Design-Space Exploration with Multi-Objective Resource-Aware Modulo Scheduling

Julian Oppermann[1], Patrick Sittel[2], Martin Kumm[3],
Melanie Reuter-Oppermann[4], Andreas Koch[1], Oliver Sinnen[5]

1: TECHNISCHE UNIVERSITÄT DARMSTADT

3: Hochschule Fulda University of Applied Sciences

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

2: Imperial College London

4: KIT Karlsruhe Institute of Technology

Agenda

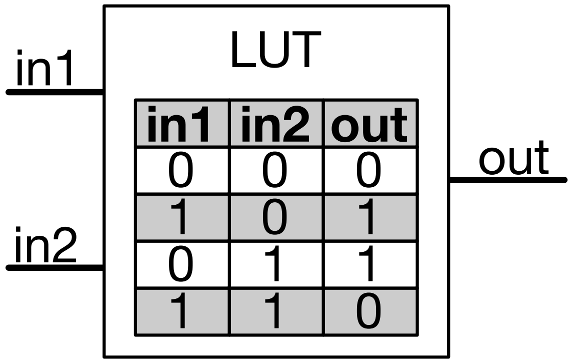
- ▶ **Introduction to high-level synthesis**
- □ Resource-aware modulo scheduling
- □ Exploration of trade-off solutions
- □ Experimental evaluation

FPGAs

- Field-Programmable Gate Arrays
  - semiconductor chips, ideal for energy-efficient, application-specific accelerators

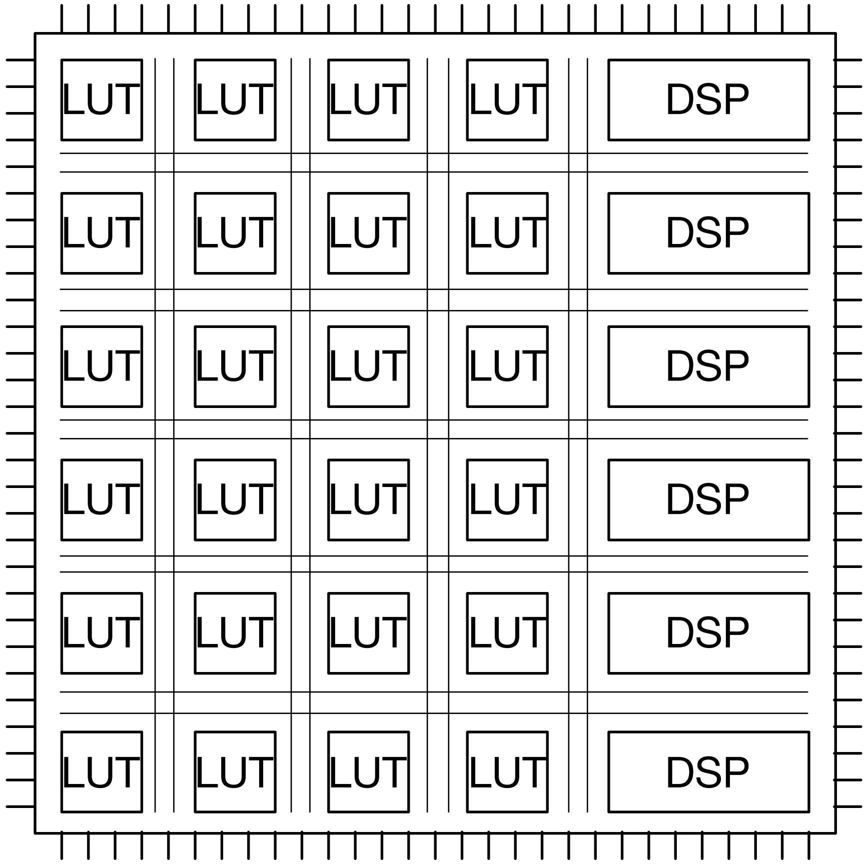
FPGAs

- Field-Programmable Gate Arrays
  - semiconductor chips, ideal for energy-efficient, application-specific accelerators
  - comprised of **resources**
    - look-up tables (LUT)
    - DSP blocks
    - …
    - programmable interconnect



in1	in2	out
0	0	0
1	0	1
0	1	1
1	1	0

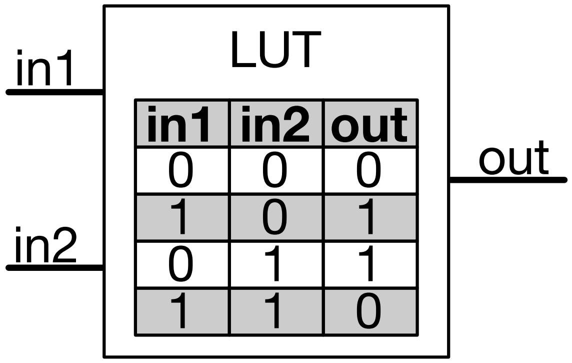
out = in1 xor in2



LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP

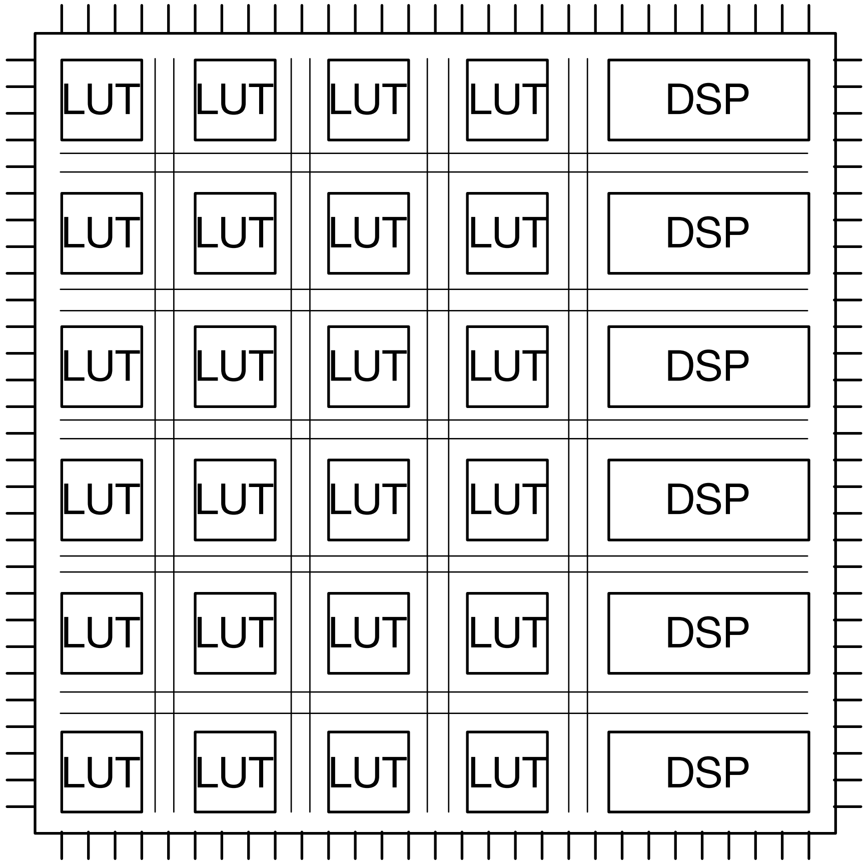
FPGAs

- Field-Programmable Gate Arrays
  - semiconductor chips, ideal for energy-efficient, application-specific accelerators
  - comprised of **resources**
    - look-up tables (LUT)
    - DSP blocks
    - …
    - programmable interconnect
  - „programming“ FPGA
    = configuring and connecting resources



in1	in2	out
0	0	0
1	0	1
0	1	1
1	1	0

out = in1 xor in2



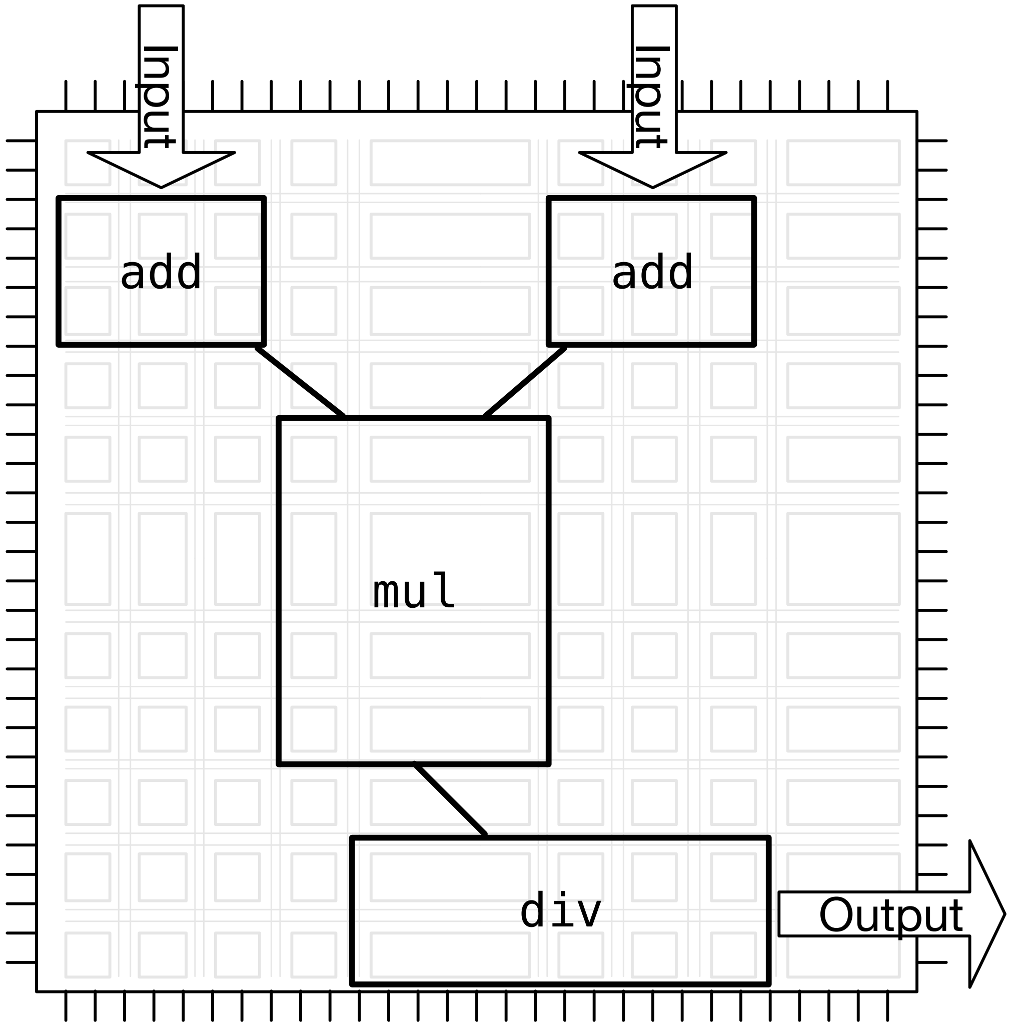
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP
LUT	LUT	LUT	LUT	DSP

Computing with FPGAs

- Spatial computation is common

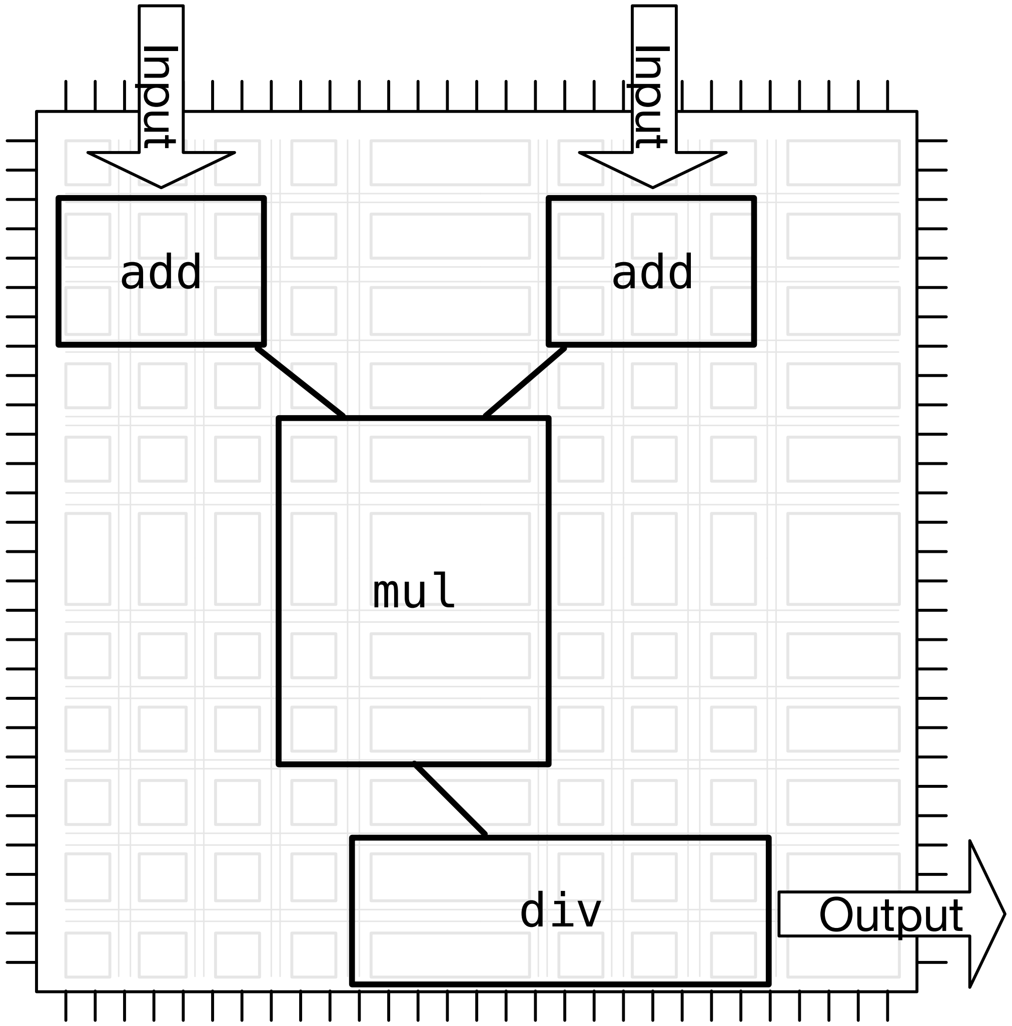
Computing with FPGAs

- Spatial computation is common
- Instantiate **operators** from a library
  - requires certain amount of FPGA resources
- Connect operators to form a datapath



Computing with FPGAs

- Spatial computation is common
- Instantiate **operators** from a library
  - requires certain amount of FPGA resources
- Connect operators to form a datapath
- Manual design in hardware description language is tedious



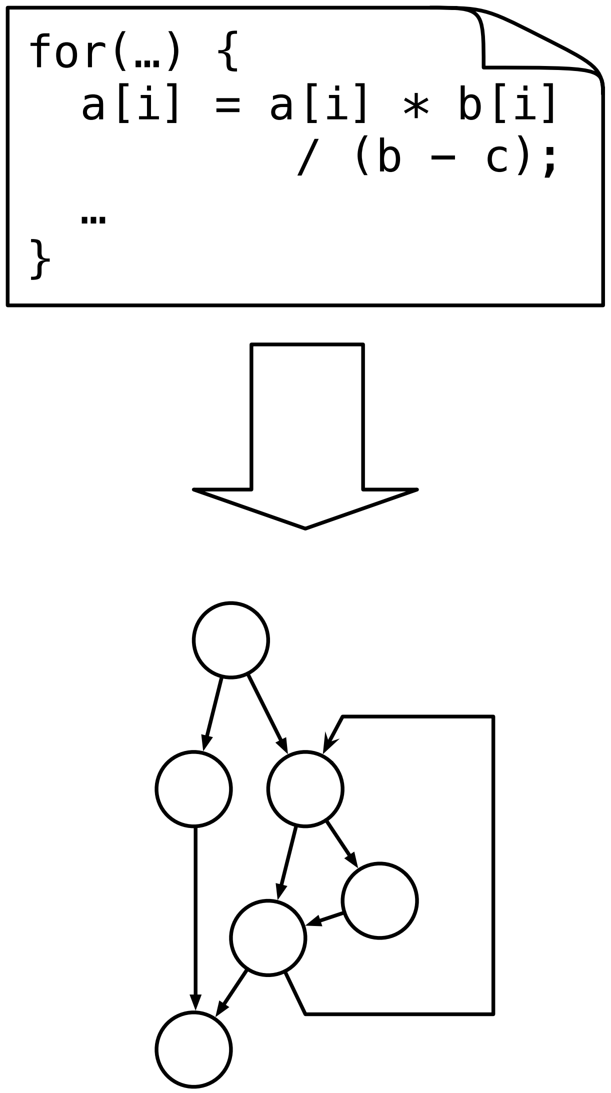
High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)

High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)

- Each loop is transformed to data-flow graph of **operations**
  - operations need a hardware operator to execute



```
for(…) {
  a[i] = a[i] * b[i]
           / (b - c);
  …
}
```

High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)

- Each loop is transformed to data-flow graph of **operations**
  - operations need a hardware operator to execute

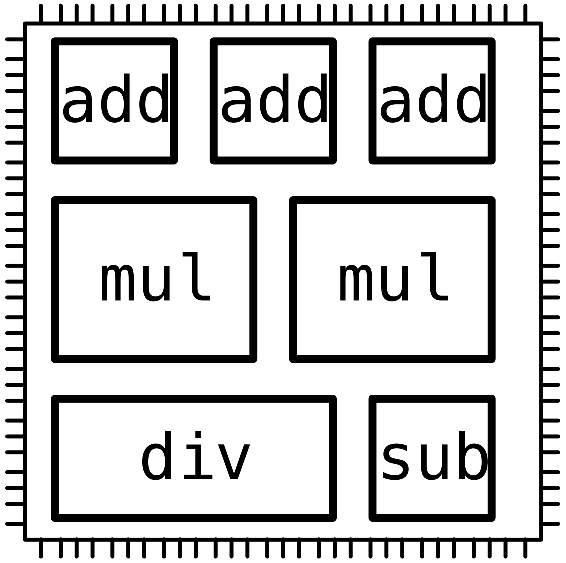
- Algorithmic steps

High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)

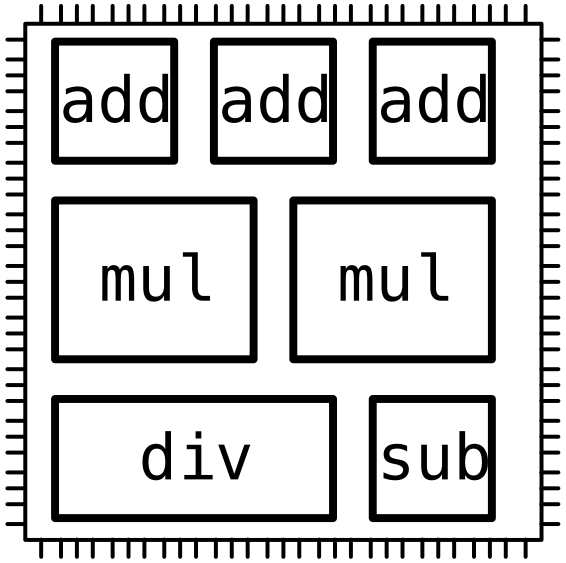
- Each loop is transformed to data-flow graph of **operations**
  - operations need a hardware operator to execute

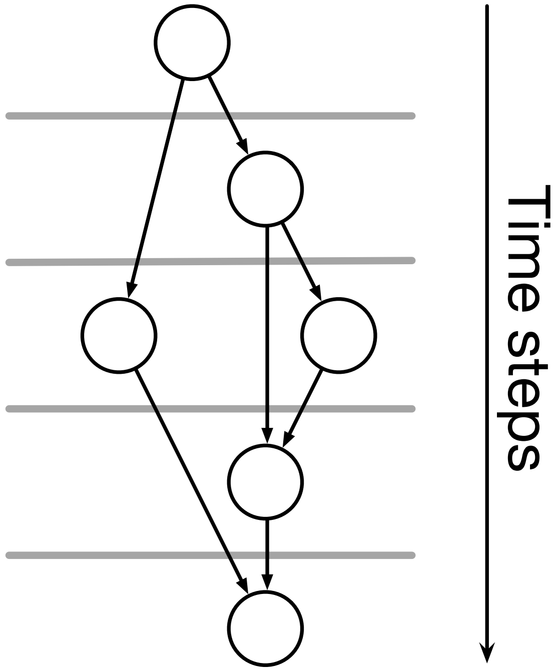
- Algorithmic steps
  - Allocation — *how many operators?*



High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)
- Each loop is transformed to data-flow graph of **operations**
  - operations need a hardware operator to execute
- Algorithmic steps
  - Allocation — *how many operators?*
  - Scheduling — *when is an operation executed?*



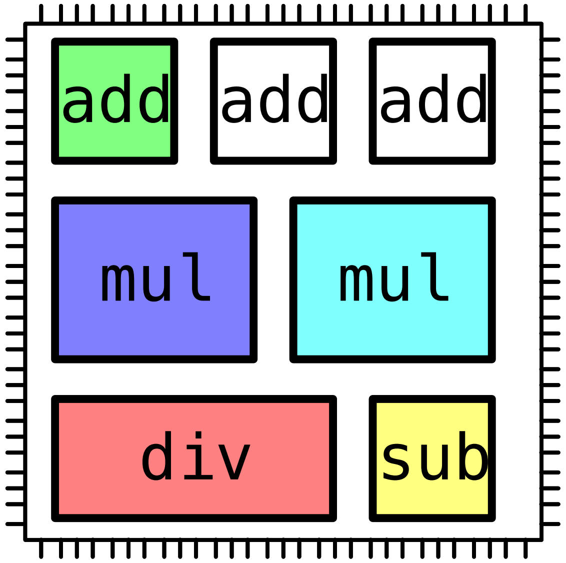


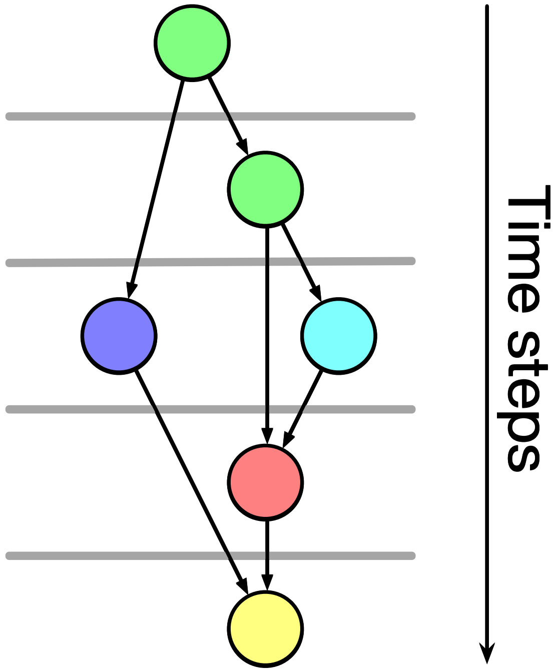
High-Level Synthesis

- HLS = Automatic design from behavioural description (*think: C code*)

- Each loop is transformed to data-flow graph of **operations**
  - operations need a hardware operator to execute

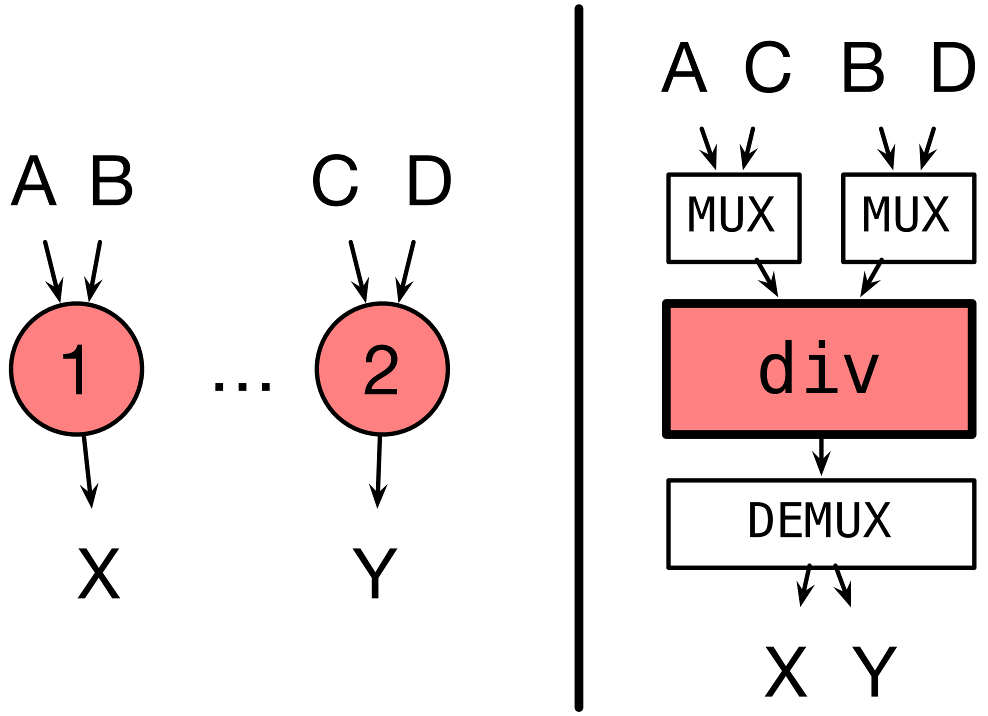
- Algorithmic steps
  - Allocation — *how many operators?*
  - Scheduling — *when is an operation executed?*
  - Binding — on which operator?





