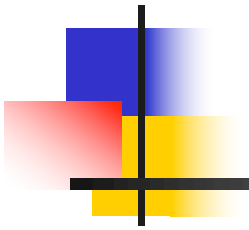


IN3030 Uke 12, v2019

Eric Jul
PSE,
Inst. for informatikk

Hva skal vi se på i Uke 12

- Review Radix sort
- Oblig 4
  - Text
  - Program
  - Parallellizing

Oblig 4 Radix sort

- Parallelliser Radix-sortering med fra 1 – 5 sifre
- Skriv rapport om speedup for n= 1000, 10 000, 100 000, 1 mill., 10 mill og 100 mill.
- Radix består av to metoder, begge skal parallelliseres.
- Den første har et steg, finn max(a[]) – løst tidligere
- Den andre har tre steg – løses i parallell effektivt:
  - b) tell hvor mange det er av hvert sifferverdi i a[] i count[]
  - c) legg sammen verdiene i count[] til 'pekere til b[]
  - d) flytt tallene fra a[] til b[]
- Steg b) er løst tidligere (hvor bla. hver tråd har sin kopi av count[])

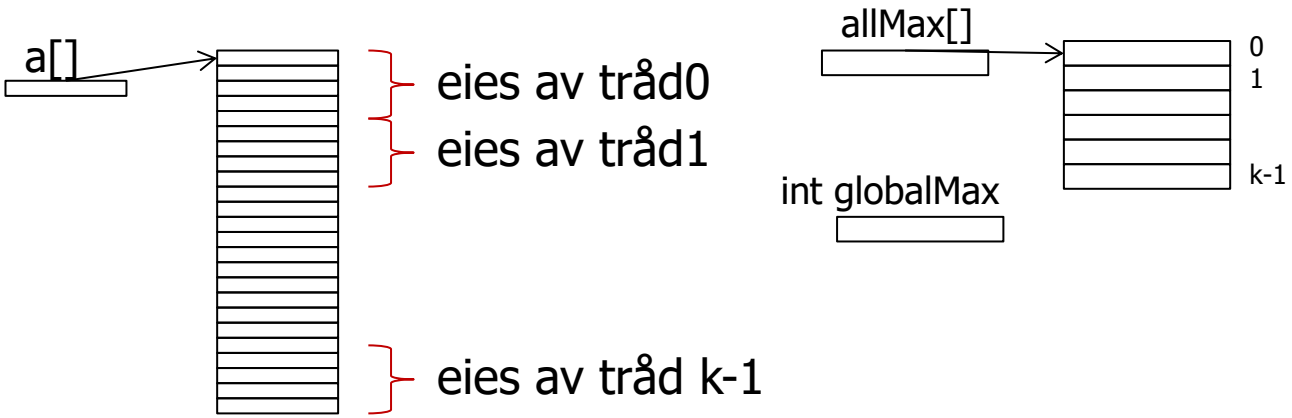
Flere tips til Oblig3 - MultiRadix

- Problemet med data-konkurranse: To eller flere tråder skriver 'samtidig' på samme variabel (i++-problemet), i samme plass i en array:
  - Løsning: Hver tråd har en kopi av disse felles variable
  - Etter at alle trådene er ferdig (f.eks. etter en barrier-synk) kan resultatene fra hver tråd samstilles (også dette helst i parallell) til et felles svar
  - Muligens må man kopiere data mer enn en gang ?
- Oppdeling av data i arrayer man skal behandle med k tråder:
  - Dele opp arrayen i like store deler (det er indeksene man deler opp)
  - Dele opp etter verdiene i elementene (tråd 0 eier de minste verdiene, tråd 1 de nest-minste,.. ,)

Datastrukturer og grep i Oblig4 (hvordan parallellisere)

