ILP-based Modulo Scheduling for High-level Synthesis

Julian Oppermann, Andreas Koch,
Melanie Reuter-Oppermann, Oliver Sinnen

TECHNISCHE UNIVERSITÄT DARMSTADT

Karlsruhe Institute of Technology

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Outline

- Introduction to loop pipelining / modulo scheduling

Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Modulo SDC
Canis et al.

state-of-the-art **heuristic**

Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Modulo SDC
Canis et al.

state-of-the-art **heuristic**

Formulation by
Eichenberger &
Davidson

state-of-the-art **exact** formulation

Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Modulo SDC
Canis et al.

Formulation by
Eichenberger &
Davidson

Moovac
Oppermann et al.

state-of-the-art **heuristic** state-of-the-art **exact** formulation novel **exact** formulation

Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches
  - result quality, heuristic vs. exact

Modulo SDC
Canis et al.

state-of-the-art **heuristic**

Formulation by **Eichenberger** & Davidson

state-of-the-art **exact** formulation

Moovac
Oppermann et al.

novel **exact** formulation

Outline

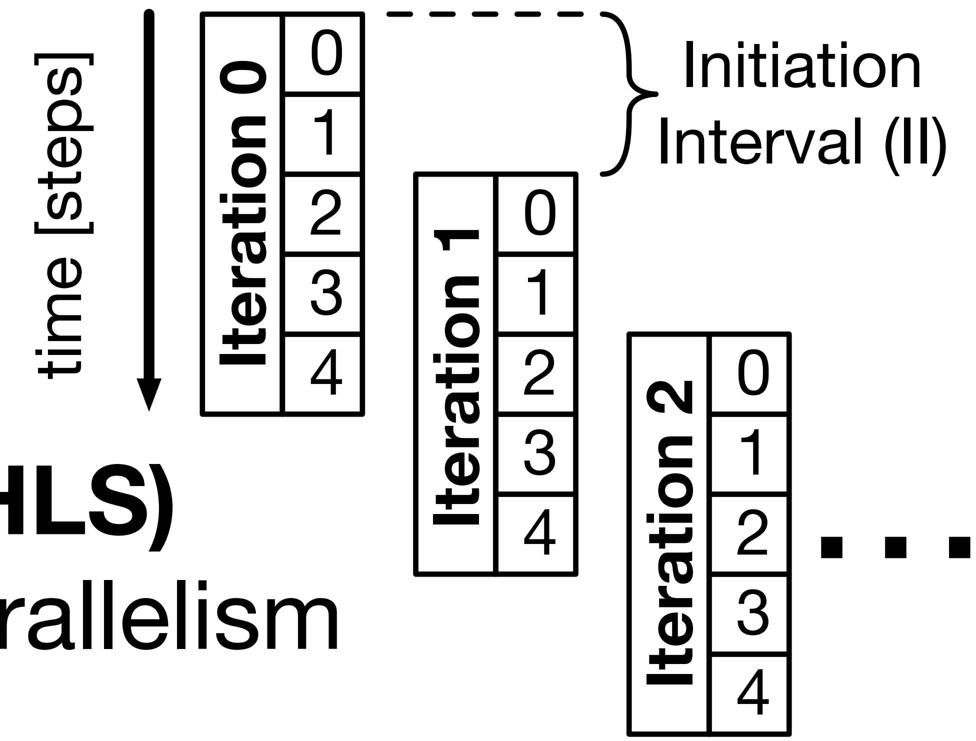
- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches
  - result quality, heuristic vs. exact
  - **time to schedule** - *it's impractical to do exact modulo scheduling, right?*

Modulo SDC Canis et al.	Formulation by **Eichenberger** & Davidson	**Moovac** Oppermann et al.
state-of-the-art **heuristic**	state-of-the-art **exact** formulation	novel **exact** formulation

Loop Pipelining

- C-based **High-level Synthesis (HLS)** needs to exploit all sources of parallelism

Loop Pipelining



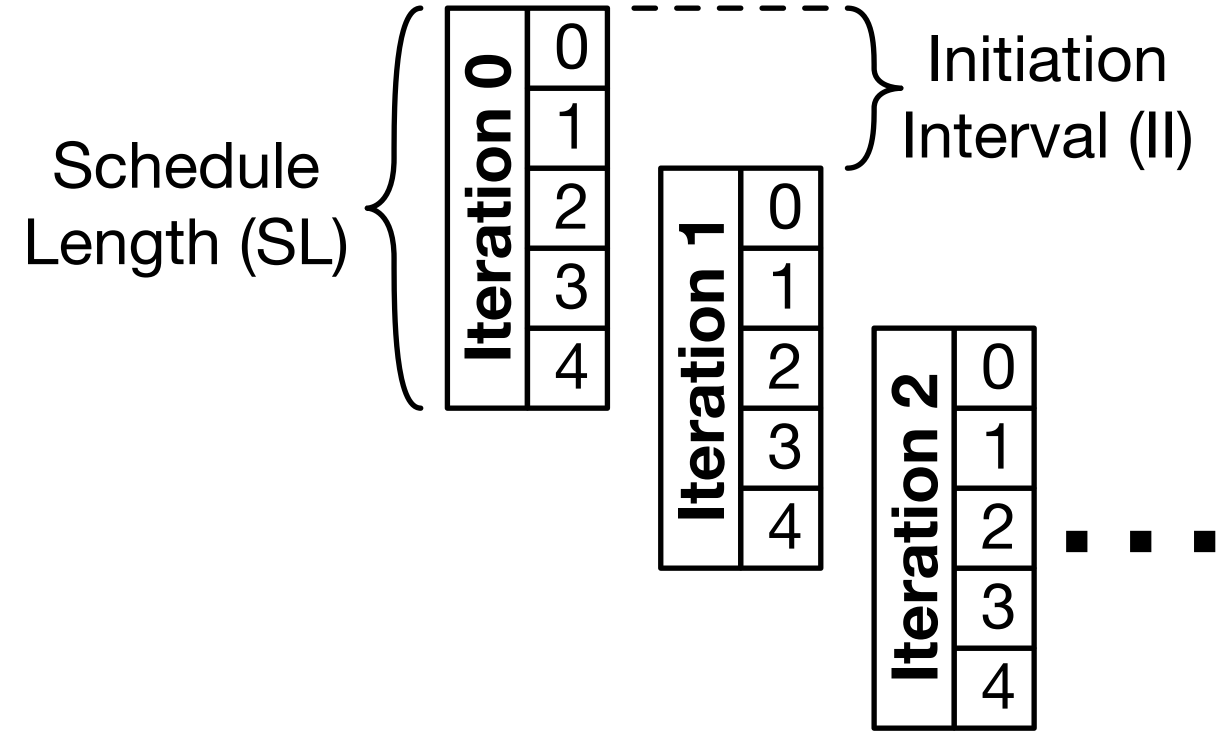
- C-based **High-level Synthesis (HLS)** needs to exploit all sources of parallelism

- **Loop pipelining**
  = new loop iterations are started after a fixed number of time steps, called **Initiation Interval (II)**

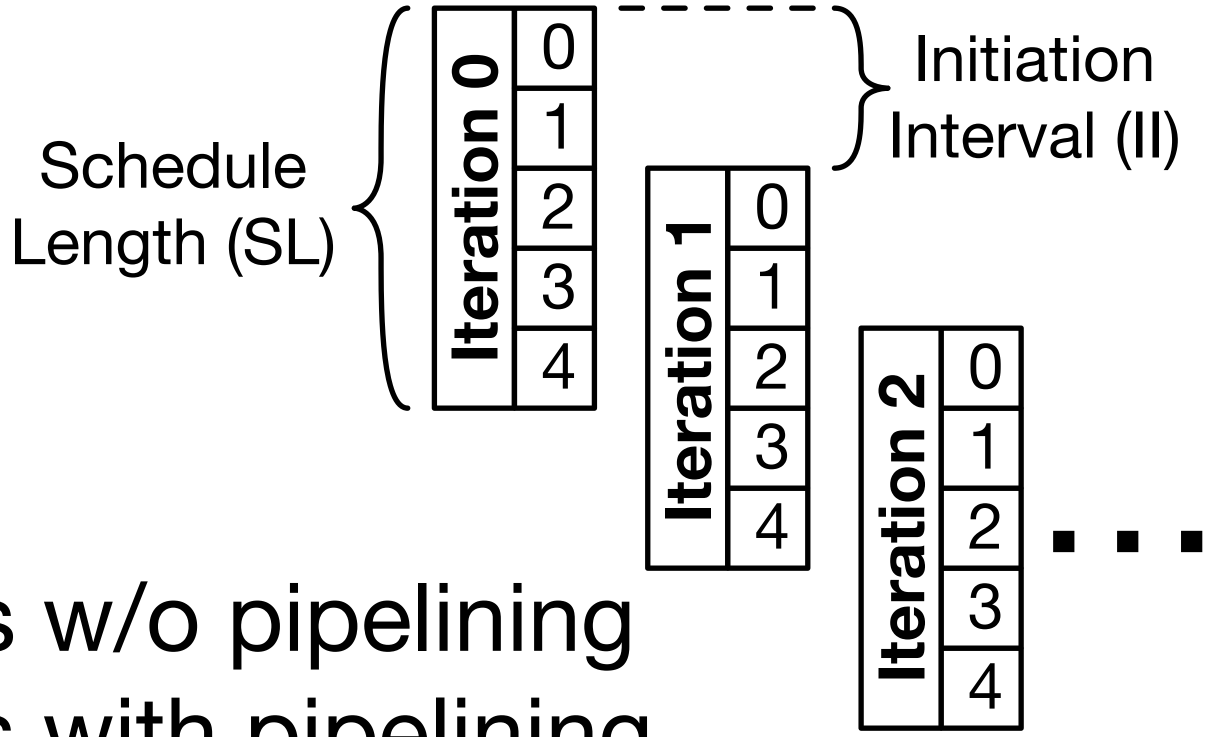
  - Partially overlapping execution of subsequent loop iterations

Loop Pipelining

- Increases throughput!

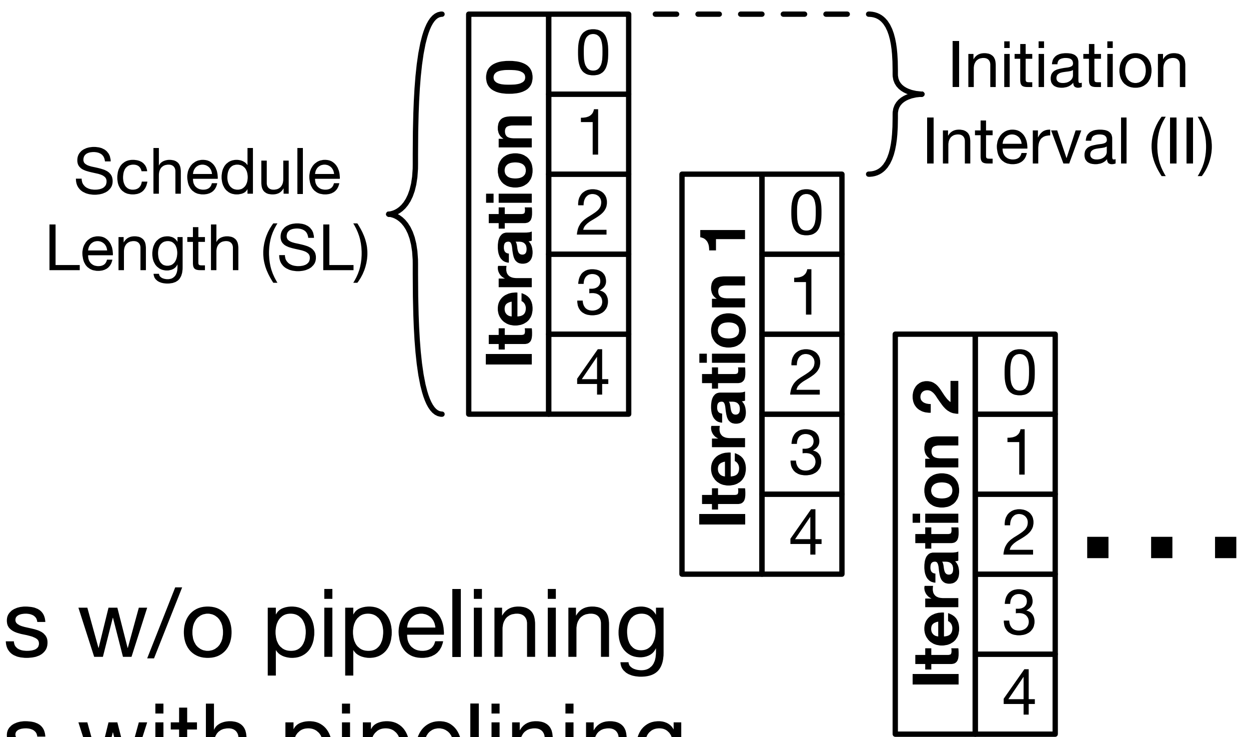


Loop Pipelining



- Increases throughput!
- Executing *n* iterations →
  $n \cdot SL$ time steps w/o pipelining
  $(n-1) \cdot II + SL$ time steps with pipelining

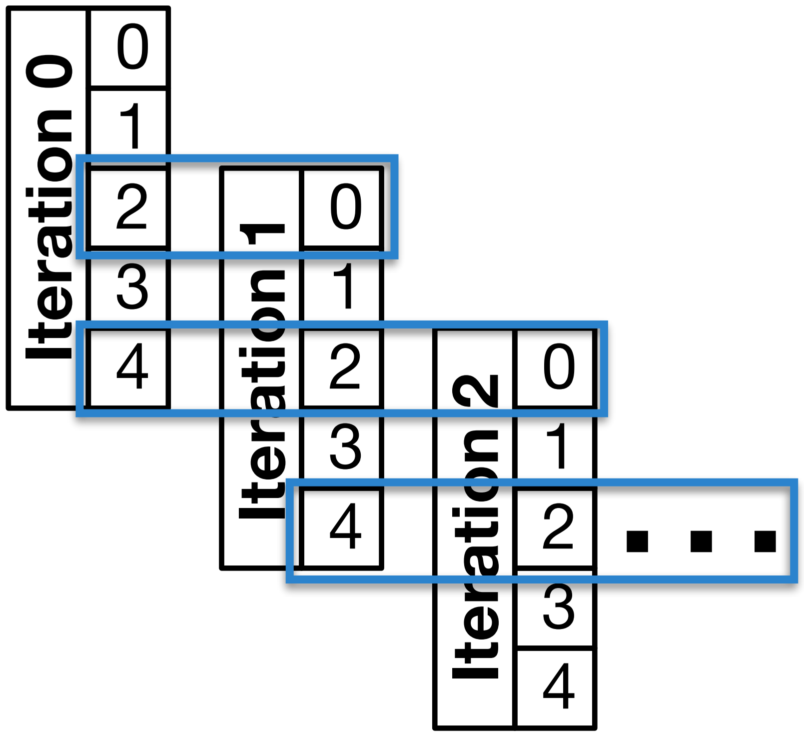
Loop Pipelining



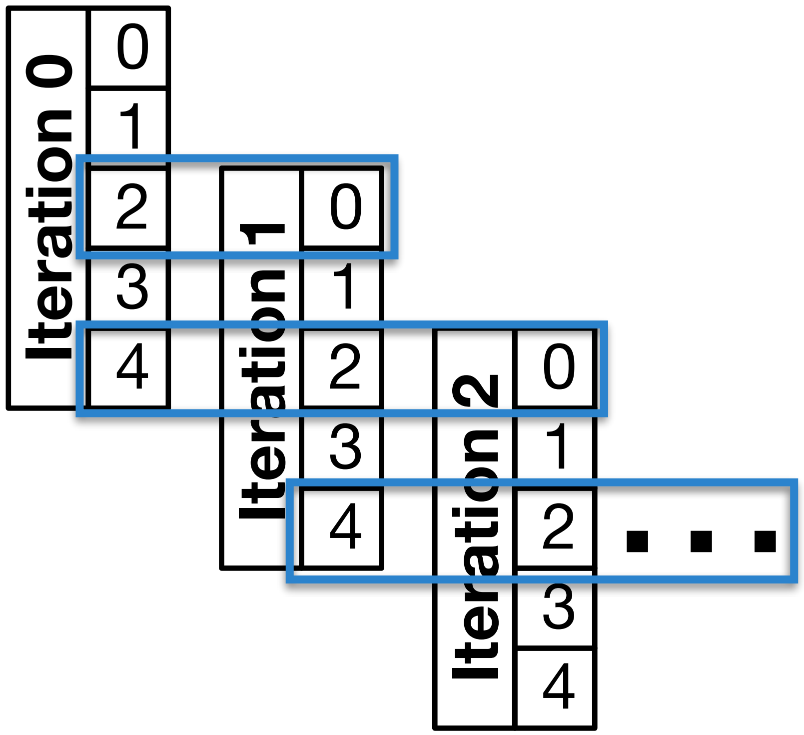
- Increases throughput!
- Executing n iterations →
  $n \cdot SL$ time steps w/o pipelining
  $(n-1) \cdot II + SL$ time steps with pipelining
- Primary objective is to find **smallest feasible II**
  - Limited by dependencies between iterations
  - Subject to resource constraints (cache ports, DSPs, …)

