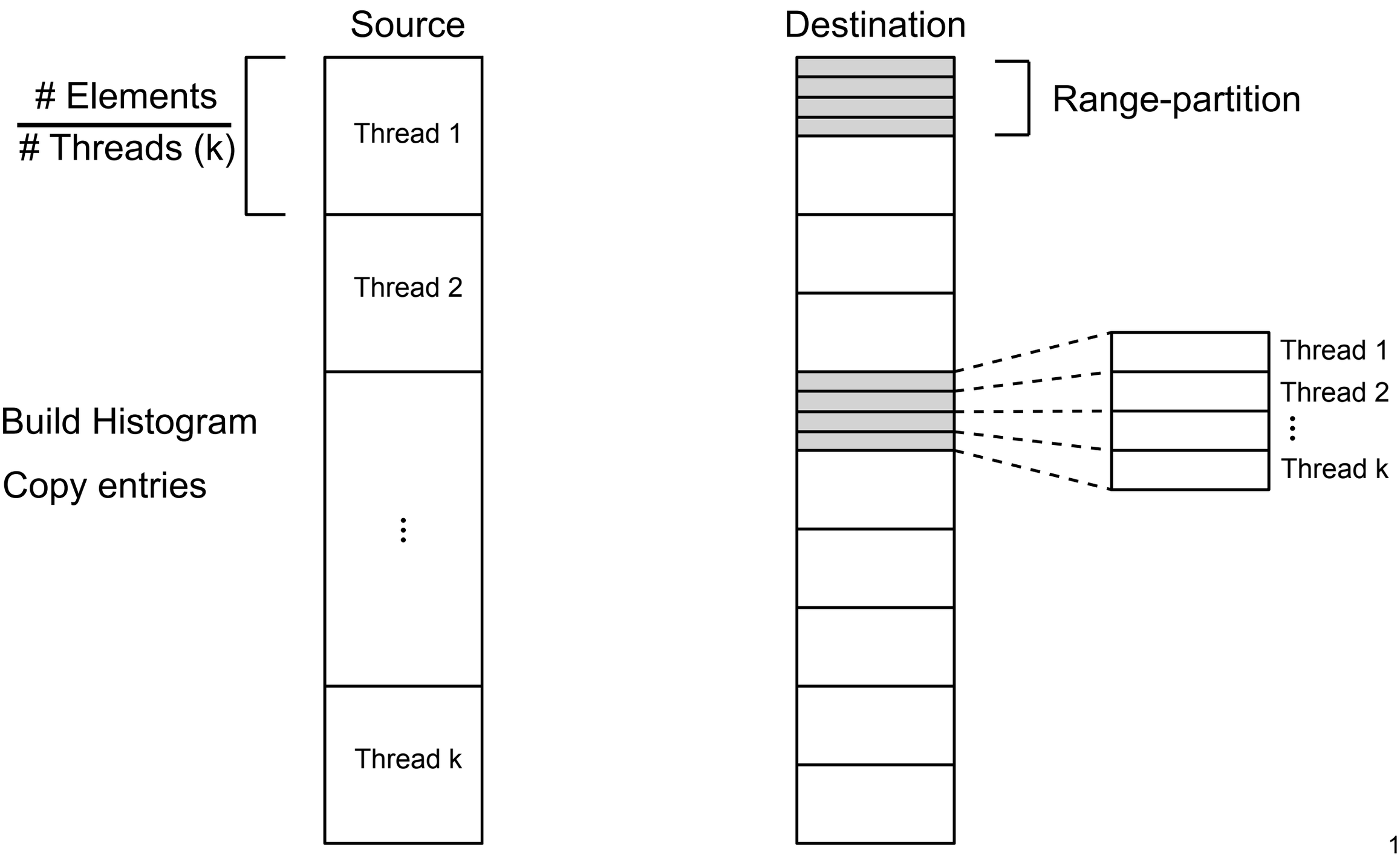
Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning

1. Build Histogram



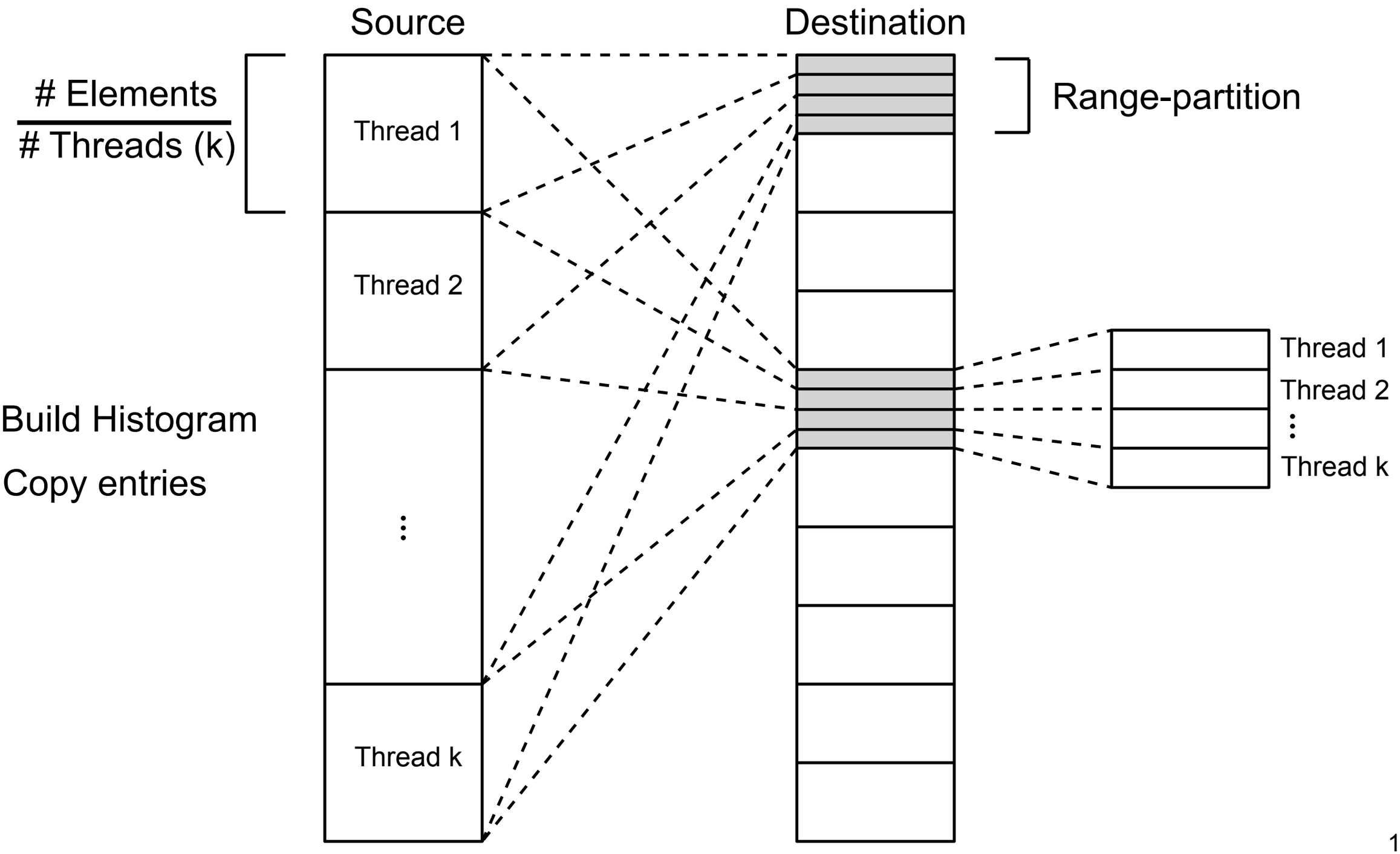
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI):
Parallel Range Partitioning

1. Build Histogram
2. Copy entries



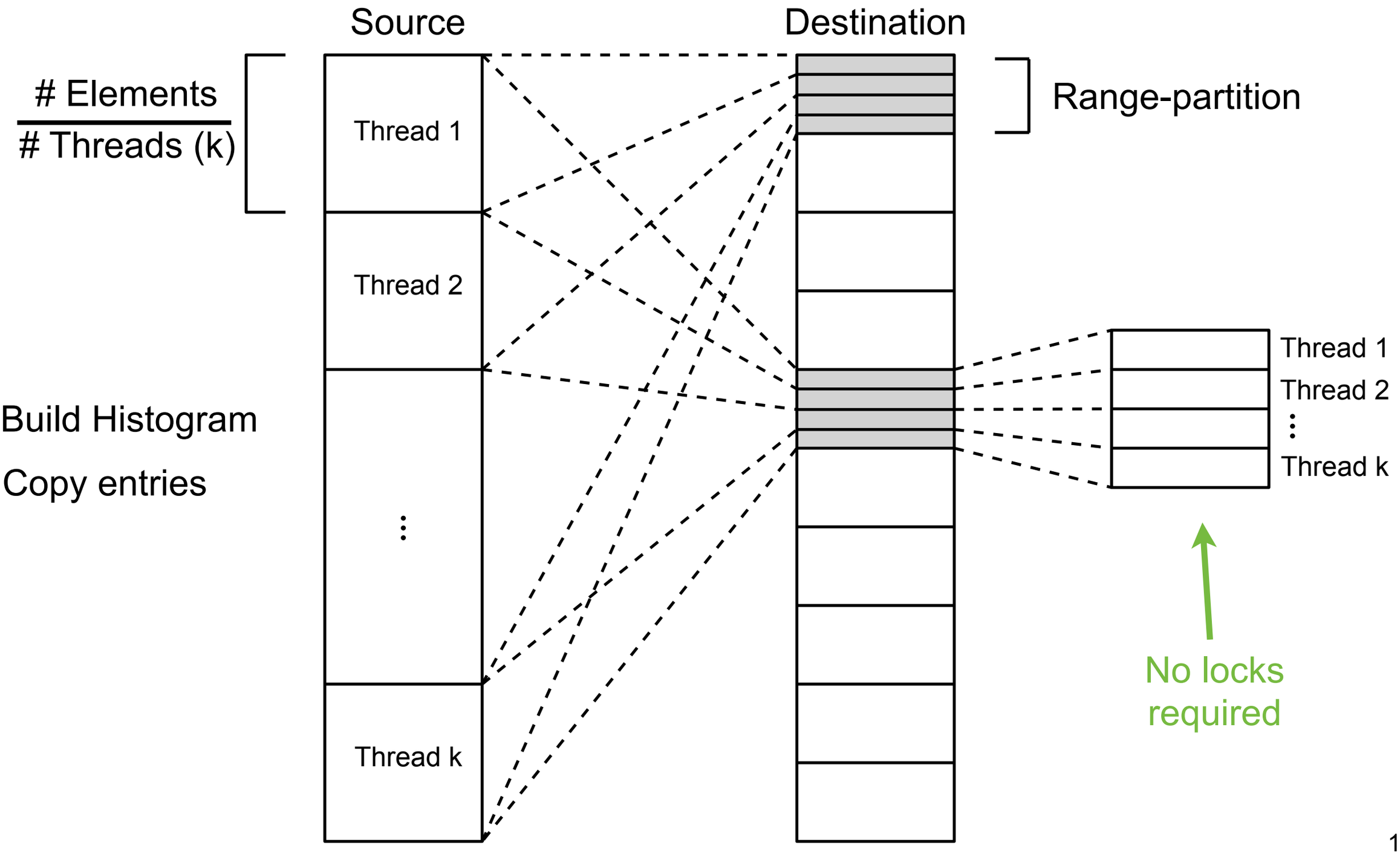
Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning

1. Build Histogram
2. Copy entries

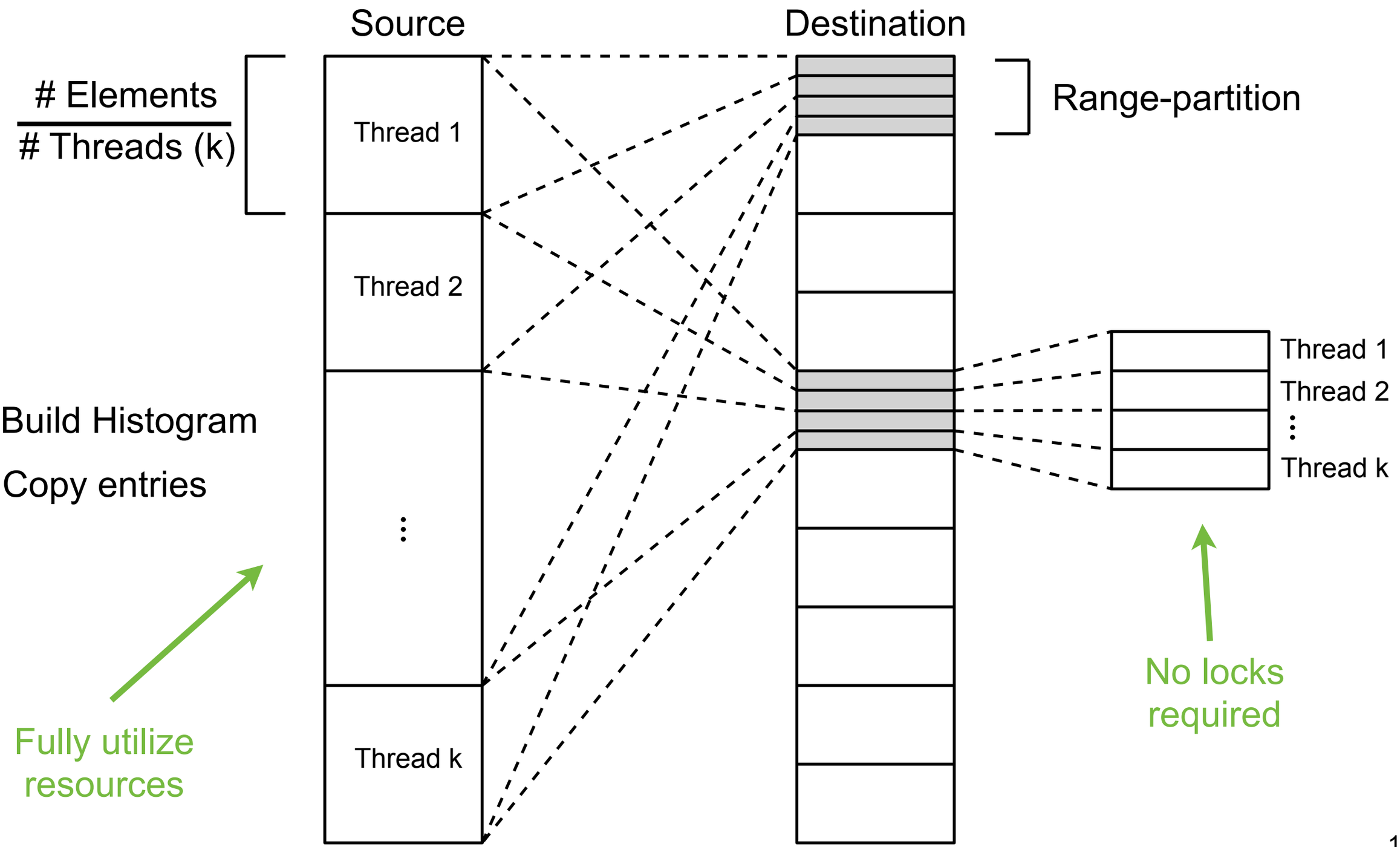


Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning

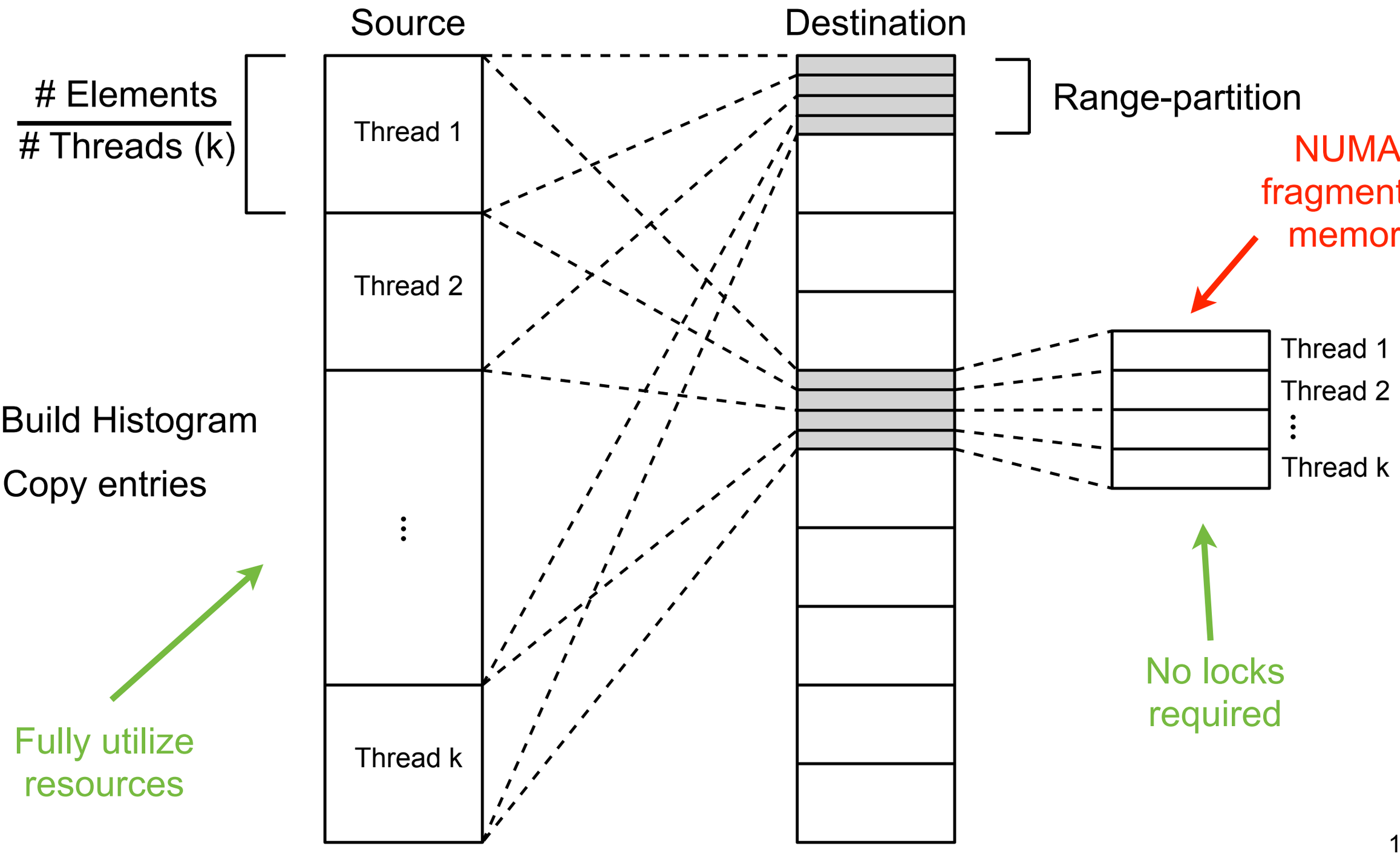
1. Build Histogram
2. Copy entries



Multi-threaded algorithms: Parallel Coarse-Granular Index (P-CGI): Parallel Range Partitioning



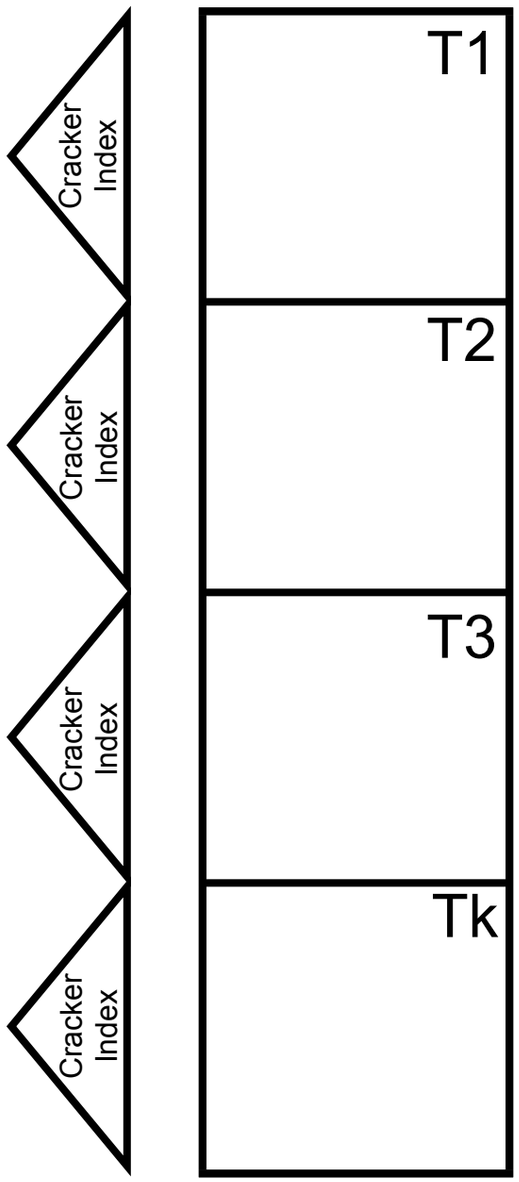
Multi-threaded algorithms:
Parallel Coarse-Granular Index (P-CGI):
Parallel Range Partitioning



1. Build Histogram
2. Copy entries

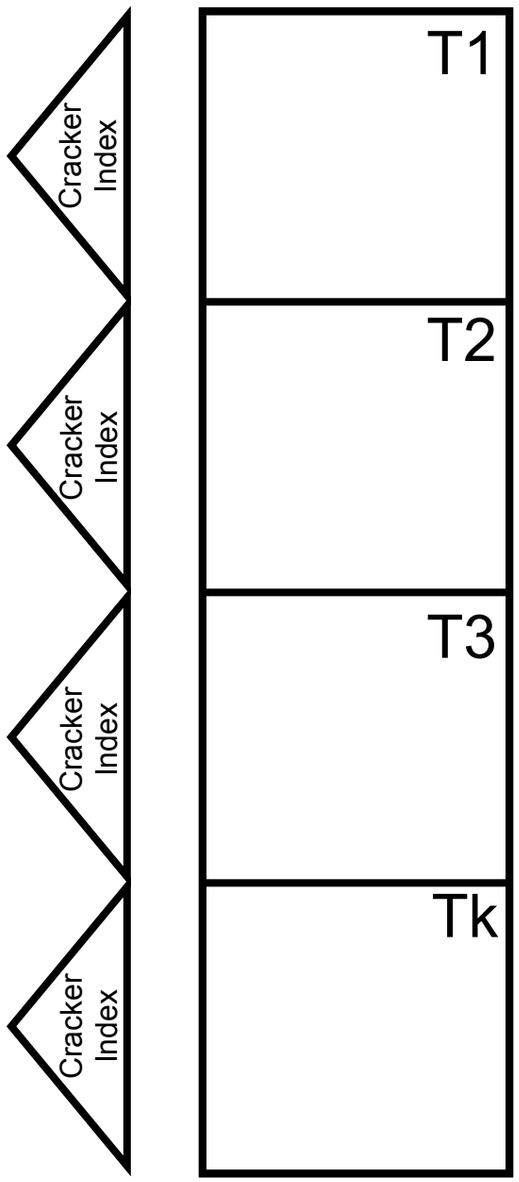
Fully utilize resources

Multi-threaded algorithms:
Parallel-chunked Standard Cracking (P-CSC)



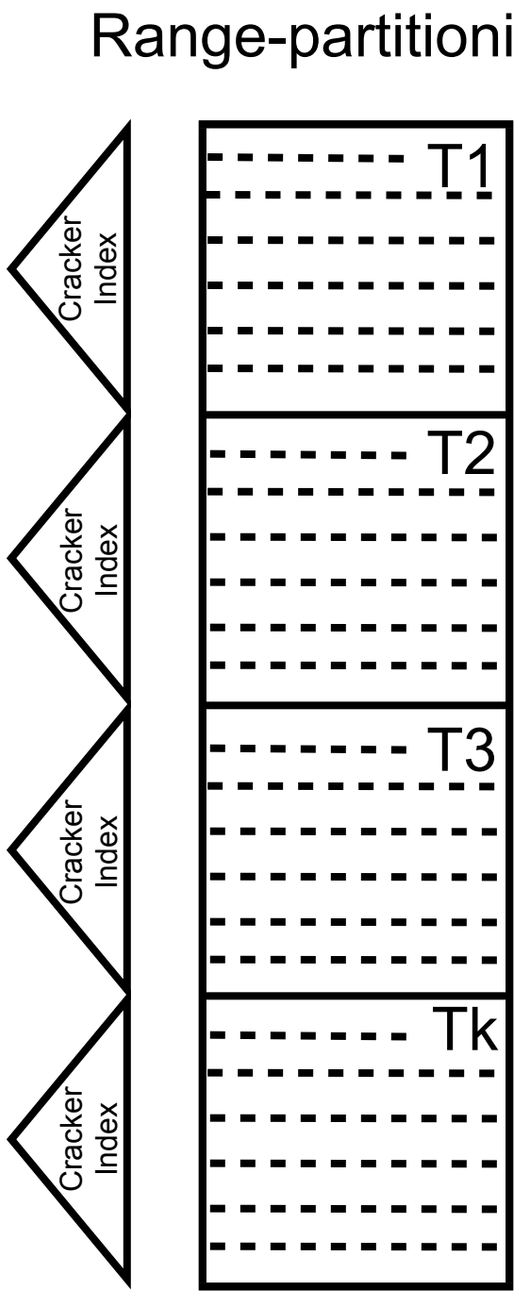
k Chunks

Multi-threaded algorithms:
Parallel-chunked Coarse-Granular Index (P-CCGI)

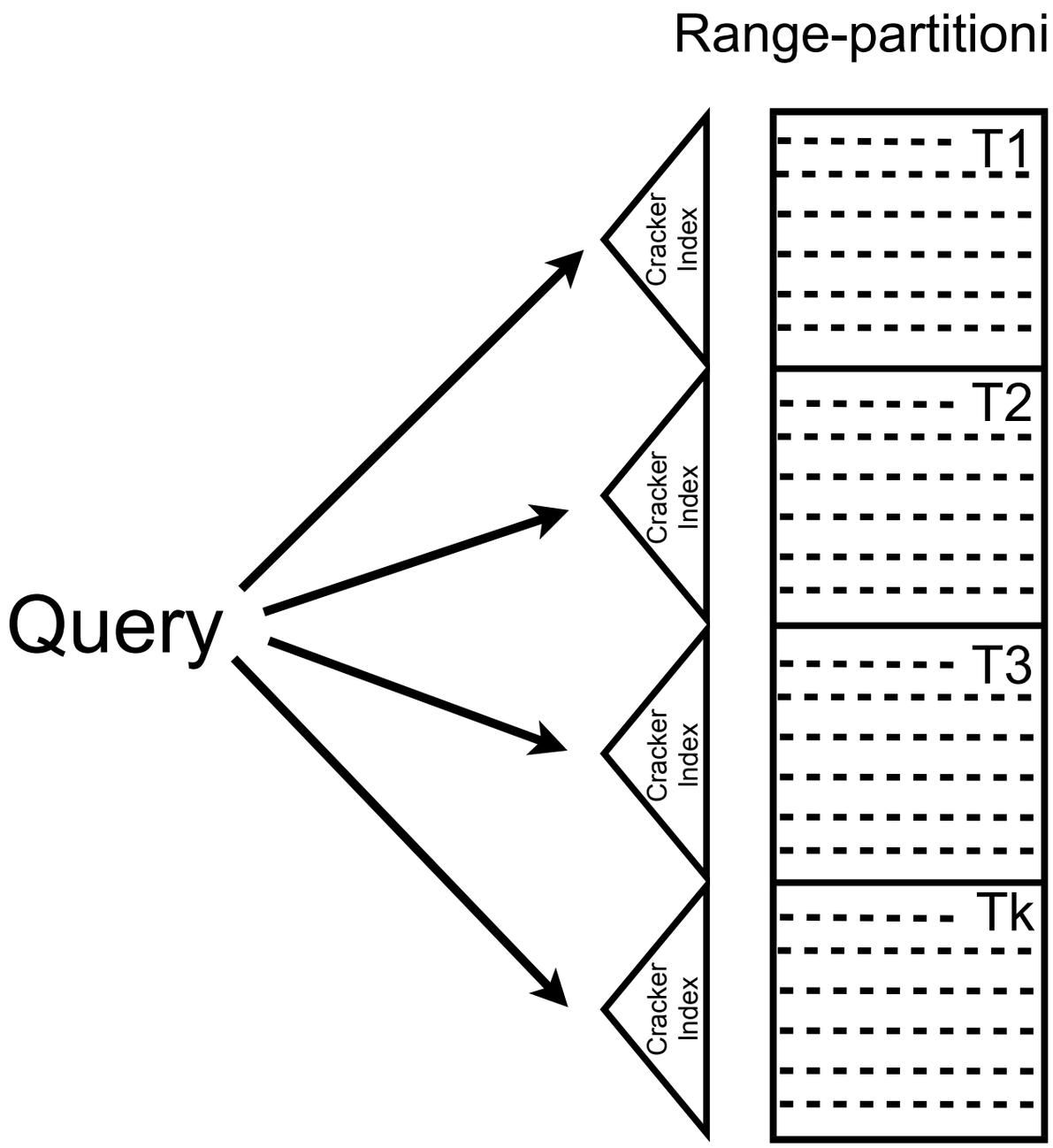


k Chunks

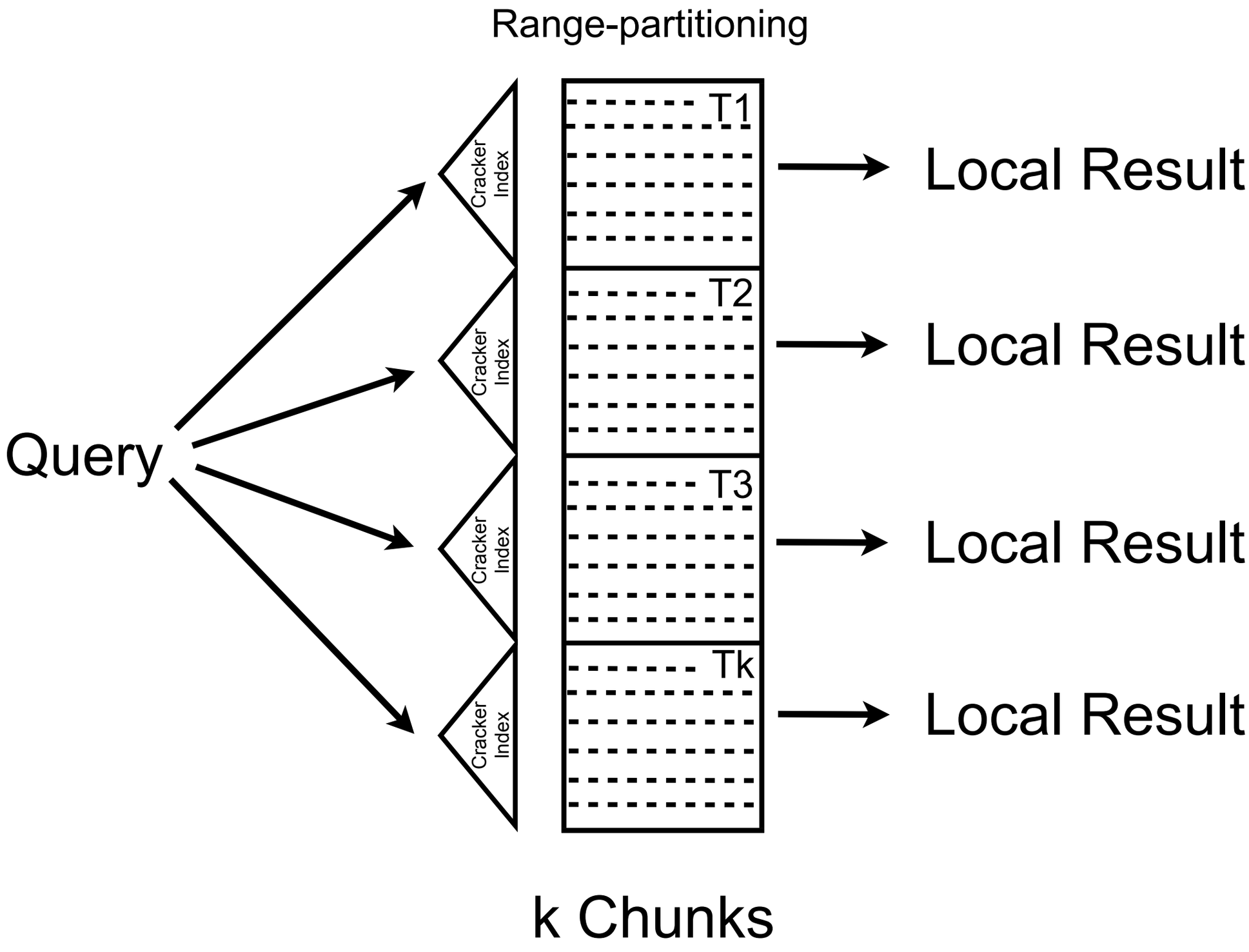
Multi-threaded algorithms:
Parallel-chunked Coarse-Granular Index (P-CCGI)



Multi-threaded algorithms:
Parallel-chunked Coarse-Granular Index (P-CCGI)

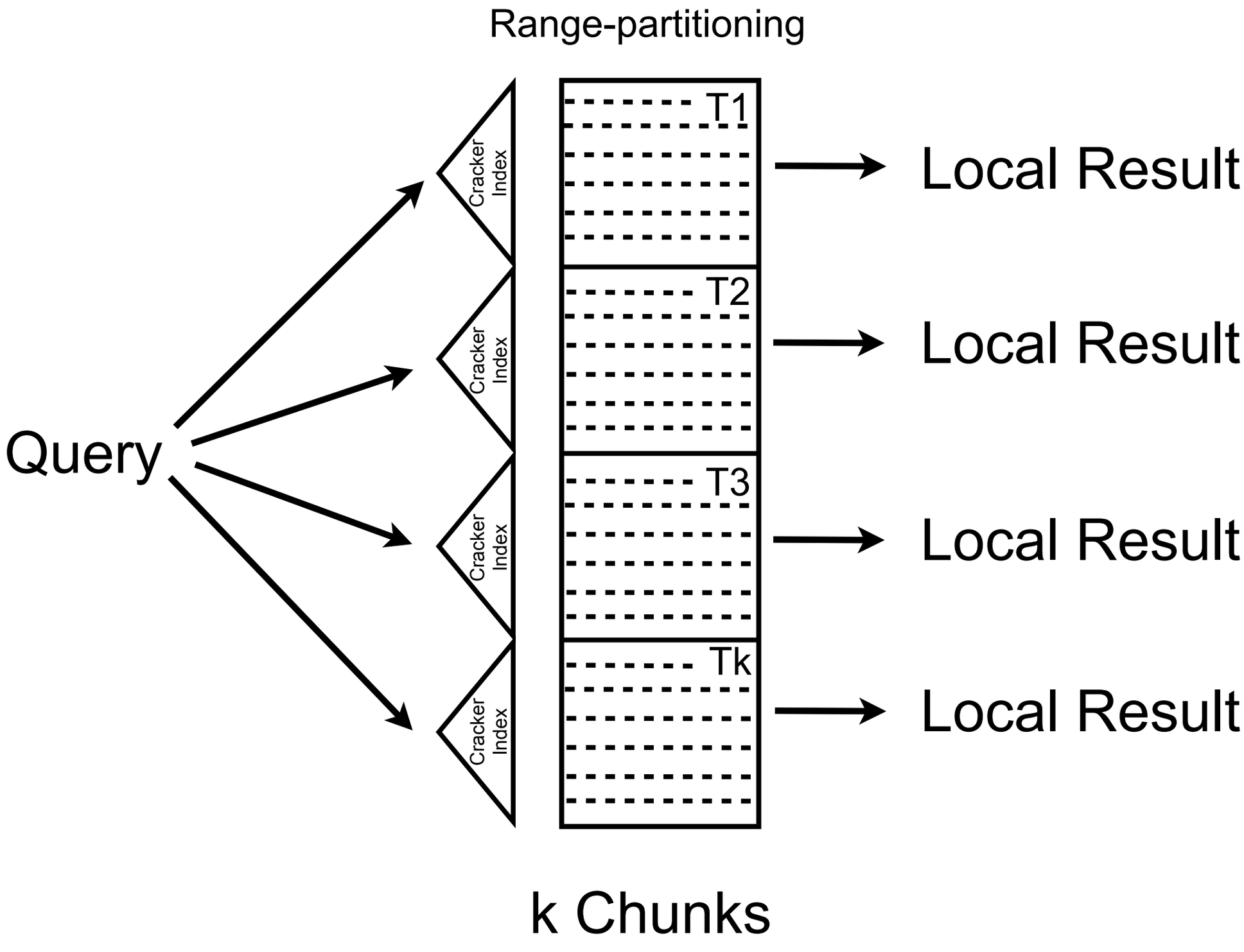


Multi-threaded algorithms:
Parallel-chunked Coarse-Granular Index (P-CCGI)

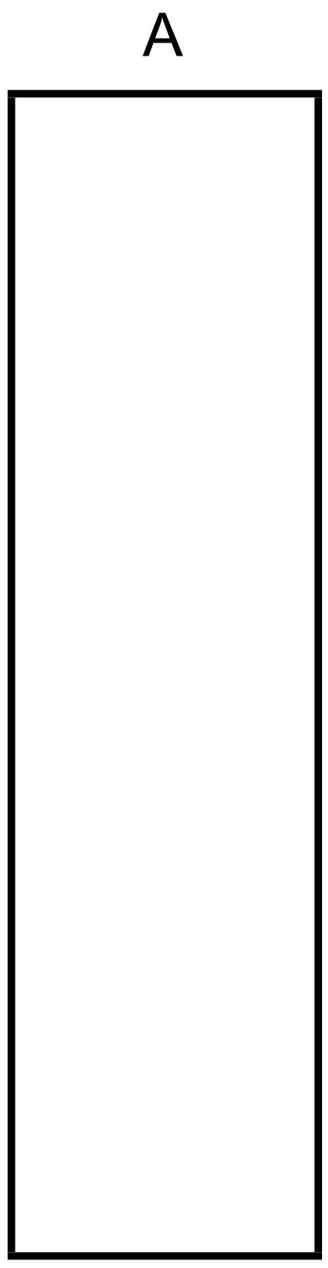


Multi-threaded algorithms:
Parallel-chunked Coarse-Granular Index (P-CCGI)

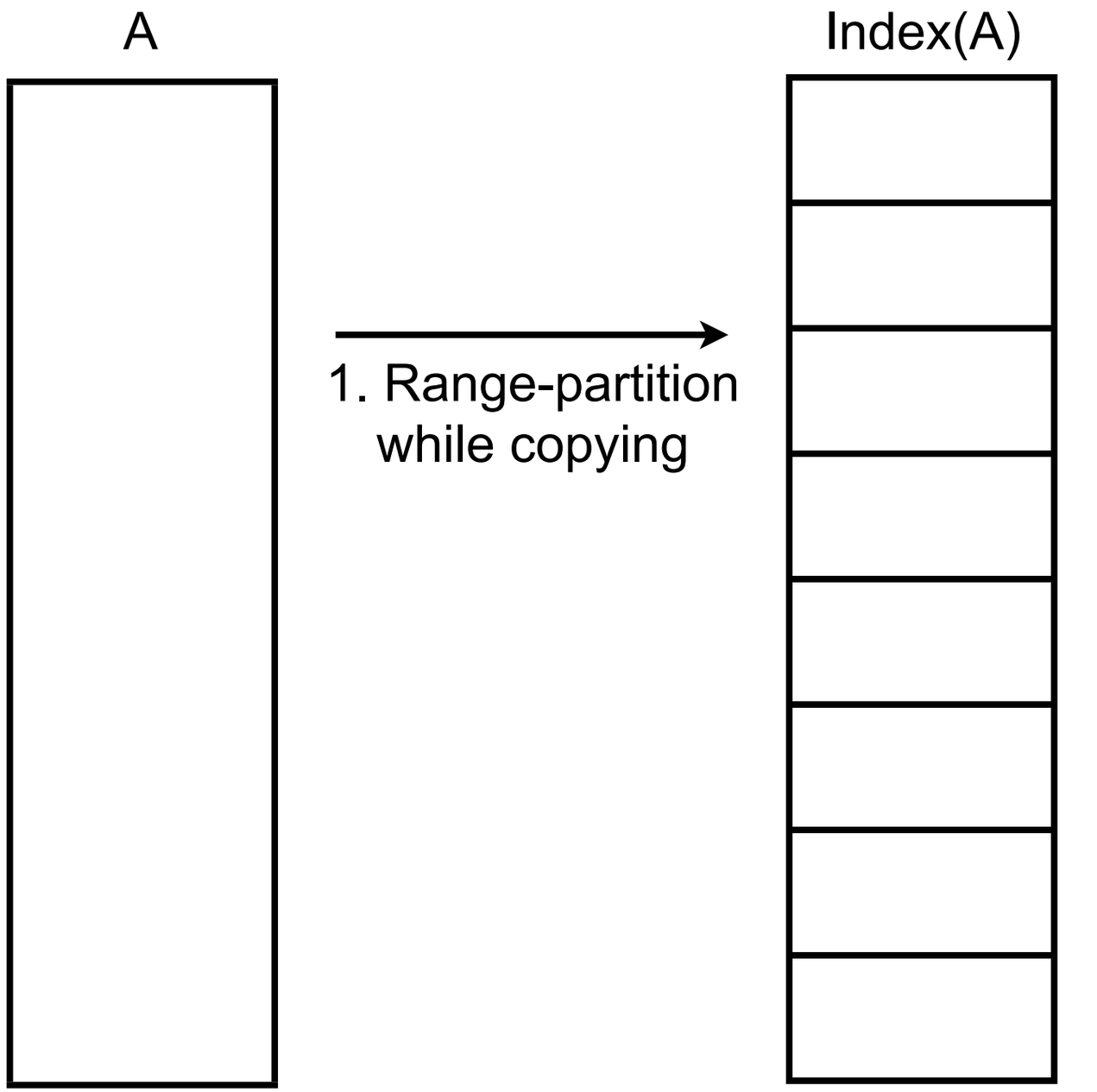
P-CSC + Range Partitioning



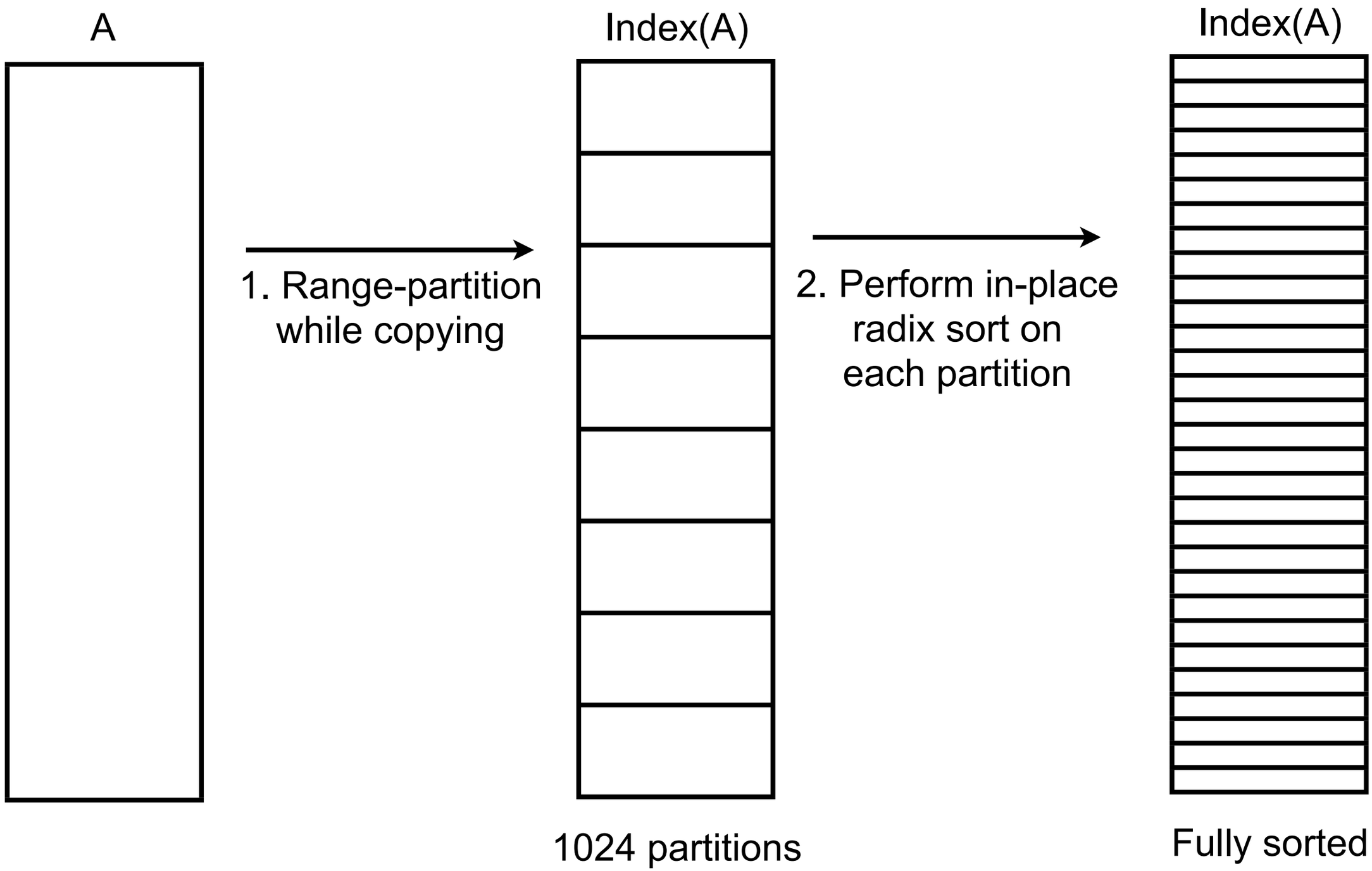
Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)



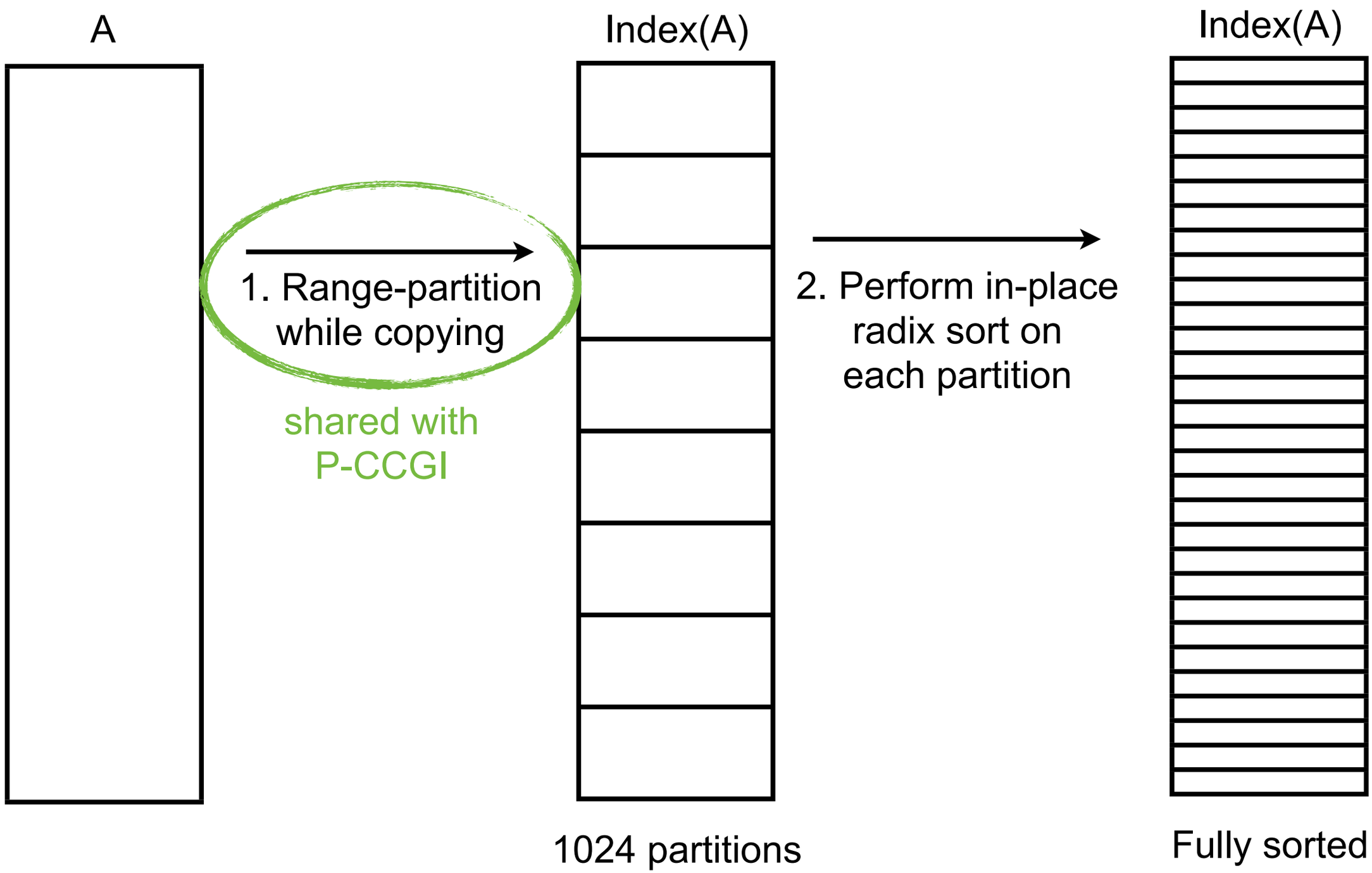
Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)



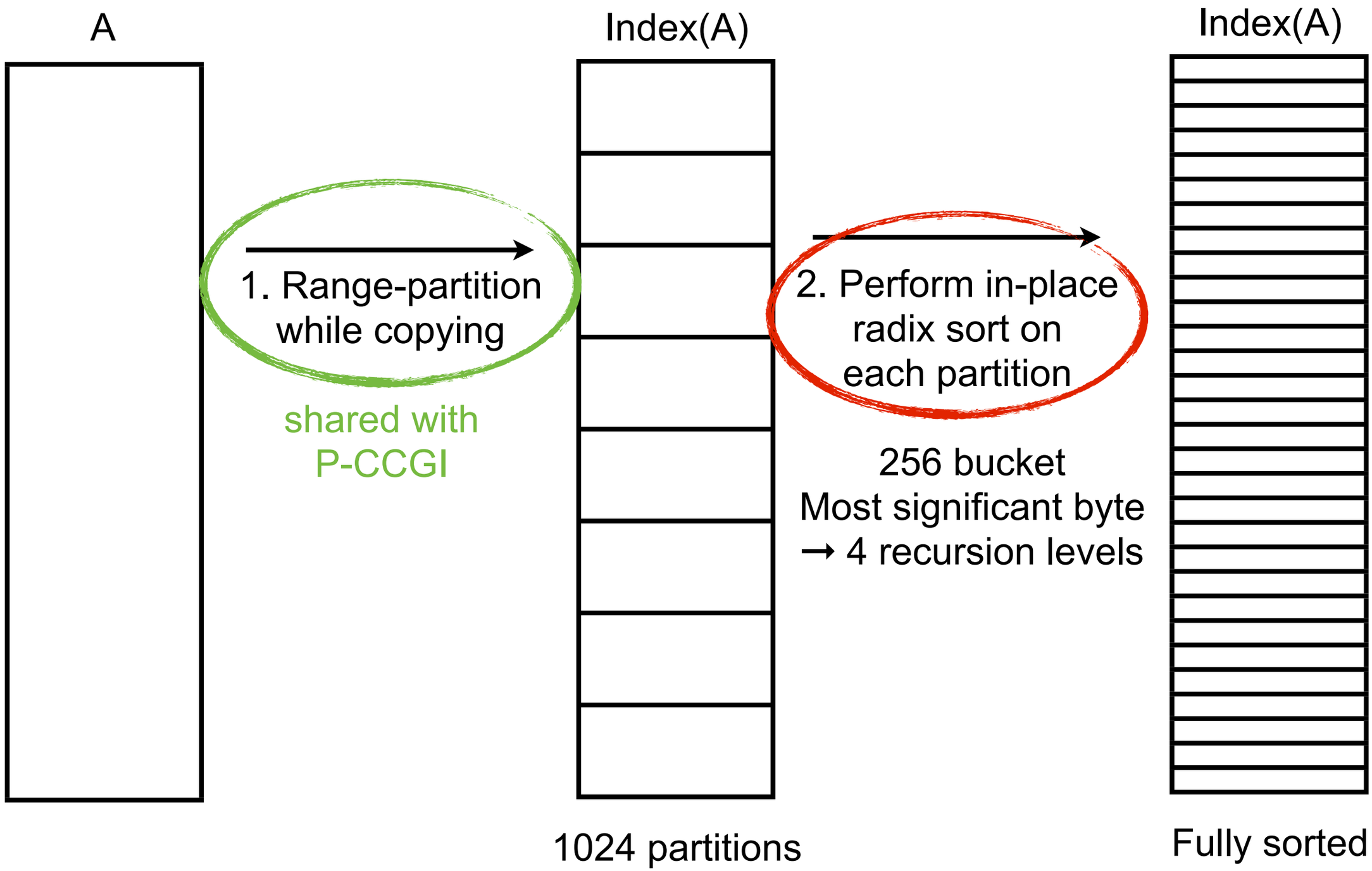
Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)



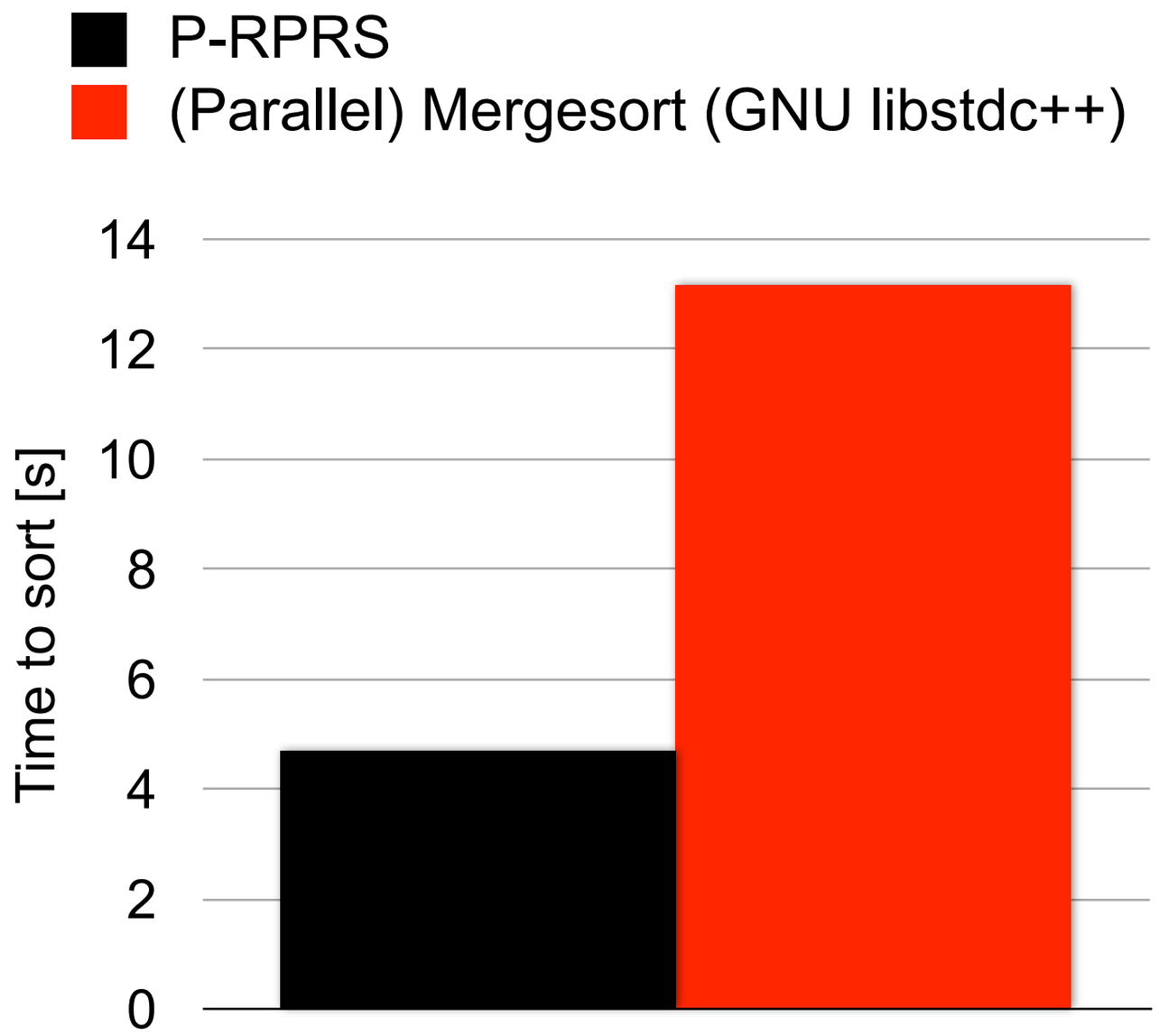
Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)



Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)

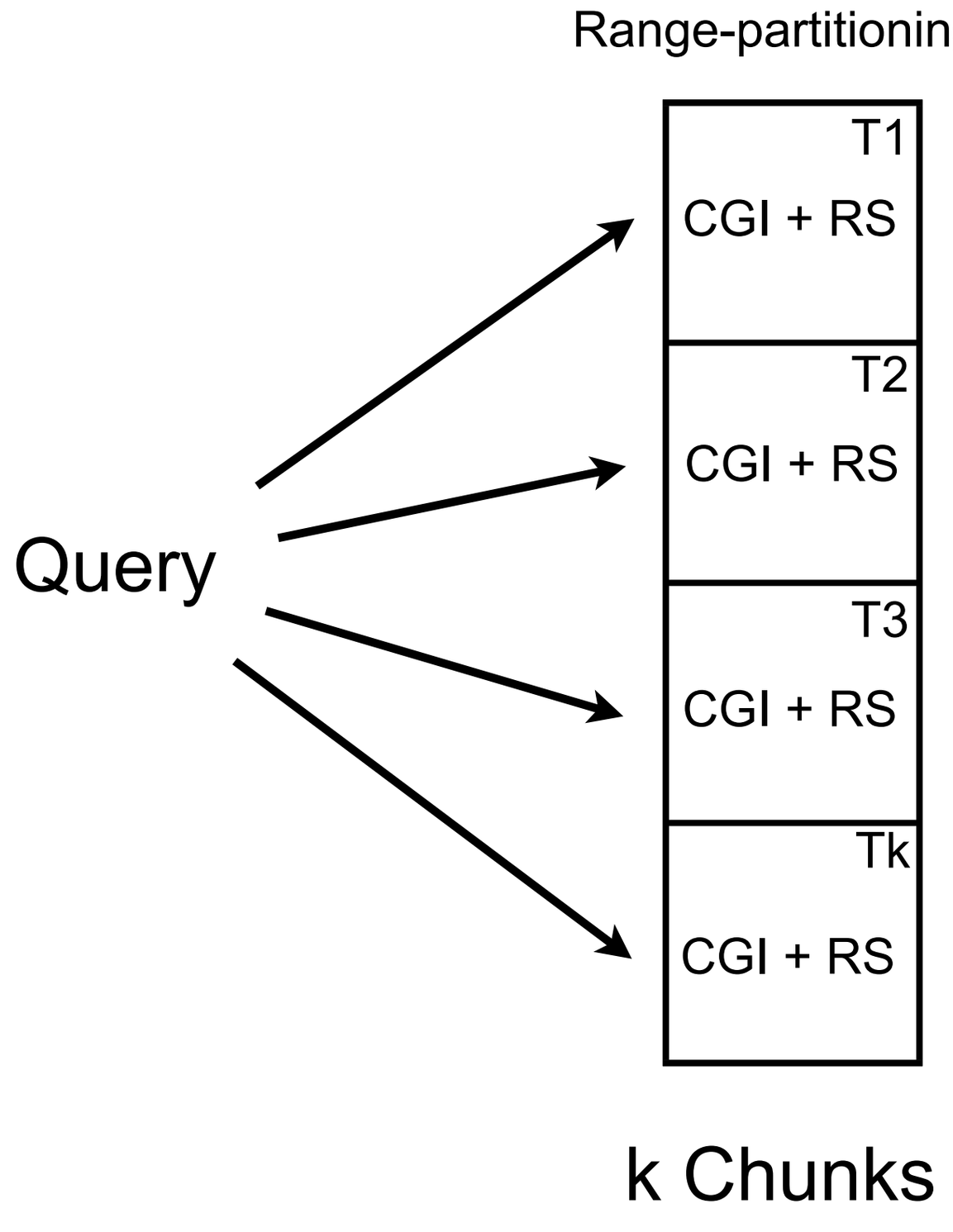


Multi-threaded algorithms:
Parallel Range-Partitioned Radix Sort (P-RPRS)

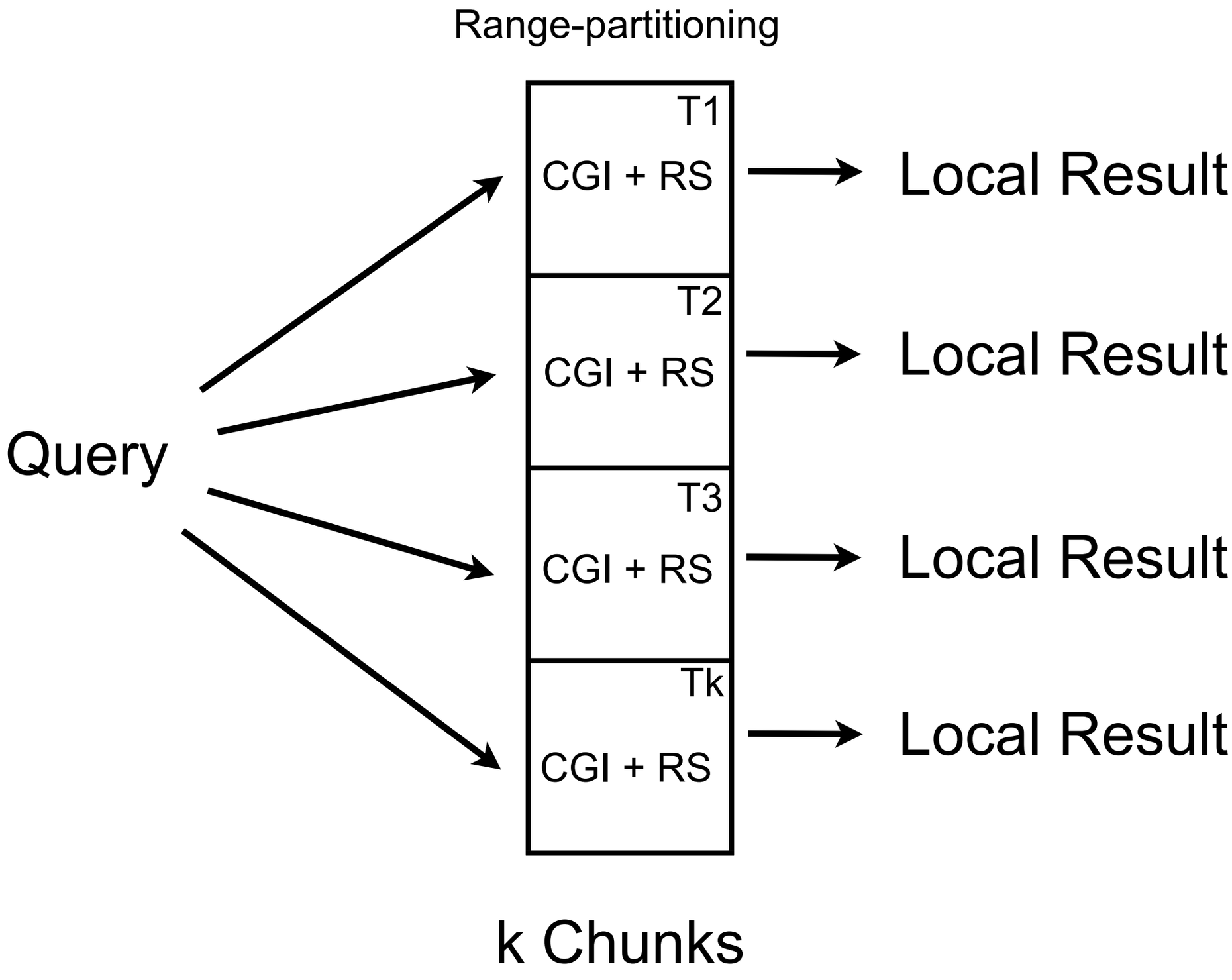


4 Cores / 8 Threads
512 million 4 byte integers
Uniform random distribution

Multi-threaded algorithms:
Parallel-chunked Range-Partitioned Radix Sort (P-CRS)

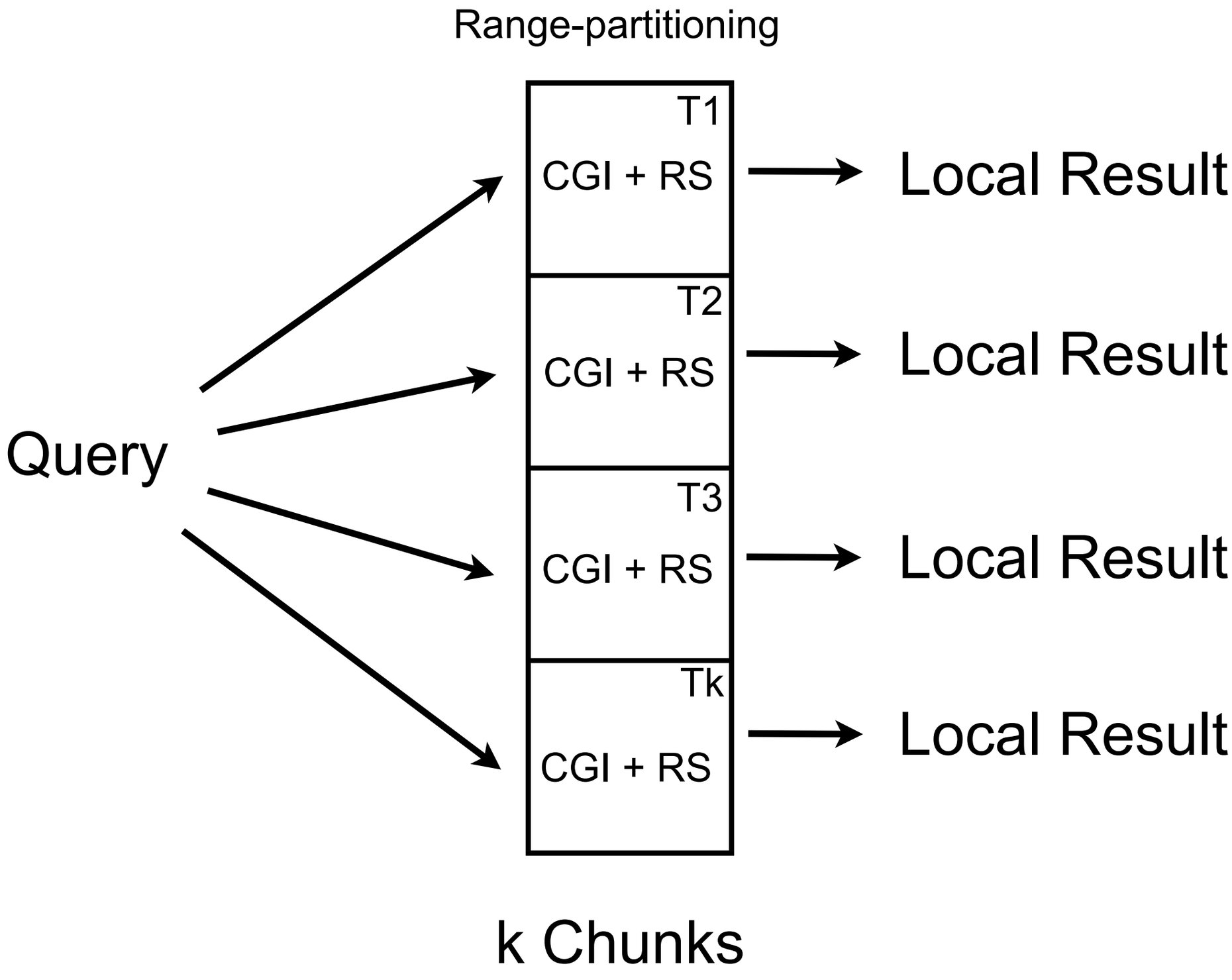


Multi-threaded algorithms:
Parallel-chunked Range-Partitioned Radix Sort (P-CRS)

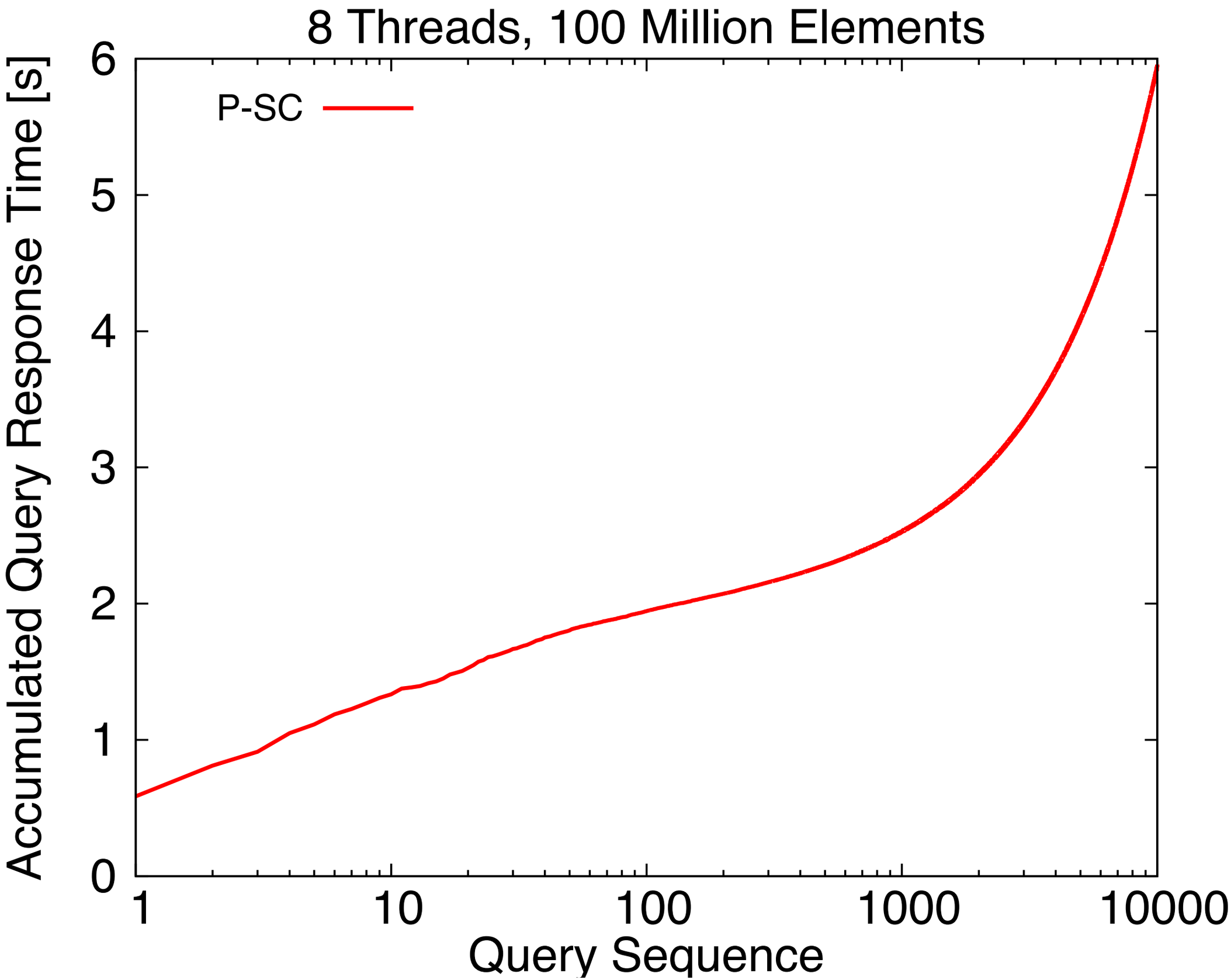


Multi-threaded algorithms:
Parallel-chunked Range-Partitioned Radix Sort (P-CRS)

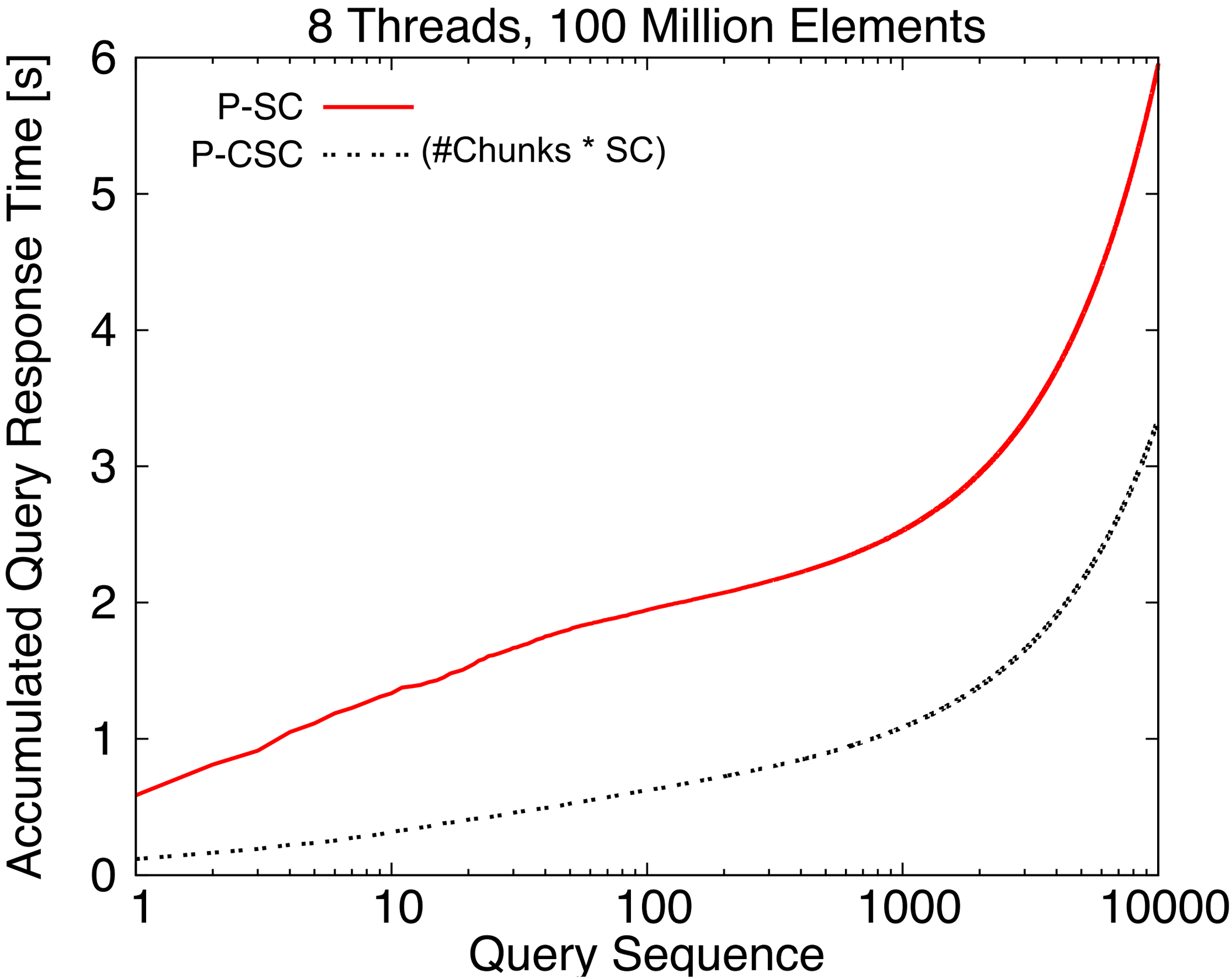
P-RPRS + Chunking



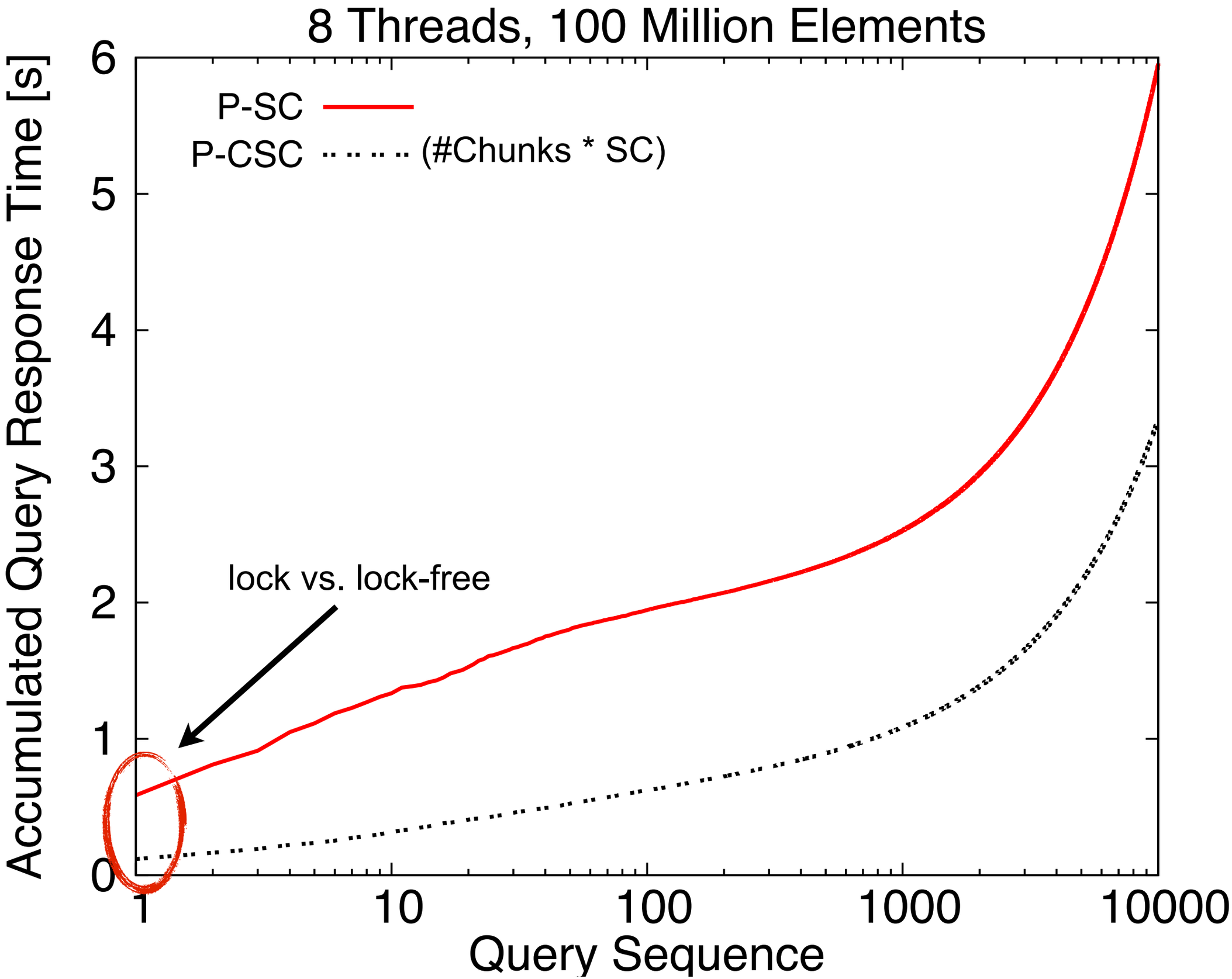
Multi-threaded Results



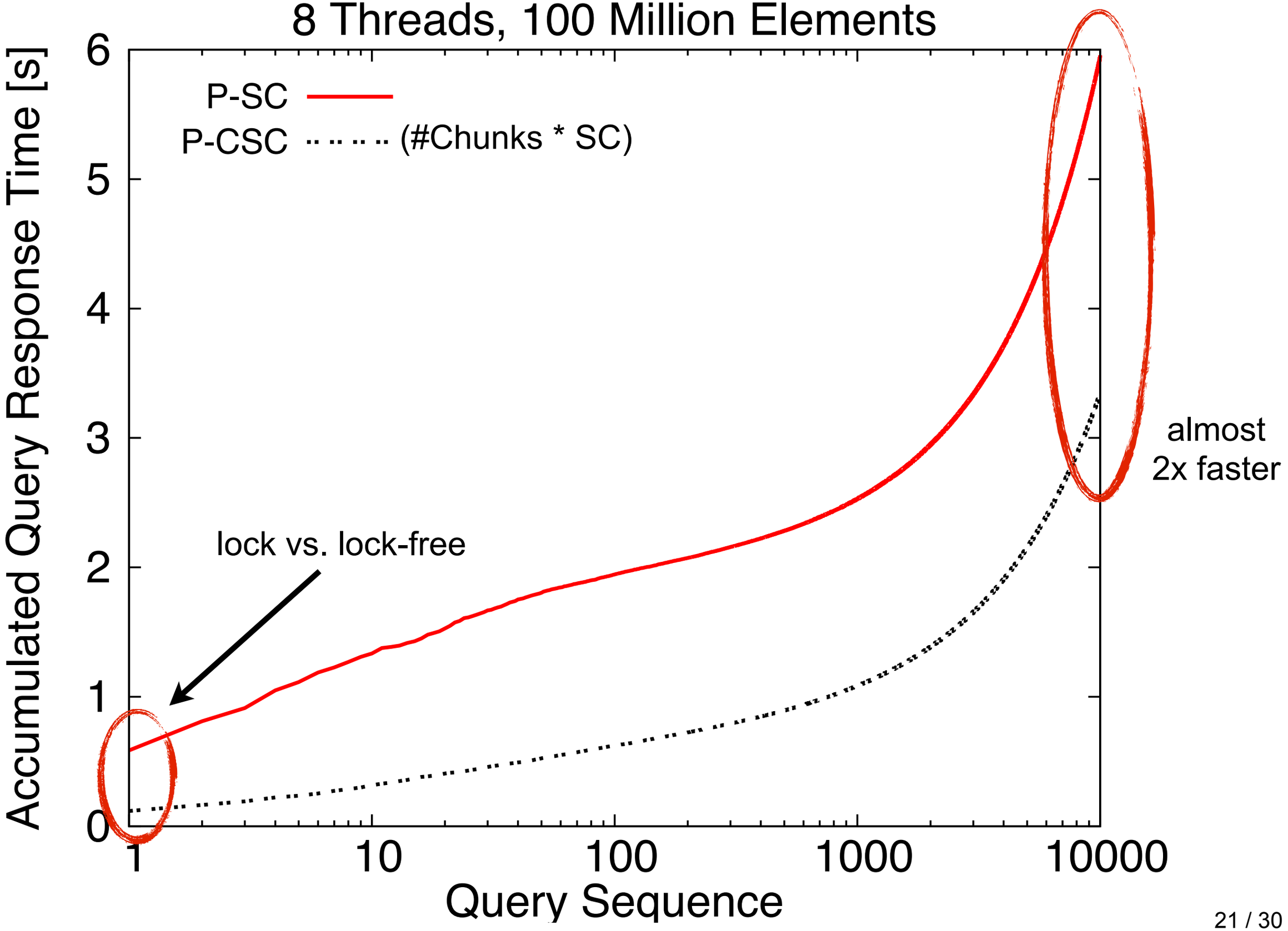
Multi-threaded Results



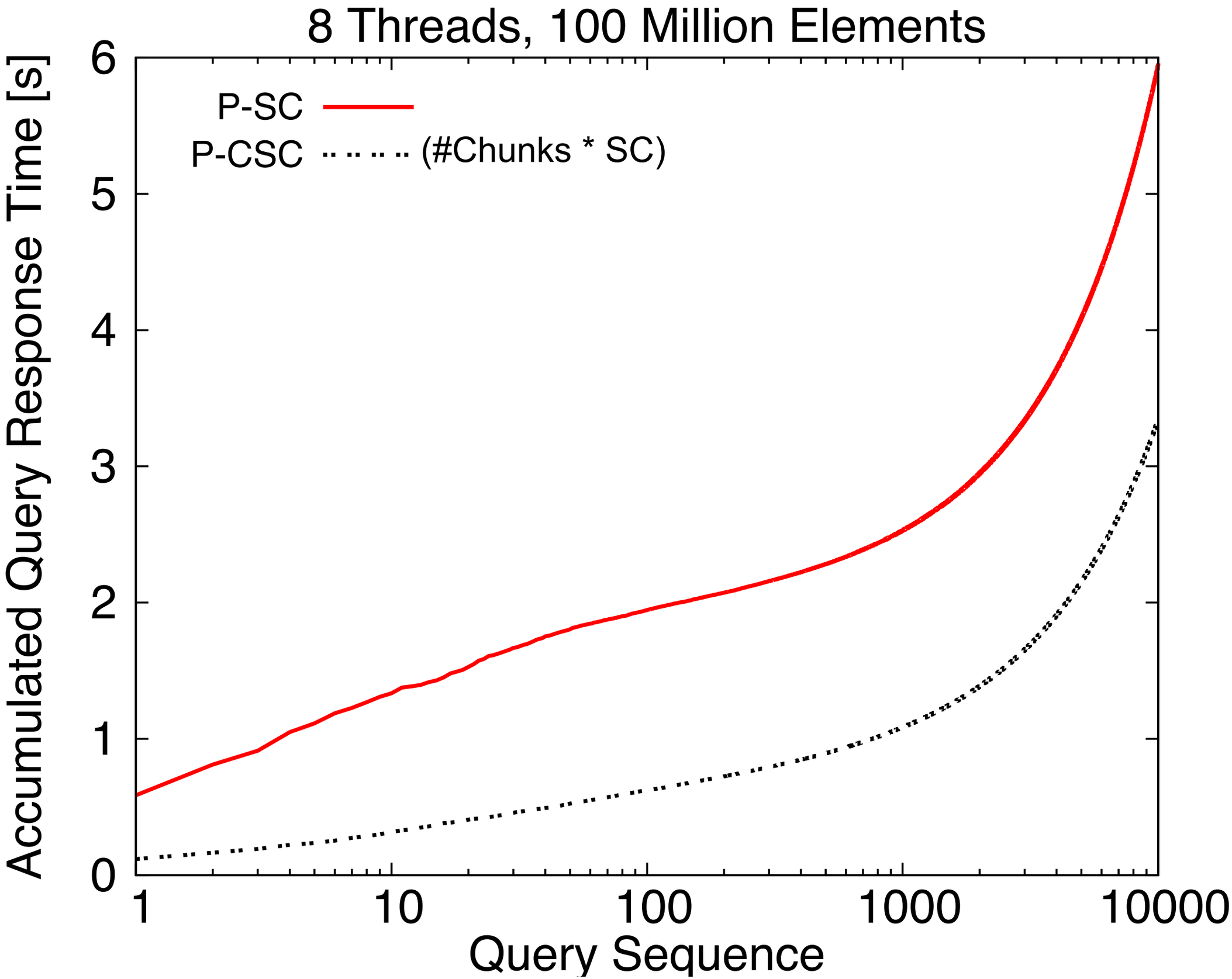
Multi-threaded Results



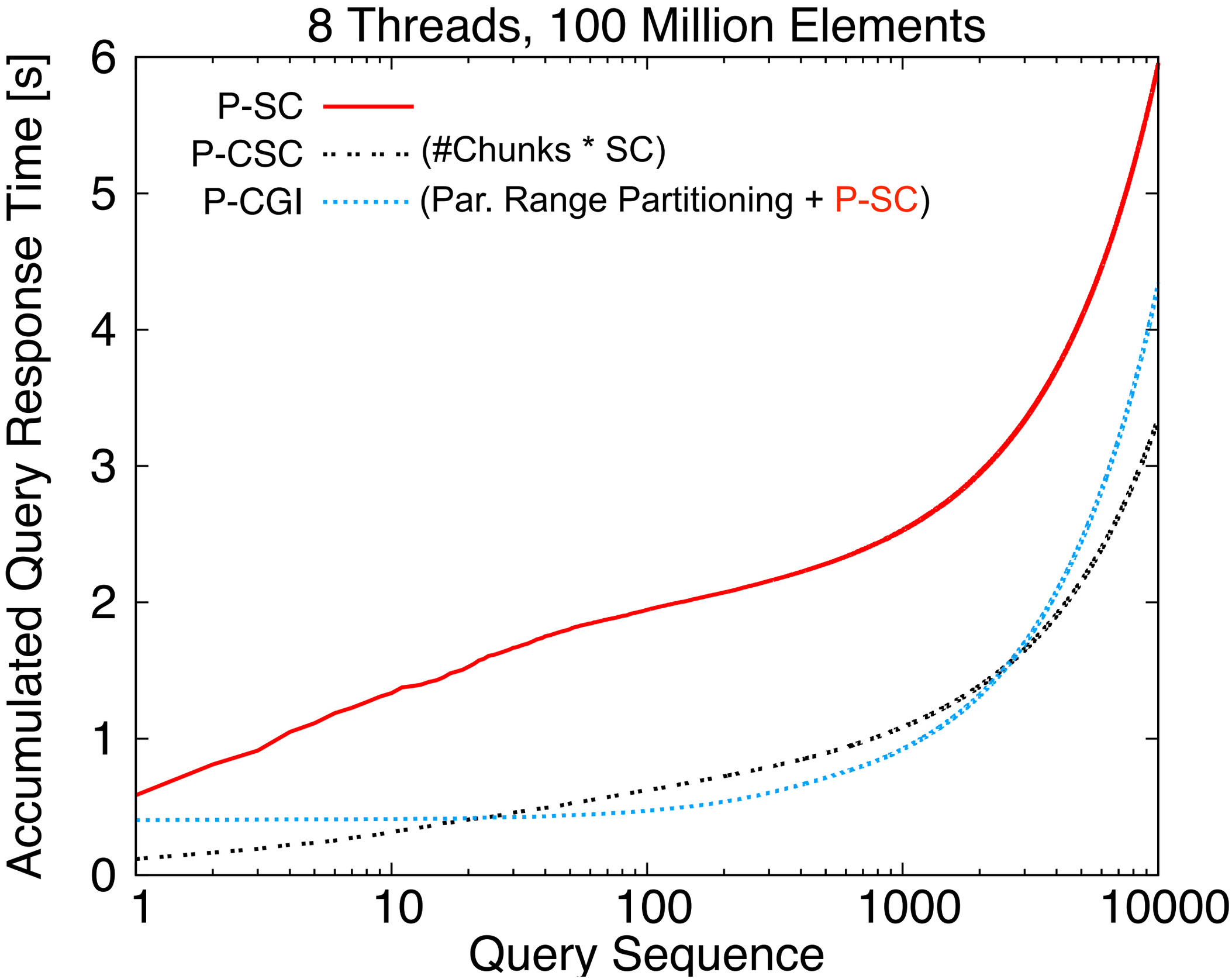
Multi-threaded Results



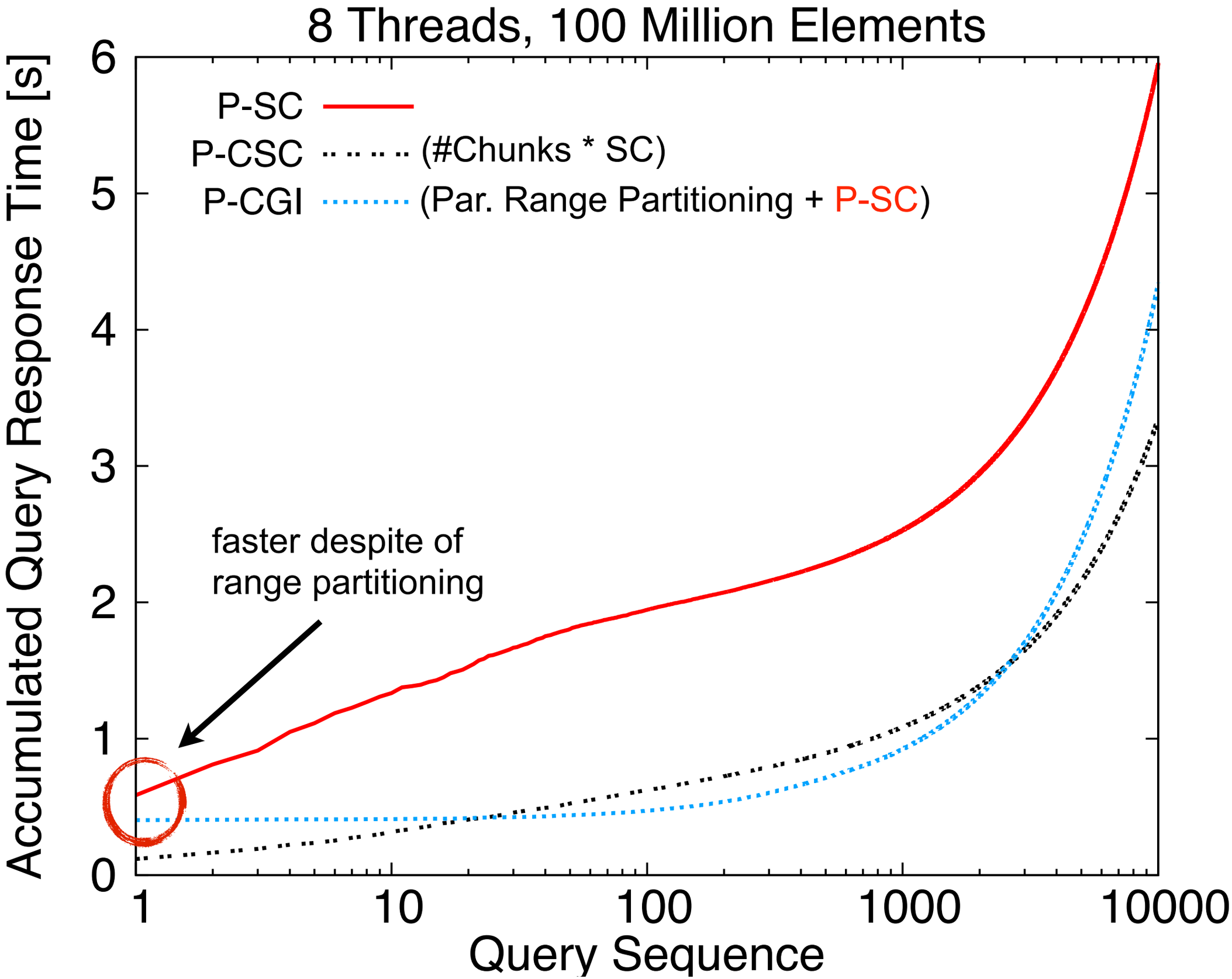
Multi-threaded Results



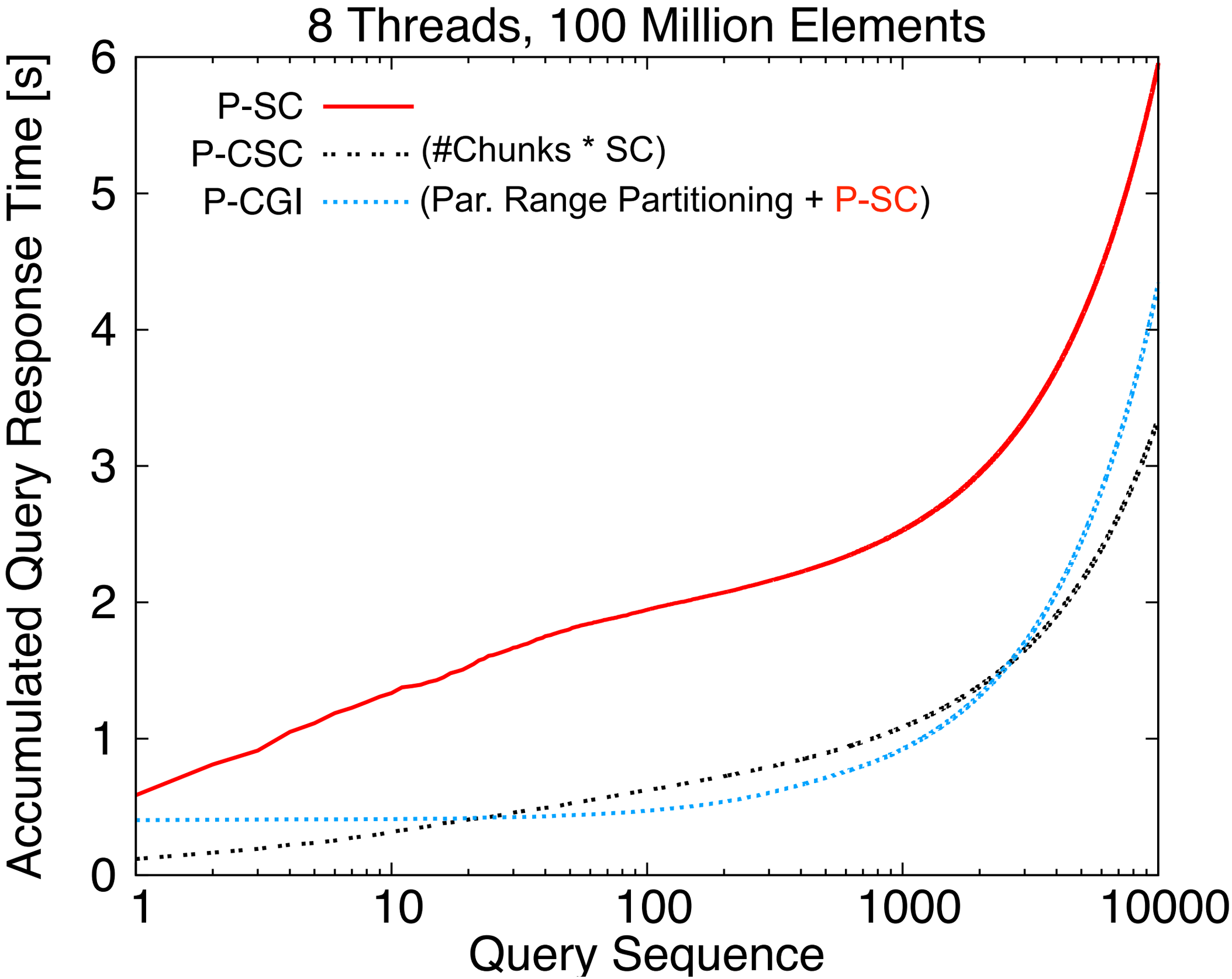
Multi-threaded Results



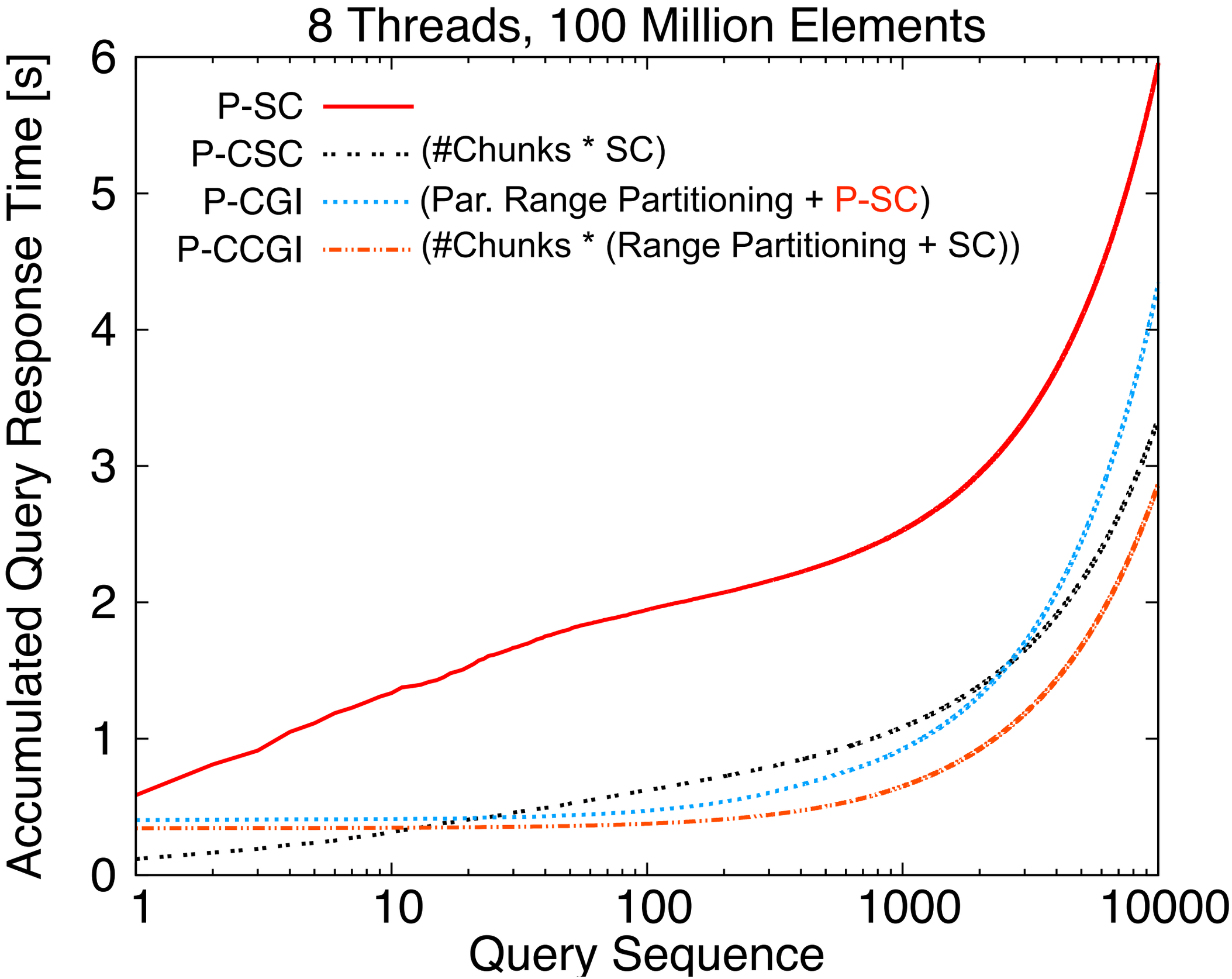
Multi-threaded Results



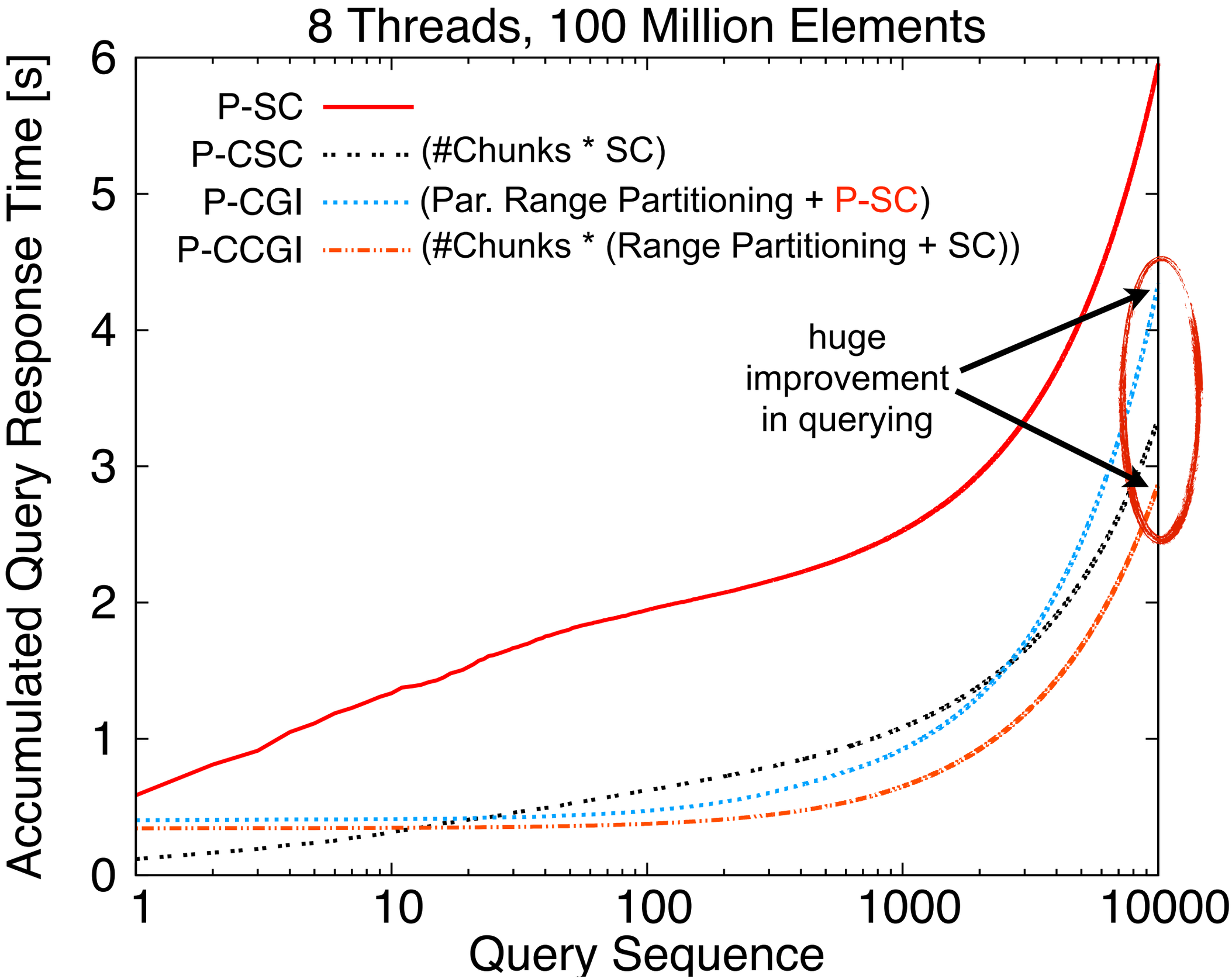
Multi-threaded Results



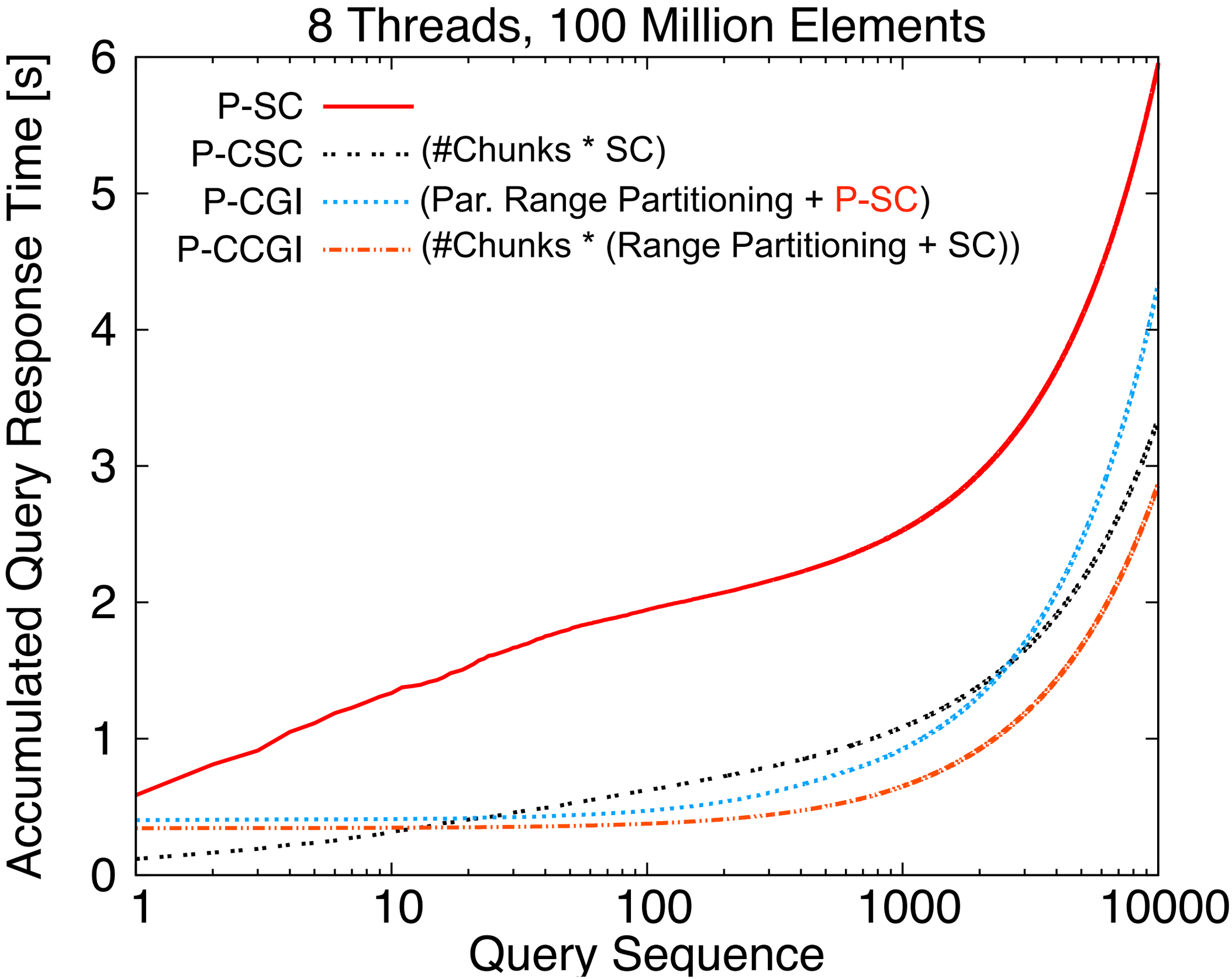
Multi-threaded Results



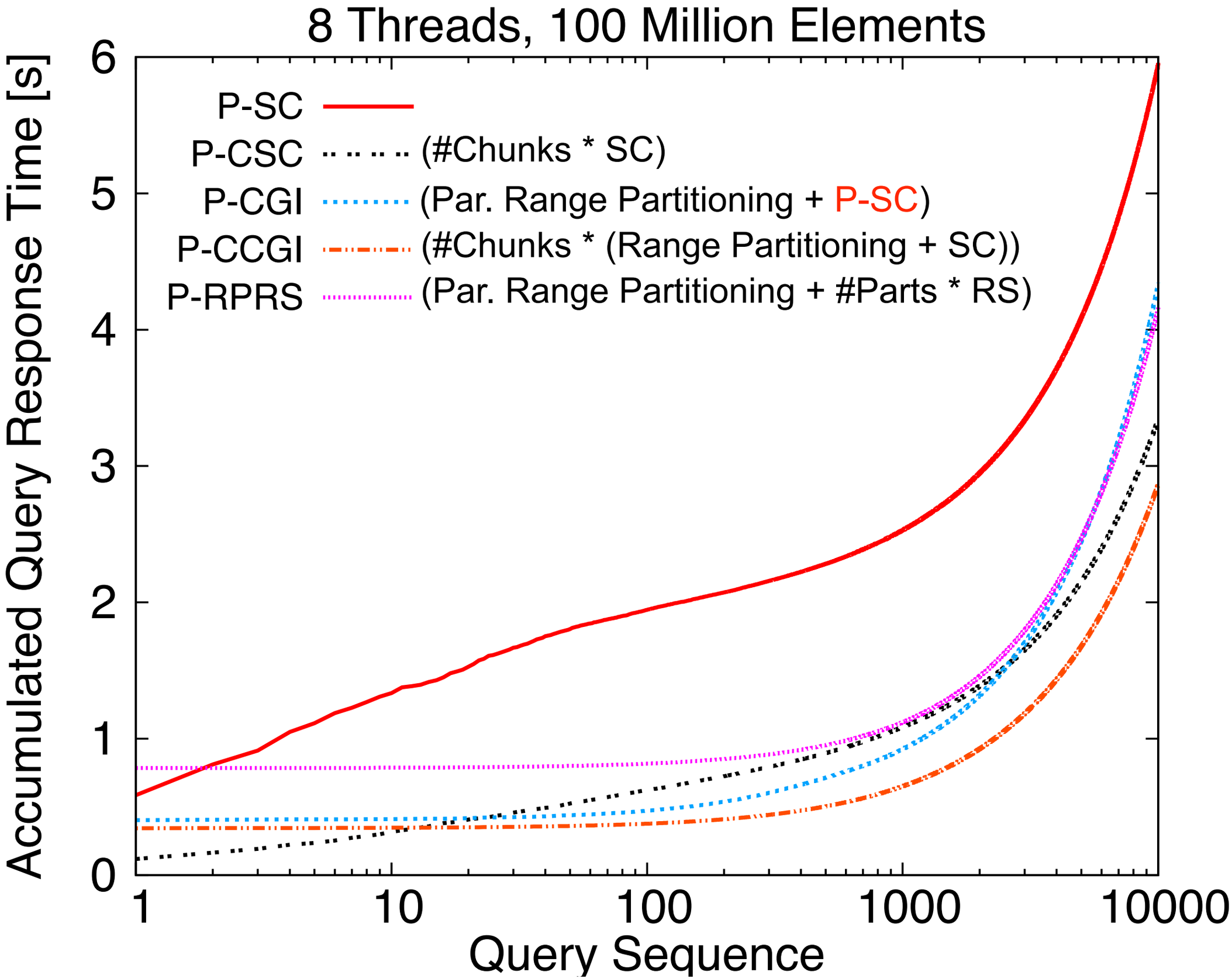
Multi-threaded Results



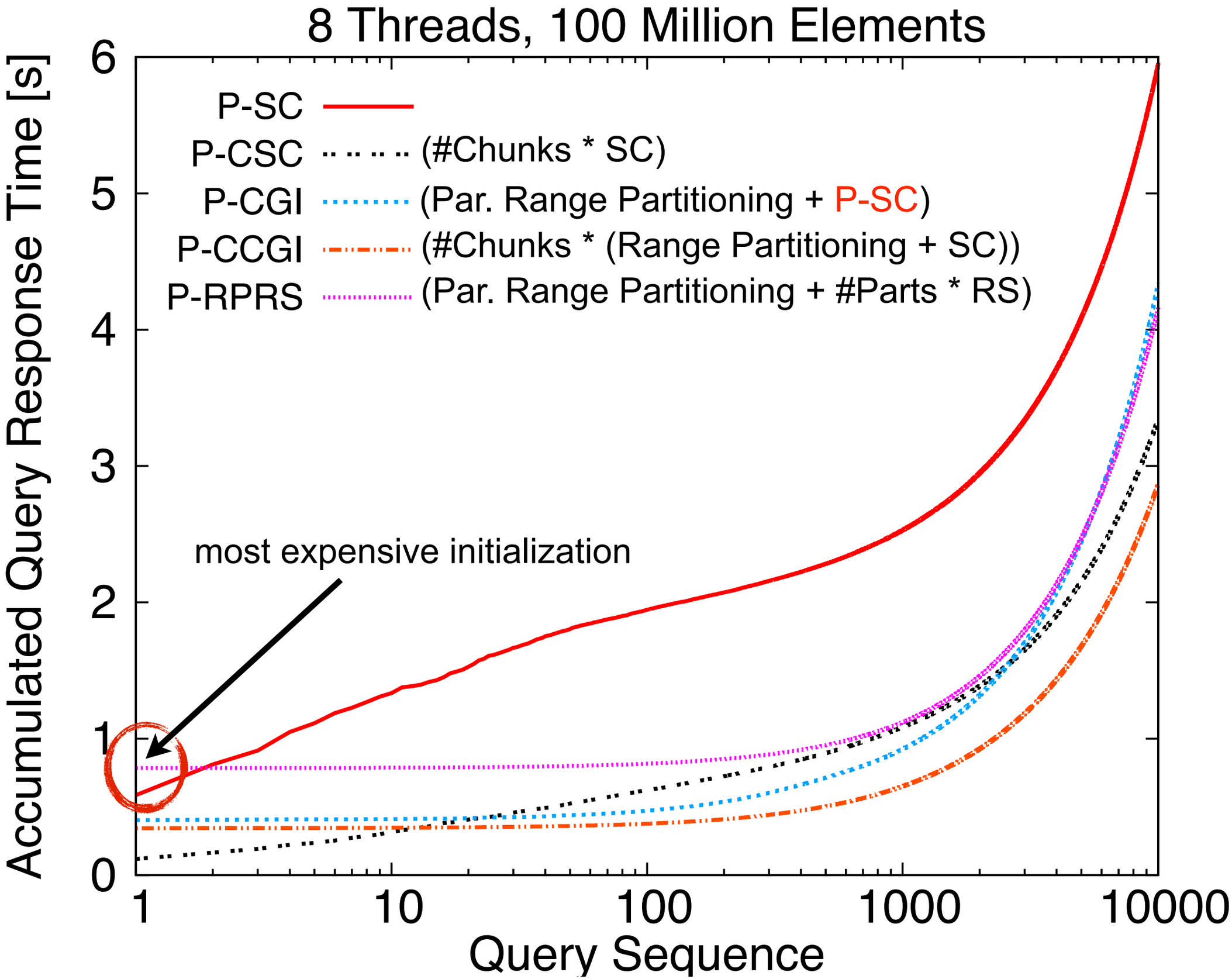
Multi-threaded Results



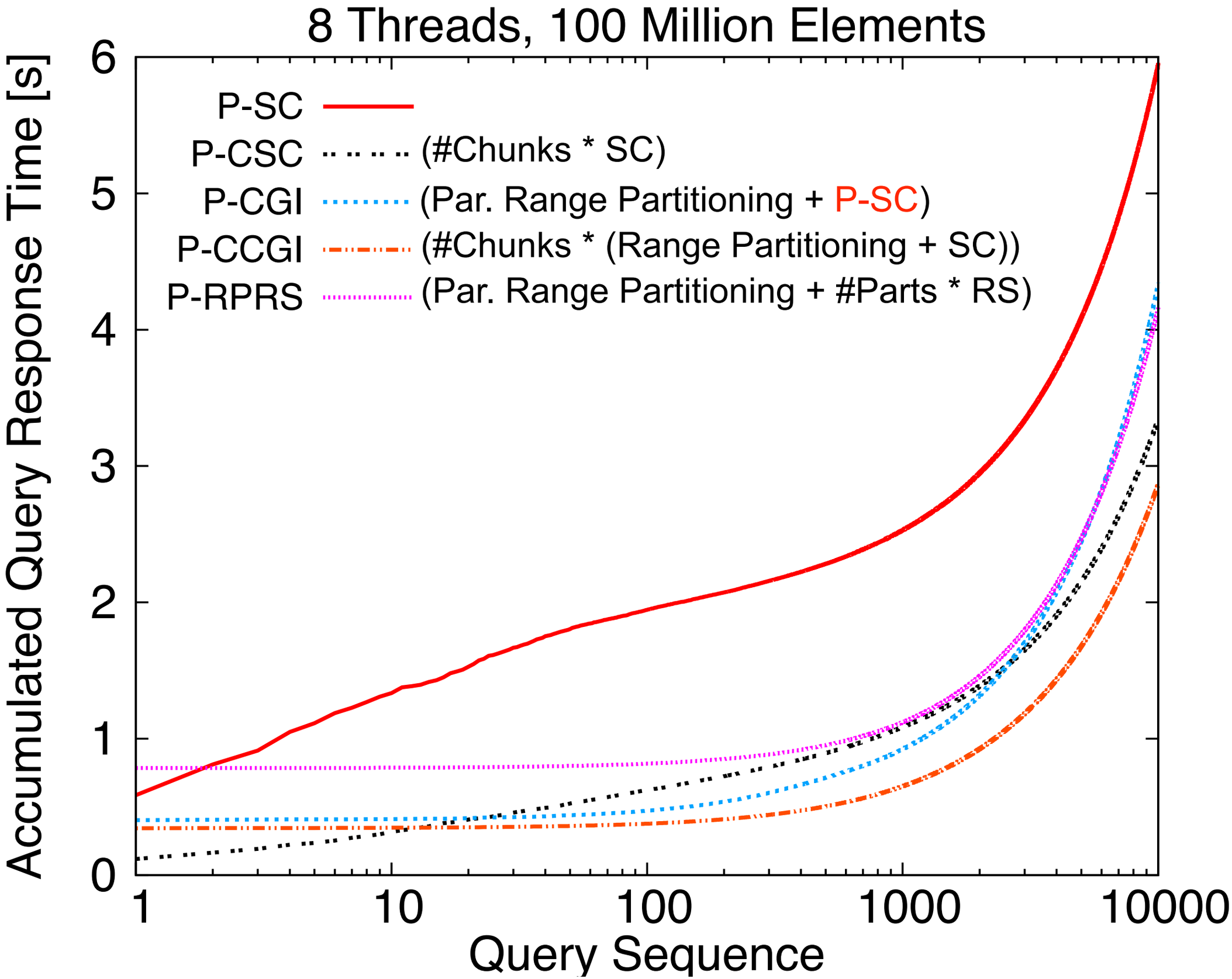
Multi-threaded Results



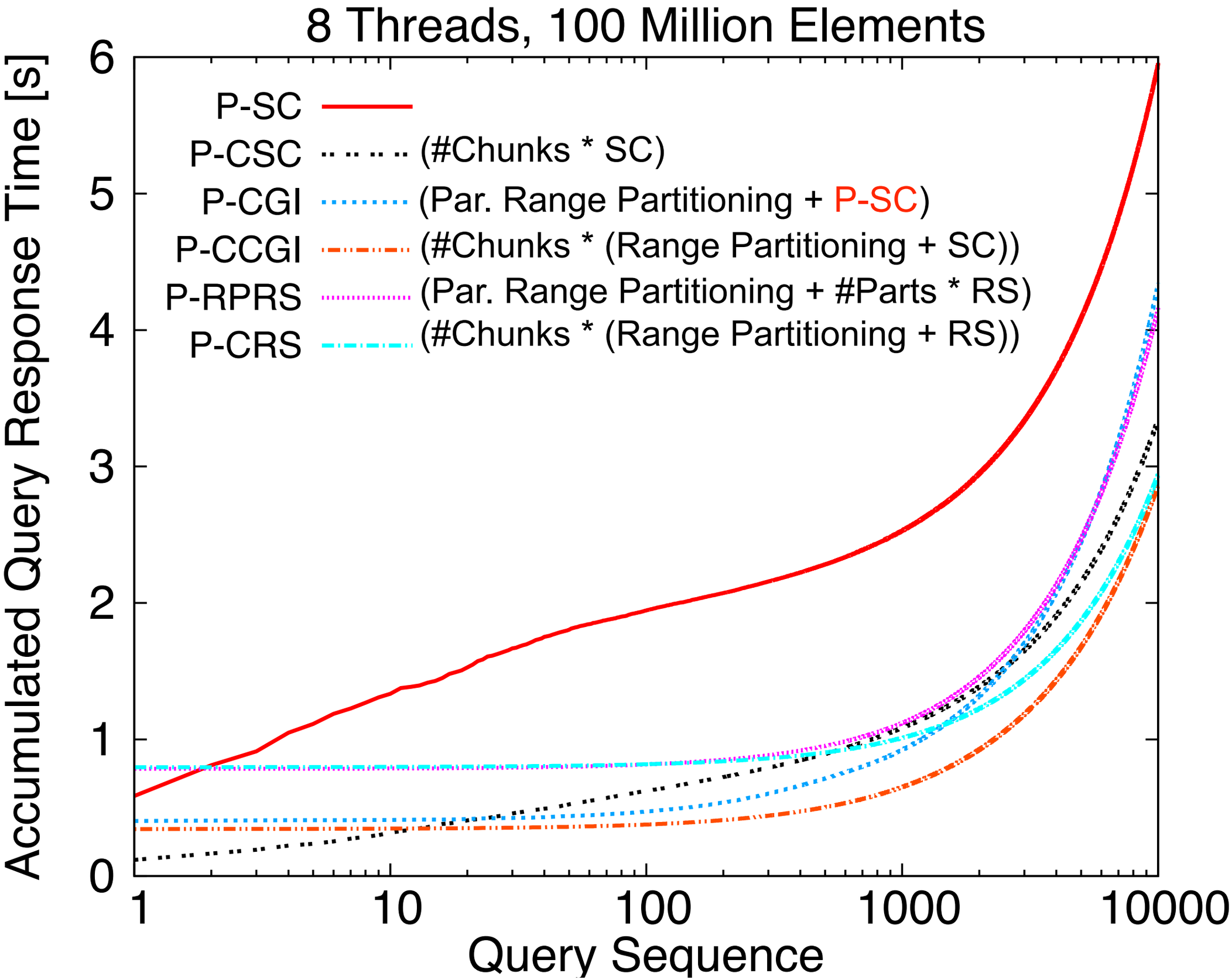
Multi-threaded Results



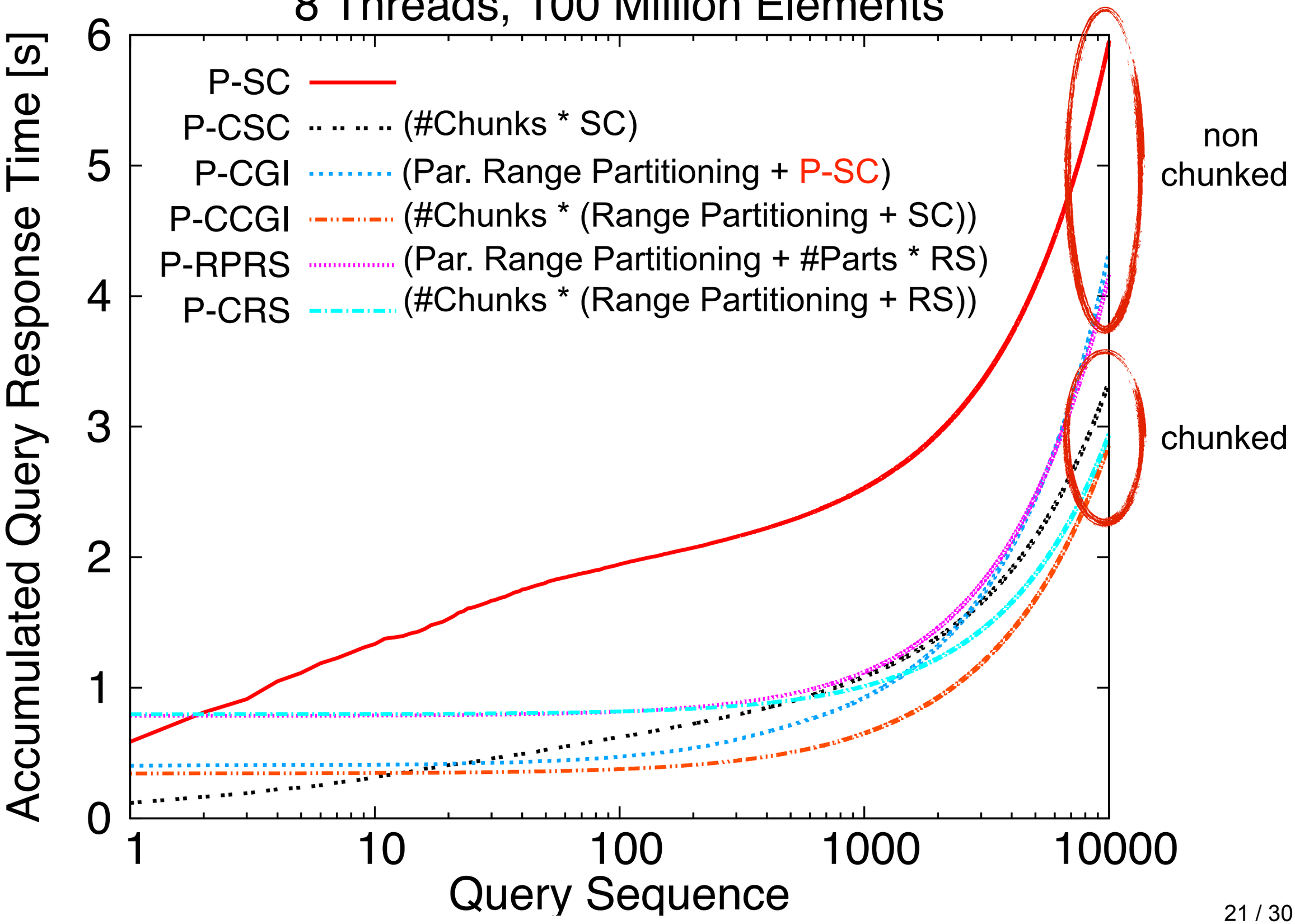
Multi-threaded Results



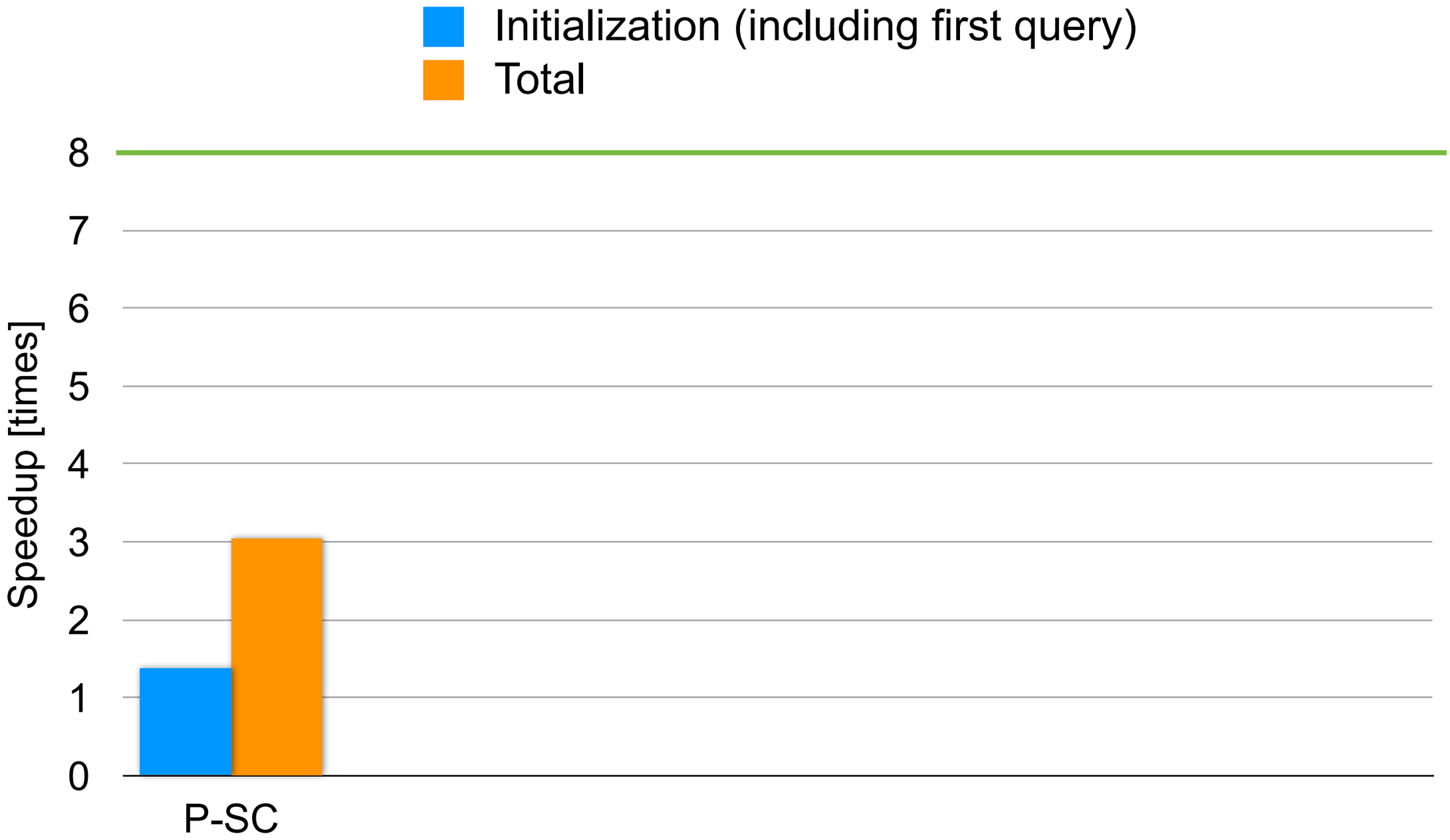
Multi-threaded Results



Multi-threaded Results

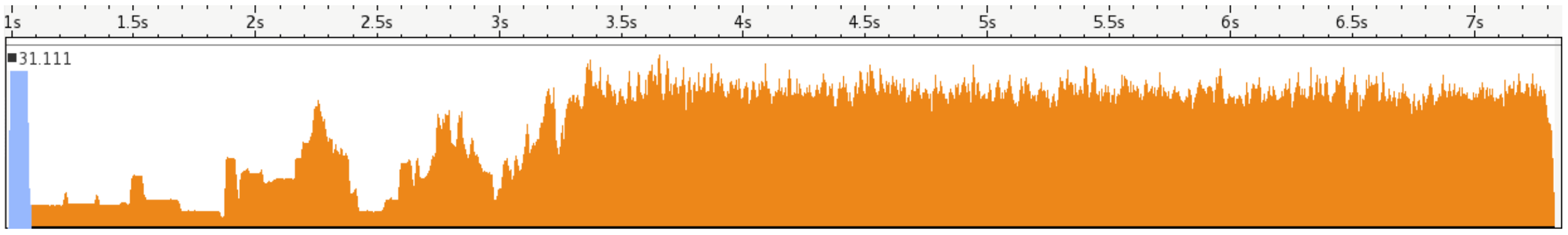


Multi-threaded Results
Factor Speedup from 1 to 8 Threads



P-SC: Analysis

Bandwidth



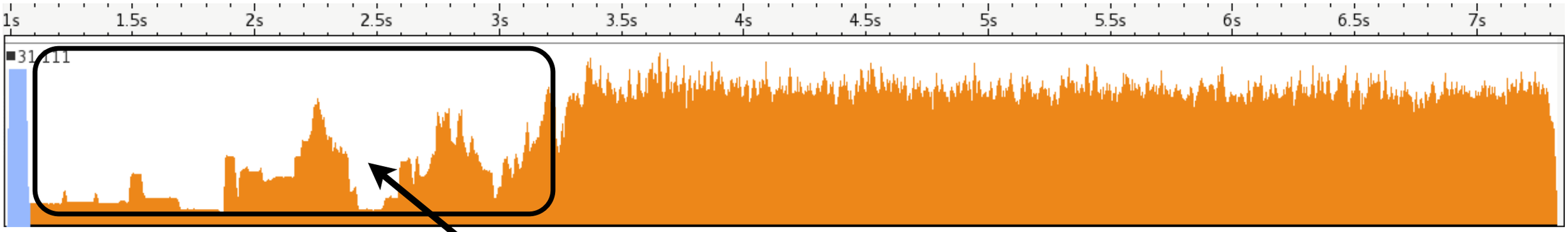
Lock Time

Mutex	Wait Time (sec)
Piece lock Cracker index lock	11.671 5.169
Total	16.84
Average (Total by 8)	2.105

Intel VTune Amplifier XE 2013 Data

P-SC: Analysis

Bandwidth

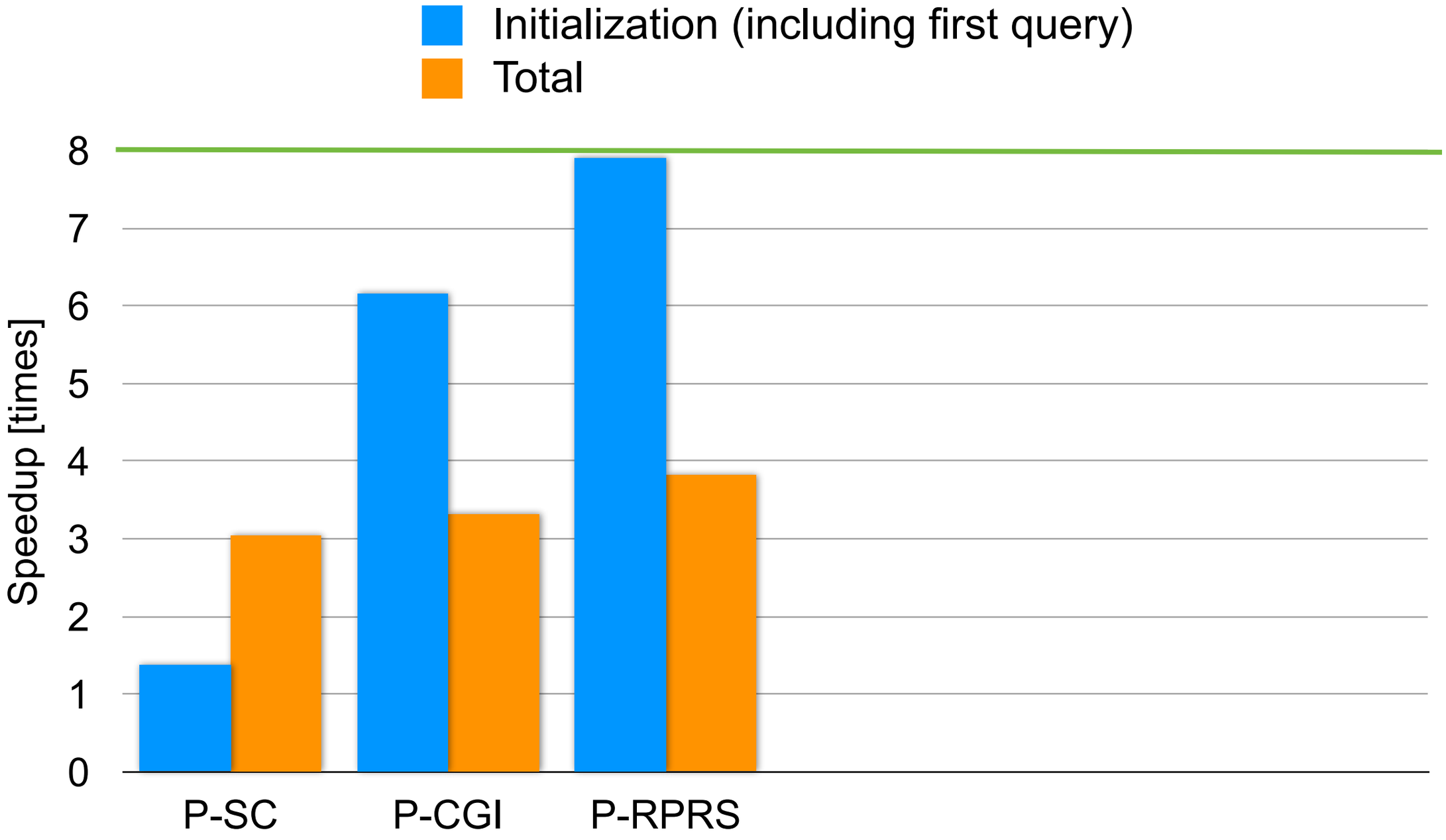


Lock Time

Mutex	Wait Time (sec)
Piece lock Cracker index lock	11.671 5.169
Total	16.84
Average (Total by 8)	2.105

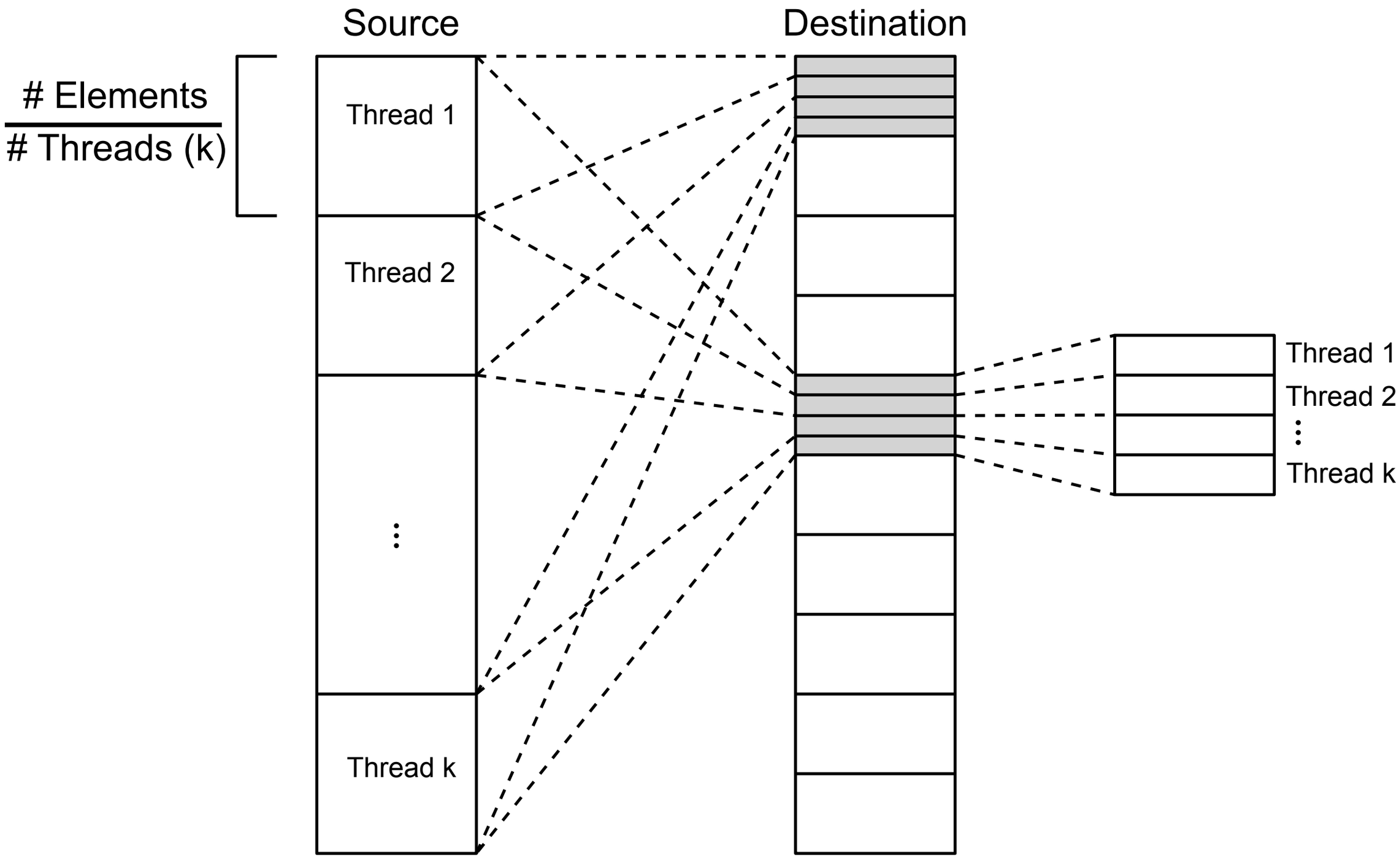
Intel VTune Amplifier XE 2013 Data

Multi-threaded Results
Factor Speedup from 1 to 8 Threads



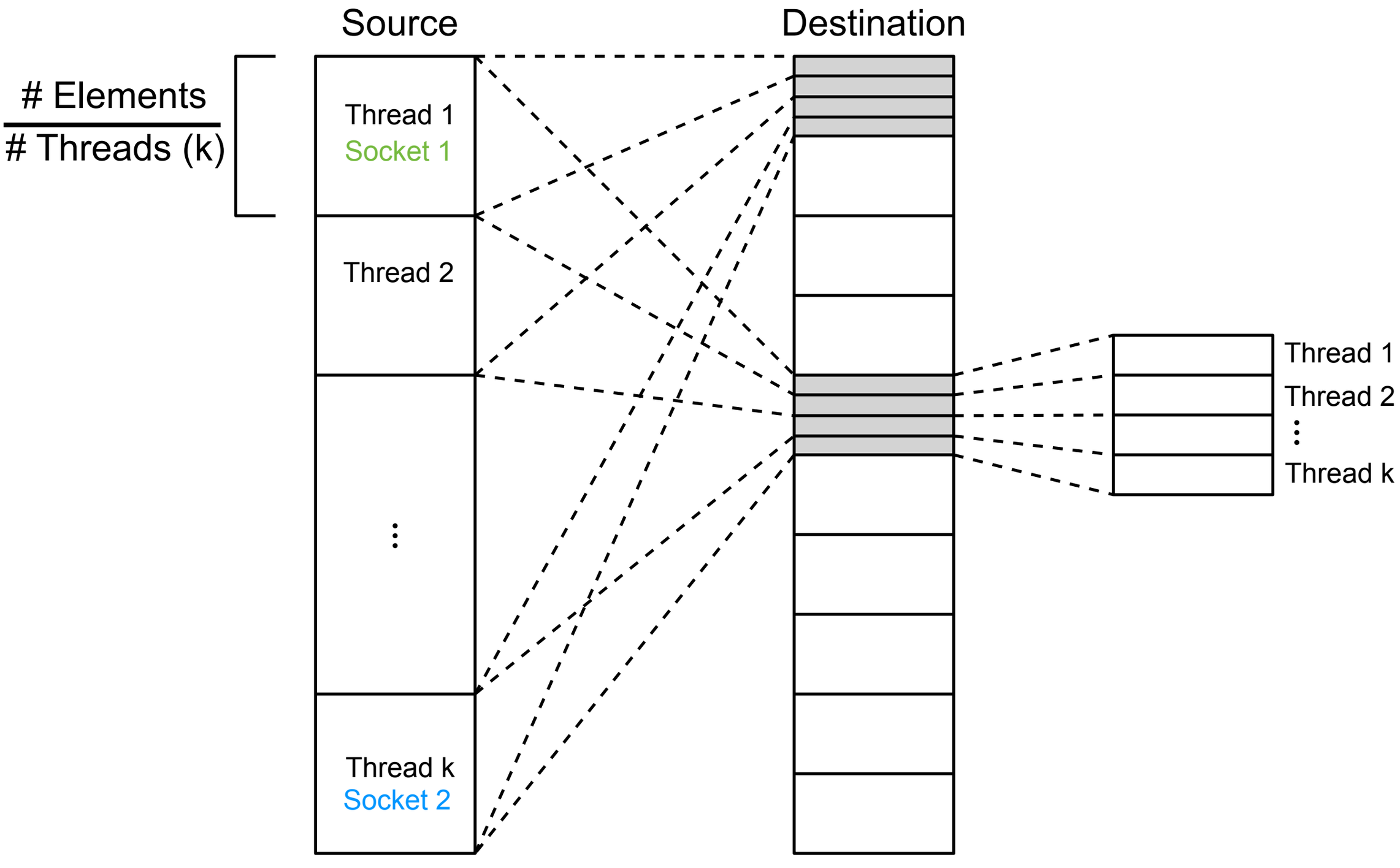
Non-Chunked Algorithms: Analysis (P-RPRS)

Range Partitioning (RP) Phase:



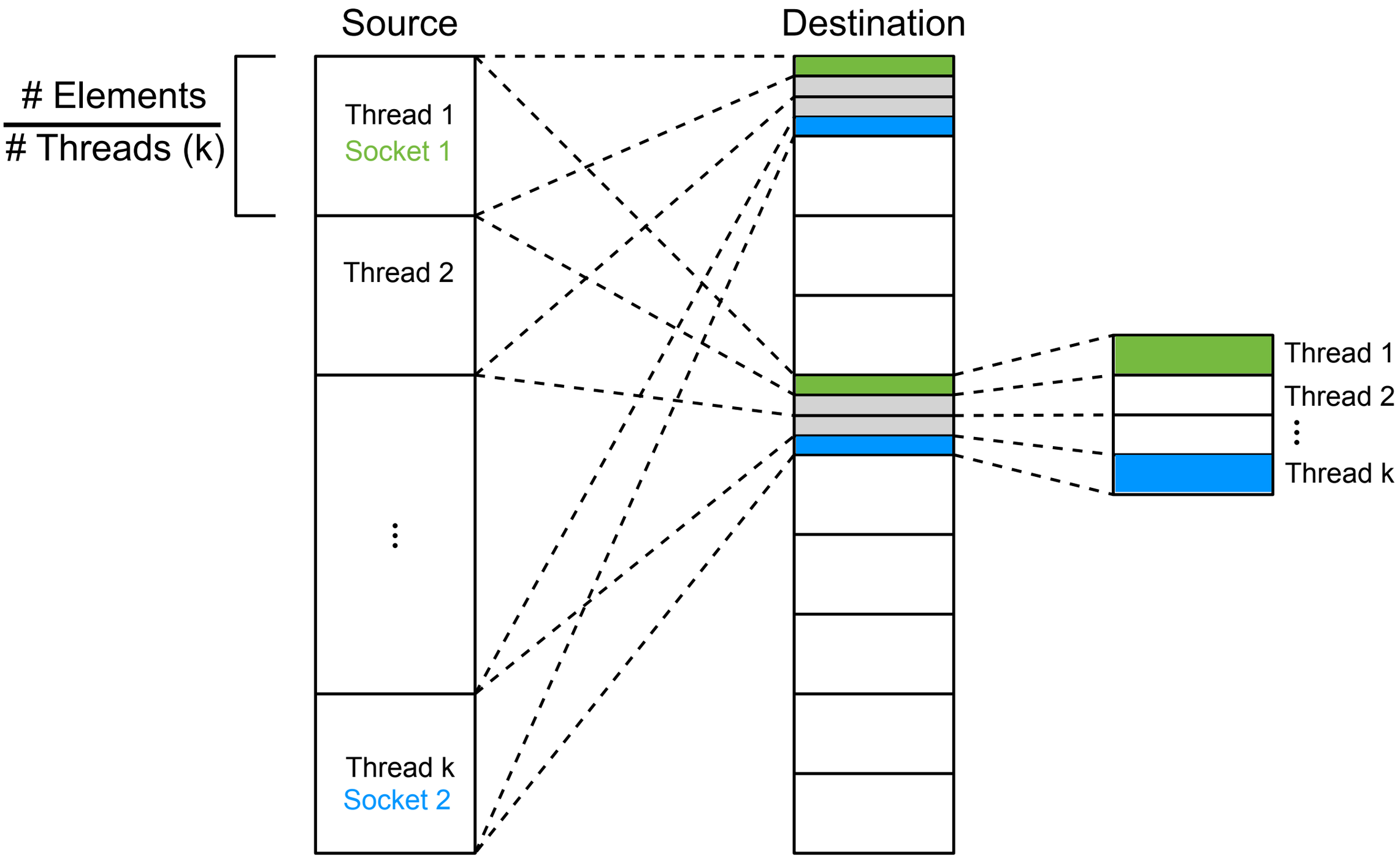
Non-Chunked Algorithms: Analysis (P-RPRS)

Range Partitioning (RP) Phase:



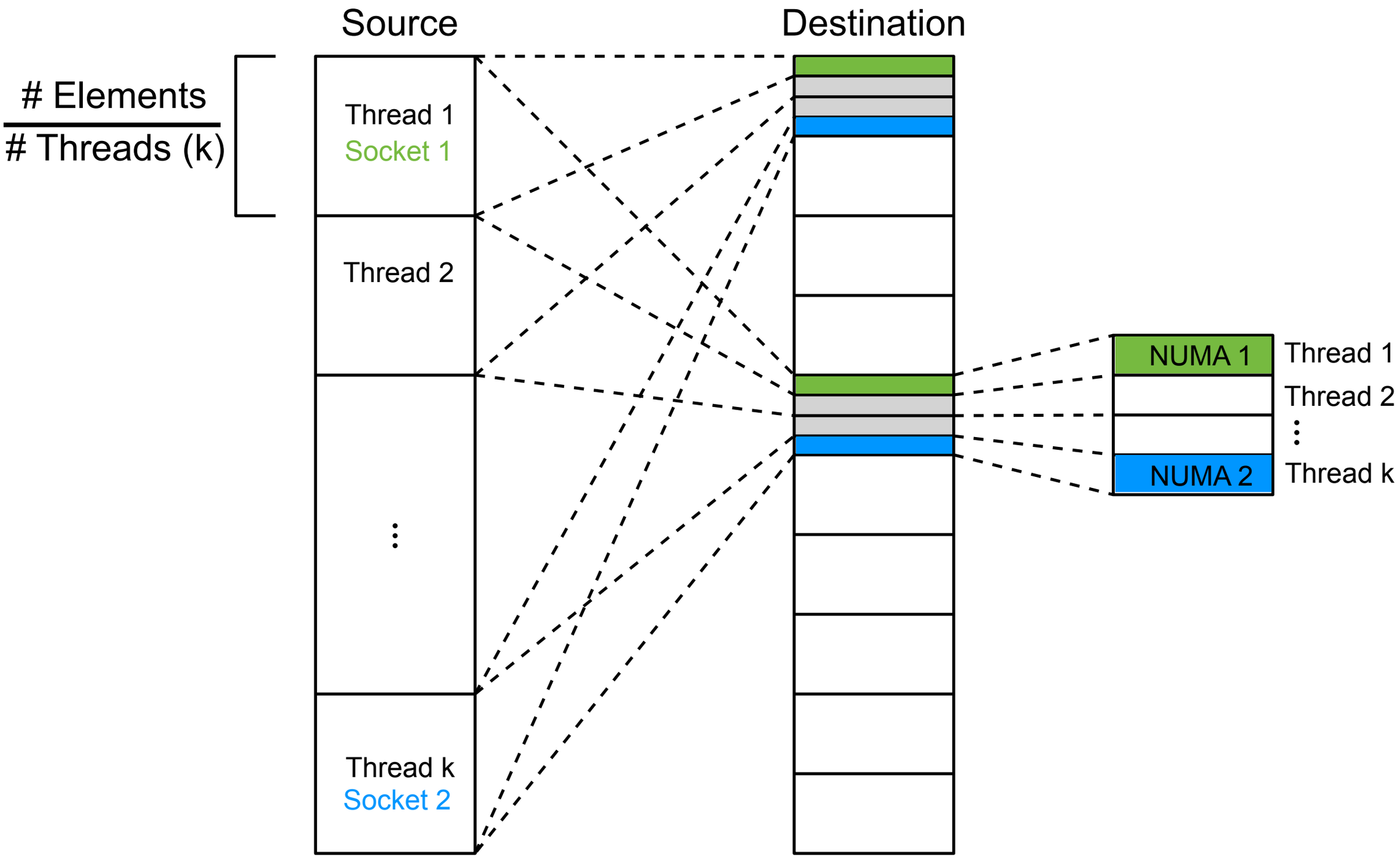
Non-Chunked Algorithms: Analysis (P-RPRS)

Range Partitioning (RP) Phase:



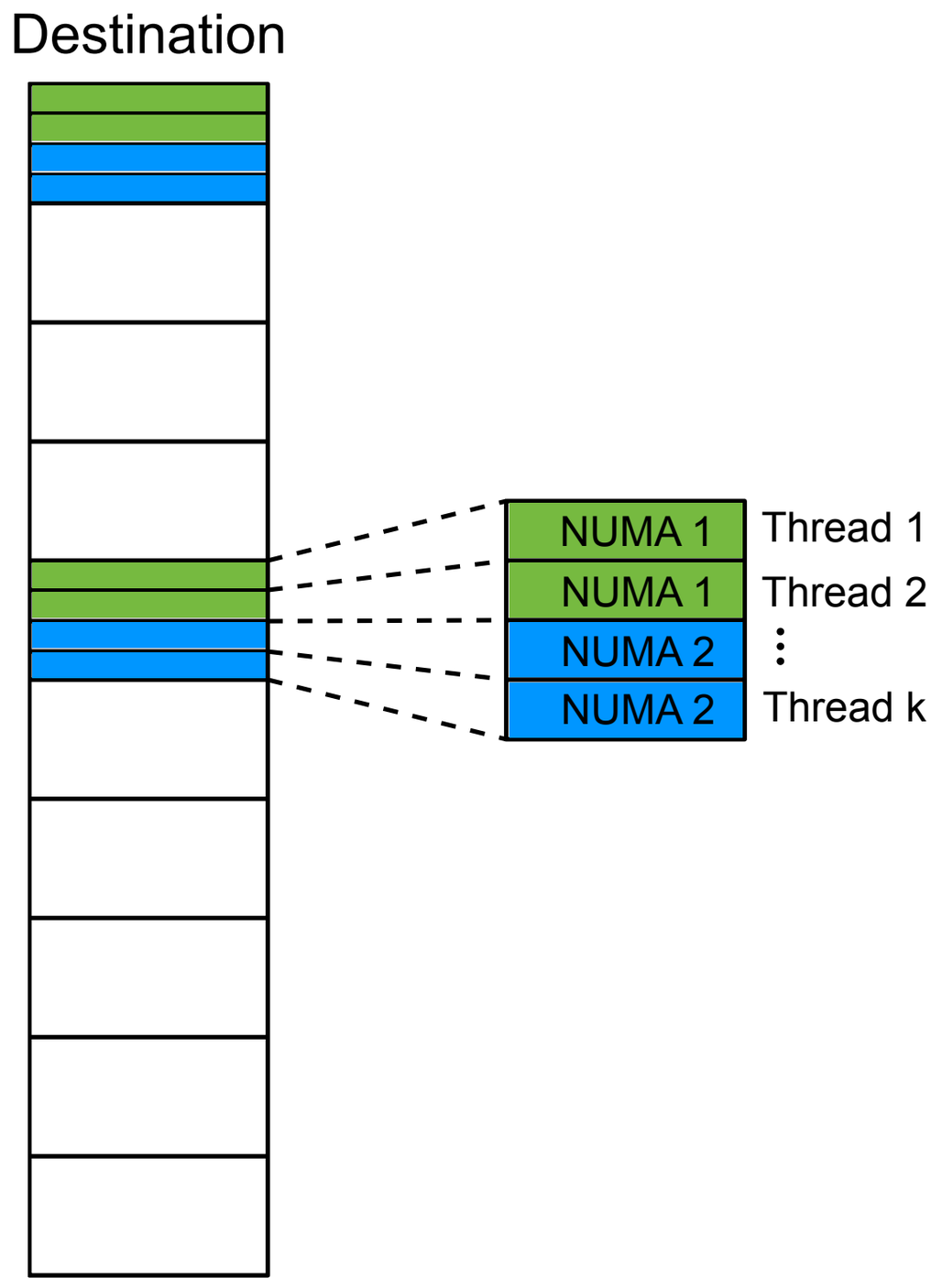
Non-Chunked Algorithms: Analysis (P-RPRS)

Range Partitioning (RP) Phase:



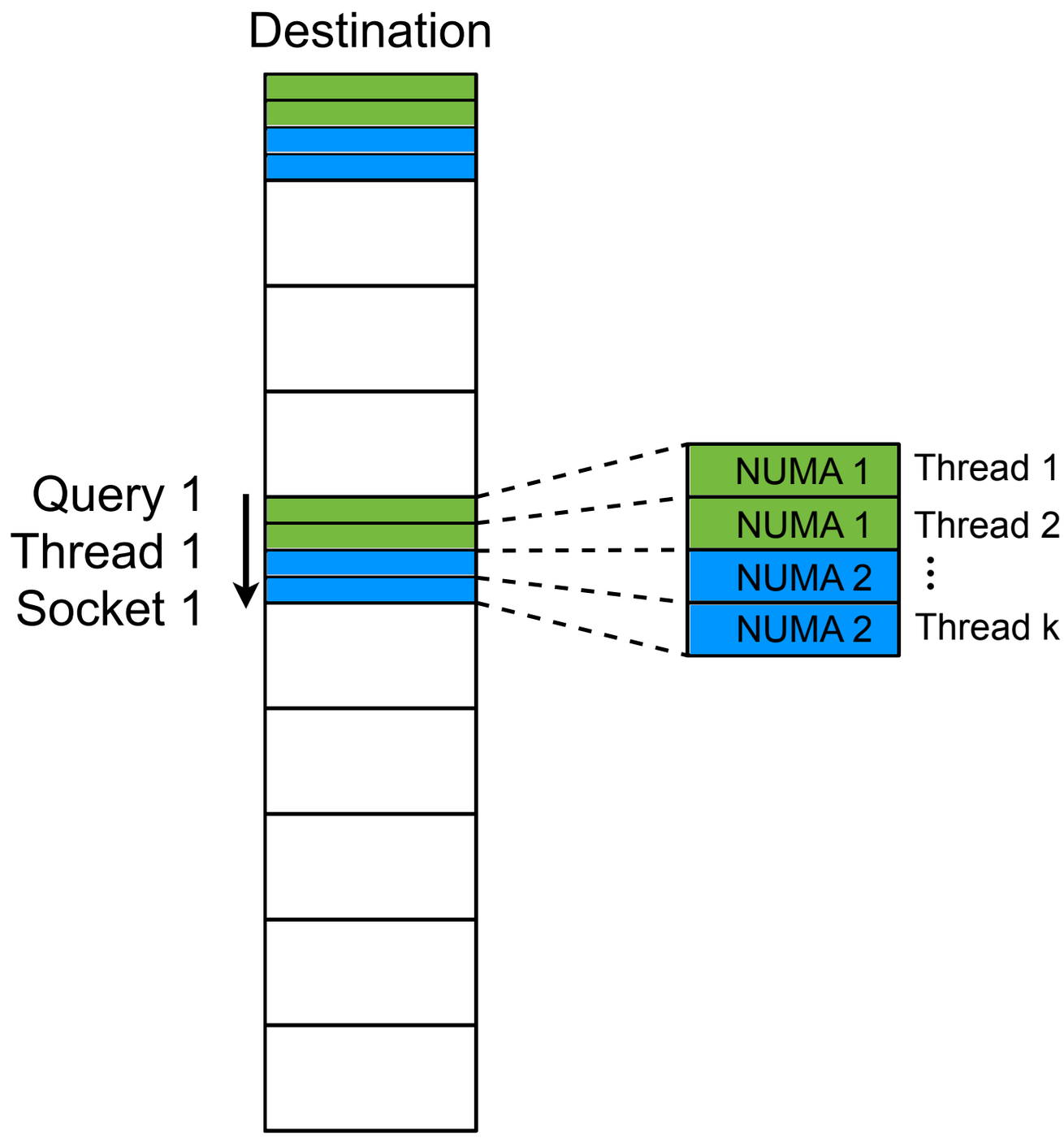
Non-Chunked Algorithms: Analysis (P-RPRS)

Query Phase:



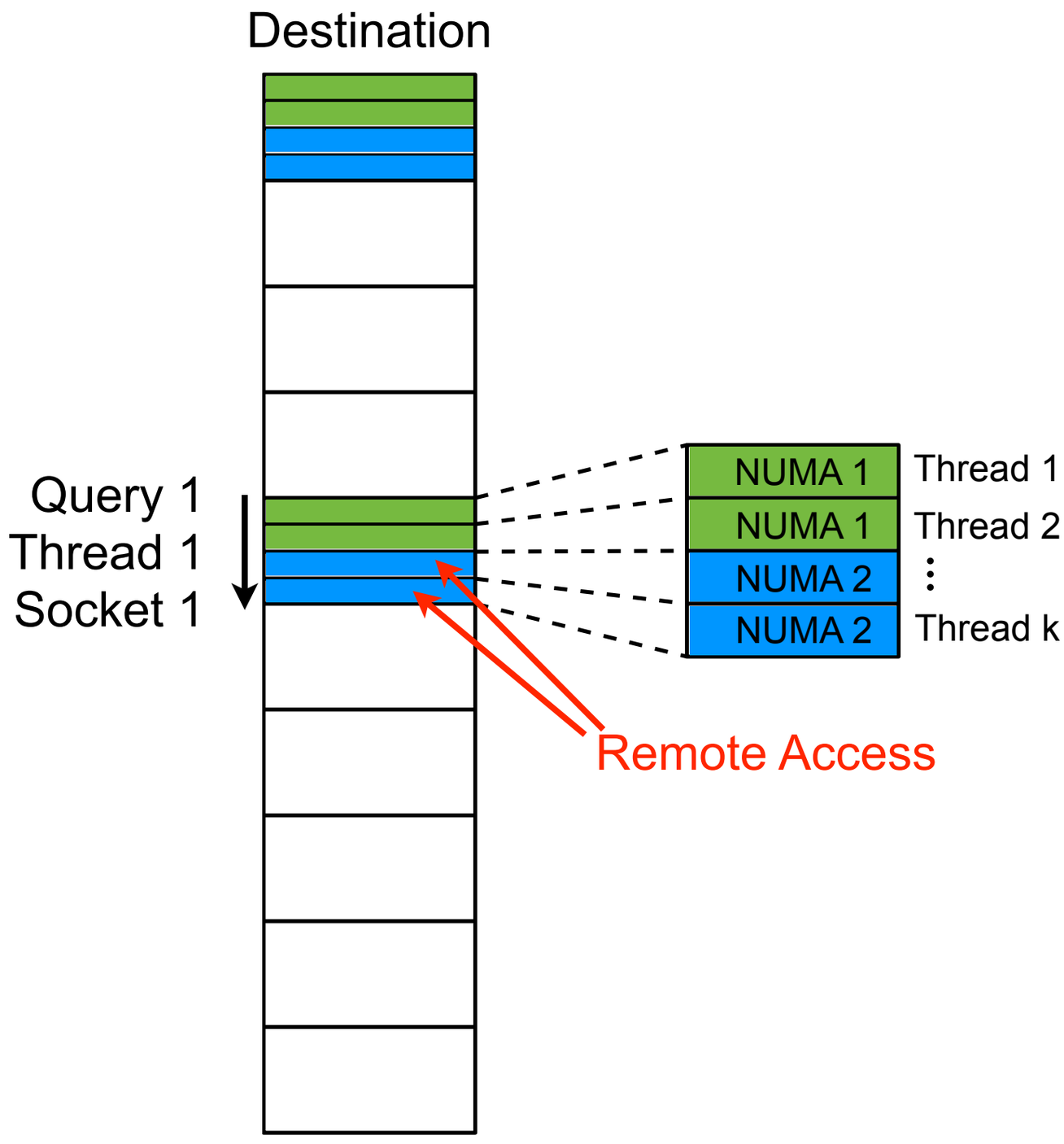
Non-Chunked Algorithms: Analysis (P-RPRS)

Query Phase:

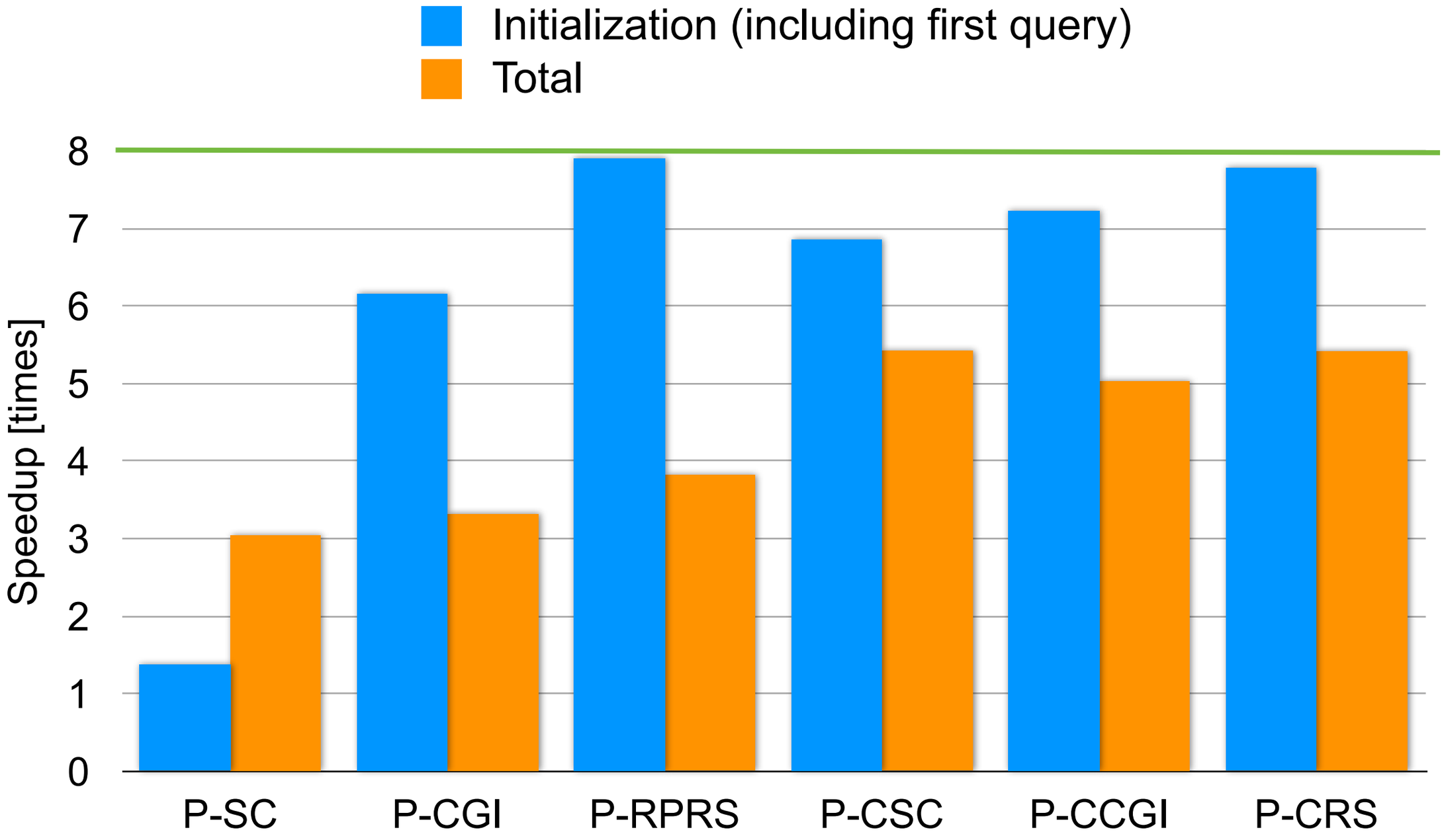


Non-Chunked Algorithms: Analysis (P-RPRS)

Query Phase:

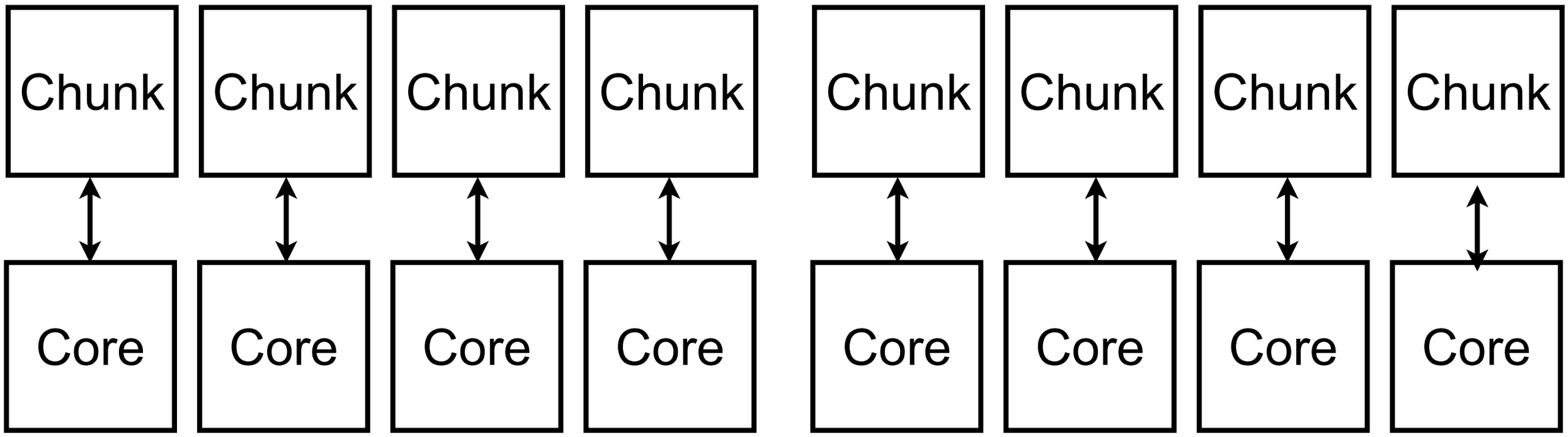


Multi-threaded Results
Factor Speedup from 1 to 8 Threads



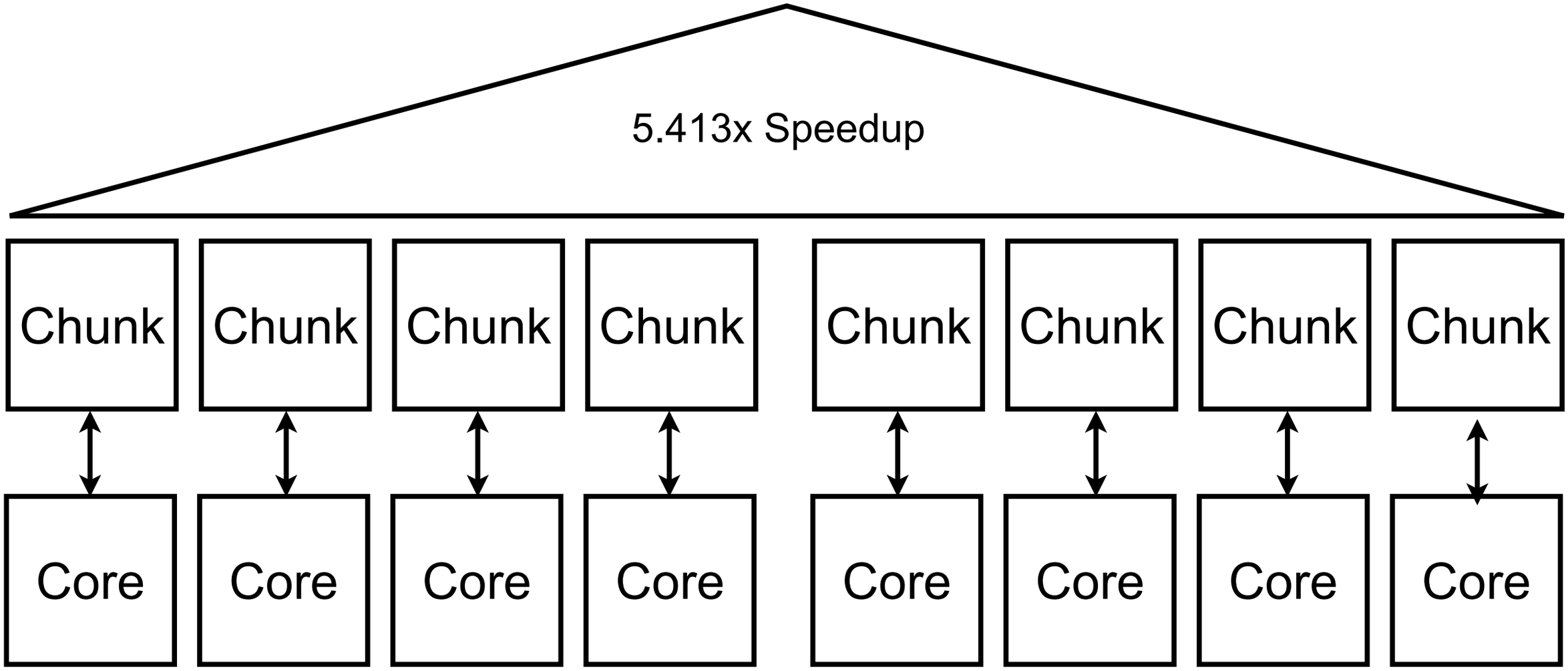
Chunked Algorithms: Analysis (P-CRS)

All chunks are completely independent - 8x Speedup?



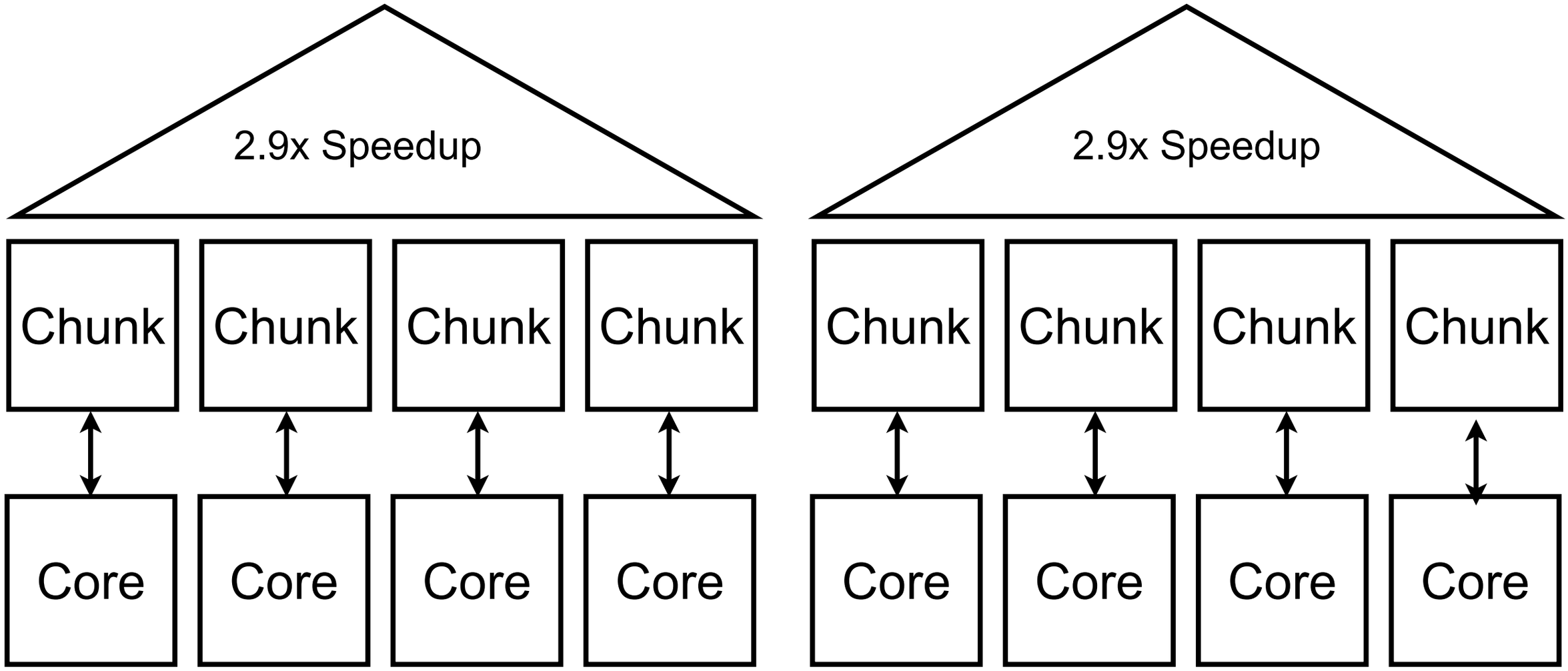
Chunked Algorithms: Analysis (P-CRS)

All chunks are completely independent - 8x Speedup?



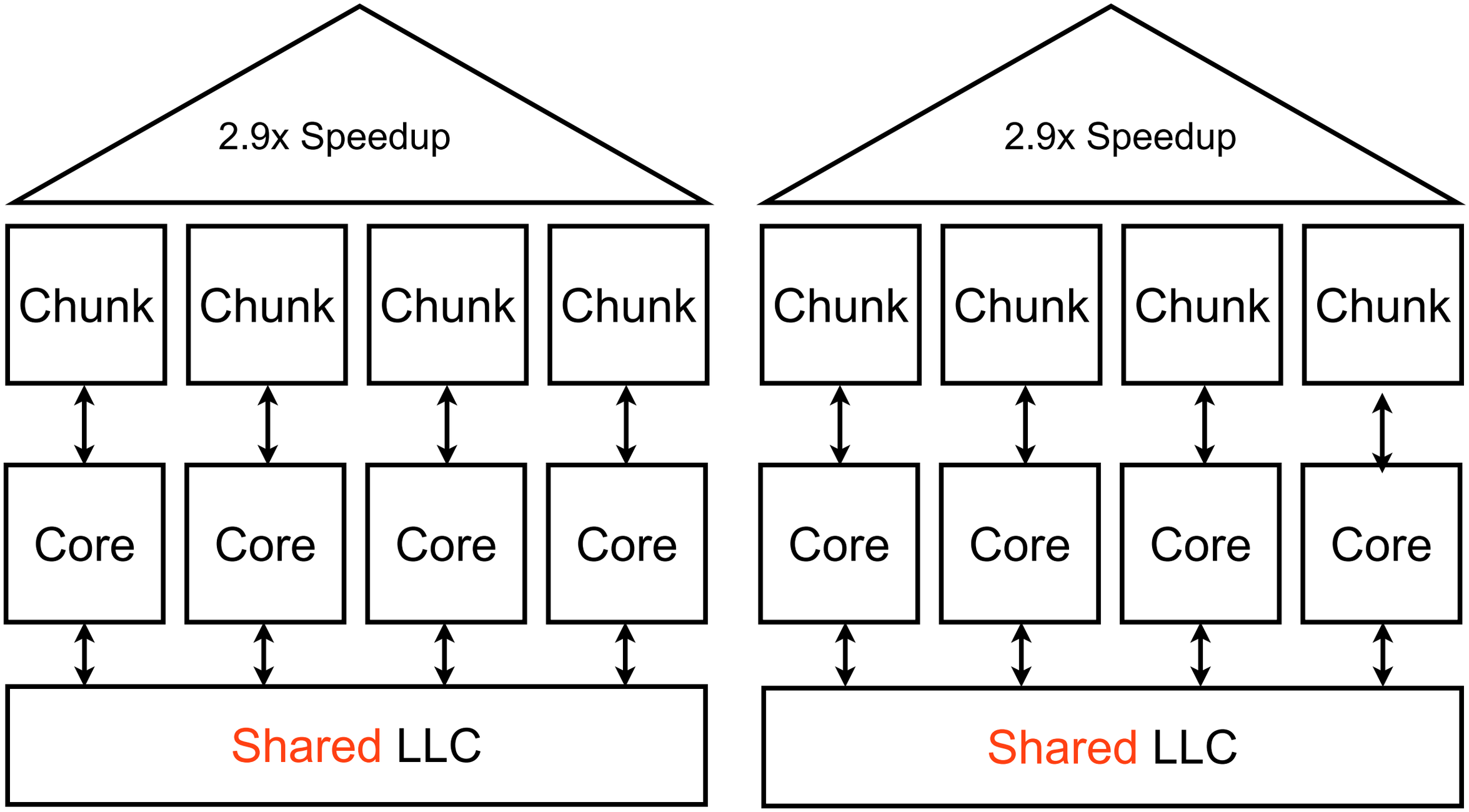
Chunked Algorithms: Analysis (P-CRS)

All chunks are completely independent - 8x Speedup?



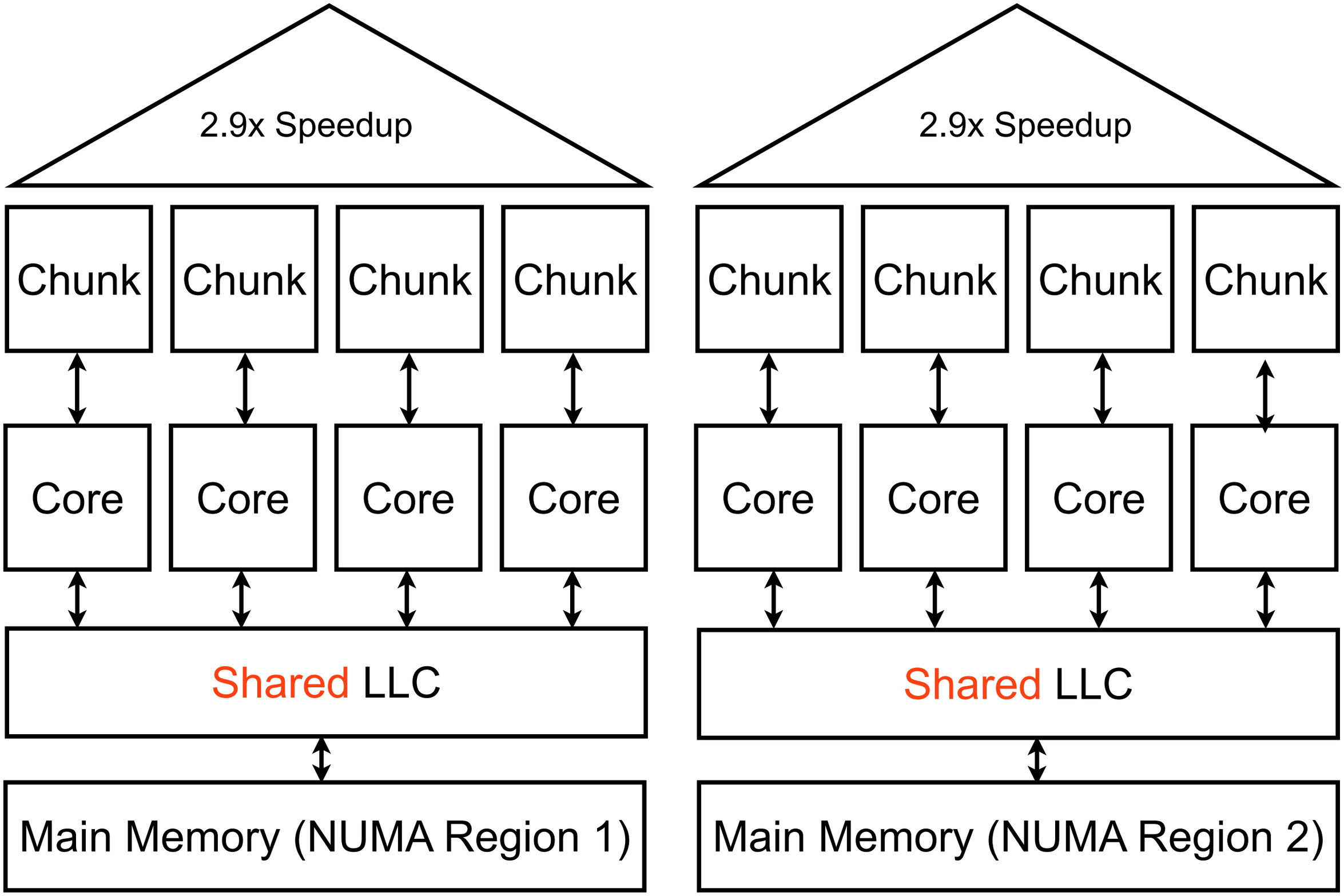
Chunked Algorithms: Analysis (P-CRS)

All chunks are ~~completely~~ independent - 8x Speedup?



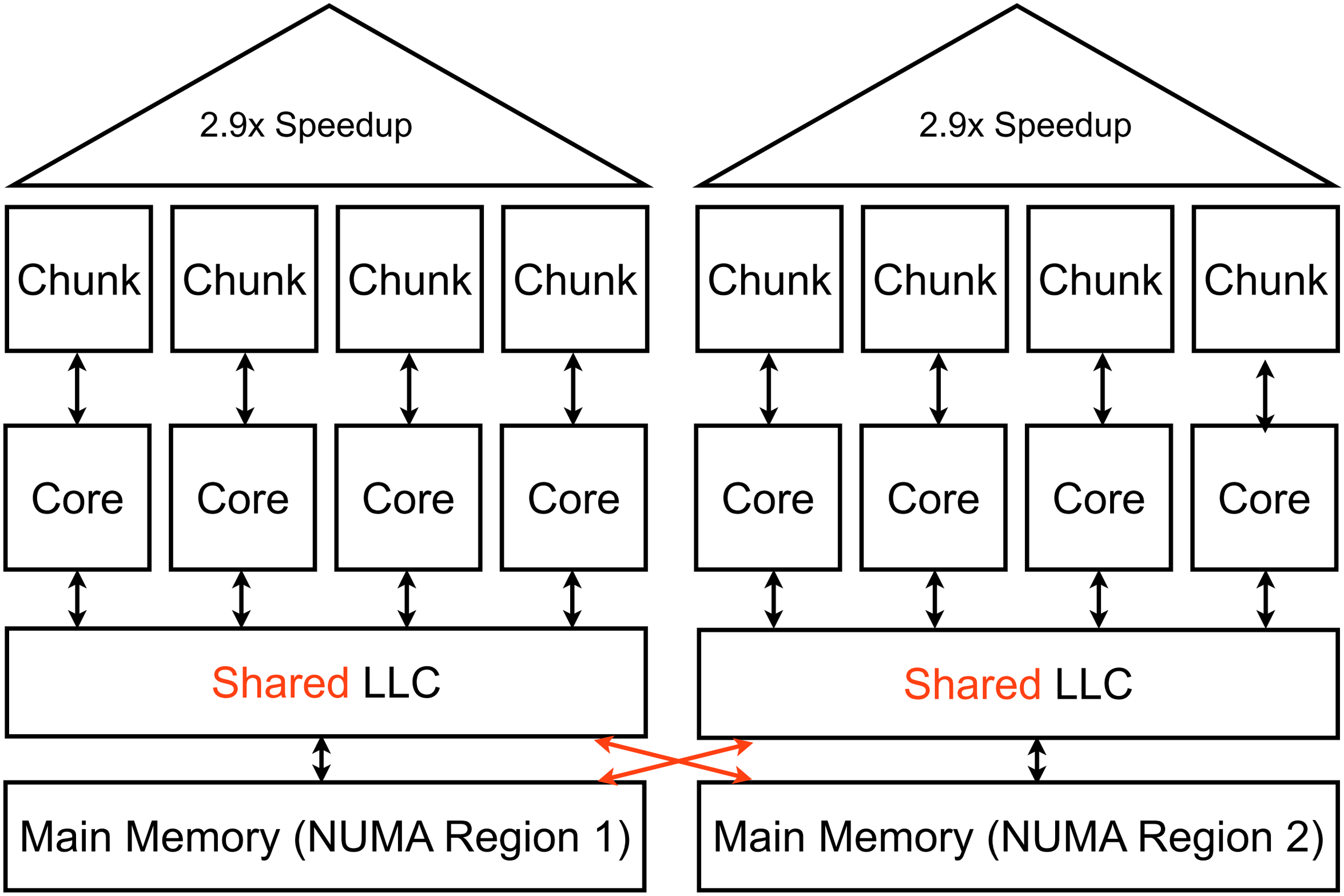
Chunked Algorithms: Analysis (P-CRS)

All chunks are ~~completely~~ independent - 8x Speedup?



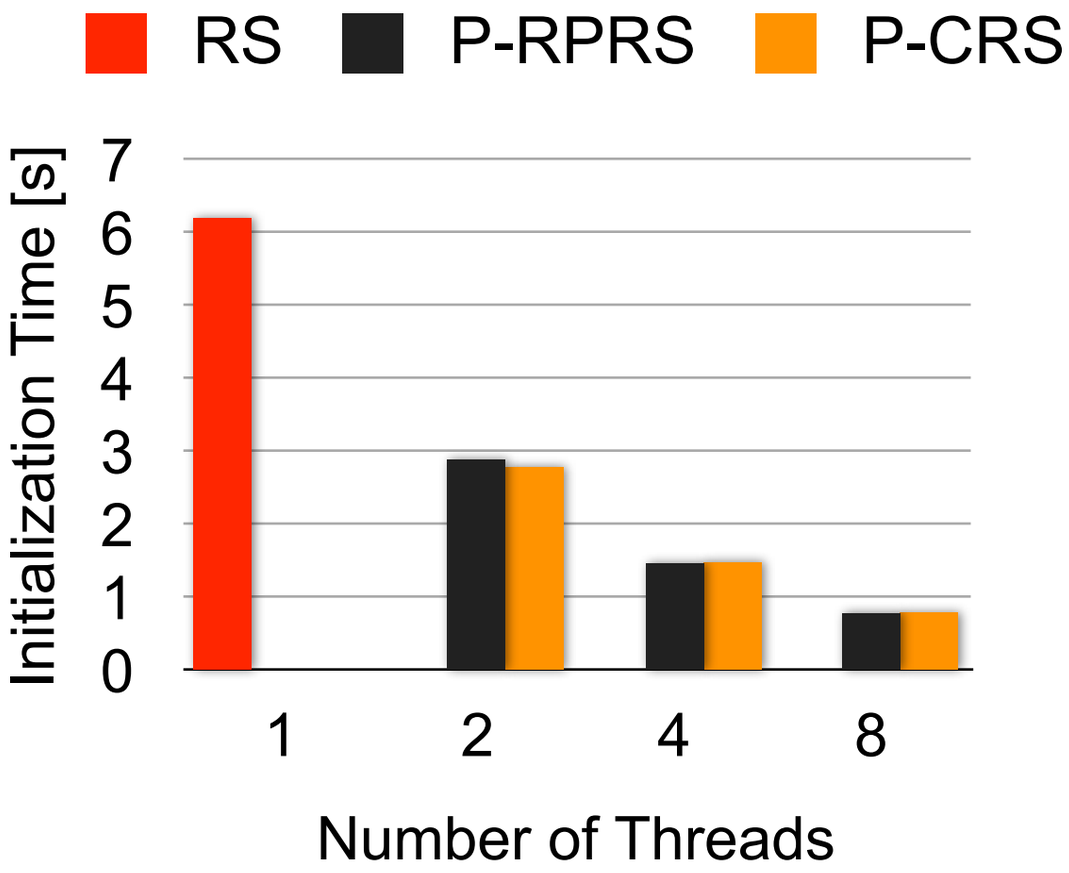
Chunked Algorithms: Analysis (P-CRS)

All chunks are ~~completely~~ independent - 8x Speedup?

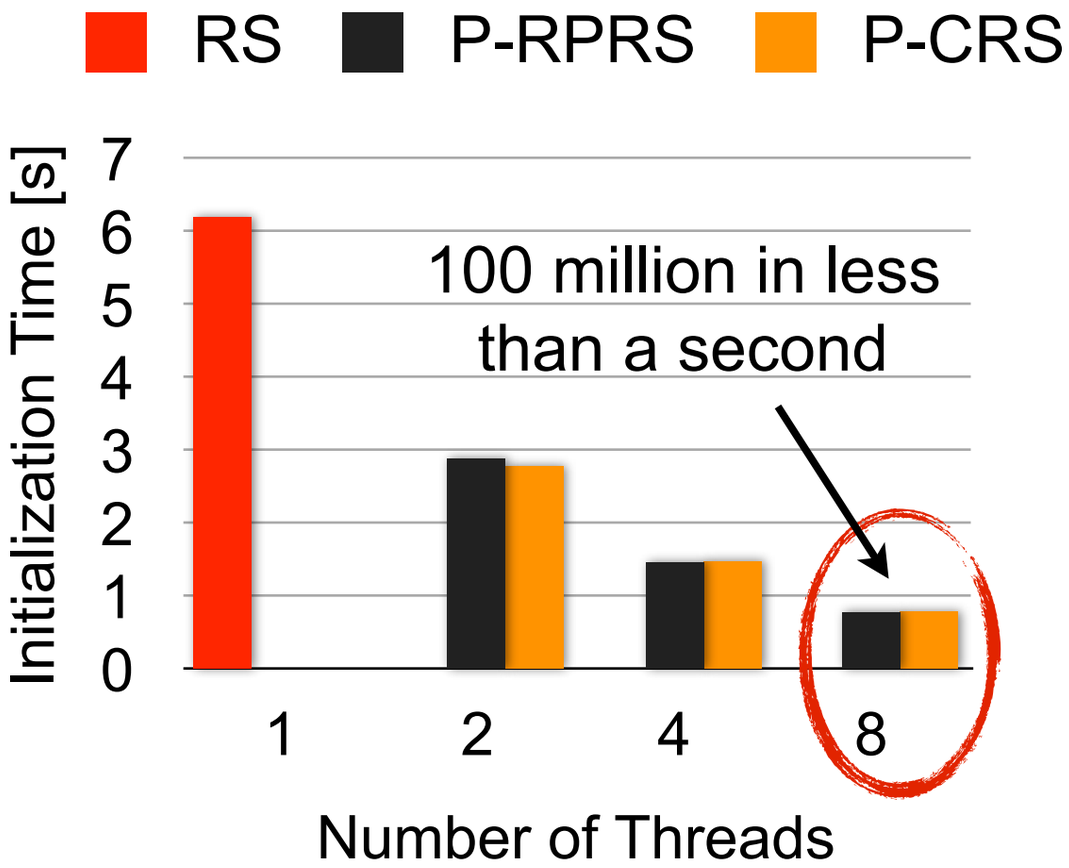


Conclusion

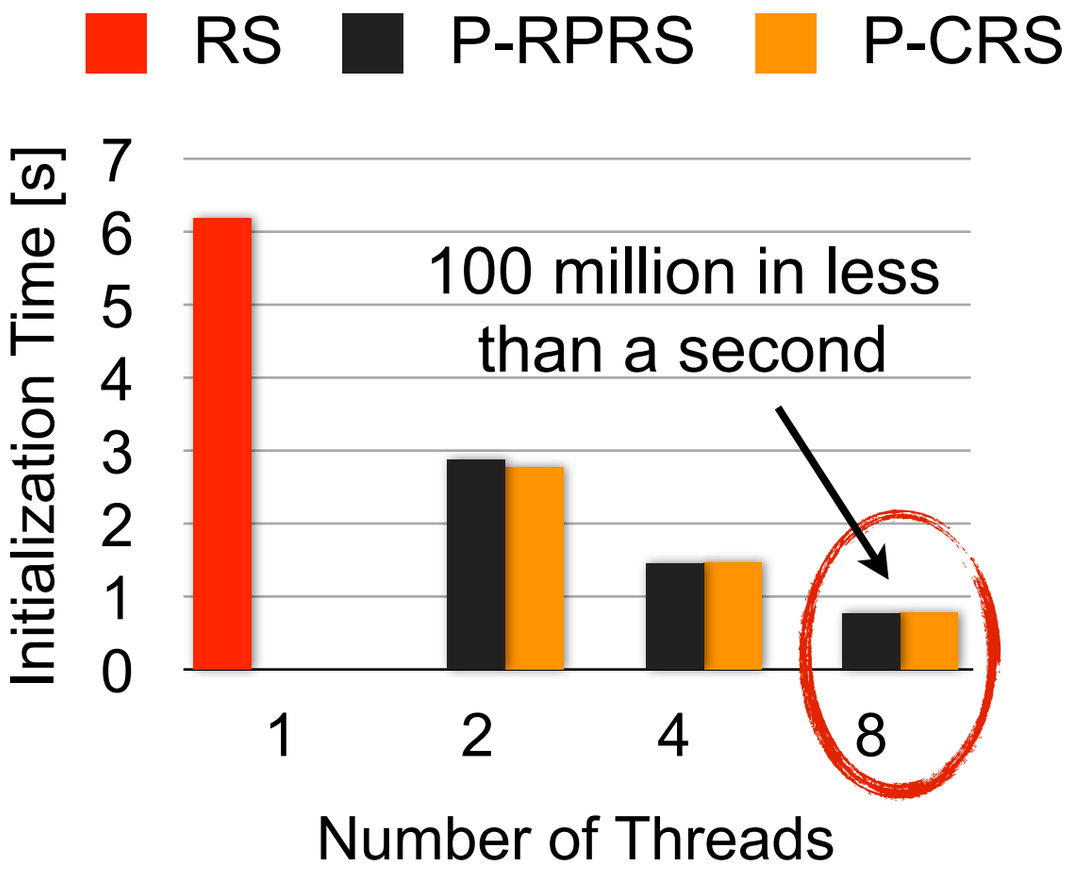
Conclusion

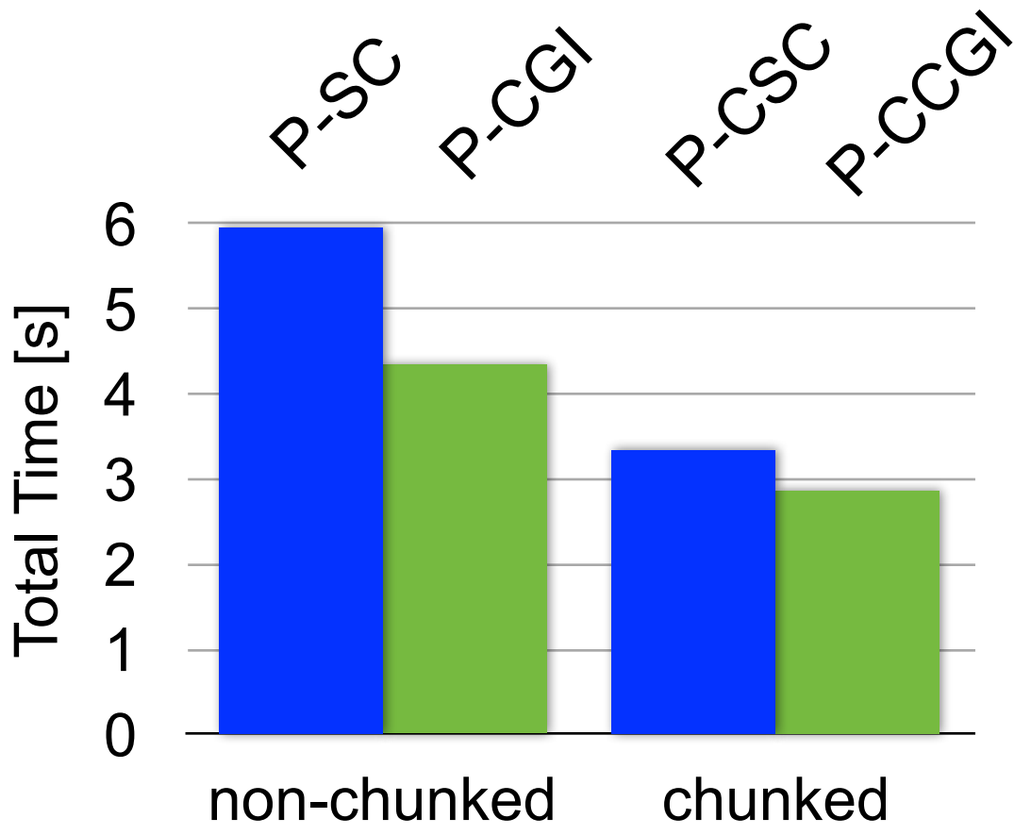

RS
P-RPRS
P-CRS
Initialization Time [s]
0
1
2
3
4
5
6
7
1
2
4
8
Number of Threads

Conclusion

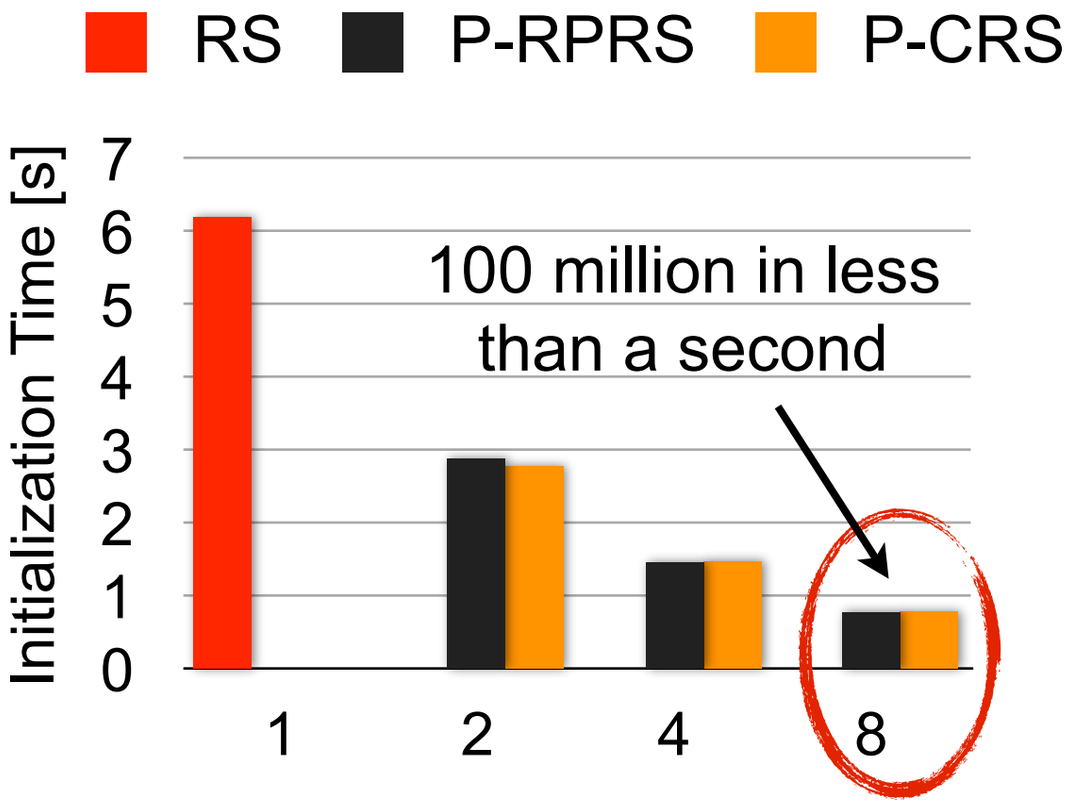

RS
P-RPRS
P-CRS
Initialization Time [s]
0
1
2
3
4
5
6
7
100 million in less than a second
1
2
4
8
Number of Threads

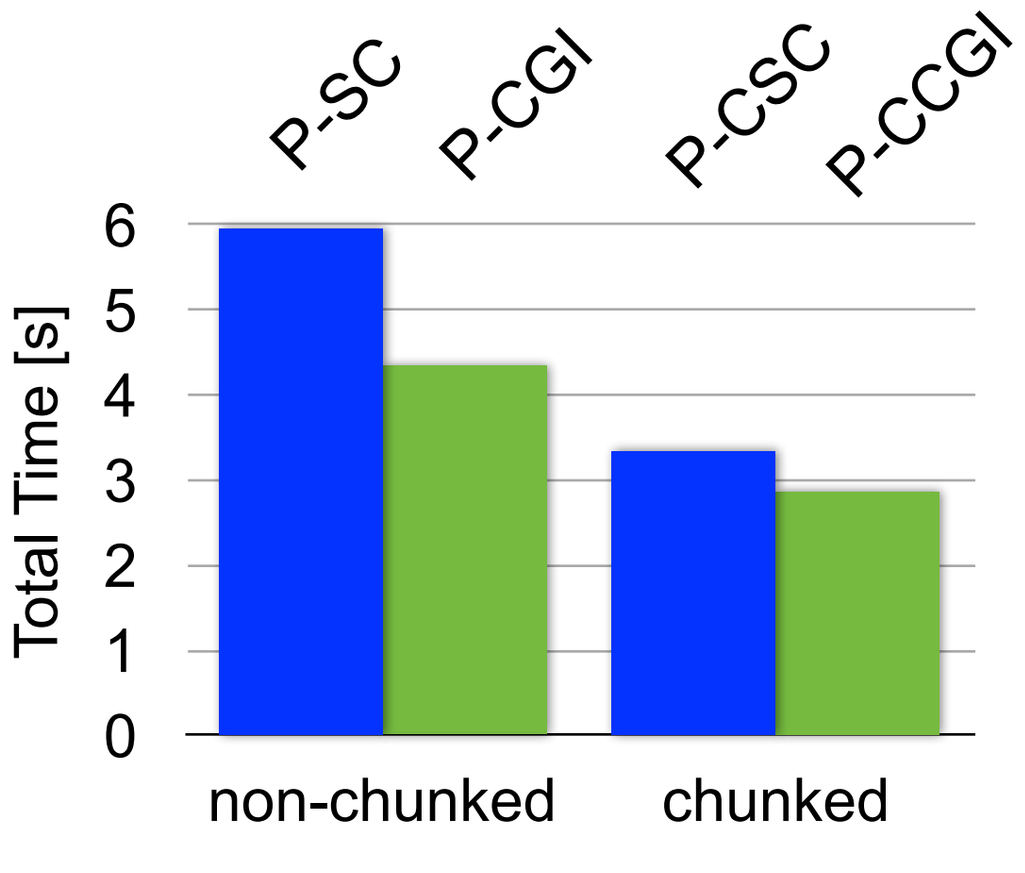
Conclusion

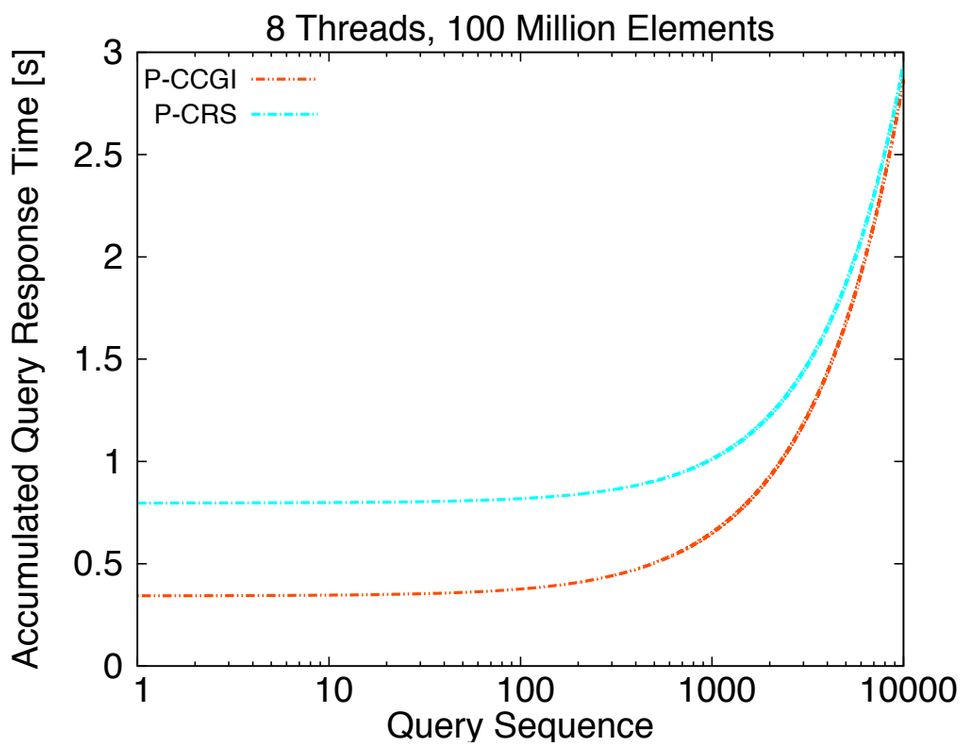

RS
P-RPRS
P-CRS
Initialization Time [s]
0
1
2
3
4
5
6
7
100 million in less than a second
1
2
4
8
Number of Threads


P-SC
P-CGI
P-CSC
P-CCGI
Total Time [s]
0
1
2
3
4
5
6
non-chunked
chunked

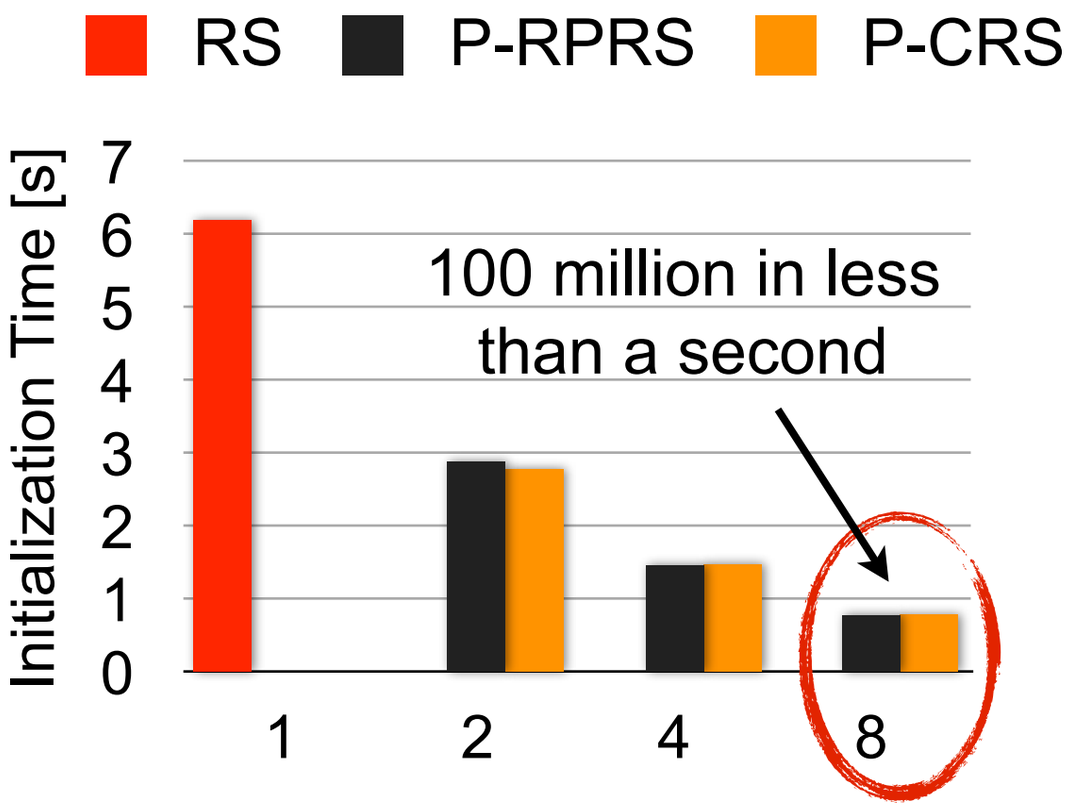
Conclusion

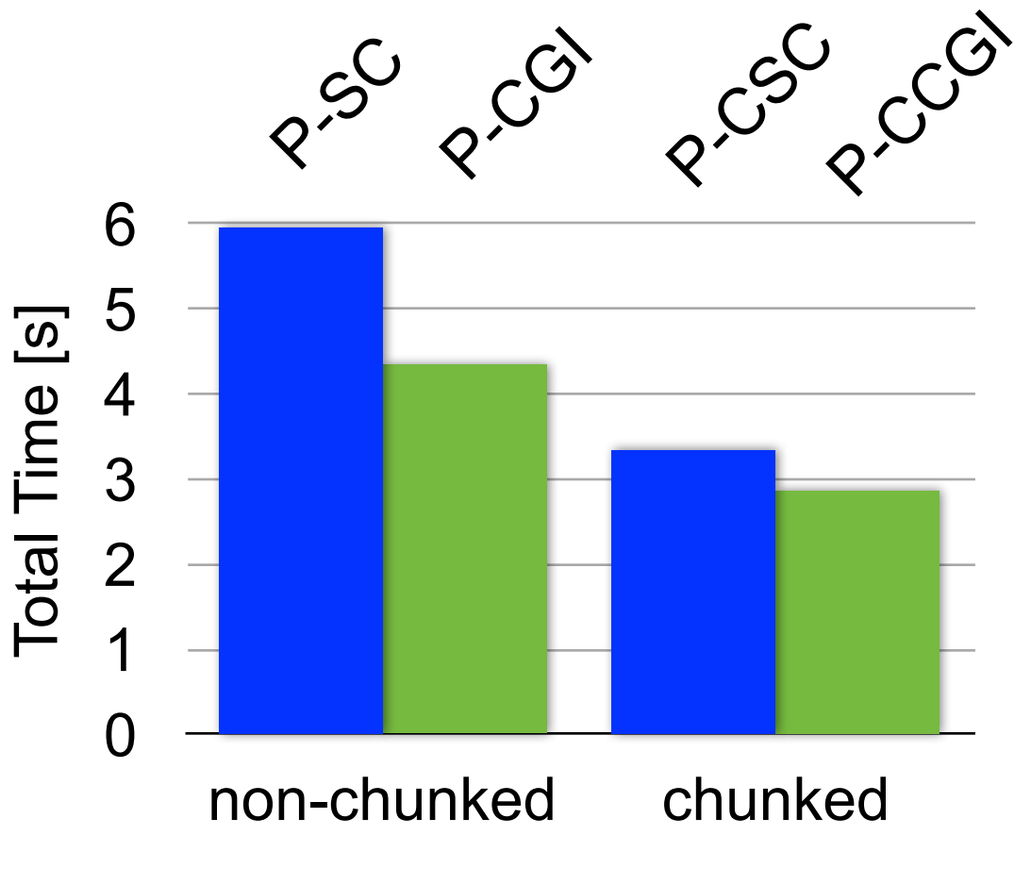

RS
P-RPRS
P-CRS
Initialization Time [s]
0
1
2
3
4
5
6
7
100 million in less than a second
1
2
4
8
Number of Threads

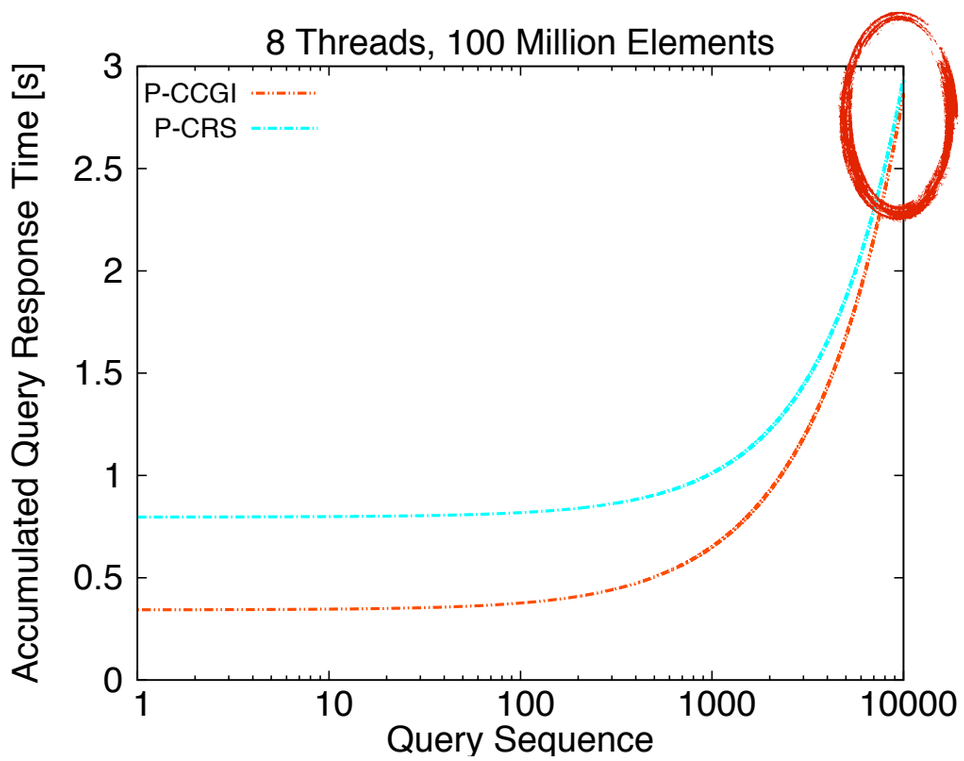

P-SC
P-CGI
P-CSC
P-CCGI
Total Time [s]
0
1
2
3
4
5
6
non-chunked
chunked


8 Threads, 100 Million Elements
P-CCGI
P-CRS
Accumulated Query Response Time [s]
0
0.5
1
1.5
2
2.5
3
1
10
100
1000
10000
Query Sequence

Conclusion

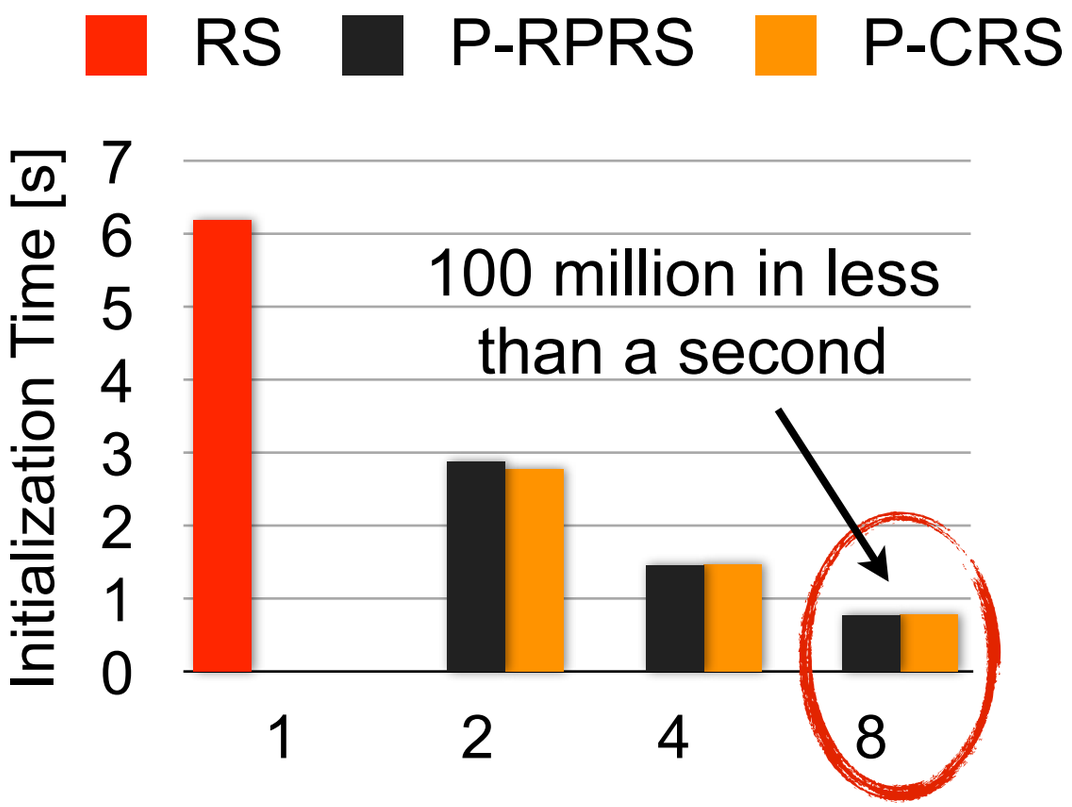


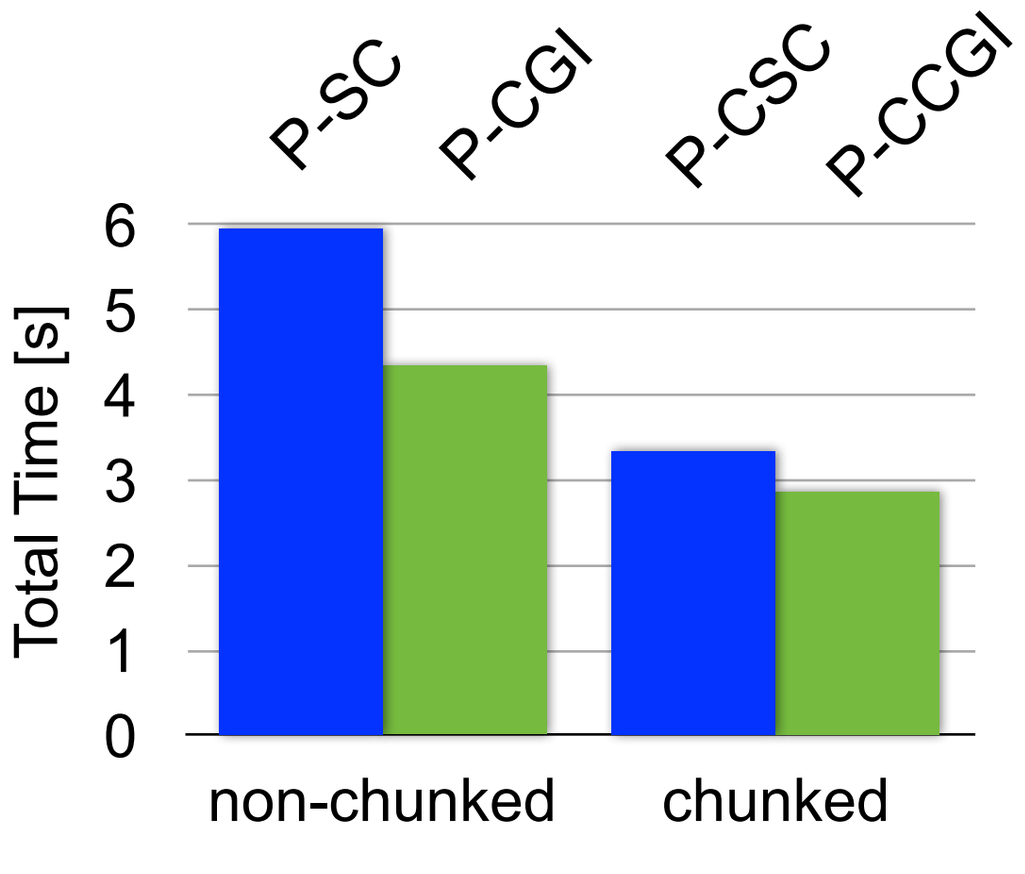




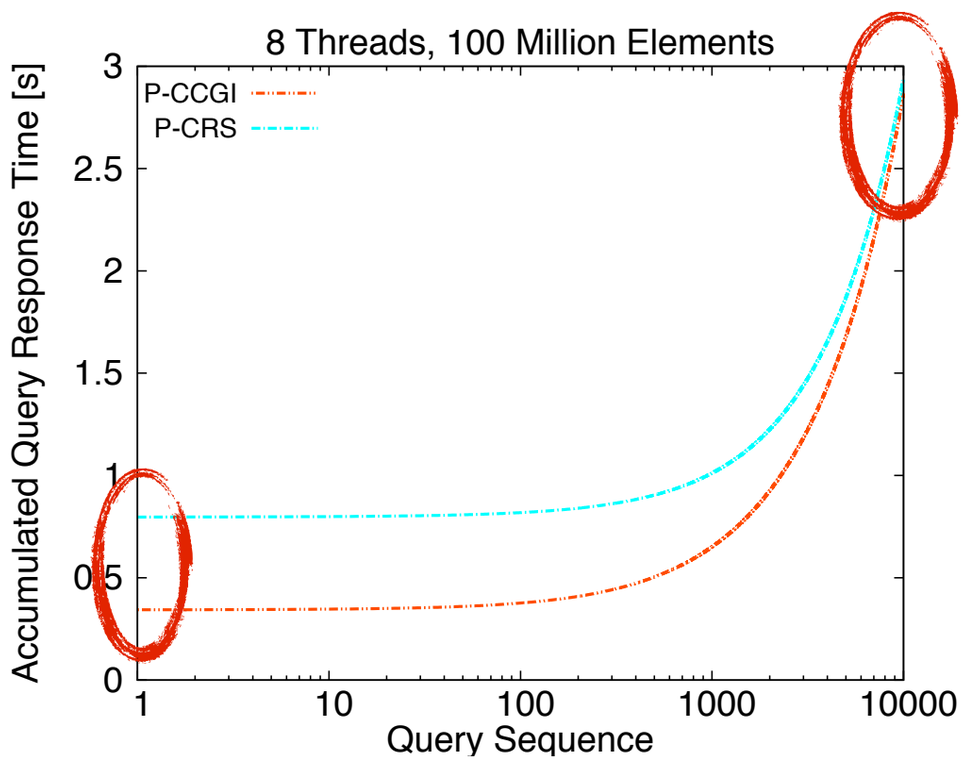
> 10000 queries to win over best cracking

Conclusion





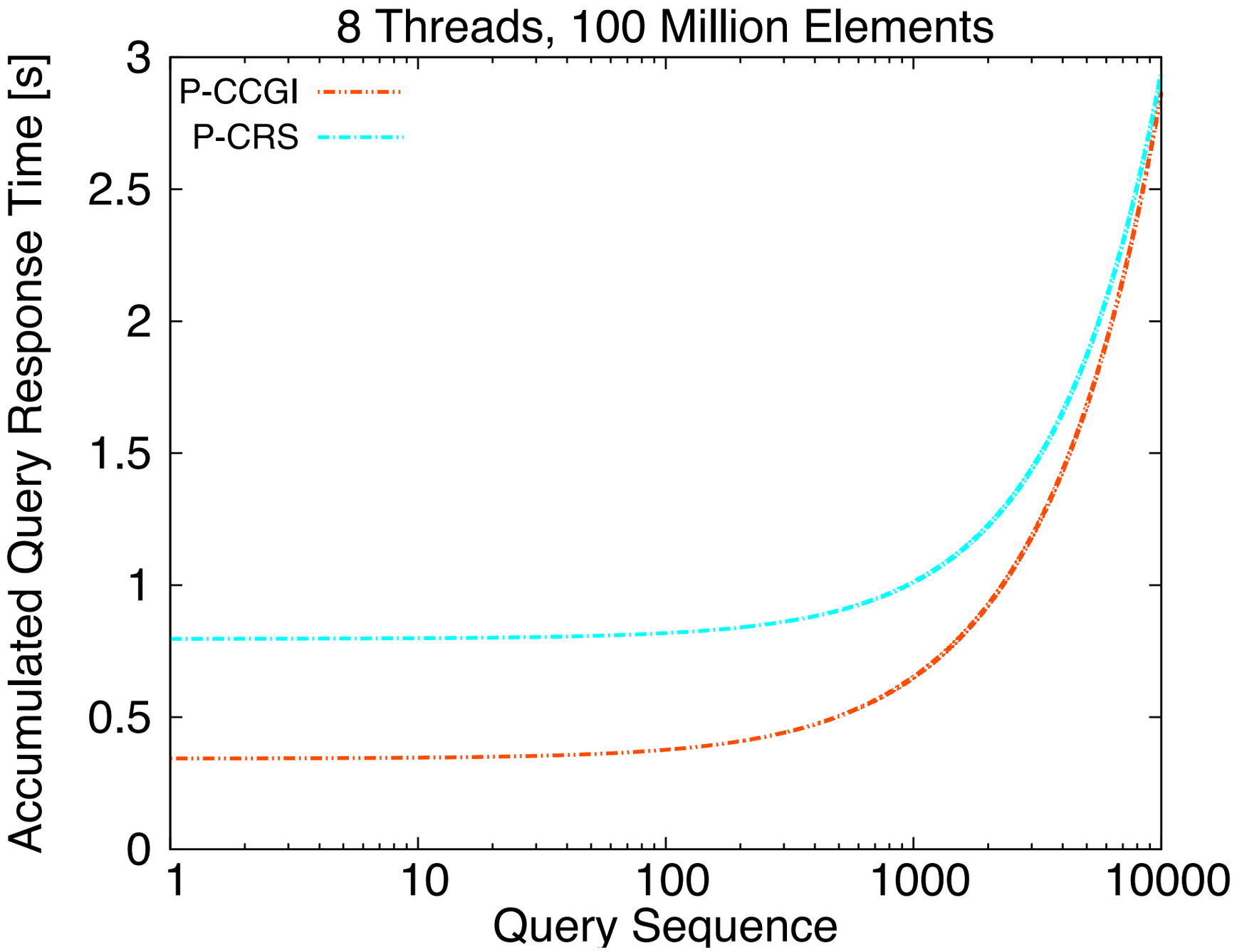
gap decreased
from 5 seconds (1T)
to 0.5 seconds (8T)



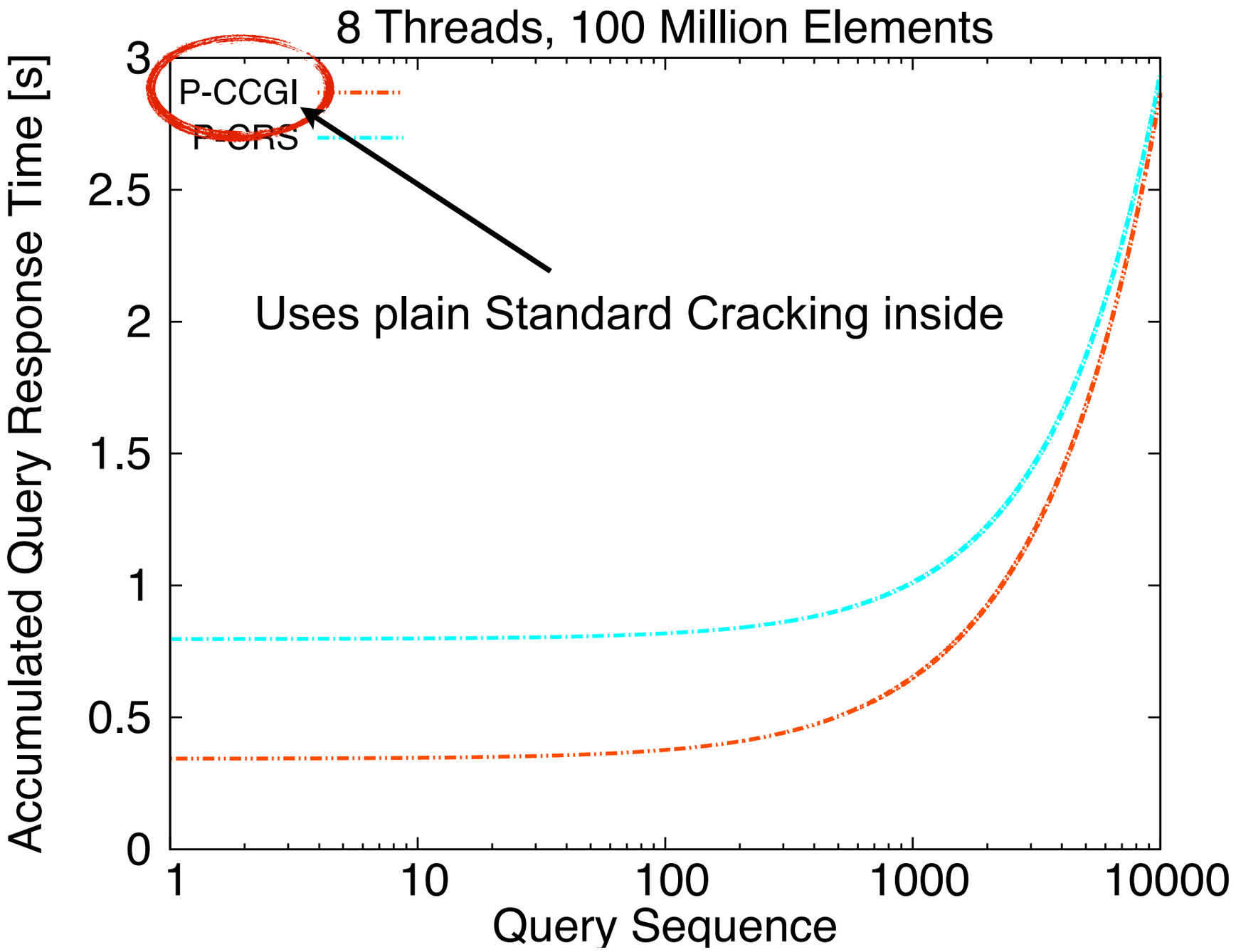
> 10000 queries
to win over
best cracking

Upcoming

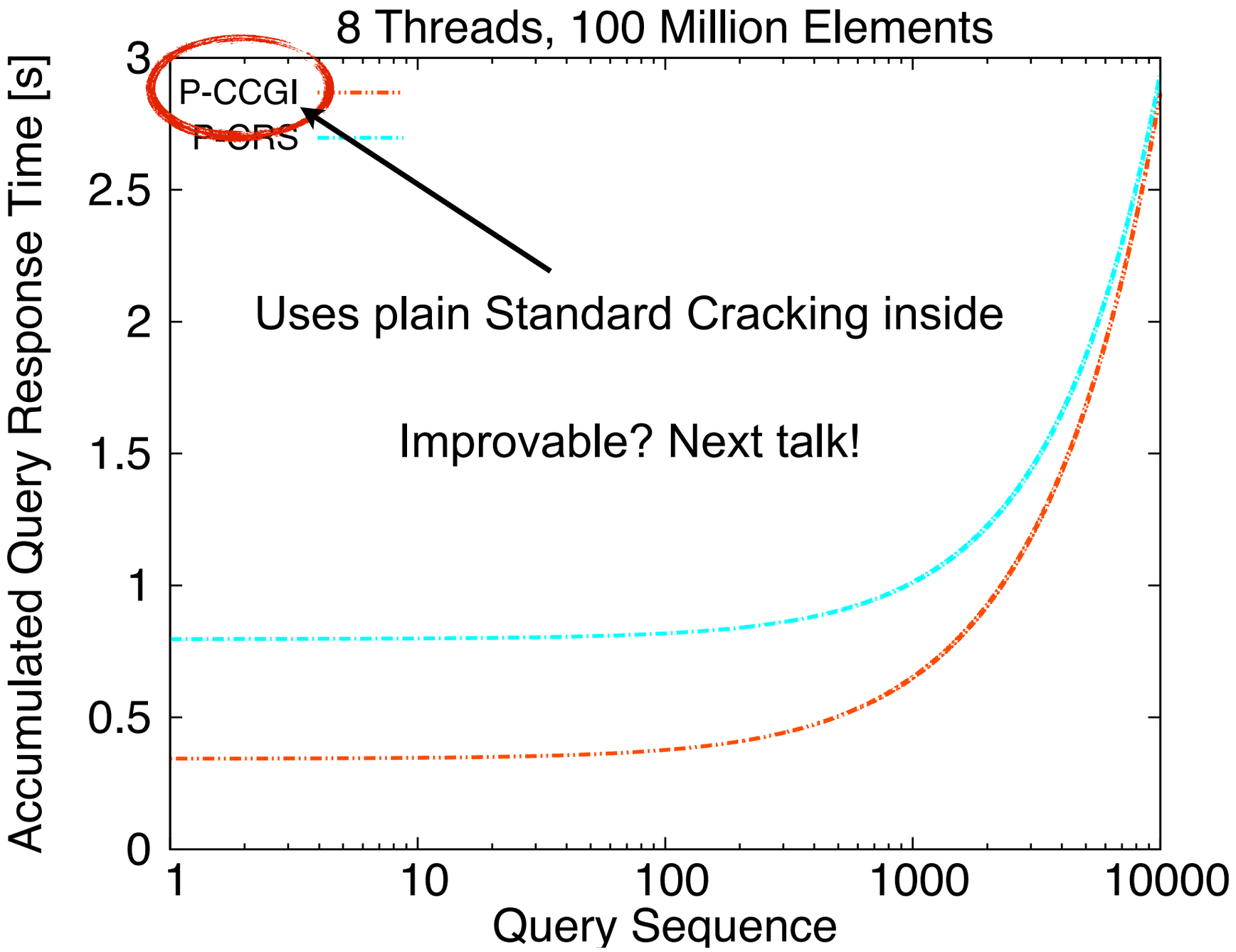
Upcoming


8 Threads, 100 Million Elements
P-CCGI
P-CRS
Accumulated Query Response Time [s]
0
0.5
1
1.5
2
2.5
3
1
10
100
1000
10000
Query Sequence

Upcoming


8 Threads, 100 Million Elements
P-CCGI
P-CRS
Uses plain Standard Cracking inside
Accumulated Query Response Time [s]
Query Sequence
0
0.5
1
1.5
2
2.5
3
1
10
100
1000
10000

Upcoming


8 Threads, 100 Million Elements
P-CCGI
P-CRS
Uses plain Standard Cracking inside
Improvable? Next talk!
Accumulated Query Response Time [s]
Query Sequence
0
0.5
1
1.5
2
2.5
3
1
10
100
1000
10000

Thank you!