Scheduling and Rostering

Marco Kuhlmann & Guido Tack

Lecture 7

The story so far

- **Modelling in Gecode/J**
- **Formal framework for constraint programming**
- **Propagation, global constraints**
- **Search**

The story so far

- **Modelling in Gecode/J**
- **Formal framework for constraint programming**
- **Propagation, global constraints**
- **Search**

complete picture of a cp system!

Advanced topics

Remainder of the course

- **Scheduling and rostering**
- **SAT solving**
- **Finite set constraints**
- **Symmetry breaking**

Remainder of the course

today

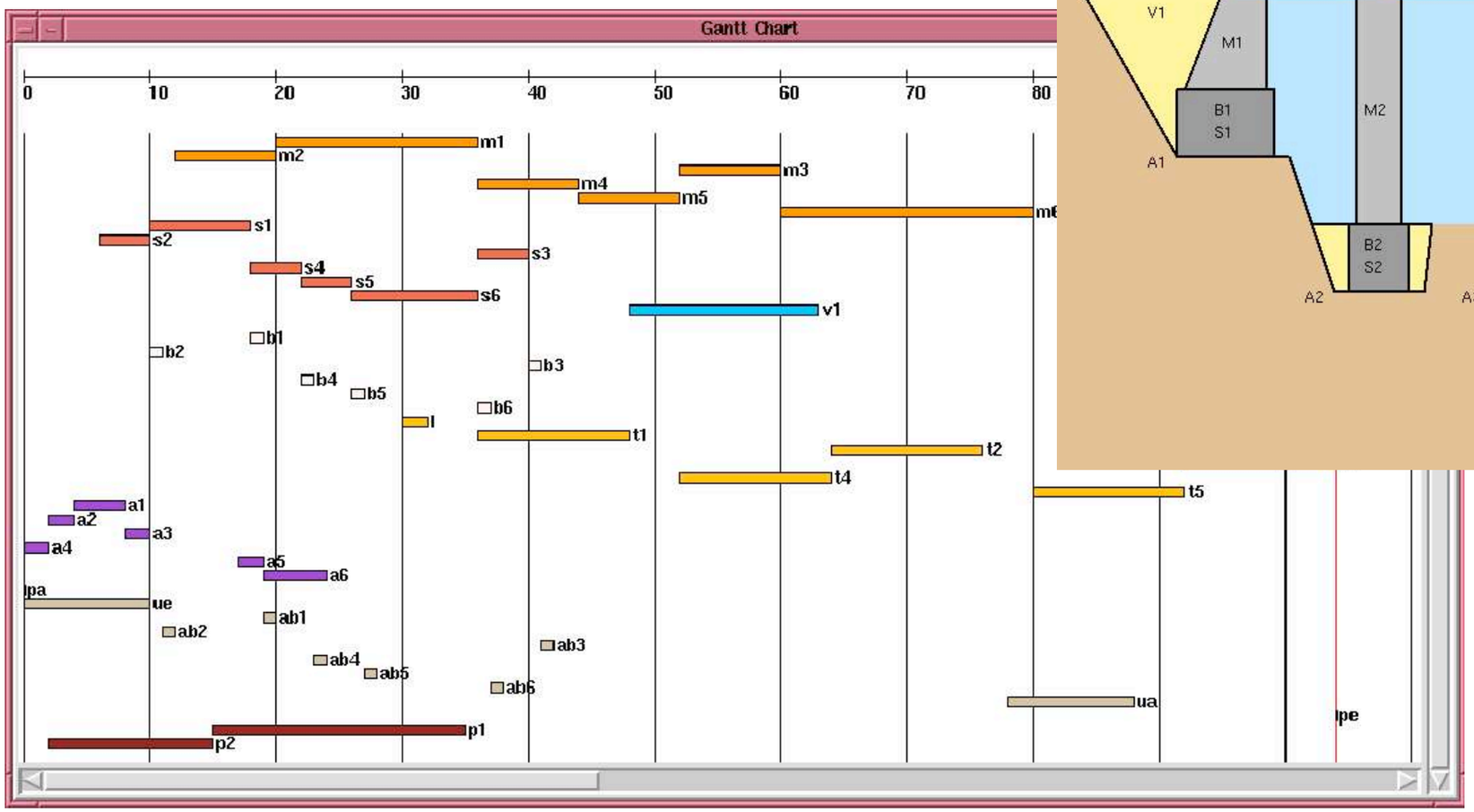
- **Scheduling and rostering**
- **SAT solving**
- **Finite set constraints**
- **Symmetry breaking**

