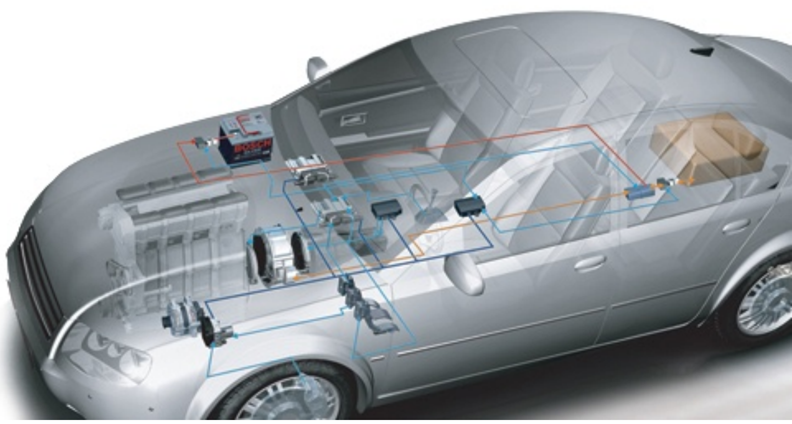
Data-flow: Mapping and Scheduling

Context

- Embedded Systems
- Multimedia Systems
- MP-SoC (MultiProcessor-System-on-Chip)

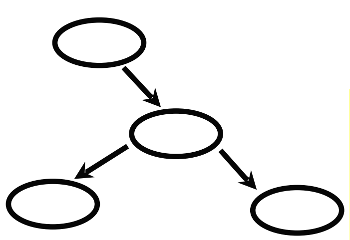


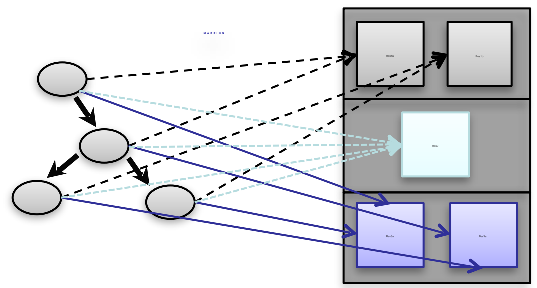
- Multimedia Applications
  - Stream Computing based on Data-Flow Model

The Problem

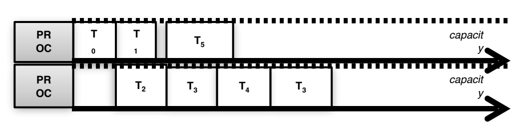
Streaming Application

Data-Flow MoC



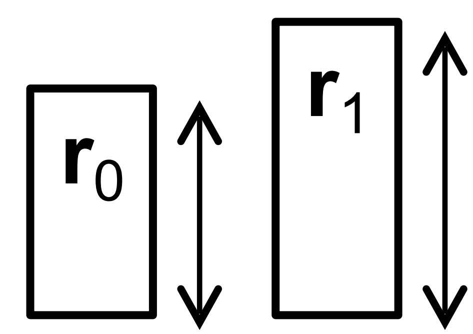


Mapping and Scheduling



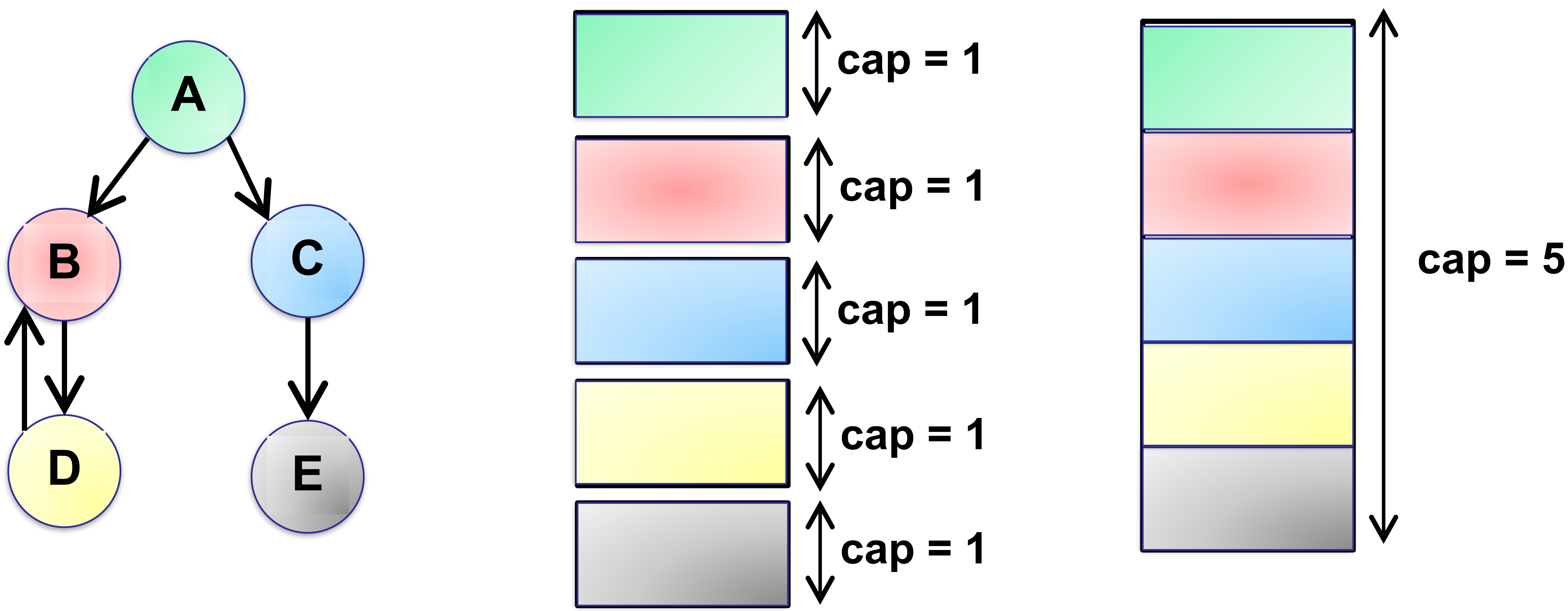
Target Platform

Resources
(CPU, Memories, buffers,…)



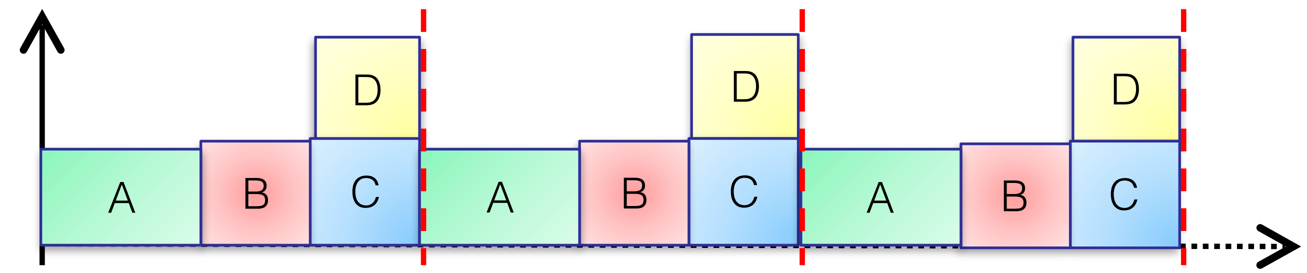
Complete approach based on Constraint Programming