Loop Pipelining

- Operations from different iterations are active at the same time



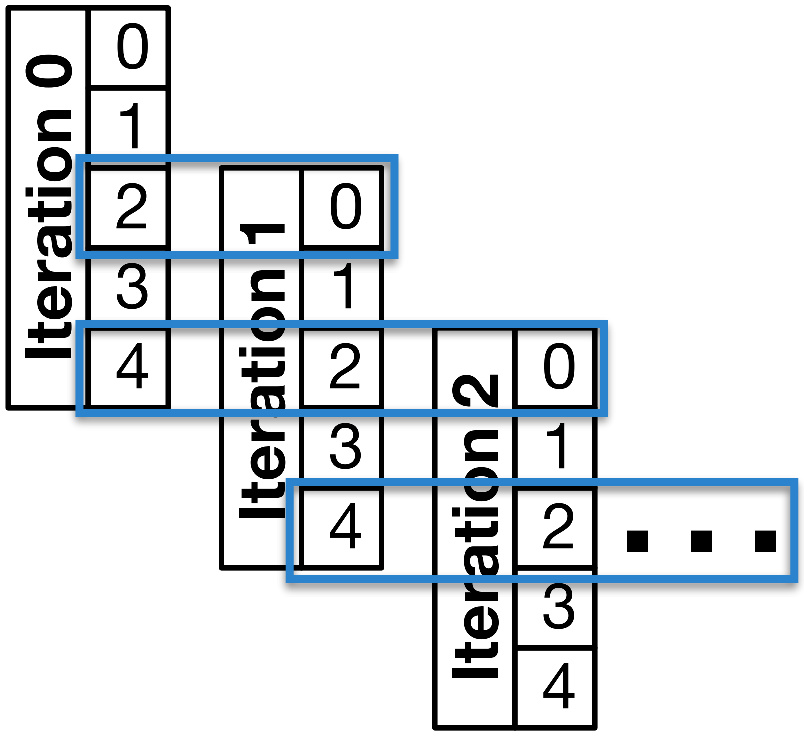
Loop Pipelining



- Operations from different iterations are active at the same time

- Resource constraints have to hold for **congruence classes (modulo II)** of time steps
  - *“modulo resource constraints”*

Loop Pipelining



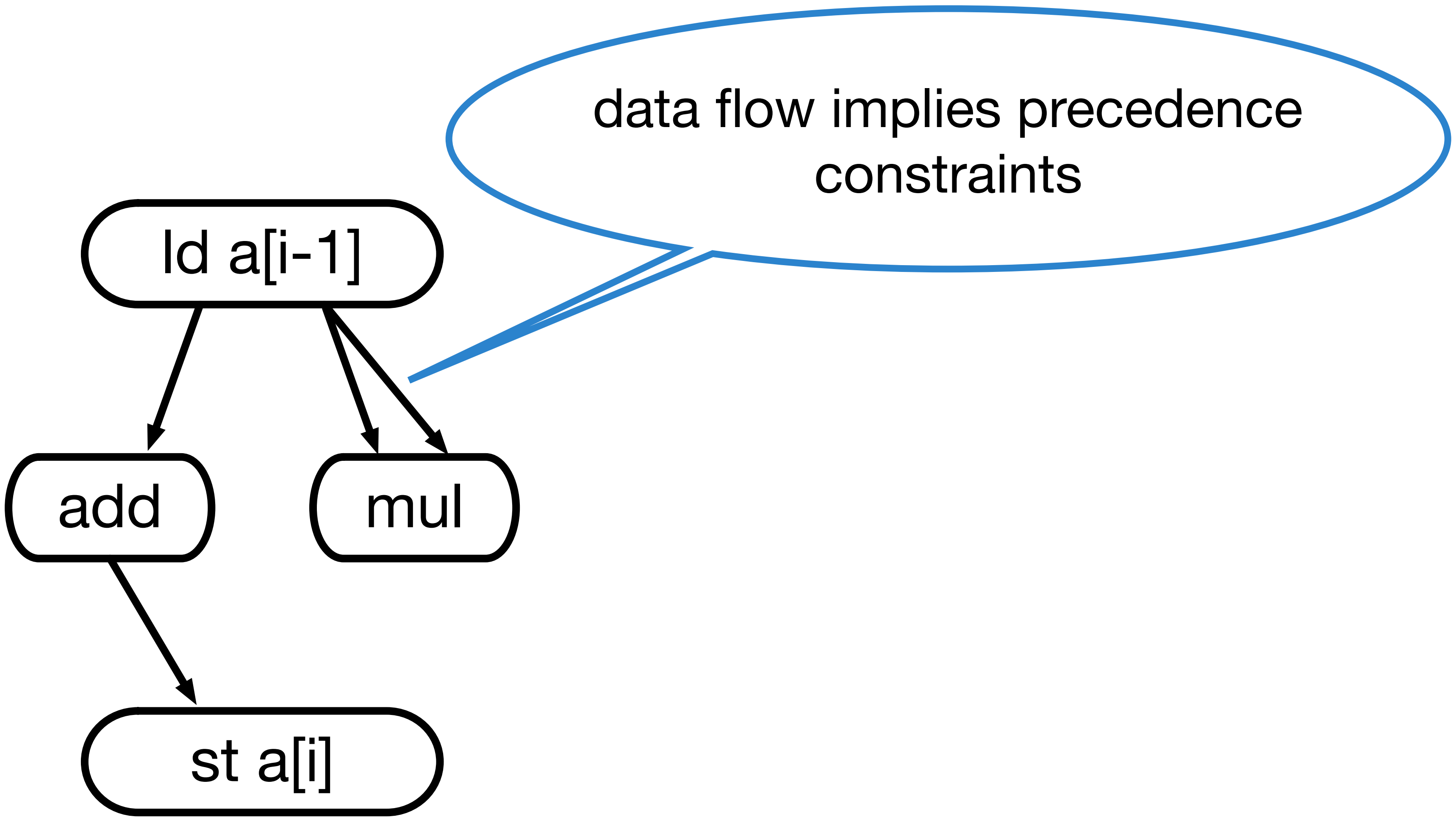
- Operations from different iterations are active at the same time
- Resource constraints have to hold for **congruence classes (modulo II)** of time steps
  - *"modulo resource constraints"*
- Suitable schedules for loop pipelining are found by **modulo schedulers**

Example

```
for (i = 1 .. N)
{
  t    = a[i-1];
  a[i] = s + t;
  s    = t * t;
}
```

Example

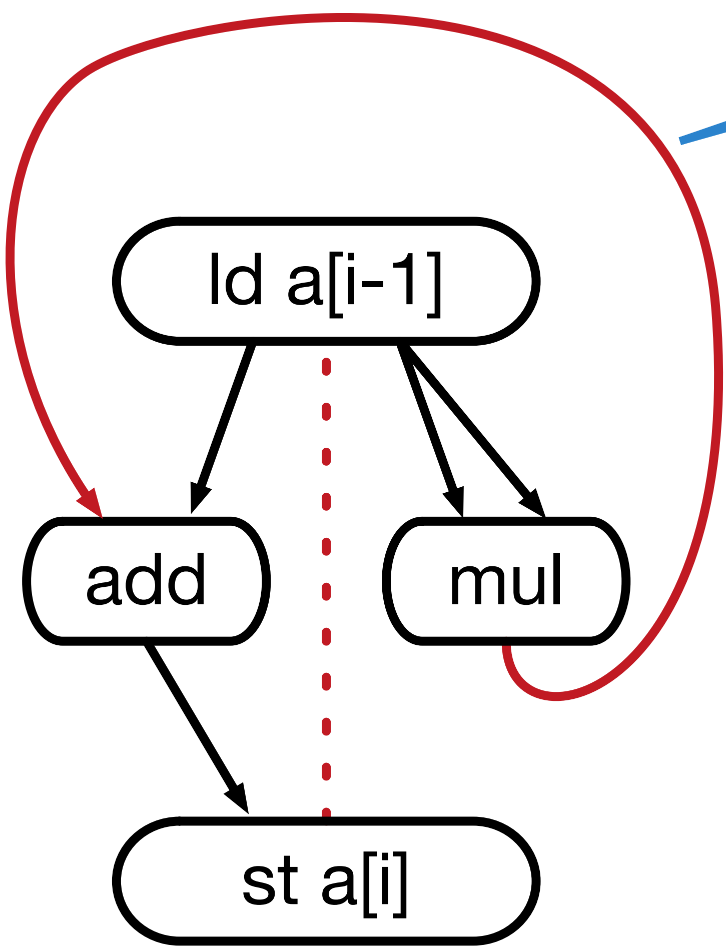
```
for (i = 1 .. N)
{
  t    = a[i-1];
  a[i] = s + t;
  s    = t * t;
}
```



Example

```
for (i = 1 .. N)
{
  t    = a[i-1];
  a[i] = s + t;
  s    = t * t;
}
```

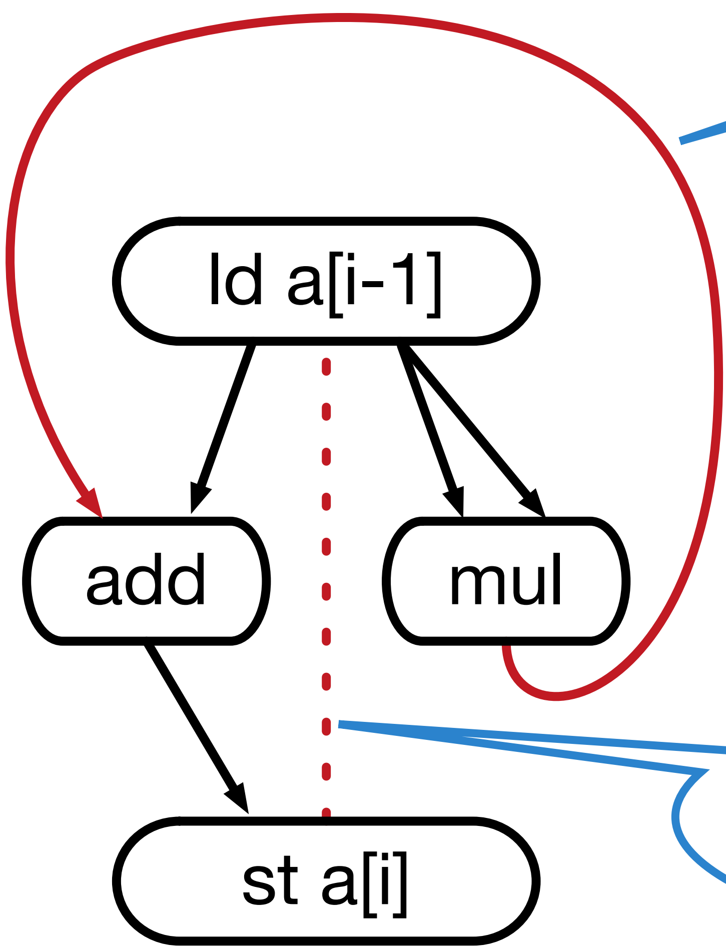
add operation depends on the value of s
from the previous iteration



Example

```
for (i = 1 .. N)
{
  t    = a[i-1];
  a[i] = s + t;
  s    = t * t;
}
```

add operation depends on the value of s
from the previous iteration

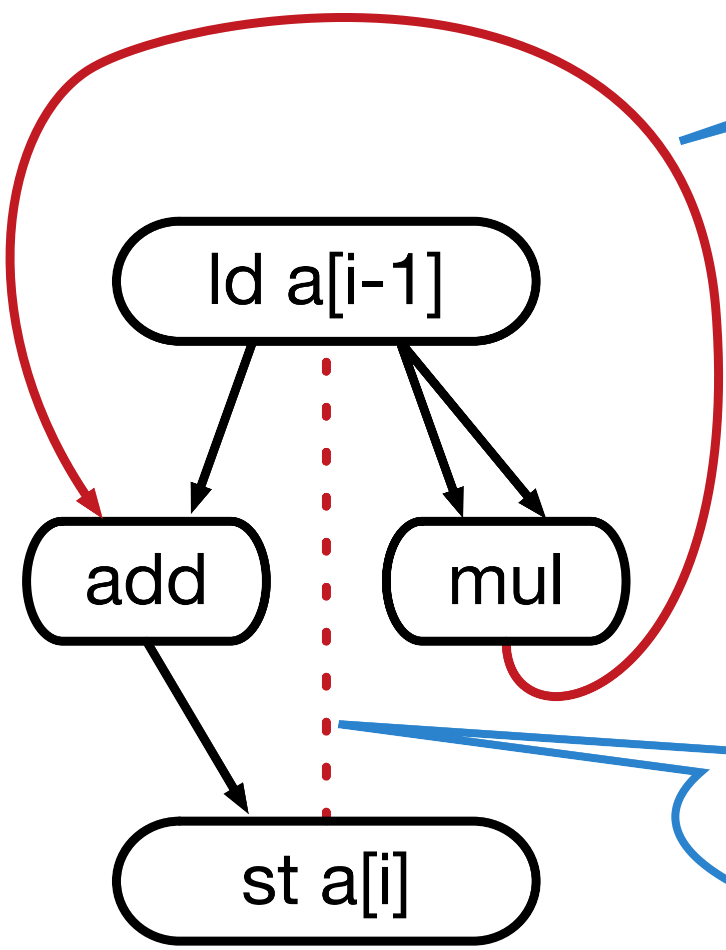


Load value only after it was
written in the previous iteration

Example

```
for (i = 1 .. N)
{
  t    = a[i-1];
  a[i] = s + t;
  s    = t * t;
}
```