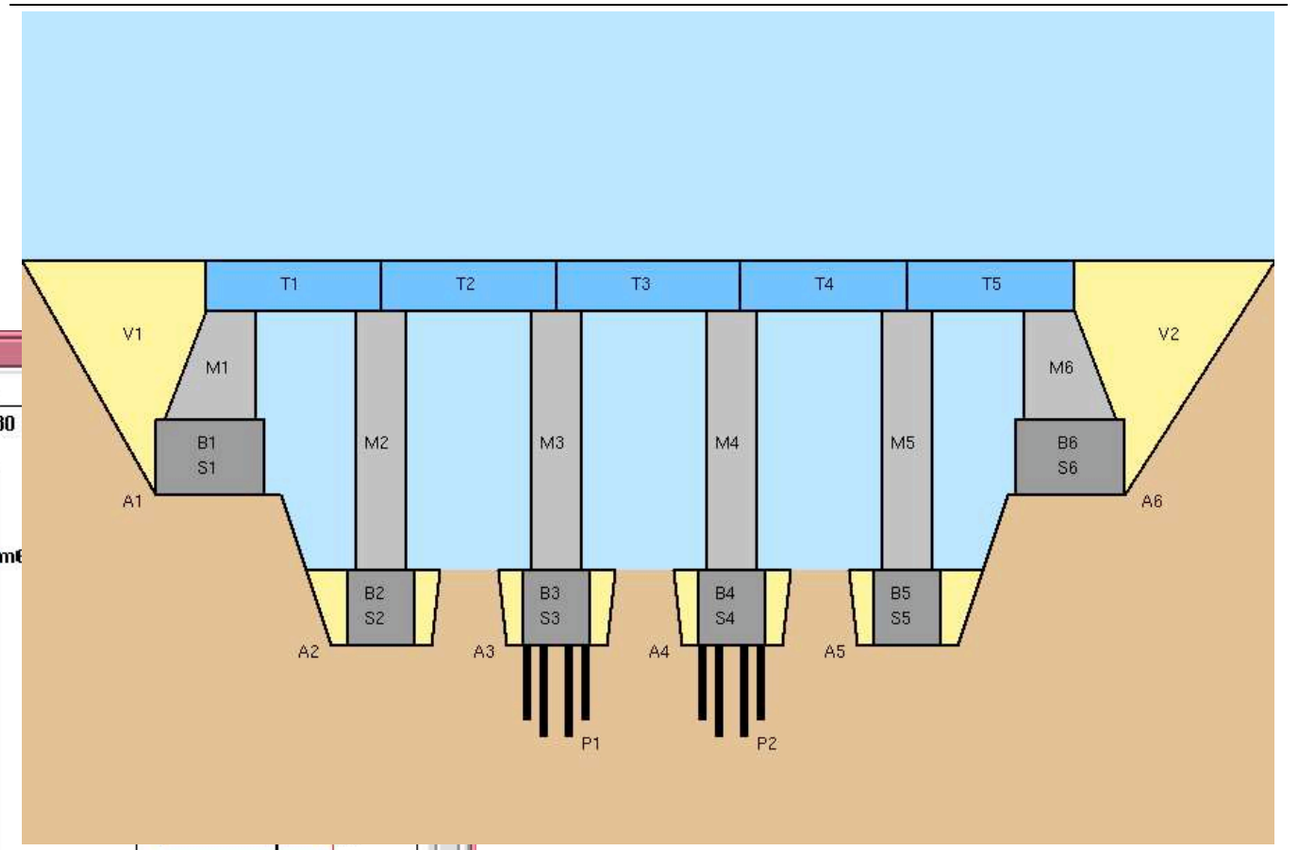
Scheduling

Scheduling

- **Tasks *a* (aka activities)**
  - duration *dur(a)*
  - resource *res(a)*
- **Precedence constraints**
  - determine order among two tasks
- **Resource constraints**
  - e.g. at most one task per resource