Implicit Mapping

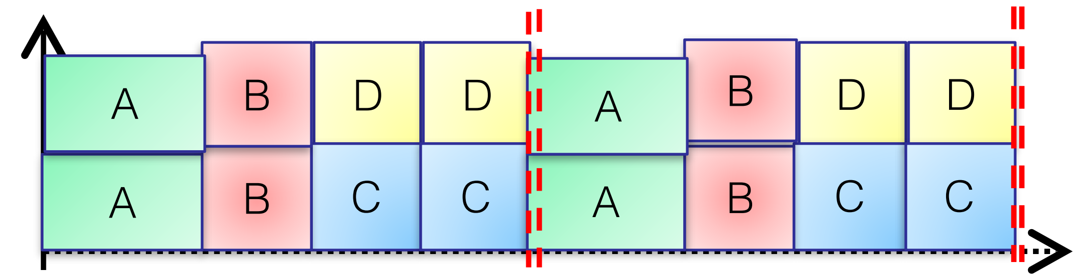


Scheduling

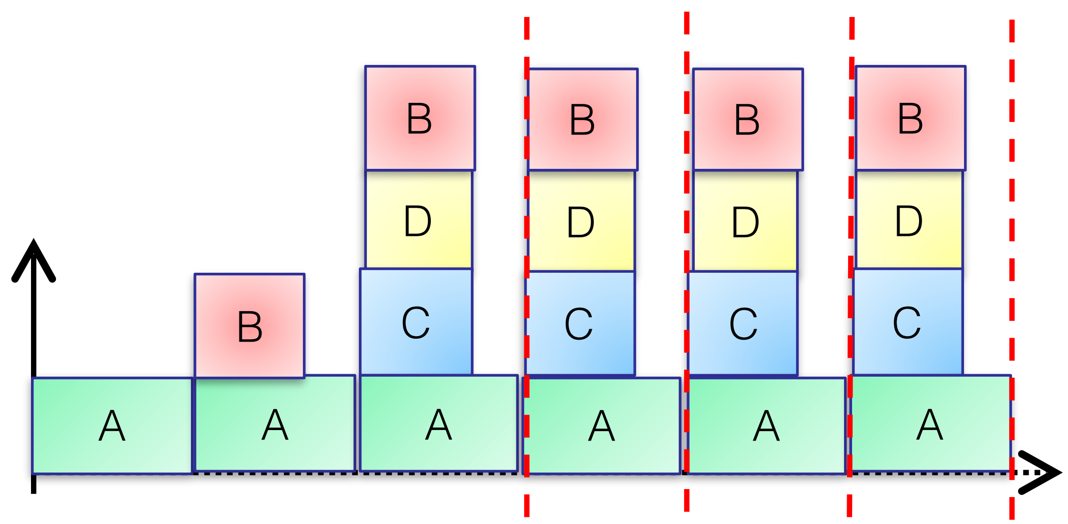
- Blocked Scheduling



- Unfolding Approach Scheduling



- **Modulo Scheduling**

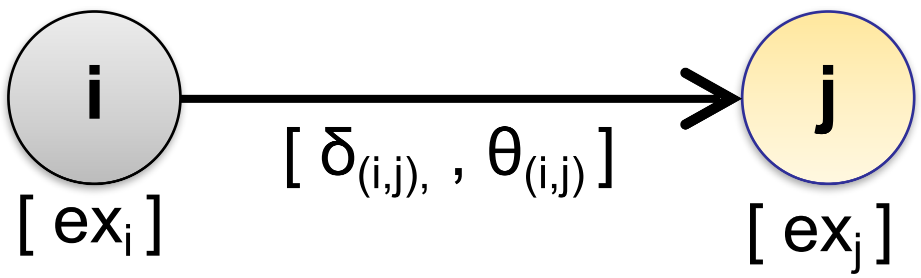


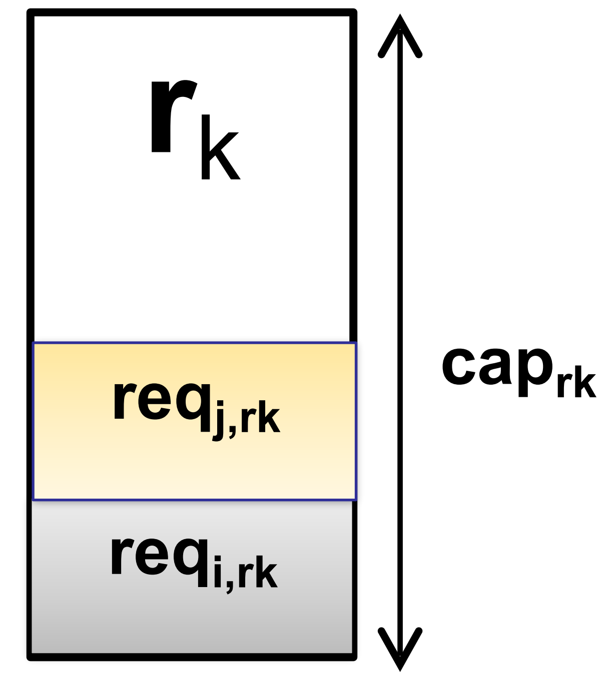
Outline

- Definition of the Problem
- Constraint Programming
- Solver: the Model
- Solver: the Search
- Experimental Results
- Current and Future Research

(Modulus) Problem

$\delta_{(i,j)}$ is the repetition distance
$\theta_{(i,j)}$ is the minimum time lag





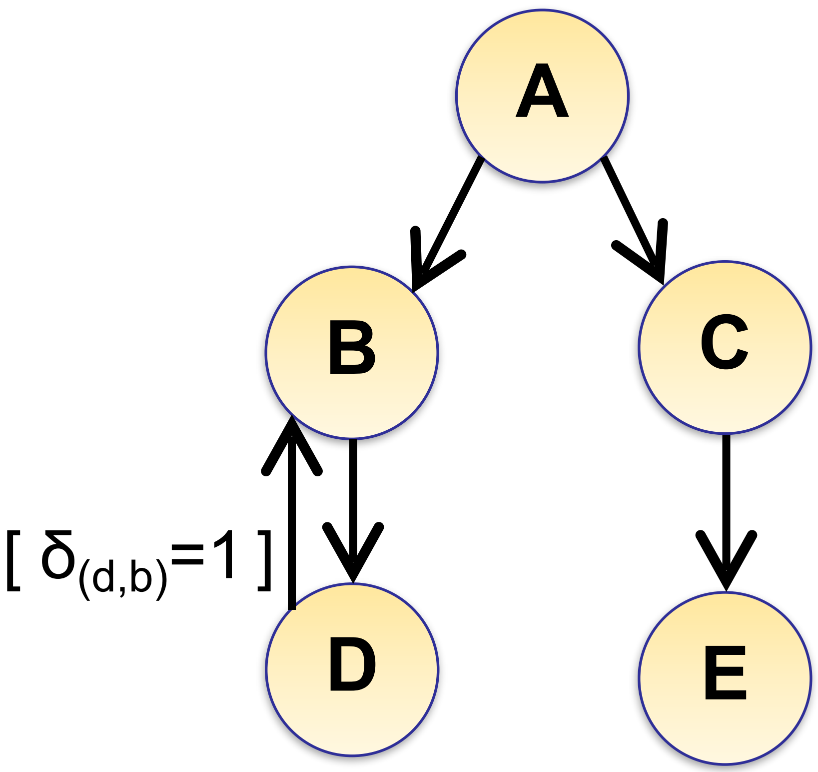
$$start(i,\omega) = start(i,0) + \omega \cdot \lambda$$