add operation depends on the value of s from the previous iteration

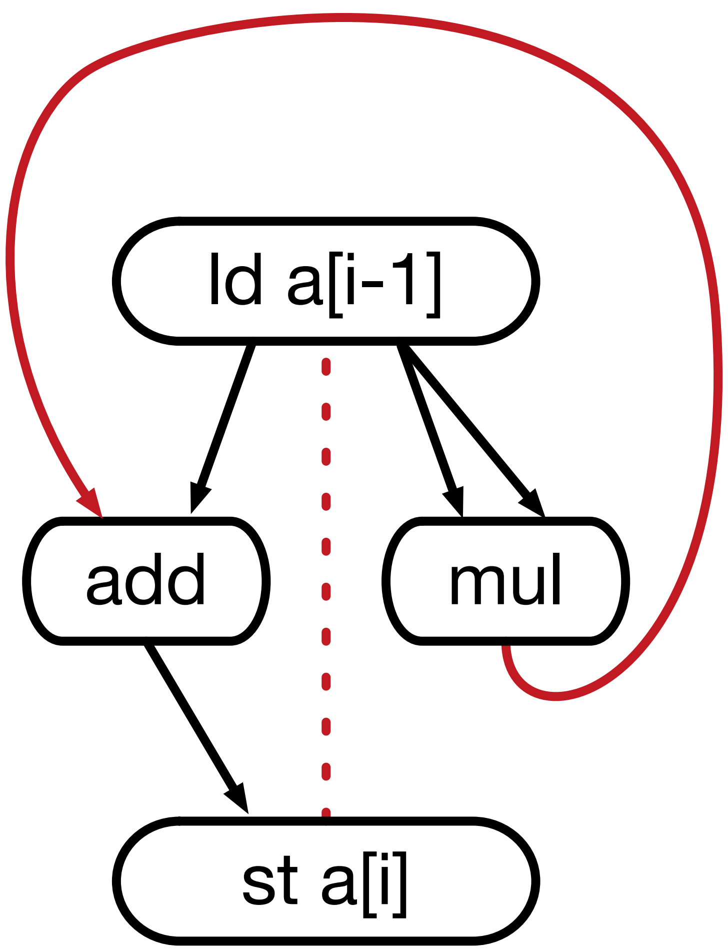


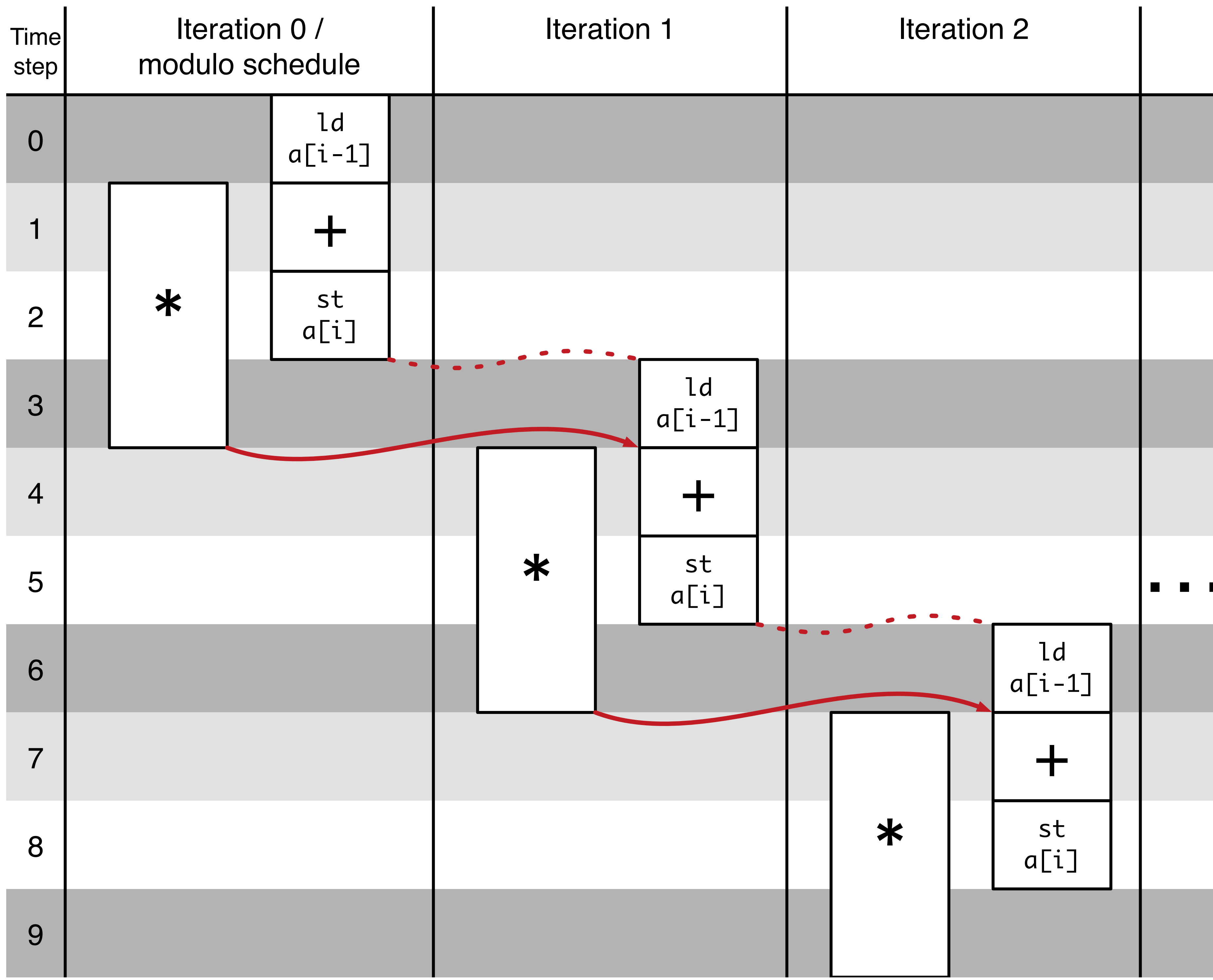
Both edges imply
inter-iteration dependencies
a.k.a “backedges”

Load value only after it was written in the previous iteration

Example

```
for (i = 1 .. N)
{
   t    = a[i-1];
   a[i] = s + t;
   s    = t * t;
}
```





General Approach

- Determine lower and upper bound for the II

General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule

General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule
  - Input: candidate II, precedence edges, resource constraints, operation latencies

General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule
  - Input: candidate II, precedence edges, resource constraints, operation latencies
  - Output: start times for operations, or attempt fails

General Approach

- Here: Compare schedulers based on **Integer Linear Programs** (ILP)

General Approach

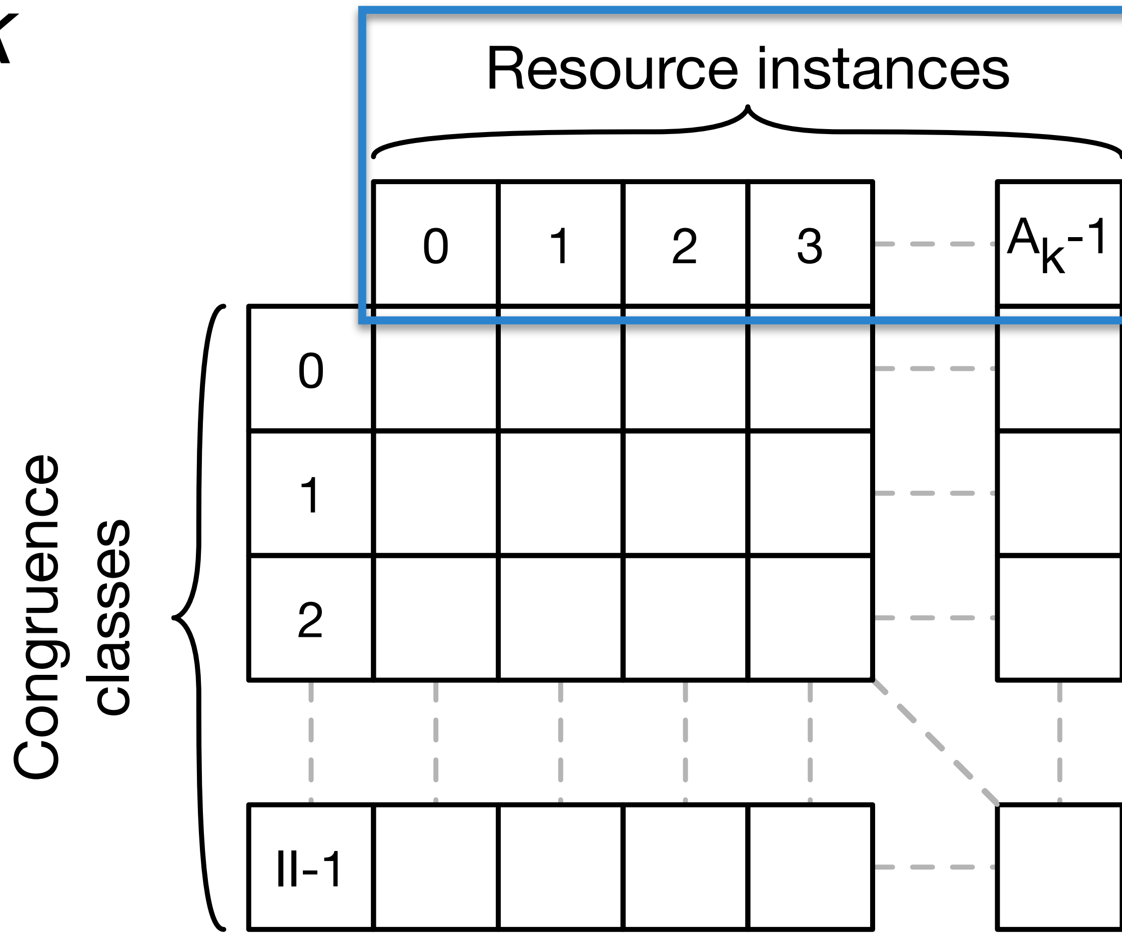
- Here: Compare schedulers based on **Integer Linear Programs** (ILP)
- Scheduling graphs with only typical HLS precedence constraints and backedges is **easy**
  - e.g. as a System of Difference Constraints (SDC), special ILP that can be solved in **polynomial** time

General Approach

- Here: Compare schedulers based on **Integer Linear Programs** (ILP)
- Scheduling graphs with only typical HLS precedence constraints and backedges is **easy**
  - e.g. as a System of Difference Constraints (SDC), special ILP that can be solved in **polynomial** time
- Approaches differ in the modelling of resource constraints

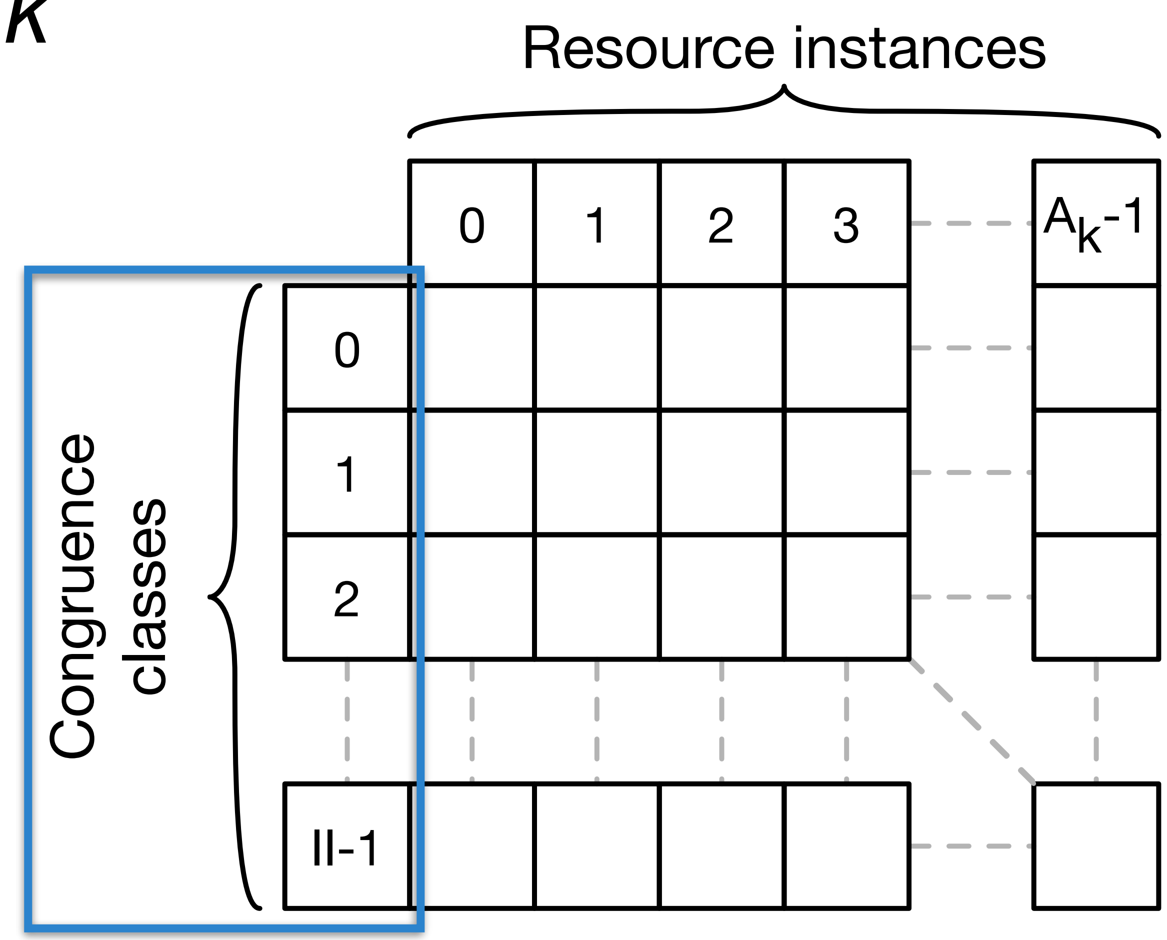
General Approach

- A_k instances/units/… of a certain scarce resource kind k



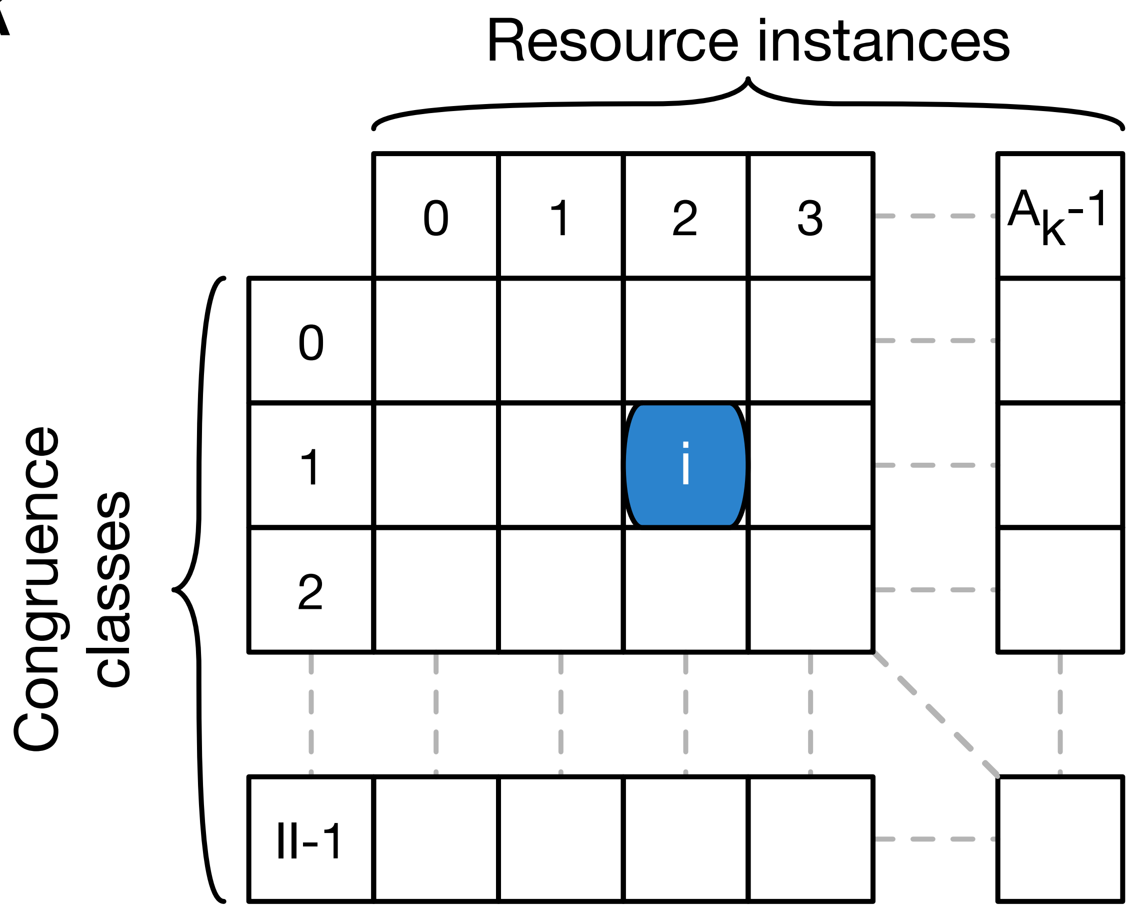
General Approach

- A_k instances/units/… of a certain scarce resource kind k
- Candidate II ⇒ congruence classes of operations' start times