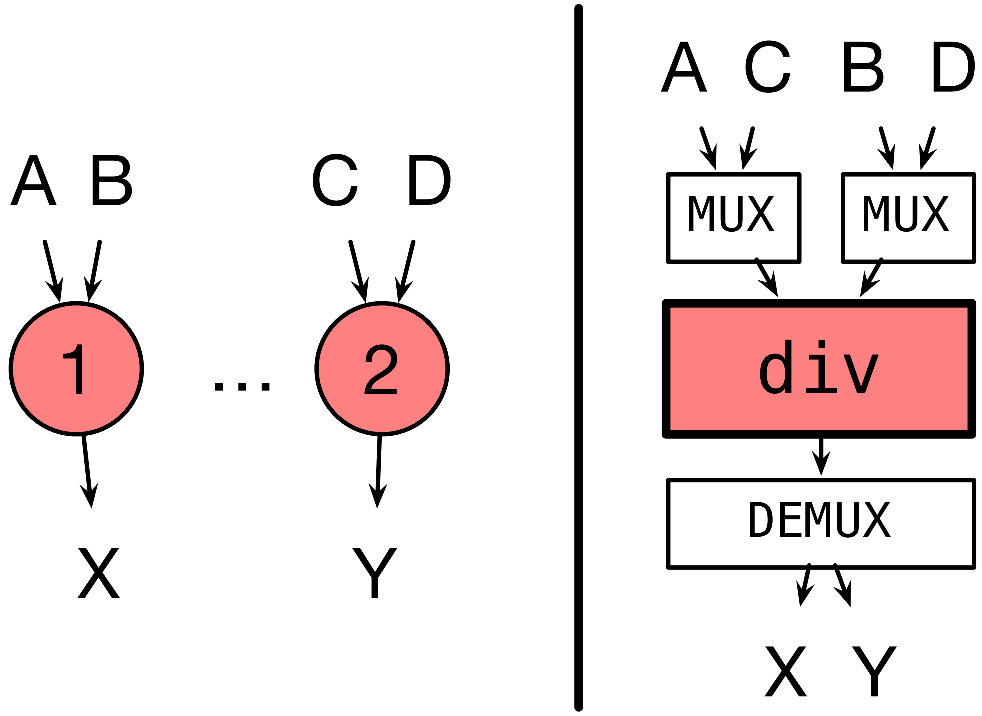
Key HLS Techniques

- **Operator sharing** = *reduce* resource demand by multiplexing



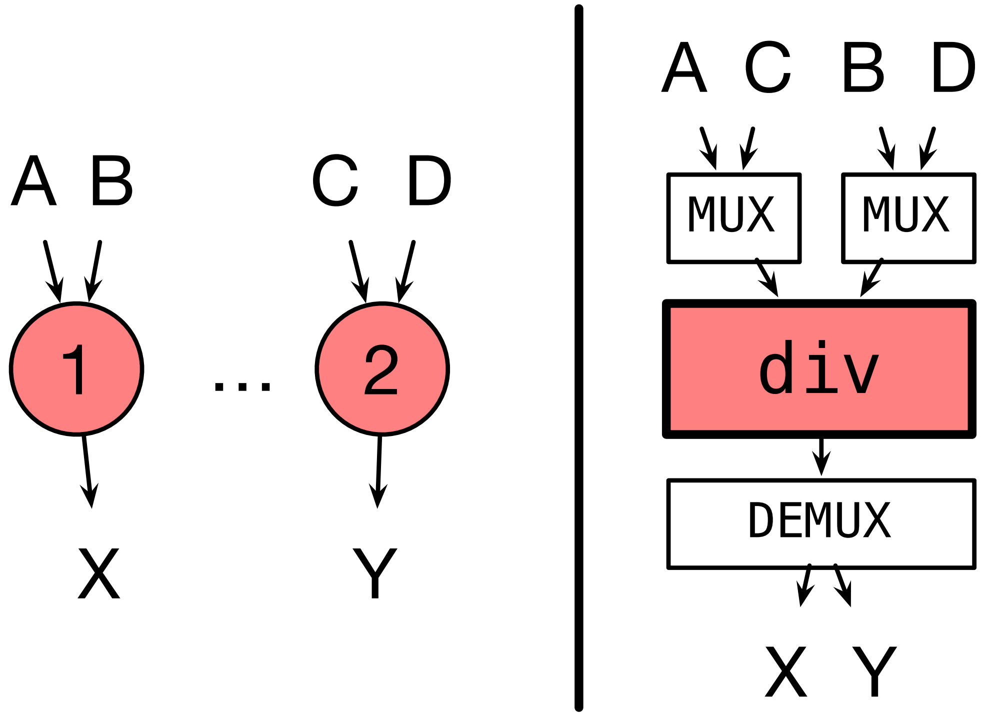
Key HLS Techniques

- **Operator sharing** = *reduce* resource demand by multiplexing
  - only if: cost of operator ≫ cost multiplexers

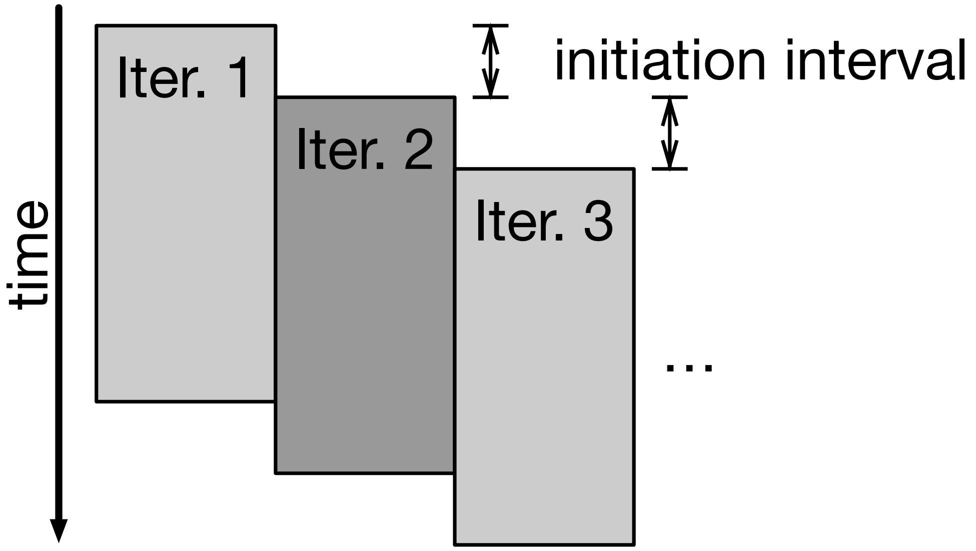


Key HLS Techniques

- **Operator sharing** = *reduce* resource demand by multiplexing
  - only if: cost of operator ≫ cost multiplexers

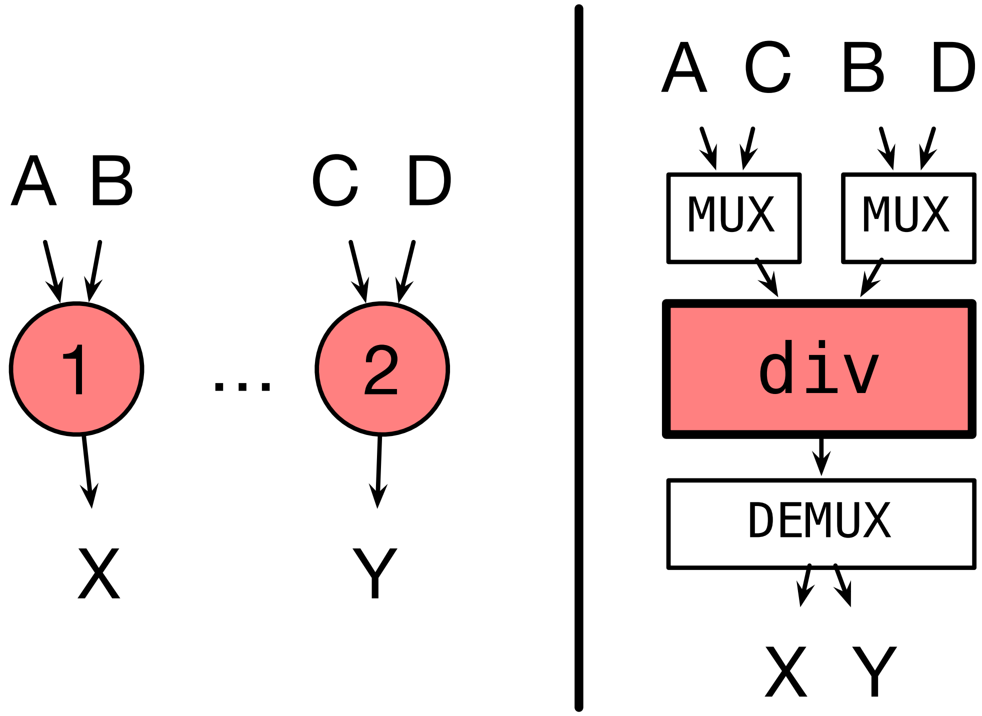


- **Loop pipelining** = *reduce* execution time by overlapping iterations



Key HLS Techniques

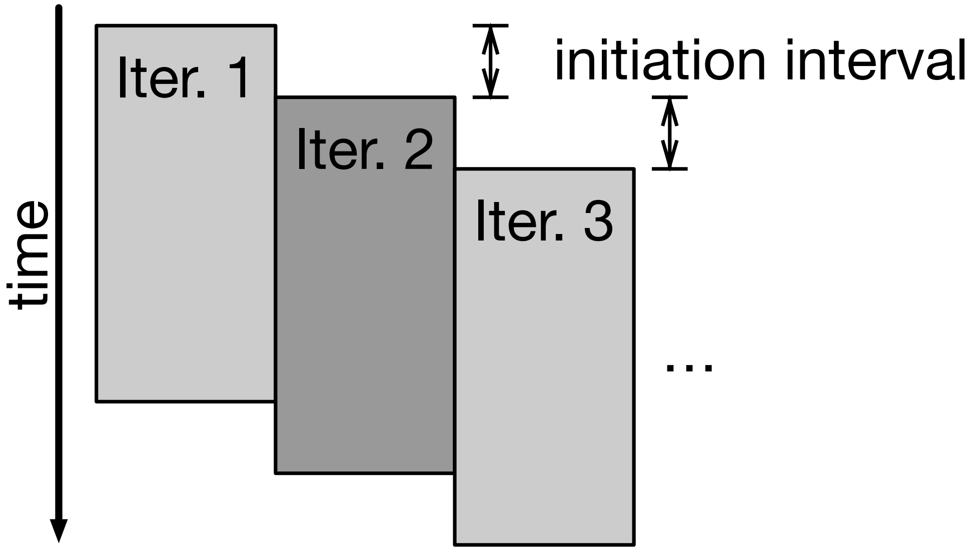
- **Operator sharing** = *reduce* resource demand by multiplexing
  - only if: cost of operator
    ≫ cost multiplexers



- **Loop pipelining** = *reduce* execution time by overlapping iterations
  - smaller **initiation interval** (II)

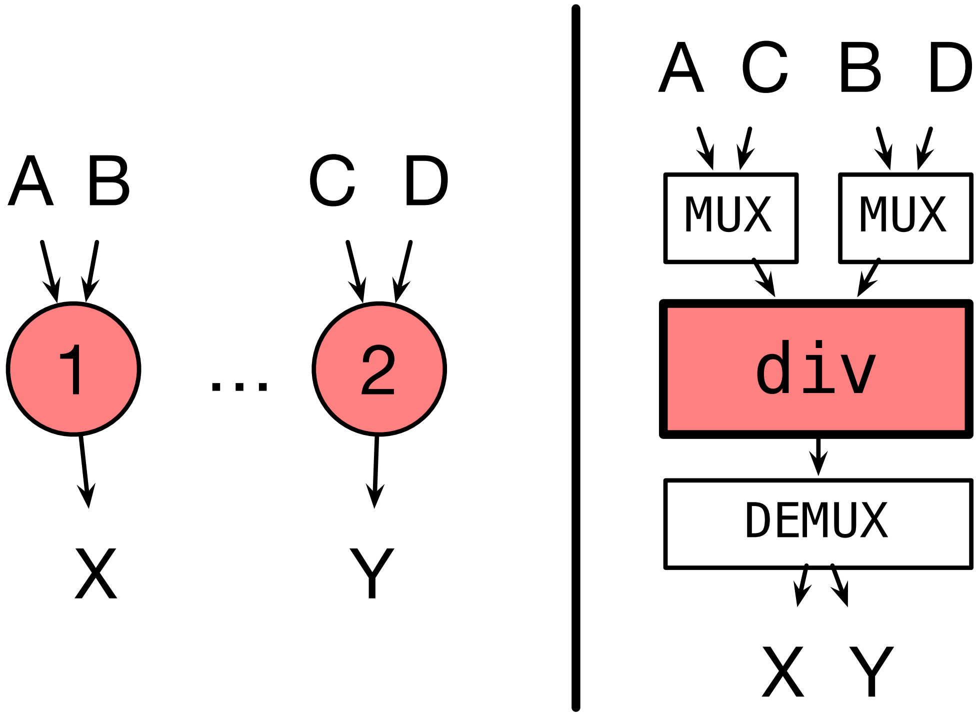
    ⇔

    shorter execution times, more overlap, but less sharing

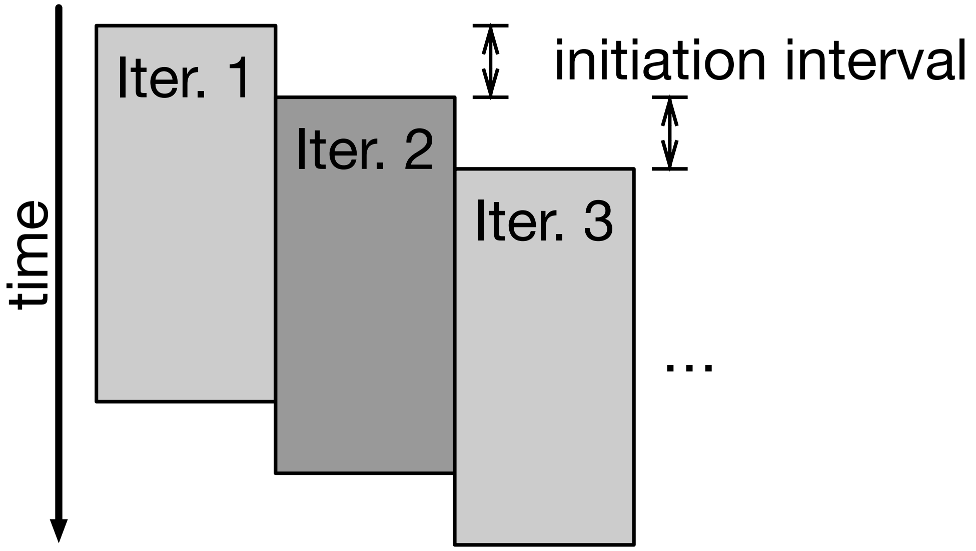


Key HLS Techniques

- **Operator sharing** = *reduce* resource demand by multiplexing
  - only if: cost of operator ≫ cost multiplexers



- **Loop pipelining** = *reduce* execution time by overlapping iterations
  - smaller **initiation interval** (II) ⇔ shorter execution times, more overlap, but less sharing
  - enabled by **modulo schedulers**



Motivation

- HLS can construct many different microarchitectures from the same input specification
  - **Conflicting** objectives

$$\begin{array}{ll} \textit{min} & \textit{initiation interval} \\ \textit{min} & \textit{resource utilisation} \end{array}$$

Motivation

- HLS can construct many different microarchitectures from the same input specification
  - **Conflicting** objectives

    min initiation interval
    min resource utilisation

- Resulting design space needs to be explored manually, or by external tools…

Motivation

- HLS can construct many different microarchitectures from the same input specification
  - **Conflicting** objectives

$$\textit{min} \quad \textit{initiation interval}$$
$$\textit{min} \quad \textit{resource utilisation}$$

- Resulting design space needs to be explored manually, or by external tools…
  - *why not directly as part of the HLS algorithms?*

Contributions

ILP = Integer Linear Program

1. A **framework** for
   Resource-Aware Modulo Scheduling (RAMS)
   - extends existing ILP-based formulations to **combine** allocation and modulo scheduling

Contributions

ILP = Integer Linear Program

1. A **framework** for
   Resource-Aware Modulo Scheduling (RAMS)
   - extends existing ILP-based formulations to **combine** allocation and modulo scheduling

Binding is guaranteed to exist for typical HLS operators

Contributions

ILP = Integer Linear Program

1. A **framework** for Resource-Aware Modulo Scheduling (RAMS)
   - extends existing ILP-based formulations to **combine** allocation and modulo scheduling

Binding is guaranteed to exist for typical HLS operators

2. We investigate ways to efficiently compute different trade-off solutions

Agenda

□ Introduction to high-level synthesis

▶ **Resource-aware modulo scheduling**

□ Exploration of trade-off solutions

□ Experimental evaluation

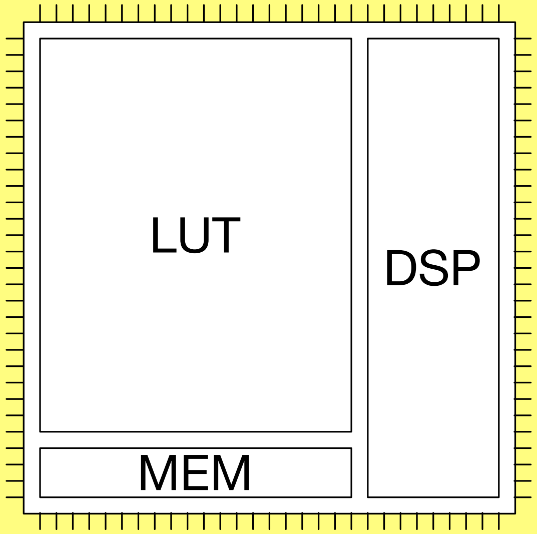
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

R



each $r \in R$ provides N_r elements

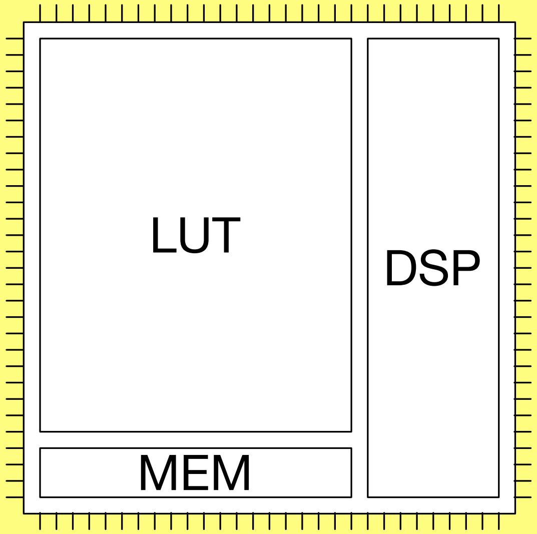
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

R

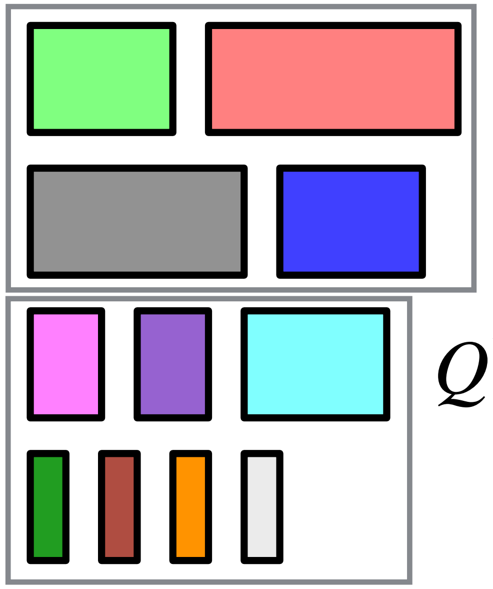


each $r \in R$ provides

N_r elements

Operator types

Q



$\hat{Q}$ „shared“

$Q \backslash \hat{Q}$ „unlimited“

each $q \in Q$ has

a latency l_q, and

resource demands $n_{q,r}$

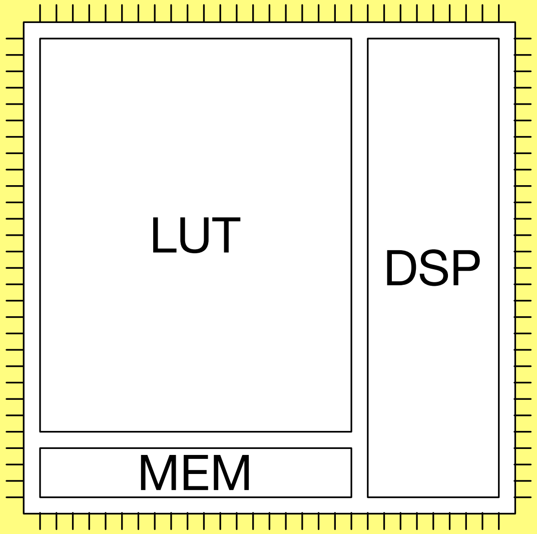
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