$$start(i,0) = \mathbf{S_i + k_i . \lambda}$$

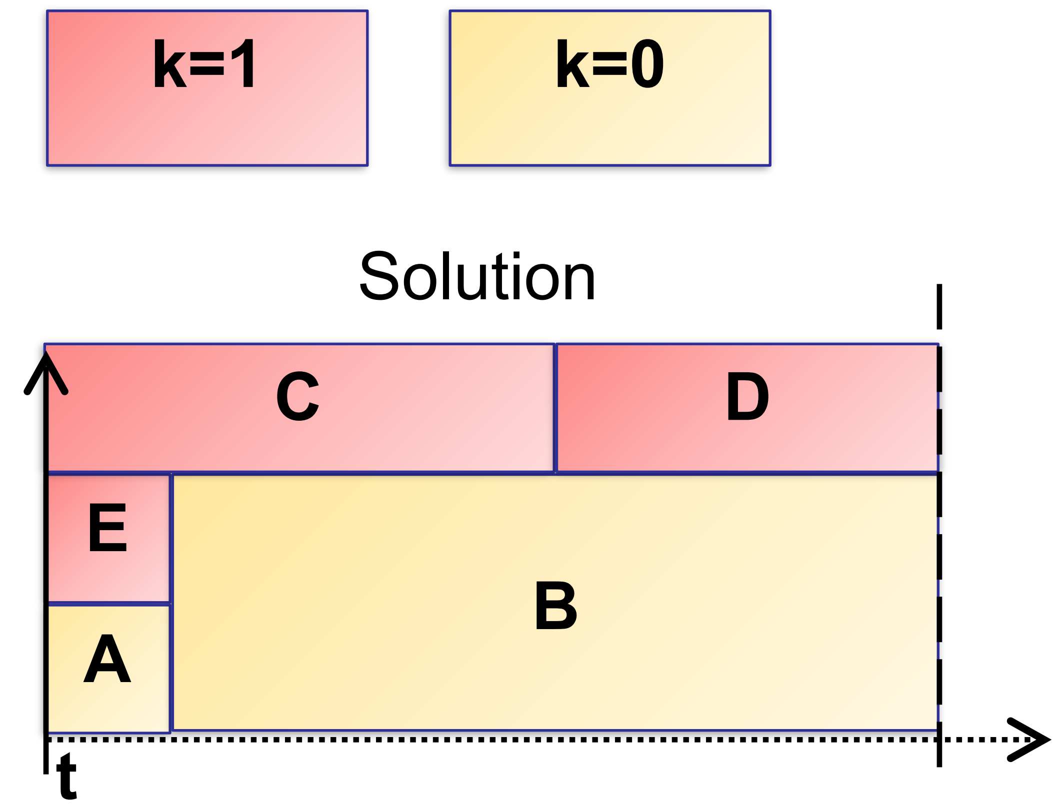
$$start(j,\omega) \geq start(i, \omega - \delta_{(i,j)}) + ex_i + \theta_{(i,j)}$$

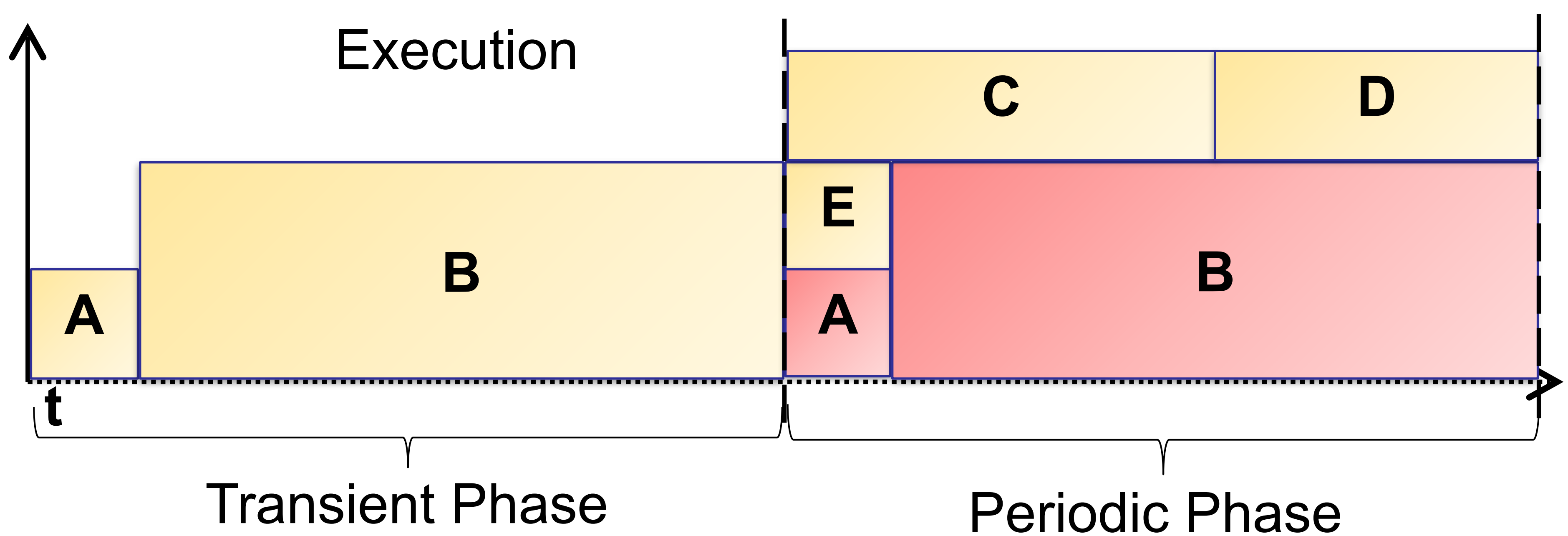
$$\mathbf{S_j + k_j * \lambda \geq S_i + ex_i + \theta_{(i,j)} + (k_i - \delta_{(i,j)}) * \lambda}$$

(Modulus) Problem



Task	Ex	Req
A	*1*	*1*
B	*6*	*2*
C	*4*	*1*
D	*3*	*1*
E	*1*	*1*





Constraint Programming

Constraint Programming is a problem-solving methodology

Solve Hard Combinatorial Problems

Model

•**Variables**

- Finite Domain: set of values that a variable can assume

•**Constraint**:

- Filtering algorithm
- Domain reduction

Constraint Programming

Solving

Consistency

Constraint Propagation: reduction of the domain of the variables to prevent search to find an infeasible solution