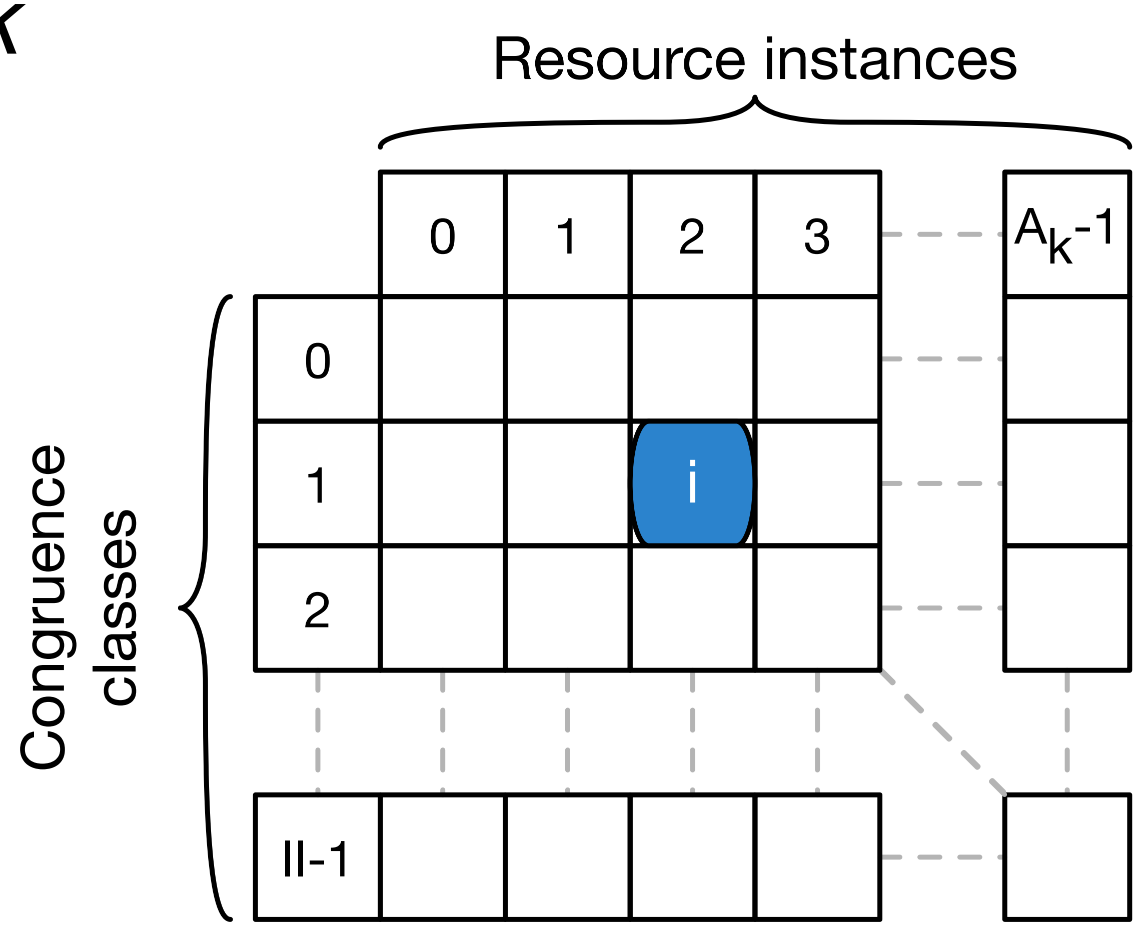
General Approach

- A_k instances/units/… of a certain scarce resource kind k
- Candidate II ⇒ congruence classes of operations' start times
- Each instance can be used once per congruence class by an operation i



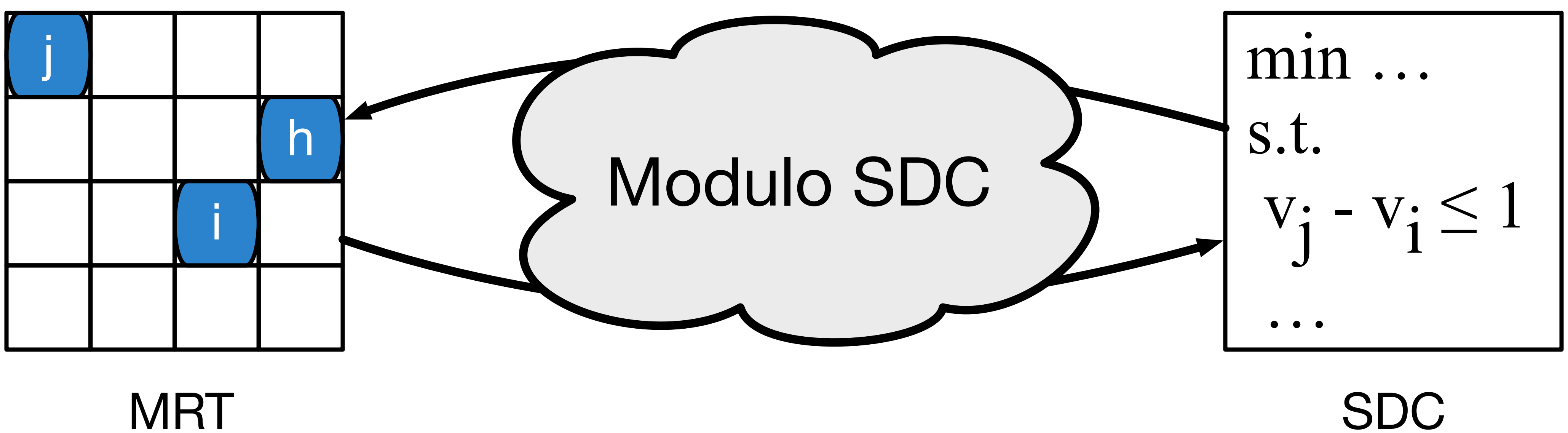
General Approach

- A_k instances/units/… of a certain scarce resource kind k
- Candidate II ⇒ congruence classes of operations' start times
- Each instance can be used once per congruence class by an operation i
- *"modulo reservation table"* (MRT)



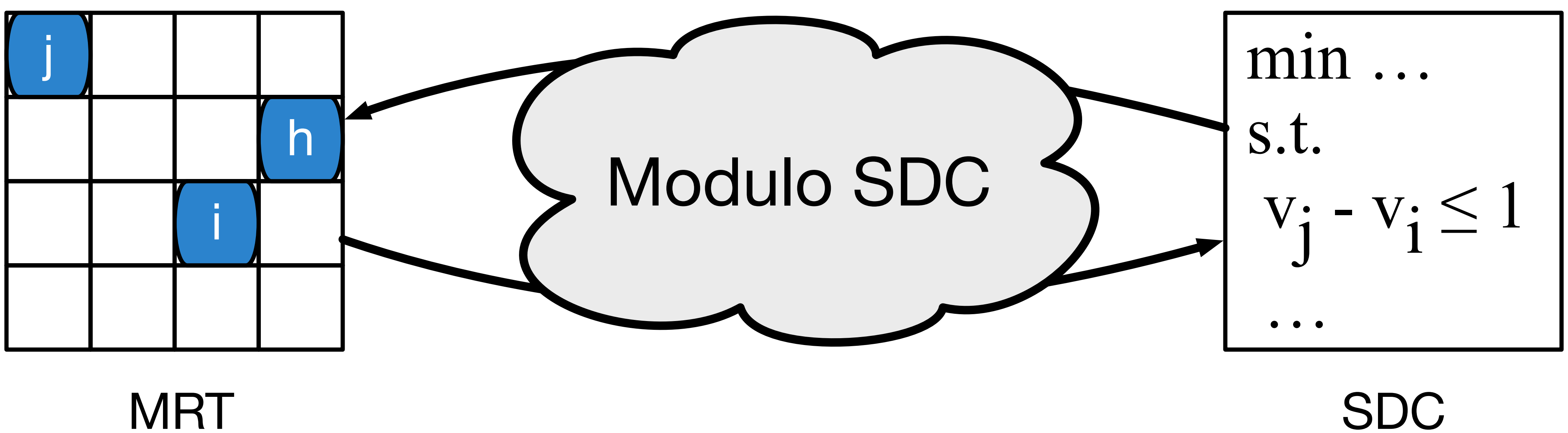
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT



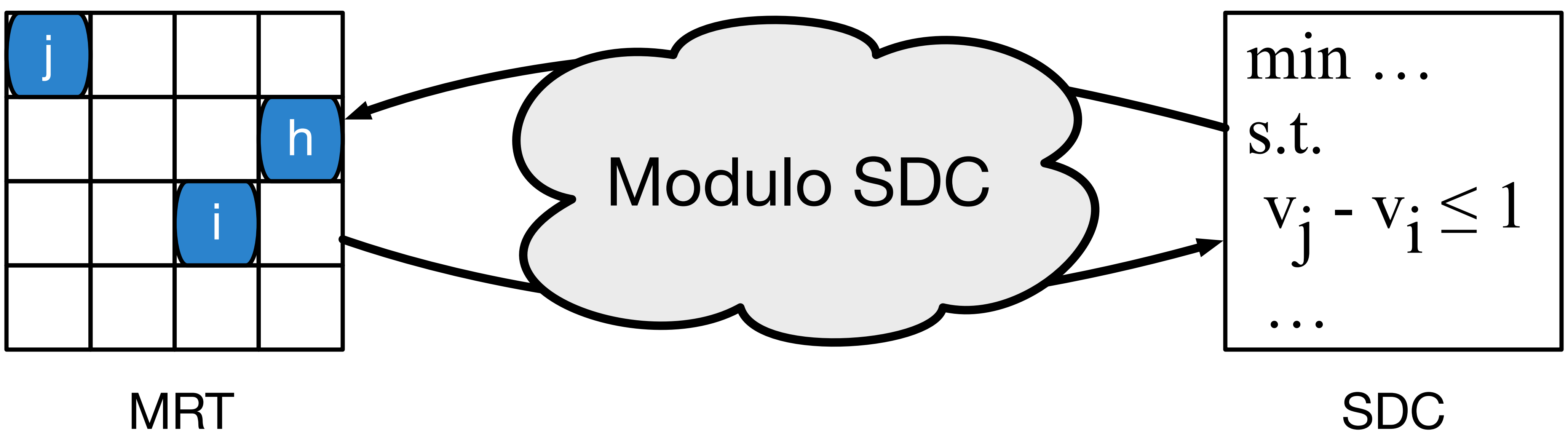
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-**un**constrained schedule



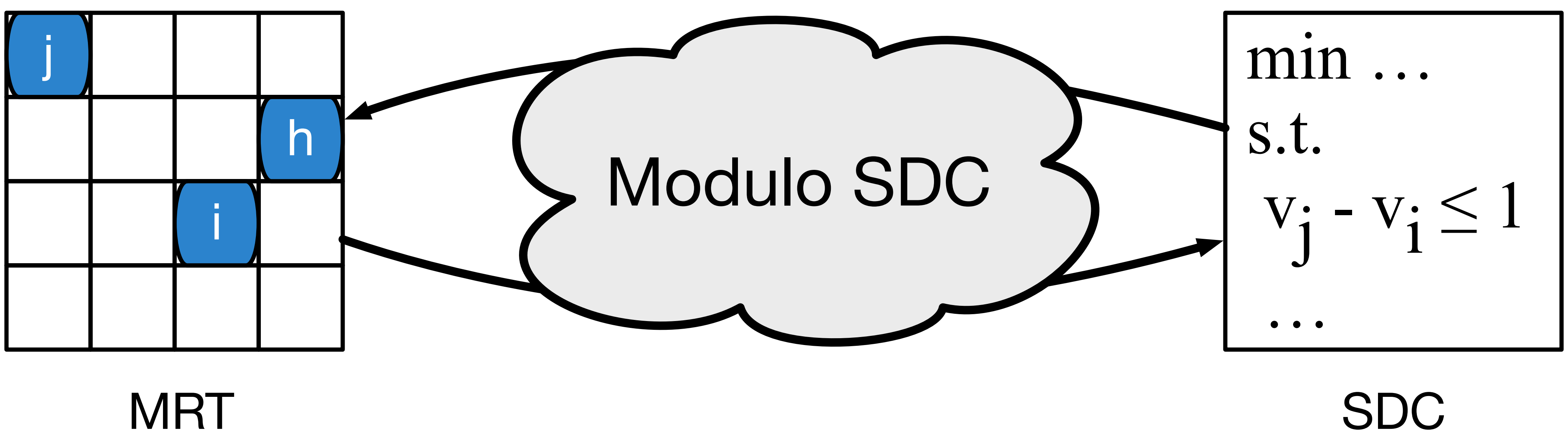
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-**un**constrained schedule
  - Incrementally try to assign operations to MRT and add constraints to SDC



