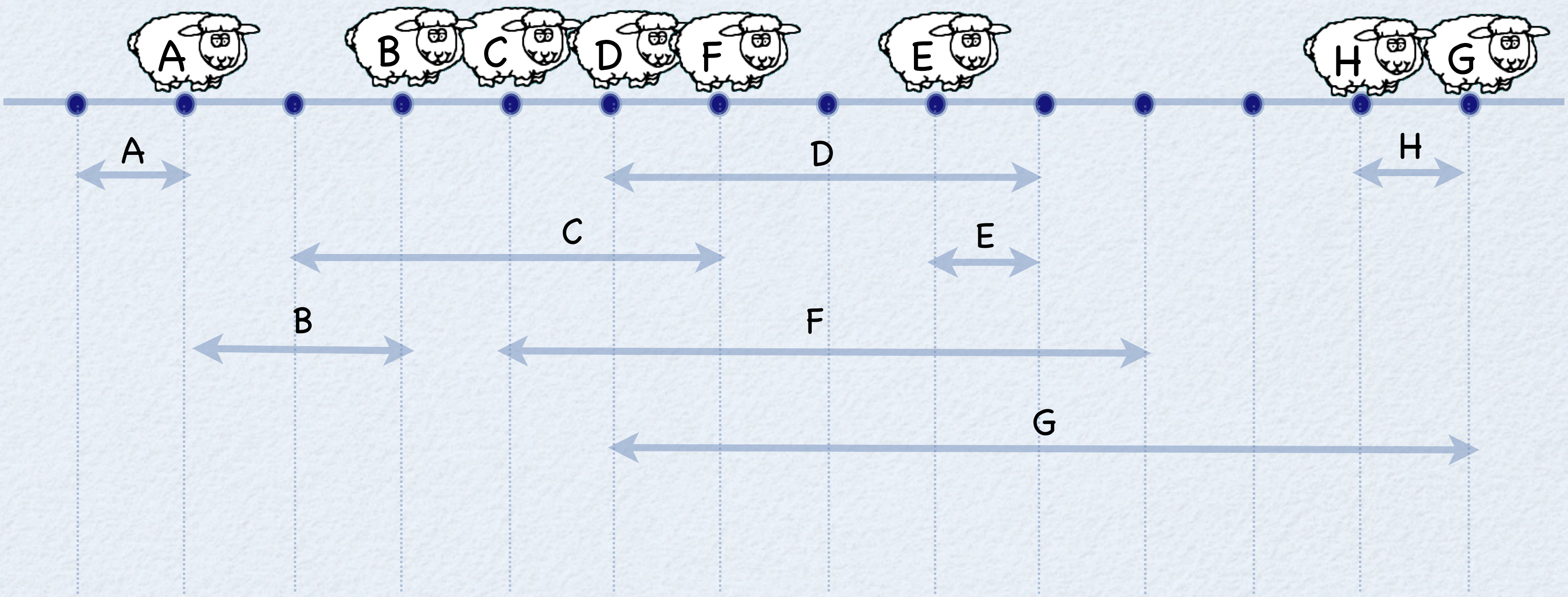
Minimum Energy Scheduling

Marek Chrobak
University of California, Riverside

How to Keep Sheep Warm in a Snow Storm?

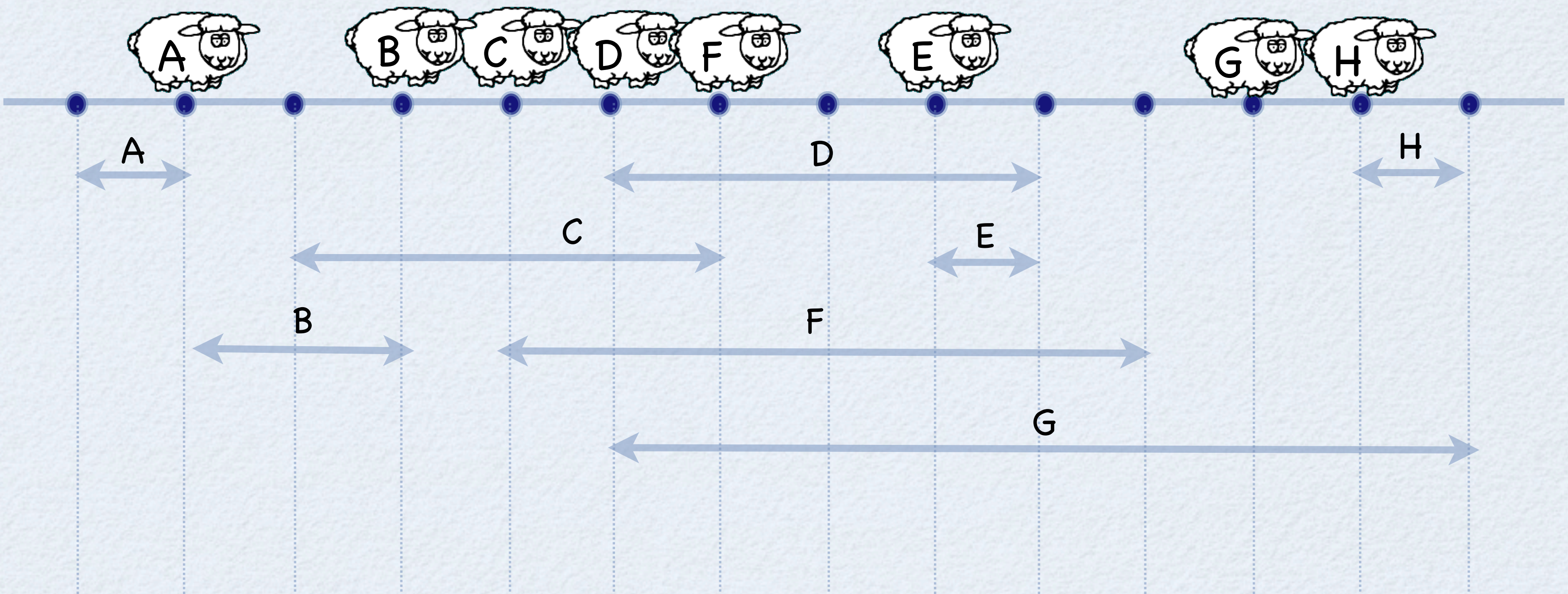
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



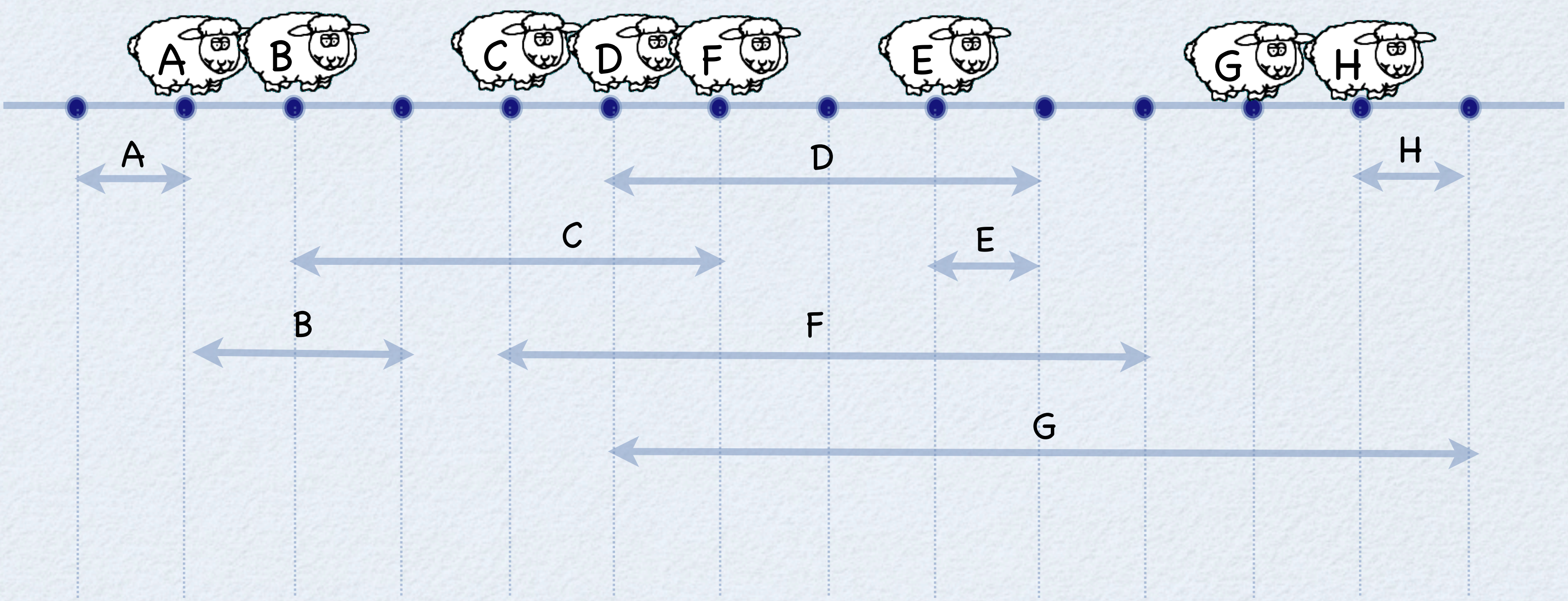
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



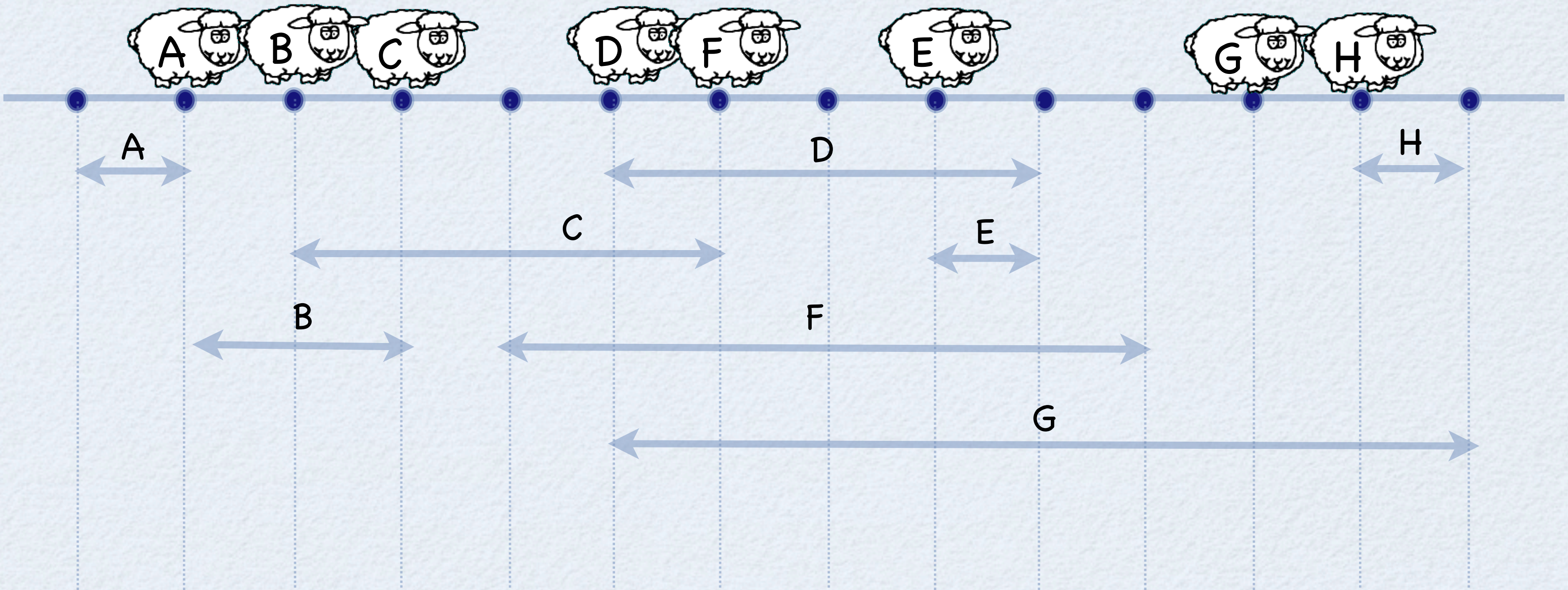
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



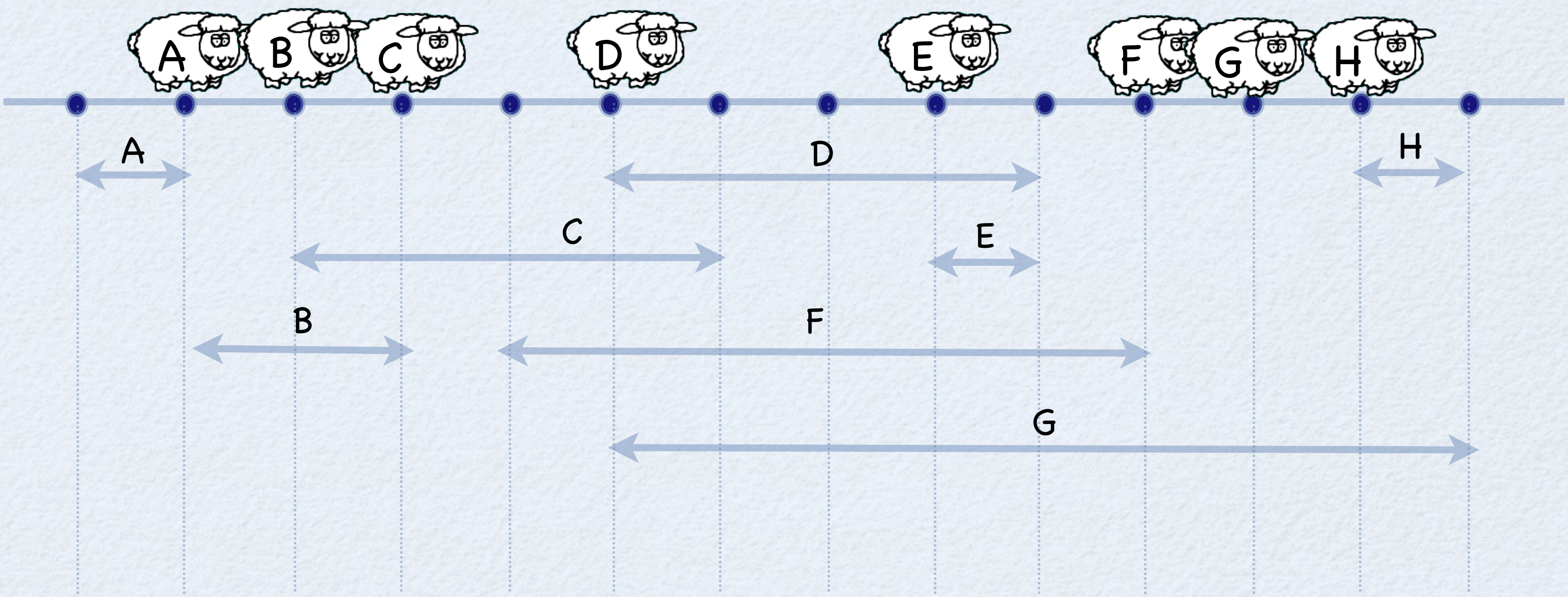
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



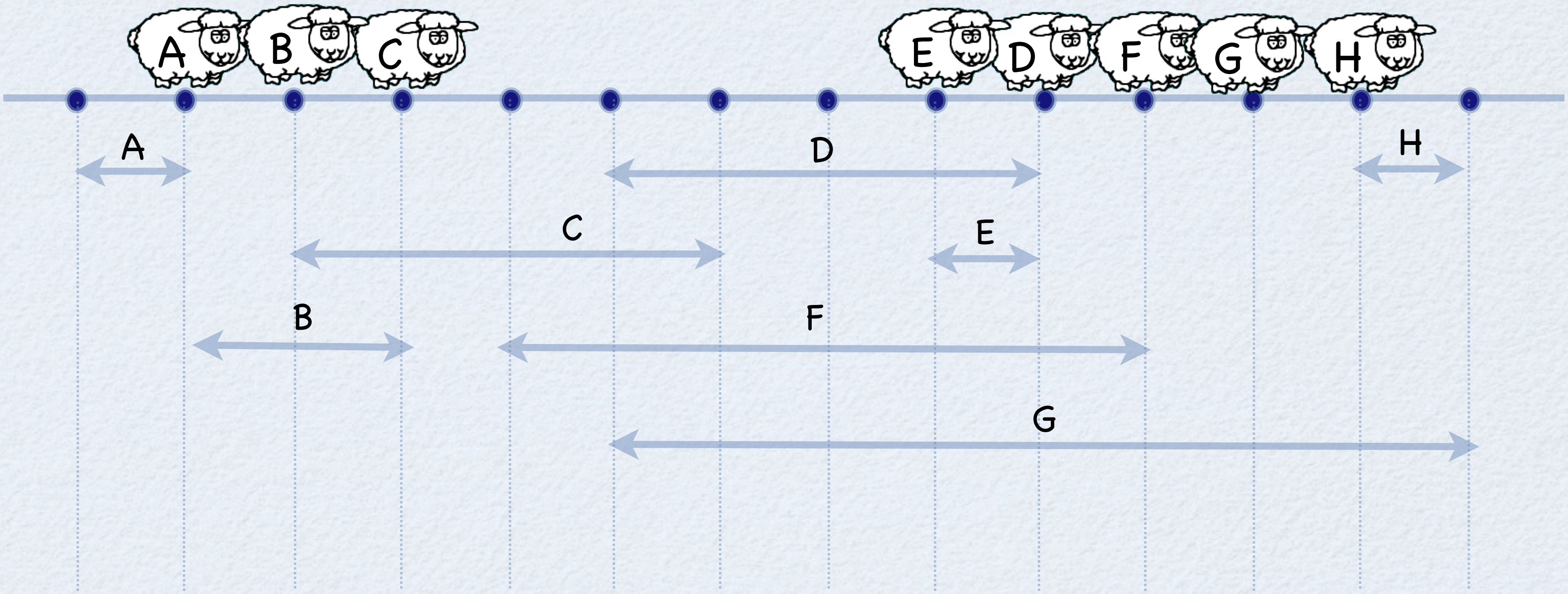
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



How to Keep Sheep Warm in a Snow Storm?

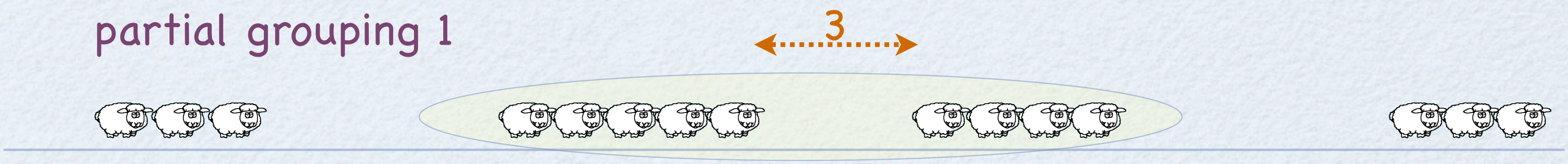
- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps



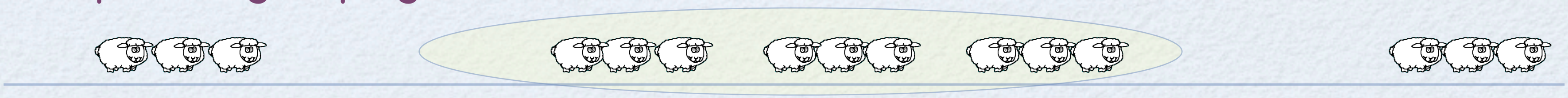
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can't work ...

partial grouping 1

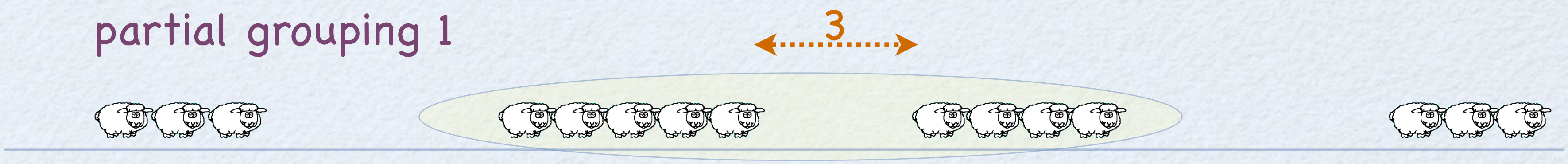


partial grouping 2

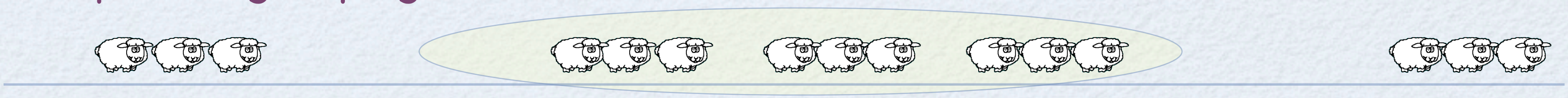


Intuition -- why naive algorithms can't work ...

partial grouping 1



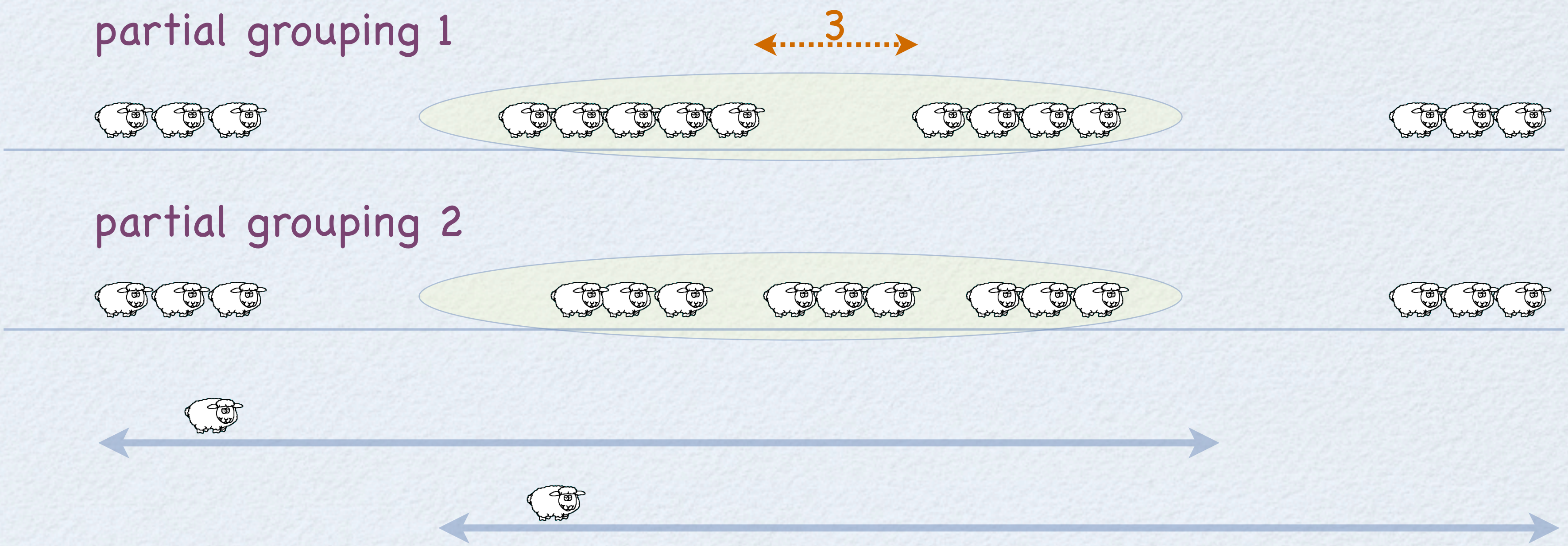
partial grouping 2



grouping 1 is better, so far ...

How to Keep Sheep Warm in a Snow Storm?

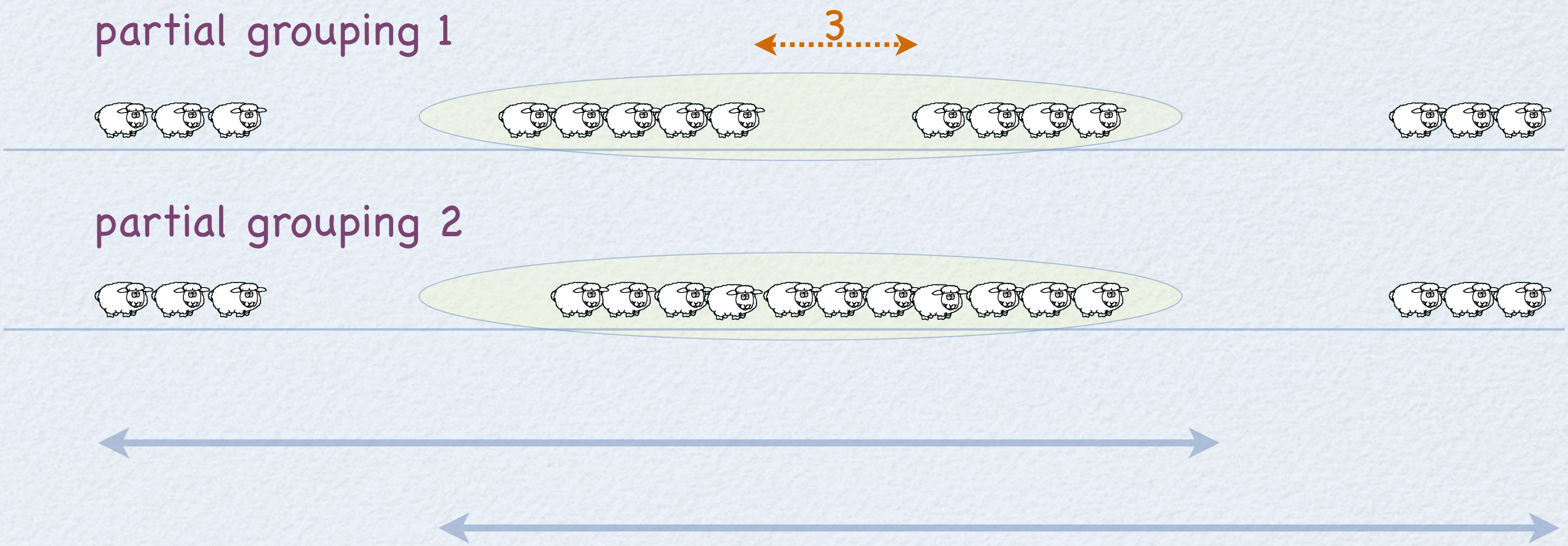
Intuition -- why naive algorithms can't work ...



grouping 1 is better, so far ...

How to Keep Sheep Warm in a Snow Storm?

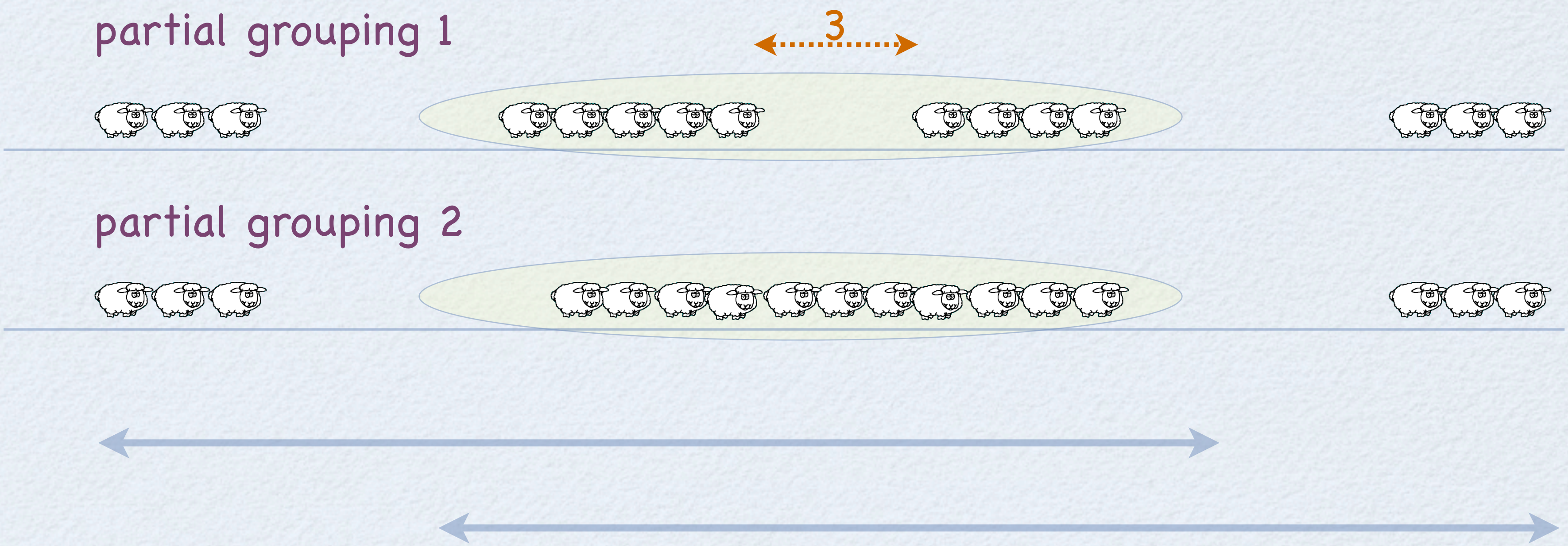
Intuition -- why naive algorithms can't work ...



grouping 1 is better, so far ...

How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can't work ...



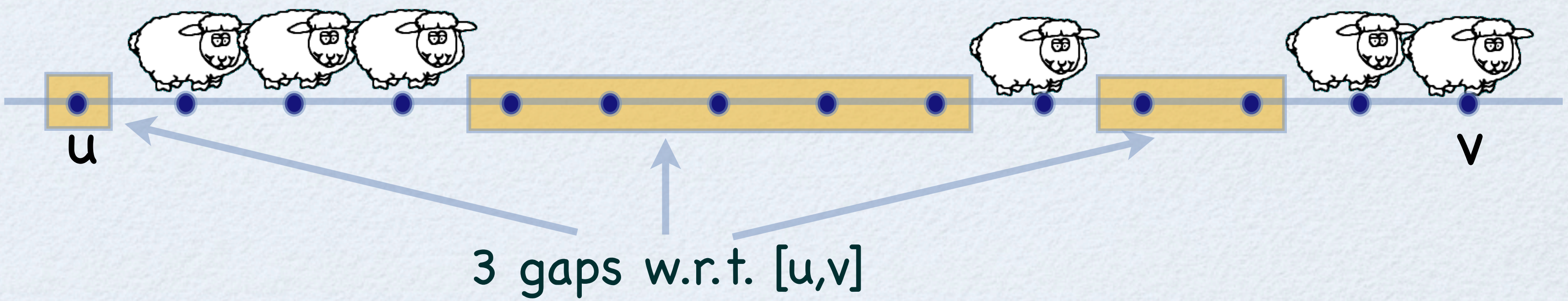
grouping 1 is better, so far ...

but only grouping 2 could be extensible to global optimum

tradeoff: # gaps vs gap sizes

How to Keep Sheep Warm in a Snow Storm?

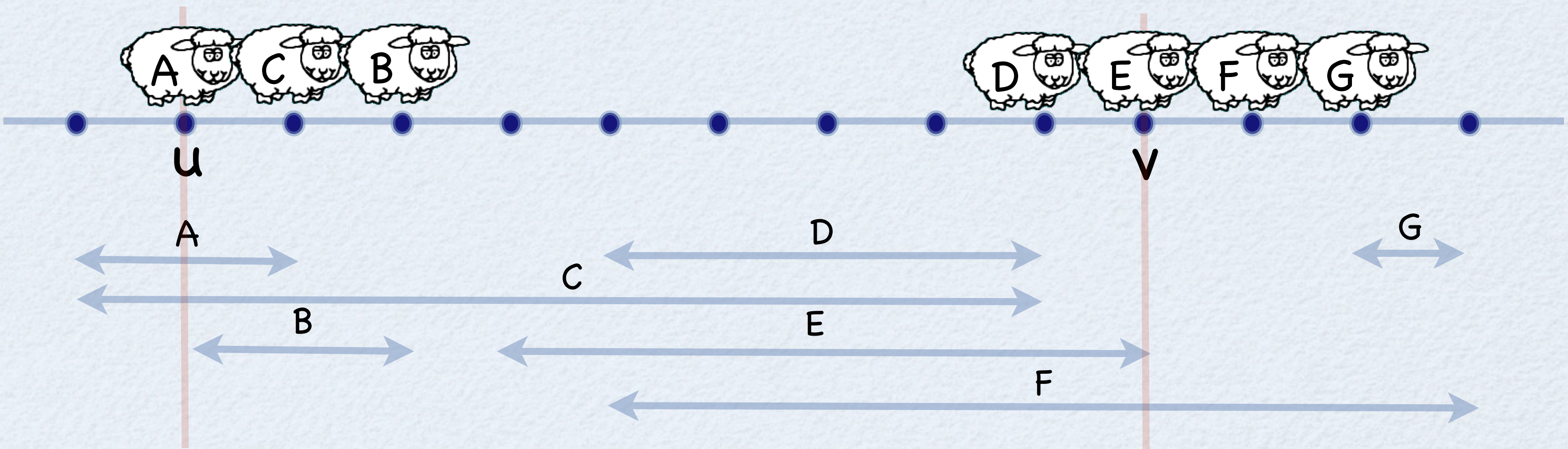
- $[a_j,b_j]$ = range of sheep j
- assume $b_1 \le b_2 \le \dots \le b_n$
- a gap with respect to interval [u,v]:
  - internal gap or
  - initial gap or
  - final gap



How to Keep Sheep Warm in a Snow Storm?

$\text{Inst}_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$\text{Gaps}_k(u,v)$ = min. number of gaps of $\text{Inst}_k(u,v)$ w.r.t $[u,v]$

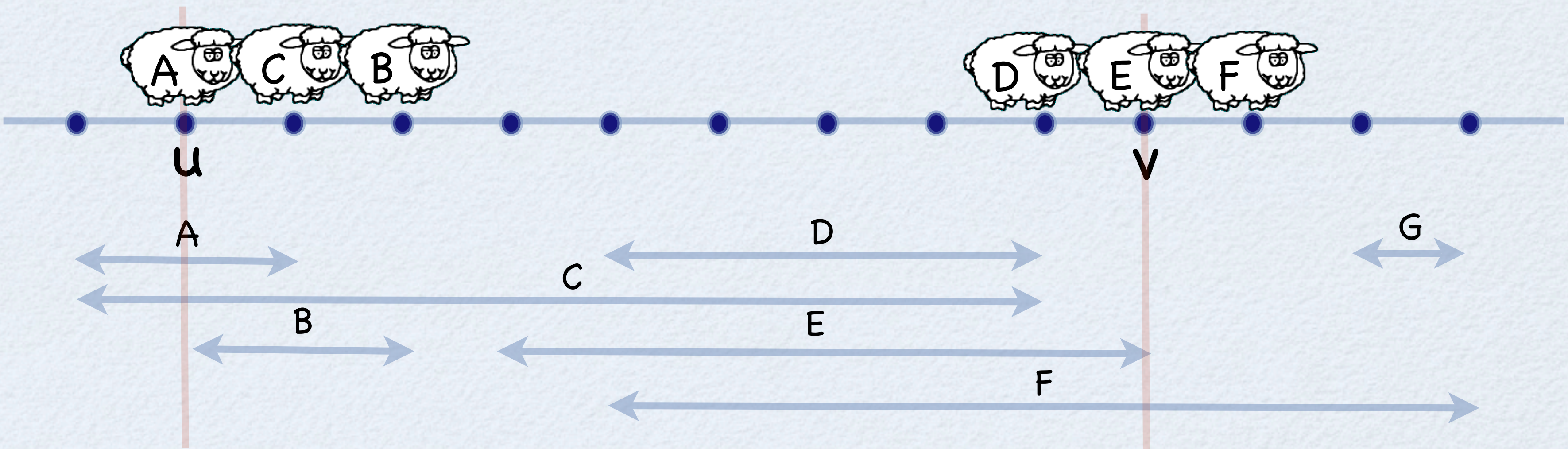


$\text{Gaps}_F(u,v)$ = ?

How to Keep Sheep Warm in a Snow Storm?

$\mathrm{Inst}_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$\mathrm{Gaps}_k(u,v)$ = min. number of gaps of $\mathrm{Inst}_k(u,v)$ w.r.t $[u,v]$

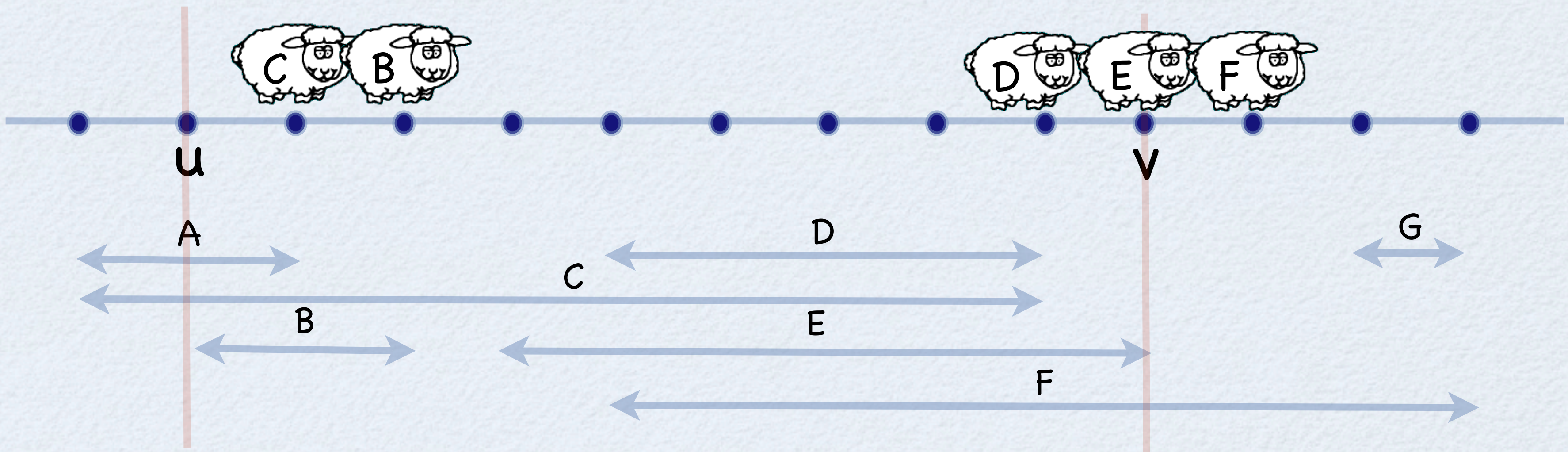


$\mathrm{Gaps}_F(u,v)$ = ?

How to Keep Sheep Warm in a Snow Storm?

$Inst_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$Gaps_k(u,v)$ = min. number of gaps of $Inst_k(u,v)$ w.r.t $[u,v]$

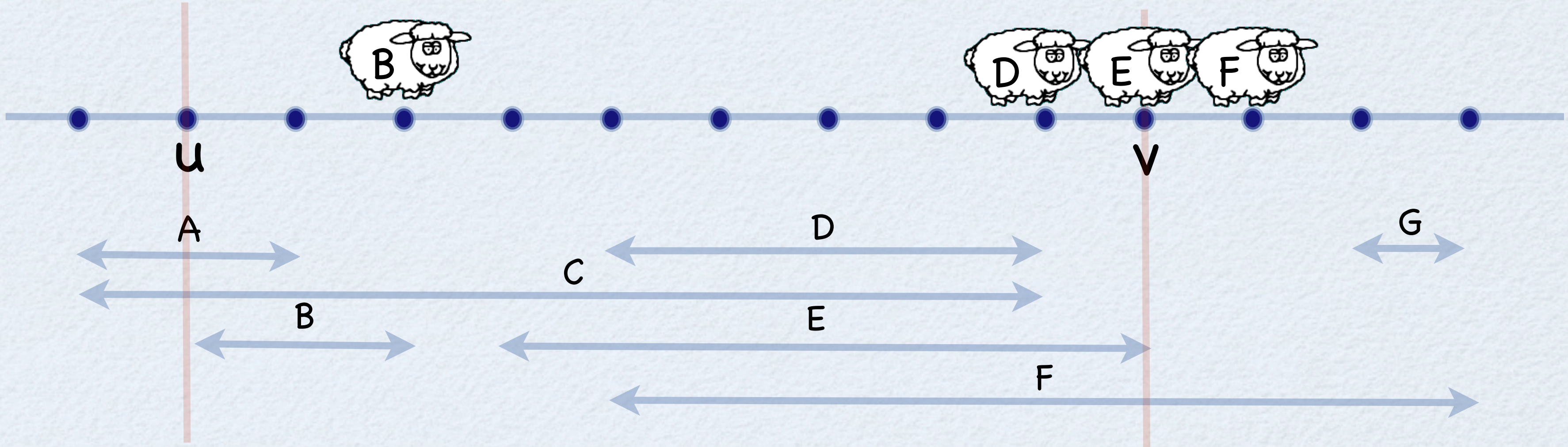


$Gaps_F(u,v)$ = ?

How to Keep Sheep Warm in a Snow Storm?

$\mathrm{Inst}_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$\mathrm{Gaps}_k(u,v)$ = min. number of gaps of $\mathrm{Inst}_k(u,v)$ w.r.t $[u,v]$

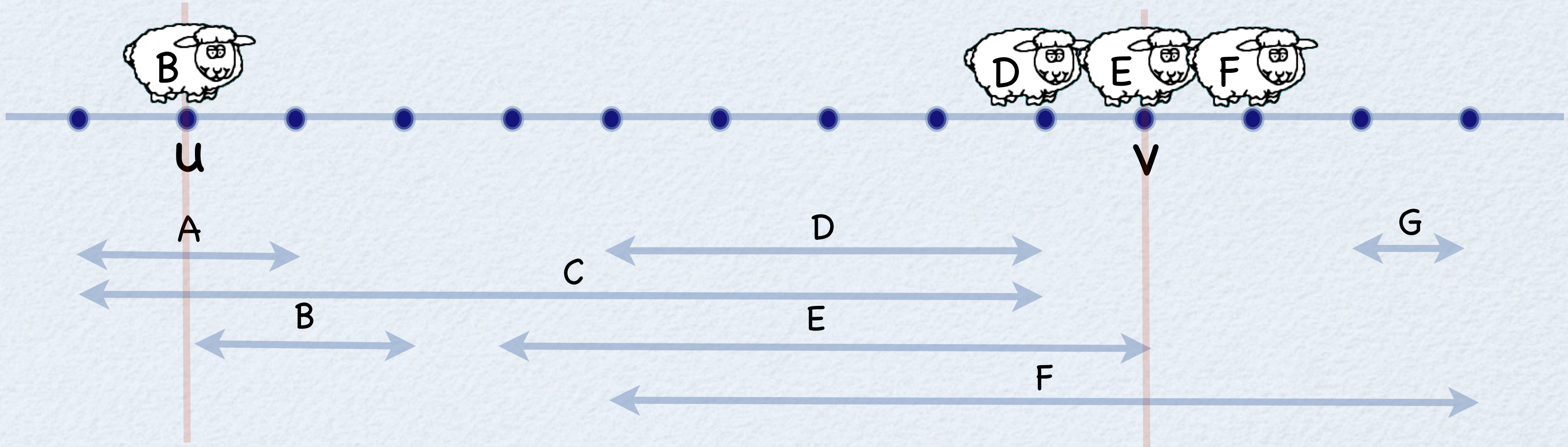


$\mathrm{Gaps}_F(u,v) =$?

How to Keep Sheep Warm in a Snow Storm?

$Inst_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$Gaps_k(u,v)$ = min. number of gaps of $Inst_k(u,v)$ w.r.t $[u,v]$

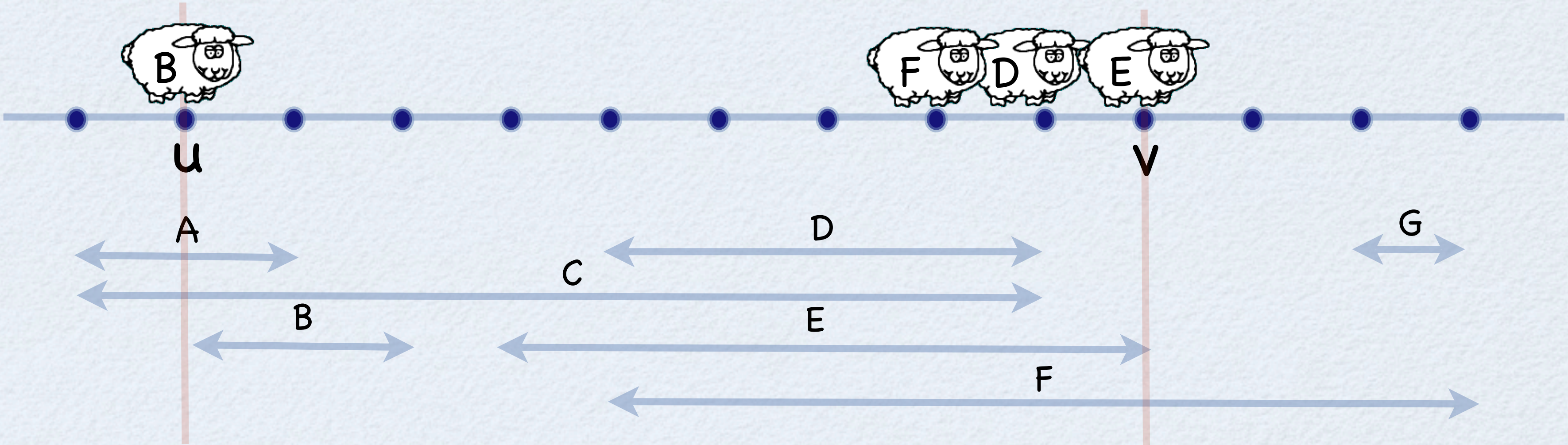


$Gaps_F(u,v)$ = ?

How to Keep Sheep Warm in a Snow Storm?

$\mathrm{Inst}_k(u,v)$ = all sheep $j \in \{1,2,\ldots,k\}$ for which $a_j \in [u,v]$

$\mathrm{Gaps}_k(u,v)$ = min. number of gaps of $\mathrm{Inst}_k(u,v)$ w.r.t $[u,v]$



$\mathrm{Gaps}_F(u,v)$ = 1

Recurrence for $\text{Gaps}_k(u,v)$

Recurrence for $\mathrm{Gaps}_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin \mathrm{Inst}_k(u,v)$, so

$$\mathrm{Gaps}_k(u,v) = \mathrm{Gaps}_{k-1}(u,v)$$

Recurrence for $Gaps_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin Inst_k(u,v)$, so

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

Case 2: $a_k \in [u,v]$, so $k \in Inst_k(u,v)$.

If k is scheduled at time t then

$$Inst_k(u,v) = \{k\} \cup Inst_{k-1}(u,t) \cup Inst_{k-1}(t+1,v)$$

How to Keep Sheep Warm in a Snow Storm?

Recurrence for $Gaps_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin Inst_k(u,v)$, so

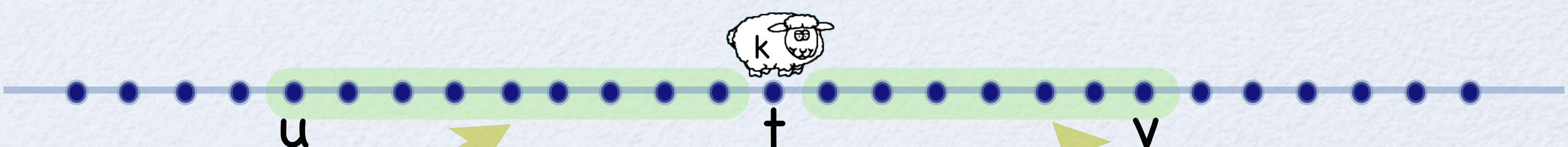
$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

Case 2: $a_k \in [u,v]$, so $k \in Inst_k(u,v)$.

If k is scheduled at time t then

$$Inst_k(u,v) = \{k\} \cup Inst_{k-1}(u,t) \cup Inst_{k-1}(t+1,v)$$

Philippe's Partition Principle: Wlog grouping looks like this:



$Inst_{k-1}(u,t) = Inst_{k-1}(u,t-1)$

$Inst_{k-1}(t+1,v)$

(in particular, no r_i at t, $i \neq k$)

Case 2: $a_k \in [u,v]$, proof of PPP.

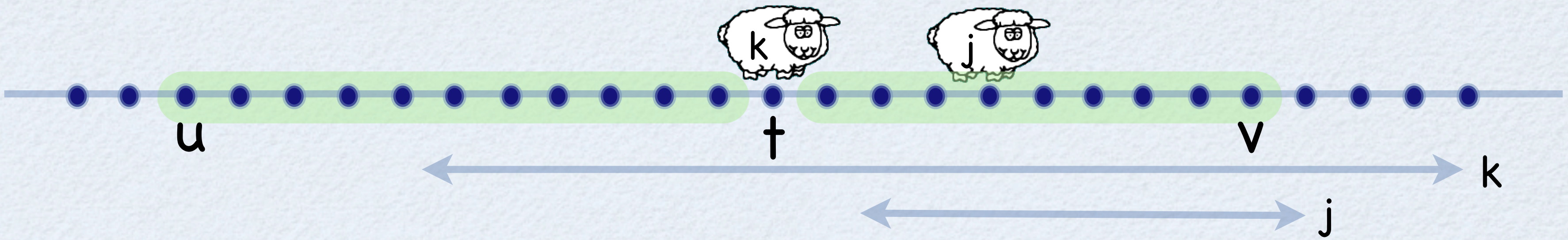
Fix an optimal grouping with maximum t (position of k)

How to Keep Sheep Warm in a Snow Storm?

Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum t (position of k)

If $j \in \mathrm{Inst}_{k-1}(t+1,v)$:

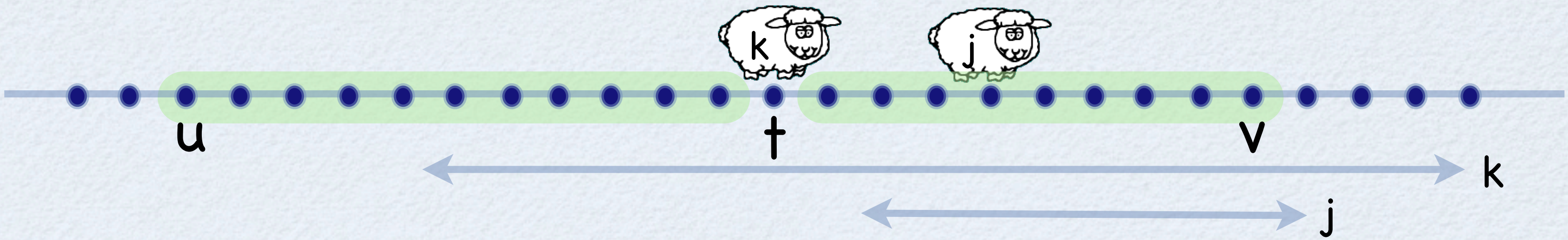


How to Keep Sheep Warm in a Snow Storm?

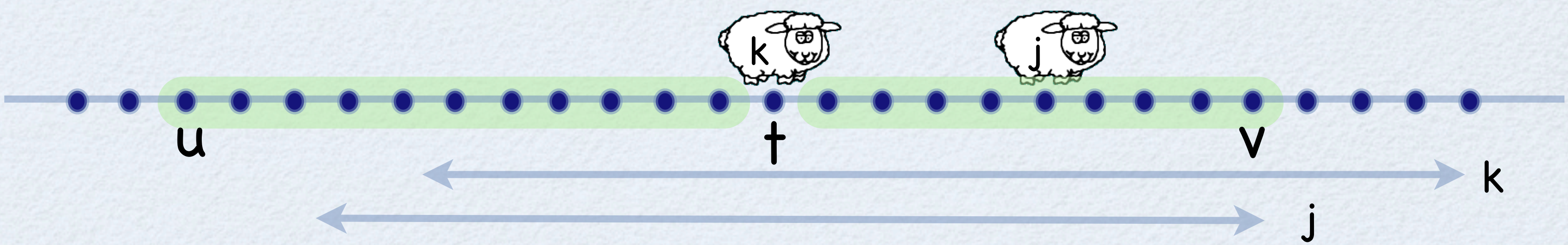
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum t (position of k)

If $j \in \text{Inst}_{k-1}(t+1,v)$:



If $j \in \text{Inst}_{k-1}(u,t)$:

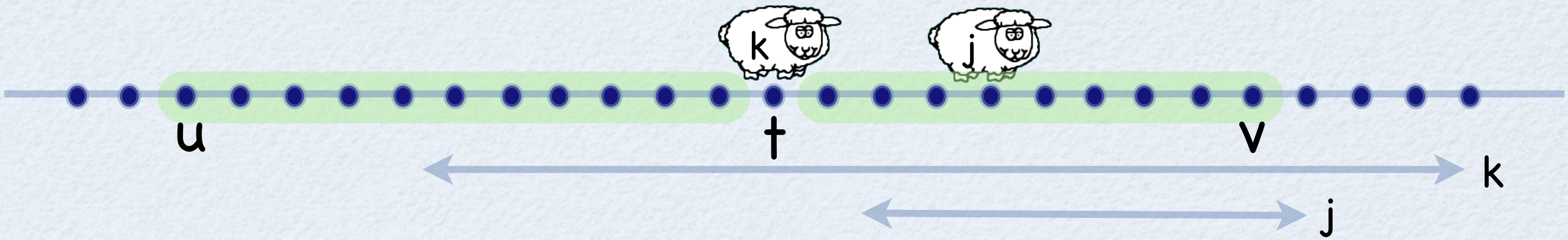


How to Keep Sheep Warm in a Snow Storm?

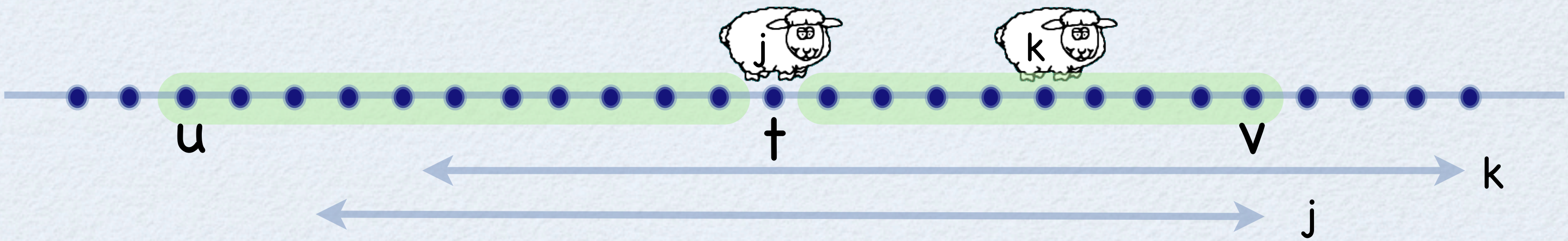
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum t (position of k)

If $j \in \mathrm{Inst}_{k-1}(t+1,v)$:



If $j \in \mathrm{Inst}_{k-1}(u,t)$:

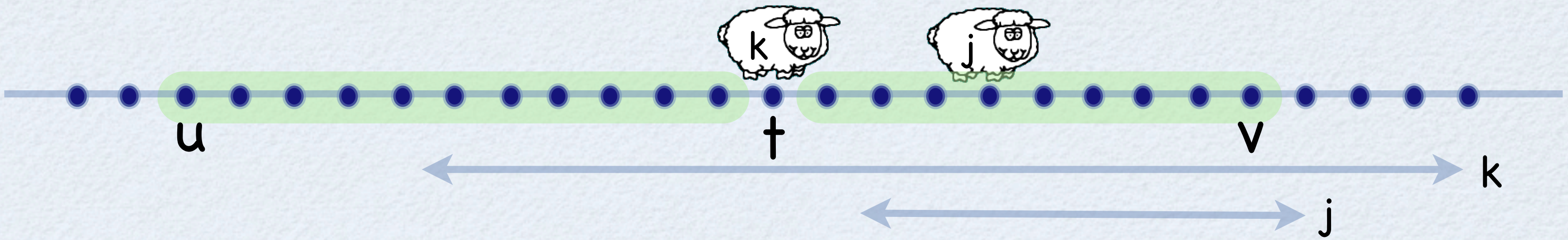


How to Keep Sheep Warm in a Snow Storm?

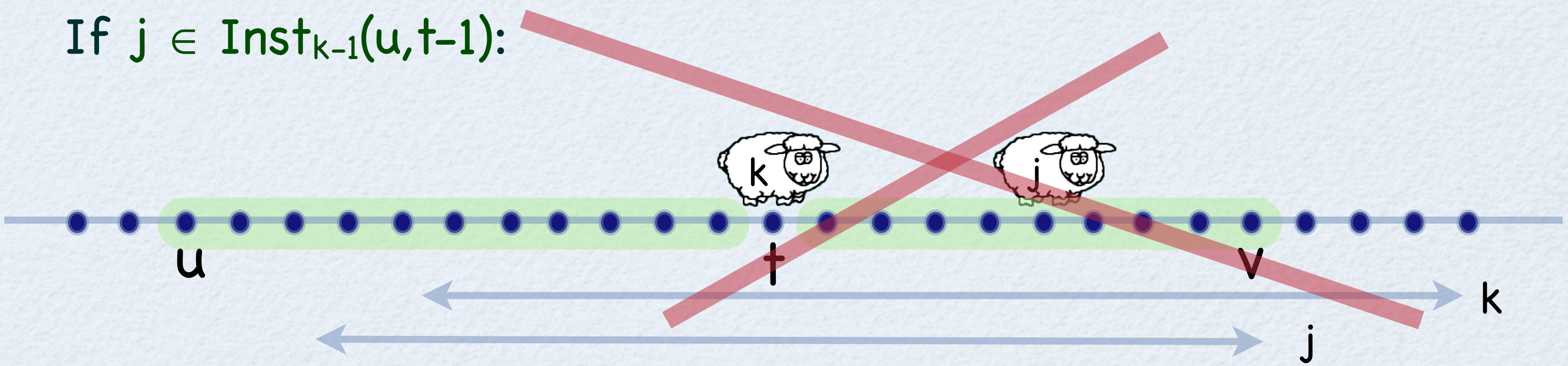
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum t (position of k)

If $j \in \mathrm{Inst}_{k-1}(t+1,v)$:



If $j \in \mathrm{Inst}_{k-1}(u,t-1)$:

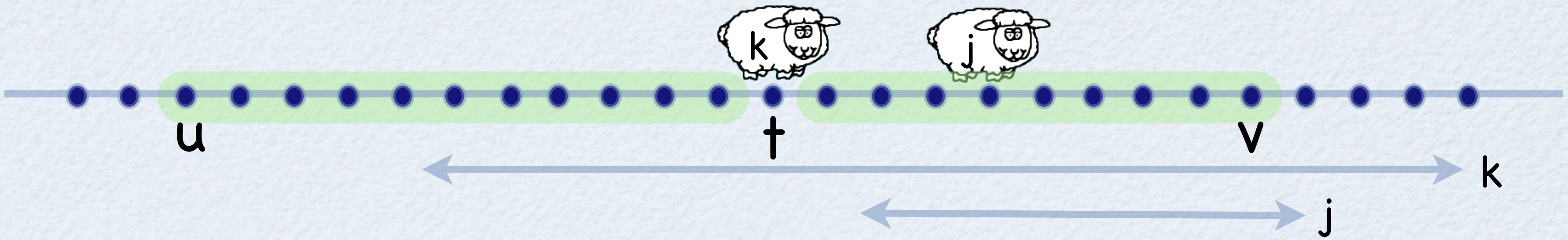


How to Keep Sheep Warm in a Snow Storm?

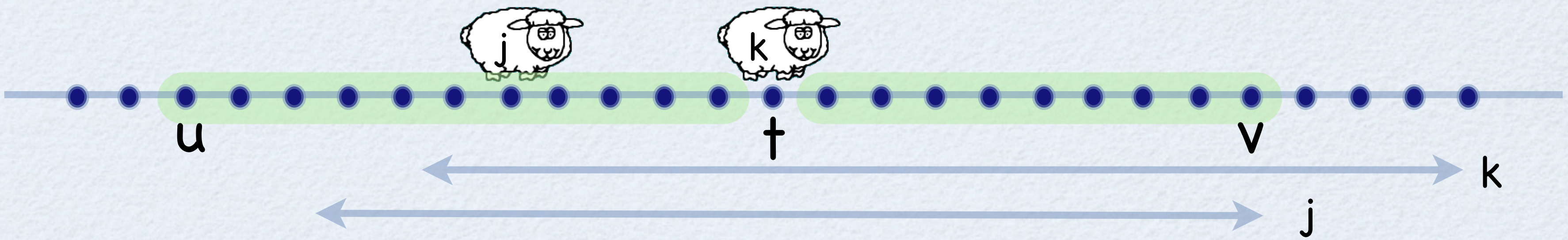
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum t (position of k)

If $j \in Inst_{k-1}(t+1,v)$:



If $j \in Inst_{k-1}(u,t-1)$:



How to Keep Sheep Warm in a Snow Storm?

Recurrence for $Gaps_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin Inst_k(u,v)$, so

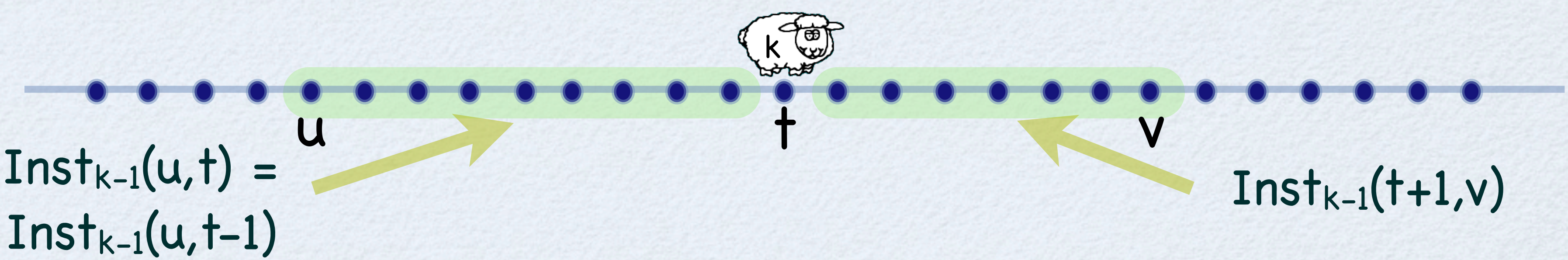
$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

Case 2: $a_k \in [u,v]$, so $k \in Inst_k(u,v)$.

If k is scheduled at time t then

$$Inst_k(u,v) = \{k\} \cup Inst_{k-1}(u,t) \cup Inst_{k-1}(t+1,v)$$

Philippe's Partition Principle: Wlog grouping looks like this:



How to Keep Sheep Warm in a Snow Storm?

Recurrence for $\mathrm{Gaps}_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin \mathrm{Inst}_k(u,v)$, so

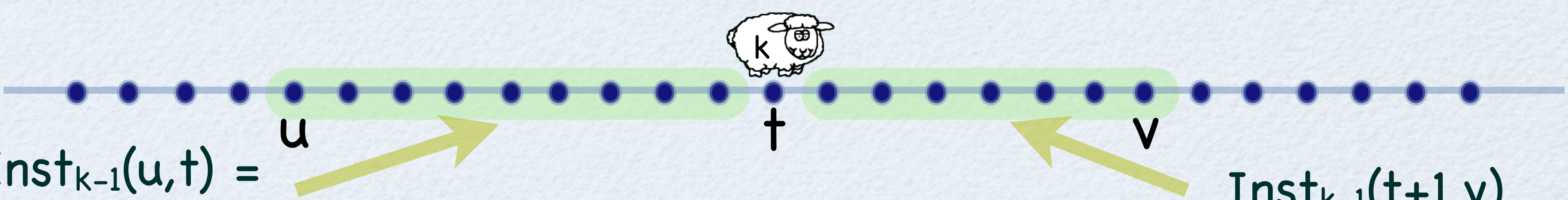
$$\mathrm{Gaps}_k(u,v) = \mathrm{Gaps}_{k-1}(u,v)$$

Case 2: $a_k \in [u,v]$, so $k \in \mathrm{Inst}_k(u,v)$.

If k is scheduled at time t then

$$\mathrm{Inst}_k(u,v) = \{k\} \cup \mathrm{Inst}_{k-1}(u,t) \cup \mathrm{Inst}_{k-1}(t+1,v)$$

Philippe's Partition Principle: Wlog grouping looks like this:



$\mathrm{Inst}_{k-1}(u,t) =$
$\mathrm{Inst}_{k-1}(u,t-1)$

$\mathrm{Inst}_{k-1}(t+1,v)$

So $\mathrm{Gaps}_k(u,v) = \mathrm{Gaps}_{k-1}(u,t-1) + \mathrm{Gaps}_{k-1}(t+1,v)$

How to Keep Sheep Warm in a Snow Storm?

Recurrence for $Gaps_k(u,v)$

Case 1: $a_k \notin [u,v]$. Then $k \notin Inst_k(u,v)$, so

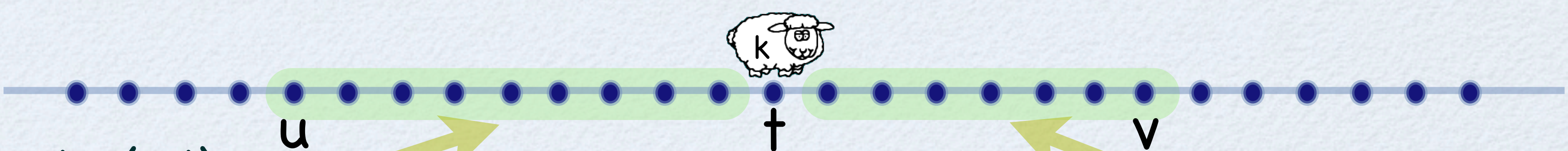
$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

Case 2: $a_k \in [u,v]$, so $k \in Inst_k(u,v)$.

If k is scheduled at time t then

$$Inst_k(u,v) = \{k\} \cup Inst_{k-1}(u,t) \cup Inst_{k-1}(t+1,v)$$

Philippe's Partition Principle: Wlog grouping looks like this:



$Inst_{k-1}(u,t) =$
$Inst_{k-1}(u,t-1)$

$Inst_{k-1}(t+1,v)$

So $Gaps_k(u,v) = Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v)$

What's t? Try all !!!

How to Keep Sheep Warm in a Snow Storm?

Algorithm B0:

if $a_k \notin [u,v]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

if $a_k \in [u,v]$ then

$$Gaps_k(u,v) = \min_t \{\ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v)\ \}$$

$$\text{where } a_k \le t \le \min(v,b_t)$$

Output $Gaps_n(\ a_{min}-1\ ,\ b_{max}+1\ )$

How to Keep Sheep Warm in a Snow Storm?

Algorithm B0:

if $a_k \notin [u,v]$ then

$$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$$

if $a_k \in [u,v]$ then

$$\text{Gaps}_k(u,v) = \min_t \{ \text{Gaps}_{k-1}(u,t-1) + \text{Gaps}_{k-1}(t+1,v) \}$$

$$\text{where } a_k \le t \le \min(v,b_t)$$

Output $\text{Gaps}_n(a_{min}-1 , b_{max}+1)$

Time $O((n\ k\text{'s})\cdot(R\ u\text{'s})\cdot(R\ v\text{'s})\cdot(R\ t\text{'s})) = O(nR^3)$ for $R = b_{max} - a_{min}$

How to Keep Sheep Warm in a Snow Storm?

Algorithm B0:

if $a_k \notin [u,v]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

if $a_k \in [u,v]$ then

$$Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v) \}$$

where $a_k \leq t \leq \min(v, b_t)$

Output $Gaps_n(a_{min}-1 , b_{max}+1)$

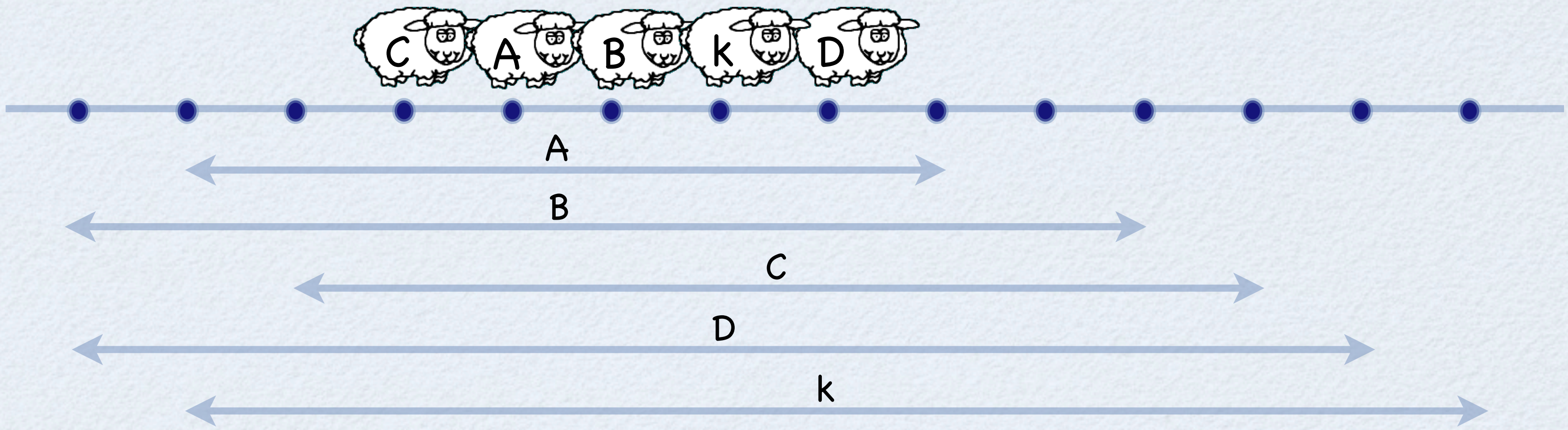
Time $O(\ (n\ k\text{'s}) \cdot (R\ u\text{'s}) \cdot (R\ v\text{'s}) \cdot (R\ t\text{'s})\ ) = O(nR^3)$ for $R = b_{max} - a_{min}$

Call set A *a minimizer set* if choosing u,v,t from A does not increase the solution

Can we find a smaller minimizer set?

How to Keep Sheep Warm in a Snow Storm?

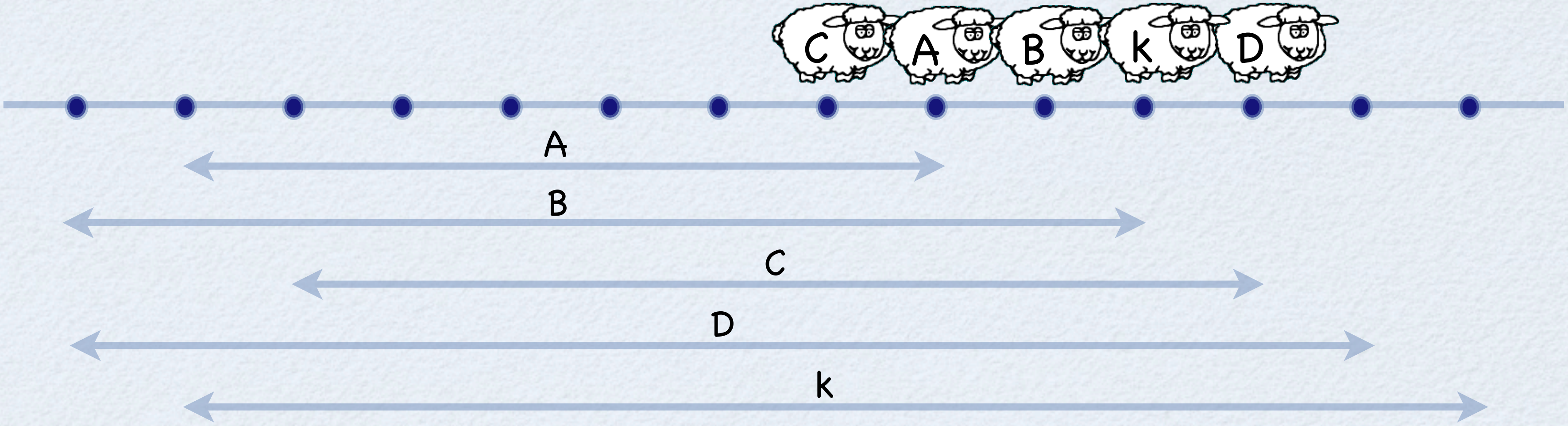
Reducing minimizer sets



How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets

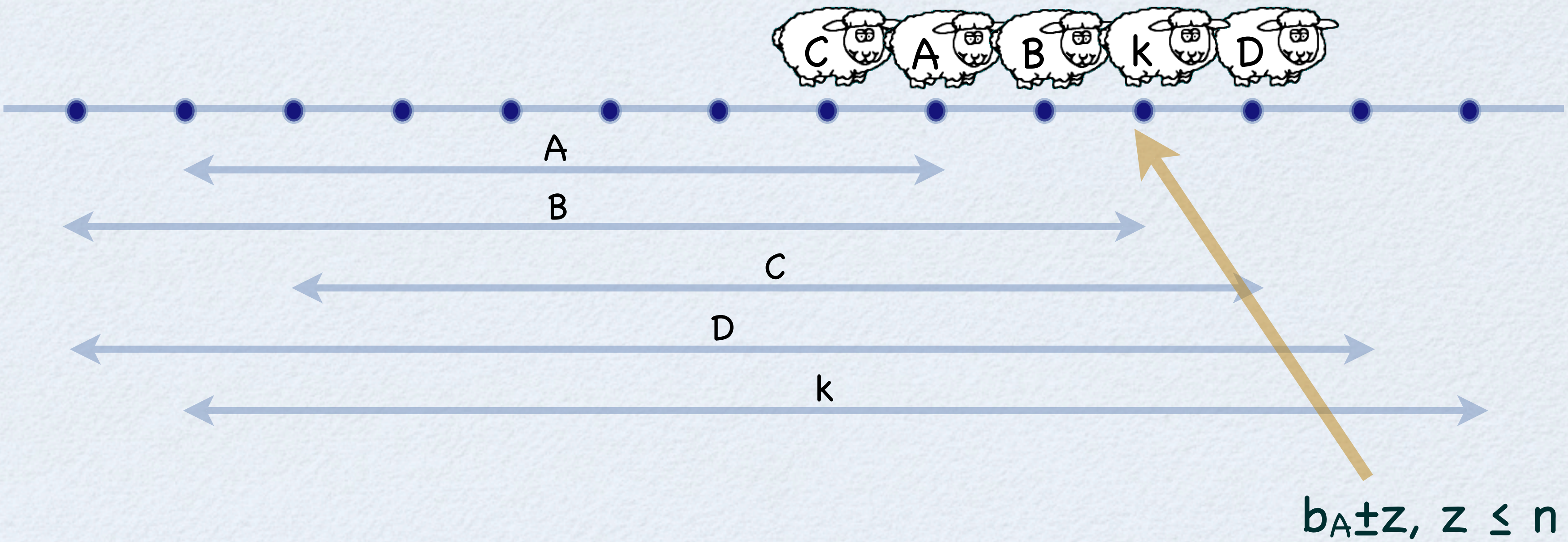
shift each group right:



How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets

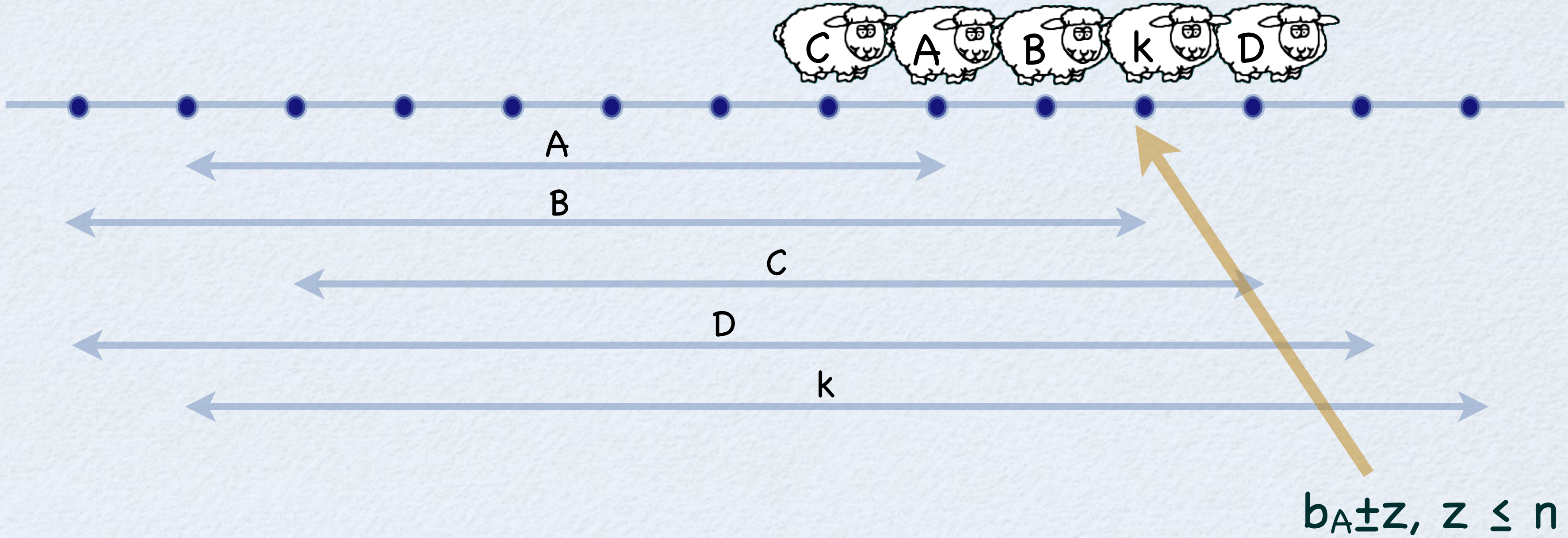
shift each group right:



How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets

shift each group right:



So $\{b_j \pm z\text{'s}\} = \{b_j \pm z : j, z \leq n\}$ is a minimizer set for t

$|\ \{b_j \pm z\text{'s}\}\ | = O(n^2)$

How to Keep Sheep Warm in a Snow Storm?

Algorithm B1:

if $a_k \notin [u,v]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

if $a_k \in [u,v]$ then

$$Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v) \}$$

$$\text{where } a_k \leq t \leq \min(v,b_t)$$

Output $Gaps_n(a_{min}-1 , b_{max}+1)$

Choose u,v,t from $\{b_j \pm z's\}$

So running time = $O(n \cdot n^2 \cdot n^2 \cdot n^2) = O(n^7)$ [Baptiste, SODA'06]

Minimum Energy Scheduling - Overview

Minimum Energy Scheduling

Instance: collection of n jobs

- job j has release time r_j, deadline d_j, processing time p_j
- 1 unit of processing costs 1 unit of energy
- turning the system on costs L units of energy

Schedule = when each job is processed and when the processor is on

Objective: Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Minimum Energy Scheduling

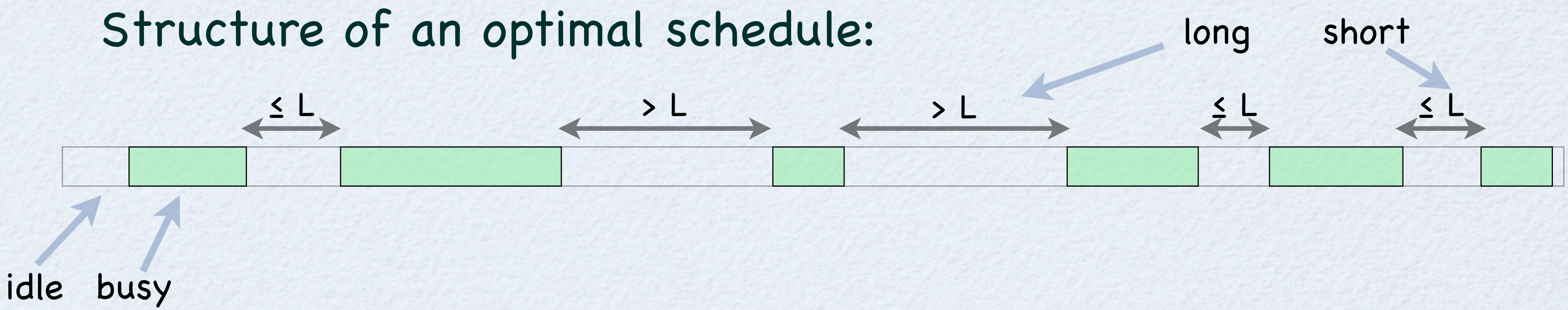
Instance: collection of n jobs

- job j has release time r_j, deadline d_j, processing time p_j
- 1 unit of processing costs 1 unit of energy
- turning the system on costs L units of energy

Schedule = when each job is processed and when the processor is on

Objective: Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Structure of an optimal schedule:



Minimum Energy Scheduling

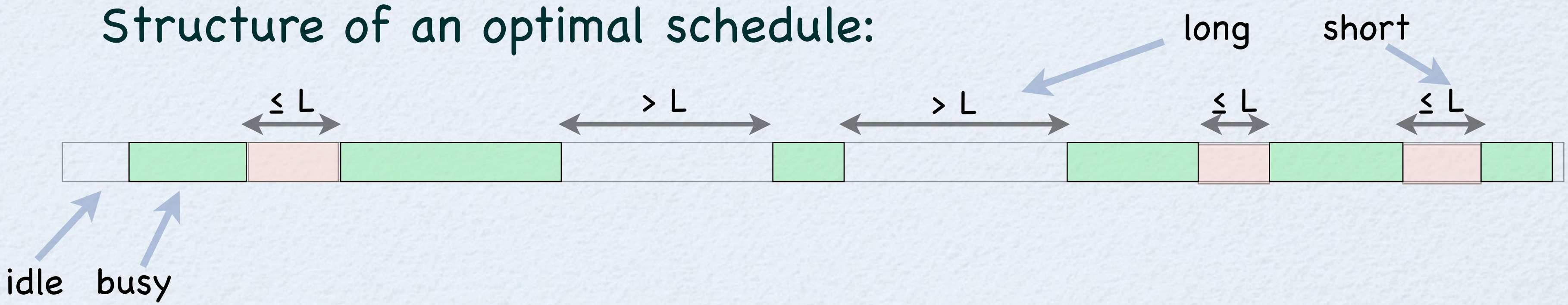
Instance: collection of n jobs

- job j has release time r_j, deadline d_j, processing time p_j
- 1 unit of processing costs 1 unit of energy
- turning the system on costs L units of energy

Schedule = when each job is processed and when the processor is on

Objective: Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Structure of an optimal schedule:



Minimum Energy Scheduling

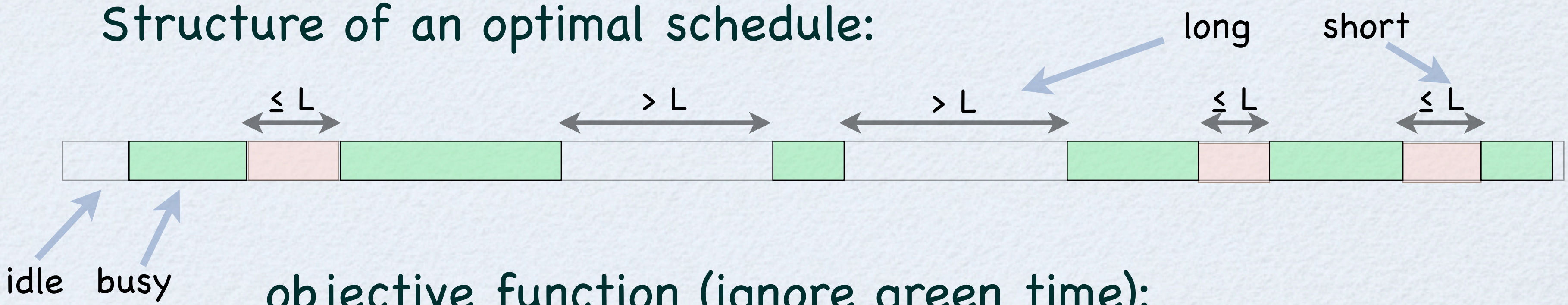
Instance: collection of n jobs

- job j has release time r_j, deadline d_j, processing time p_j
- 1 unit of processing costs 1 unit of energy
- turning the system on costs L units of energy

Schedule = when each job is processed and when the processor is on

Objective: Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Structure of an optimal schedule:



objective function (ignore green time):

$$E = \Sigma_{\text{gaps } g} \min(\text{ length}(g) , L)$$

Minimum Energy Scheduling

What sheep have to do with it?

For $L = 1$ and all $p_j = 1$:

- E = # of gaps
- $\boxed{j}$ = 
- $r_j = a_j$ and $d_j = b_j$

Minimum Energy Scheduling

What sheep have to do with it?

For $L = 1$ and all $p_j = 1$:

- E = # of gaps
- $\boxed{j}$ = 
- $r_j = a_j$ and $d_j = b_j$

So the case (unit jobs, $L \leq 1$) can be solved in time $O(n^7)$

Minimum Energy Scheduling

What's known?

Posed as open: Sviridenko [05], Irani, Pruhs [05]

# proc.	L	p_j	assumption	time
1	any	1		$O(n^7)$ [B'06] $O(n^4)$ [BCD'08]
1	any	any		$O(n^5)$ [BCD'08]
m	any	1		$O(n^7m^5)$ [DG...'07]
1	1	any	agreeable	$O(n \log n)$ [GJS'10]
1	any	1	agreeable	$O(n^3)$ [GJS'10]
m	1	1	agreeable	$O(n^3m^2]$ [GJS'10]

[B'06] = Baptiste

[BCD'08] = Baptiste, Chrobak, Dürr

[DG...'07] = Demaine, Ghodsi, Hajiaghayi, Sayedi-Roshkhar, Zadimoghaddam

[GJS'10] = Gururaj, Jalan, Stein

Minimum Energy Scheduling

What's known?

Posed as open: Sviridenko [05], Irani, Pruhs [05]

# proc.	L	p_j	assumption	time
1	any	1		$O(n^7)$ [B'06] $O(n^4)$ [BCD'08]
1	any	any		$O(n^5)$ [BCD'08]
m	any	1		$O(n^7m^5)$ [DG...'07]
1	1	any	agreeable	O(nlogn) [GJS'10]
1	any	1	agreeable	$O(n^3)$ [GJS'10]
m	1	1	agreeable	$O(n^3m^2]$ [GJS'10]

$r_i < r_j \Leftrightarrow d_j < d_j$

[B'06] = Baptiste

[BCD'08] = Baptiste, Chrobak, Dürr

[DG...'07] = Demaine, Ghodsi, Hajiaghayi, Sayedi-Roshkhar, Zadimoghaddam

[GJS'10] = Gururaj, Jalan, Stein

Minimum Energy Scheduling – Techniques

Minimum Energy Scheduling

Main techniques:

* Philippe's partitioning trick √
* Reducing the minimizer sets
* Inversion trick ("large" parameter ⇔ "small" value)
* $O(n^2)$-time reduction: Energy $\preccurlyeq$ Gaps

Reducing Minimizer Sets ($L=1$, $p_j=1$)

Algorithm B1-L1P1:

if $r_k \notin [u,v-1]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

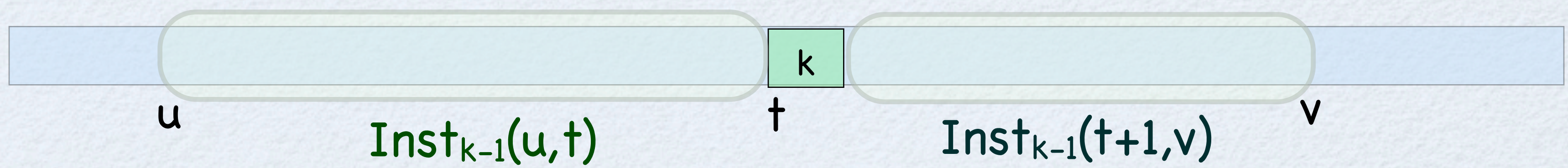
if $r_k \in [u,v-1]$ then

$$Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t) + Gaps_{k-1}(t+1,v) \}$$

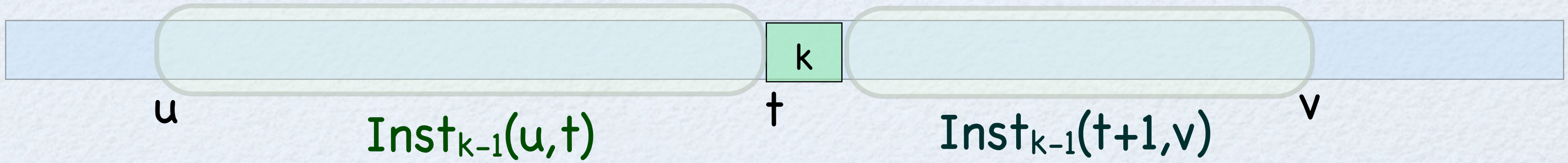
where $r_k \leq t < \min(v,d_t)$

Output $Gaps_n(\ r_{min}-1\ ,\ d_{max}+1\ )$

Reminder: $u,v,t \in \{d_j \pm z\}$, $|\ \{d_j \pm z\}\ | = O(n^2)$

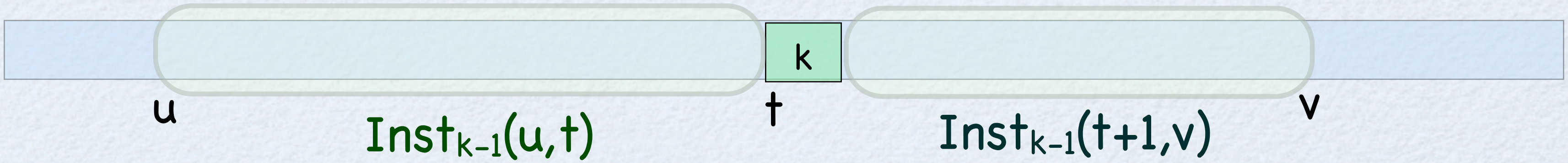


Reducing Minimizer Sets ($L=1$, $p_j=1$)



Three WLOG observations:

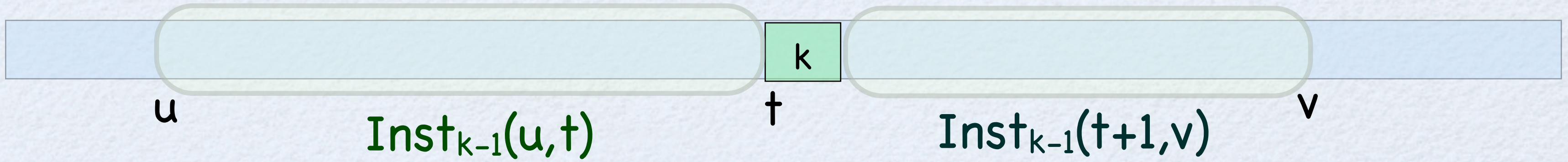
Reducing Minimizer Sets (L=1, p_j=1)



Three WLOG observations:

- t is maximum possible

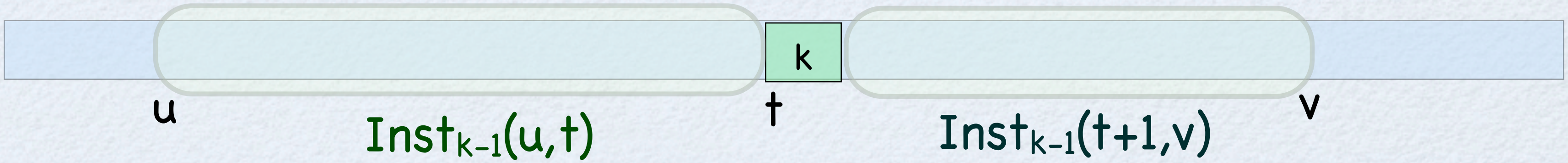
Reducing Minimizer Sets ($L=1$, $p_j=1$)



Three WLOG observations:

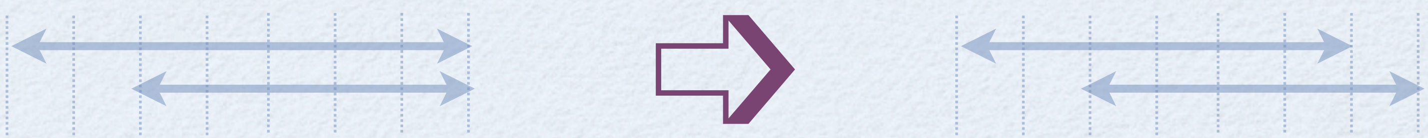
- t is maximum possible
- all deadlines are different

Reducing Minimizer Sets ($L=1$, $p_j=1$)

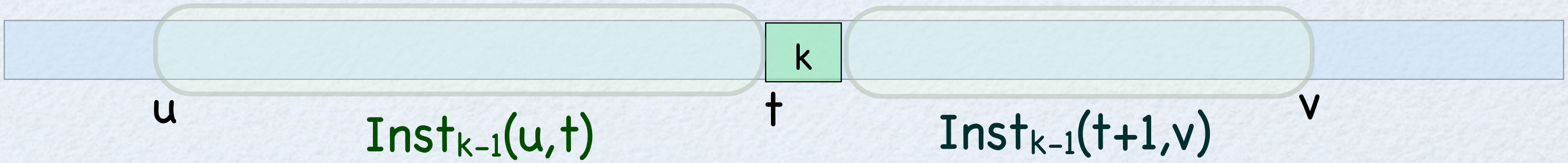


Three WLOG observations:

- t is maximum possible
- all deadlines are different

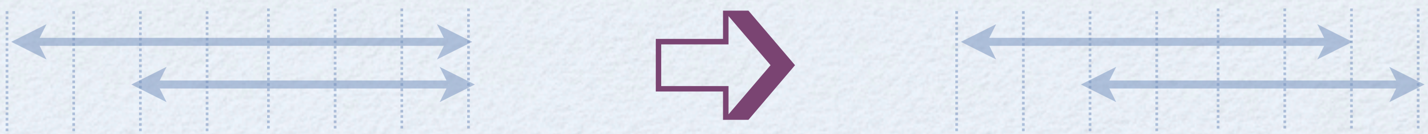


Reducing Minimizer Sets ($L=1$, $p_j=1$)



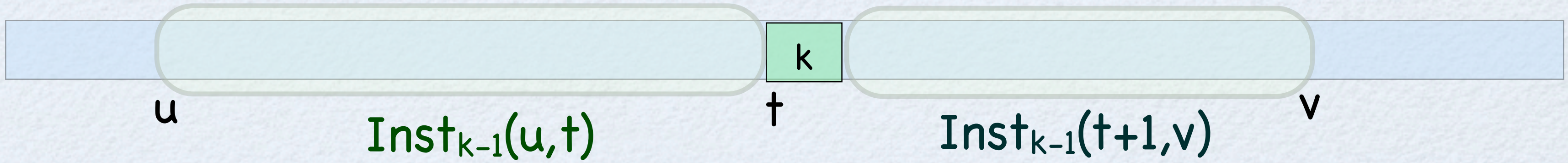
Three WLOG observations:

- t is maximum possible
- all deadlines are different



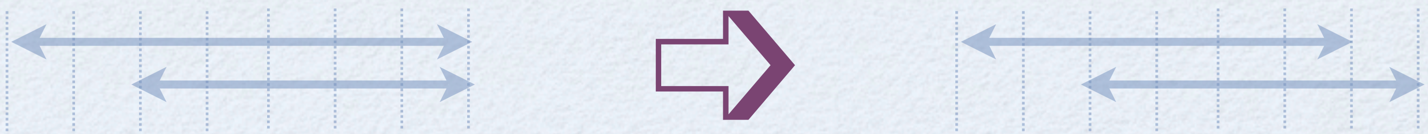
- release times are different

Reducing Minimizer Sets ($L=1$, $p_j=1$)



Three WLOG observations:

- t is maximum possible
- all deadlines are different



- release times are different
- $v \leq d_k+1$

Reducing Minimizer Sets ($L=1$, $p_j=1$)

Assume $k \in Inst_k(u,v)$ and k scheduled at t (latest possible)

Reducing Minimizer Sets ($L=1$, $p_j=1$)

Assume $k \in \mathrm{Inst}_k(u,v)$ and k scheduled at t (latest possible)

If k scheduled last then $\mathrm{Inst}_{k-1}(t+1,v)$ is empty

So assume k is not last

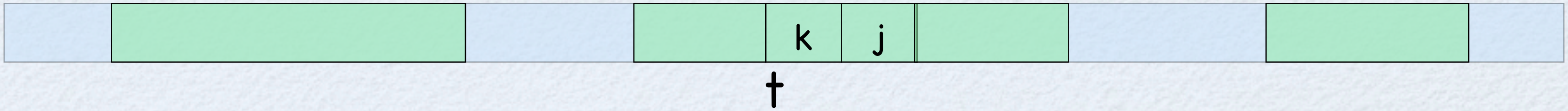
Reducing Minimizer Sets ($L=1$, $p_j=1$)

Assume $k \in Inst_k(u,v)$ and k scheduled at t (latest possible)

If k scheduled last then $Inst_{k-1}(t+1,v)$ is empty

So assume k is not last

Claim: There is j right after k



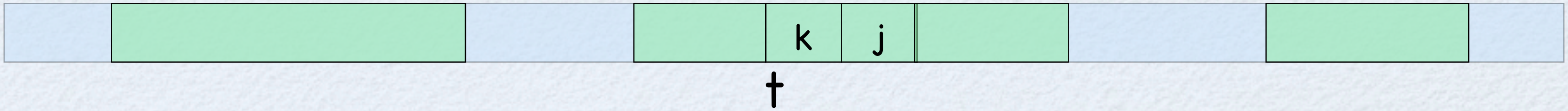
Reducing Minimizer Sets (L=1, p_j=1)

Assume $k \in Inst_k(u,v)$ and k scheduled at t (latest possible)

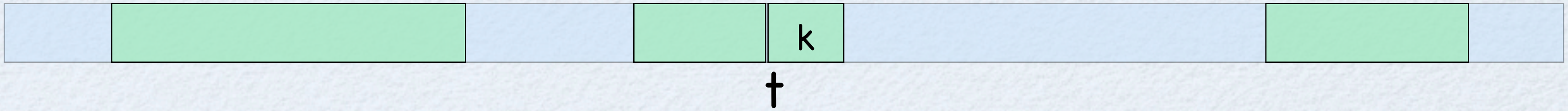
If k scheduled last then $Inst_{k-1}(t+1,v)$ is empty

So assume k is not last

Claim: There is j right after k



Proof: If not

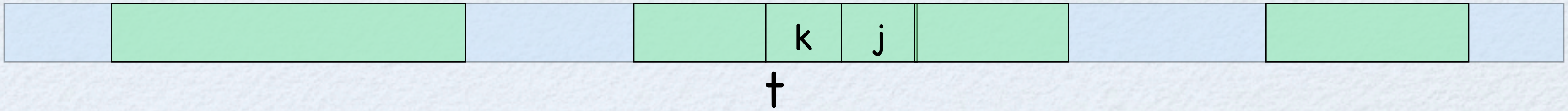


Reducing Minimizer Sets ($L=1$, $p_j=1$)

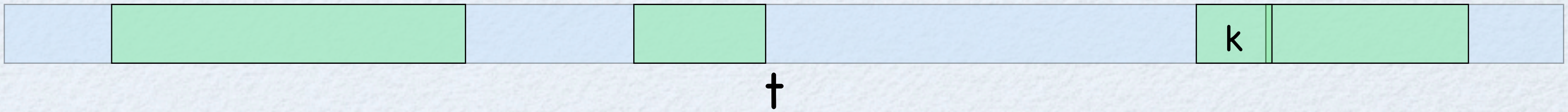
Assume $k \in Inst_k(u,v)$ and k scheduled at t (latest possible)

If k scheduled last then $Inst_{k-1}(t+1,v)$ is empty
So assume k is not last

Claim: There is j right after k



Proof: If not



move k to later (possible, because d_k is largest)
-- contradiction

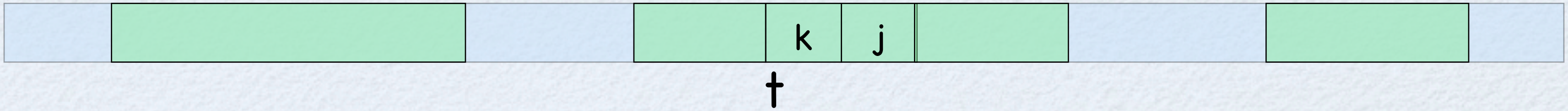
Reducing Minimizer Sets ($L=1$, $p_j=1$)

Assume $k \in Inst_k(u,v)$ and k scheduled at t (latest possible)

If k scheduled last then $Inst_{k-1}(t+1,v)$ is empty

So assume k is not last

Claim: There is j right after k



From claim: $t = r_j-1$

so

- minimizer set for t's = $\{r_j-1\text{'s}\}$
- minimizer set for u's = $\{r_j\text{s}\}$

Reducing Minimizer Sets ($L=1$, $p_j=1$)

Algorithm B2-L1P1:

if $r_k \notin [u,v-1]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

if $r_k \in [u,v-1]$ then

$$Gaps_k(u,v) = \min \begin{cases} Gaps_{k-1}(u,v-1) \\ \min_t \{ Gaps_{k-1}(u,t) + Gaps_{k-1}(t+1,v) \} \\ \text{where } a_k \leq t < \min(v,d_t) \end{cases}$$

Output $Gaps_n(r_{min}-1 , d_{max}+1)$

Above, choose: $u \in \{r_j\text{'s}\}$, $t \in \{r_j+1\text{'s}\}$ and $v \in \{d_j \pm z\text{'s}\}$

$\Rightarrow$ Running time = $O(\ (n\ k\text{'s}) \cdot (n\ u\text{'s}) \cdot (n^2\ v\text{'s}) \cdot (n\ t\text{'s})\ ) = O(n^5)$

Minimum Energy Scheduling

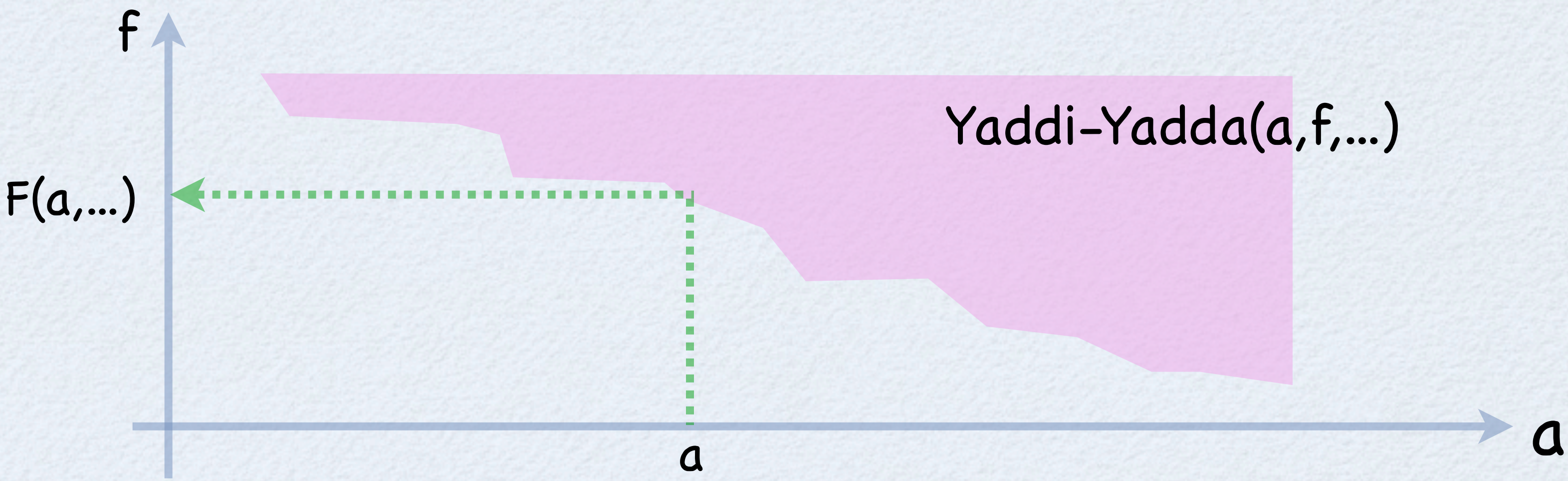
Main techniques:

- Philippe's partitioning trick ✓
- Reducing the minimizer sets ✓
- Inversion trick ("large" parameter ⇔ "small" value)
- $O(n^2)$-time reduction: Energy $\preccurlyeq$ Gaps

Inversion Trick

Consider a function F(a,…) = min{f : Yaddi-Yadda(a,f,…)} s.t.

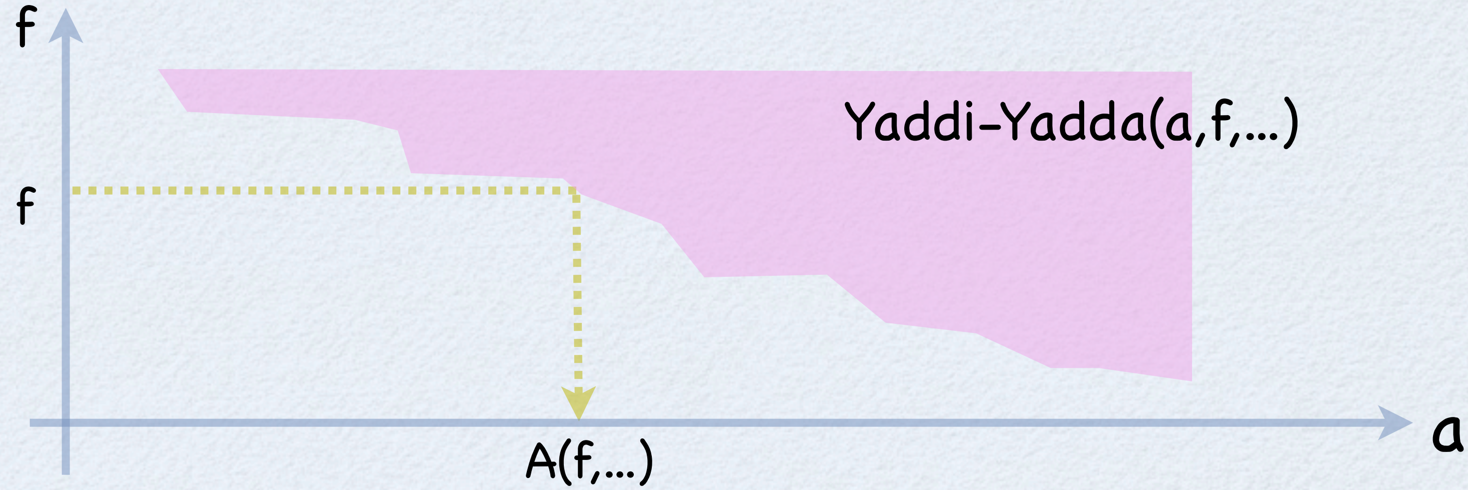
- F(a,…) is monotone w.r.t. a
- range of a is large (exponential)
- range of F(a,…) is small (polynomial)



Inversion Trick

Consider a function F(a,...) = min{f : Yaddi-Yadda(a,f,...)} s.t.

- F(a,...) is monotone w.r.t. a
- range of a is large (exponential)
- range of F(a,...) is small (polynomial)



Instead compute A(f,...) = min{a : Yaddi-Yadda(a,f,...)}

and then F(a,...) = min{f : A(f,...) $\geq$ a}
(binary search ....)

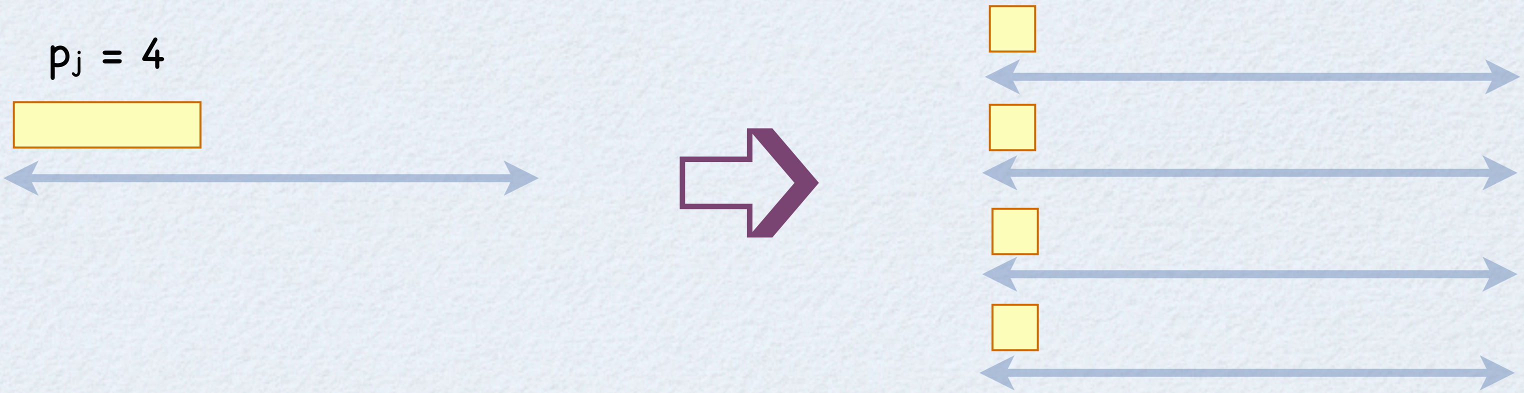
Inversion Trick (any p_j, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps,unit jobs) to arbitrary processing times

Inversion Trick (any p_j, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps,unit jobs) to arbitrary processing times

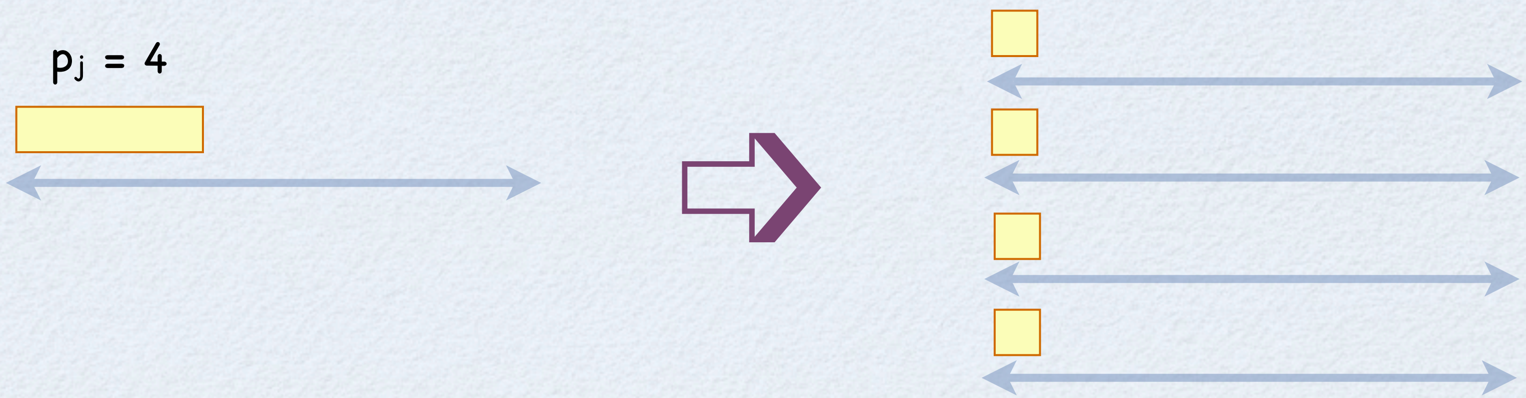
Obvious approach: break each job into unit jobs



Inversion Trick (any p_j, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps,unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs



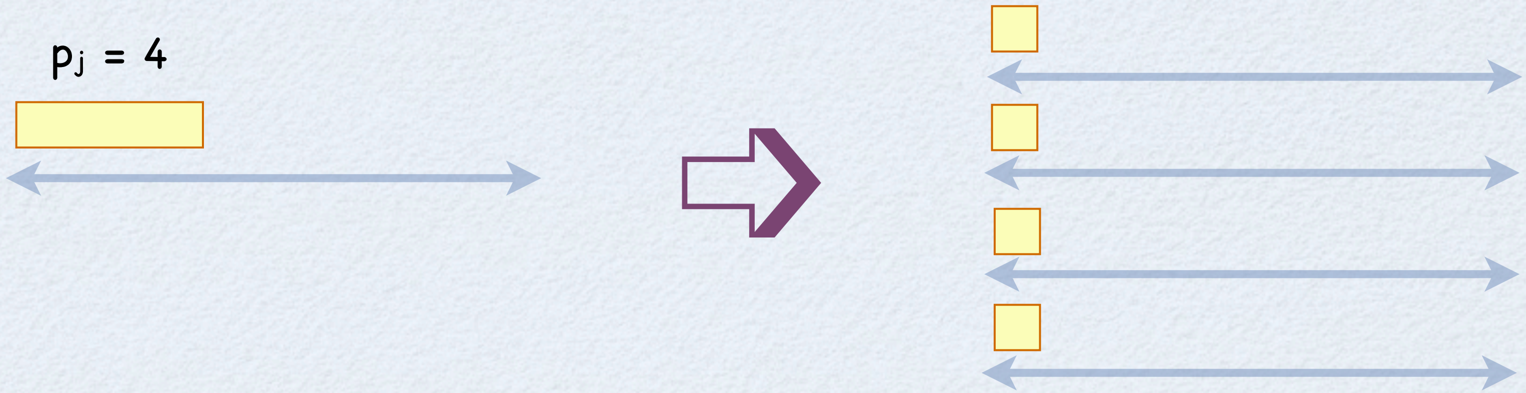
This leads to:

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $G_{k,p}(u,v)$ = minimum # gaps w.r.t. [u,v] for $Inst_{k,p}(u,v)$

Inversion Trick (any p_j, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps,unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs



This leads to:

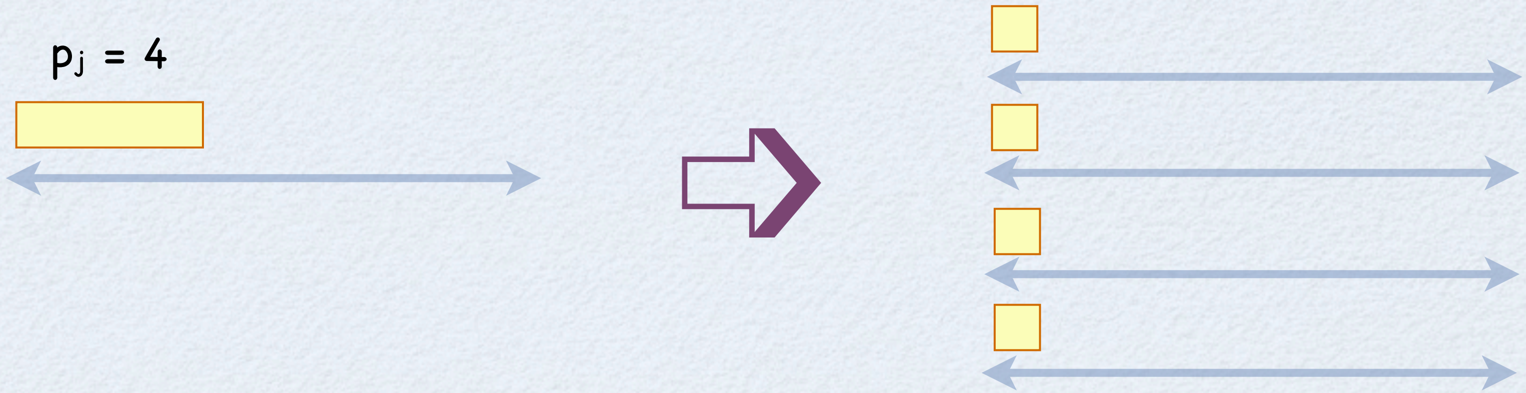
- $\mathrm{Inst}_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $G_{k,p}(u,v)$ = minimum # gaps w.r.t. [u,v] for $\mathrm{Inst}_{k,p}(u,v)$

We can apply Algorithm 2 but ... range of p not polynomial

Inversion Trick (any p_j, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps,unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs



This leads to:

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $G_{k,p}(u,v)$ = minimum # gaps w.r.t. $[u,v]$ for $Inst_{k,p}(u,v)$

So we invert:

$A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\le$ g gaps

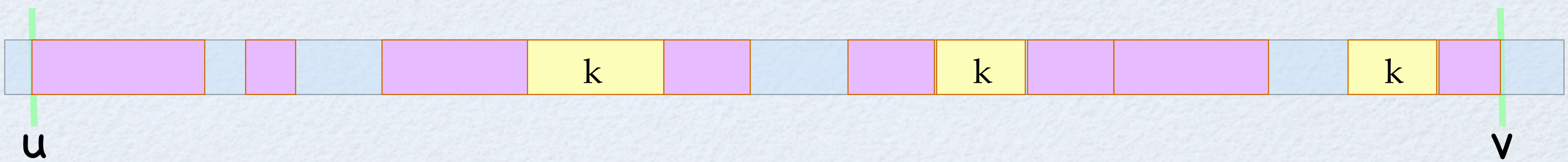
Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which
  $Inst_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Inversion Trick (any p_j, gaps)

- $\mathrm{Inst}_{k,p}(u,v)$ = jobs 1,2,…,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $\mathrm{Inst}_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

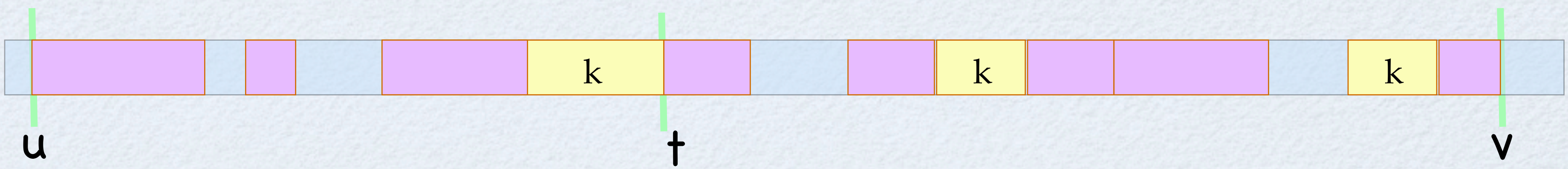
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

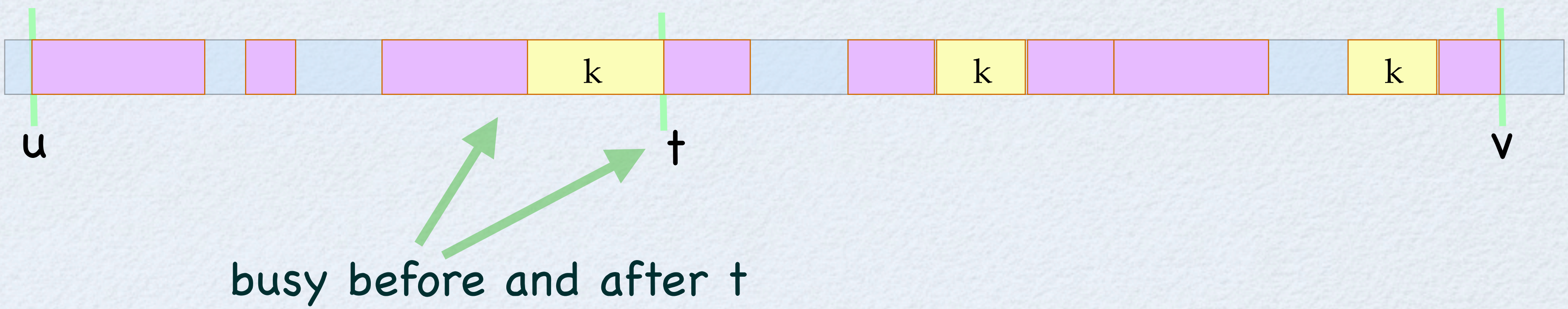
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $\mathrm{Inst}_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $\mathrm{Inst}_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

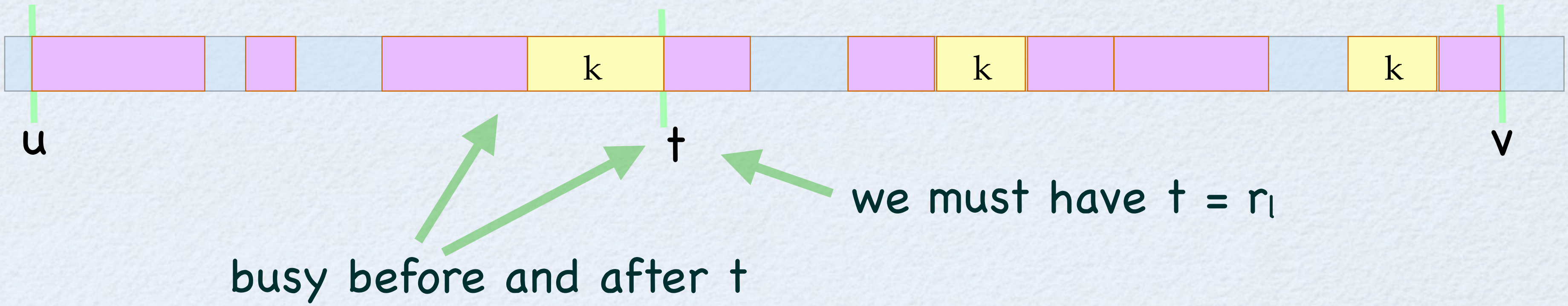
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

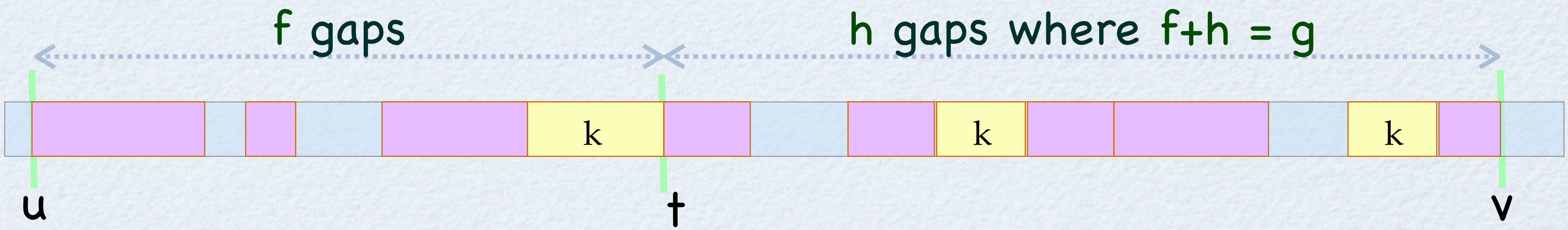
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

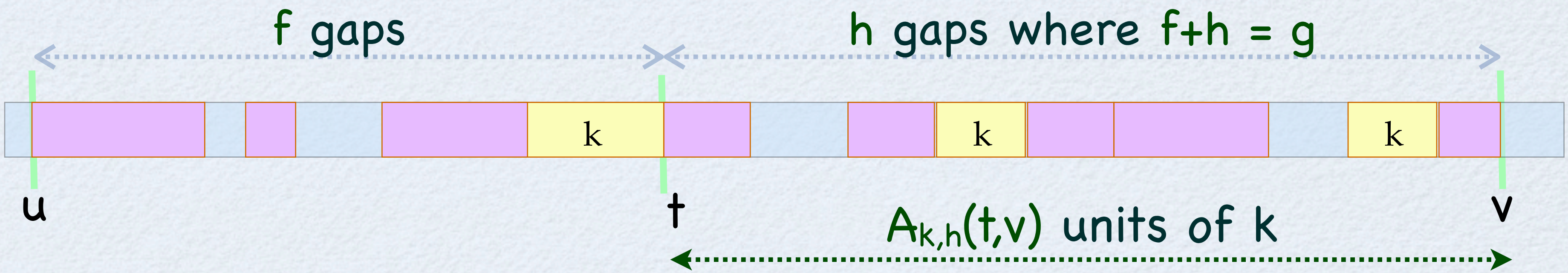
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs $1,2,\ldots,k$ with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

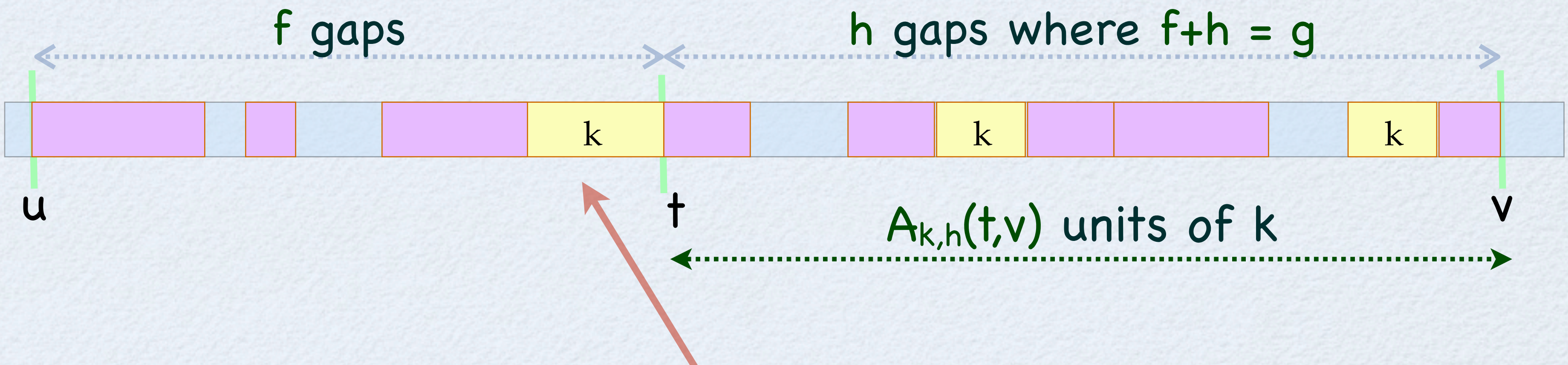
Assume not whole k executed at the end:



Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,…,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with ≤ g gaps

Assume not whole k executed at the end:

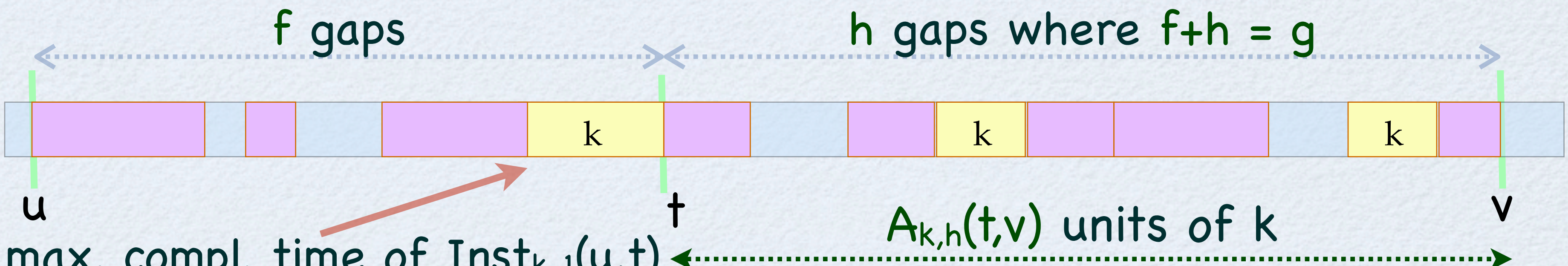


minimum amount of k
with f gaps before t

Inversion Trick (any p_j, gaps)

- $\text{Inst}_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

Assume not whole k executed at the end:

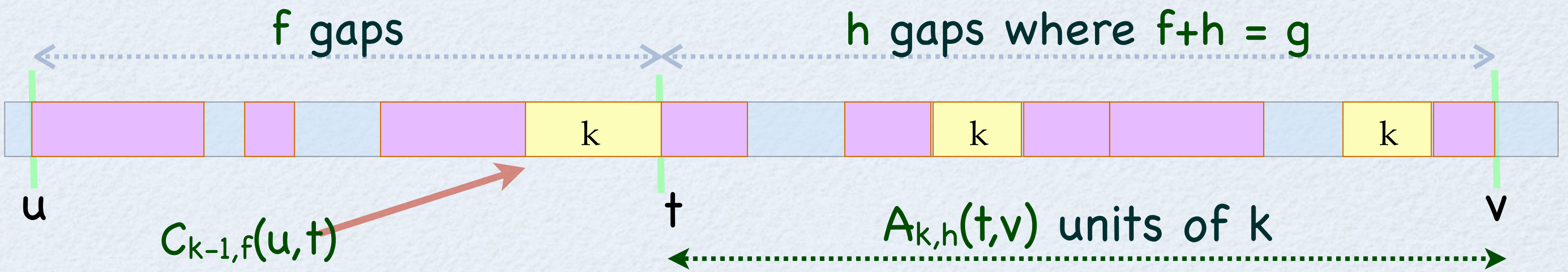


max. compl. time of $\text{Inst}_{k-1}(u,t)$
with f gaps = $C_{k-1,f}(u,t)$

Inversion Trick (any p_j, gaps)

- $Inst_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount p of job k for which $Inst_{k,p}(u,v)$ has a schedule with $\leq$ g gaps

Assume not whole k executed at the end:



$$A_{k,g}(u,v) = \min_t \min_{f+h=g} \left(t - C_{k-1,f}(u,t) + A_{k,h}(t,v) \right)$$

where $t \in \{r_j\}$, $u \in \{r_j\}$, $v \in \{d_j \pm z\}$

Also, we need recurrence for $C_{k,f}(u,v)$ using A(...)

Running time $O(n^7)$

Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$G_k(u,v) = \min_t \{ \text{\# gaps : yaddi yadda} \}$

$O(n)$ $O(n)$ $O(n^2)$ $O(n)$ $O(n)$ values

Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$$G_k(u,v) = \min_t \{ \text{\# gaps : yaddi yadda} \}$$

$O(n)$ $O(n)$ $O(n^2)$ $O(n)$ $O(n)$ values

Invert: compute

$$V_k(u,g) = \max \{ v : \text{Inst}_k(u,v) \text{ has schedule with } g \text{ gaps} \}$$

Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$$G_k(u,v) = \min_t \{\ \#\ \text{gaps} : \text{yaddi yadda}\ \}$$

$O(n)$ $O(n)$ $O(n^2)$ $O(n)$ $O(n)$ values

Invert: compute

$$V_k(u,g) = \max \{\ v : \text{Inst}_k(u,v) \text{ has schedule with } g \text{ gaps}\ \}$$

Gives $O(n^4)$ [BCD'08]

Can be extended to any p_j's in time $O(n^5)$ [BCD'08]

Minimum Energy Scheduling

Main techniques:

- Philippe's partitioning trick ✓
- Reducing the minimizer sets ✓
- Inversion trick ("large" parameter ⇔ "small" value) ✓
- $O(n^2)$-time reduction: Energy $\preccurlyeq$ Gaps

Reduction: Energy $\preccurlyeq$ Gaps

S = lex-minimal energy-optimal schedule
[u,v) = short gap in S

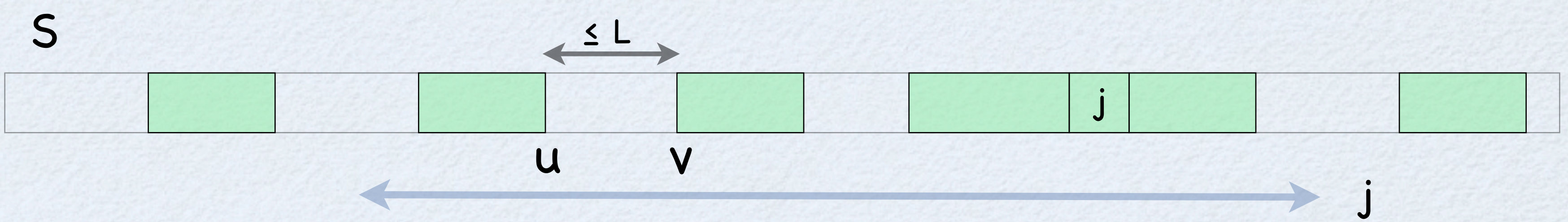
Claim: wlog, if $r_j < v$ then j is executed before v

Reduction: Energy $\preccurlyeq$ Gaps

S = lex-minimal energy-optimal schedule
[u,v) = short gap in S

Claim: wlog, if $r_j < v$ then j is executed before v

Proof: suppose not



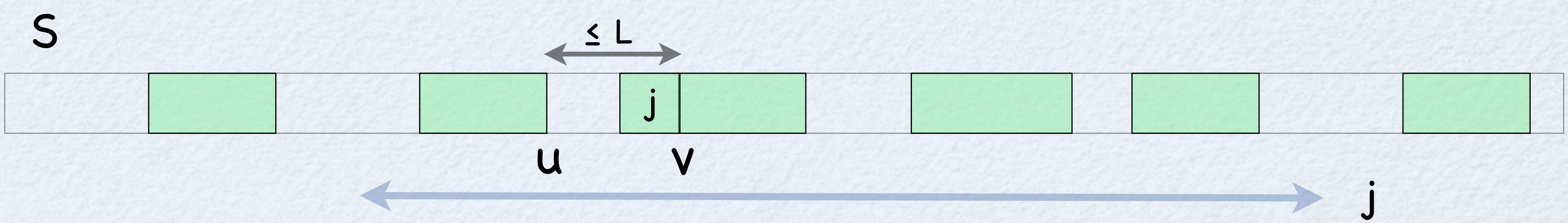
Reduction: Energy ≼ Gaps

S = lex-minimal energy-optimal schedule

[u,v) = short gap in S

Claim: wlog, if $r_j < v$ then j is executed before v

Proof: suppose not

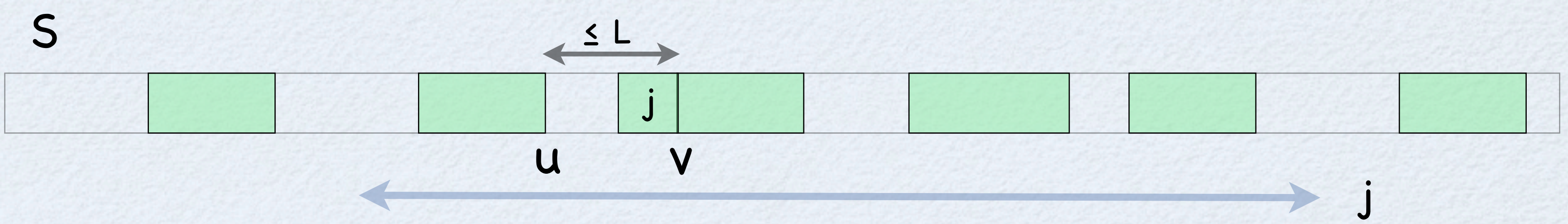


Reduction: Energy ≼ Gaps

S = lex-minimal energy-optimal schedule
[u,v) = short gap in S

Claim: wlog, if $r_j < v$ then j is executed before v

Proof: suppose not



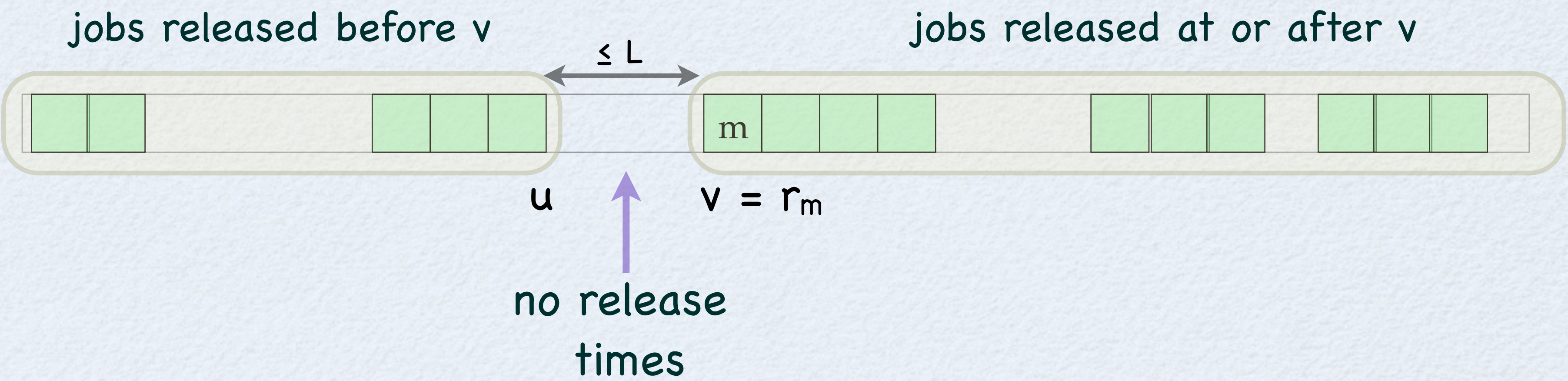
Then E(new S) ≤ E(S) and new S is lex-smaller than S
-- contradiction

Reduction: Energy ≼ Gaps

S = lex-minimal energy optimal schedule

[u,v) = short gap in S

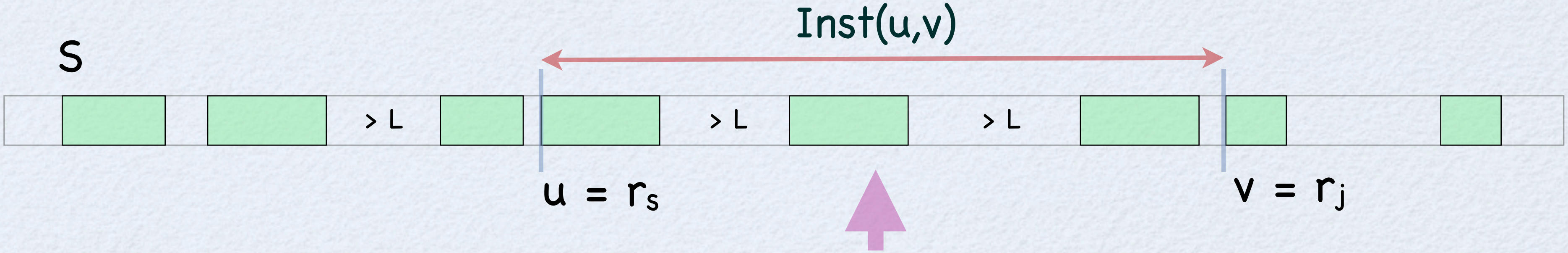
So S looks like this



Reduction: Energy ≼ Gaps

S = lex-minimal energy-optimal schedule

So S looks like this:

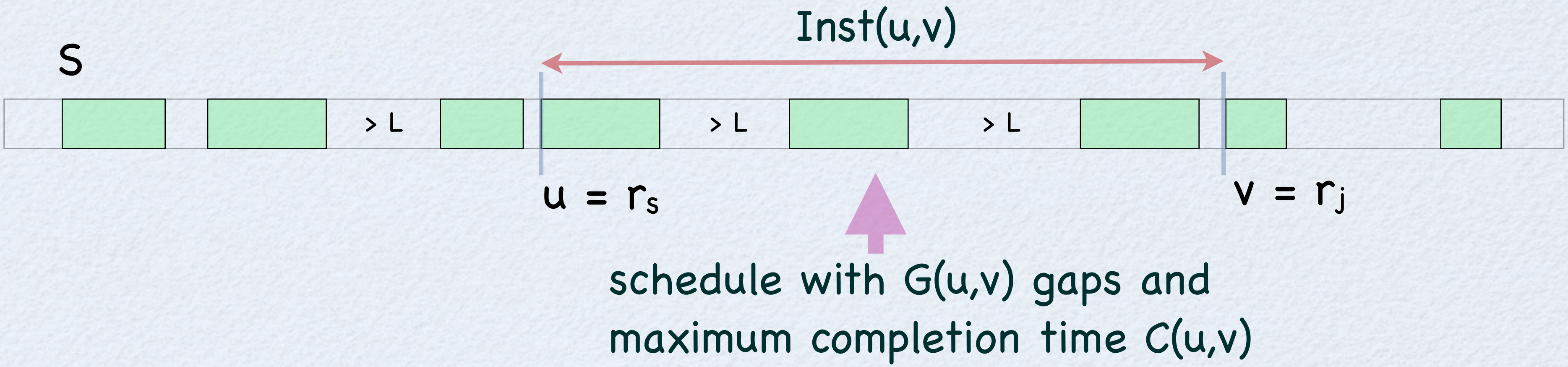


schedule with G(u,v) gaps and maximum completion time C(u,v)

Reduction: Energy ≼ Gaps

S = lex-minimal energy-optimal schedule

So S looks like this:

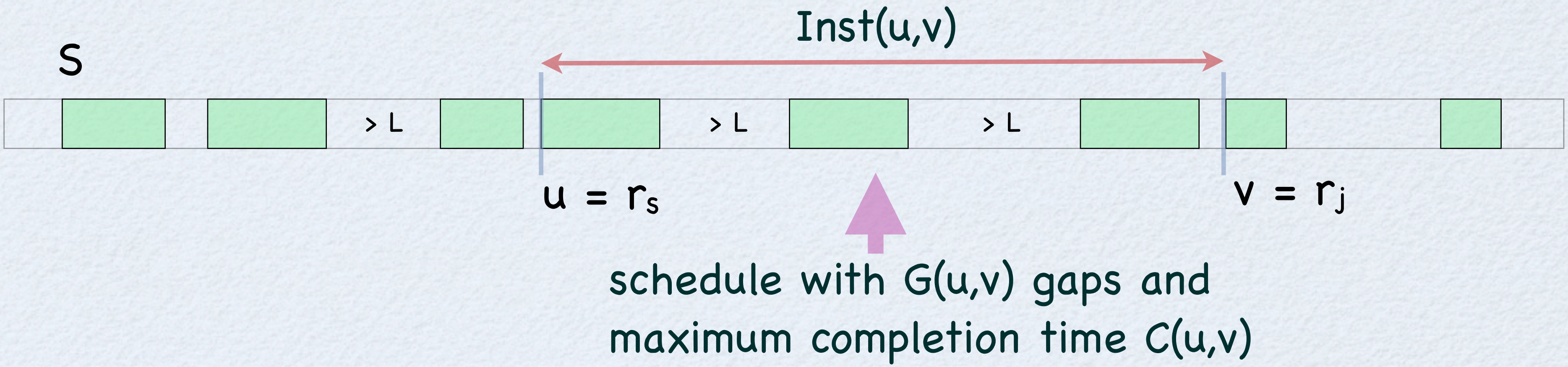


Denote E_s = minimum energy schedule of jobs released $\geq r_s$

Reduction: Energy $\preccurlyeq$ Gaps

S = lex-minimal energy-optimal schedule

So S looks like this:



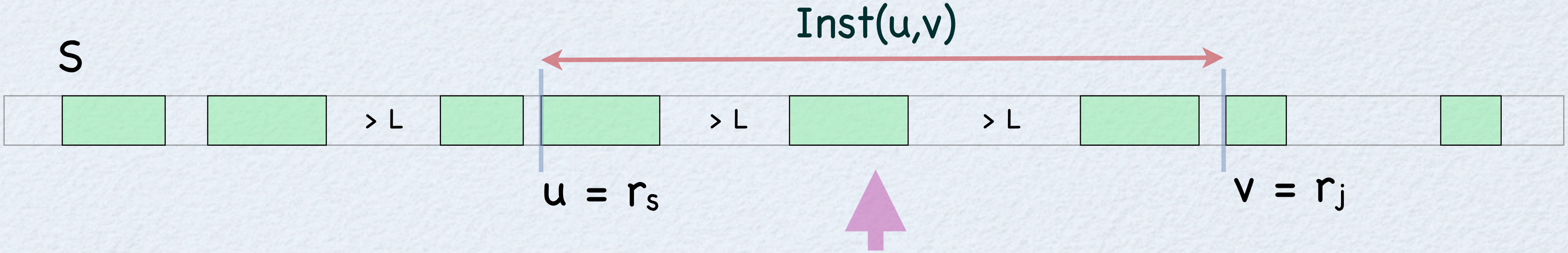
Denote E_s = minimum energy schedule of jobs released $\geq r_s$

$$E_s = \min_{r_j > r_s} \{ L \cdot [\, G(r_s,r_j) - 1 \,] + [\, r_j - C(r_s,r_j) \,] + E_j \}$$

Reduction: Energy ≼ Gaps

S = lex-minimal energy-optimal schedule

So S looks like this:



schedule with G(u,v) gaps and maximum completion time C(u,v)

Denote E_s = minimum energy schedule of jobs released $\geq r_s$

$$E_s = \min_{r_j > r_s} \{ L \cdot [\, G(r_s,r_j)-1 \,] + [\, r_j - C(r_s,r_j) \,] + E_j \}$$

Running time: $O(n^2)$ + (time to compute all G(), C() values)

Minimum Energy Scheduling

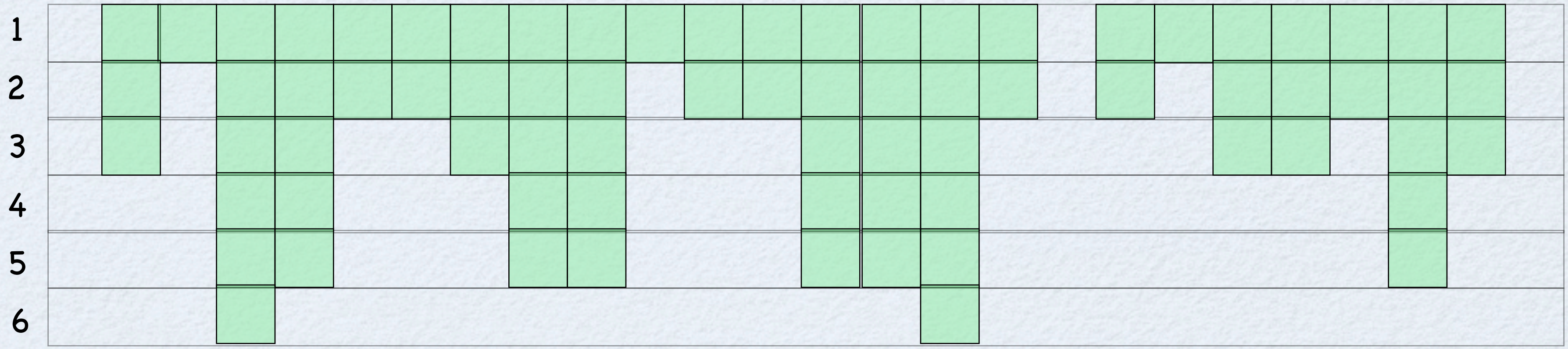
Main techniques:

* Philippe's partitioning trick ✓
* Reducing the minimizer sets ✓
* Inversion trick ("large" parameter ⇔ "small" value) ✓
* $O(n^2)$-time reduction: Energy $\preccurlyeq$ Gaps ✓

Minimum Energy Scheduling - Other Results

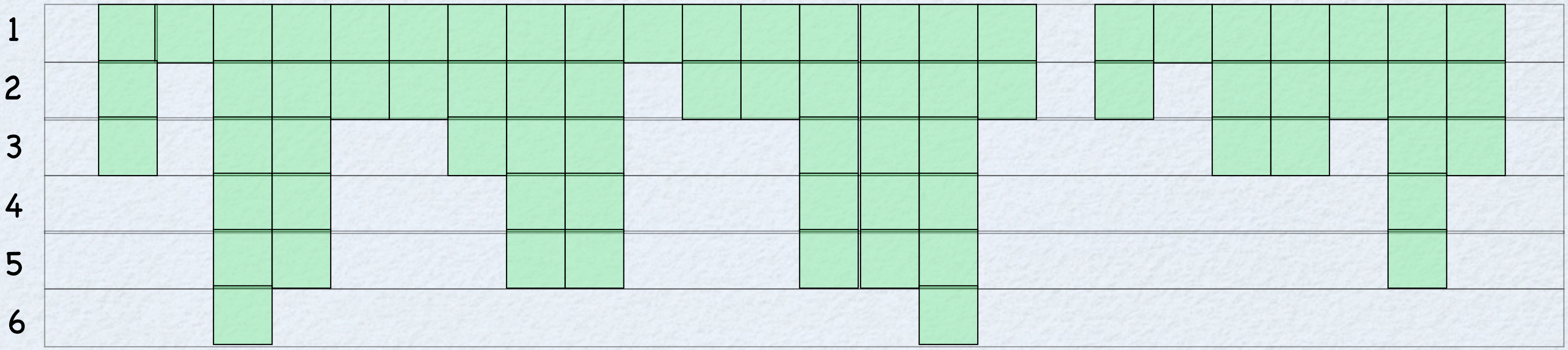
m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:

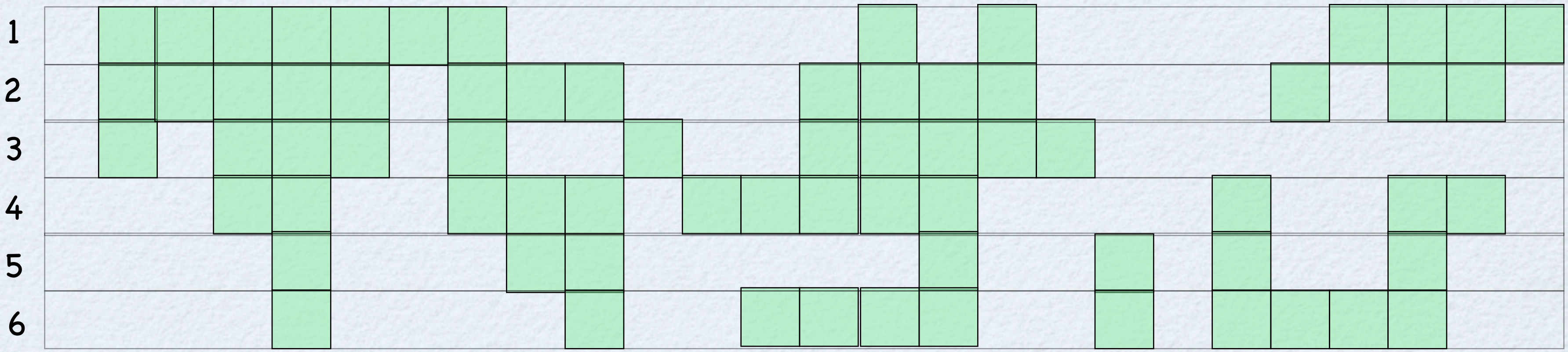


m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:

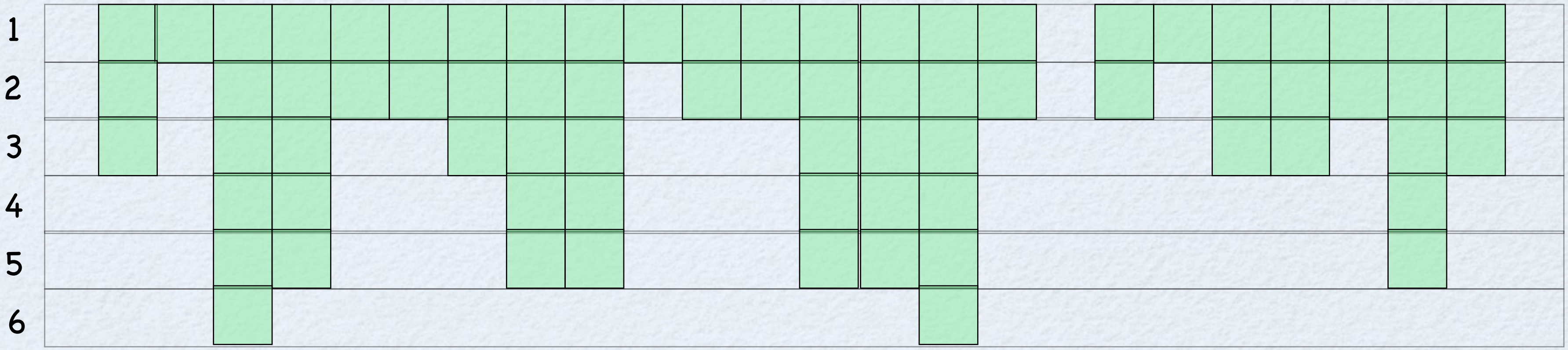


Proof: Suppose not:

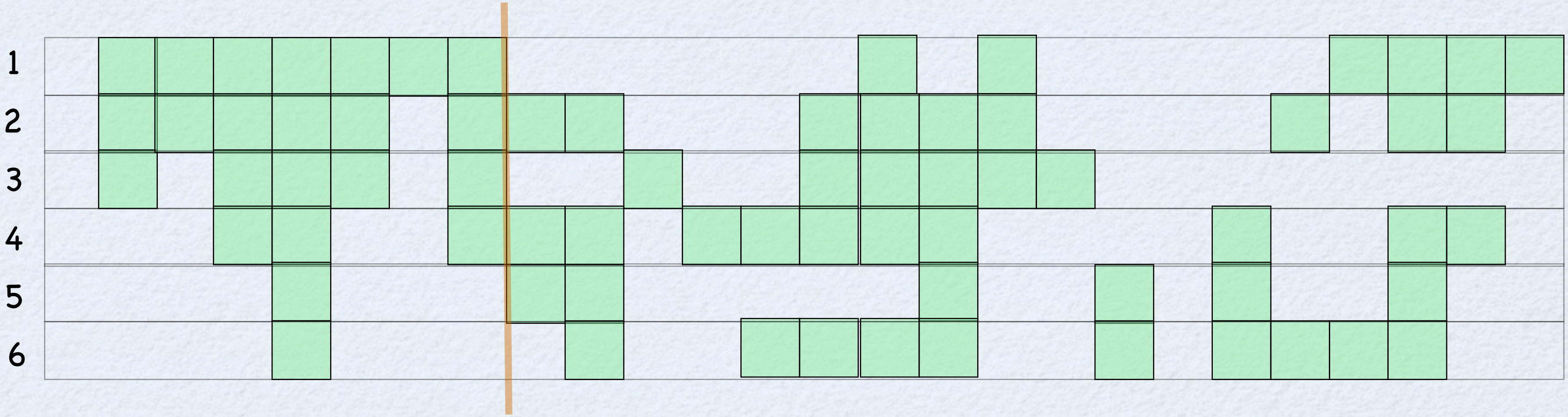


m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:

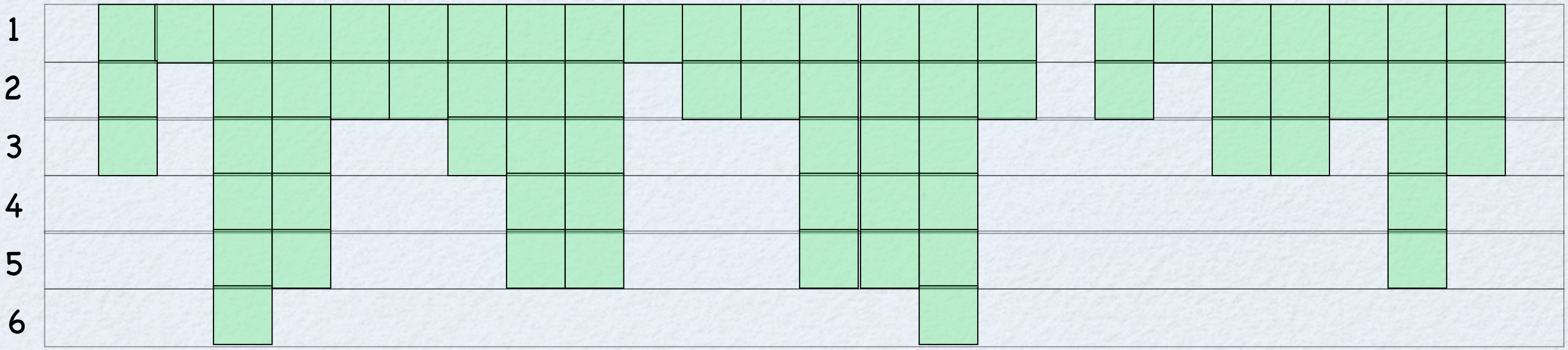


Proof: Suppose not:

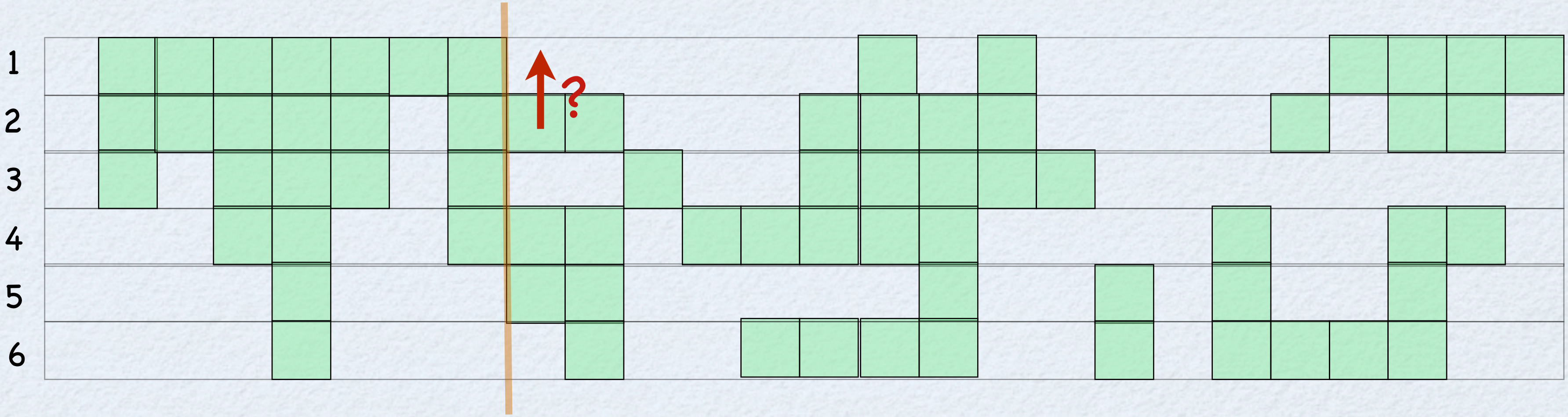


m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:

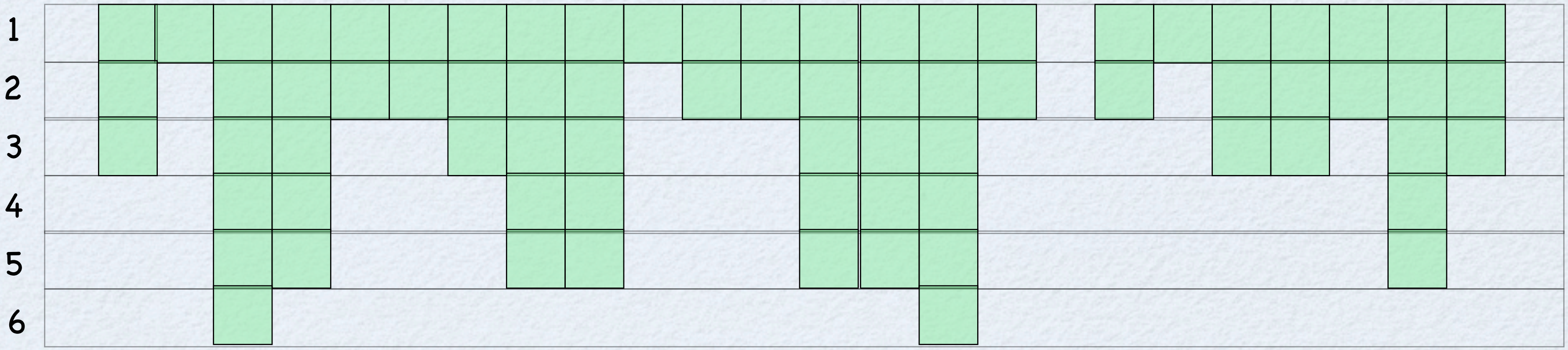


Proof: Suppose not:

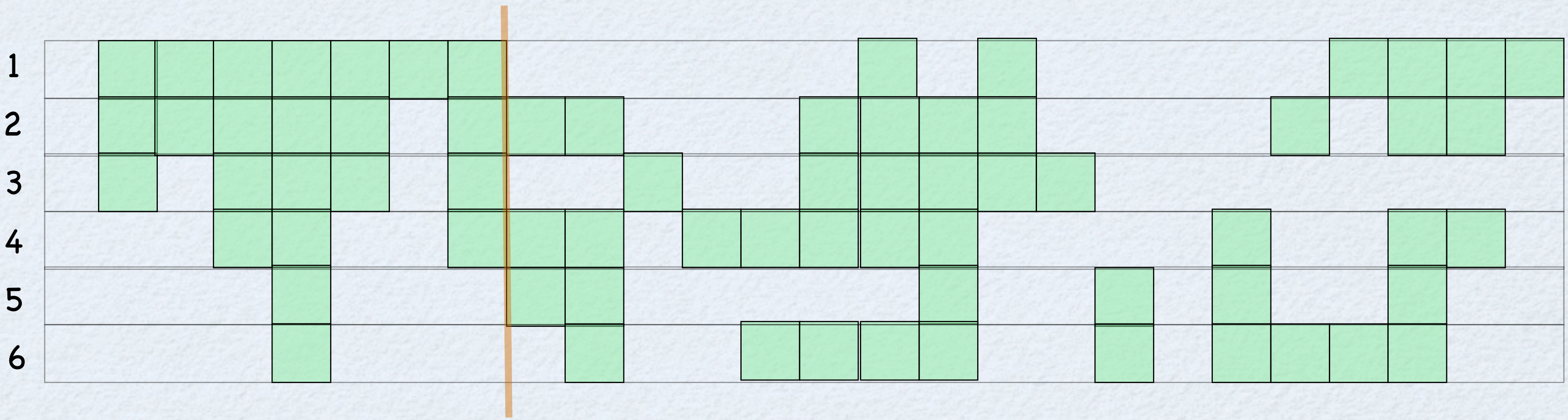


m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:

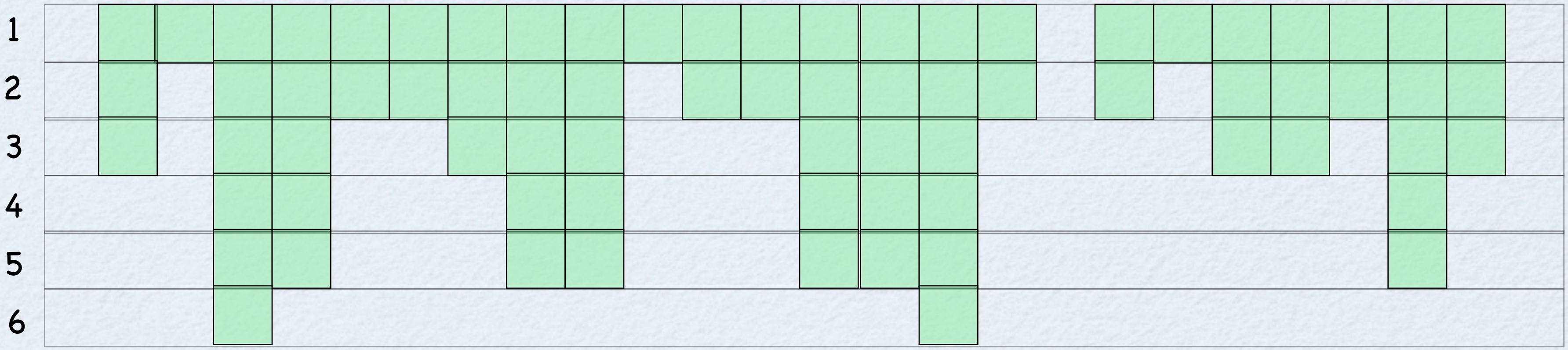


Proof: Suppose not:

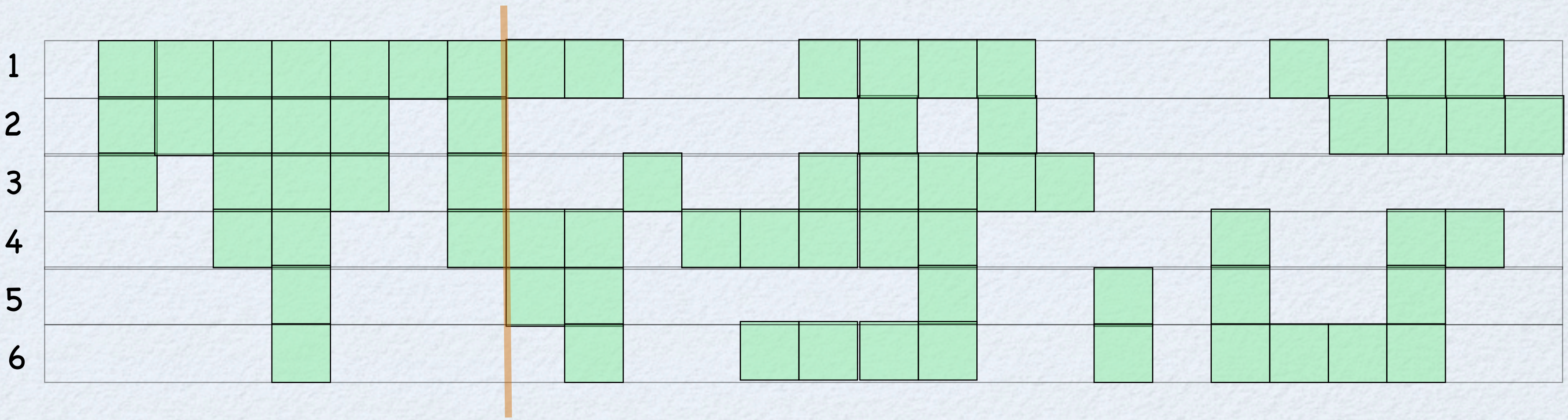


m processors, unit jobs, gaps [DG...'07]

Claim: WLOG, optimal schedule is compact:



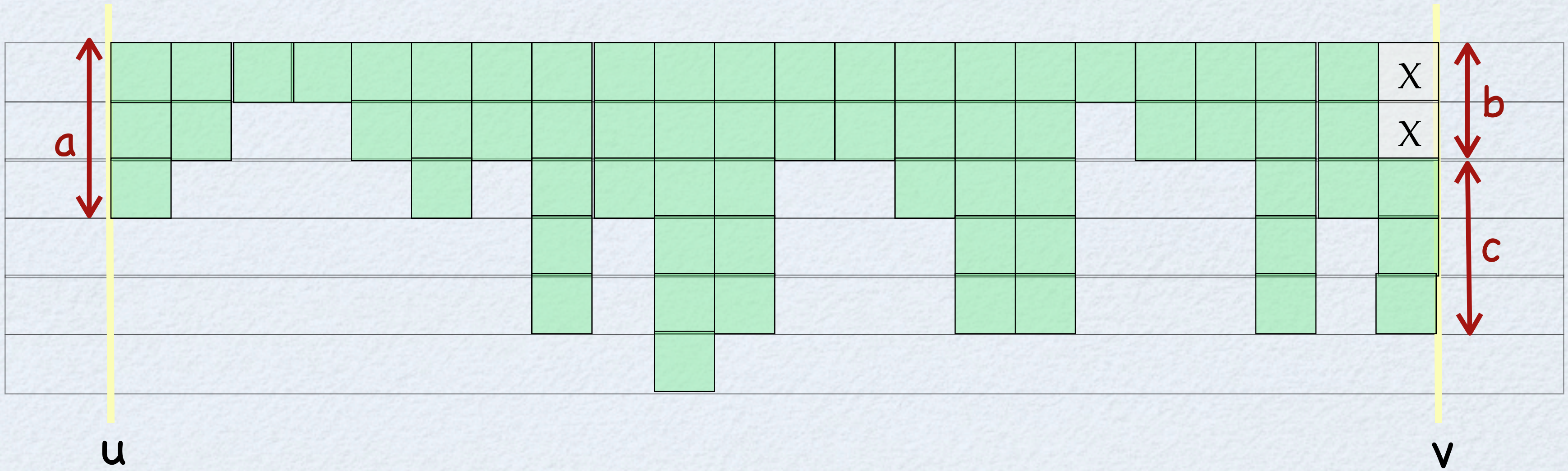
Proof: Suppose not:



switch cannot increase # gaps, so repeat till schedule is compact

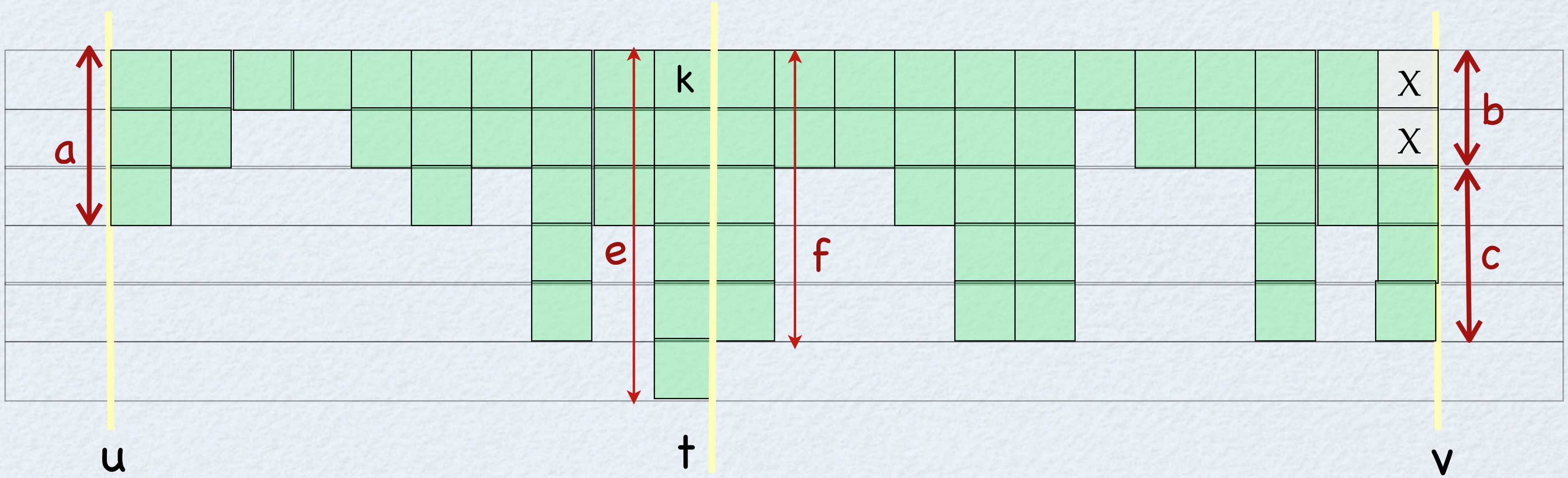
m processors, unit jobs, gaps [DG...'07]

Generalize Philippe's partition trick: Sub-instance



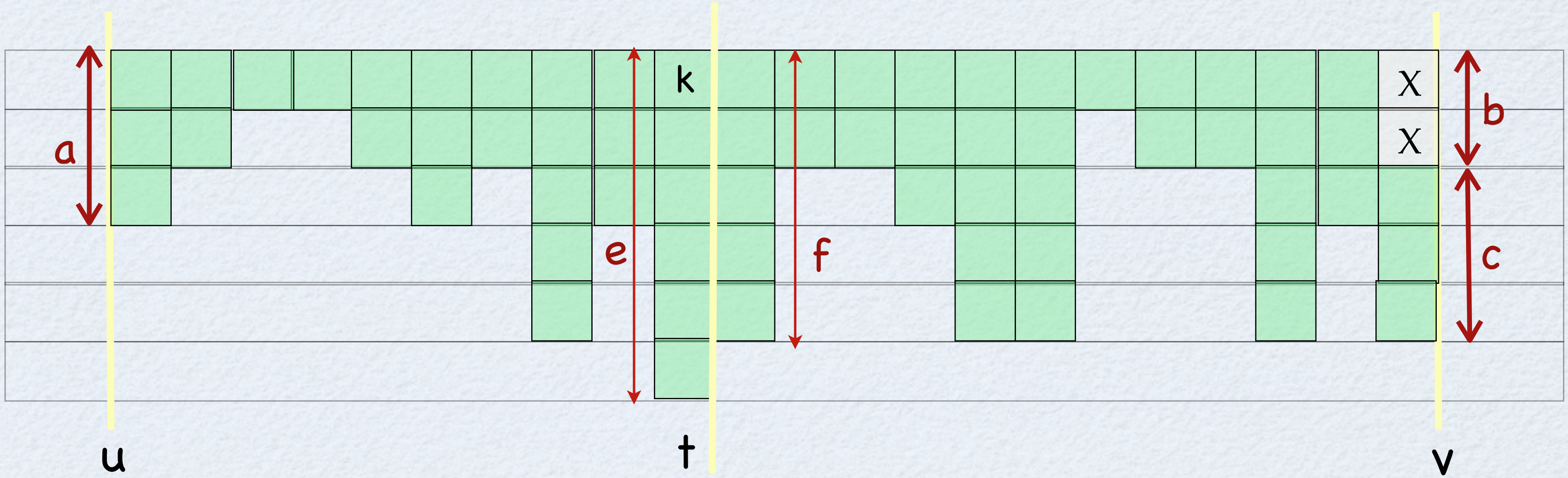
m processors, unit jobs, gaps [DG...'07]

Generalize Philippe's partition trick: Sub-instance



m processors, unit jobs, gaps [DG...'07]

Generalize Philippe's partition trick: Sub-instance



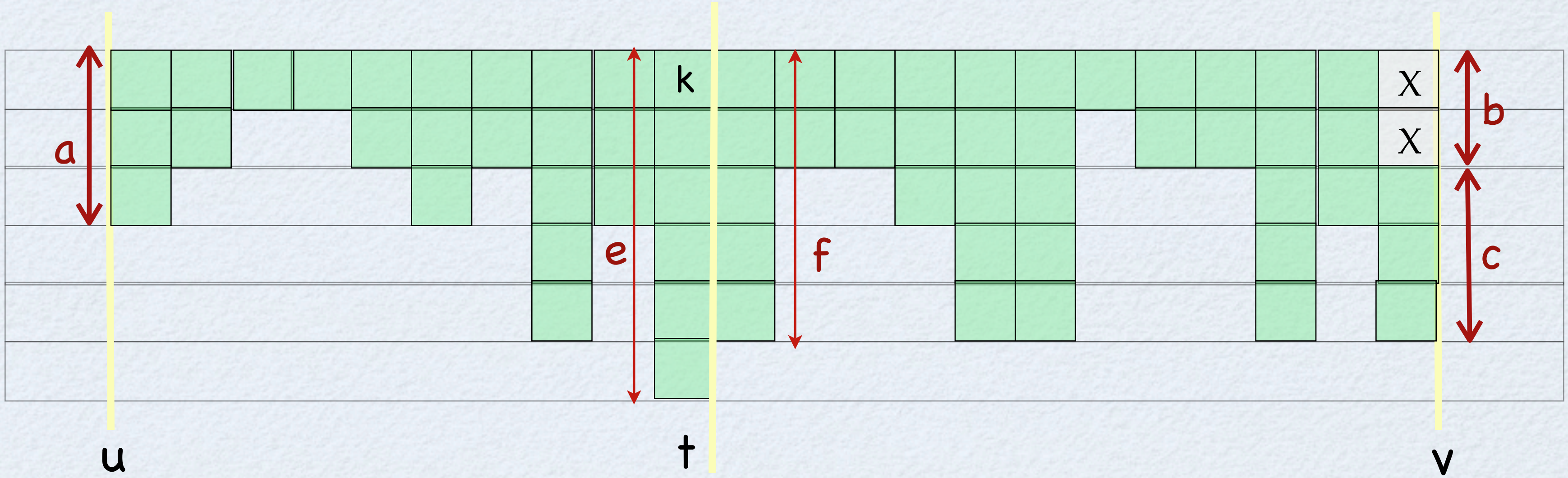
Recurrence:

$$G_k(u,a,b,v,c) = \min_t \min_{e,f} \{ G(\ldots) + G(\ldots .) \}$$

Running time $O(n^7 m^5)$ [DG...'07]

m processors, unit jobs, gaps [DG...'07]

Generalize Philippe's partition trick: Sub-instance



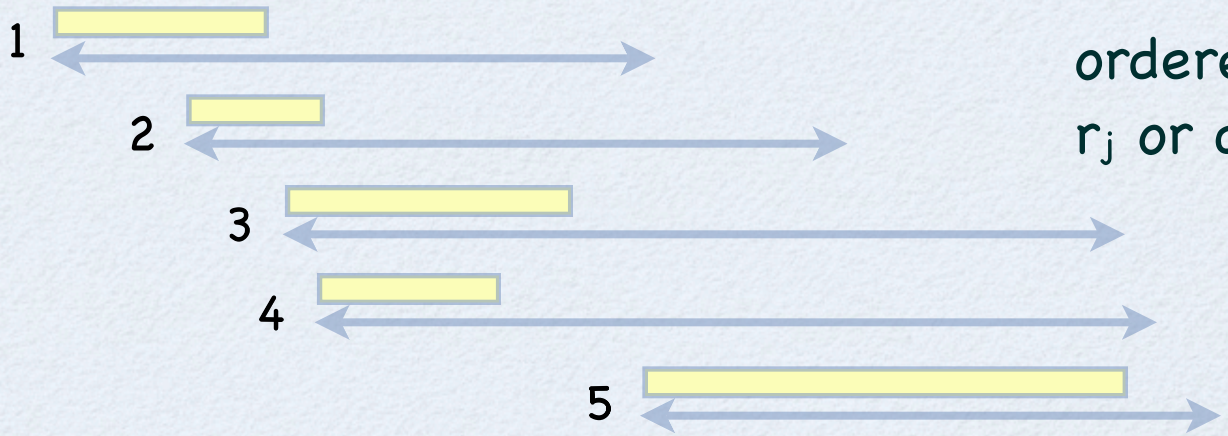
Recurrence:

$$G_k(u,a,b,v,c) = \min_t \min_{e,f} \{ G(\ldots) + G(\ldots) \}$$

Running time $O(n^7m^5)$ [DG...'07]

Can be improved to $O(n^5m^5)$ using smaller maximizer sets

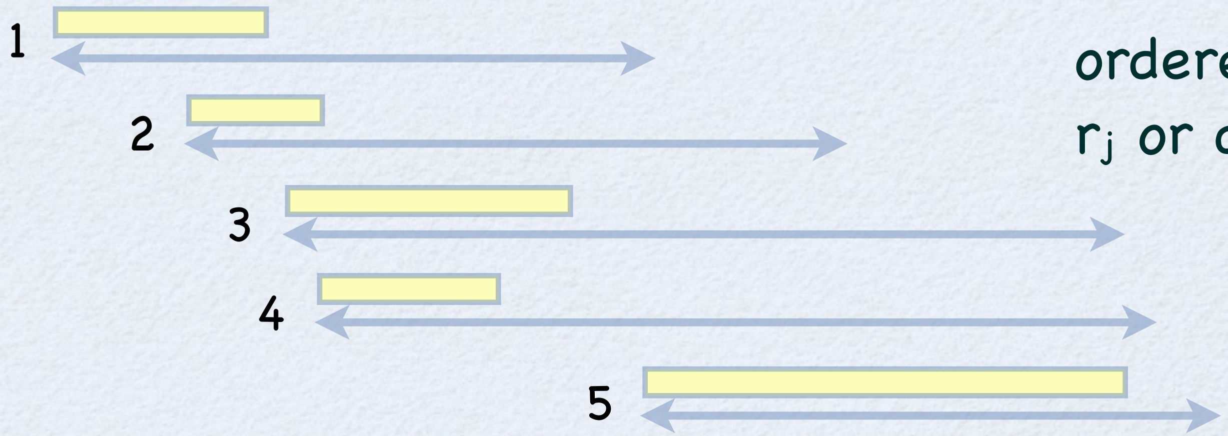
1 processor, agreeable [GJS'10]



ordered by

r_j or d_j

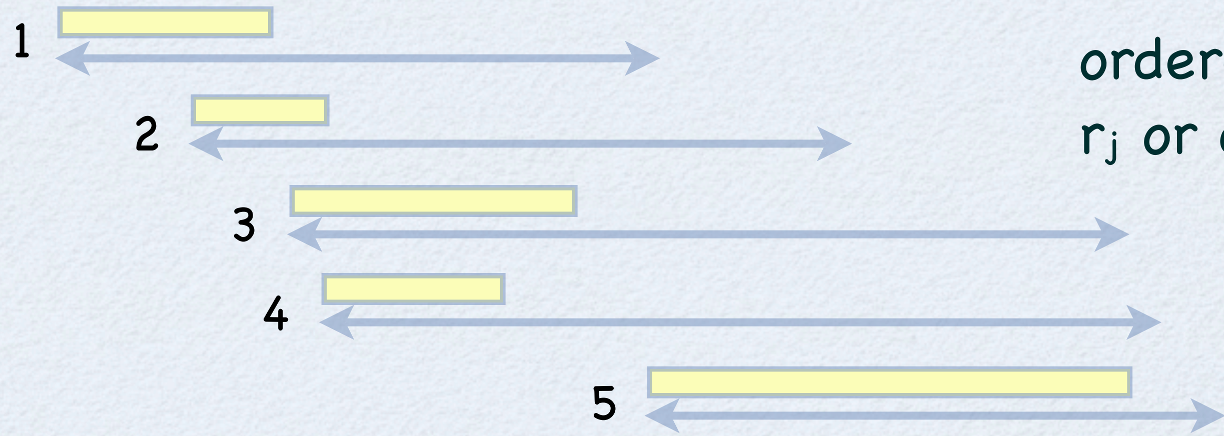
1 processor, agreeable [GJS'10]



ordered by r_j or d_j

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

1 processor, agreeable [GJS'10]

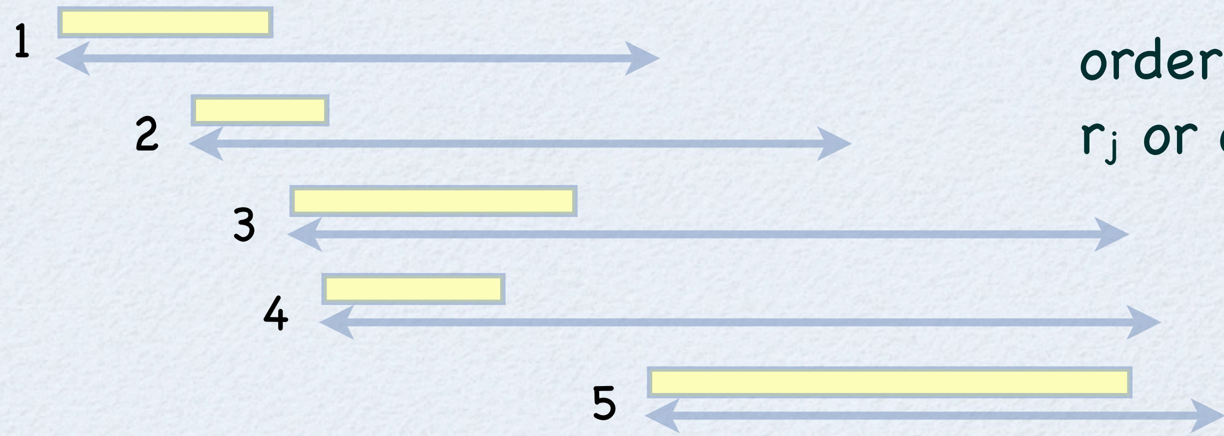


ordered by
r_j or d_j

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, $d_j + p_{j+1} \le d_{j+1}$

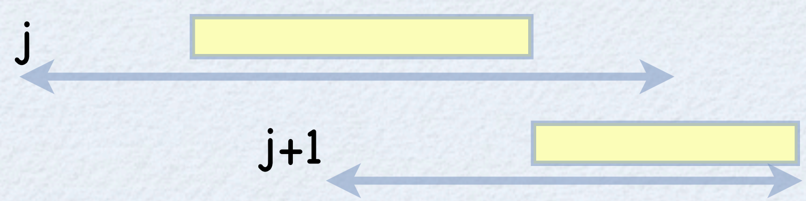
1 processor, agreeable [GJS'10]



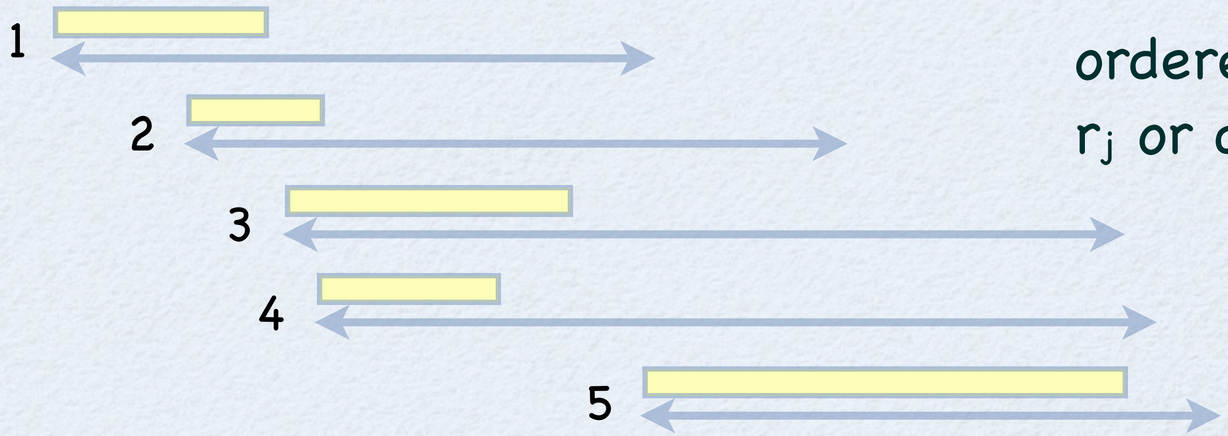
ordered by r_j or d_j

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, $d_j + p_{j+1} \leq d_{j+1}$



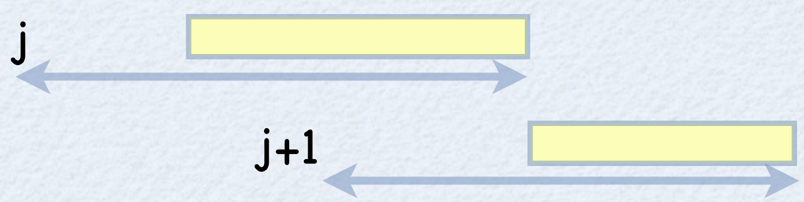
1 processor, agreeable [GJS'10]



ordered by
r_j or d_j

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, $d_j + p_{j+1} \le d_{j+1}$



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

preprocess jobs as in Claim 2
Schedule 1 at d_1-p_1
for any other j
    if possible, schedule j right after j-1
    else schedule j at d_j-p_j

1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

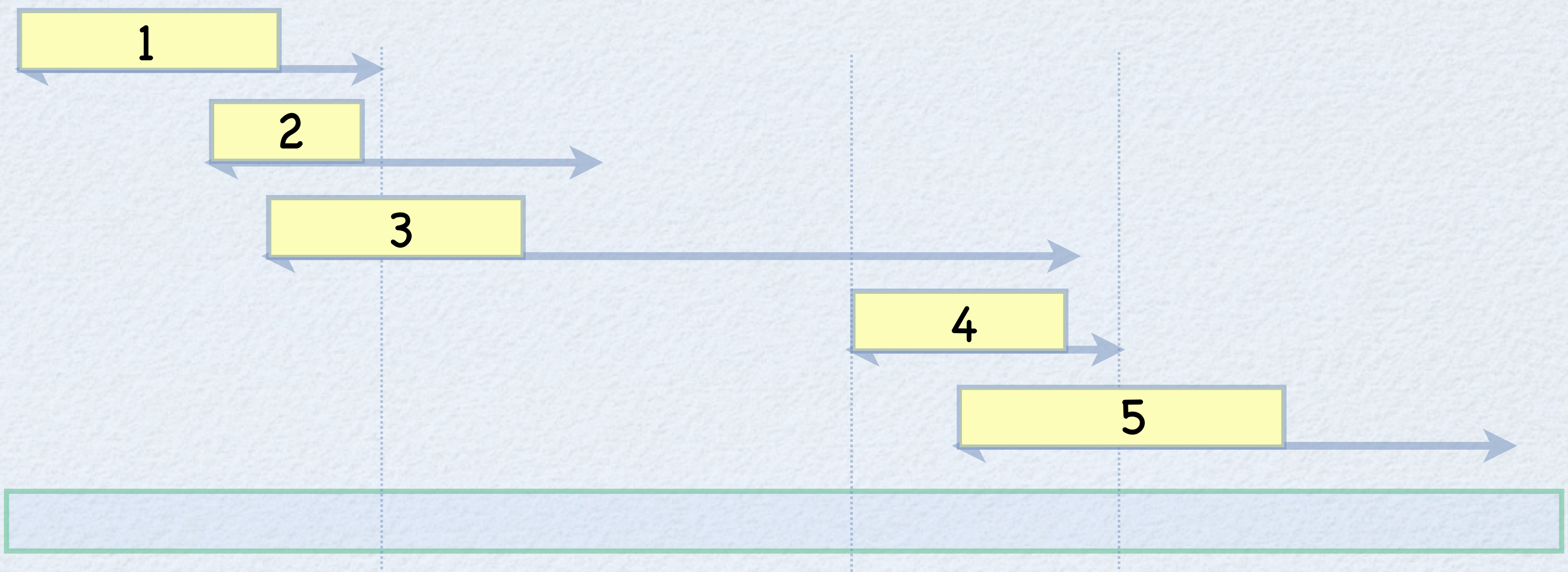
preprocess jobs as in Claim 2

Schedule 1 at d_1–p_1

for any other j

if possible, schedule j right after j-1

else schedule j at d_j–p_j



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

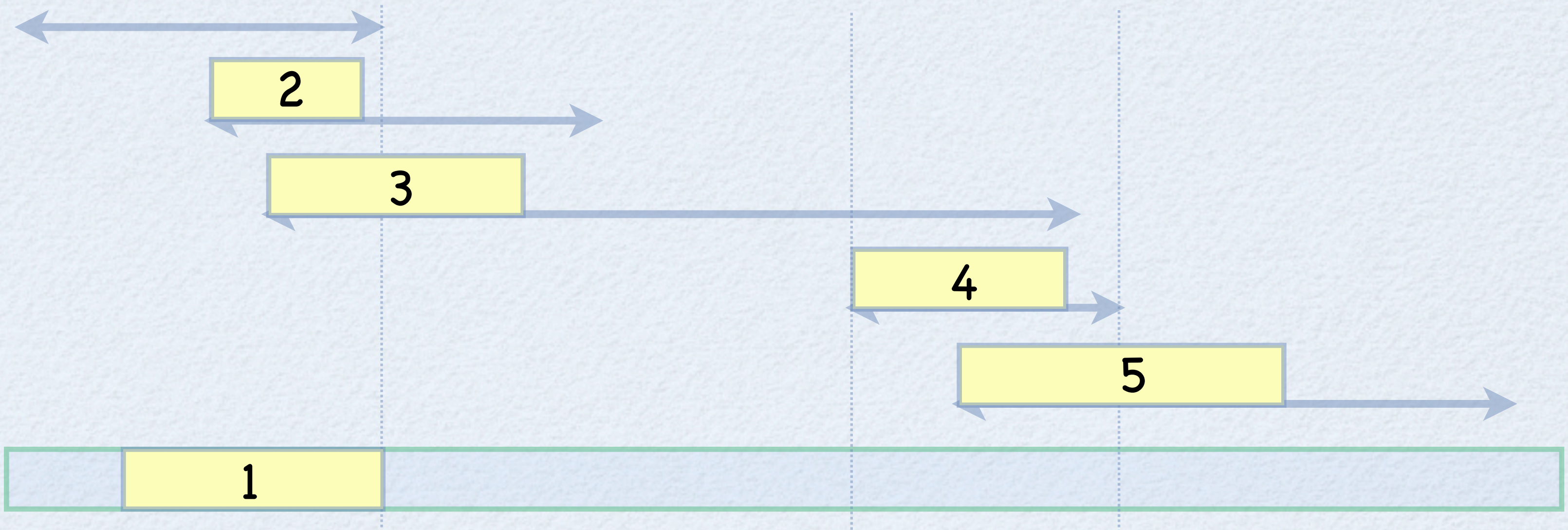
preprocess jobs as in Claim 2

Schedule 1 at $d_1 - p_1$

for any other j

  if possible, schedule j right after j-1

  else schedule j at $d_j - p_j$



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

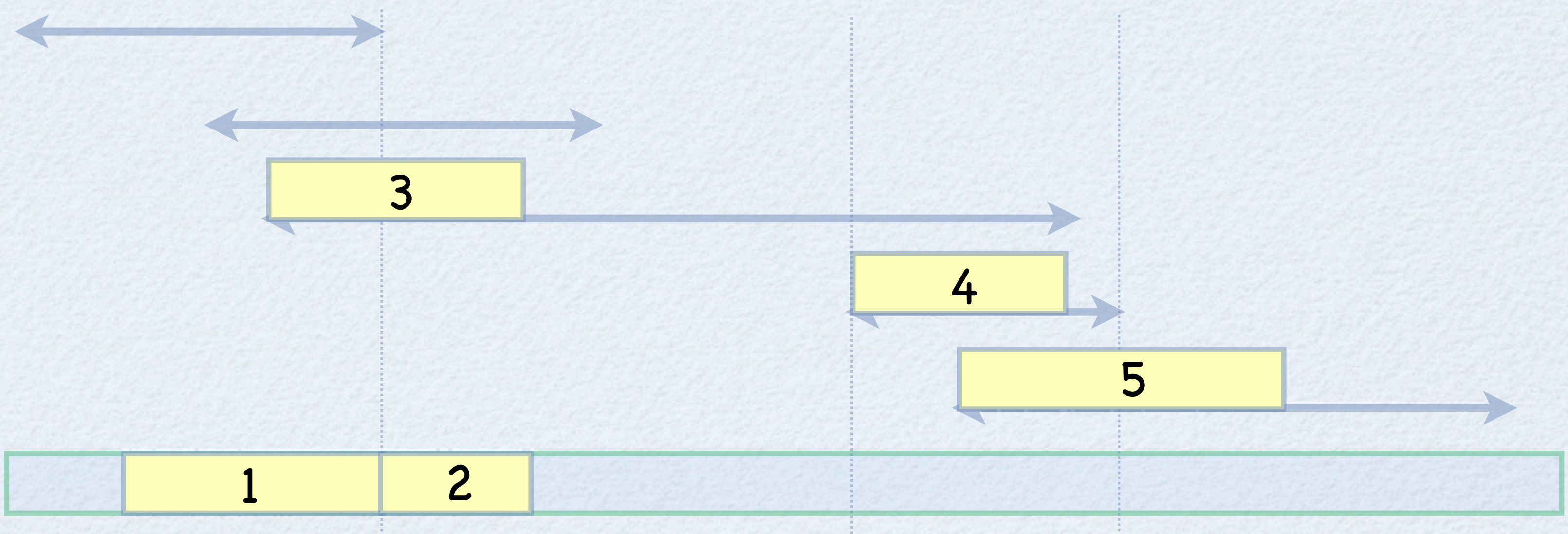
preprocess jobs as in Claim 2

Schedule 1 at $d_1 - p_1$

for any other j

if possible, schedule j right after j-1

else schedule j at $d_j - p_j$



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

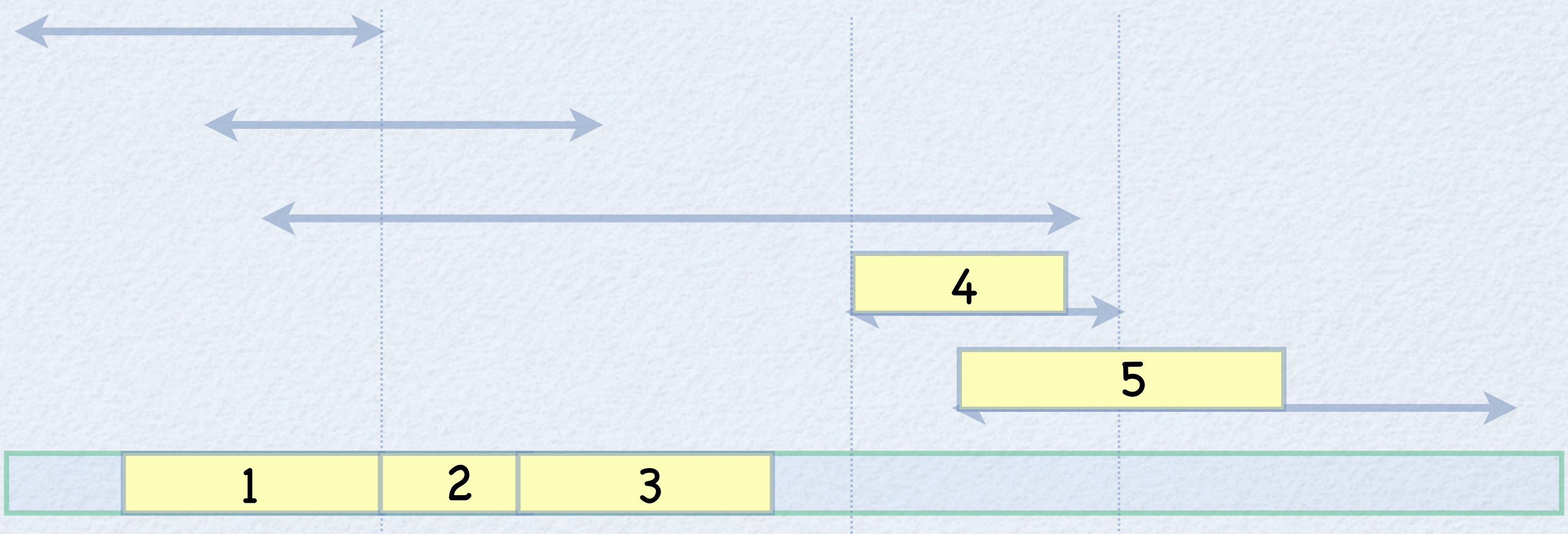
preprocess jobs as in Claim 2

Schedule 1 at d_1-p_1

for any other j

if possible, schedule j right after j-1

else schedule j at d_j-p_j



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

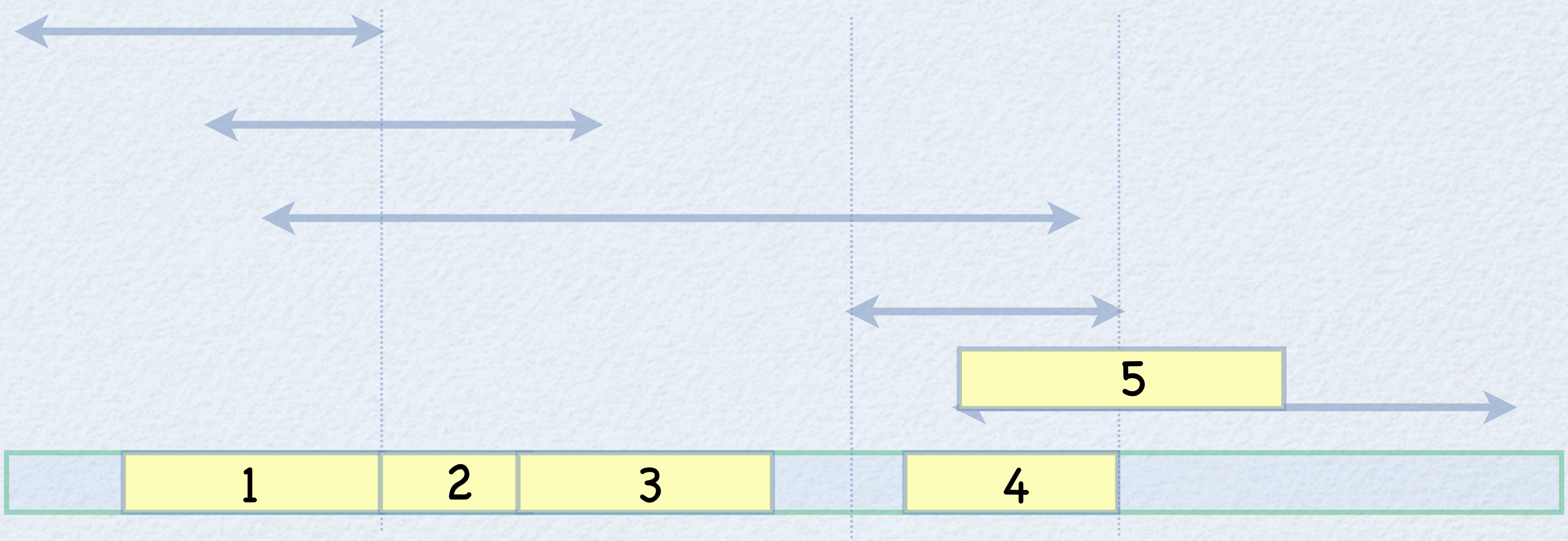
preprocess jobs as in Claim 2

Schedule 1 at $d_1 - p_1$

for any other j

if possible, schedule j right after j-1

else schedule j at $d_j - p_j$



1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

preprocess jobs as in Claim 2

Schedule 1 at d_1-p_1

for any other j

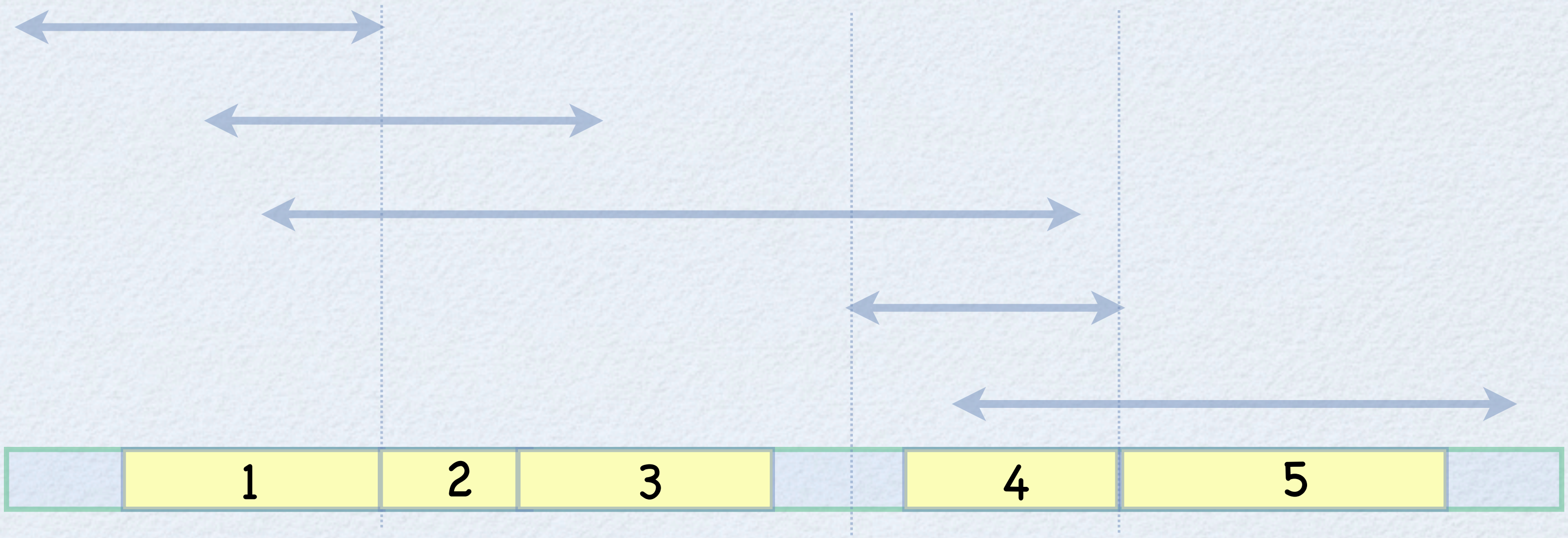
  if possible, schedule j right after j-1

  else schedule j at d_j-p_j

1 processor, agreeable [GJS'10]

Algorithm GJS-L1SO:

preprocess jobs as in Claim 2
Schedule 1 at d_1-p_1
for any other j
 if possible, schedule j right after j-1
 else schedule j at d_j-p_j



Running time: sorting + O(n) = O(n logn)

Open (Easy?) Questions

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system

Open (Easy?) Questions

1. Back to sheep: if any group of ≤ g sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time $O(n^3)$?

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time $O(n^3)$?
6. Fast approximations: $1+\varepsilon$-approx. in $\tilde{O}(n)$ time?

Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time $O(n^3)$?
6. Fast approximations: $1+\varepsilon$-approx. in $\tilde{O}(n)$ time?
7. ...