R

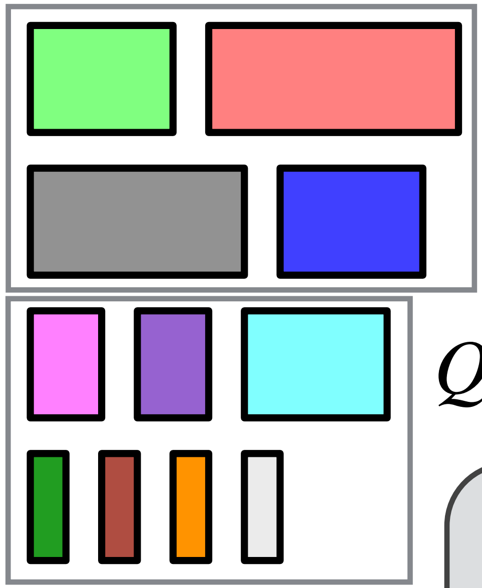


each $r \in R$ provides

N_r elements

Operator types

Q



$\hat{Q}$ „shared“

$Q \backslash \hat{Q}$ „unlimited“

…too „cheap“
to share

each $q \in Q$ has

a latency l_q, and

resource demands $n_{q,r}$

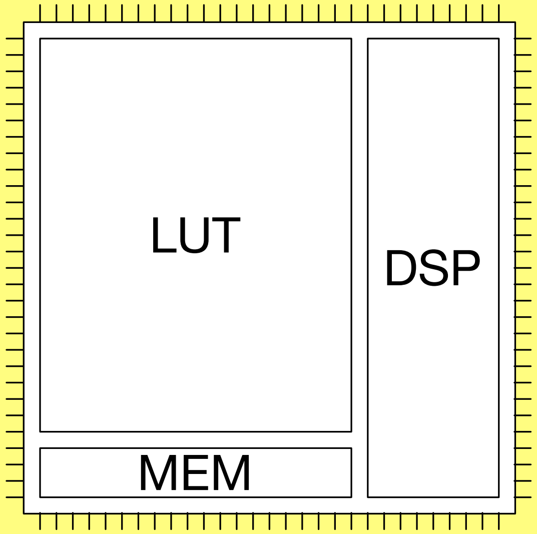
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

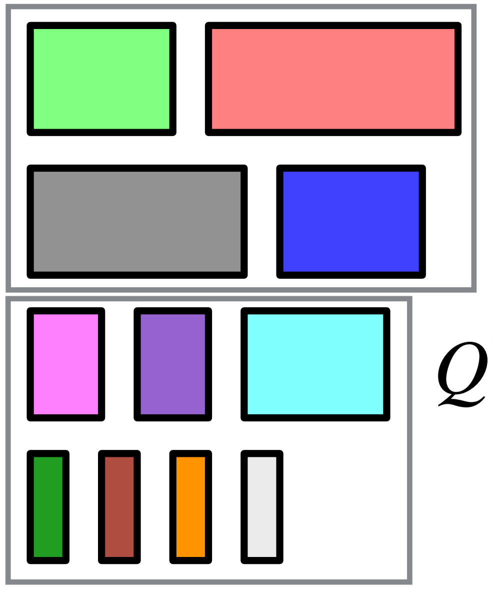
R



each $r \in R$ provides N_r elements

Operator types

Q



$\hat{Q}$ „shared“

$Q \backslash \hat{Q}$ „unlimited“

each $q \in Q$ has

a latency l_q, and

resource demands $n_{q,r}$

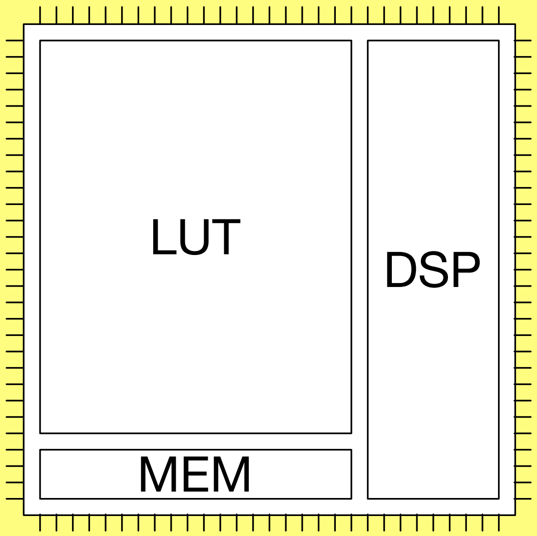
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

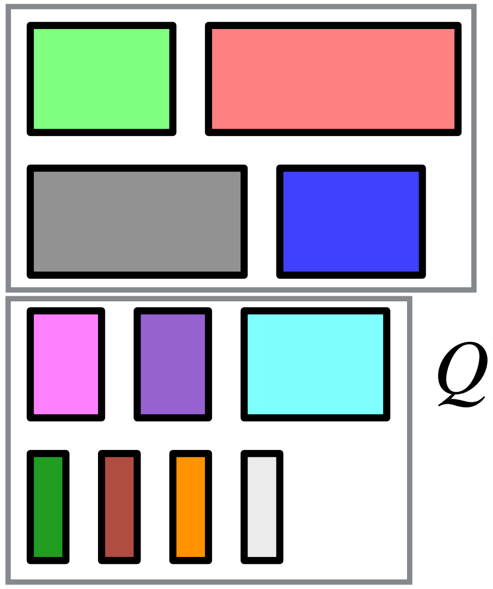
R



each $r \in R$ provides N_r elements

Operator types

Q



$\hat{Q}$ „shared“

$Q \backslash \hat{Q}$ „unlimited“

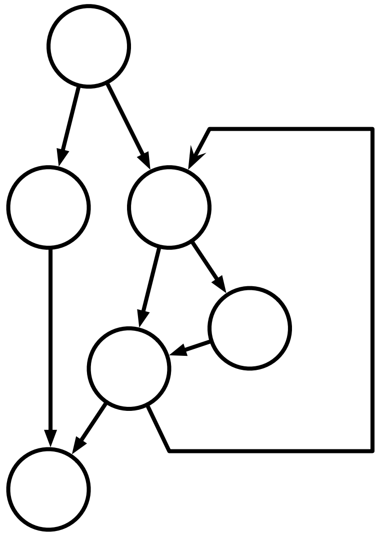
each $q \in Q$ has

a latency l_q, and

resource demands $n_{q,r}$

Dependence graph

$G = (O, E)$



subsets O_q group operations using operator type q

each $i \rightarrow j \in E$ has a distance β_{ij}

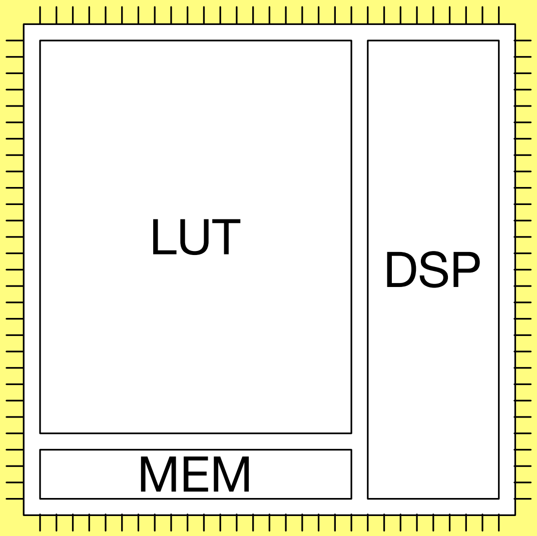
Problem Signature

🟨 = resource-aware extension

Input:

Resource types

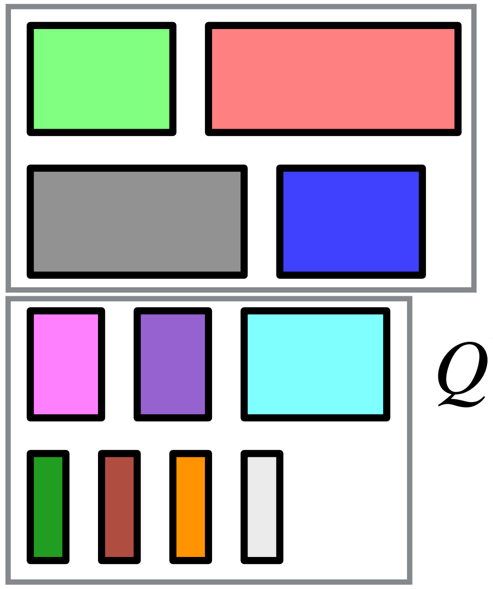
R



each $r \in R$ provides N_r elements

Operator types

Q



$\hat{Q}$ „shared“

$Q \setminus \hat{Q}$ „unlimited“

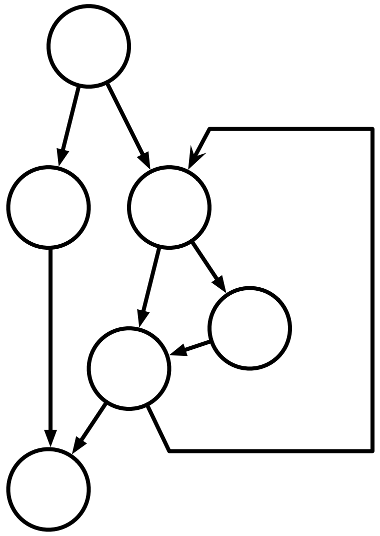
each $q \in Q$ has

a latency l_q, and

resource demands $n_{q,r}$

Dependence graph

$G = (O, E)$



subsets O_q group operations using operator type q

each $i \to j \in E$ has a distance β_{ij}

Output: A solution S, comprised of:

II^S initiation interval

t_i^S start times for each operation („schedule“)

a_q^S number of instances for each operator type („allocation“, A^S)

Feasibility

🟨 = resource-aware extension

- A solution S is **feasible** if and only if

Feasibility

🟨 = resource-aware extension

- A solution S is **feasible** if and only if

1. *„all precedence constraints are satisfied“*

$$t_i^S + l_q \leq t_j^S + \beta_{ij} \cdot II^S \quad \forall i \to j \in E \land i \in O_q$$

Feasibility

🟨 = resource-aware extension

- A solution S is **feasible** if and only if

1. *„all precedence constraints are satisfied“*

$$t_i^S + l_q \;\leq\; t_j^S + \beta_{ij} \cdot II^S \quad \forall i \to j \in E \land i \in O_q$$

2. *„no more than the allocated number of operators are used at any time“*

$$\left|\left\{ i \in O_q : t_i^S \bmod II^S = m \right\}\right| \;\leq\; a_q^S \quad \forall q \in Q \land m \in [0,\, II^S - 1]$$

Feasibility

🟨 = resource-aware extension

- A solution S is **feasible** if and only if

1. *„all precedence constraints are satisfied“*

$$t_i^S + l_q \;\leq\; t_j^S + \beta_{ij} \cdot II^S \quad \forall i \rightarrow j \in E \wedge i \in O_q$$

2. *„no more than the allocated number of operators are used at any time“*

$$\left| \left\{ i \in O_q : t_i^S \bmod II^S = m \right\} \right| \;\leq\; a_q^S \quad \forall q \in Q \wedge m \in [0,\, II^S - 1]$$

3. *„the resource demand is within the device capacity“*

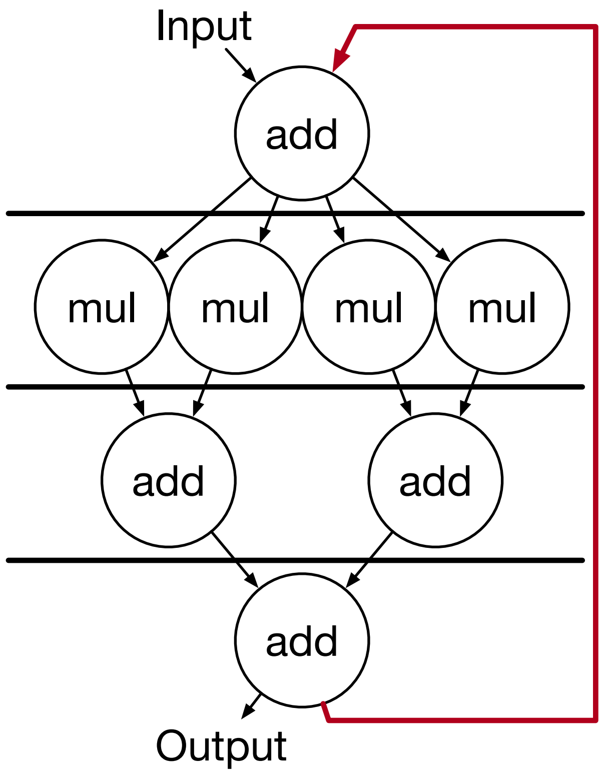
$$\sum_{q \in Q} a_q^S \cdot n_{q,r} \;\leq\; N_r \quad \forall r \in R$$

Trade-offs & (In-)Feasibility

- Example: Different trade-offs

Trade-offs & (In-)Feasibility

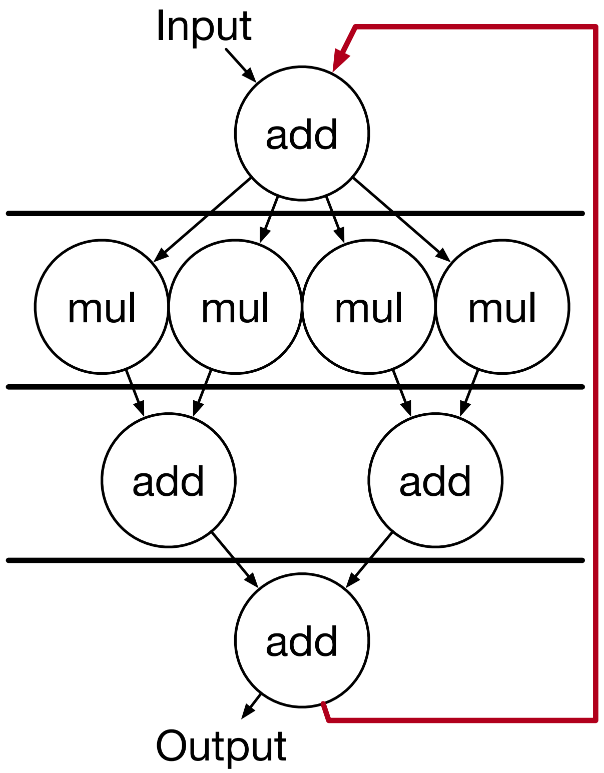
II=1 latency=4
4 mul 3 add



- Example: Different trade-offs

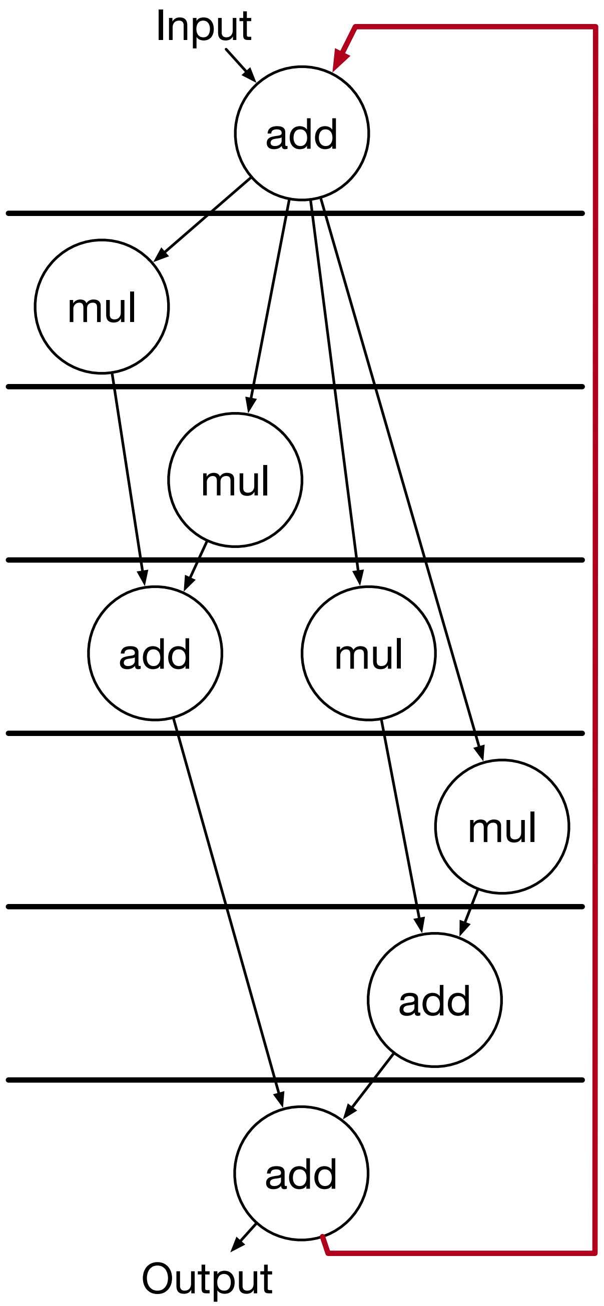
Trade-offs & (In-)Feasibility

II=1 latency=4
4 mul 3 add



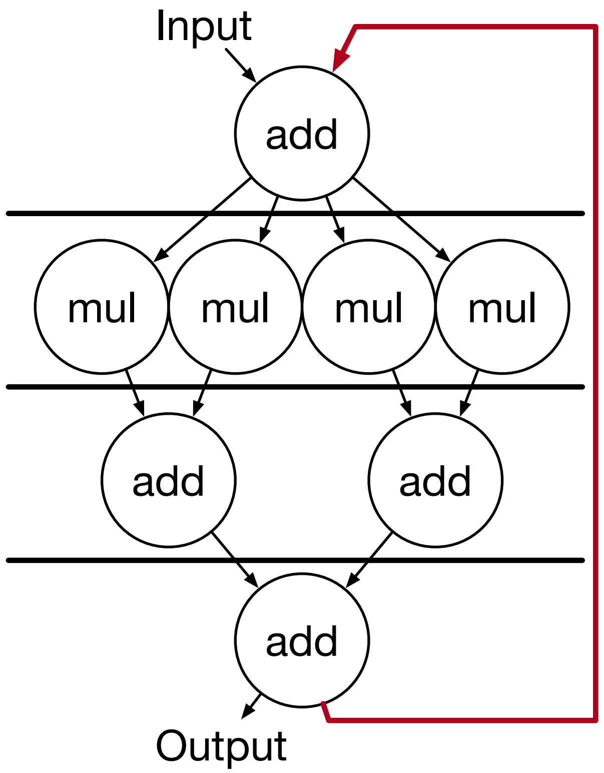
- Example: Different trade-offs

II=4 latency=7
1 mul 1 add



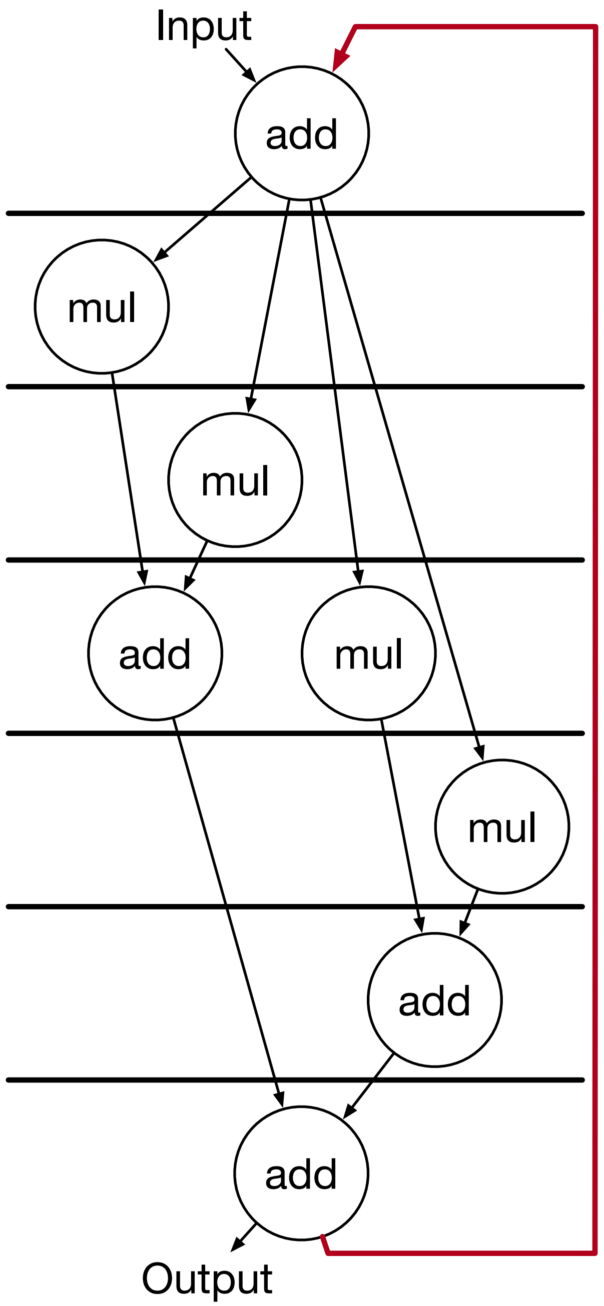
Trade-offs & (In-)Feasibility

II=1 latency=4
4 mul 3 add



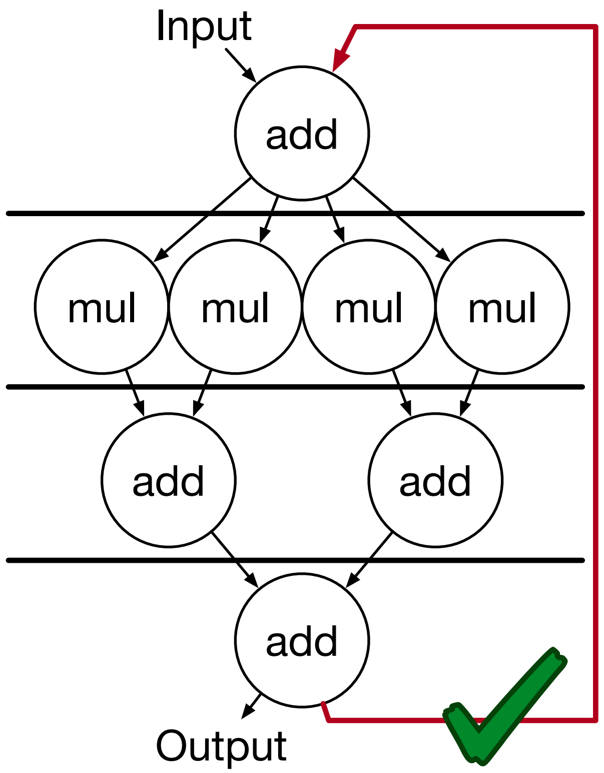
- Example: Different trade-offs
- *Isn't there an II to makes any allocation feasible (& vice versa)?*