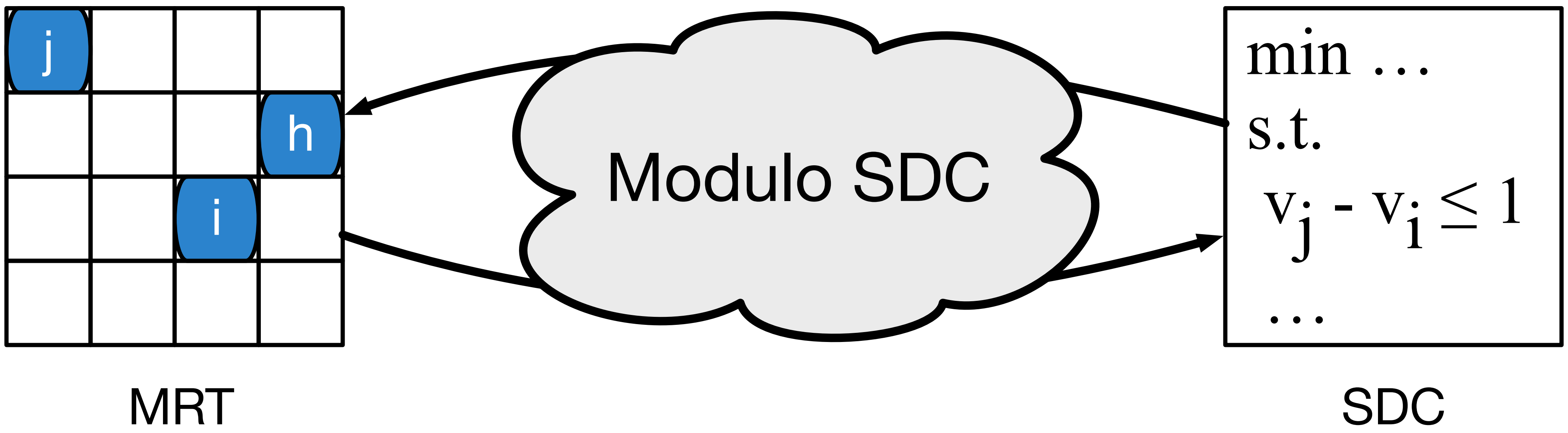
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-**un**constrained schedule
  - Incrementally try to assign operations to MRT and add constraints to SDC
  - Backtracking required if SDC becomes infeasible



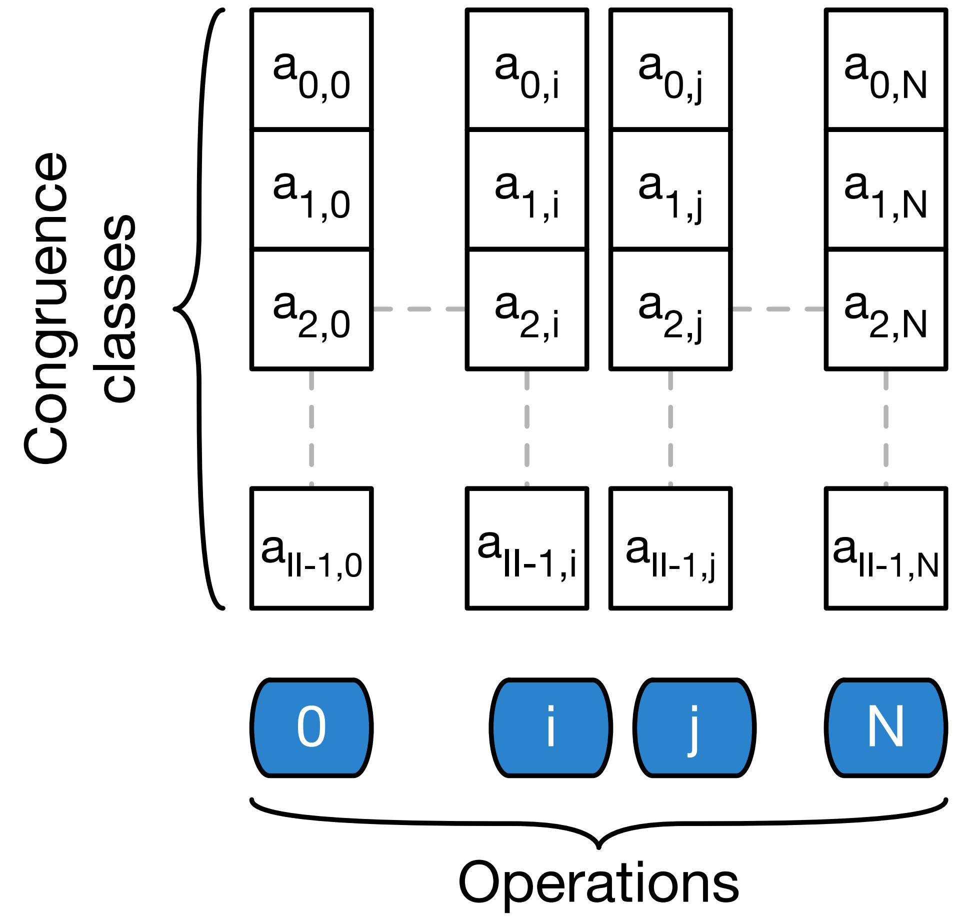
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-**un**constrained schedule
  - Incrementally try to assign operations to MRT and add constraints to SDC
  - Backtracking required if SDC becomes infeasible
  - Successful if all resource-constrained ops fit in MRT



Eichenberger's Formulation

- **Exact** formulation
  general ILP with time-indexed binary variables
  $a_{m,i}$:= "operation i starts in congruence class m"



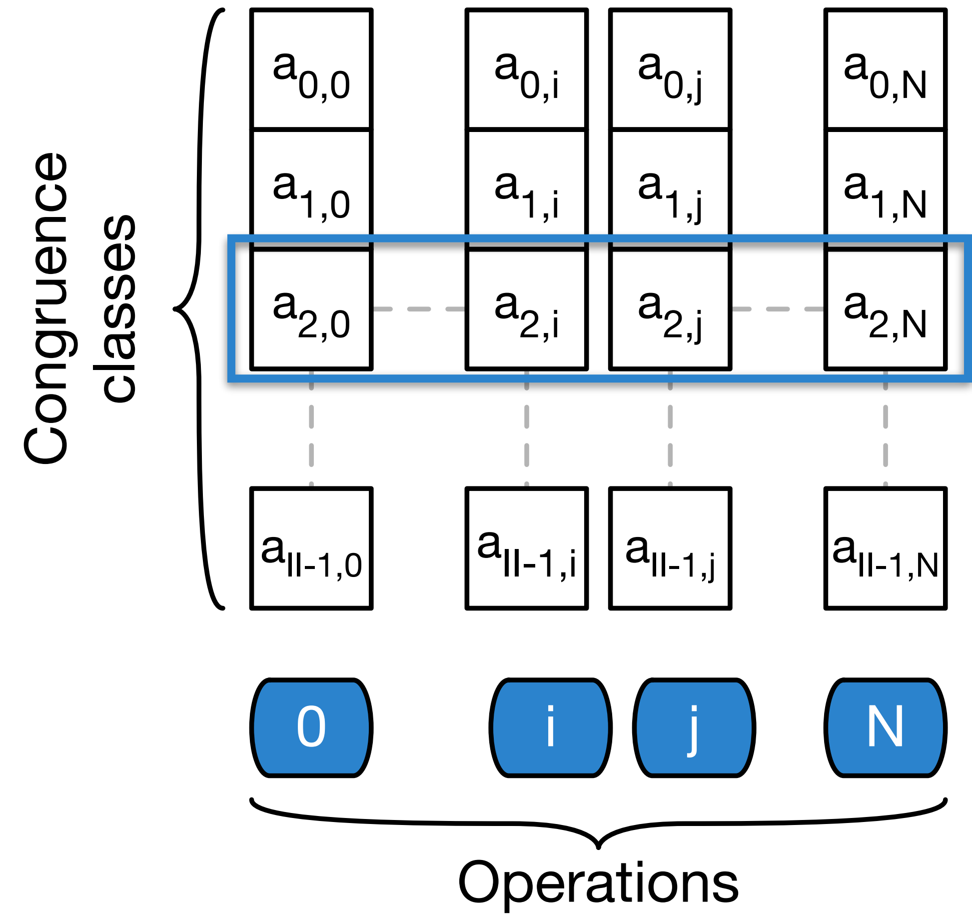
Eichenberger's Formulation

- **Exact** formulation
  general ILP with time-indexed binary variables
  $a_{m,i}$:= "operation *i* starts in congruence class *m*"

- Example: Resource constraint for
  kind *k*, congruence class 2
  fulfilled iff.

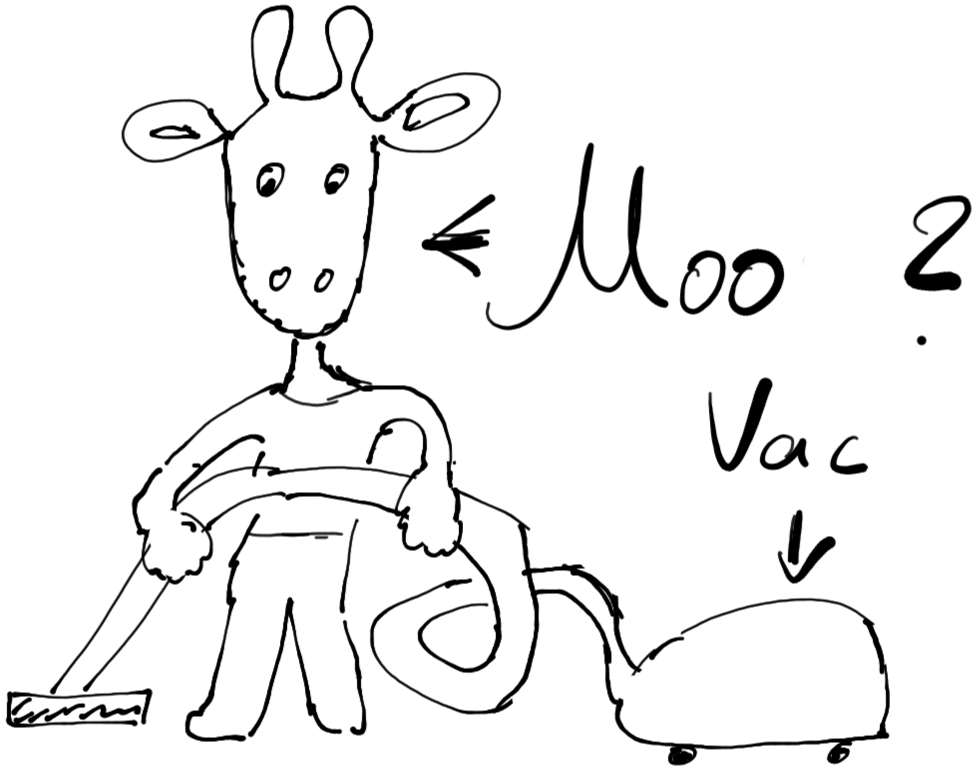
  $$\sum_x a_{2,x} \leq A_k$$

  for all operations *x* that use
  a *k*-resource



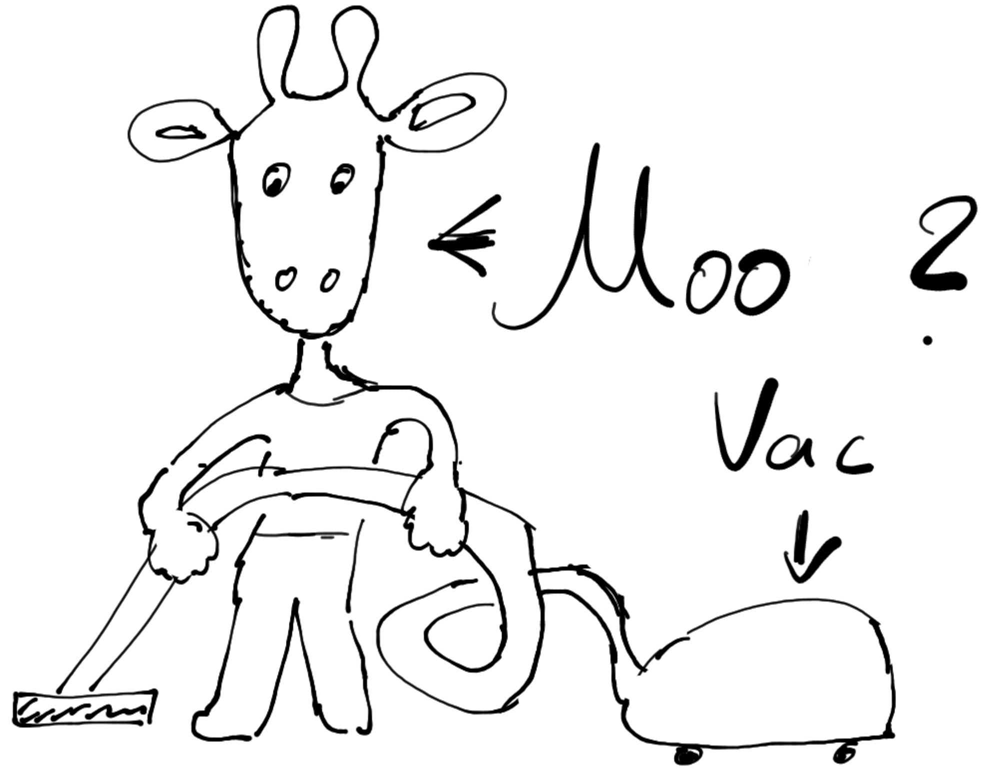
Moovac

- Moovac = **Mo**dulo **O**verlap **Va**riable **C**onstraints
- Adapted task scheduling formulation
  based on overlap variables



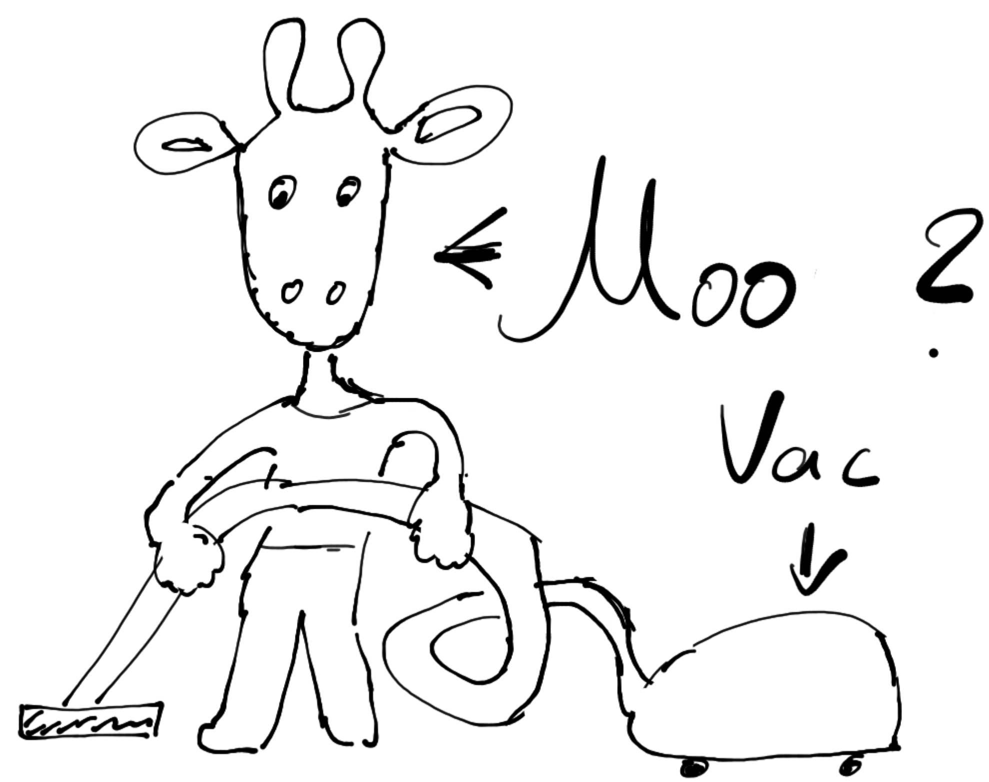
Moovac

- Moovac = **Mo**dulo **O**verlap **Va**riable **C**onstraints
- Adapted task scheduling formulation based on overlap variables
- **Exact** formulation, general ILP