- Radix består av 4 steg (løkker) – flere valg for datastruktur og oppdeling ved parallellisering
  - a) finn max verdi i a[]
  - b) count= oppttelling av ulike sifferverdier i a[]
  - c) Summér opp i count[] akkumulerte verdier (pekere)
  - d) Flytt elementer fra a[] til b[] etter innholdet i count[]
- Generelt skal vi se følgende teknikker med k tråder:
  - Dele opp data i a [] i k like deler
  - Dele opp verdiene i a[] i k like deler => dele opp count[] i k like deler
  - Kopiere delte data til hver tråd – her lokal count[] kopi en eller to ganger
  - Innføre ekstra datastrukturer som ikke er i den sekvensielle løsningen

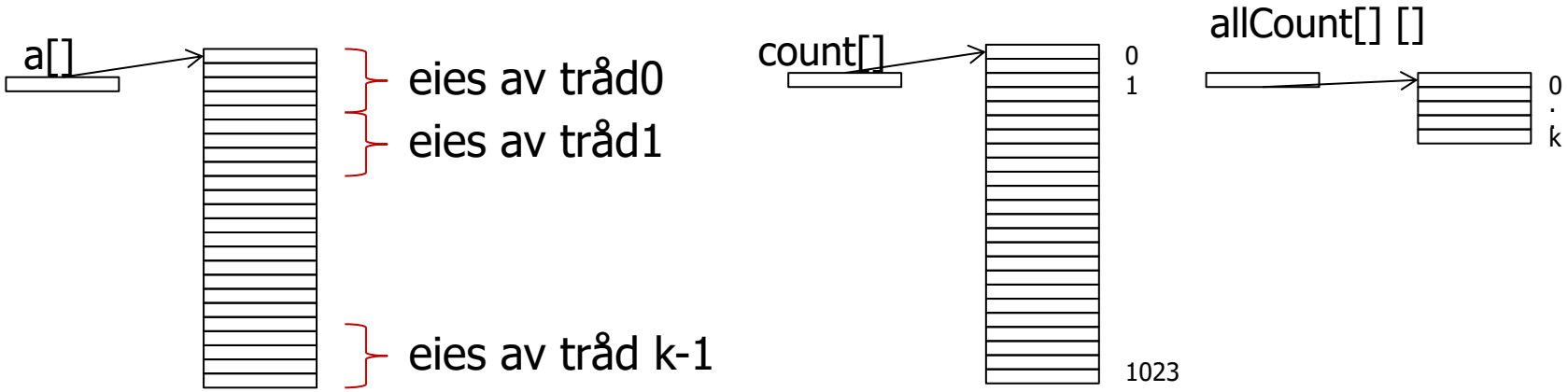
a) finn max verdi i a[]



- Tråd-i finner max i sin del av a[] og legger svaret i allMax[i]
- <sync på en CyclicBarrier cb>
- Nå har alle trådene sin max i allMax[] – valg nå:
  - Skal en av trådene (f.eks. tråd-0) finne svaret og legge det i en felles globalMax (mens de andre trådene venter i så fall nok en <sync på en CyclicBarrier cb>) ?
  - Skal alle trådene hver regne ut en lokal globalMax (de får vel samme svar?) og fortsette direkte til steg b)