Search

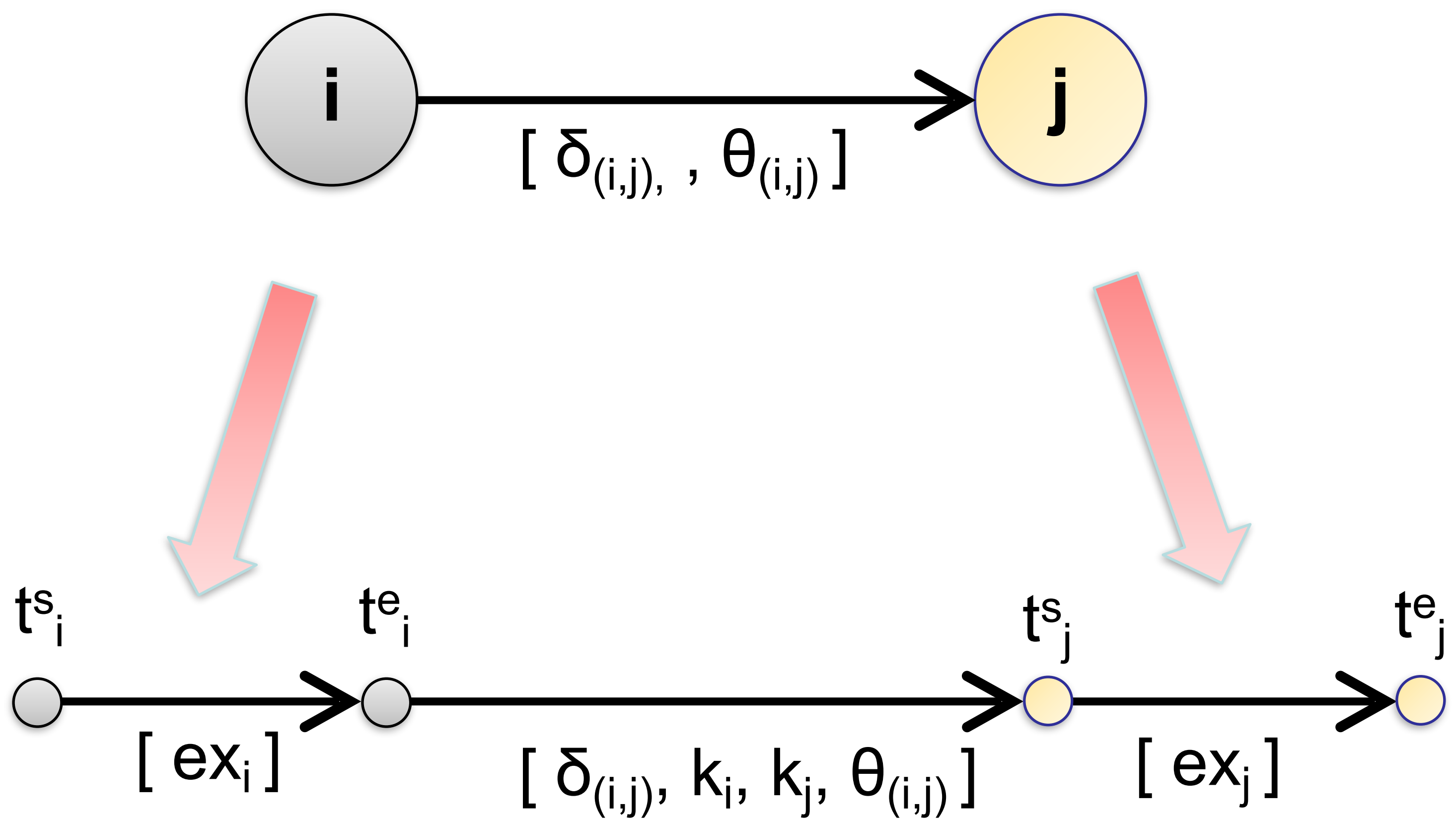
Solve Model

•Define/choose search algorithm

•Define/choose heuristics

Once the problem is modeled using constraints, a wide selection of solution techniques are available

Simple Temporal Network Model



R. **Dechter**. *Temporal constraint networks*. Articial Intelligence, 49:61,95, 1991.

CP Model

Variables

$t^e_i = t^s_i + ex_i$

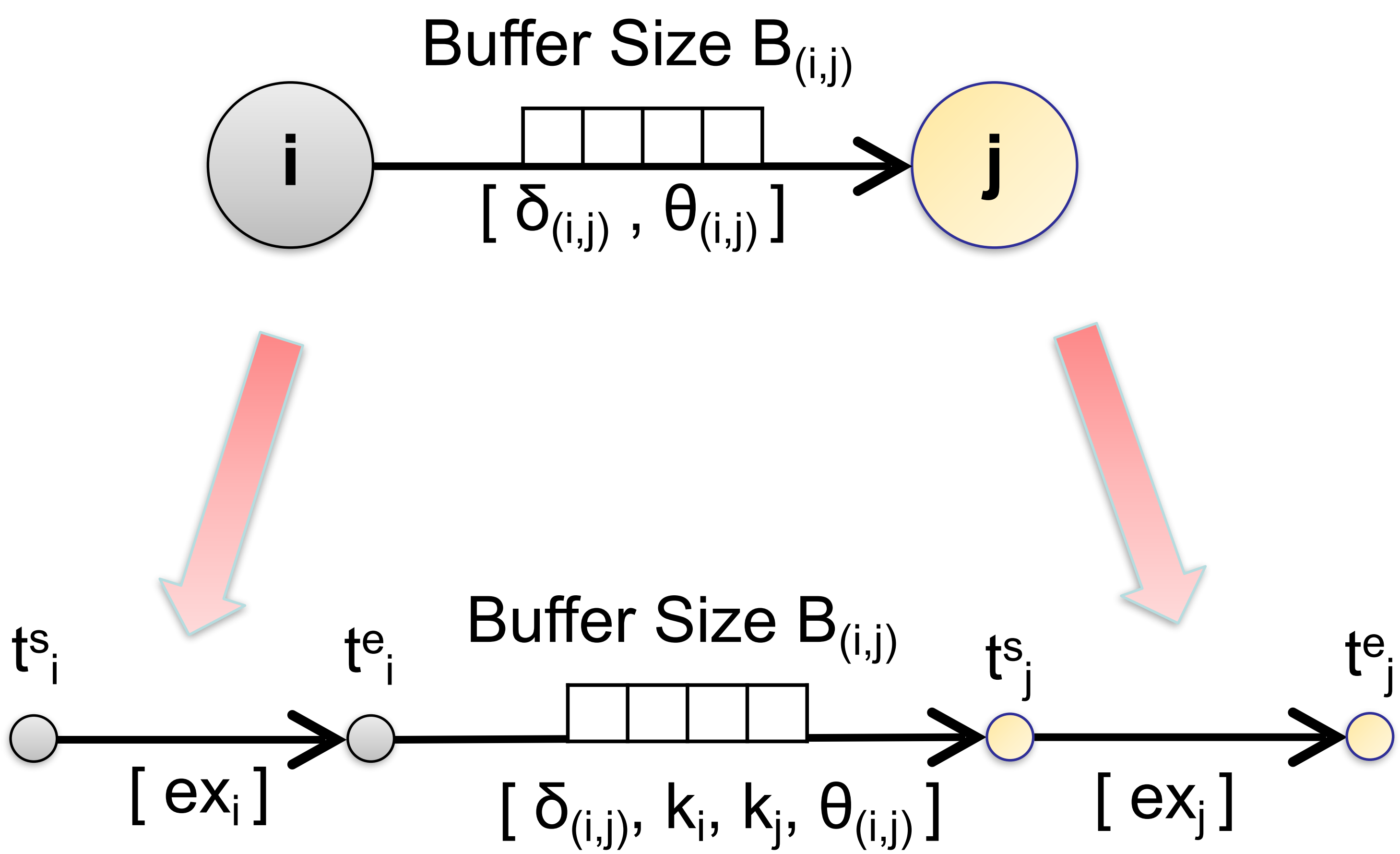
- **Start Times t^s_i** $[0 .. \lambda]$
- **Iteration Values k_i** $[- \infty .. + \infty]$
- **Modulus Variable λ** $[0 .. + \infty]$

Constraints

- **Resource Constraints (including buffers)**
- **Symmetry Breaking Constraints**
- **Temporal Constraints**

Resource (Buffer) Constraints

In real context, the precedence constraint often implies an exchange of intermediate step products between activities that should be stored in buffers.



$$\mathbf{k_j - k_i + (\ t^e_i \leq t^s_j\ ) \leq B_{(i,j)} - \delta_{(i,j)}}$$

Symmetry Breaking Constraints

The assignment of different iteration values *k* to communicating tasks allows one to break precedence relation on the modular time horizon.

Edge (i,j)
$\delta_{(i,j)} = 0$,
$\theta_{(i,j)} = 0$

$$k_j \geq k_i + \left\lceil \frac{t^e{}_i - t^s{}_j}{\lambda} \right\rceil$$

$$k_j \leq \max_{i \in P_j} \left(k_i + \left\lceil \frac{t^e{}_i - t^s{}_j + \theta_{(i,j)}}{\lambda} \right\rceil - \delta_{(i,j)} \right) + 1$$

where P_j is the set of predecessors of j

Modular Precedence Constraint

$$s_j + k_j \cdot \lambda \geq e_i + (k_i - \delta) \cdot \lambda + \theta$$

Filtering on iteration variables ***k***

Maintain a proper distance between the iteration variables

Filtering on start time variables ***s***

Modify the start time to avoid infeasible overlapping of activities

Filtering on the modulus variable ***λ***

Computes a lower bound for the modulus

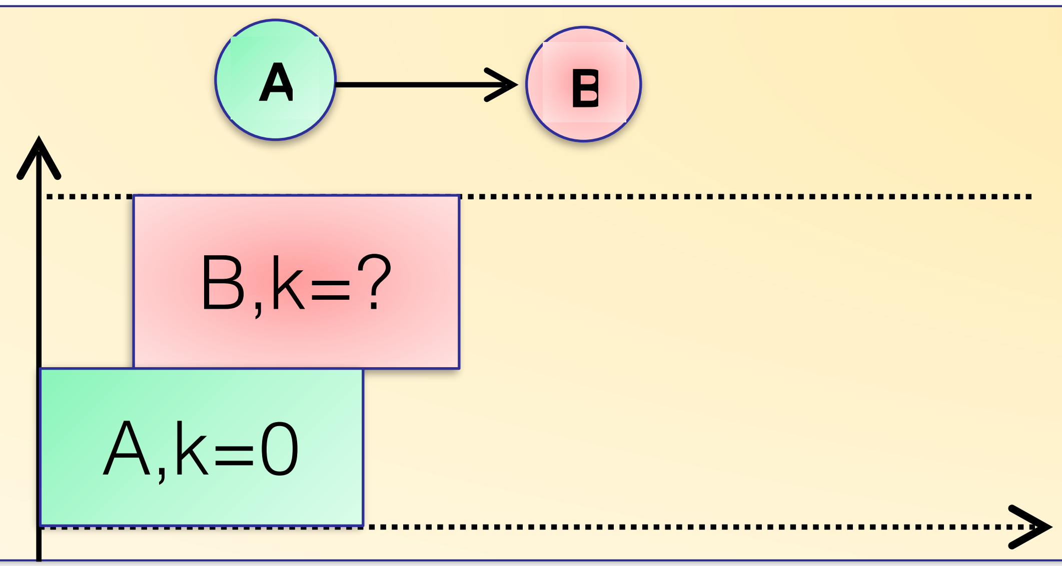
Modular Precedence Constraint

Filtering on iteration variables ***k***

$$k_i \le UB(k_j) + \delta + \left\lfloor \frac{UB(s_j) - LB(e_i) - \theta}{UB(\lambda)} \right\rfloor$$

$$k_j \ge LB(k_i) - \delta - \left\lfloor \frac{UB(s_j) - LB(e_i) - \theta}{UB(\lambda)} \right\rfloor$$

We refer to UB(x), LB(x) as the highest and the lowest values of the domain of the x variable



$$1 \ge \left\lfloor \frac{UB(s_j) - LB(e_i) - \theta}{UB(\lambda)} \right\rfloor \ge -1$$

$$k_j \ge LB(k_i) - \delta - 1$$

$$k_j > k_i - \delta$$

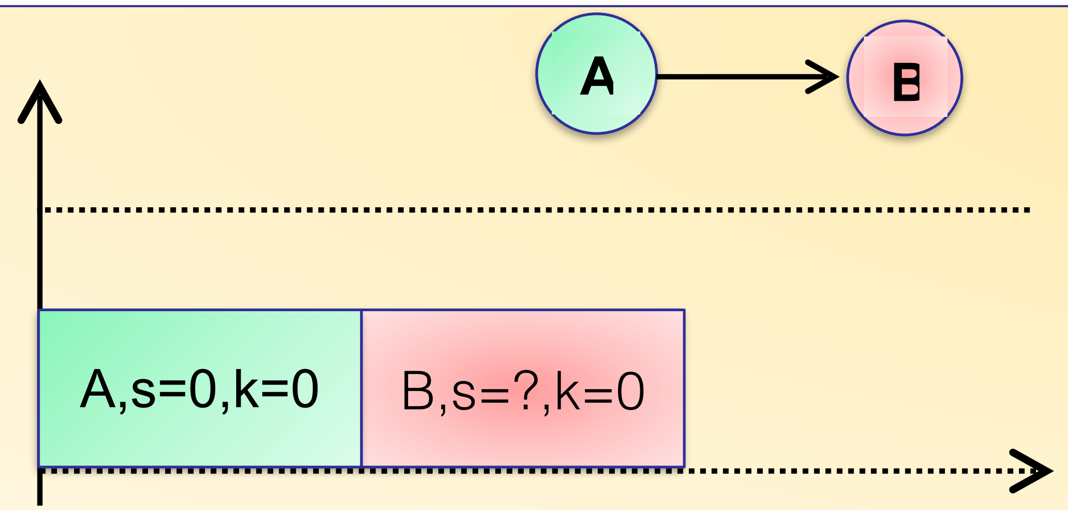
Modular Precedence Constraint

Filtering on start time variables ***s***

$$\Delta_k = k_i - k_j - \delta$$

$$s_j \geq LB(e_i) + \Delta_k \cdot UB(\lambda) + \theta$$

$$e_i \leq UB(s_j) - \Delta_k \cdot UB(\lambda) - \theta$$



$$\Delta_k = 0$$

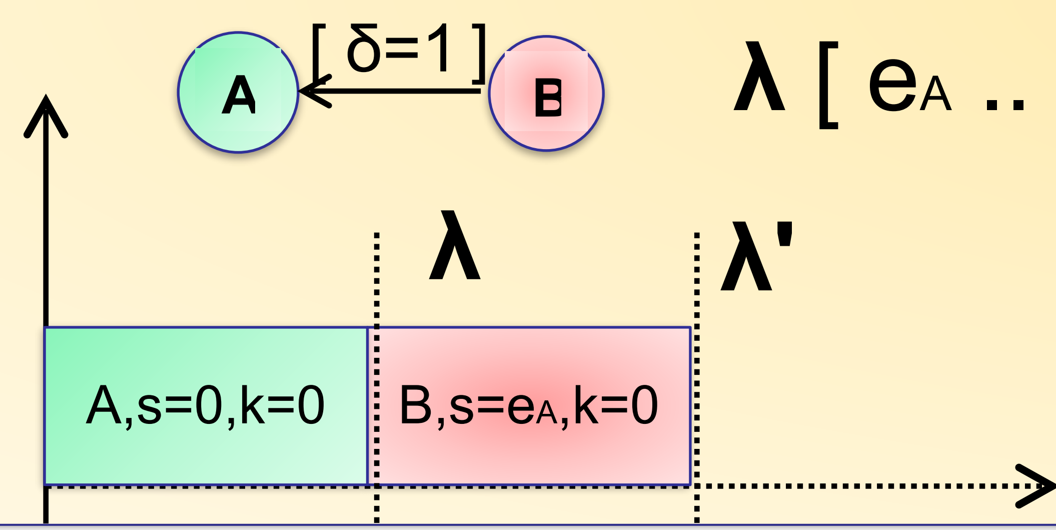
$$s_j \geq LB(e_i) + \theta$$

Modular Precedence Constraint

Filtering on the modulus variable $\boldsymbol{\lambda}$

Condition: $\Delta_k < 0$

$$\lambda \geq \left\lceil \frac{LB(e_i) - UB(s_j) + \theta}{UB(k_j) - LB(k_i) + \delta} \right\rceil$$



$\boldsymbol{\lambda}$ [e_A .. ∞]

$$\lambda \geq \left\lceil \frac{LB(e_B)}{1} \right\rceil$$

$\boldsymbol{\lambda'}$ [e_B .. ∞]

Search

The solver is based on **tree search** adopting a ***schedule or postpone*** approach.

Choose … nd $k_i = 0$

Fixing the start time … omize completeness

Each task can be:

- Unbounded
- Scheduled
- Postponed

The search interleaves the assignment of start times and iteration values.

C. Le **Pape** and P. **Couronné**. *Time-versus-capacity compromises in project scheduling*. In In Proc. of the 13th Workshop of the UK Planning Special Interest Group, 1994.

Experimental Results

ST200 processor (VLIW) instruction scheduling instances [ST-Microelectronics]

From 10 nodes, 42 arcs to 214 nodes and 1063 arcs.

Two sets:
- Industrial
- Modified (more challenging set)

Comparison:
- ILP Optimal Value [Ayala&Artigues]
- SMS (heuristic) Solution [Hagog&Zaks]

			Industrial			Modified		
Instances	nodes	arcs	time(sec)	Gap(%)	SMS(%)	time(sec)	Gap(%)	SMS(%)
adpcm-st231.1	86	405	14400	0%	19.23%	X	X	X
adpcm-st231.2	142	722	582362	2.44/2.44%	0%	X	X	X
gsm-st231.1	30	190	0.05	0%	0%	250	10.7/10.7%	10.7%
gsm-st231.2	101	462	79362	0%	0%	X	X	X
gsm-st231.5	44	192	0.05	0%	13.33%	280	0%	5.26%
gsm-st231.6	30	130	17	0%	31.25%	152	0%	0%
gsm-st231.7	44	192	0.05	0%	41.66%	92	0%	2.38%
gsm-st231.8	14	66	0.05	0%	31.25%	0.27	0%	0%
gsm-st231.9	34	154	0.05	0%	0%	0.56	5.88/0%	8.57%
gsm-st231.10	10	42	0.05	0%	0%	0.1	0%	0%
gsm-st231.11	26	137	0.05	0%	0%	0.37	0%	0%
gsm-st231.12	15	70	0.05	0%	0%	12.65	0%	0%
gsm-st231.13	46	210	1856	0%	0%	985.03	0%	0%
gsm-st231.14	39	176	301.25	0%	17.39%	220	2.94/2.94%	0%
gsm-st231.15	15	70	0.05	0%	28.57%	12.36	0%	8.33%
gsm-st231.16	65	323	7520	0%	0%	X	X	X
gsm-st231.17	38	173	0.05	0%	23.81%	90	0%	0%
gsm-st231.18	214	1063	X	0%	30.76%	X	X	X
gsm-st231.19	19	86	0.05	0%	0%	38.23	0%	6.25%
gsm-st231.20	23	102	0.05	0%	0%	123	3.23/3.23%	4.76%
gsm-st231.21	33	154	0.05	0%	45.45%	42.06	0%	3.24%
gsm-st231.22	31	146	0.05	0%	0%	80.36	0%	0%
gsm-st231.25	60	273	3652	0%	0%	(604800)	0%	1.75%
gsm-st231.29	44	192	12.6	0%	23.81%	210	0%	0%
gsm-st231.30	30	130	12	0%	0%	58	0%	3.84%
gsm-st231.31	44	192	47	0%	41.67%	142	0%	2.5%
gsm-st231.32	32	138	0.05	0%	31.25%	0.25	0	0%
gsm-st231.33	59	266	2365	0%	11.76%	(604800)	0%	0%
gsm-st231.34	10	42	0.05	0%	6.25%	5.05	0%	0%
gsm-st231.35	18	80	0.05	0%	0%	52	0%	0%
gsm-st231.36	31	143	27	0%	14.29%	230	0%	7.69%
gsm-st231.39	26	118	0.05	0%	0%	95	0%	4.55%
gsm-st231.40	21	103	0.05	0%	0%	15	0%	5.56%
gsm-st231.41	60	315	2356	0%	0%	X	X	X
gsm-st231.42	23	102	0.05	0%	0%	12	0%	14.29%
gsm-st231.43	26	115	0.05	0%	21.73%	15	0%	9.1%

Mostafa **Hagog** and Ayal **Zaks**. *Swing modulo scheduling for gcc*, 2004.

M. **Ayala** and C. **Artigues**. *On integer linear programming formulations for the resource-constrained modulo scheduling problem*, 2010. http://hal.archives-ouvertes.fr/docs/00/53/88/21/PDF/ArticuloChristianMaria.pdf

Experimental Results

Solution Quality Tests

1200 synthetic cyclic graphs with 20 to 100 nodes

Average, best and worst gap between the best solution found within a time limit and the ideal lower bound[1].

time(s)	avg(%)	best(%)	worst(%)
1	3.706%	2.28%	5.18%
2	3.68%	2.105%	5.04%
5	3.51%	1.81%	5.015%
10	3.37%	1.538%	4.98%
60	3.14%	1.102%	4.83%
300	2.9%	0.518%	4.73%

The approach converges very quickly close to the ideal optimal value.

The *real* optimal value lies somewhere in-between the two values.

1) The ideal lower bound is the maximum between the intrinsic iteration bound *ib* of the graph and the ratio between the sum of the execution times and the total capacity.

Buffer-size constraint Tests

400 synthetic cyclic graphs with 20 nodes and intrinsic buffer size of 6.

Highlight the efficiency of the buffer and symmetry breaking propagation.

buffesSize	avg(s)	median(s)	gap%
1	1.1423	0.05	4.925%
2	52.1894	0.1	0.052%
3	157.4673	0.31	0%
6	599.9671	1.215	0%
9	791.5552	1.83	0%

Reasonable limits on the buffer size do not compromise solution quality.

Experimental Results

Modulo Vs Unfolding Scheduling

Study the impact of the overlapped schedule (Modulo S.) w.r.t. the blocked and unfolded approaches.

220 synthetic cyclic graphs with 14 to 65 nodes divided into three classes:

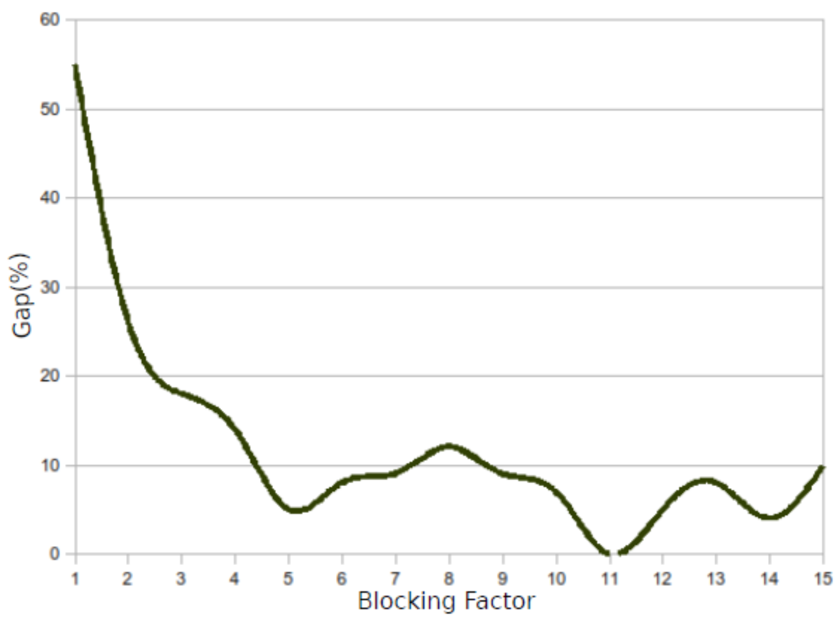
- Small: featuring 14 to 24 nodes
- Medium-size: 25 to 44
- Big: 45-65

The worst gap is relative to the blocked schedule, while the unfolded ones tend to have an oscillatory behavior.

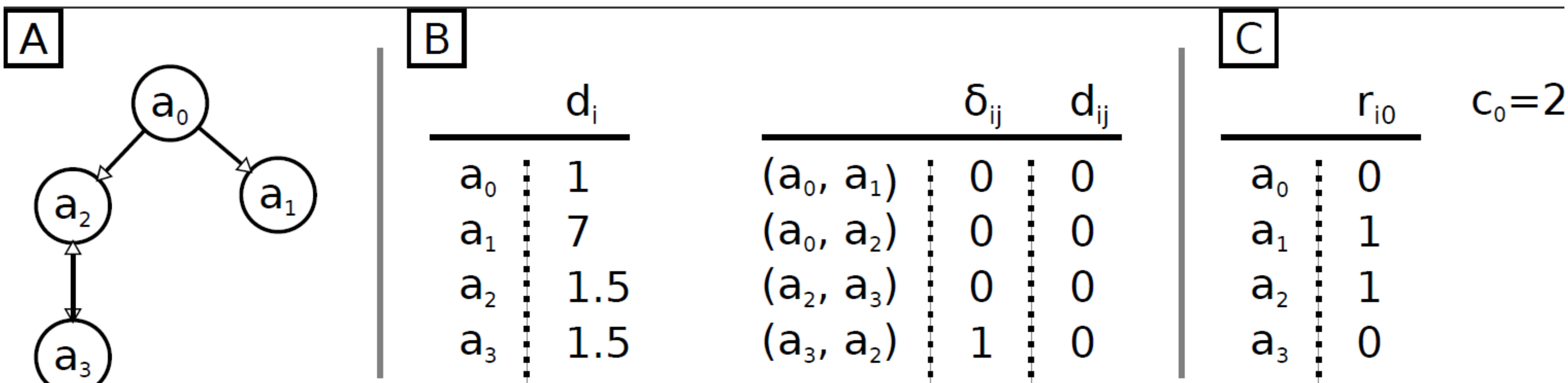
Eight different solver configurations

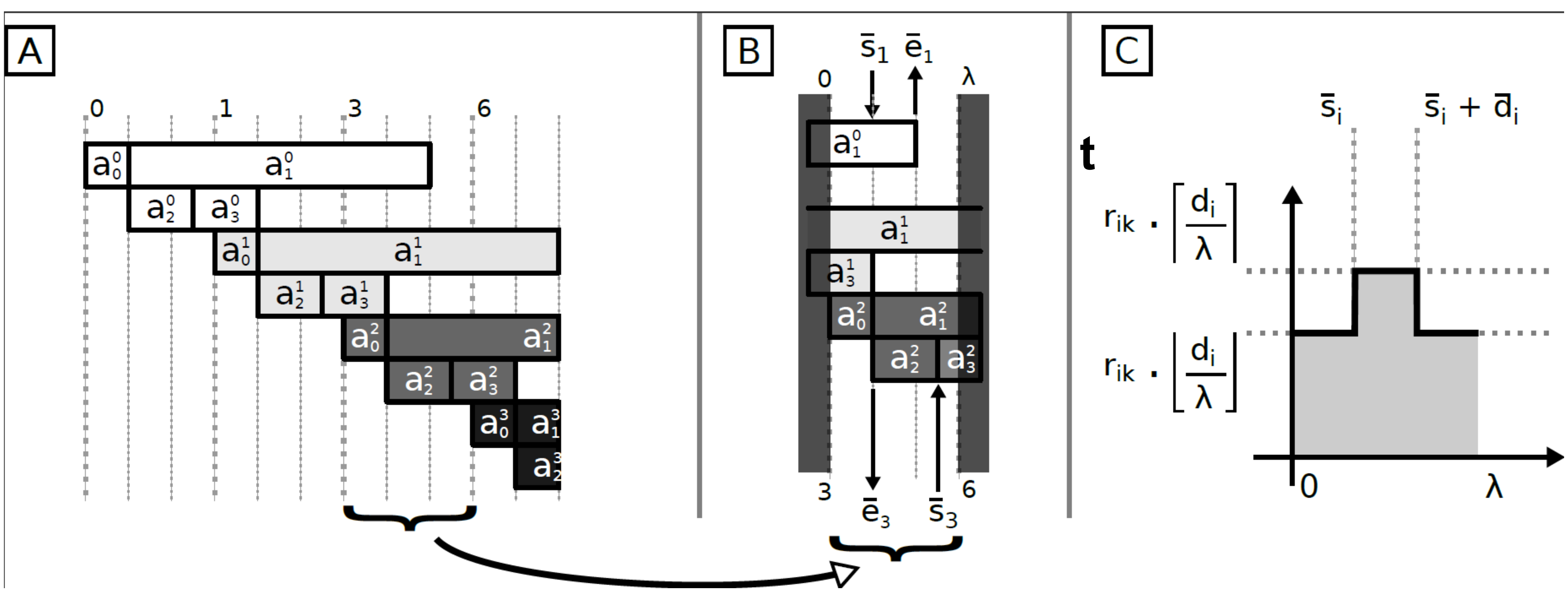
Gap between the solver solution and the Modulo one

	Solution Gap (%)			
Solver	[14-20]	[25-40]	[45-65]	AVG
Blocked	108.16%	65.45%	38.83%	**55.32%**
Unfold2	55.92%	26.06%	19.89%	**26.23%**
Unfold3	33.31%	16.15%	9.99%	**18.6%**
Unfold4	29.41%	14.27%	6.278	**14.13%**
Unfold5	21.35%	5.33%	8.76%	**5.67%**
Unfold6	39.06%	8.67%	4.39%	**8.67%**
Unfold8	78.31%	10.71%	7.65%	**12.44%**
Unfold10	16.95%	10.21%	10.03%	**8.65%**