Moovac

- Moovac = **Mo**dulo **O**verlap **Va**riable **C**onstraints
- Adapted task scheduling formulation based on overlap variables
- **Exact** formulation, general ILP
- **Integer** variables model start times t_i



Moovac

- Let i, j be operations that require a resource of kind k

Moovac

- Let i, j be operations that require a resource of kind k
- Resource assignment modelled by
  - Integer variables
    r_i resource instance ID $\in [0 \ldots A_k - 1]$
    m_i congruence class ID $\in [0 \ldots candidate\ II - 1]$

Moovac

- Let i, j be operations that require a resource of kind k
- Resource assignment modelled by
  - Integer variables
    r_i resource instance ID $\in [0 \dots A_k - 1]$
    m_i congruence class ID $\in [0 \dots candidate\ II - 1]$
  - Binary overlap variables
    $\varepsilon_{ij} := 1$ iff. $r_i < r_j$
    $\mu_{ij} := 1$ iff. $m_i < m_j$

Moovac

- Let i, j be operations that require a resource of kind k
- Resource assignment modelled by
  - Integer variables
    r_i resource instance ID $\in [0 \dots A_k - 1]$
    m_i congruence class ID $\in [0 \dots candidate\ II - 1]$
  - Binary overlap variables
    $\varepsilon_{ij} := 1$ iff. $r_i < r_j$
    $\mu_{ij} := 1$ iff. $m_i < m_j$

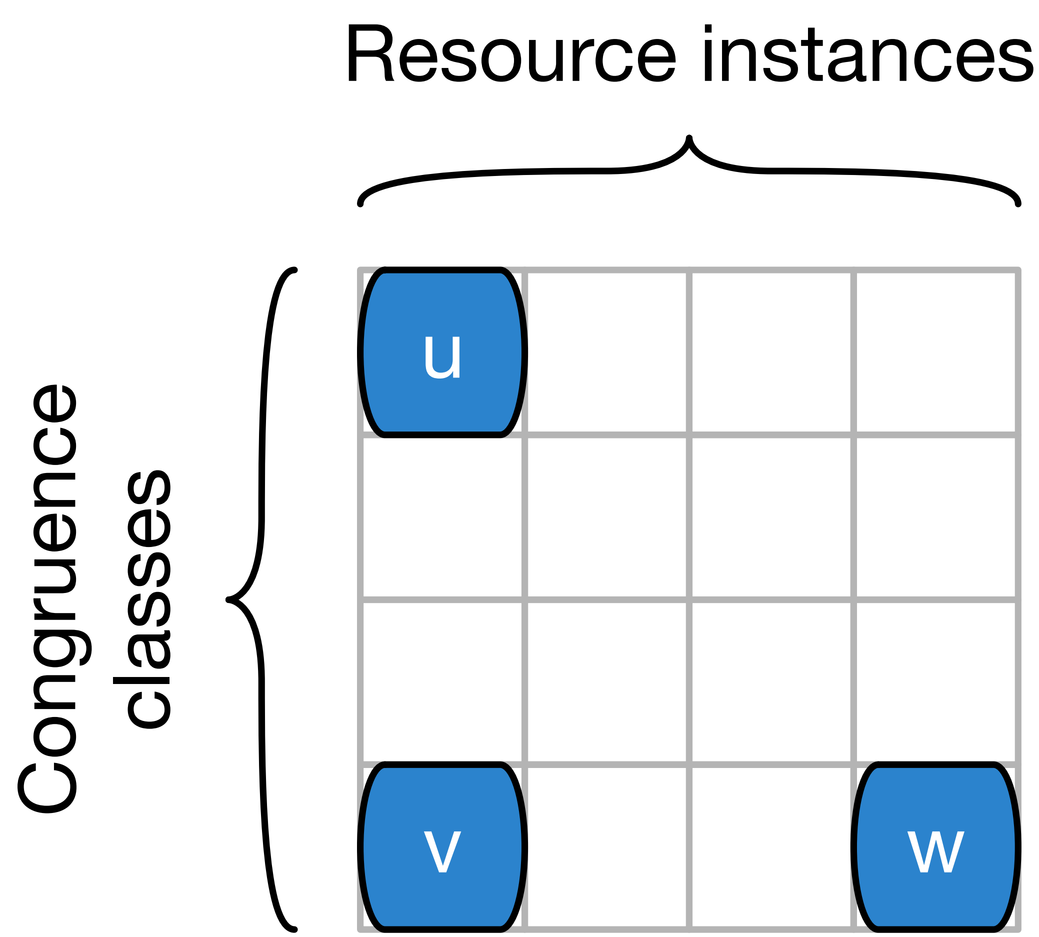
"i and j are either assigned to **different resource instances**, or scheduled to **different congruence classes**"

- No resource conflict iff.

$$\varepsilon_{ij} + \varepsilon_{ji} + \mu_{ij} + \mu_{ji} \geq 1$$

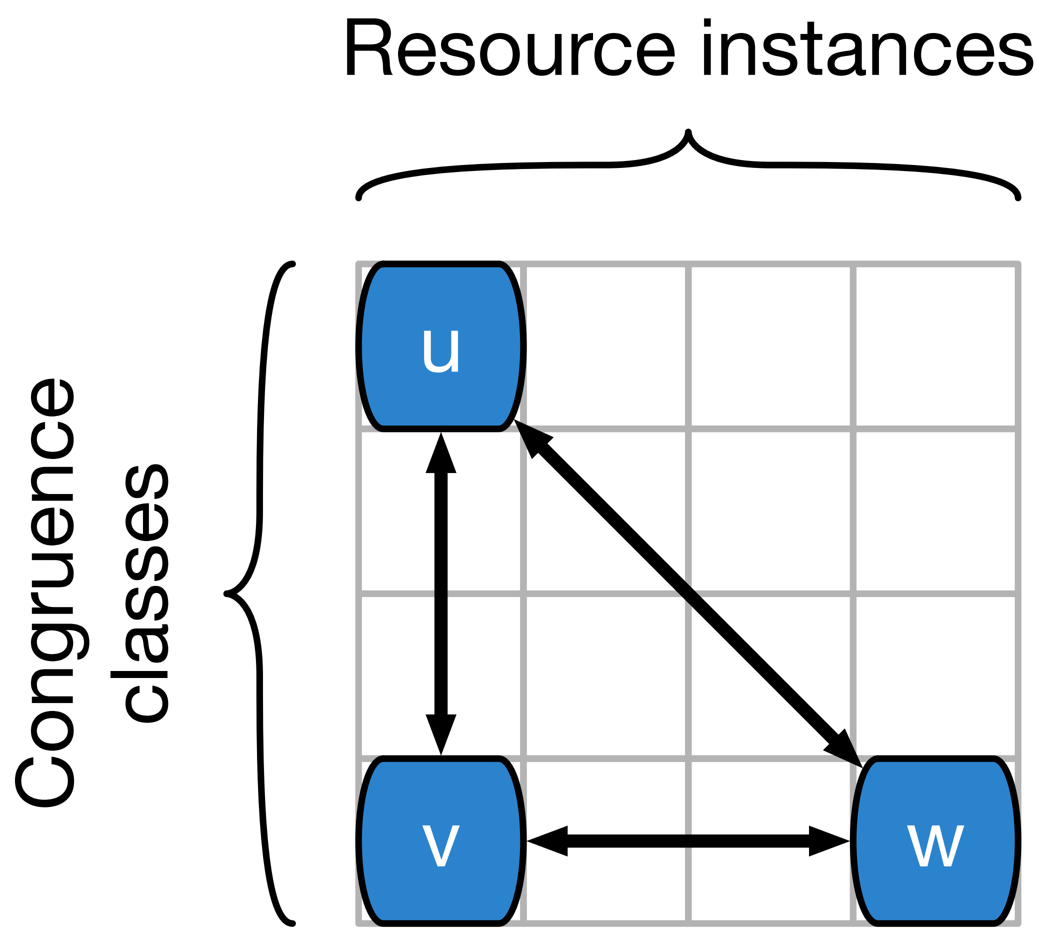
Moovac

- Tuples *(m_i, r_i)* ⇒ cell in MRT for operation *i*



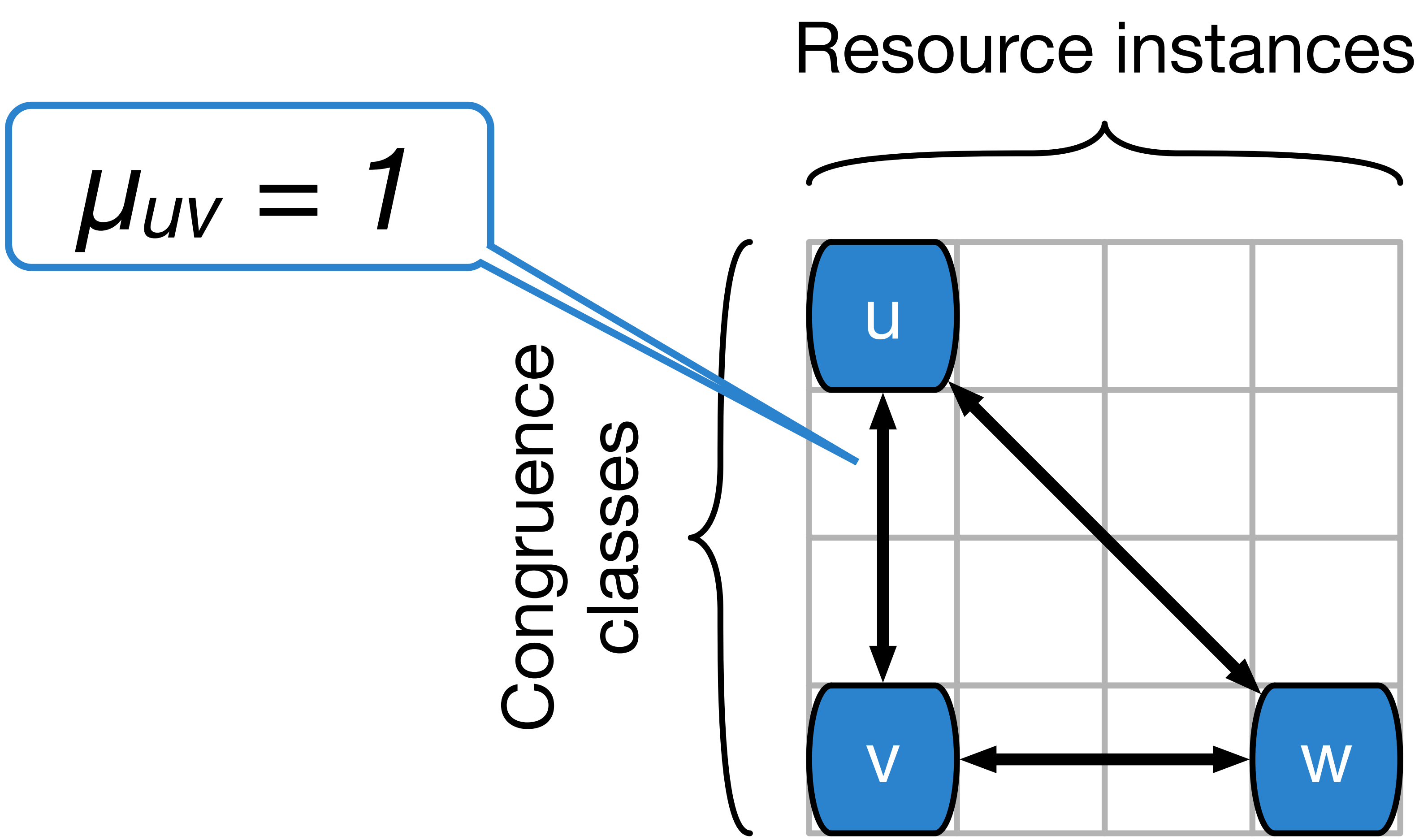
Moovac

- Tuples $(m_i, r_i) \Rightarrow$ cell in MRT for operation i
- Overlap variables model relations between operations



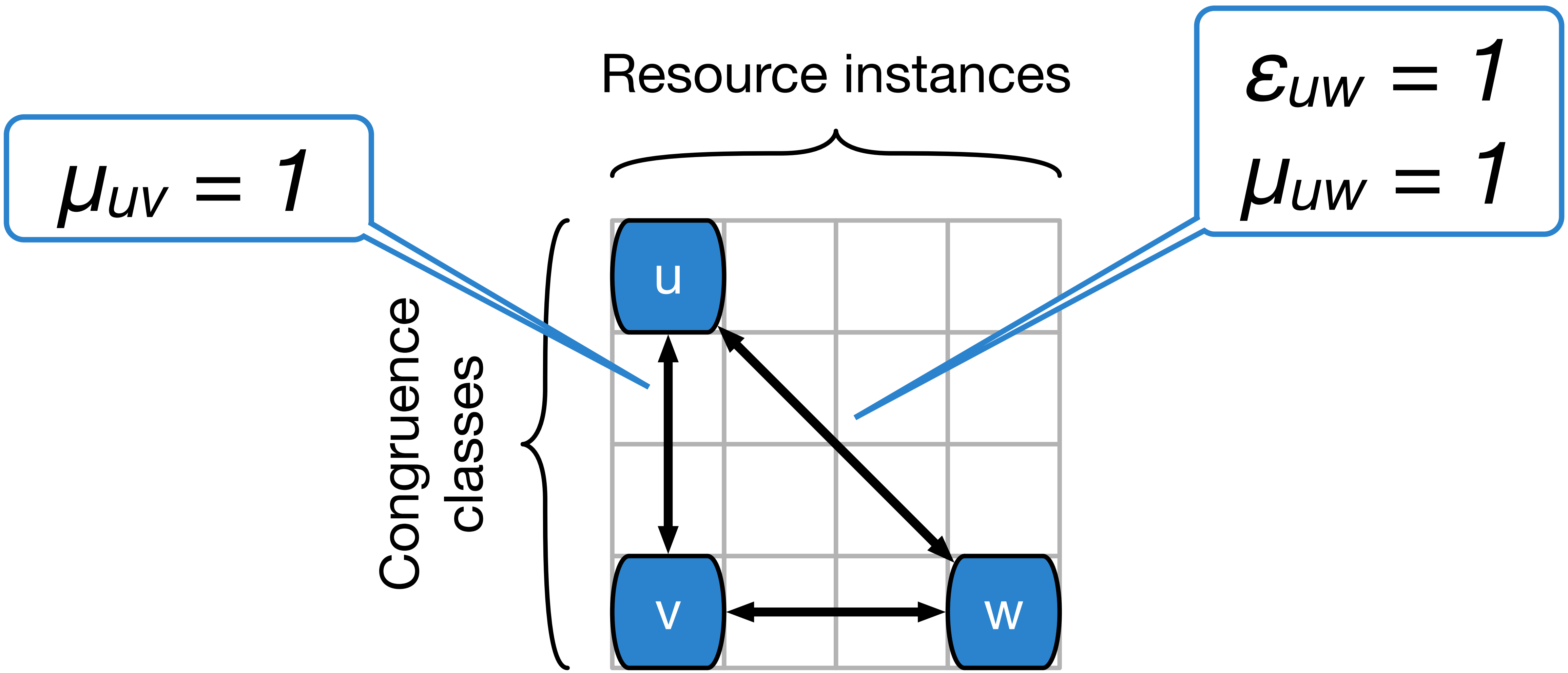
Moovac

- Tuples (m_i, r_i) ⇒ cell in MRT for operation i
- Overlap variables model relations between operations