II=4 latency=7
1 mul 1 add



Trade-offs & (In-)Feasibility

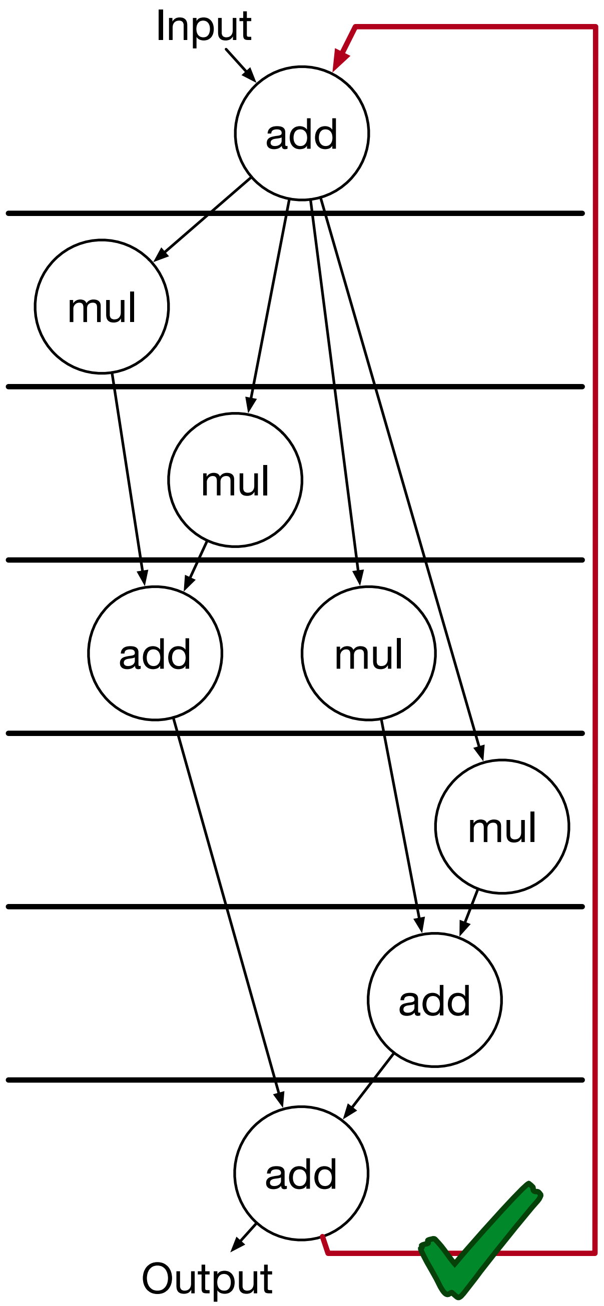
II=1 **latency=4**
4 mul **3 add**



Inter-iteration dependence

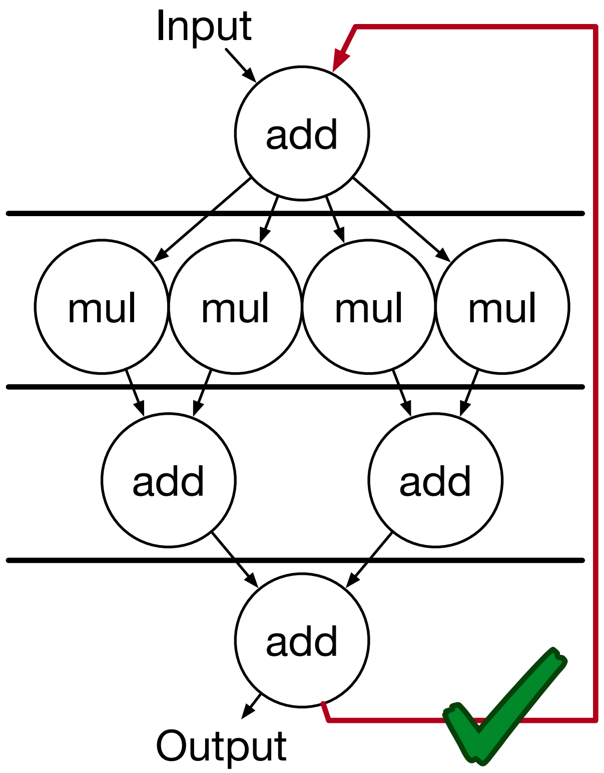
- Example: Different trade-offs
- *Isn’t there an II to makes any allocation feasible (& vice versa)?*
- Not if an operation is subject to a **deadline**
  - Inter-iteration dependences „`a[i] = f(a[i-4])`“

II=4 **latency=7**
1 mul **1 add**



Trade-offs & (In-)Feasibility

II=1 latency=4
4 mul 3 add

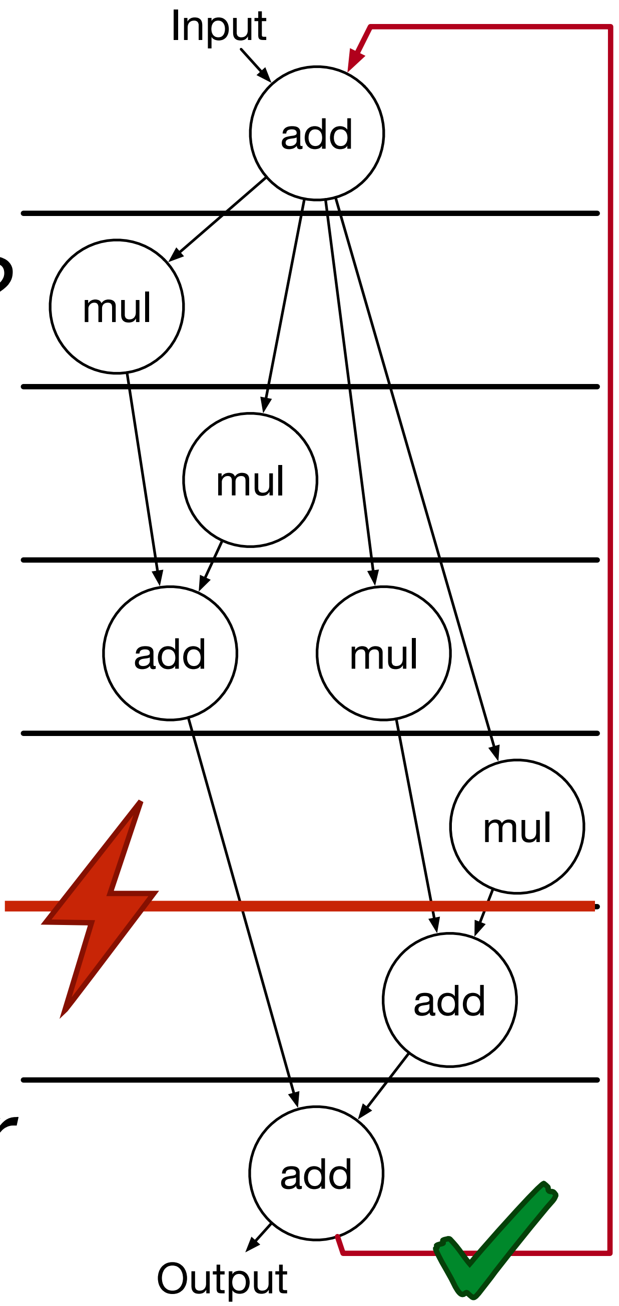


Inter-iteration dependence

External latency constraint

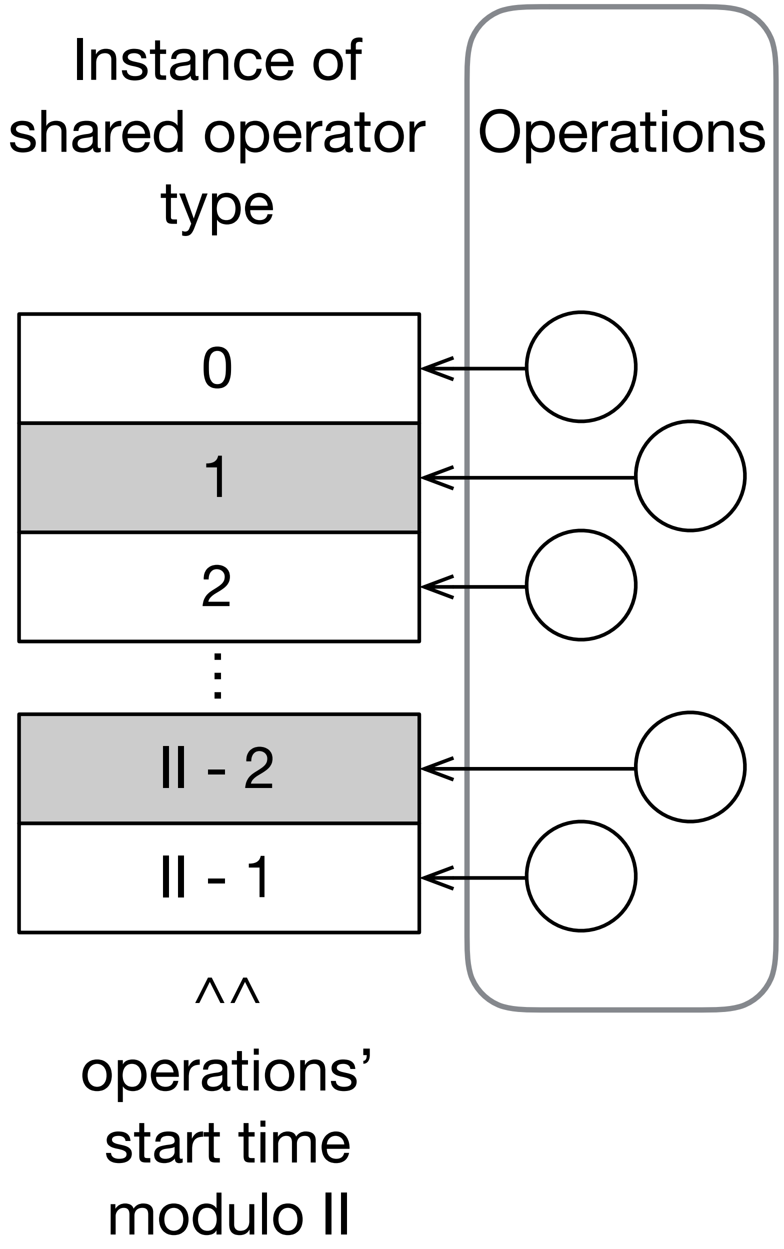
- Example: Different trade-offs
- *Isn’t there an II to makes any allocation feasible (& vice versa)?*
- Not if an operation is subject to a **deadline**
  - Inter-iteration dependences „`a[i] = f(a[i-4])`“
  - External latency constraints *„output must be available after 5 cycles“*

II=4 latency=7
1 mul 1 add



Trivial Allocation

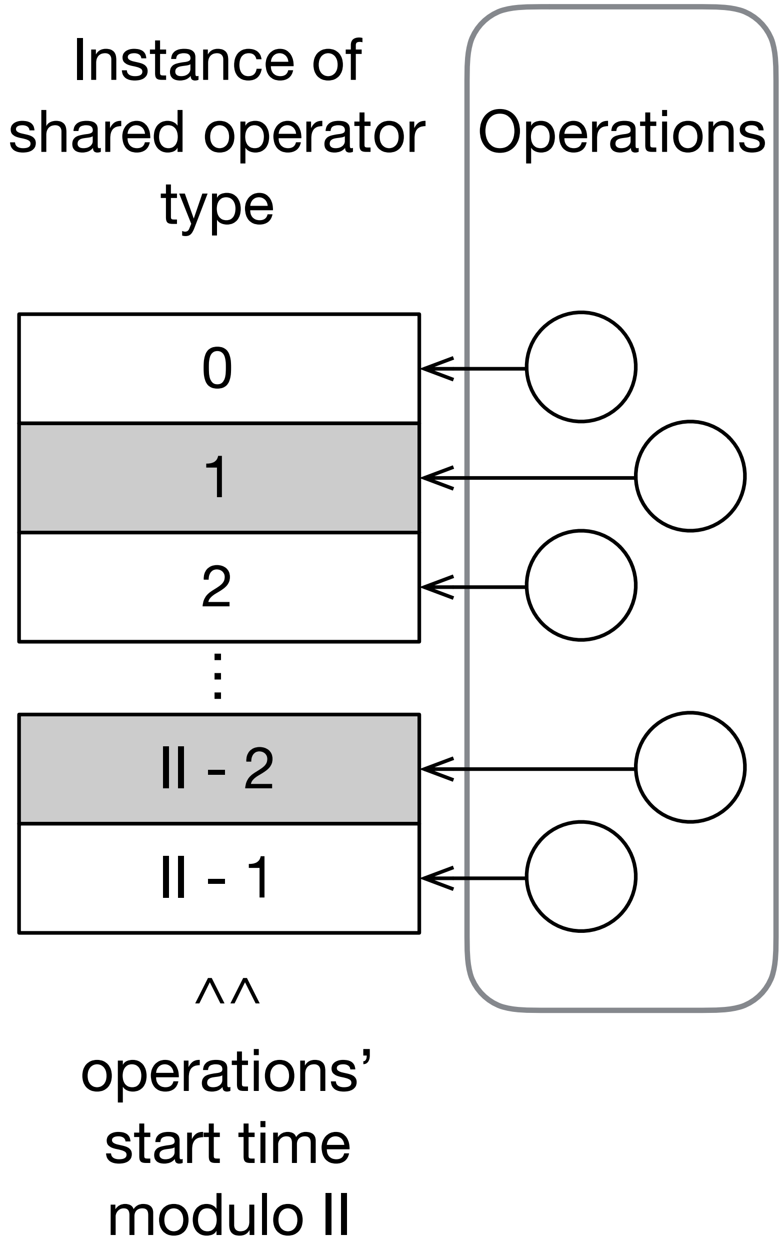
- One operator provides II-many slots


Instance of shared operator type
Operations
0
1
2
⋮
II - 2
II - 1
^^
operations' start time modulo II

Trivial Allocation

- One operator provides II-many slots
- Pigeonhole principle: *„Need enough slots to bind all operations“*

$$a_q^S \geq \left\lceil \frac{|O_q|}{II^S} \right\rceil \quad \forall q \in \hat{Q}$$



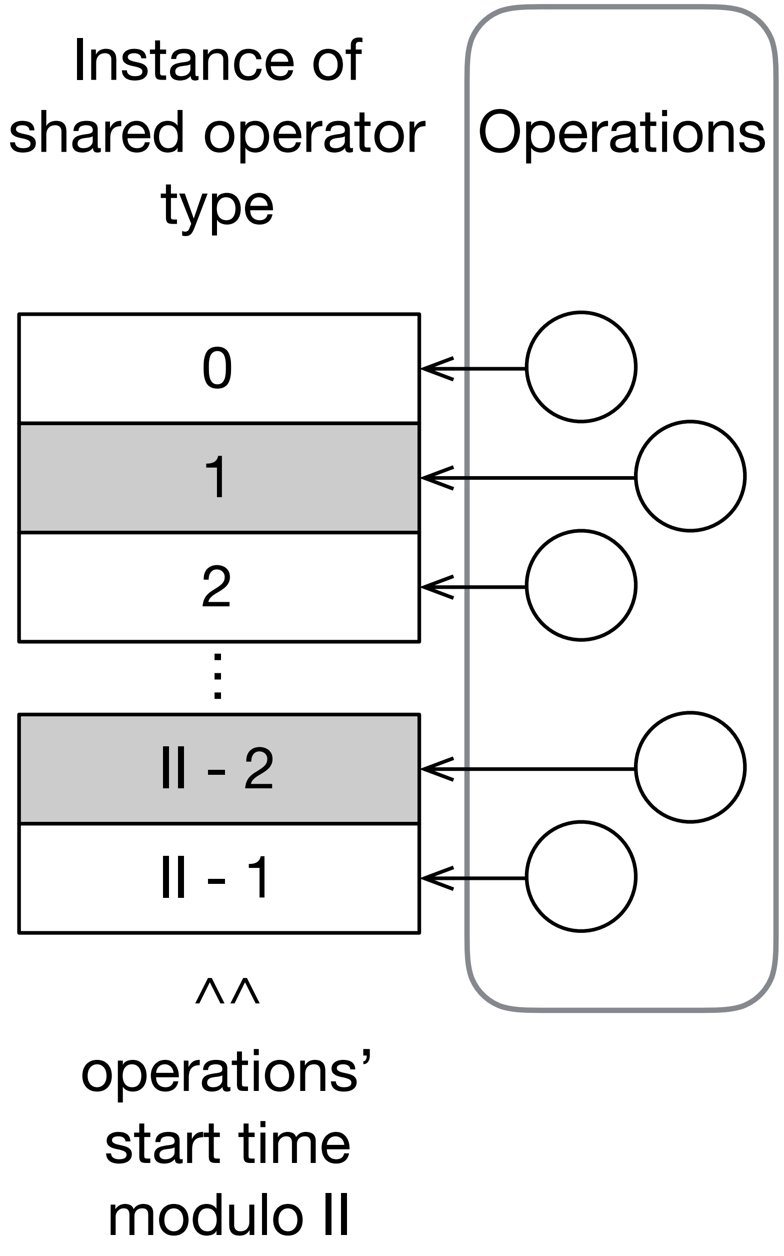
Instance of shared operator type
Operations
0
1
2
⋮
II - 2
II - 1
^^
operations' start time modulo II

Trivial Allocation

- One operator provides II-many slots
- Pigeonhole principle: *„Need enough slots to bind all operations“*

$$a_q^S \geq \left\lceil \frac{|O_q|}{II^S} \right\rceil \quad \forall q \in \hat{Q}$$

- We call an allocation A^S **trivial** iff a_q^S is equal to the RHS for all $q \in \hat{Q}$

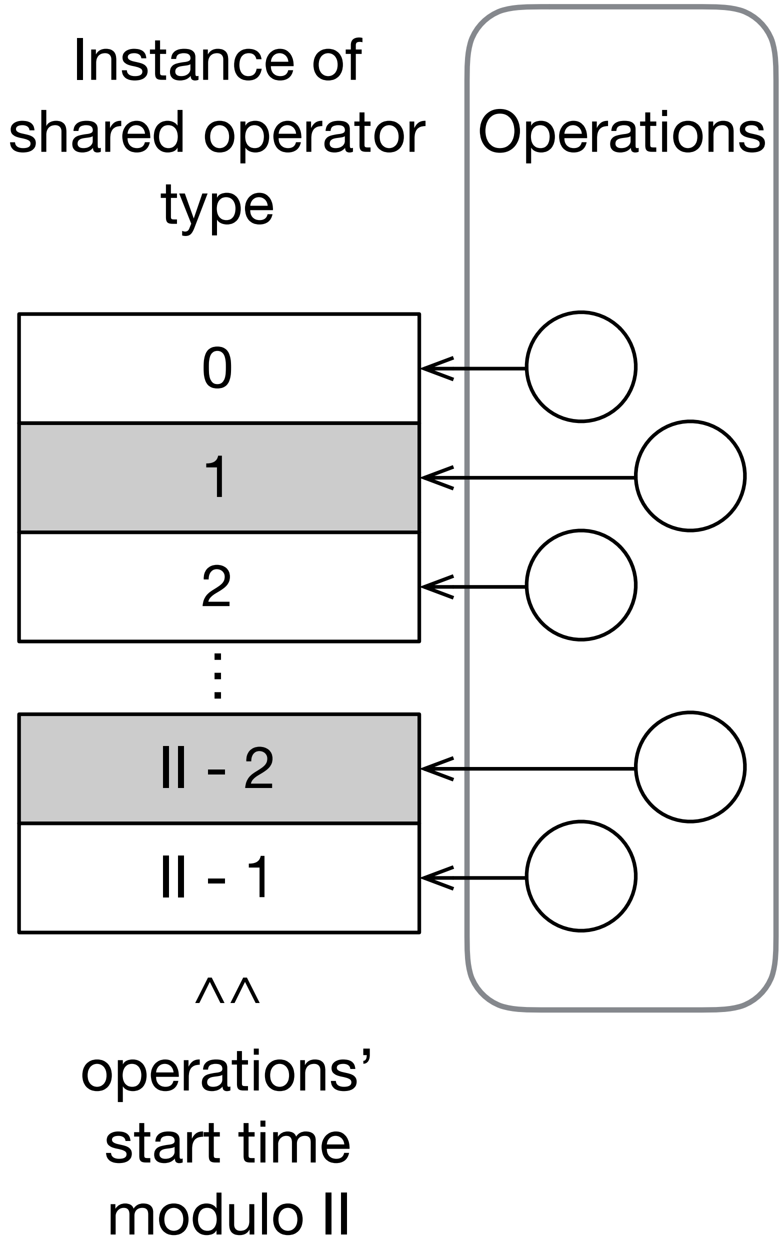

Instance of shared operator type
Operations
0
1
2
⋮
II - 2
II - 1
^^
operations' start time modulo II

Trivial Allocation

- One operator provides II-many slots
- Pigeonhole principle: *„Need enough slots to bind all operations“*

$$a_q^S \geq \left\lceil \frac{|O_q|}{II^S} \right\rceil \quad \forall q \in \hat{Q}$$

- We call an allocation A^S **trivial** iff a_q^S is equal to the RHS for all $q \in \hat{Q}$
- Trivial allocation may be infeasible!


Instance of shared operator type
Operations
0
1
2
⋮
II - 2
II - 1
^^
operations' start time modulo II

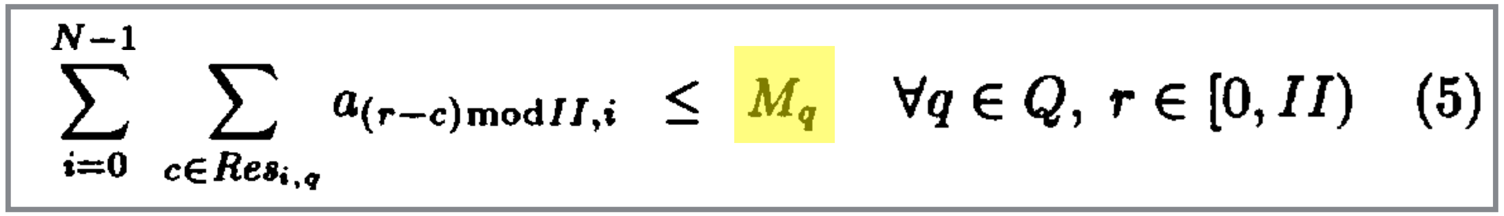
Resource-Aware Extension

- Making an ILP-based modulo scheduler resource-aware = replace formerly constant limits, adapt objectives

Resource-Aware Extension

- Making an ILP-based modulo scheduler resource-aware = replace formerly constant limits, adapt objectives

- We extended:
  - **ED**: formulation by Eichenberger & Davidson (1997)

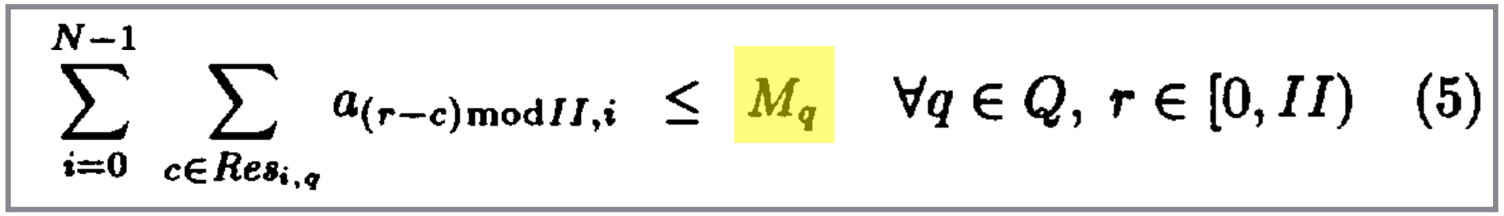


$$\sum_{i=0}^{N-1} \sum_{c \in Res_{i,q}} a_{(r-c)\mathrm{mod}\,II,i} \leq M_q \quad \forall q \in Q,\ r \in [0, II) \quad (5)$$

Resource-Aware Extension

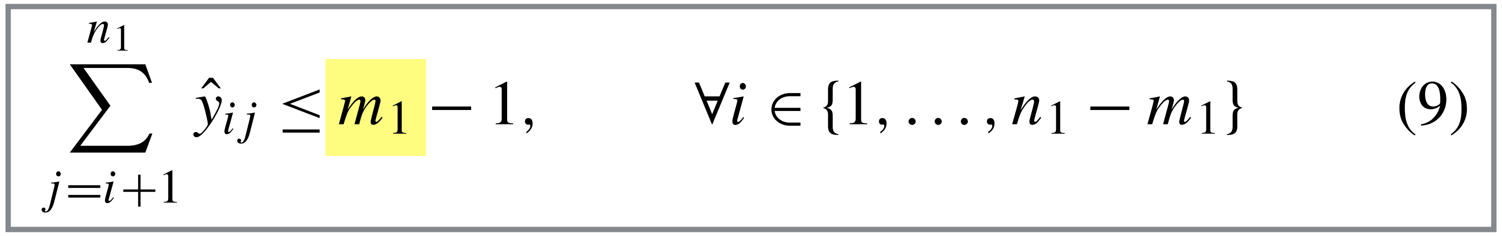
- Making an ILP-based modulo scheduler resource-aware = replace formerly constant limits, adapt objectives

- We extended:
  - **ED**: formulation by Eichenberger & Davidson (1997)



$$\sum_{i=0}^{N-1} \sum_{c \in Res_{i,q}} a_{(r-c)\bmod II,i} \le M_q \qquad \forall q \in Q,\ r \in [0, II) \tag{5}$$

  - **SH**: formulation by Šůcha & Hanzálek (2009)



$$\sum_{j=i+1}^{n_1} \hat{y}_{ij} \le m_1 - 1, \qquad \forall i \in \{1, \ldots, n_1 - m_1\} \tag{9}$$

Resource-Aware Extension

- Making an ILP-based modulo scheduler resource-aware = replace formerly constant limits, adapt objectives

- We extended:
  - **ED**: formulation by Eichenberger & Davidson (1997)

$$\sum_{i=0}^{N-1} \sum_{c \in Res_{i,q}} a_{(r-c) \bmod II, i} \leq M_q \qquad \forall q \in Q,\ r \in [0, II) \qquad (5)$$

  - **SH**: formulation by Šůcha & Hanzálek (2009)

$$\sum_{j=i+1}^{n_1} \hat{y}_{ij} \leq m_1 - 1, \qquad \forall i \in \{1, \ldots, n_1 - m_1\} \qquad (9)$$

  - **MV**: „Moovac“ formulation by Oppermann et al. (2019)

$$r_i \leq a_k - 1 \qquad \forall k \in R : \forall i \in L_k \qquad (\mathrm{M11})$$

Agenda

- □ Introduction to high-level synthesis
- □ Resource-aware modulo scheduling
- ▷ **Exploration of trade-off solutions**
- □ Experimental evaluation

MORAMS

- Multi-Objective Resource-Aware Modulo Scheduling

MORAMS

- Multi-Objective Resource-Aware Modulo Scheduling
- Given a RAMS problem, the goal is to compute all **Pareto-optimal** solutions

MORAMS

- Multi-Objective Resource-Aware Modulo Scheduling
- Given a RAMS problem, the goal is to compute all **Pareto-optimal** solutions
  - i.e. a solution that is **not dominated** by any other solution

MORAMS

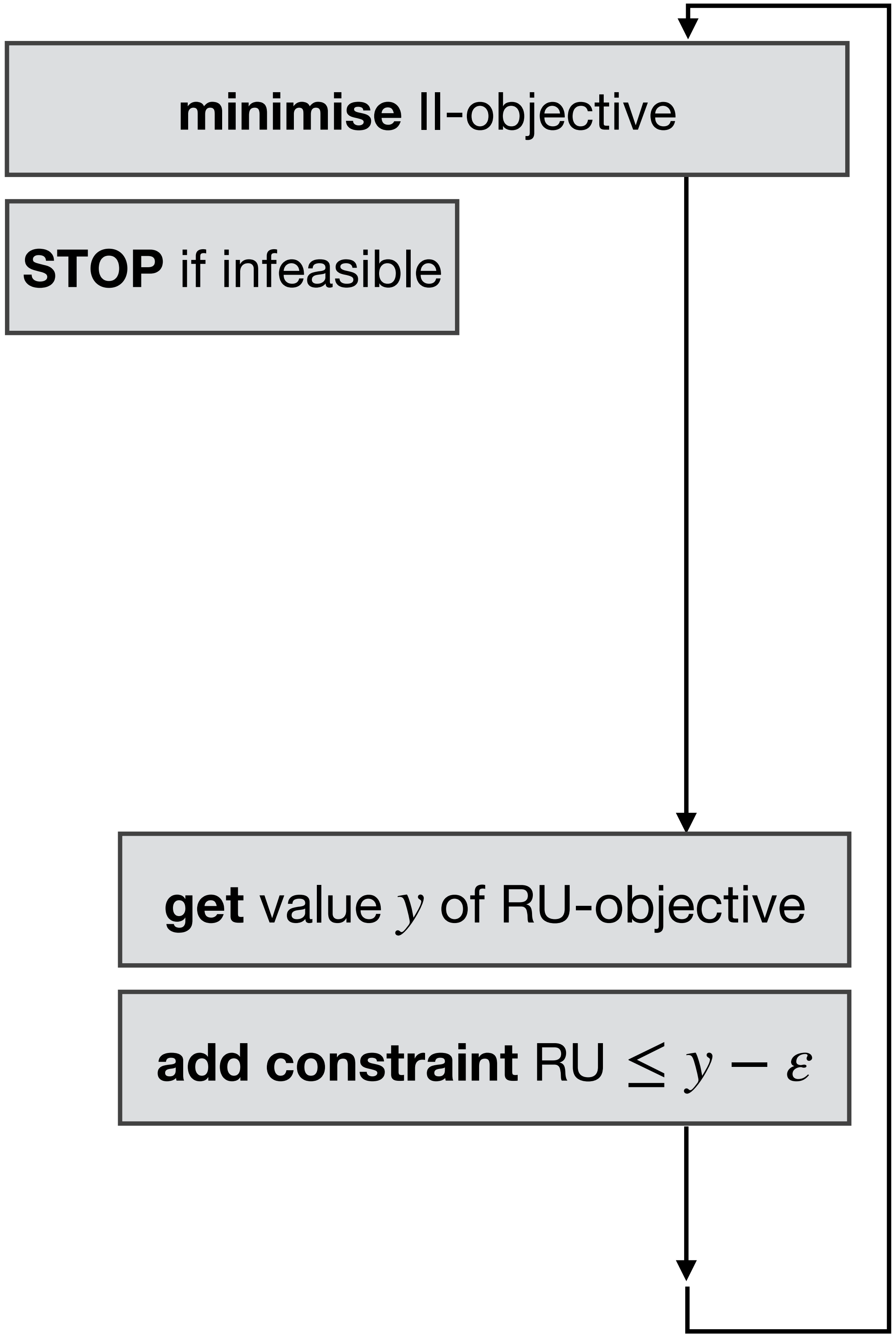
- Multi-Objective Resource-Aware Modulo Scheduling
- Given a RAMS problem, the goal is to compute all **Pareto-optimal** solutions
  - i.e. a solution that is **not dominated** by any other solution
  - Example:
    II=3, RU=30% …
    … dominates II=4, RU=40%
    … does not dominate II=2, RU=40%

ε-Approach

- standard method from multi-criteria optimisation

ε-Approach

- standard method from multi-criteria optimisation
  - basically: optimise one objective, and successively add constraints for the other



minimise II-objective

STOP if infeasible

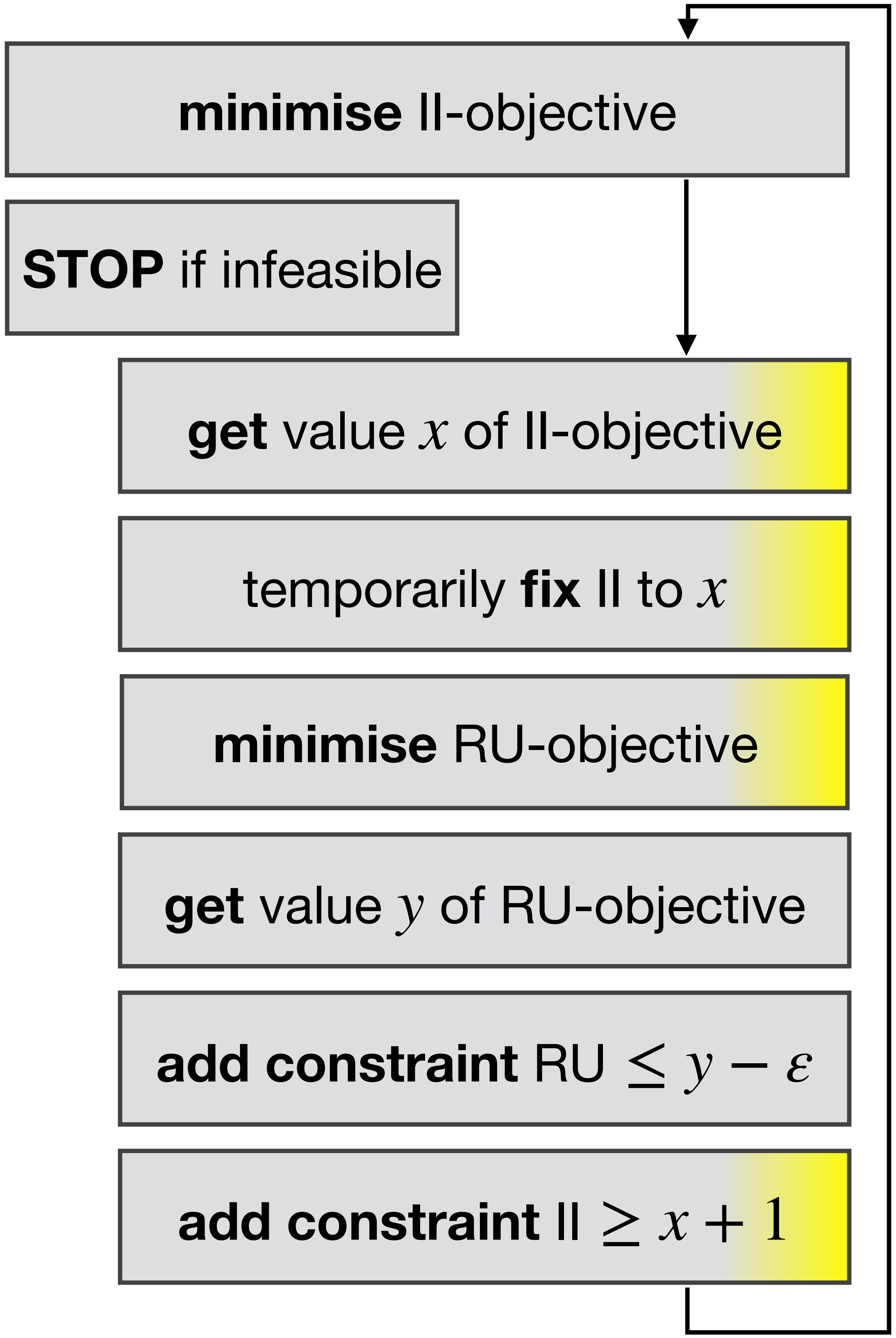
get value y of RU-objective

add constraint RU $\leq y - \varepsilon$

ε-Approach

🟨 = MORAMS-specific

- standard method from multi-criteria optimisation
  - basically: optimise one objective, and successively add constraints for the other
  - we minimise both objectives to speed up convergence



minimise II-objective

STOP if infeasible

get value x of II-objective

temporarily **fix** II to x

minimise RU-objective

get value y of RU-objective

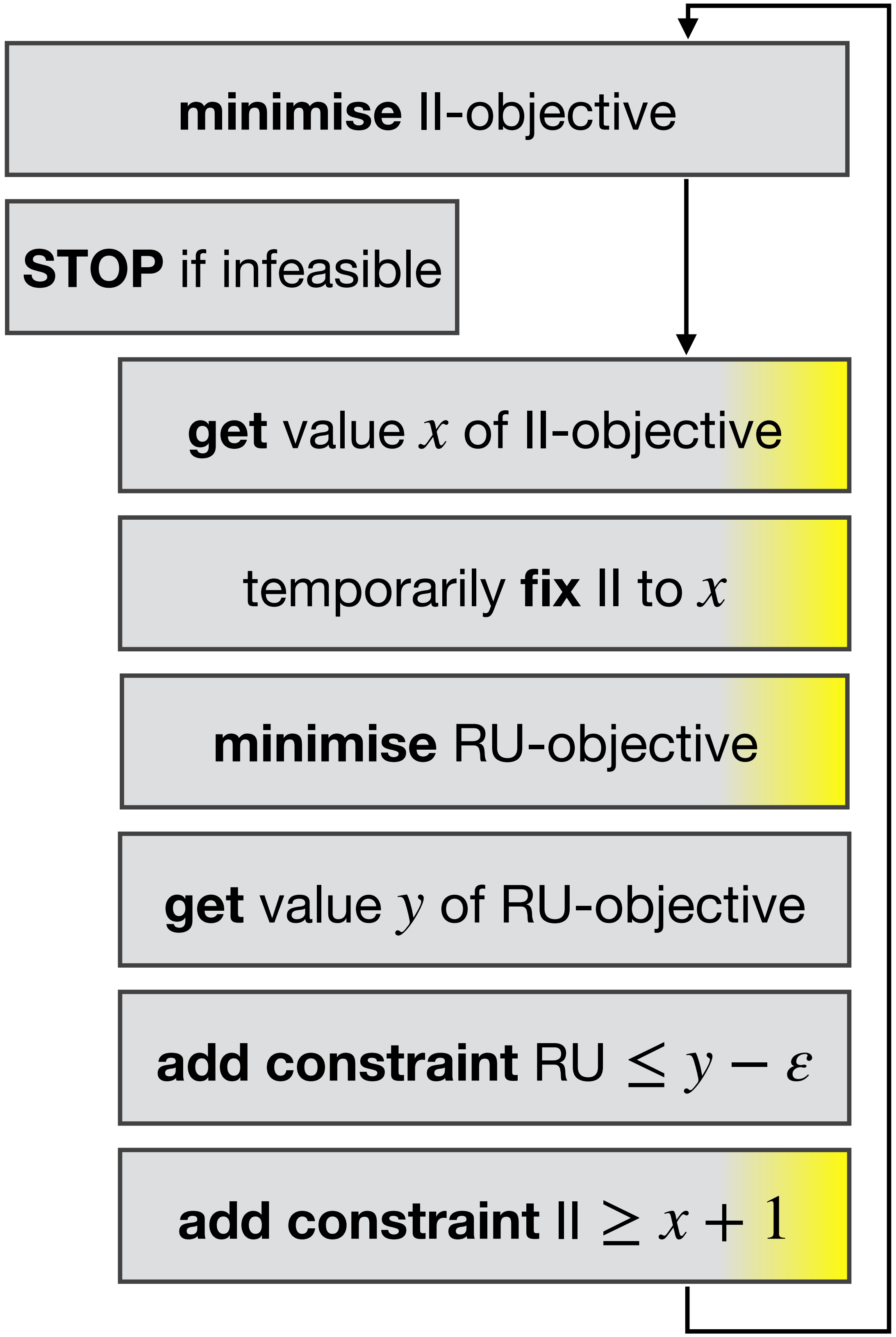
add constraint RU $\leq y - \varepsilon$

add constraint II $\geq x + 1$

ε-Approach

🟨 = MORAMS-specific

- standard method from multi-criteria optimisation
  - basically: optimise one objective, and successively add constraints for the other
  - we minimise both objectives to speed up convergence
  - requires RAMS formulation with *variable* II
    Here: extended „Moovac-I“ formulation (Oppermann et al., 2019)



minimise II-objective

STOP if infeasible

get value x of II-objective

temporarily **fix** II to x

minimise RU-objective

get value y of RU-objective

add constraint RU $\leq y - \varepsilon$

add constraint II $\geq x + 1$

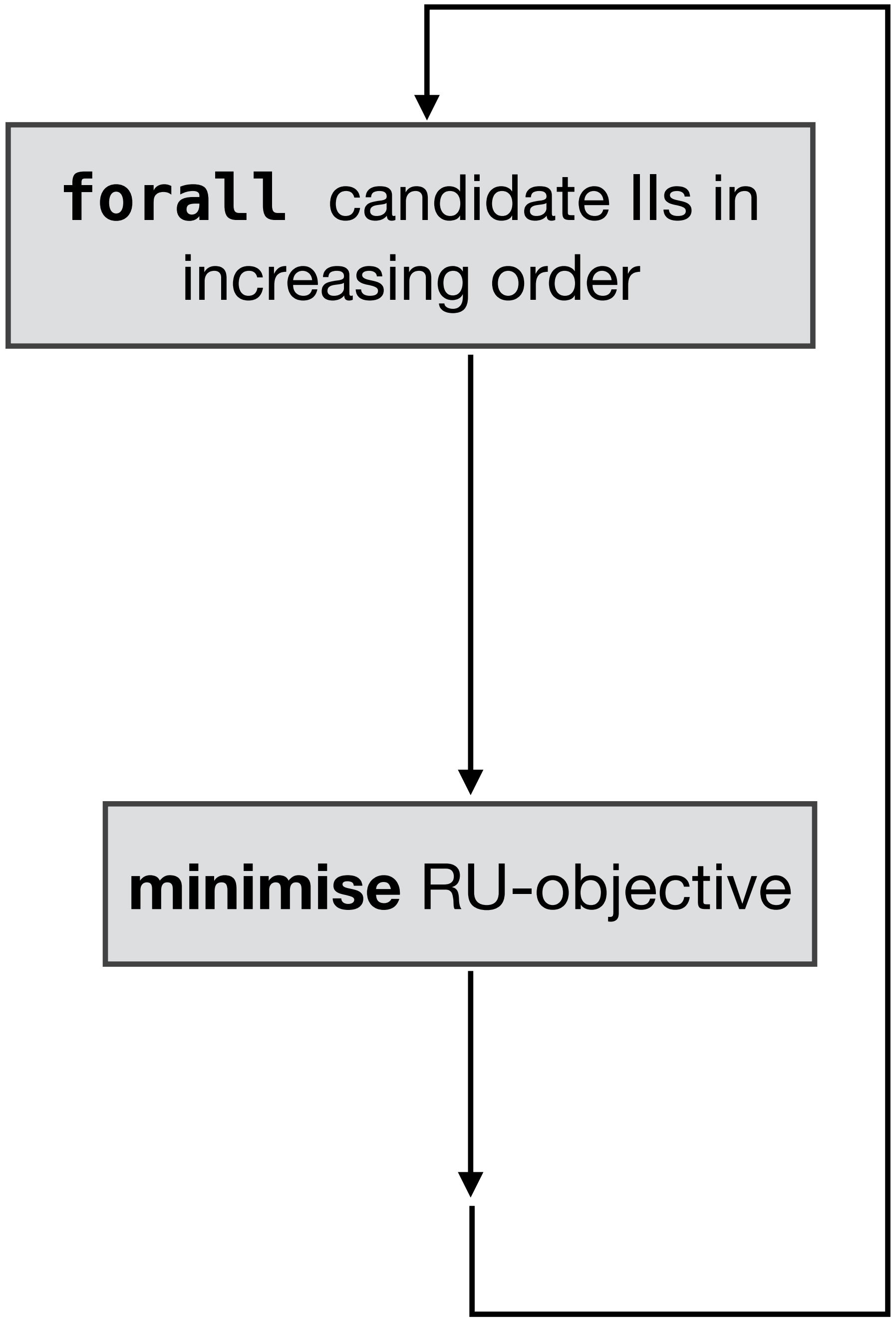
Iterative Approach

- Most existing modulo schedulers already try **candidate IIs** until the first feasible solution is found

Iterative Approach

- Most existing modulo schedulers already try **candidate IIs** until the first feasible solution is found

- Instead, we explore by trying all candidate IIs ...

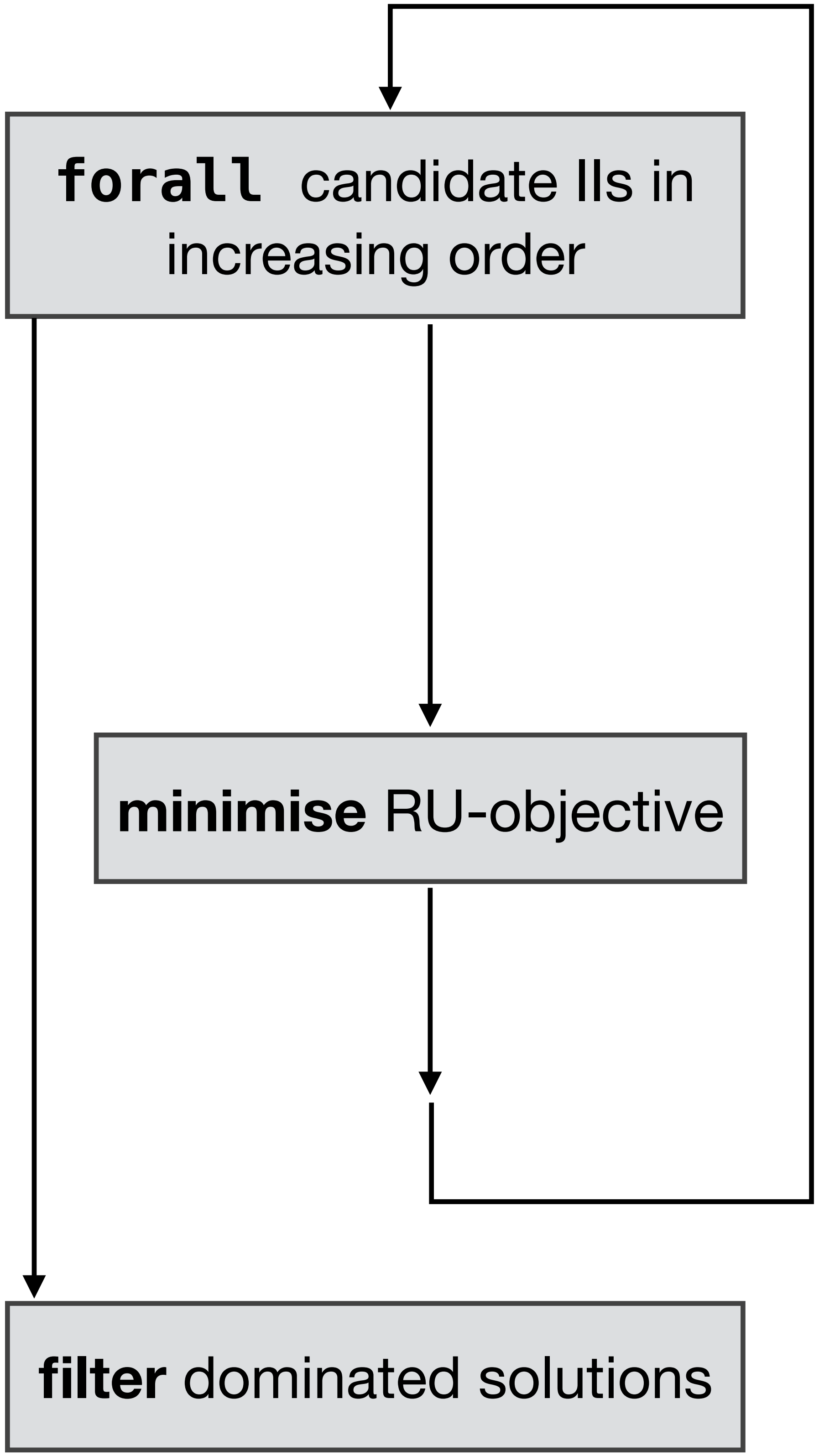


forall candidate IIs in increasing order

minimise RU-objective

Iterative Approach

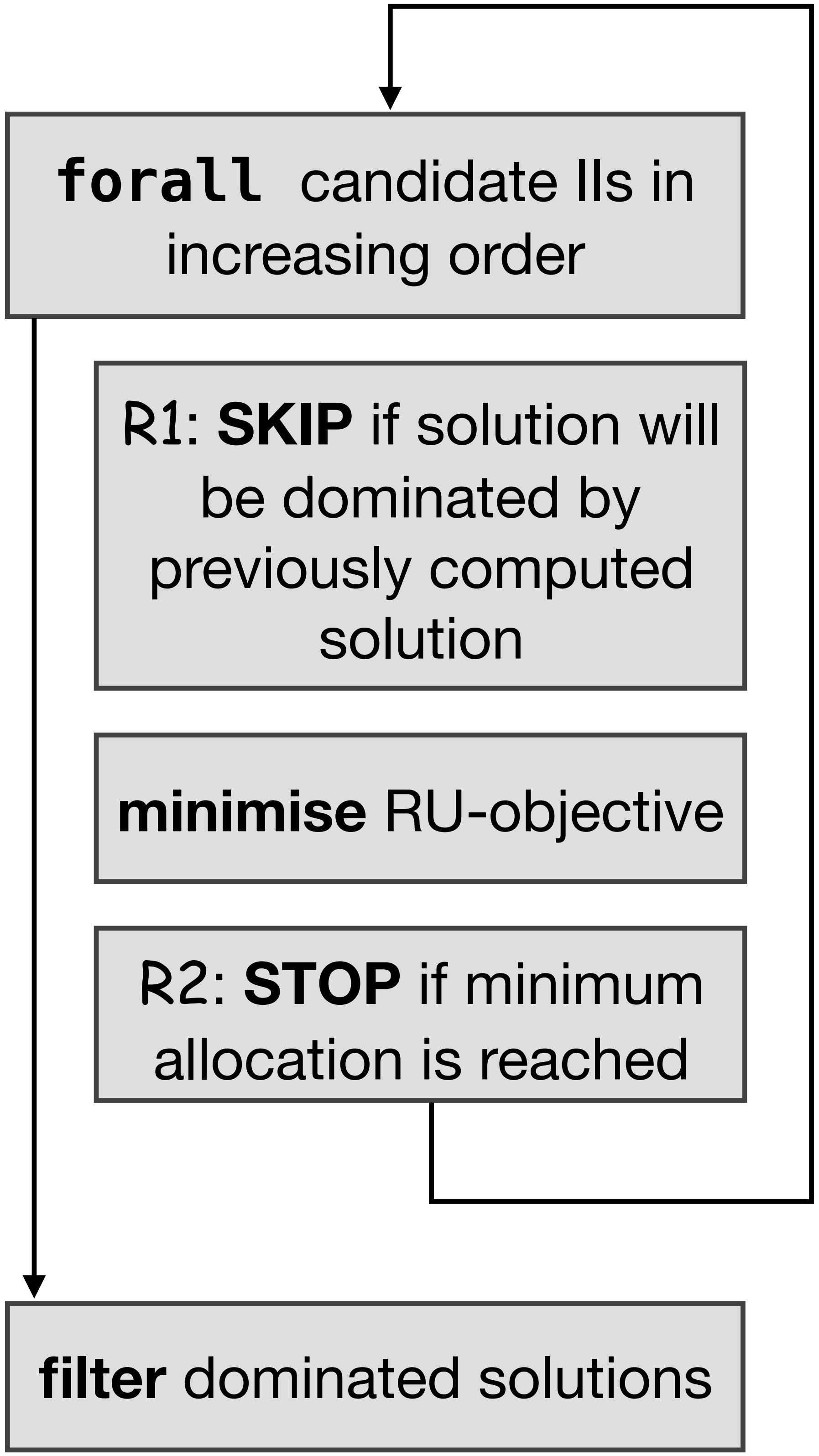
- Most existing modulo schedulers already try **candidate IIs** until the first feasible solution is found
- Instead, we explore by trying all candidate IIs ...
- ... and filter out dominated solutions afterwards


forall candidate IIs in increasing order
minimise RU-objective
filter dominated solutions

]

Iterative Approach

- Most existing modulo schedulers already try **candidate IIs** until the first feasible solution is found
- Instead, we explore by trying all candidate IIs ...
- ... and filter out dominated solutions afterwards
- We propose rules R1, R2 to compute fewer dominated solutions → next slide



forall candidate IIs in increasing order
R1: **SKIP** if solution will be dominated by previously computed solution
minimise RU-objective
R2: **STOP** if minimum allocation is reached
filter dominated solutions

Domain-Specific Rules

- **R1**: Skip candidate interval II^X if

Domain-Specific Rules

- R1: Skip candidate interval II^X if
  - we have a solution P with $II^P < II^X$, and

Domain-Specific Rules

- R1: Skip candidate interval II^X if
  - we have a solution P with $II^P < II^X$, and
  - its allocation A^P is trivial for II^X

Domain-Specific Rules

- R1: Skip candidate interval II^X if
  - we have a solution P with $II^P < II^X$, and
  - its allocation A^P is trivial for II^X
- R2: Stop exploration if

Domain-Specific Rules

- R1: Skip candidate interval II^X if
  - we have a solution P with $II^P < II^X$, and
  - its allocation A^P is trivial for II^X

- R2: Stop exploration if
  - solution with the minimum allocation

    $$a_q = 1 \quad q \in \hat{Q}$$

    is found

Agenda

- □ Introduction to high-level synthesis
- □ Resource-aware modulo scheduling
- □ Exploration of trade-off solutions
- ▷ **Experimental evaluation**

Setup

- **Gurobi 8.1, 8 threads, 16 GiB RAM, 6h time limit per instance** on Xeon E5-2680 v3 servers @ 2.8 GHz

Setup

- **Gurobi 8.1, 8 threads, 16 GiB RAM, 6h time limit per instance** on Xeon E5-2680 v3 servers @ 2.8 GHz

- **204 realistic test instances** from Simulink models and C-based HLS benchmark suites

	min.	median	mean	max.
# operations	14	49	104	1374
# shared operations	0	4	16	416
# edges	17	81	237	4441
# backedges	0	3	23	1155

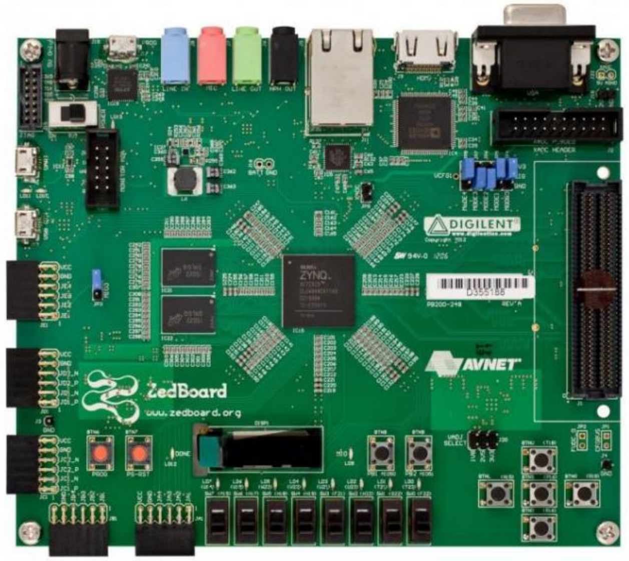
Setup

- **Gurobi 8.1, 8 threads, 16 GiB RAM, 6h time limit per instance** on Xeon E5-2680 v3 servers @ 2.8 GHz

- **204 realistic test instances** from Simulink models and C-based HLS benchmark suites

	min.	median	mean	max.
# operations	14	49	104	1374
# shared operations	0	4	16	416
# edges	17	81	237	4441
# backedges	0	3	23	1155

- Modelled resources for Xilinx XC7Z020 FPGA: **LUT (53,200), DSP (220), memory ports (16)**



Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions

Effectiveness of Rules

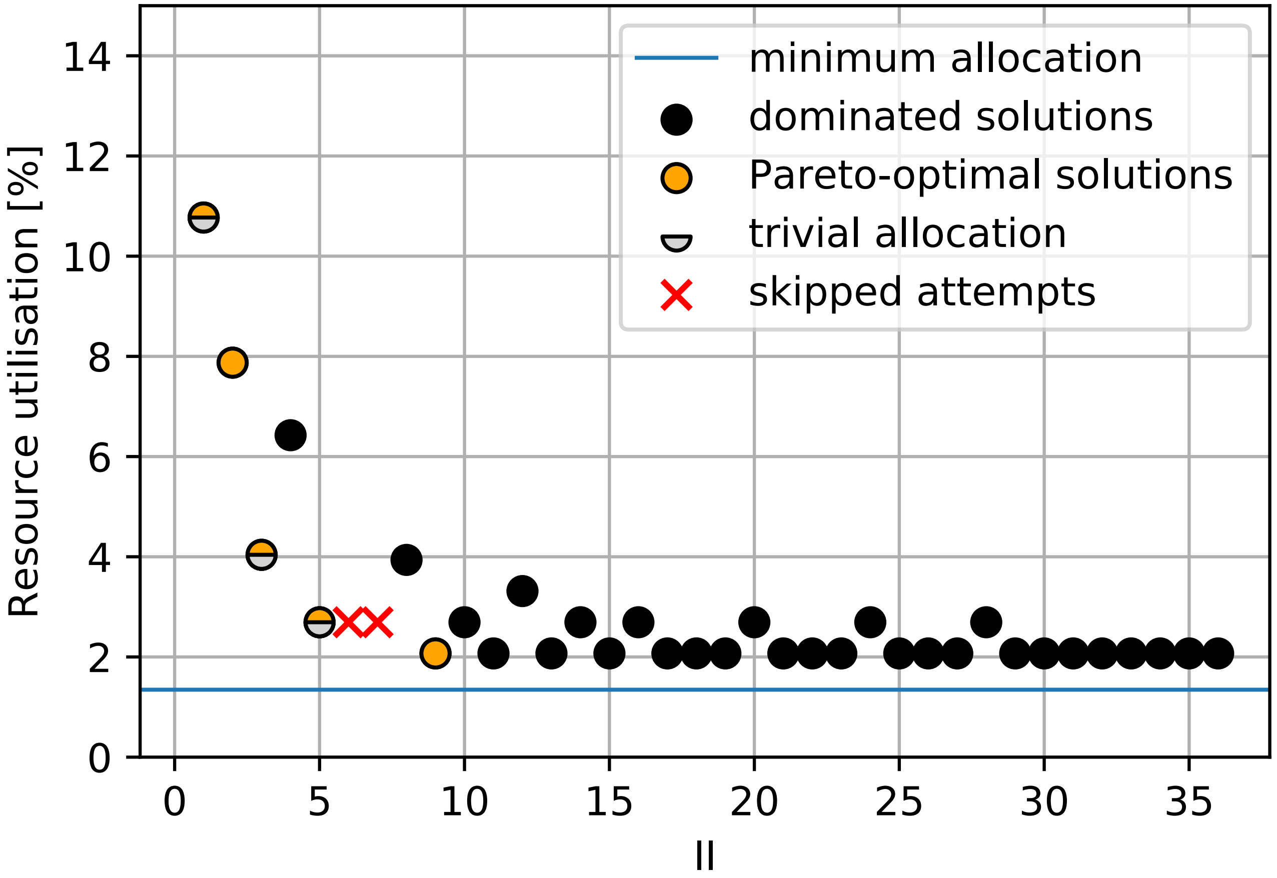
- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions
- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions

- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps

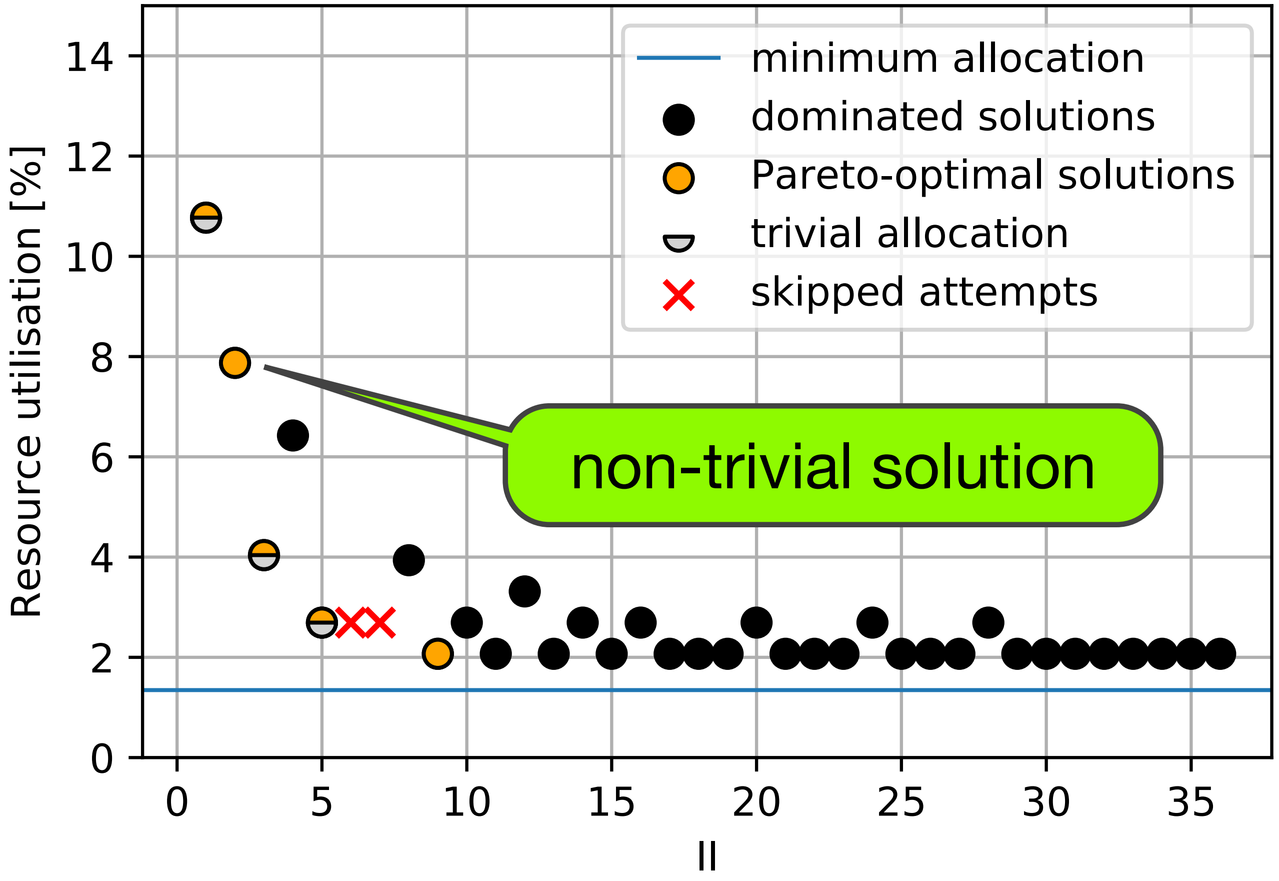


Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions

- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps



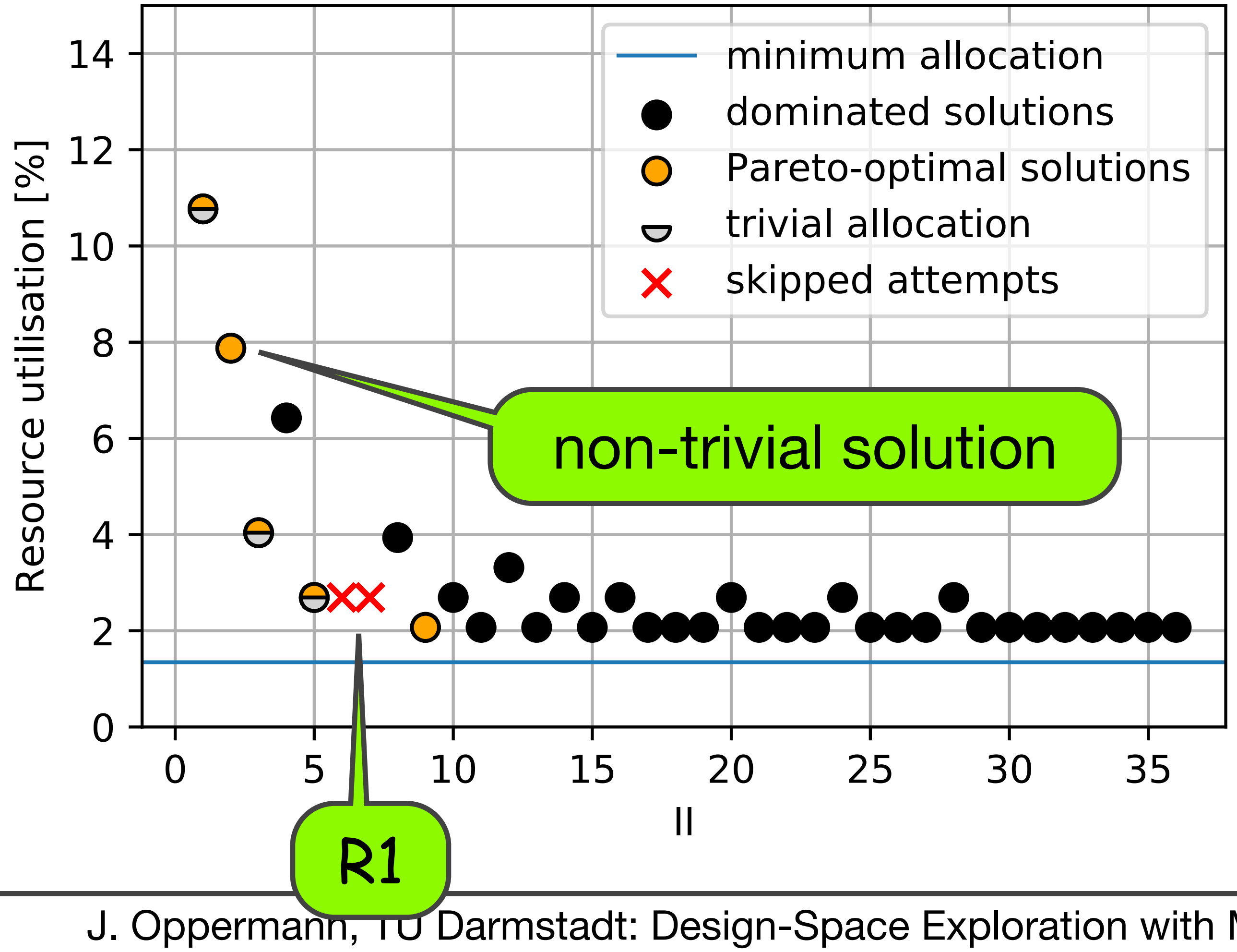
non-trivial solution

Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions

- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps



non-trivial solution

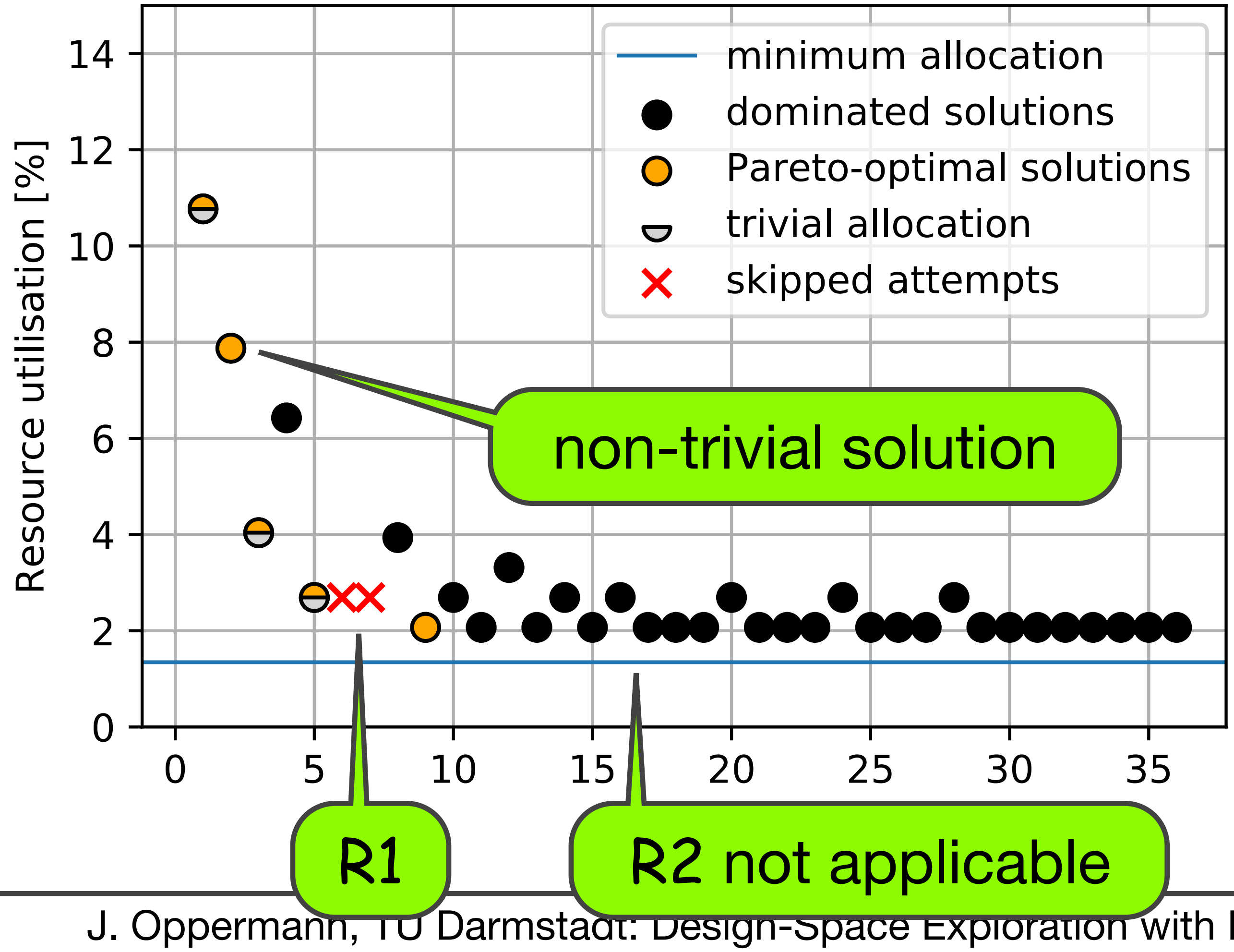
R1

Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions

- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps



non-trivial solution

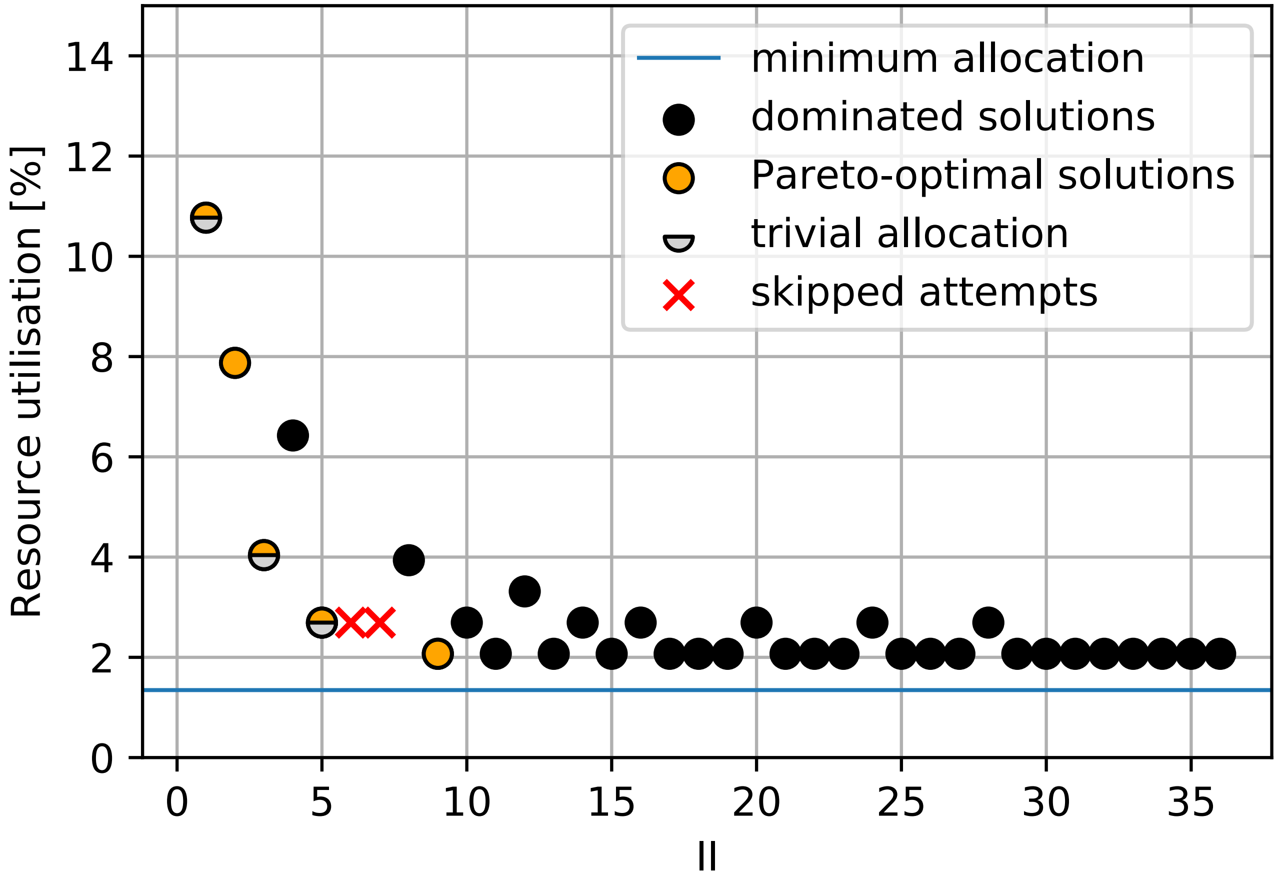
R1

R2 not applicable

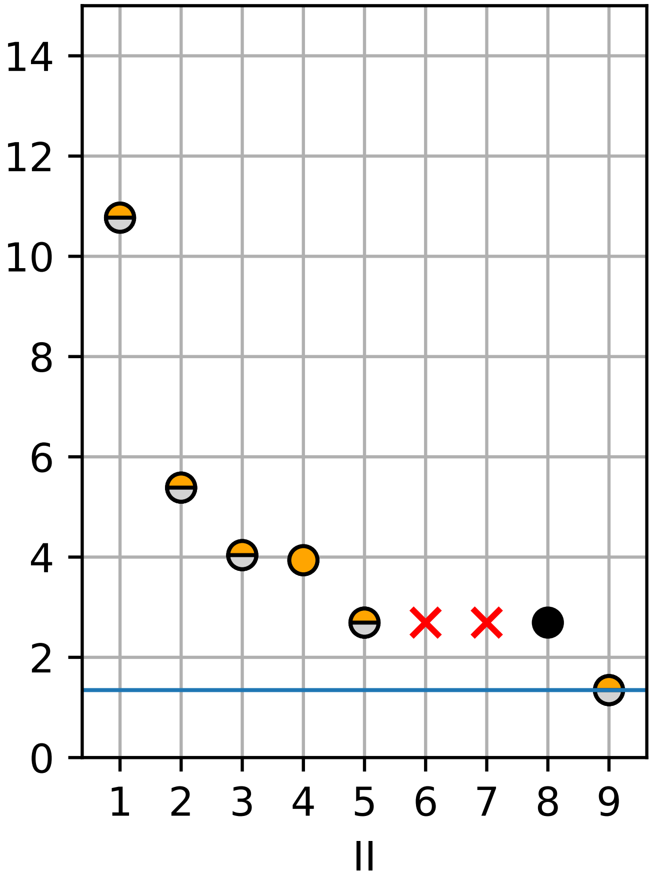
Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions
- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps



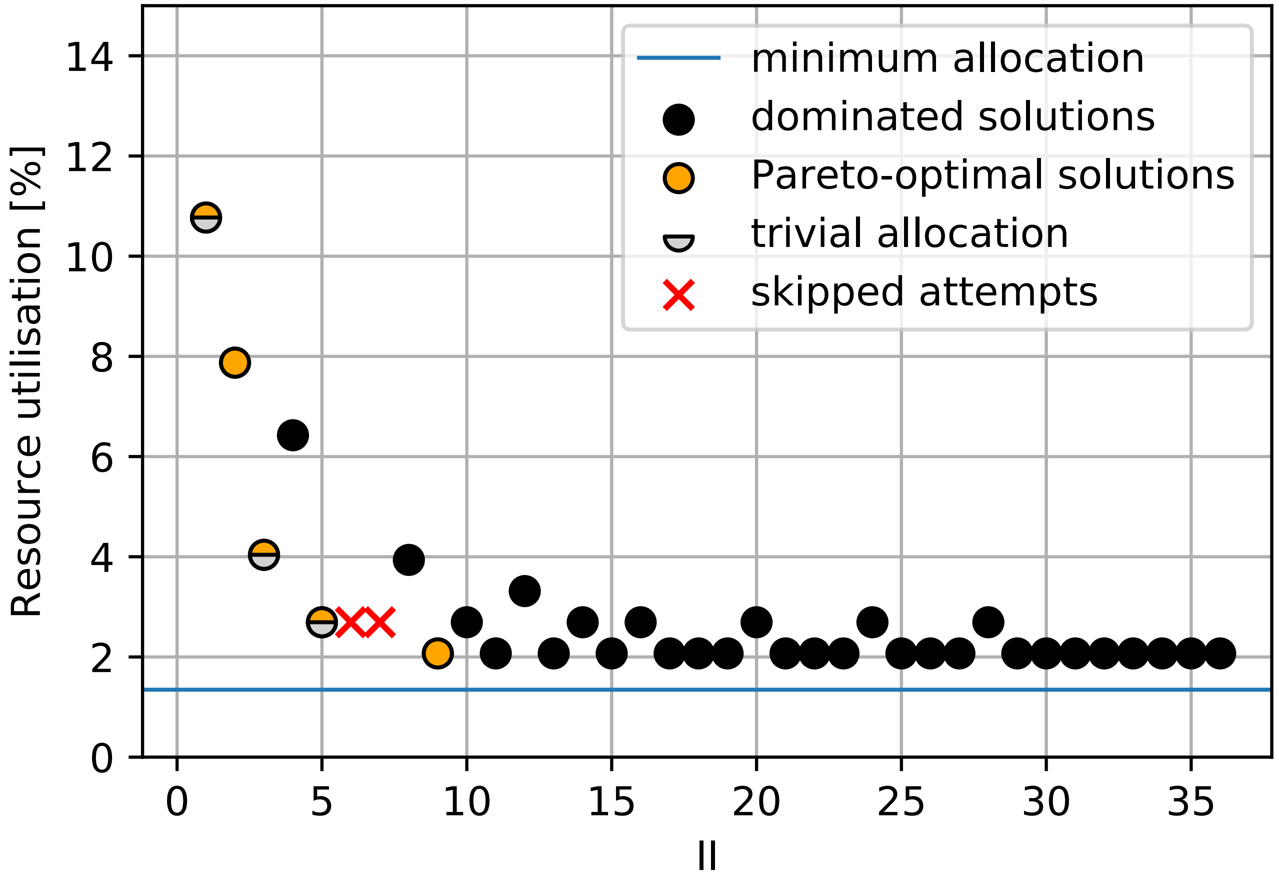
moderate = 37



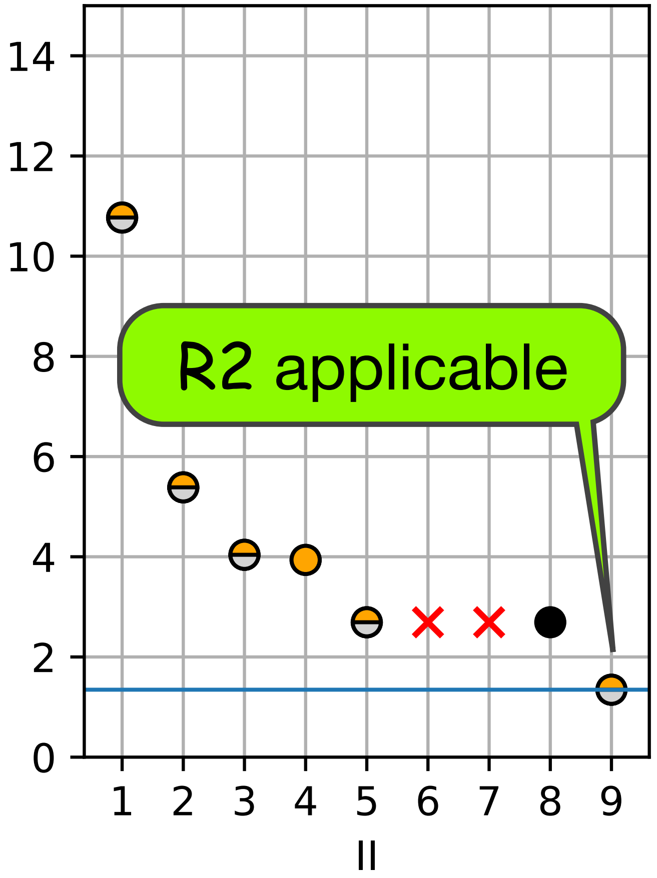
Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions
- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“

tight = 36 time steps



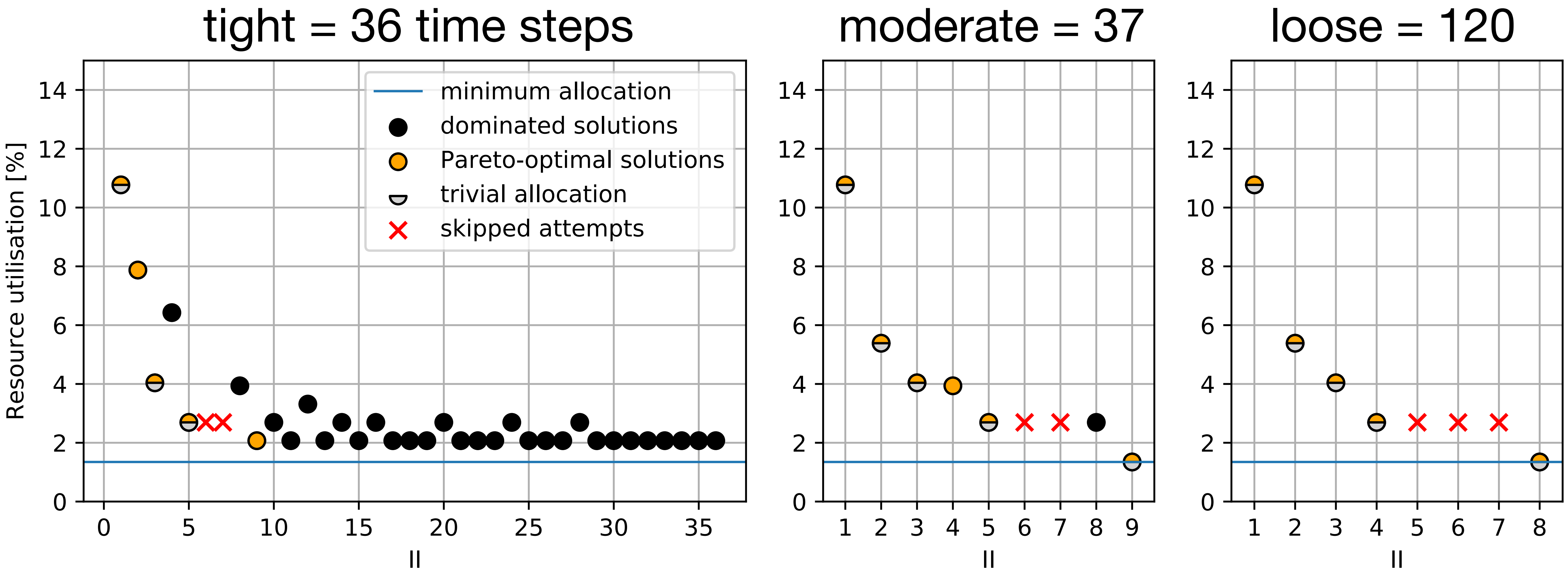
moderate = 37



R2 applicable

Effectiveness of Rules

- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions
- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“



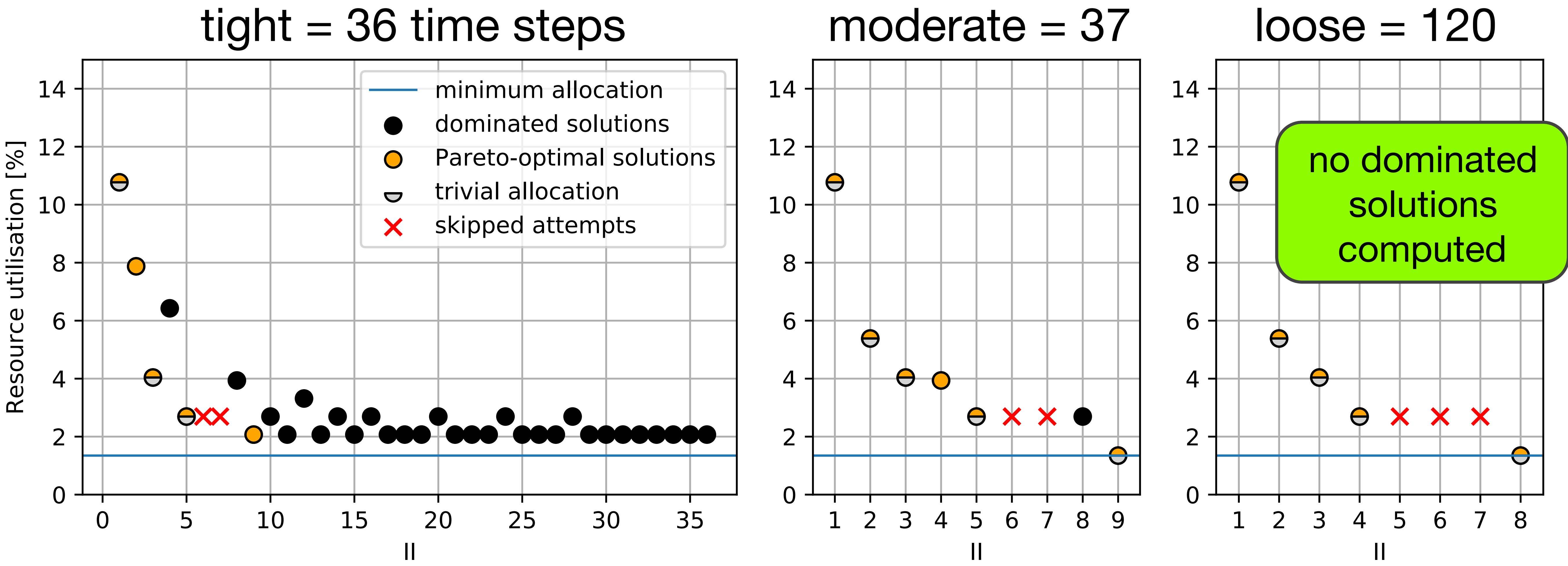
tight = 36 time steps

moderate = 37

loose = 120

Effectiveness of Rules

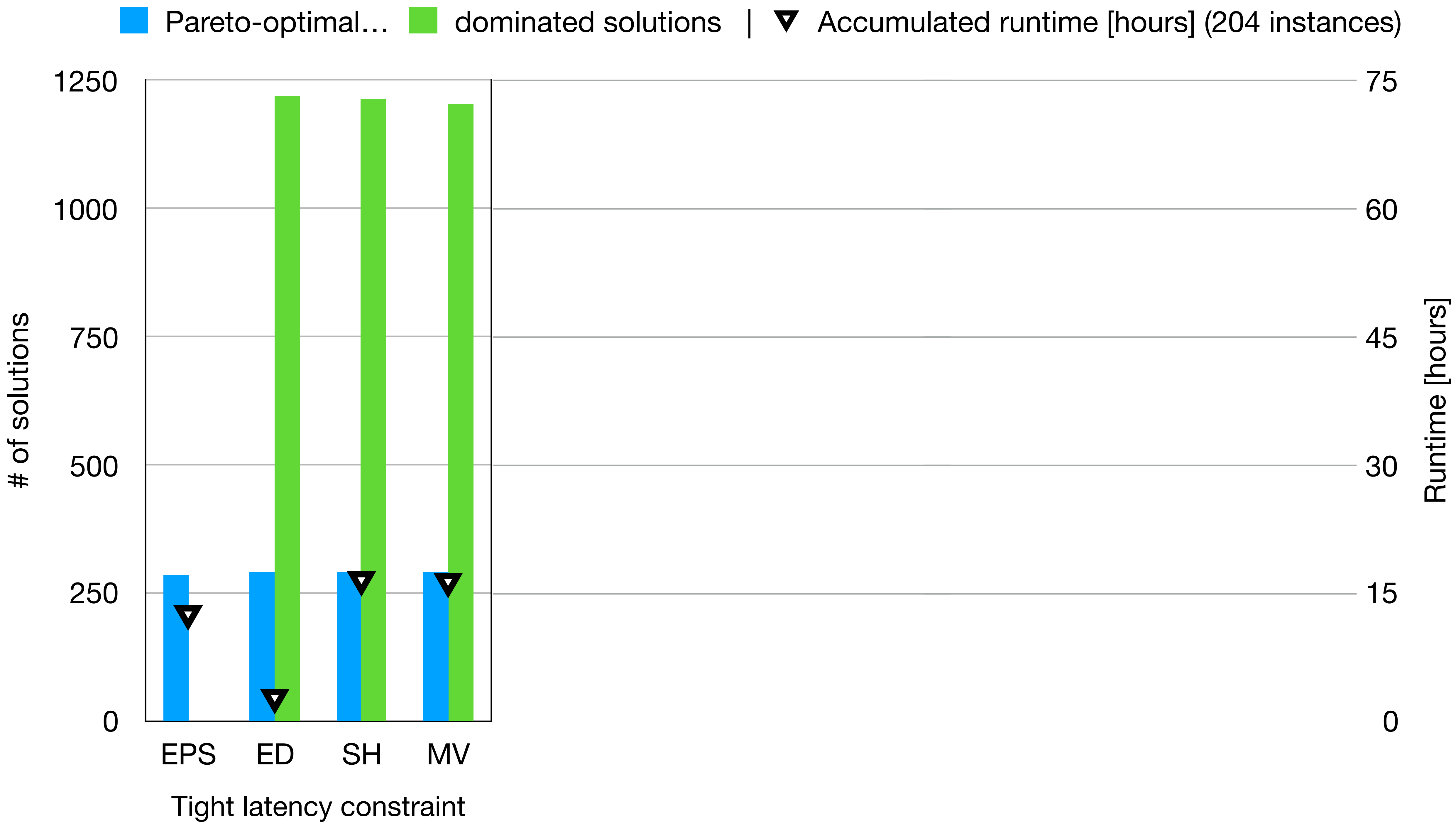
- Effectiveness of R1, R2 for iterative approach depends on feasibility of trivial solutions
- Experimented with different external latency constraints: „tight“, „moderate“ and „loose“



no dominated solutions computed

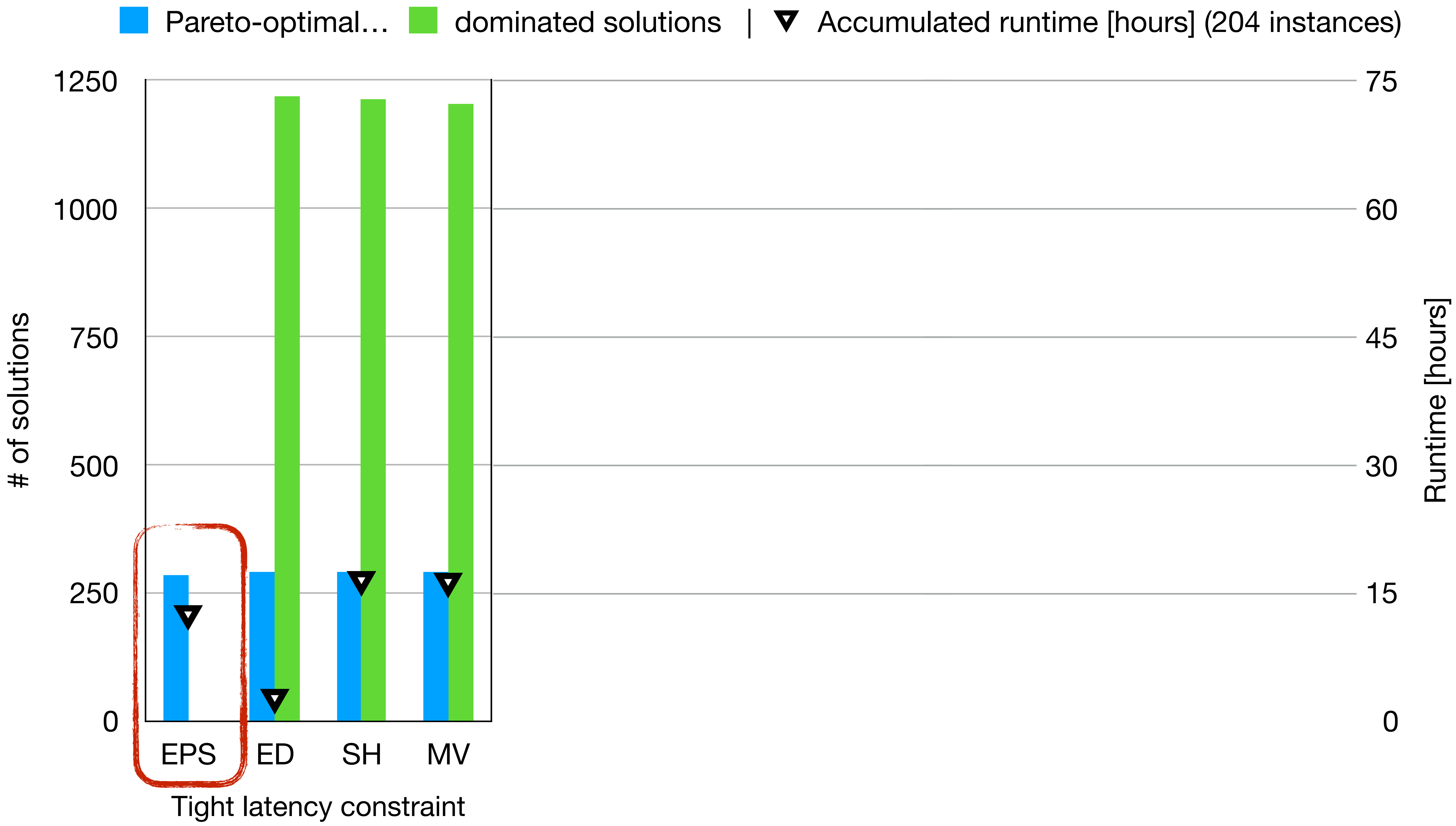
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



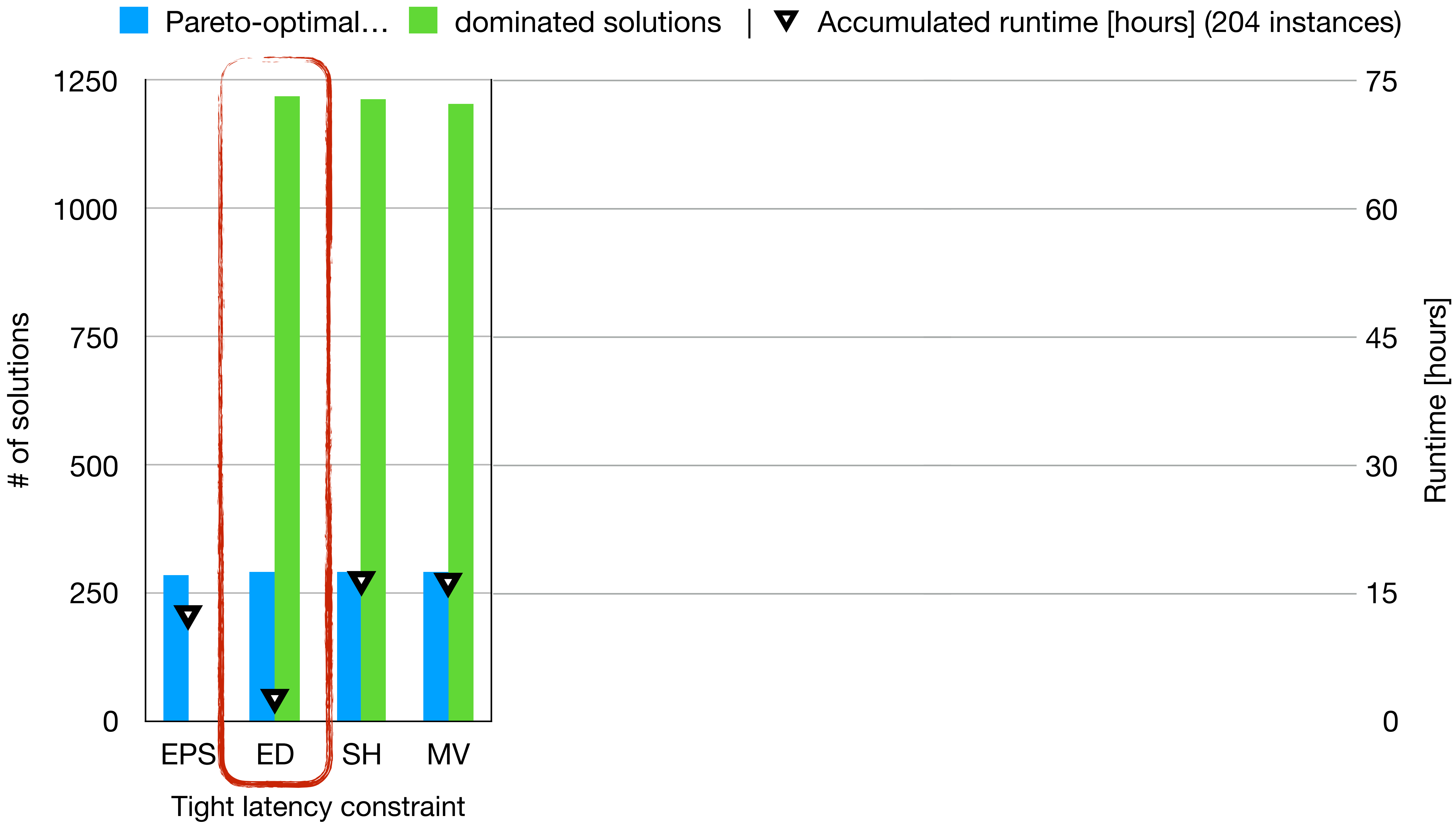
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



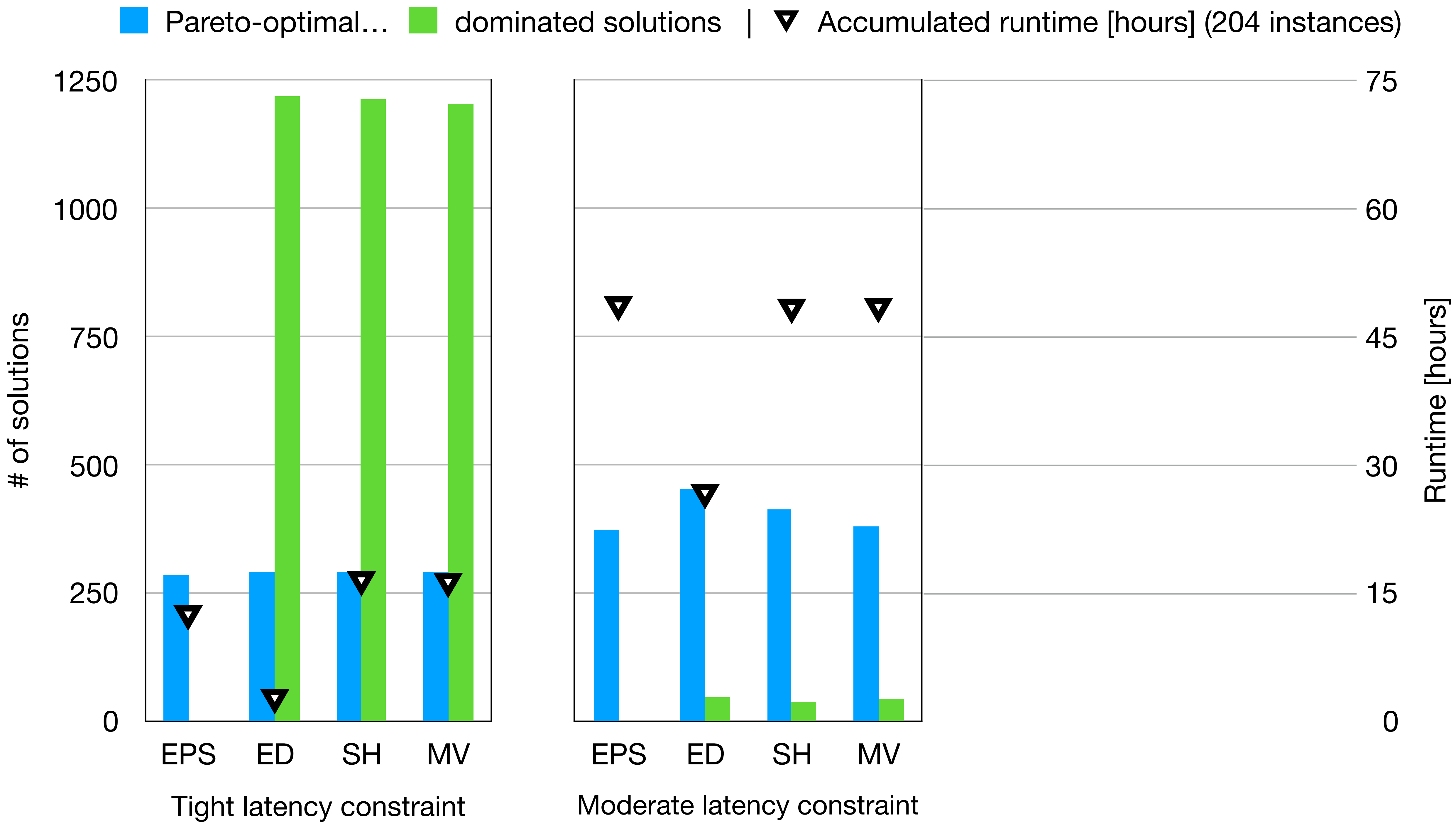
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



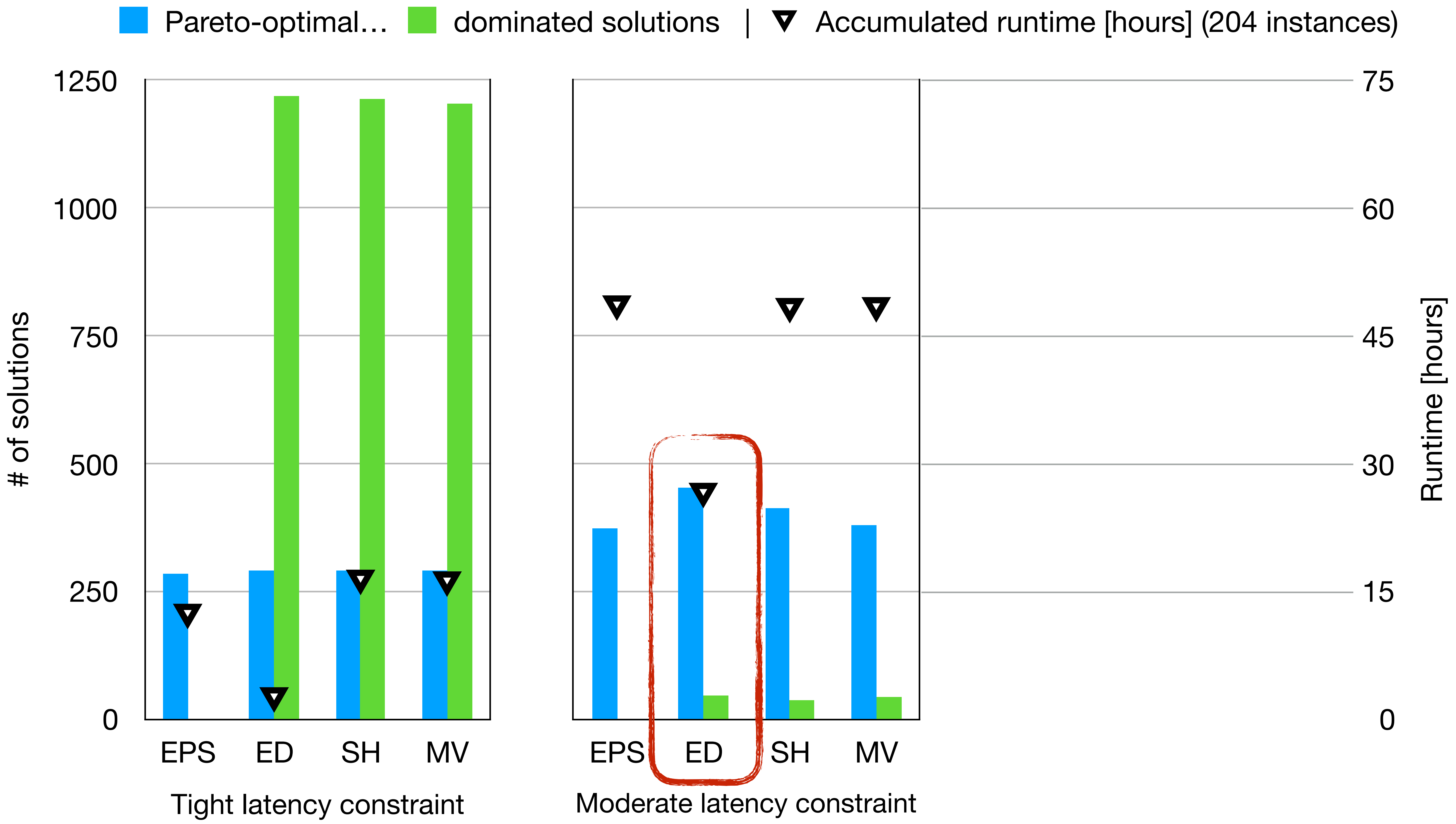
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



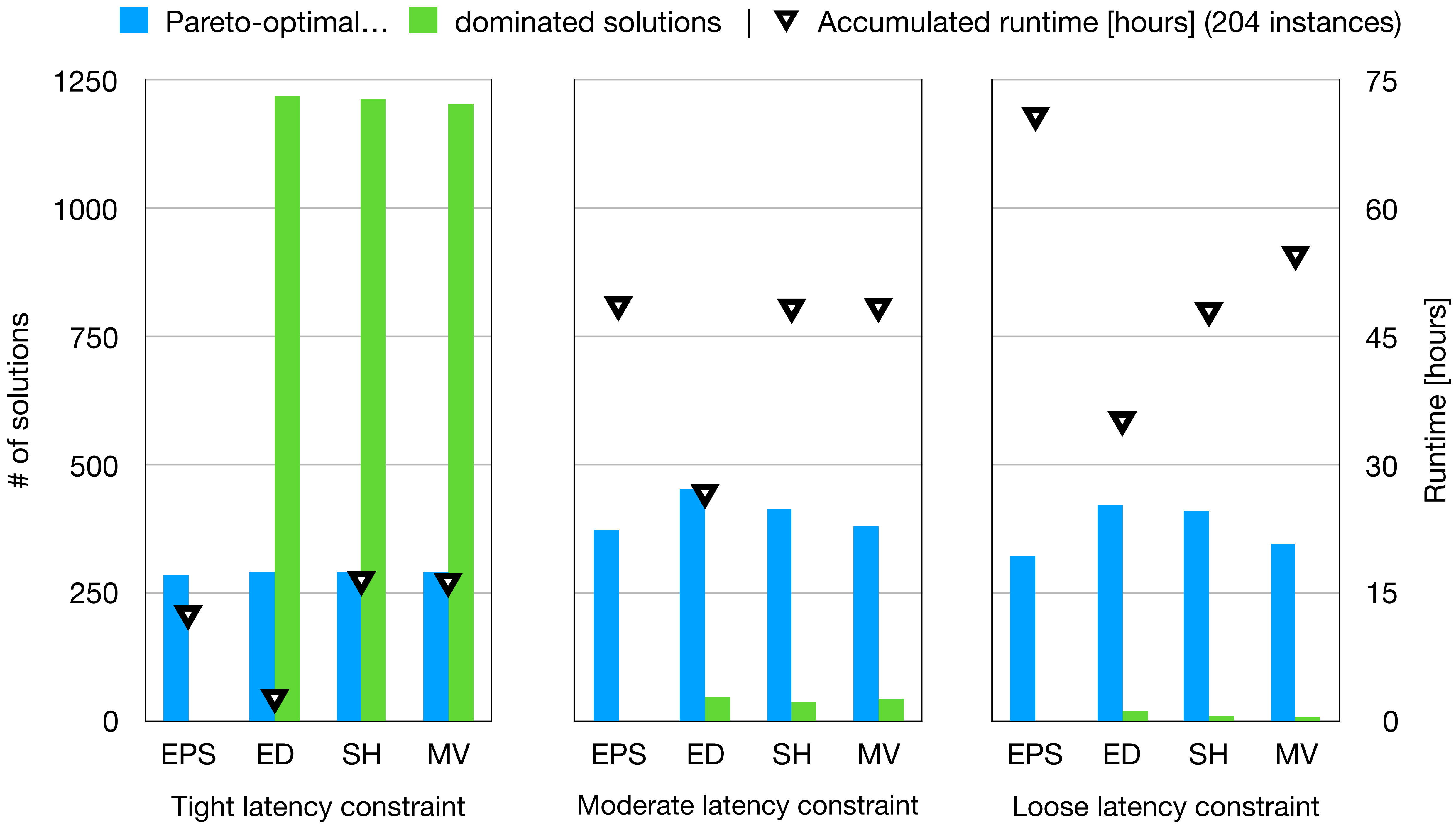
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



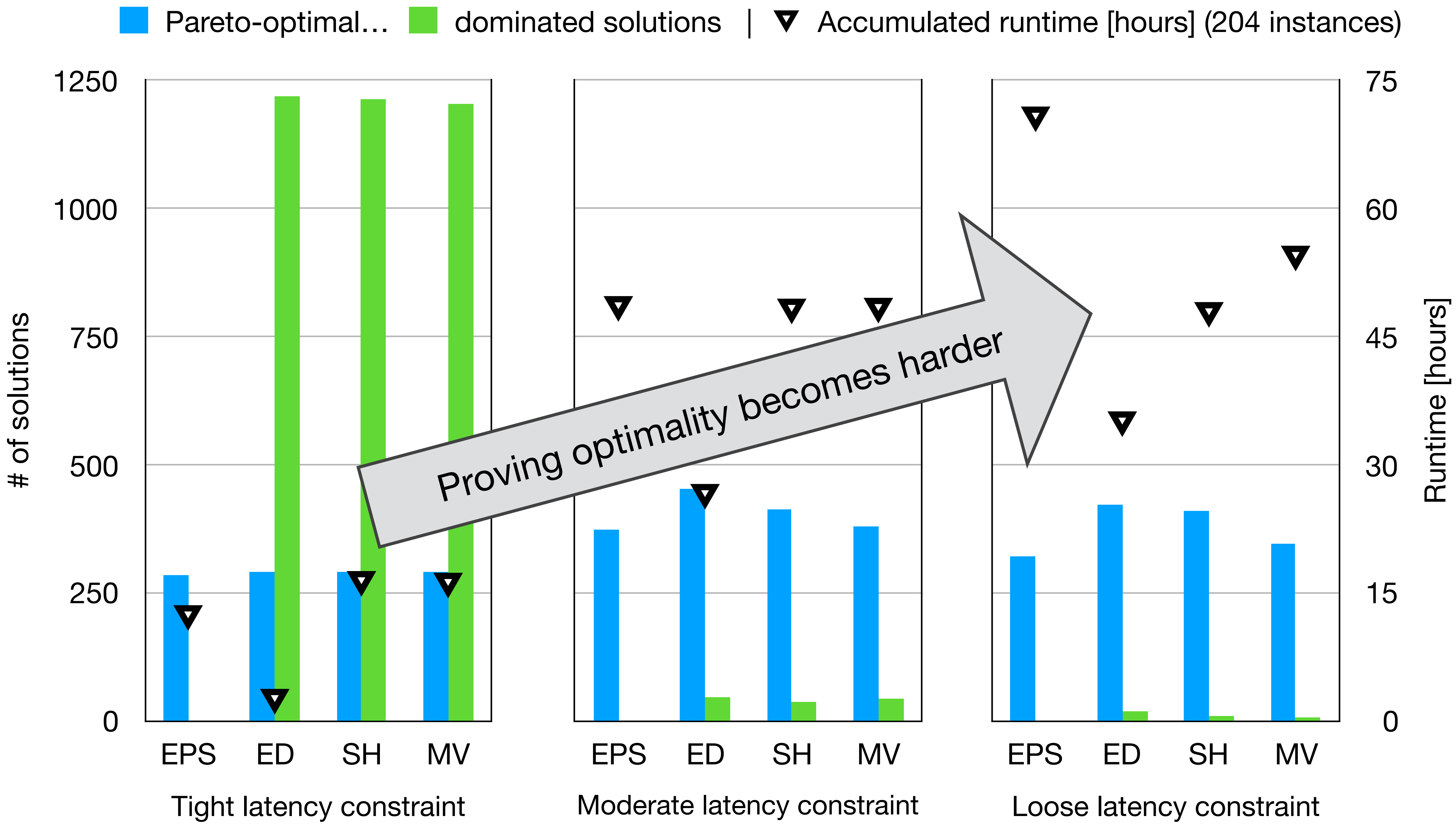
Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



Scheduling Results

- Question: which approach computes the most Pareto-optimal solutions within 6 hours per instance?



Conclusion

- Presented framework to make ILP-based modulo schedulers resource aware

Conclusion

- Presented framework to make ILP-based modulo schedulers resource aware
- ED-formulation + iterative approach is fastest, and computes the most Pareto-optimal solutions

Conclusion

- Presented framework to make ILP-based modulo schedulers resource aware
- ED-formulation + iterative approach is fastest, and computes the most Pareto-optimal solutions
- Outlook: probably too many solutions per instance — how to determine relevant ones?

Thank you!

Check out the **HatScheT** scheduler library

http://uni-kassel.de/go/hatschet

TECHNISCHE UNIVERSITÄT DARMSTADT

Hochschule Fulda
University of Applied Sciences

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

Imperial College London

KIT
Karlsruhe Institute of Technology