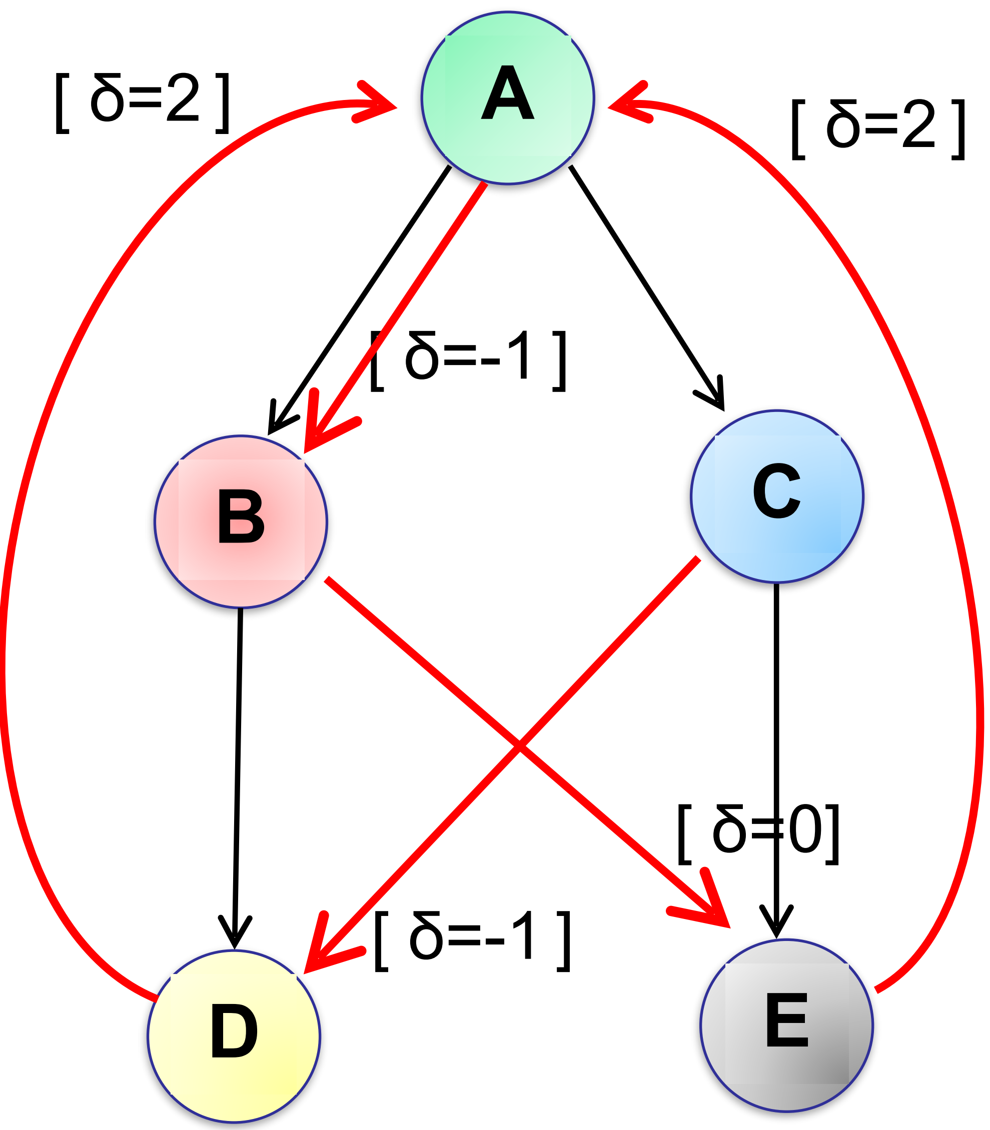


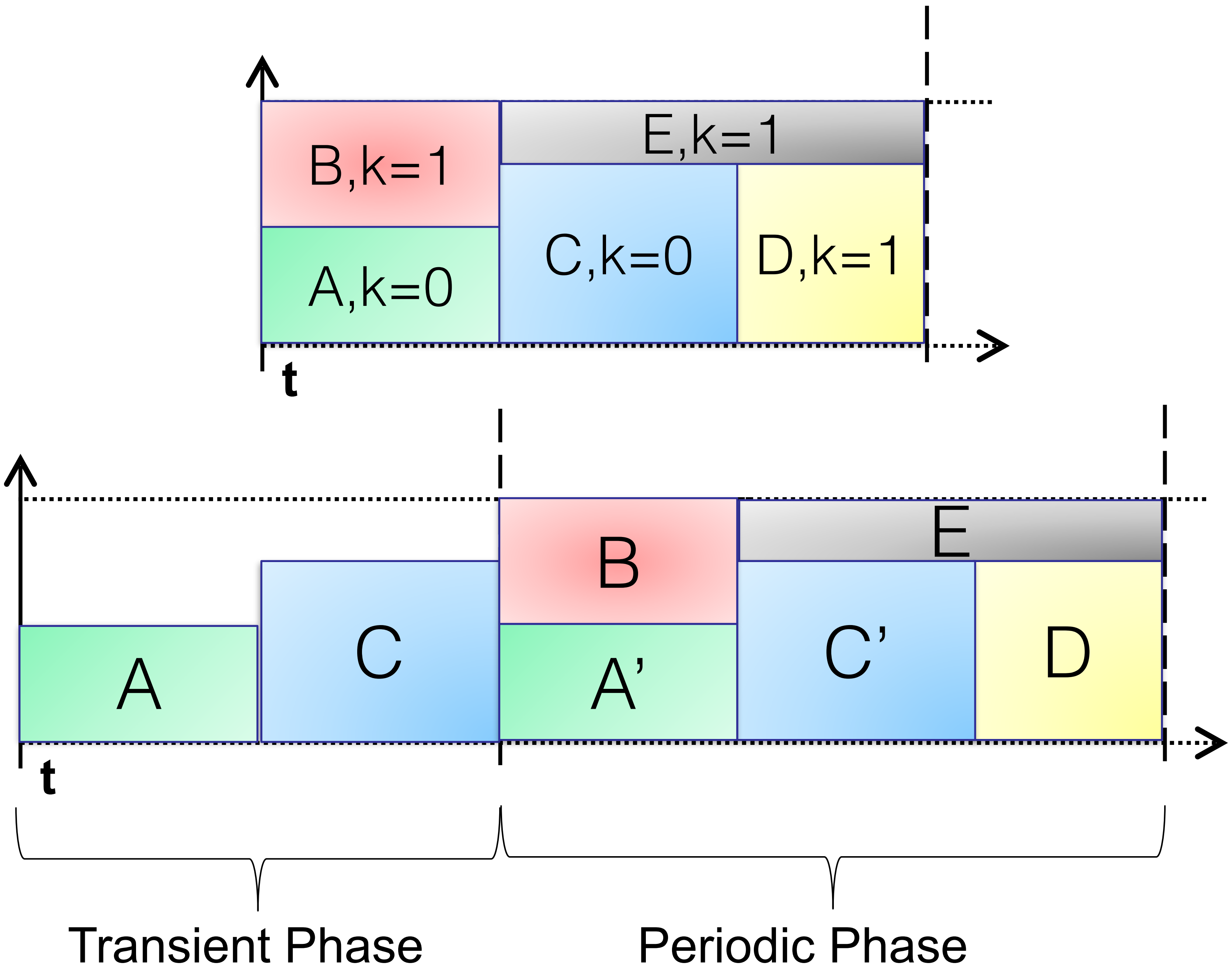
Current and Future Work





Scheduling Representation





Questions ?