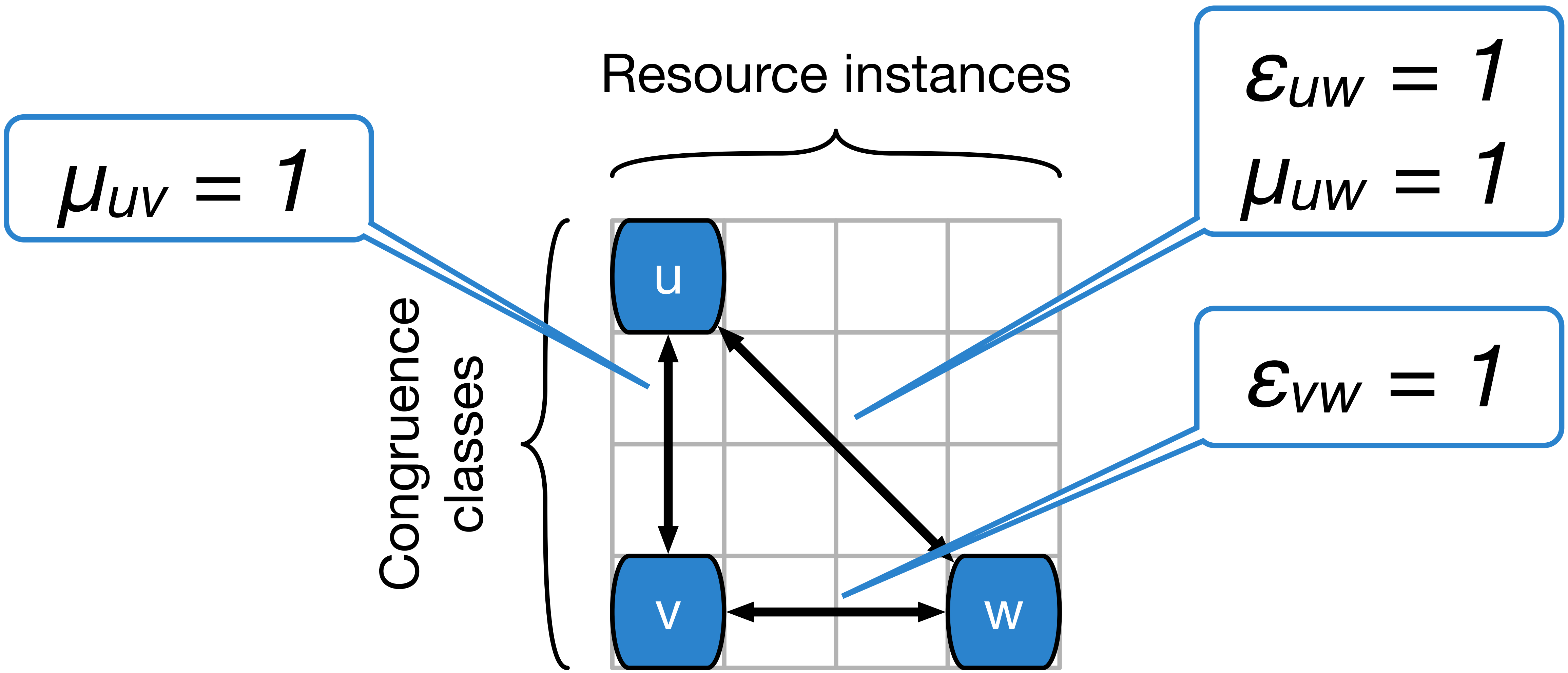
Moovac

- Tuples $(m_i, r_i) \Rightarrow$ cell in MRT for operation i
- Overlap variables model relations between operations



Moovac

- Tuples $(m_i, r_i) \Rightarrow$ cell in MRT for operation i
- Overlap variables model relations between operations



Approaches At A Glance

Modulo SDC
Canis et al.

- Resource constraints are not part of the linear program
- Operations are assigned **heuristically** to MRT

Approaches At A Glance

Modulo SDC
Canis et al.

- Resource constraints are not part of the linear program
- Operations are assigned **heuristically** to MRT

Formulation by
Eichenberger &
Davidson

- **Exact** formulation
- Time-indexing → large number of binary variables, complicated constraints

Approaches At A Glance

Modulo SDC
Canis et al.

- Resource constraints are not part of the linear program
- Operations are assigned **heuristically** to MRT

Formulation by **Eichenberger** & Davidson

- **Exact** formulation
- Time-indexing → large number of binary variables, complicated constraints

Moovac
Oppermann et al.

- Novel **exact** formulation
- Uses fewer integer variables and overlap variables to model inequality between them

Evaluation

- Schedulers implemented with CPLEX 12.6.3

Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667's at 3.3 GHz

Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667’s at 3.3 GHz
- Time limit of **5 min** or 60 min **per candidate II**
  - increment II if no solution was found

Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667's at 3.3 GHz
- Time limit of **5 min** or 60 min **per candidate II**
  - increment II if no solution was found
- Attempted to schedule 225 graphs from CHStone and MachSuite
  - up to 1124 operations /
up to 107 resource-constrained operations

Results (Quality)

- 5 min time limit

Graphs		Moovac vs. Modulo SDC *shorter II found by …*			Moovac vs. Eichenberger's ILP *shorter II found by …*		
Size	*#*	*Moovac*	*Same*	*M. SDC*	*Moovac*	*Same*	*E.B.'s*
all	225	6	217	2	6	219	0
small	203	1	202	0	0	203	0
large	22	5	15	2	6	16	0

Results (Quality)

- 5 min time limit

Graphs		**Moovac vs. Modulo SDC** *shorter II found by …*			**Moovac vs. Eichenberger's ILP** *shorter II found by …*		
Size	*#*	*Moovac*	*Same*	*M. SDC*	*Moovac*	*Same*	*E.B.'s*
all	225	6	217	2	6	219	0
small	203	1	202	0	0	203	0
large	22	5	15	2	6	16	0

Modulo SDC delivers high-quality results

Results (Quality)

- 5 min time limit

Graphs		**Moovac vs. Modulo SDC** *shorter II found by …*			**Moovac vs. Eichenberger's ILP** *shorter II found by …*		
Size	*#*	*Moovac*	*Same*	*M. SDC*	*Moovac*	*Same*	*E.B.'s*
all	225	6	217	2	6	219	0
small	203	1	202	0	0	203	0
large	22	5	15	2	6	16	0

Modulo SDC delivers high-quality results

Modulo SDC found schedules where Moovac ran out of time

Results (Quality)

- 5 min time limit

Graphs		Moovac vs. Modulo SDC *shorter II found by …*			Moovac vs. Eichenberger's ILP *shorter II found by …*		
Size	*#*	*Moovac*	*Same*	*M. SDC*	*Moovac*	*Same*	*E.B.'s*
all	225	6	217	2	6	219	0
small	203	1	202	0	0	203	0
large	22	5	15	2	6	16	0

Modulo SDC delivers high-quality results

Modulo SDC found schedules where Moovac ran out of time

Exact schedulers should find same II, but E.B. hit time limit

Results (Time)

- Scheduling duration with 5 min time limit:

Graphs		Moovac		Modulo SDC		Eichenberger's ILP	
Size	*#*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*
all	225	489	96	753	148	932	177
small	203	3	0	131	26	5	0
large	22	486	96	623	122	927	177

Results (Time)

- Scheduling duration with 5 min time limit:

Graphs		Moovac		Modulo SDC		Eichenberger's ILP	
Size	*#*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*
all	225	489	96	753	148	932	177
small	203	3	0	131	26	5	0
large	22	486	96	623	122	927	177

Moovac is faster than the other approaches

Results (Time)

- Scheduling duration with 5 min time limit:

Graphs		Moovac		Modulo SDC		Eichenberger's ILP	
Size	*#*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*
all	225	489	96	753	148	932	177
small	203	3	0	131	26	5	0
large	22	$\boxed{486}$	96	623	122	927	177

Moovac is faster than the other approaches

The timeouts dominate the overall time e.g. 96 x 5 min = 480 min

Results (Time)

- Scheduling duration with 5 min time limit:

Graphs		Moovac		Modulo SDC		Eichenberger's ILP	
Size	*#*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*	*Time [min]*	*Timeouts*
all	225	489	96	753	148	932	177
small	203	3	0	131	26	5	0
large	22	486	96	623	122	927	177

Moovac is faster than the other approaches

The timeouts dominate the overall time e.g. 96 x 5 min = 480 min

M. SDC seems to get stuck even on small graphs

Insights

- How can an exact formulation be faster overall than the heuristic?

Insights

- How can an exact formulation be faster overall than the heuristic?
    - ILP solver “sees” whole problem, can **prove** infeasibility of scheduling attempt (often: **fast**)

Insights

- How can an exact formulation be faster overall than the heuristic?
  - ILP solver “sees” whole problem, can **prove** infeasibility of scheduling attempt (often: **fast**)
  - Heuristic can only fail to find a solution in the given time budget

Insights

- Modulo SDC and Moovac complement each other

Insights

- Modulo SDC and Moovac complement each other
- "Synergistic scheduling"

Moovac:	489 min
Modulo SDC:	753 min
Combined:	**429 min**

Insights

- What makes Moovac better suited for HLS modulo scheduling than Eichenberger's ILP?

Insights

- What makes Moovac better suited for HLS modulo scheduling than Eichenberger's ILP?
    - Up to 1000+ operations, candidate IIs > 50 require humongous amounts of decision variables in time-indexed formulation

Insights

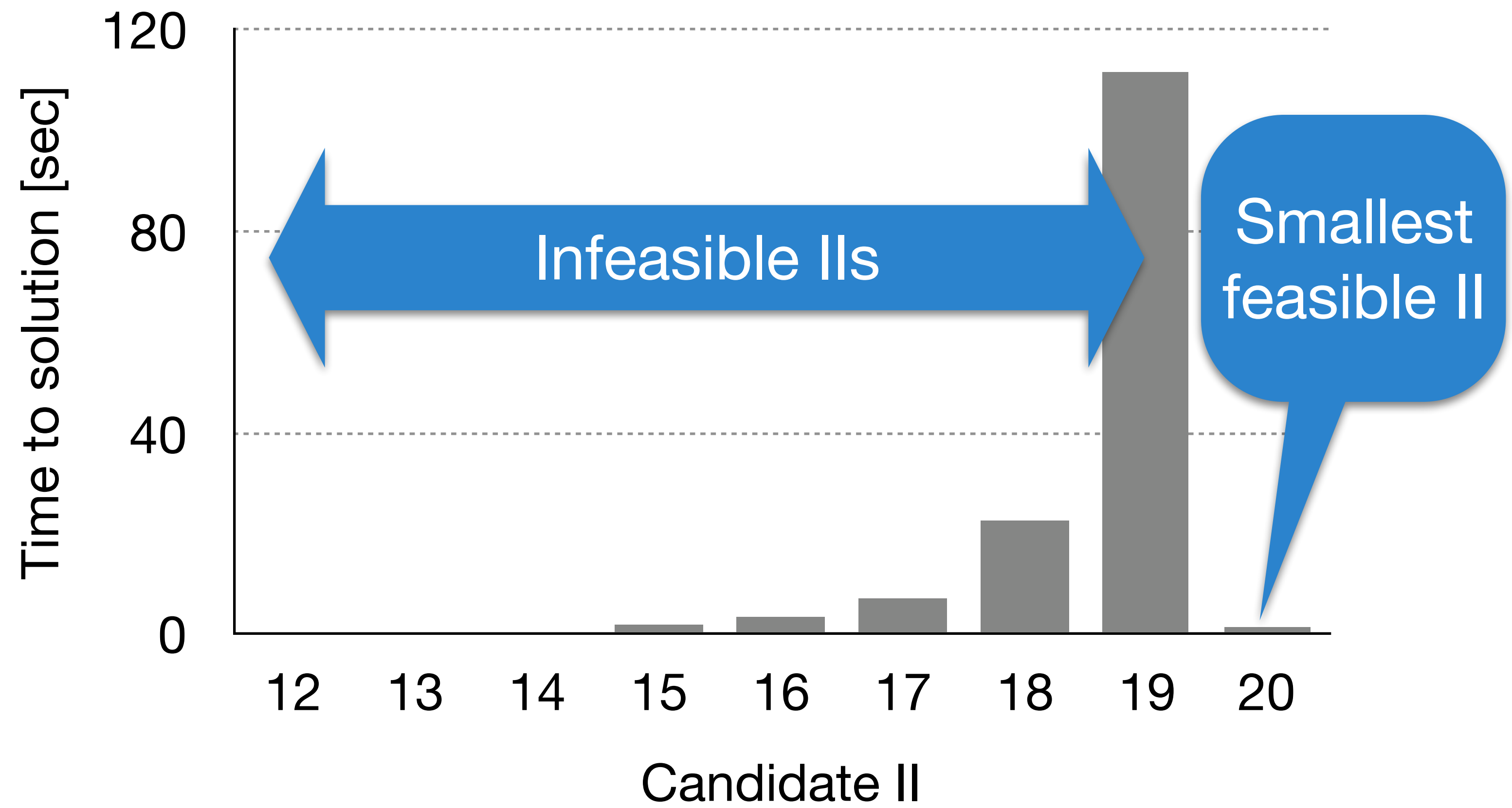
- What makes Moovac better suited for HLS modulo scheduling than Eichenberger's ILP?
  - Up to 1000+ operations, candidate IIs > 50 require humongous amounts of decision variables in time-indexed formulation
  - Majority of ops is unconstrained, only subject to precedence constraints and exempt from all MRT handling in Moovac