Building a Bridge





Application Areas

- Creating **time tables**
- Planning **workflow**
- Scheduling **instruction sequences** in a compiler
- ...
- Overlap with **Operations Research (OR) !**

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$a \textbf{ before } b$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$a \textbf{ before } b$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

similar to temporal relations

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$start(a) + dur(a) \leq start(b)$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$start(a) + dur(a) \leq start(b)$$

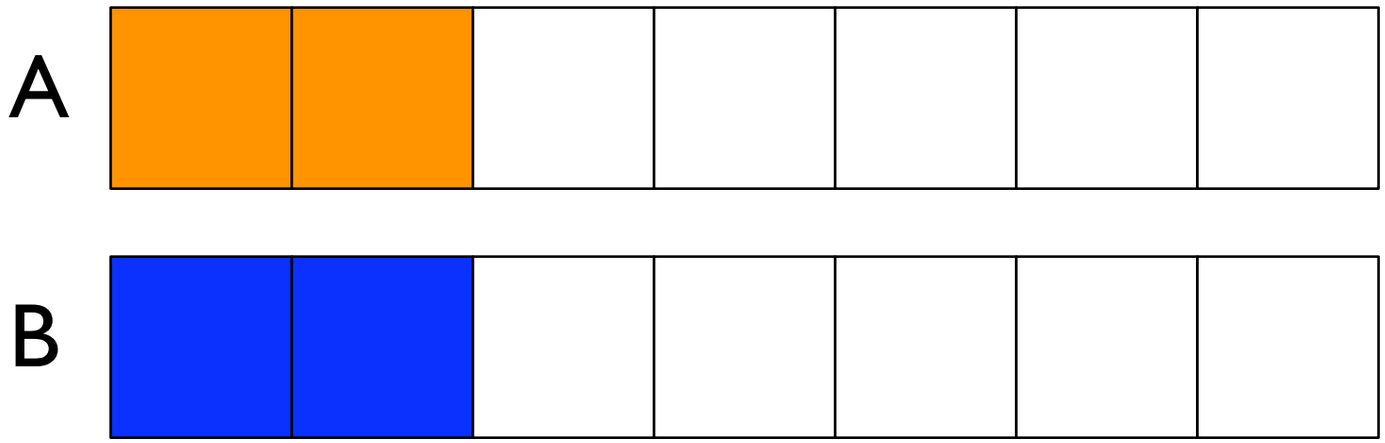
- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

reification

Propagate Precedence

A before B



$start(A) \in \{0,\ldots,5\}$ $dur(A) = 2$

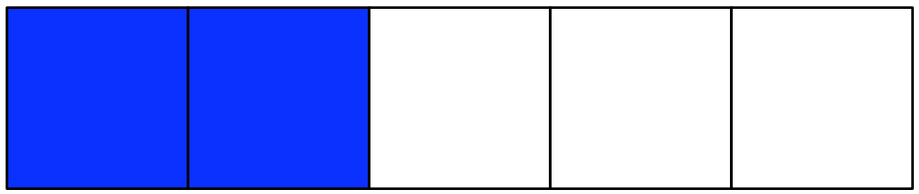
$start(B) \in \{0,\ldots,5\}$ $dur(B) = 2$

Propagate Precedence

A before B

A

B



$start(A) \in \{0,\ldots,3\}$ $dur(A) = 2$

$start(B) \in \{2,\ldots,5\}$ $dur(B) = 2$

So what's new?

- **Concrete start and end times**
- **Optimization**

  (e.g. find earliest completion time, minimal makespan,...)
- **This makes the problem hard**

Classes of problems I

- **Resource type**

  disjunctive (at most one task at a time)

  cumulative (fixed capacity per resource)

- **Task type**

  non-preemptive (not interruptible)

  preemptive

Classes of problems II

- **Optimization: minimize**

  makespan (latest end time of any task)

  number of late jobs (that miss their due date)

  ...

- **Give more weight to more important jobs**

Special case we discuss

- **disjunctive, non-preemptive**
- **cumulative (briefly)**

Comprehensive discussion:

Baptiste, Le Pape, Nuijten. *Constraint-based Scheduling*. Kluwer, 2001.

Again: Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$start(a) + dur(a) \leq start(b)$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

reification

Think global

- **Model employs local view:**
  - constraints on pairs of tasks
  - $O(n^2)$ propagators for n tasks
- **Global view:**
  - order all tasks on one resource
  - employ smart global propagator

Ordering Tasks: Serialization

- Consider all tasks on one resource
- Deduce their order as much as possible

Ordering Tasks: Serialization

- Consider all tasks on one resource
- Deduce their order as much as possible
- Propagators:
  - **Timetabling:** look at free/used time slots
  - **Edge-finding:** which task first/last?
  - **Not-first / not-last**

Special case

- Consider disjunctive, non-preemptive scheduling where the duration is 1 for all tasks
- Do you know a good propagator for serialization?

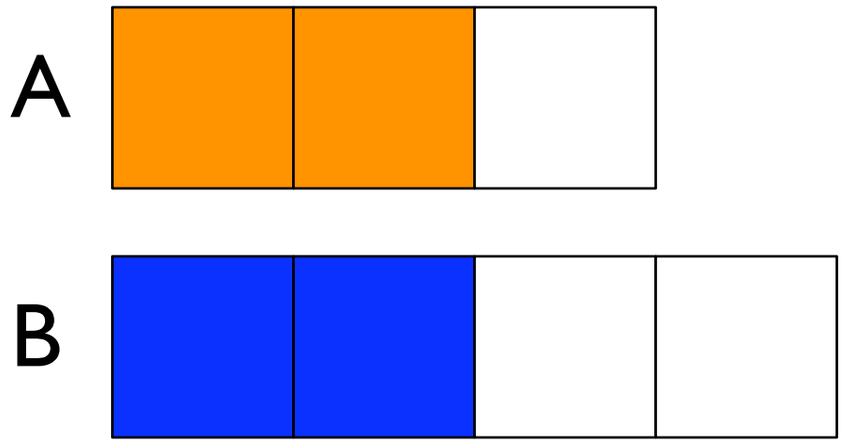
Timetable propagation

- **Timetable:**

  data structure that records per resource where some task definitively uses the resource

- **Propagate**
  - from tasks to timetable
  - from timetable to tasks

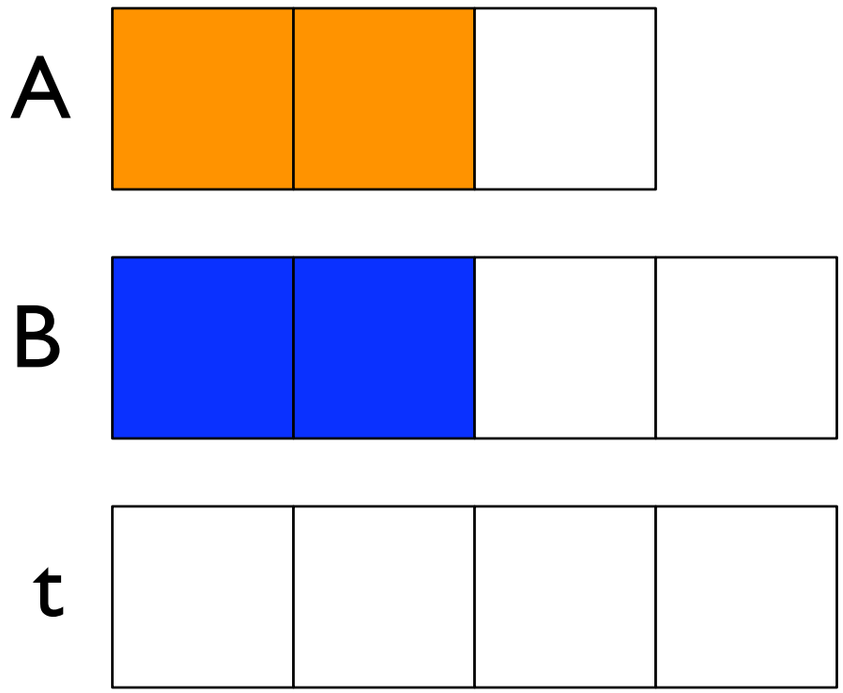
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{0,1,2\}$ $dur(B) = 2$

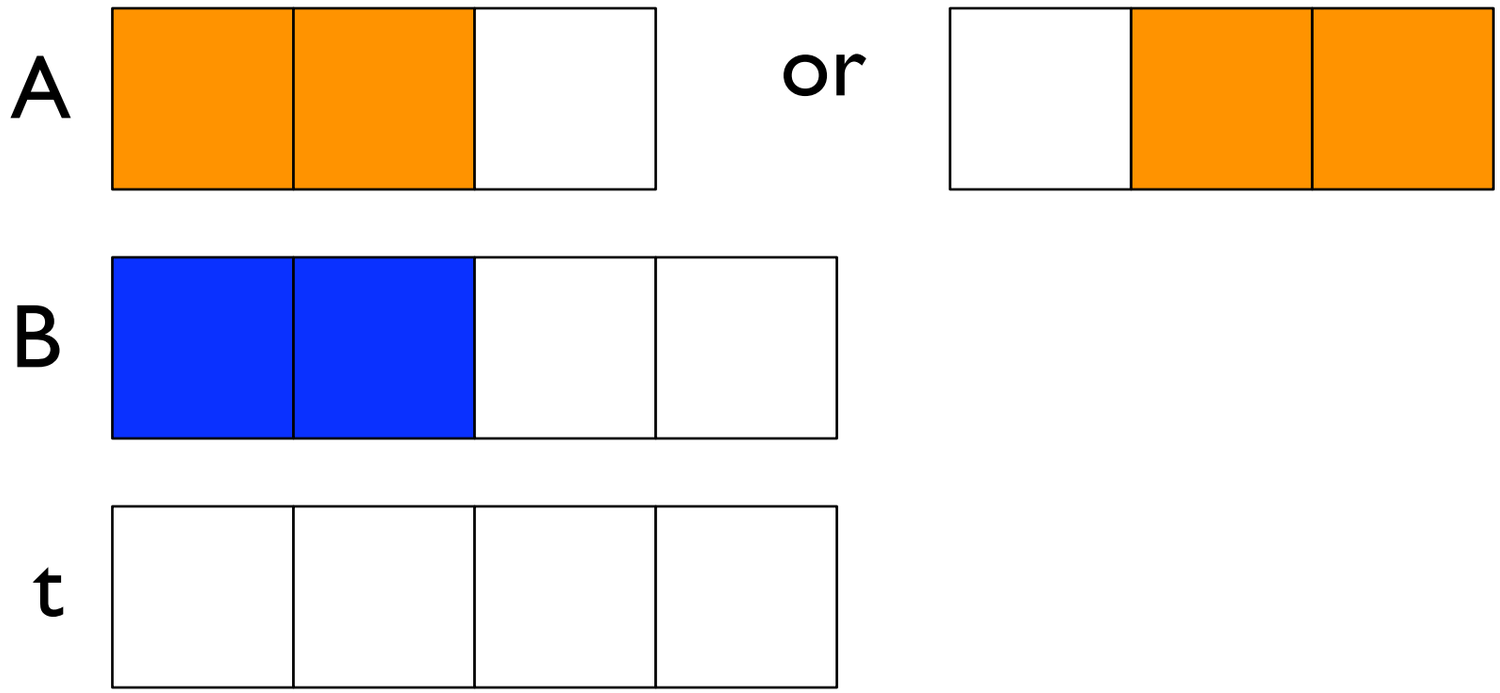
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{0,1,2\}$ $dur(B) = 2$

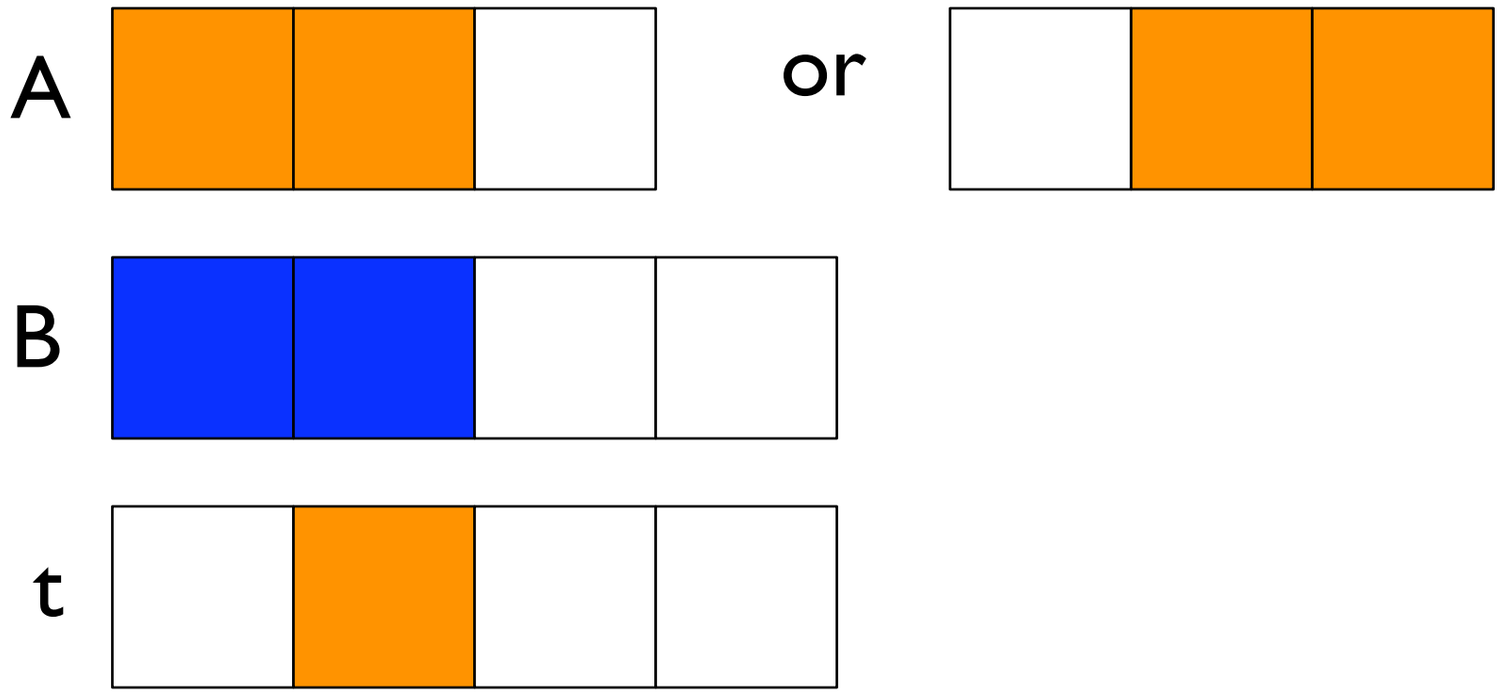
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{0,1,2\}$ $dur(B) = 2$

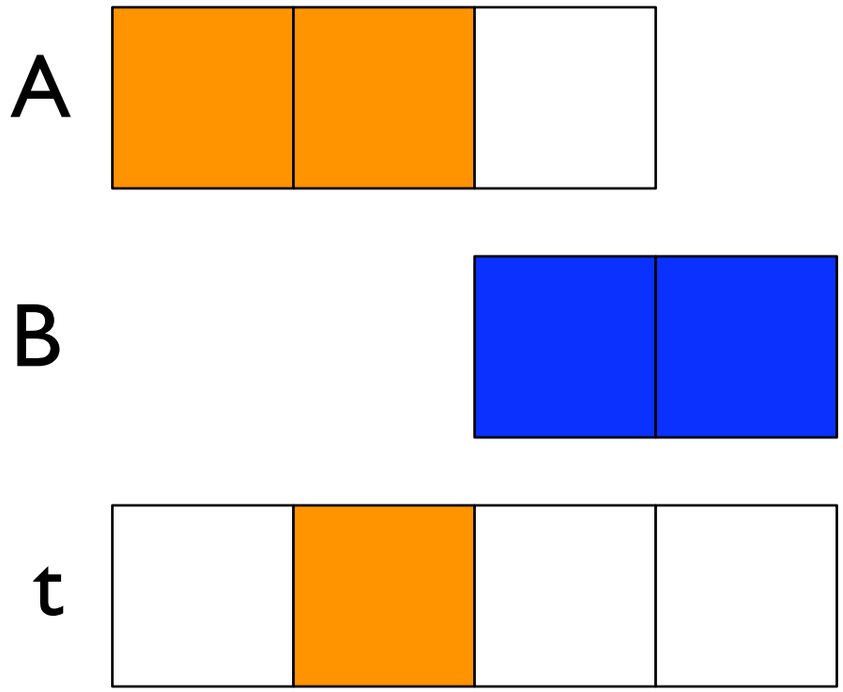
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{0,1,2\}$ $dur(B) = 2$