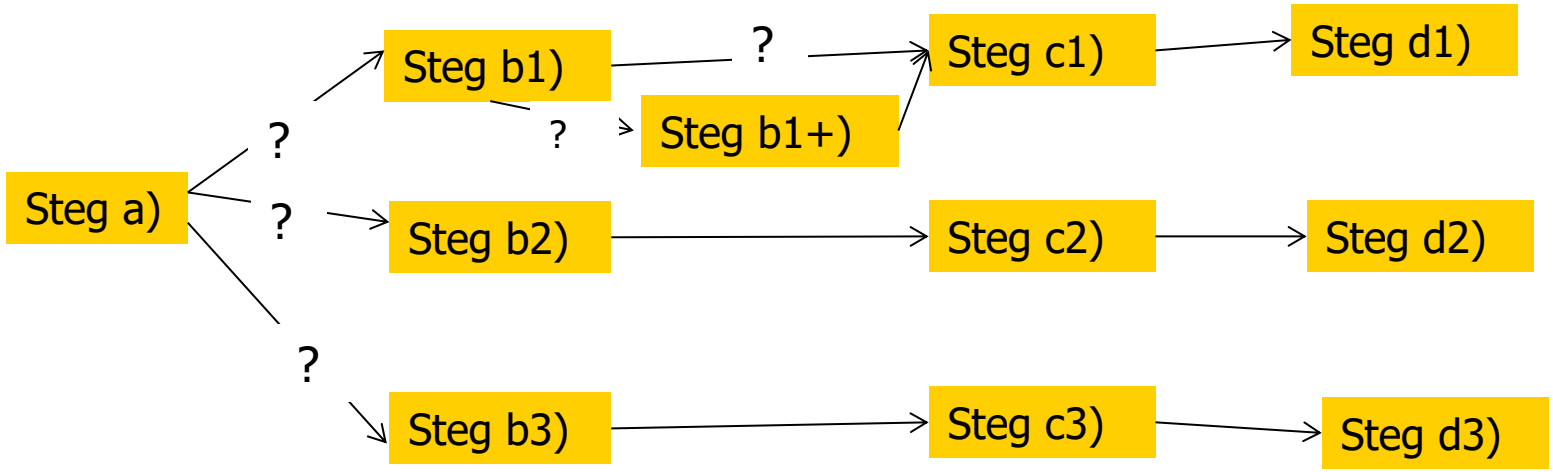
b) count= oppttelling av ulike sifferverdier i a[]

Anta at det er 10 bit i et siffer – dvs. 1024 mulige sifferverdier



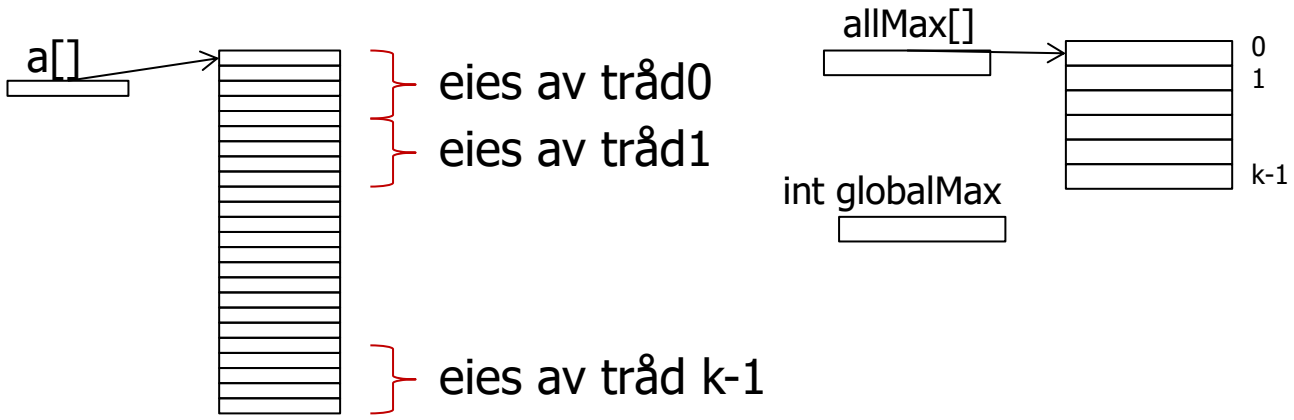
- Skal:
  1. Hver tråd ha en kopi av count[]
  2. Eller skal count være en AtomicIntegerArray
  3. Eller skal de ulike trådene gå gjennom hele a[] og tråd-0 bare ta de små verdiene, tråd-1 de nest minste verdien,..(dvs: dele verdiene mellom trådene)

Roadmap – oversikt over drøftelsen av Oblig4 – flere alternativer på steg b og senere endringer i c og d



FinnMax Tell sifferverdier lag pekere flytt a[] til b[]

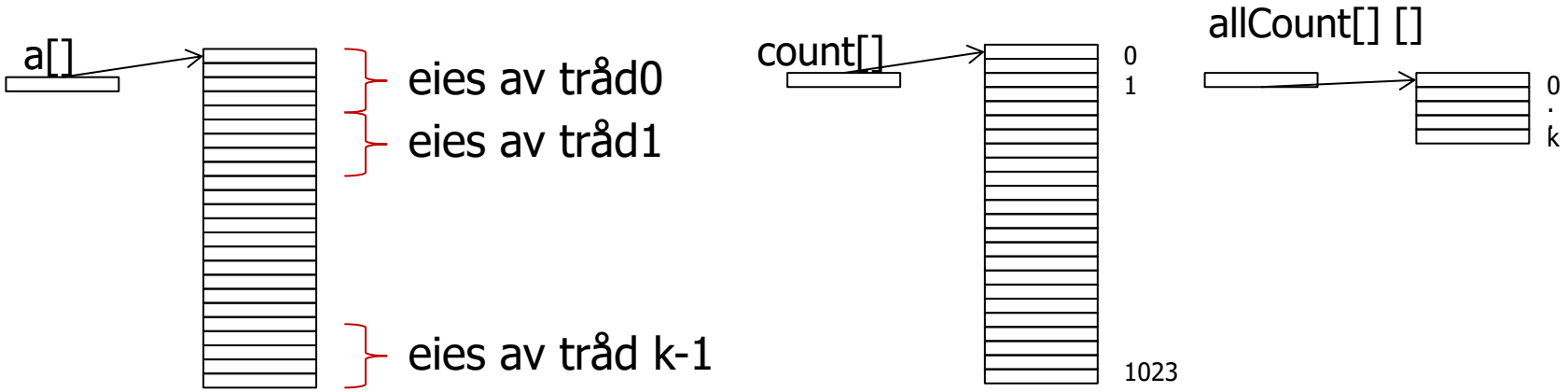
a) finn max verdi i a[]



- Tråd-i finner **max** i sin del av a[] og legger svaret i allMax[i]
- <sync på en CyclicBarrier cb>
- Nå har alle trådene sin max i allMax[] – valg nå:
  - Skal en av trådene (f.eks. tråd-0) finne svaret og legge det i en felles globalMax (mens de andre trådene venter i så fall nok en <sync på en CyclicBarrier cb>) ?
  - Skal alle trådene hver regne ut en lokal globalMax (de får vel samme svar?) og fortsette direkte til steg b)

b) count= oppttelling av ulike sifferverdier i a[]

Anta at det er 10 bit i et siffer – dvs. 1024 mulige sifferverdier



- Skal:
  1. Hver tråd har en kopi av count[]
  2. Eller skal count være en AtomicIntegerArray
  3. Eller skal de ulike trådene gå gjennom hele a[] og tråd-0 bare ta de små verdiene, tråd-1 de nest minste verdien,..(dvs: dele verdiene mellom trådene)

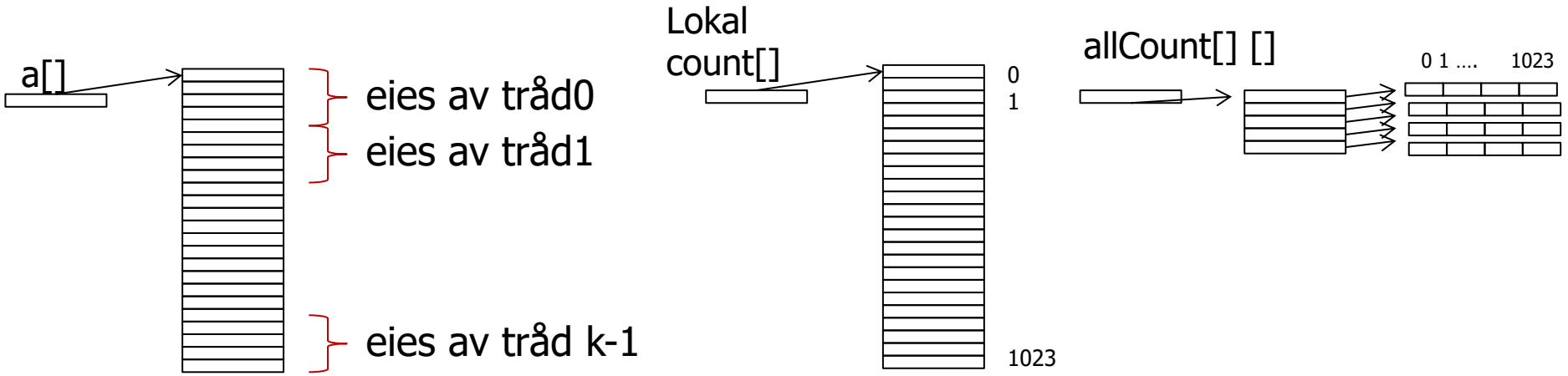
Om delvis deklarasjoner av arrayer

int allCount[] []; - da lages ?

int allCount[] [] = new int [k][]; - da lages ?

int [] count = new int[10] ;
allCount[k] [0] = x;

Løsning b1) – lokale count[] i hver tråd som etter opptelling settes inn i allCount[] []

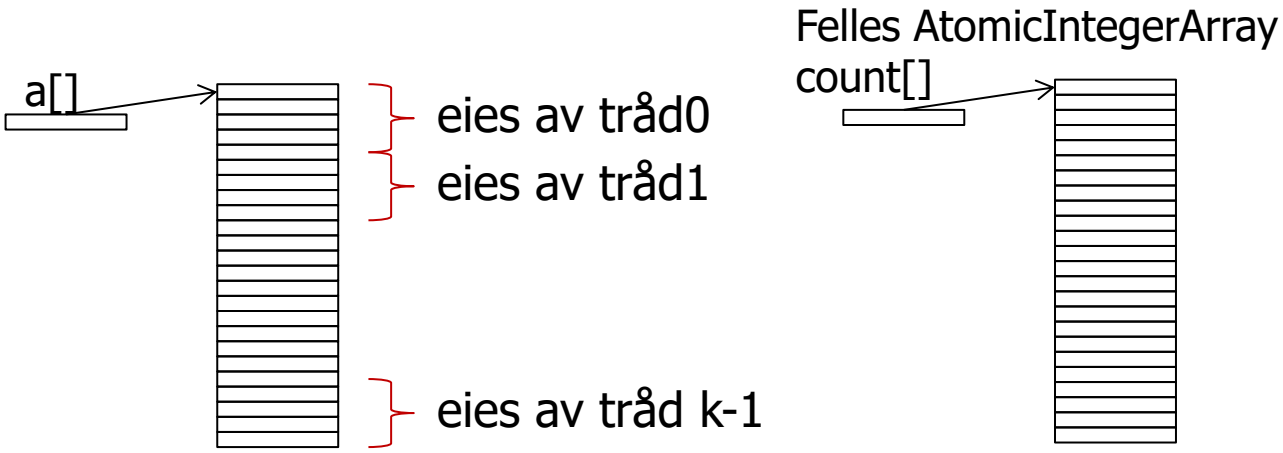


- Hver tråd-i teller opp de ulike sifferverdien i sin del av a[] opp i sin lokale count[] som så hektes inn i allCount[] []
- <sync på en CyclicBarrier cb>
- Nå er hele opptellingen av a[] i de k count[]-ene som henger i allCount[] []
- Denne løsningen trenger (kanskje) et tillegg b1-b:
  - Summering av verdiene i allCount[][] til en felles globalCount[]

b1+) : Summering av verdiene i allCount[][] til en felles: globalCount[]

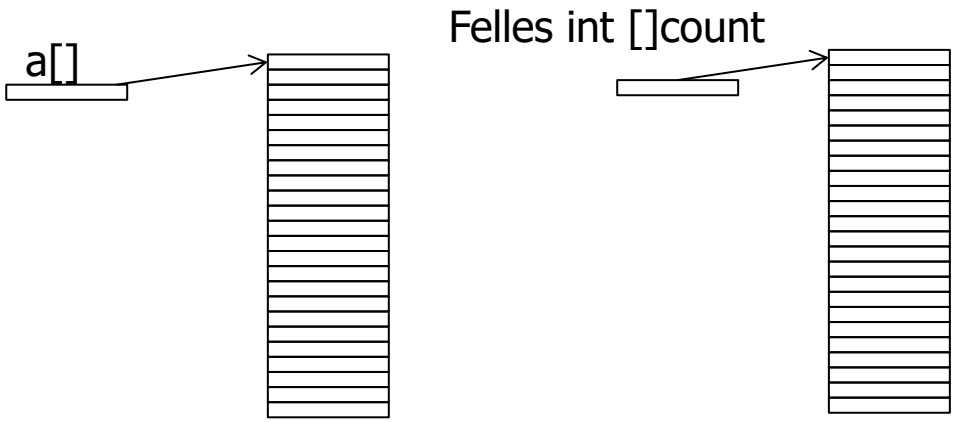
- Deler opp de mulige sifferverdiene (1024) på de k trådene slik at tråd-0 for de 1024/k minste, tråd-1 de 1024/k nest minste,…
- Tråd-i summerer sine verdier (sine kolonner) 'på tvers' i de k count[] – arrayene som henger i allCount[] [], inn i globalCount[]
- Bør allCount[][] transponeres før summering (mer cache-vennlig) ??
- <sync på en CyclicBarrier cb>
- Da er globalCount[] fullt oppdatert
- Spørsmål:
  - Trenger vi globalCount[] , eller holder det med allCount[][]?
  - Svar : Avhenger av neste steg c)

Løsning b2) – Eller skal count[] være en AtomicIntegerArray?



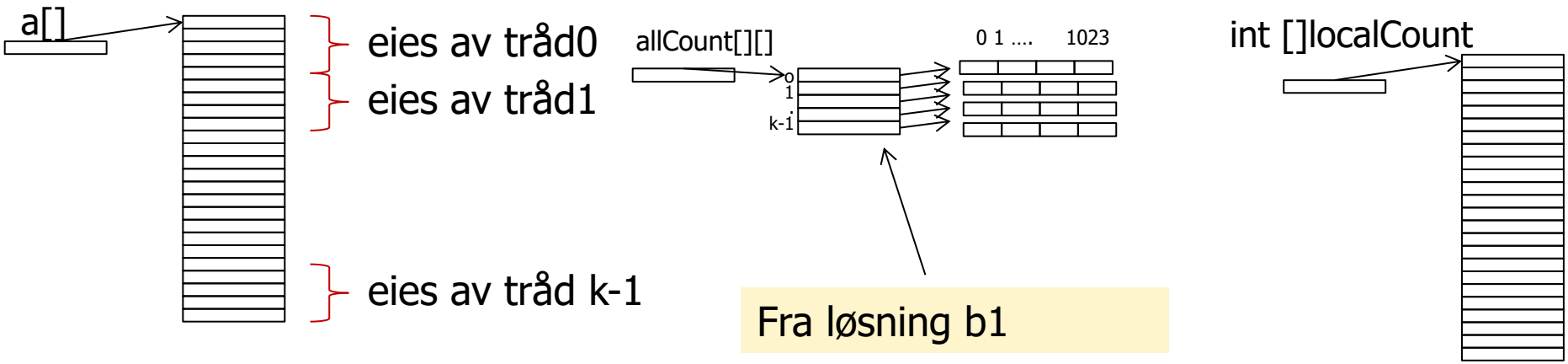
- Alle tråder oppdaterer AtomicIntegerArray count[] samtidig uten fare for synkroniserings-problemer med sifferverdiene fra sin del av a[] pga synkroniseringa av AtomicIntegerArray-elementene
- <sync på en CyclicBarrier cb>
- Spørsmål:
  - Hvor mange synkroniseringer trenger vi med denne løsningen

Løsning b3) – Eller skal de ulike trådene gå gjennom hele a[] og tråd-0 bare ta de n/k minste verdiene, tråd-1 de n/k nest minste verdiene,.. (dvs. dele verdiene mellom trådene)



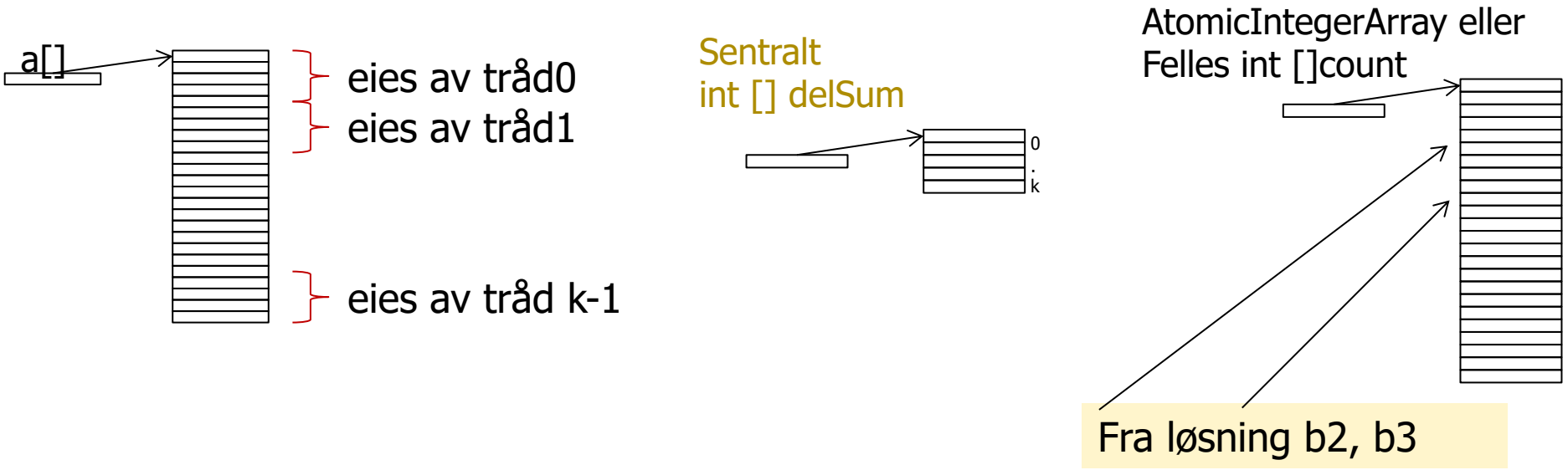
- Tråd-i oppdaterer count[] bare med **de verdiene** som tråd-i eier uten fare for synkroniserings-problemer med sifferverdiene fra hele a[] – ingen andre skriver slike sifferverdier.
- <sync på en CyclicBarrier cb>
- Spørsmål:
  - Hvor mange synkroniseringer trenger vi med denne løsningen
  - Hvor mye av a[] leser hver tråd

c1) **Gitt b1**: Summér opp i count[] akkumulerte verdier (pekere)



- Vi trenger et prinsipp for at hver tråd finner akkurat hvor den skal plassere sin del av a[].
  - Hver tråd får **nok en kopi** av count[]: localCount[] – initielt tom (=0)
  - Tråd-i sin localCount[t] er lik summen av alle tråder sin opptelling i allCount[r][s] for alle r og s<t + sum av allCount[r][t] for r < i når s = t
  - Vitsen er at før tråd-i plasserer sine sifferverdier s, må det først gjøres plass til alle lavere sifferverdier + de sifferverdiene i tråder med indeks mindre enn i.
- <sync på en CyclicBarrier cb>

c2 og c3) Summér opp i count[] akkumulerte verdier (pekere)



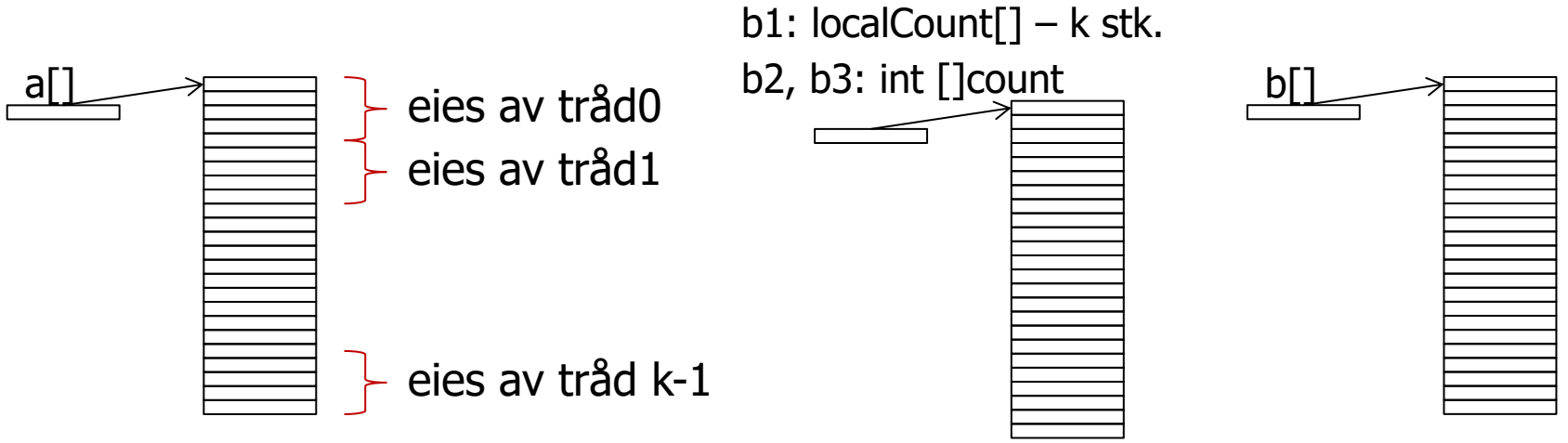
- I løsning b2 og b3 har vi samme situasjon som sekvensiell løsning. Kan parallelliseres som følger :Deler verdiene i count[] mellom de k trådene som før. Alle summerer sine verdier og legger de i en separat array delSum[k].
- <sync på en CyclicBarrier cb>
- Deretter justerer du dine plasser i count[] med summen av delsummene i delSum[k] fra tråder med mindre indeks enn deg.
- Spørsmål:
  - Hvor lang tid tar dette kontra at tråd-0 gjør hele jobben og de andre venter.

c4+b1) Summér opp count[][] OG juster pekere

- Etter b1) mangler: oppsummering af alle de lokale count[] i et globalCount[]
- <sync på en CyclicBarrier cb>
- Justering af globalCount[] så sum bliver til pekere.
- Dette kan kombineres i EN løkka, hvis man gjør det sekvensielt i EN tråd.
- <sync på en CyclicBarrier cb>
- Sekvensiell løsning muligvis svær at gjøre raskere i parallell?

- Spørsmål:
  - Hvor lang tid tar dette kontra at en tråd gjør hele jobben og de andre venter?
  - Prøv at regne på antallet av operationer versus antal operationer i andre steg.

d1,2 og 3) Flytt elementer fra a[] til b[] etter innholdet i count[]



- Løsning b1: Nå kan alle trådene i full parallell kopiere fordi hver localCount[] peker inn i ulike plasser i b[].
- Løsning b2: Alle trådene kan i full parallell kopiere over fra a[] til b[] fordi hver gang blir synkronisering foretatt av AtomicIntegerArray.
- Løsning b3: Nå kan alle trådene i full parallell flytte elementer fra a[] til b[] fordi hver tråd bare flytter sine sifferverdier. Hver tråd går gjennom hele a[] .
- Alle b1,b2,b3: <sync på en CyclicBarrier cb>
  - Spørsmål: Hvilken av løsningene tar lengst tid?