Outlook

- Smarter search through the (rather large) II space

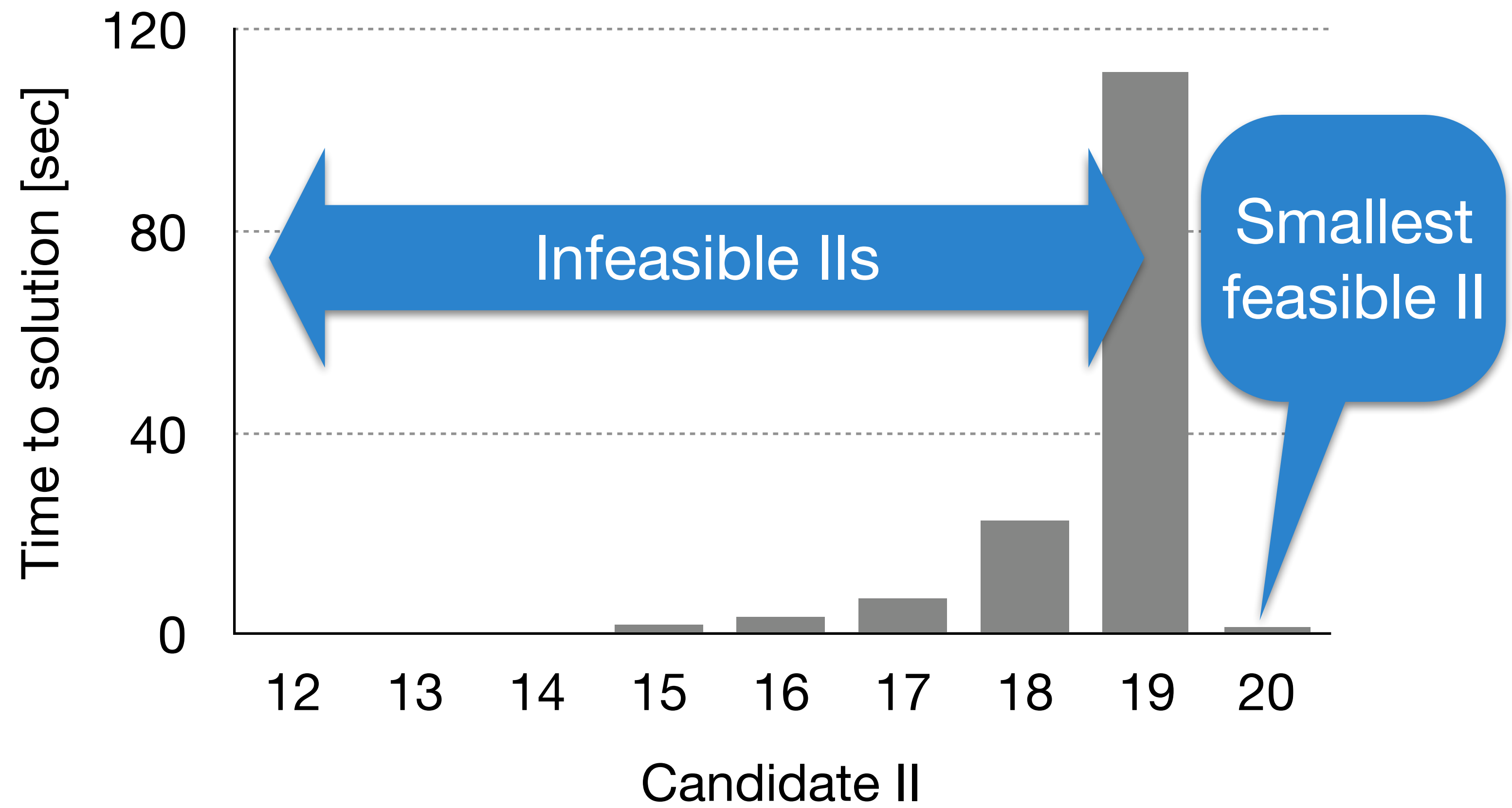
Outlook

- Smarter search through the (rather large) II space
  - Observation:



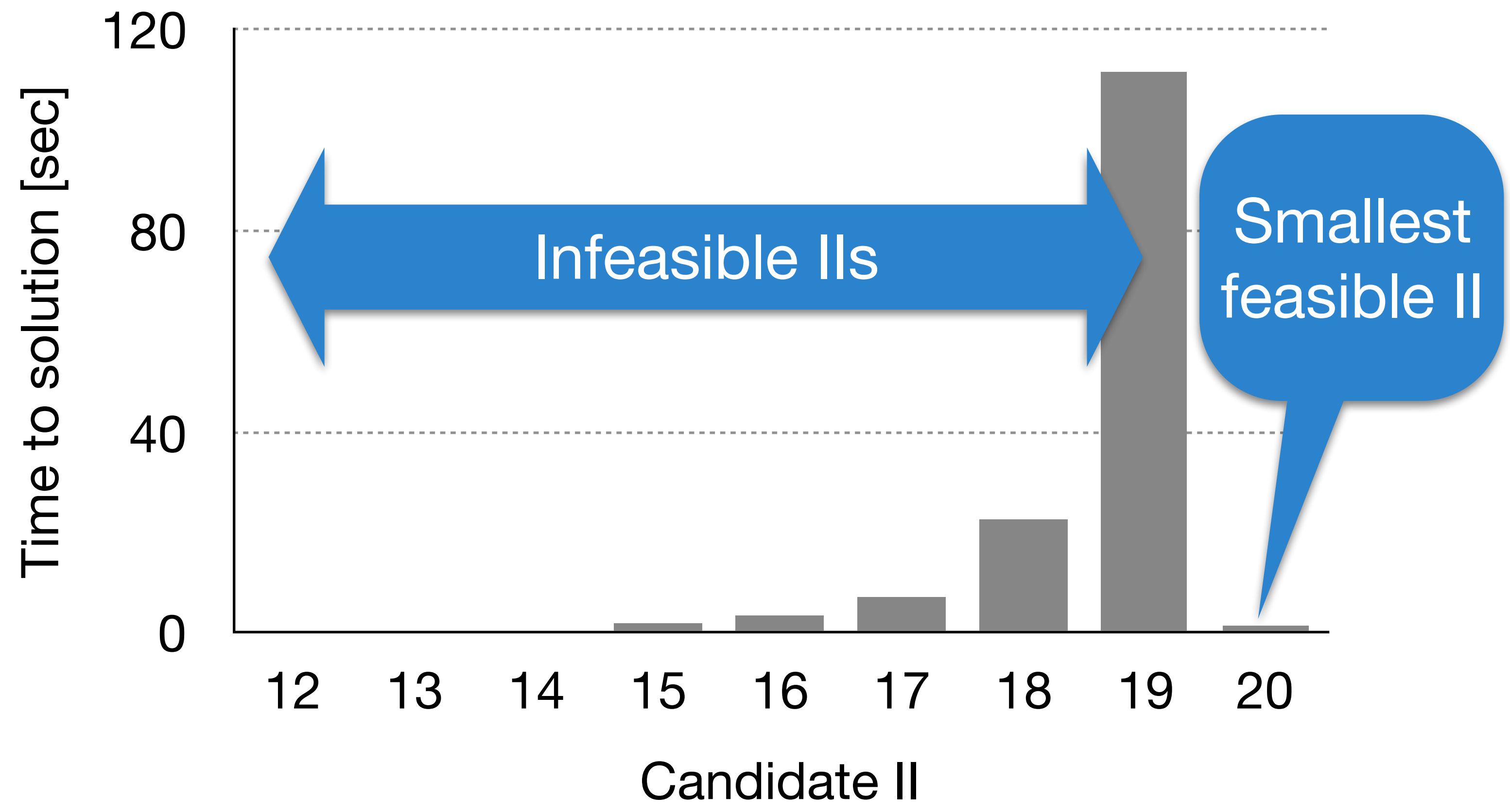
Outlook

- Smarter search through the (rather large) II space
  - Observation:
  - MaxII → MinII ?



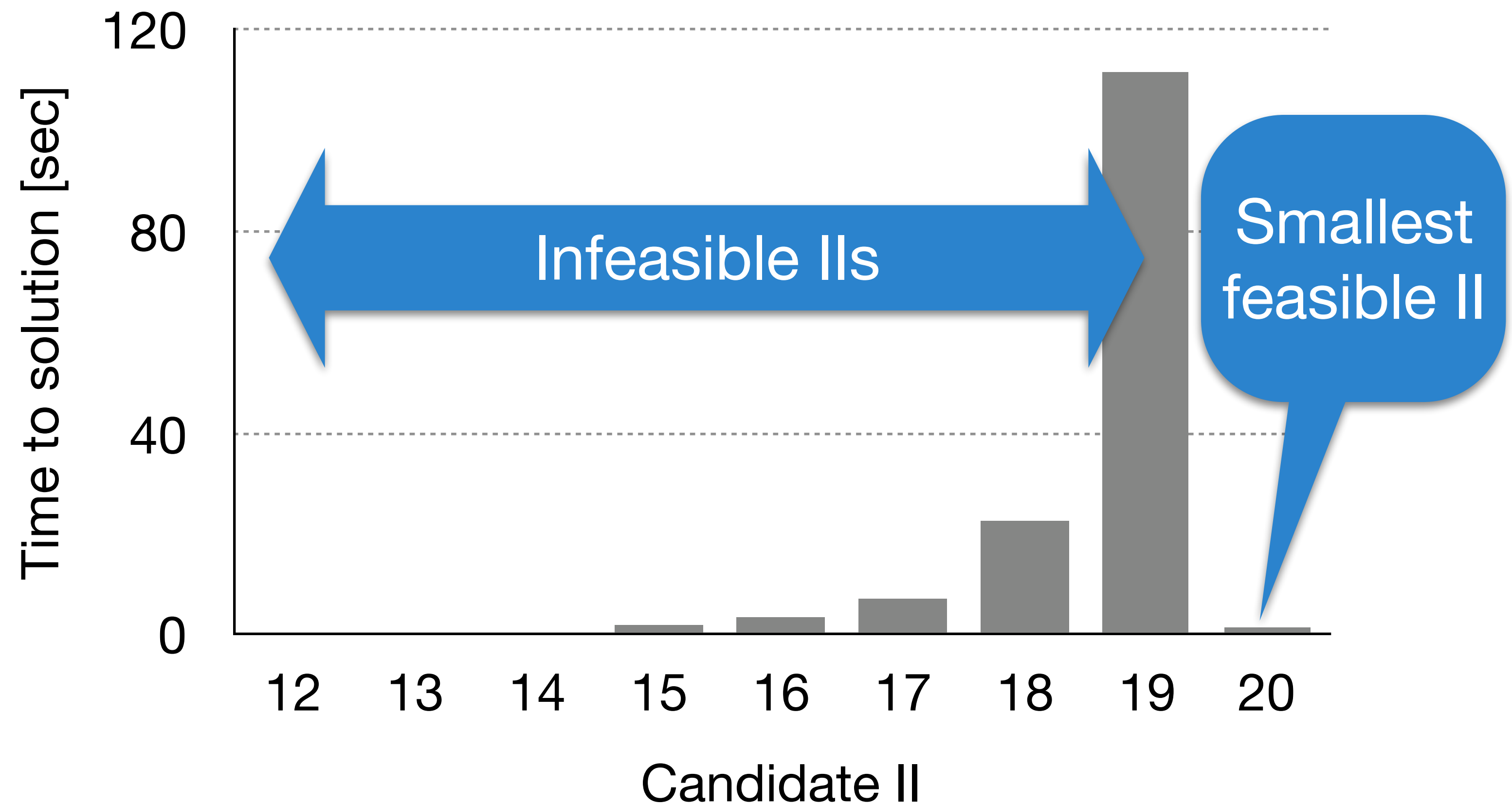
Outlook

- Smarter search through the (rather large) II space
  - Observation:
  - MaxII → MinII ?
  - Binary search ?



Outlook

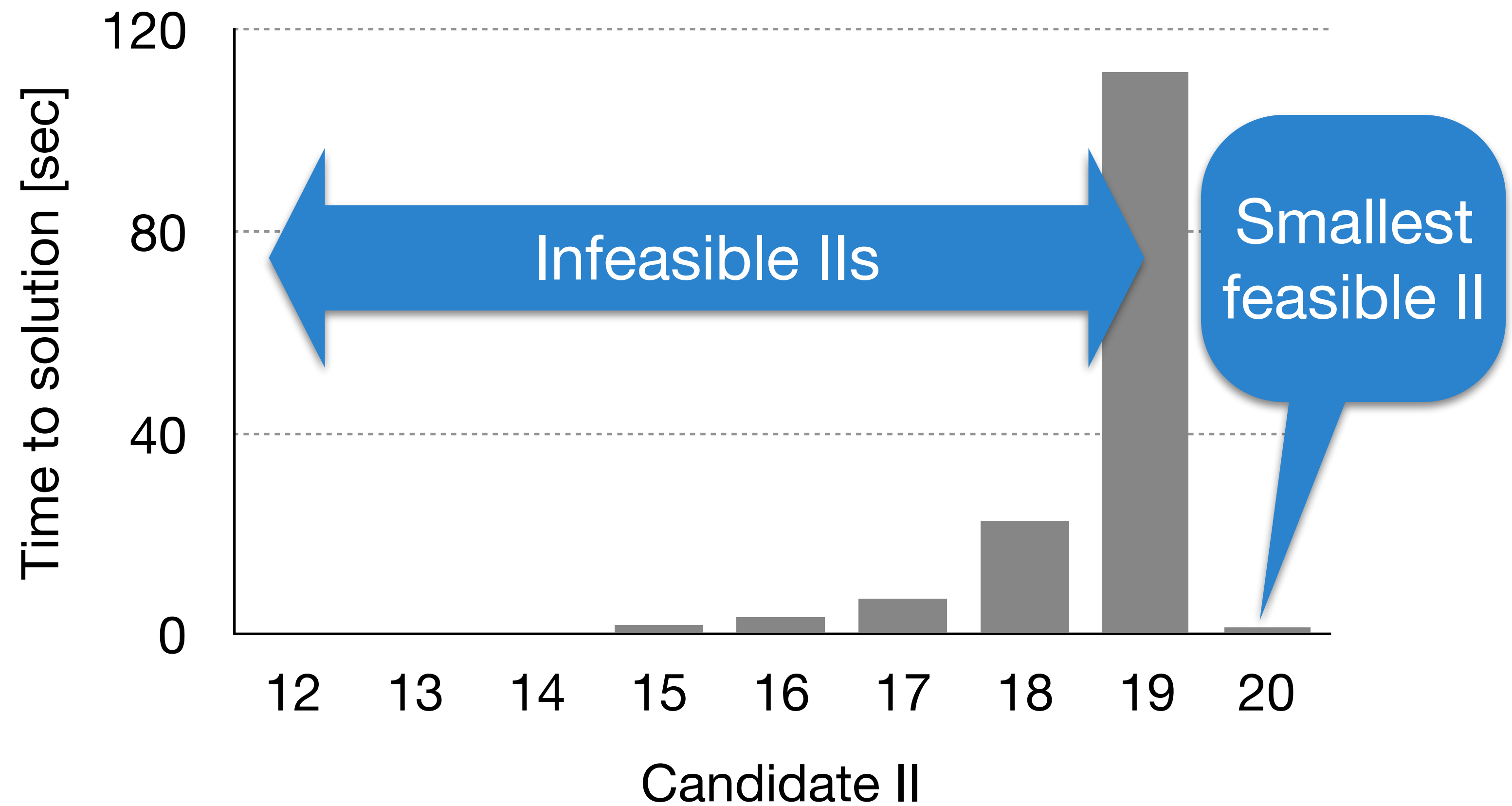
- Smarter search through the (rather large) II space
  - Observation:
  - MaxII → MinII ?
  - Binary search ?



- Integrate II search into the Moovac formulation

Outlook

- Smarter search through the (rather large) II space
  - Observation:
  - MaxII → MinII ?
  - Binary search ?



- Integrate II search into the Moovac formulation
  - Time-indexed formulations:
    # decision variables dependent on candidate II

Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops

Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations

Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations
- The novel, exact Moovac formulation is surprisingly practical in its time-limited mode

Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations
- The novel, exact Moovac formulation is surprisingly practical in its time-limited mode
- Diverse options to reduce the scheduling time even further exist

Thank you!

oppermann@esa.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

KIT Karlsruhe Institute of Technology

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND