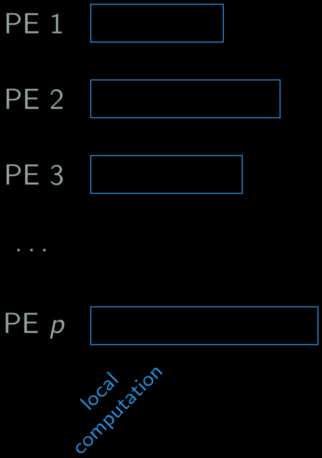
Lightweight Distributed Suffix Array Construction

Johannes Fischer *Florian Kurpicz*

	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	0	1	2	3	4	5	6	7	8	9	10	11	12
	c	c	e	c	e	c	e	d	c	c	e	d	$
	c	e	c	e	c	e	d	c	c	e	d	$	
	e	c	e	c	e	d	c	c	e	d	$		
	c	e	c	e	d	c	c	e	d	$			
	e	c	e	d	c	c	e	d	$				
	c	e	d	c	c	e	d	$					
	e	d	c	c	e	d	$						
	d	c	c	e	d	$							
	c	c	e	d	$								
	c	e	d	$									
	e	d	$										
	d	$											
	$												

	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Bulk Synchronous Parallel Model



PE 1

PE 2

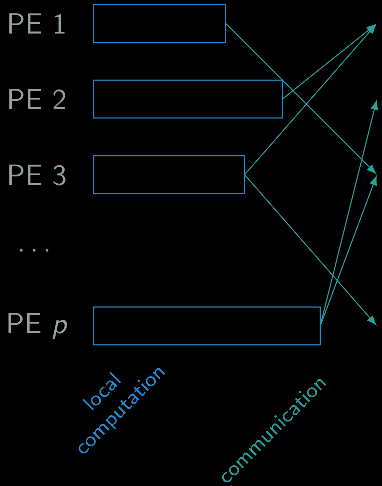
PE 3

. . .

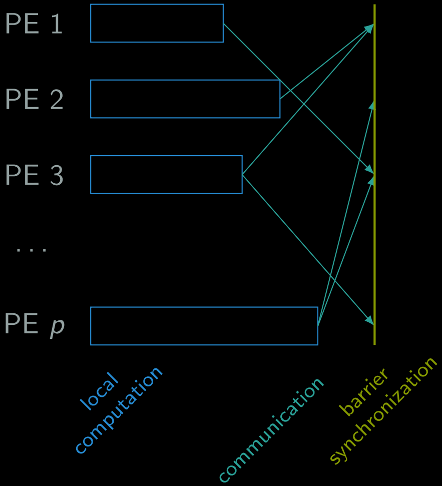
PE p

local computation

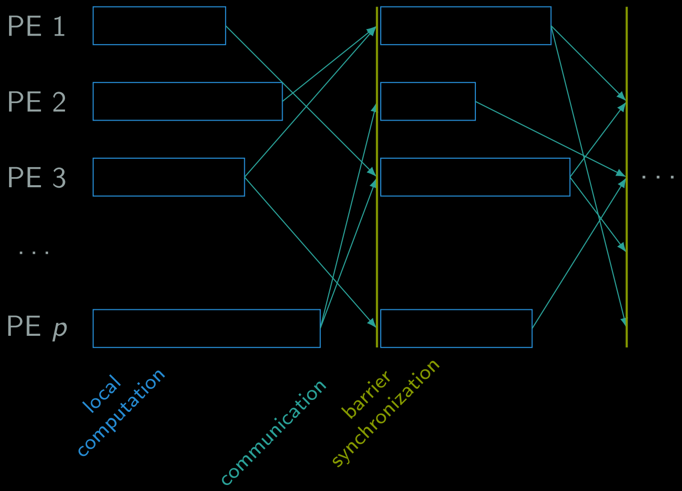
Bulk Synchronous Parallel Model

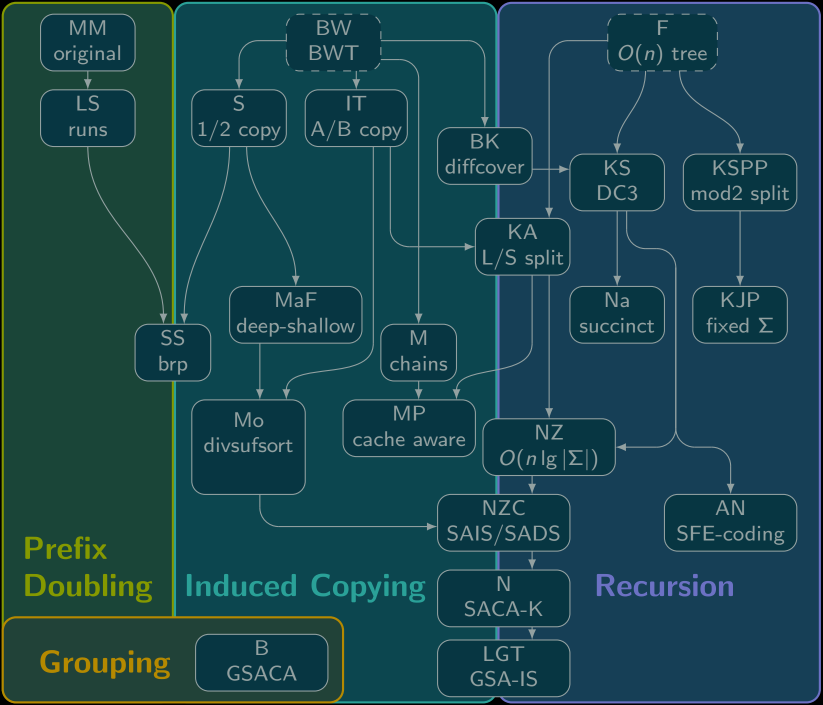


Bulk Synchronous Parallel Model



Bulk Synchronous Parallel Model





MM
original

LS
runs

SS
brp

BW
BWT

S
1/2 copy

IT
A/B copy

BK
diffcover

MaF
deep-shallow

M
chains

Mo
divsufsort

MP
cache aware

F
$O(n)$ tree

KS
DC3

KSPP
mod2 split

KA
L/S split

Na
succinct

KJP
fixed Σ

NZ
$O(n \lg |\Sigma|)$

NZC
SAIS/SADS

AN
SFE-coding

N
SACA-K

LGT
GSA-IS

B
GSACA

Prefix Doubling

Induced Copying

Recursion

Grouping

[PST07]
[BFO13]
updated

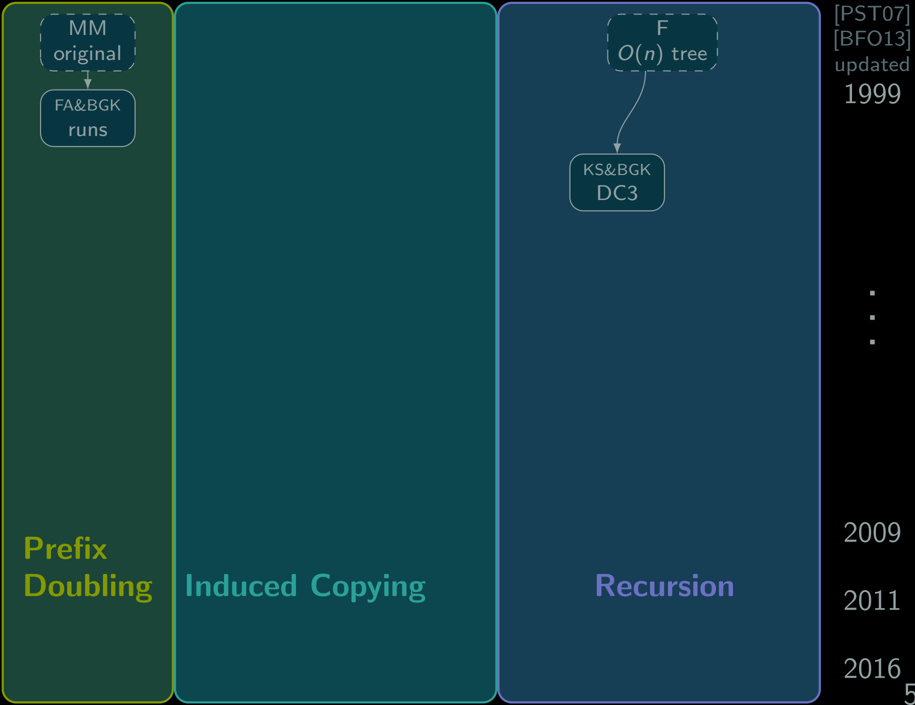
1999

⋮

2009

2011

2016



[PST07]
[BFO13]
updated

1999

MM
original

FA&BGK
runs

F
$O(n)$ tree

KS&BGK
DC3

⋮

2009

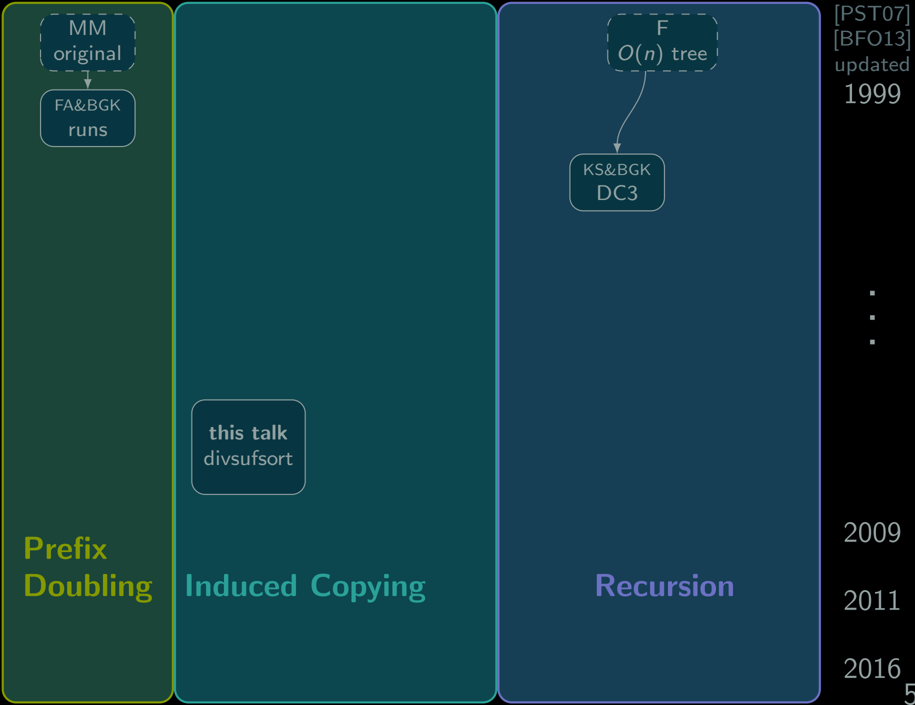
2011

2016

Prefix Doubling

Induced Copying

Recursion



MM
original

FA&BGK
runs

this talk
divsufsort

F
$O(n)$ tree

KS&BGK
DC3

Prefix
Doubling

Induced Copying

Recursion

[PST07]
[BFO13]
updated

1999

⋮

2009

2011

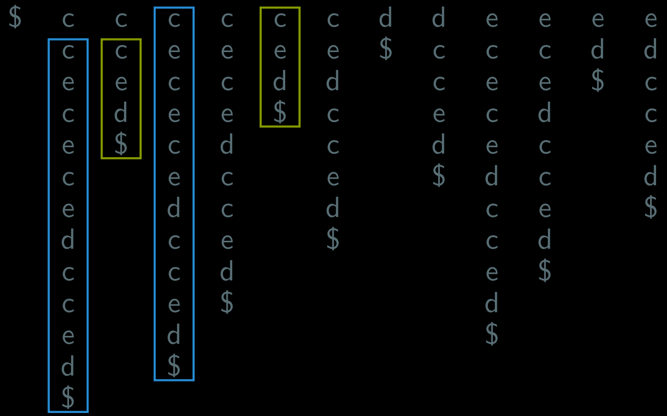
2016

What is Induced Copying?

	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6



$

ccececedcced$

cced$

cececedcced$

cecedcced$

ced$

cedcced$

d$

dcced$

ececedcced$

ecedcced$

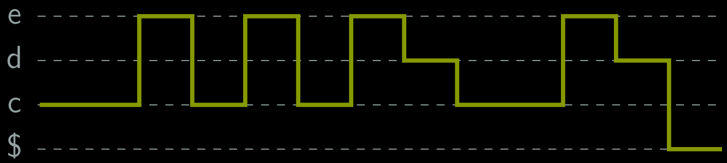
ed$

edcced$

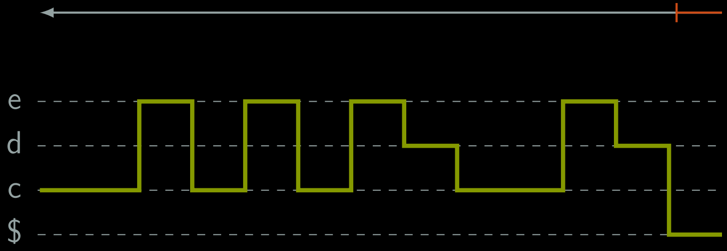
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedccedd$	cced$	cecececdcced$	cecedcced$	ced$	cedccedd$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Identify suffixes that must be sorted

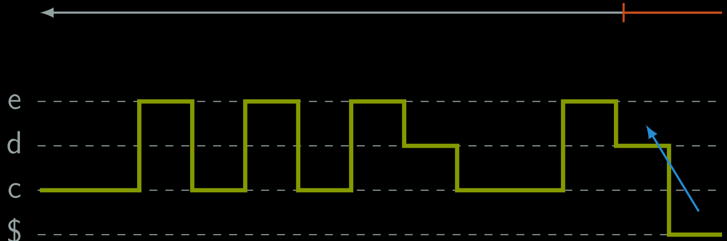
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



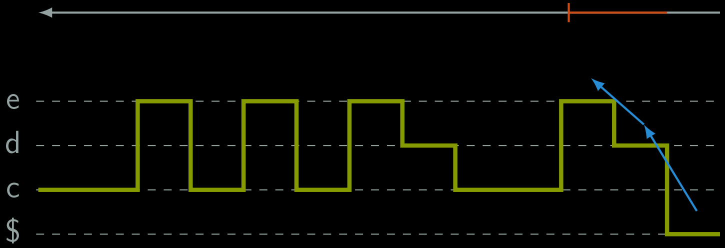
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



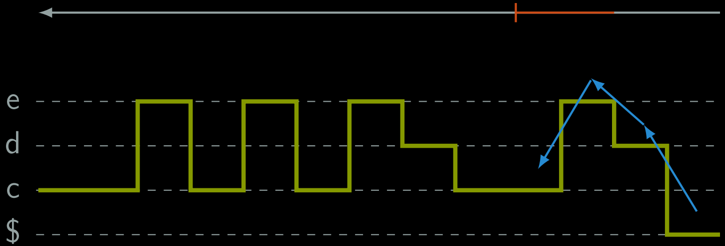
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



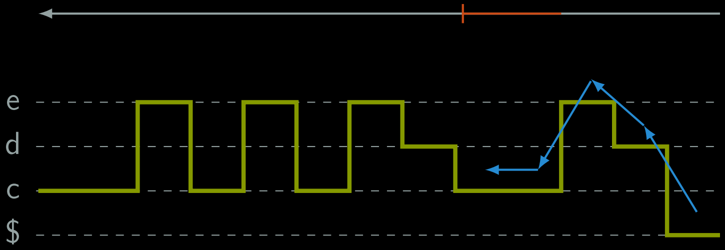
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



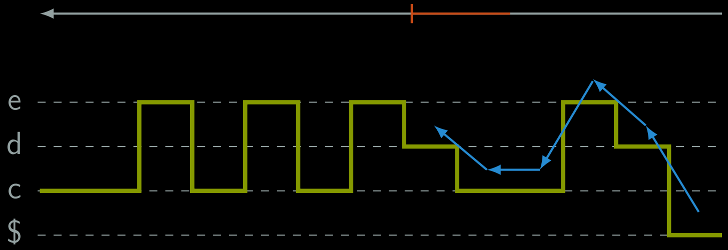
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



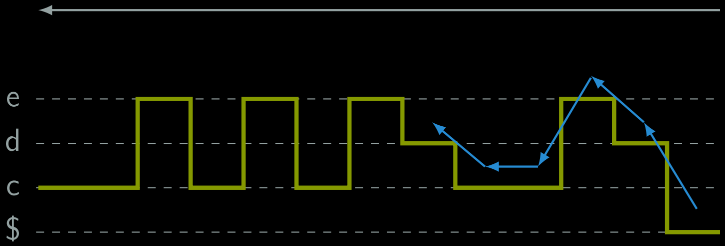
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



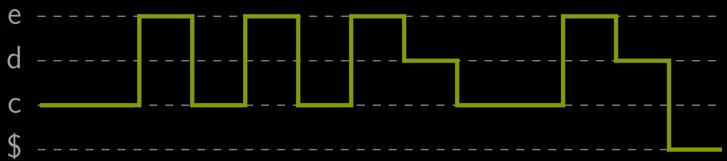
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



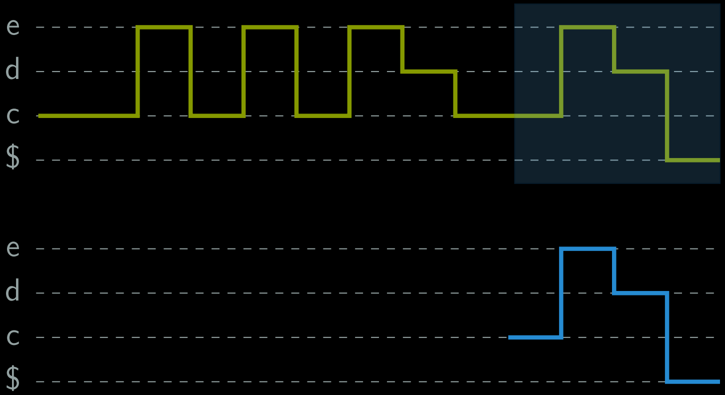
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



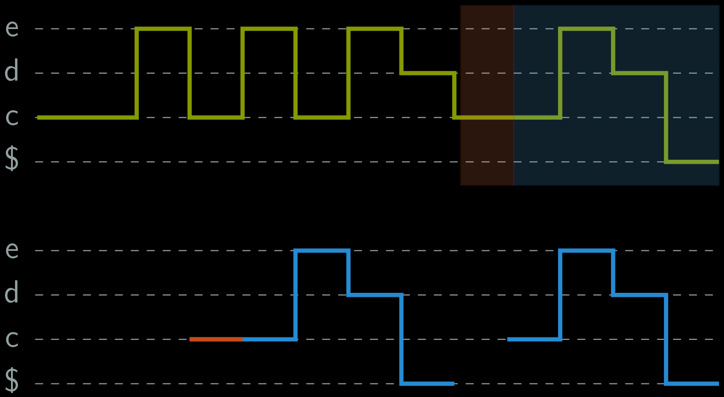
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



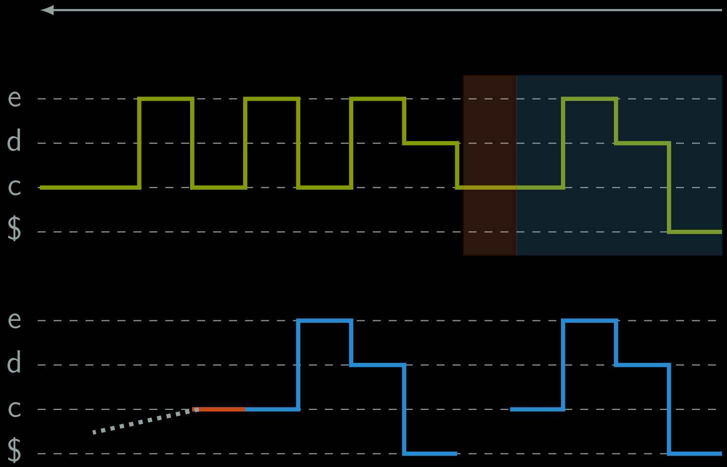
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



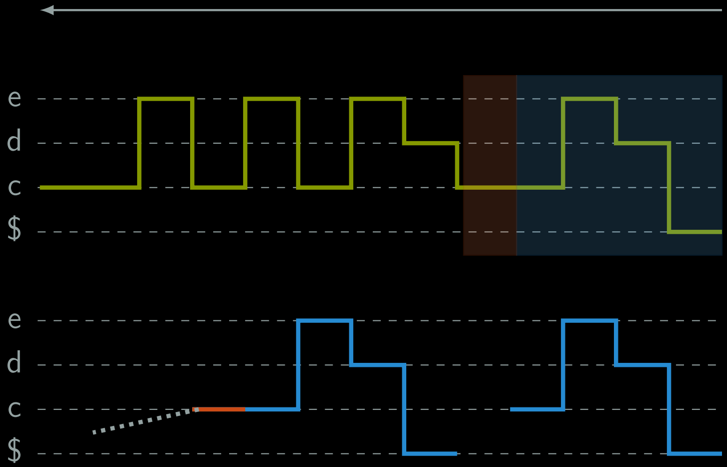
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$

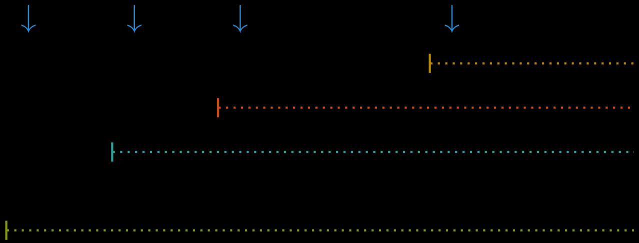


0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$

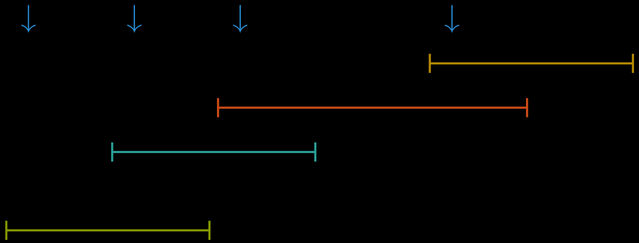


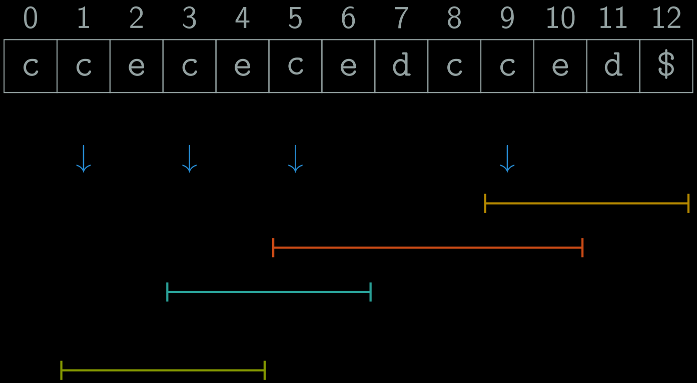
We only need to sort $\leq n/2$ suffixes

0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$





0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$

→ Distributed String Sorting

Distributed String Sample/Merge Sort

- Sort strings locally
- Compute splitters
- Distribute strings accordingly
- Merge received strings

Distributed String Sample/Merge Sort

- Sort strings locally → use string sorters by Bingmann and Rantala
- Compute splitters
- Distribute strings accordingly
- Merge received strings

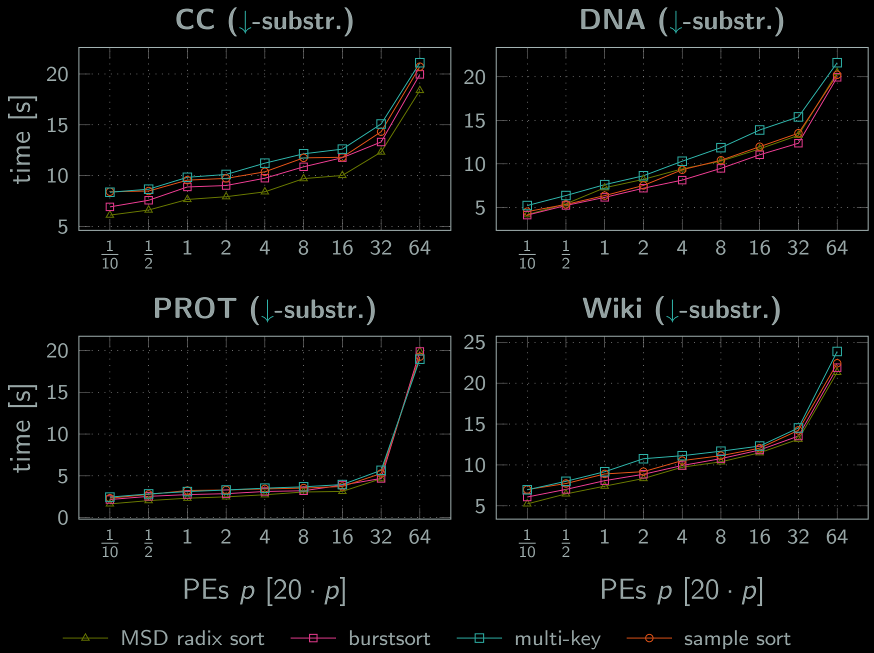
Experiments

node two Intel Xeon E5-2640v4 (10 cores each) and 64 GB RAM

PE one MPI thread per CPU core

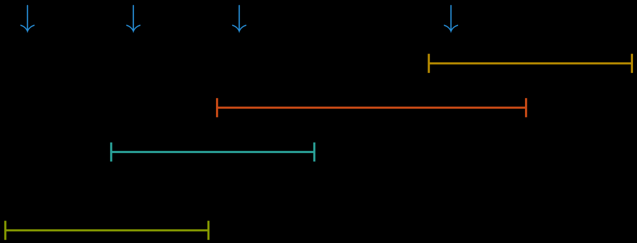
input real world texts (90 MB or 28 MB per PE) up to 115 GB

CC $\sigma = 242$
DNA $\sigma = 4$
Prot $\sigma = 26$
Wiki $\sigma = 213$

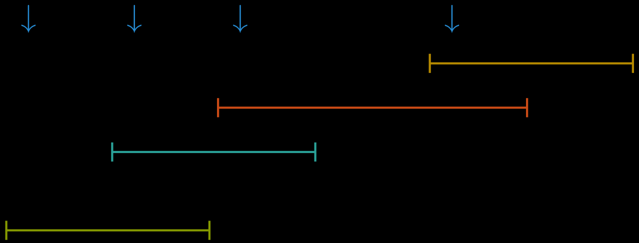


CC (↓-substr.)
DNA (↓-substr.)
PROT (↓-substr.)
Wiki (↓-substr.)
time [s]
time [s]
5
10
15
20
0
25
1/10
1/2
1
2
4
8
16
32
64
PEs p [$20 \cdot p$]
PEs p [$20 \cdot p$]
MSD radix sort
burstsort
multi-key
sample sort

0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$

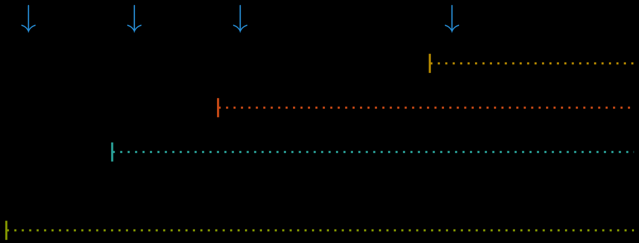


0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



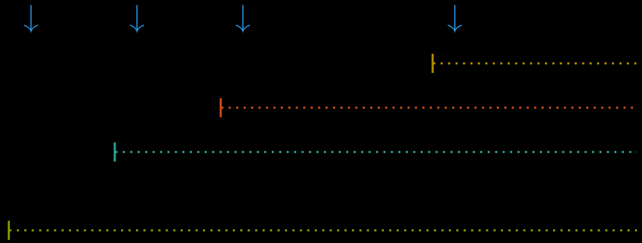
0	0	2	1

0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



0	0	2	1

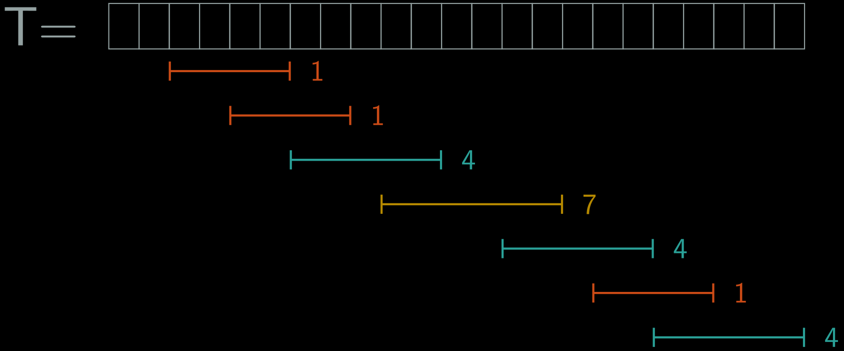
0	1	2	3	4	5	6	7	8	9	10	11	12
c	c	e	c	e	c	e	d	c	c	e	d	$



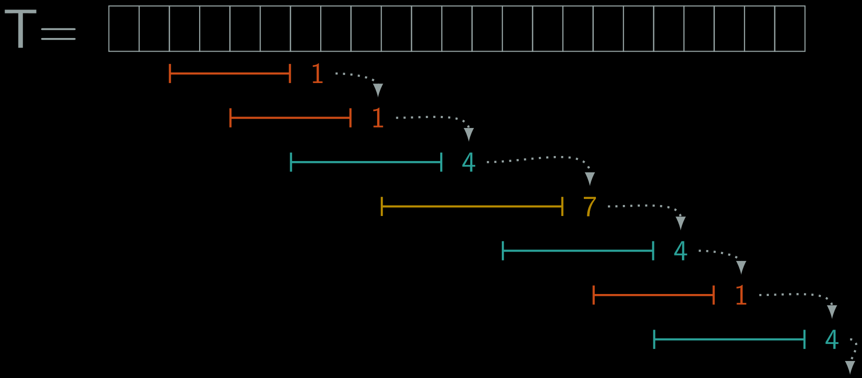
0	0	2	1

→ Distributed Suffix Array Construction

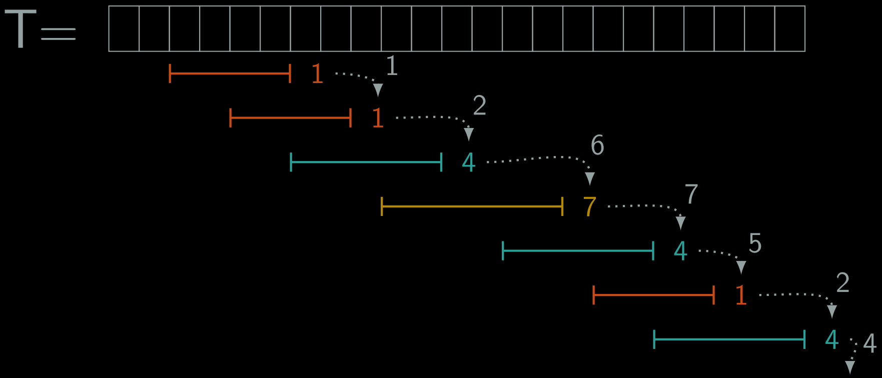
Prefix Doubling



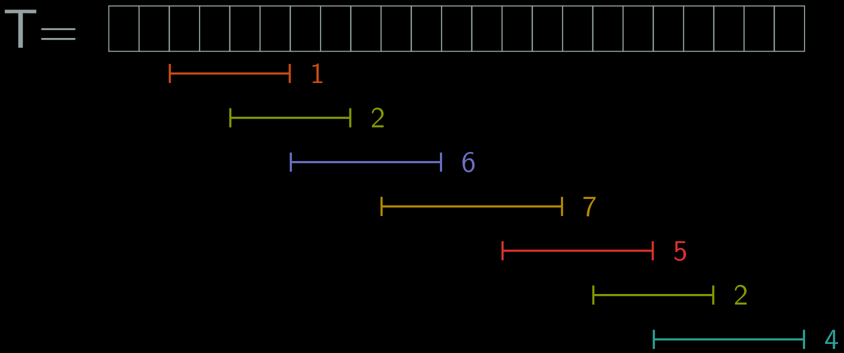
Prefix Doubling



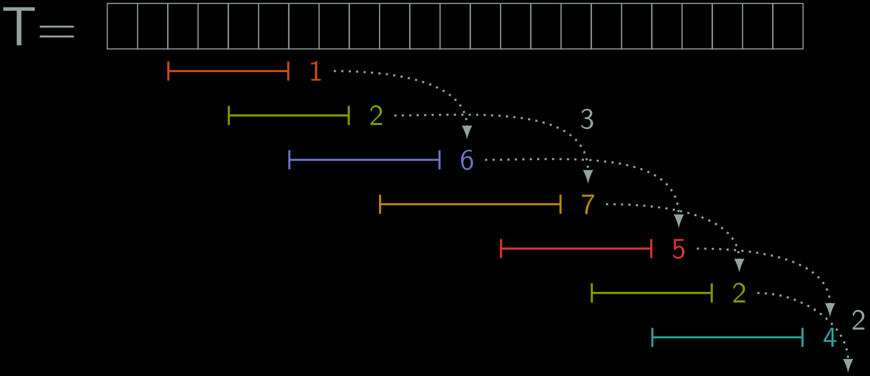
Prefix Doubling



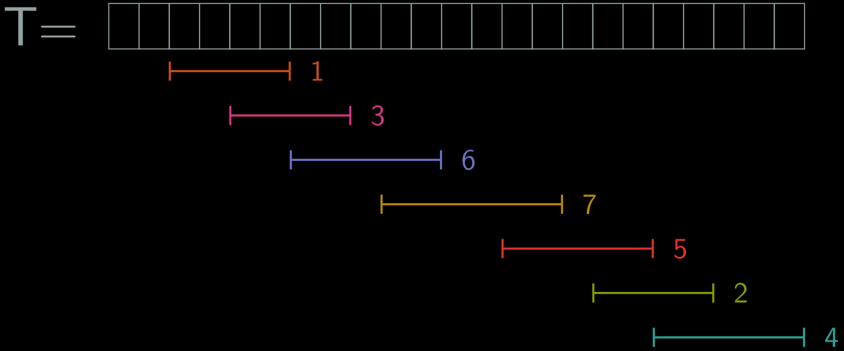
Prefix Doubling



Prefix Doubling



Prefix Doubling



Induce other suffixes

Inducing ↓ and ↲

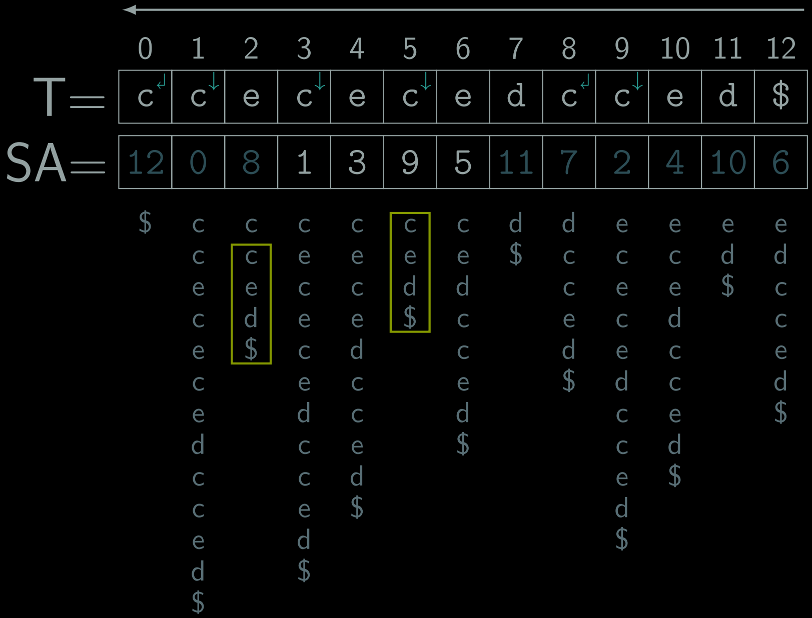
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c	c	e	c	e	c	e	d	c	c	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	c	c	c	c	c	c	d	d	e	e	e	e
		c	c	e	e	e	e	$	c	c	c	d	d
		e	e	c	c	d	d		c	e	e	$	c
		c	d	e	e	$	c		e	c	d		c
		e	$	c	d		c		d	e	c		e
		c		e	c		e		$	d	c		d
		e		d	c		d			c	e		$
		d		c	e		$			c	d		
		c		c	d					e	$		
		c		e	$					d			
		e		d						$			
		d		$									
		$											

Inducing ↓ and ↲

←

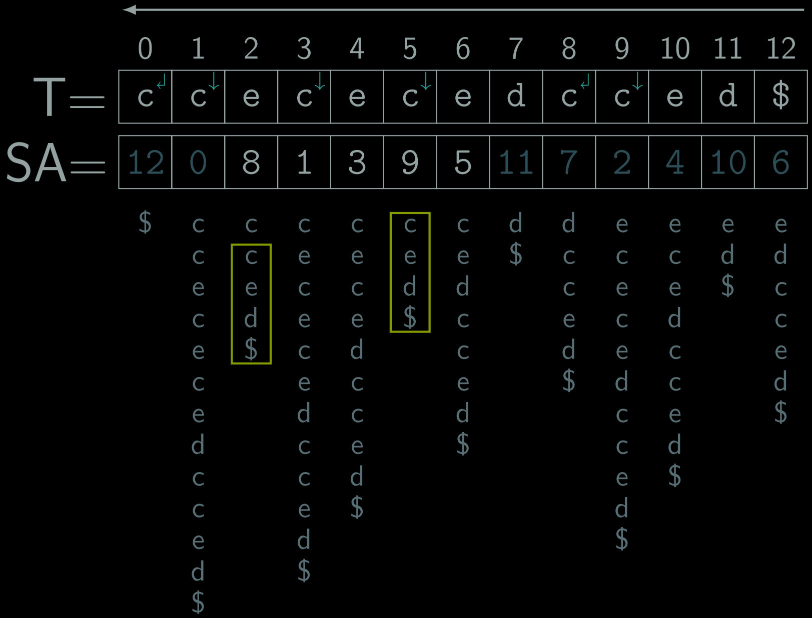
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲



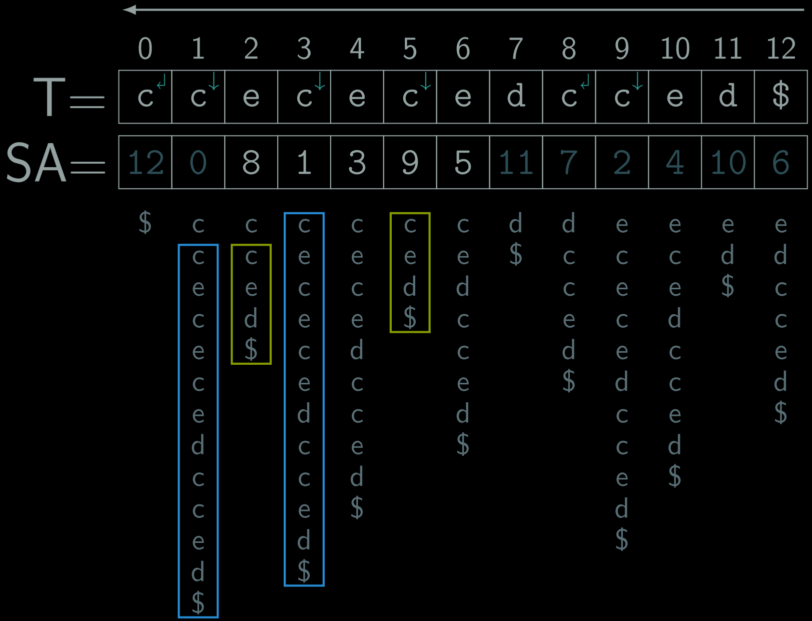
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲



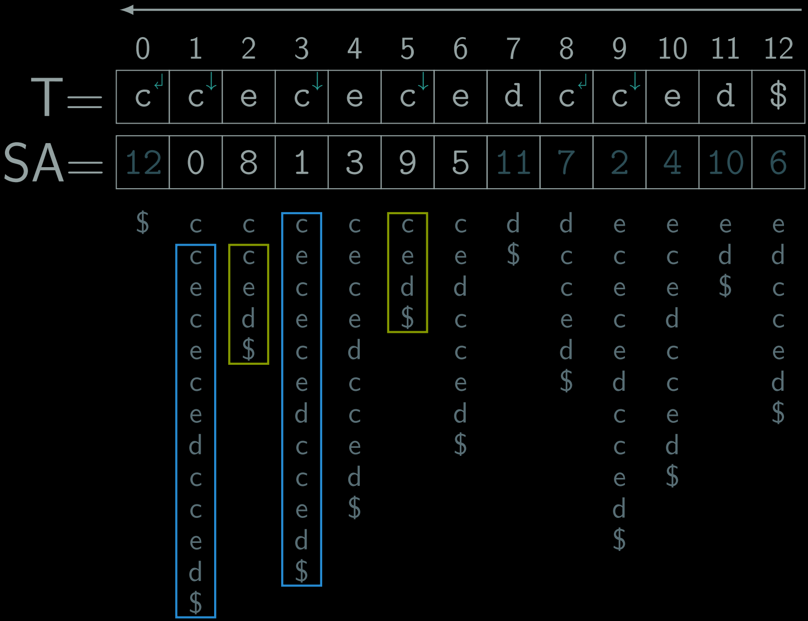
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲



	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲



	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedccced$	cced$	cececedccced$	cecedccced$	ced$	cedccced$	d$	dccced$	ecececedccced$	ececedccced$	ed$	edccced$

Inducing ↓ and ↲

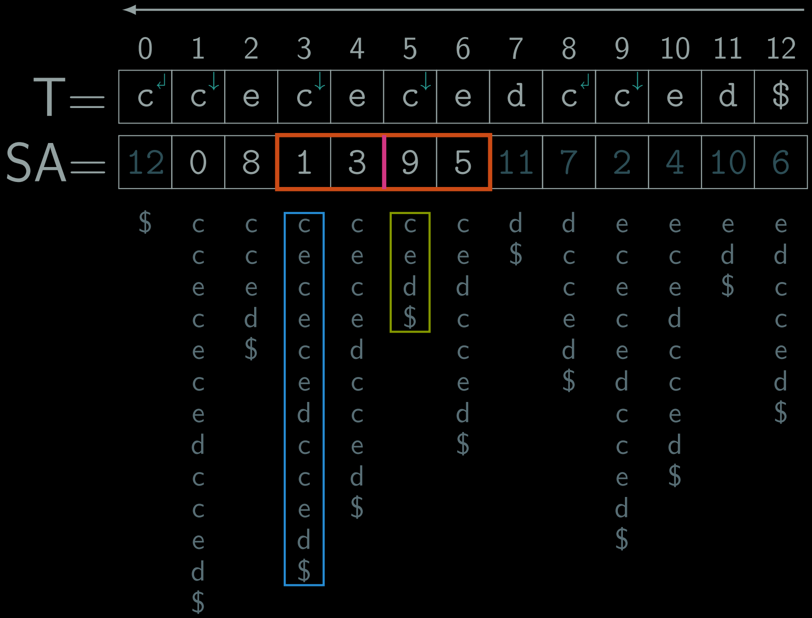
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲

←

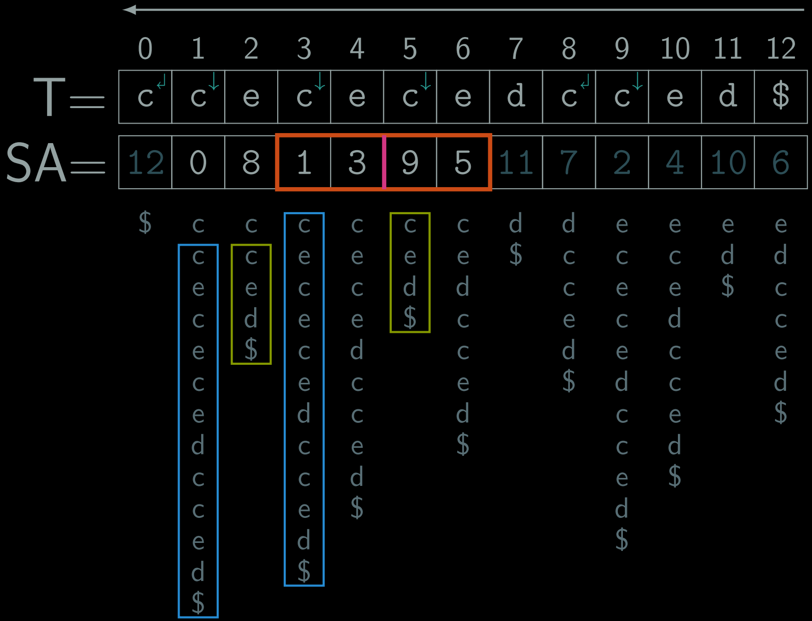
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

Inducing ↓ and ↲



	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedccced$	cced$	cececedccced$	cecedccced$	ced$	cedccced$	d$	dcced$	ececedccced$	ecedccced$	ed$	edcced$

Inducing ↓ and ↲



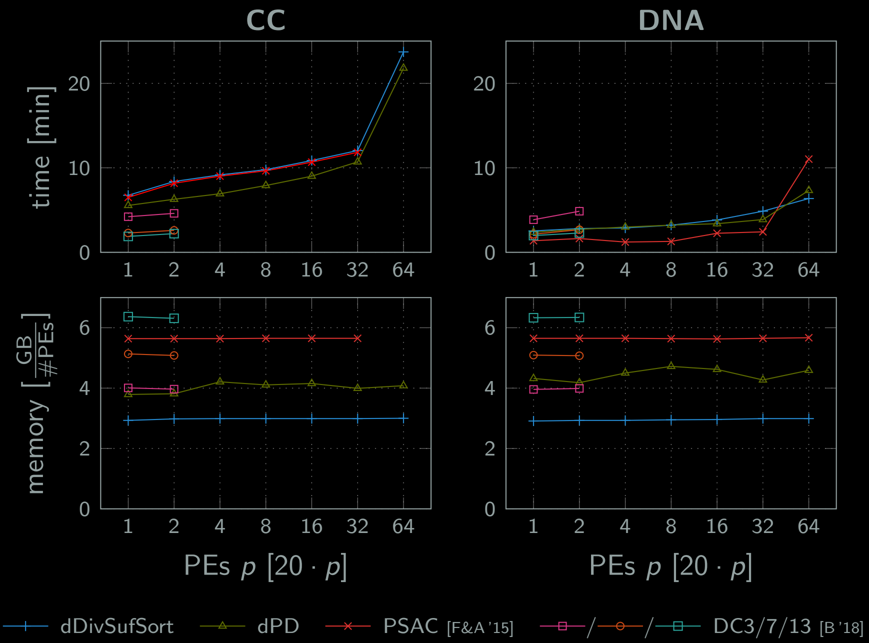
	0	1	2	3	4	5	6	7	8	9	10	11	12
T=	c↲	c↓	e	c↓	e	c↓	e	d	c↲	c↓	e	d	$
SA=	12	0	8	1	3	9	5	11	7	2	4	10	6
	$	ccececedcced$	cced$	cececedcced$	cecedcced$	ced$	cedcced$	d$	dcced$	ececedcced$	ecedcced$	ed$	edcced$

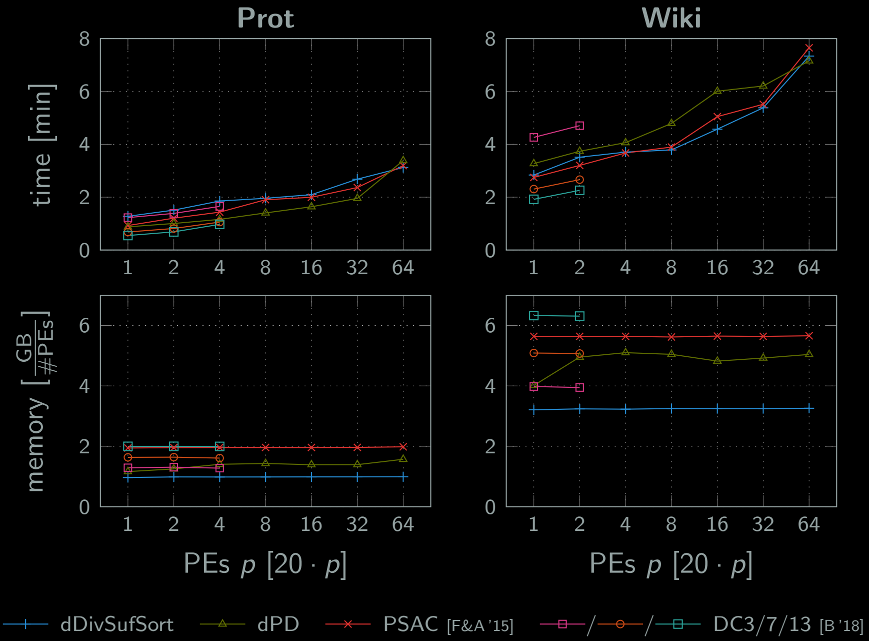
Key characteristics

- $2\sigma^2$ synchronizations
- work roughly equal on every MPI thread
- repetitions aaaa. . . aa require special case

Experiments

- same set-up as before
- measure construction time and
- memory peak


CC
DNA
time [min]
0
10
20
memory [$\frac{GB}{\#PEs}$]
0
2
4
6
1
2
4
8
16
32
64
PEs p [20 · p]
dDivSufSort
dPD
PSAC [F&A '15]
DC3/7/13 [B '18]


Prot
Wiki
time [min]
memory [$\frac{\text{GB}}{\#\text{PEs}}$]
0
2
4
6
8
1
2
4
8
16
32
64
PEs p [$20 \cdot p$]
dDivSufSort
dPD
PSAC [F&A '15]
DC3/7/13 [B '18]

Conclusion

- very memory efficient and
- reasonable fast distributed suffix array construction algorithms

Future Work

- computing the LCP-array

Conclusion

- very memory efficient and
- reasonable fast distributed suffix array construction algorithms

Future Work

- computing the LCP-array

Thank You