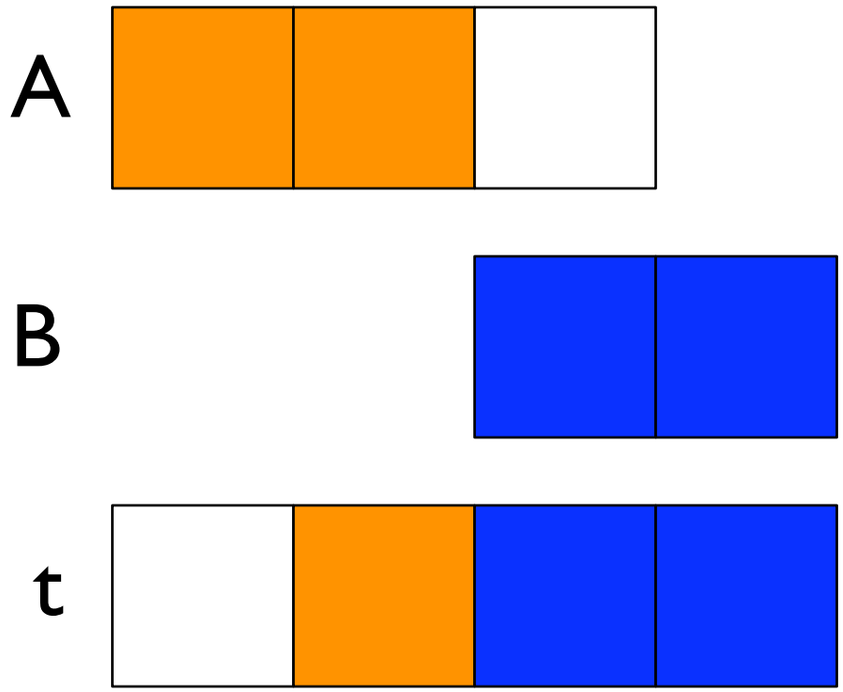
Example: Timetable



$start(A) \in \{0, 1\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

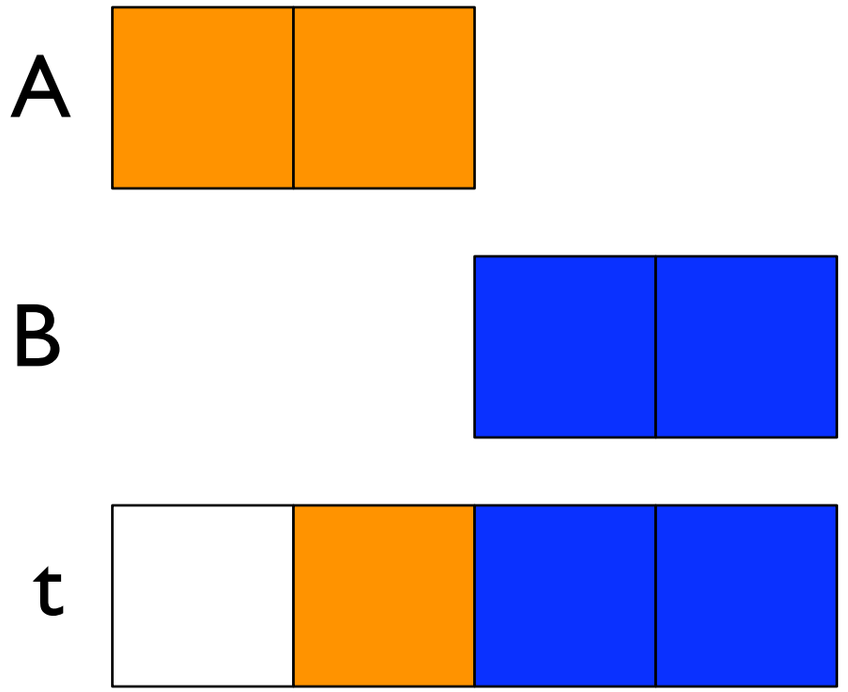
Example: Timetable



$start(A) \in \{0, 1\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

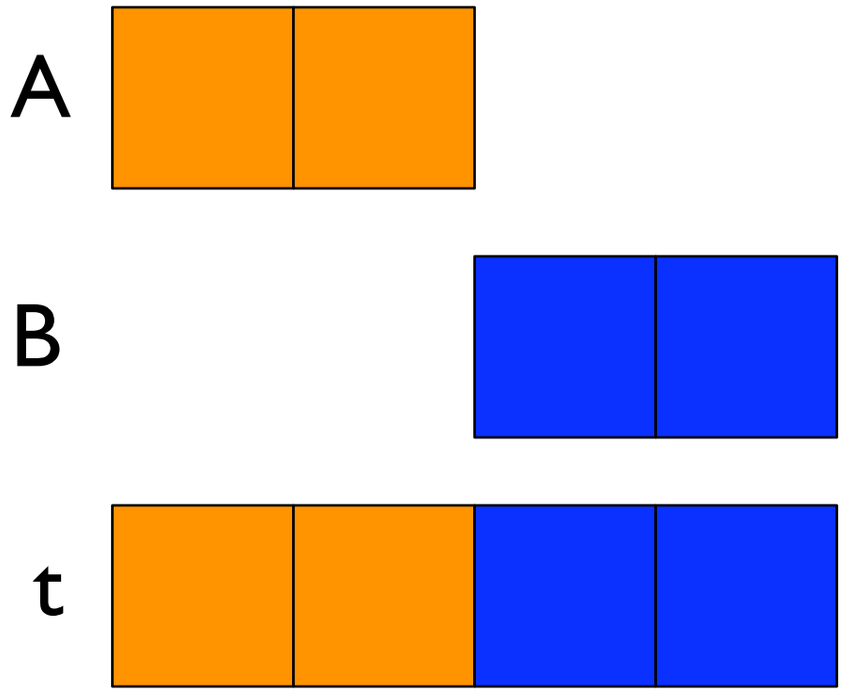
Example: Timetable



$start(A) \in \{0\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

Example: Timetable



$start(A) \in \{0\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

Edge Finding

- **Time tabling**

  is often weaker than reification

- **But important idea:**

  record when resource is used

- **Edge finding**
  - more general scheme to propagate order between tasks
  - well-known, important OR algorithm ported to CP

Edge finding: Idea

- Given a set *O* of tasks and *T*∈*O*

  find out whether T must execute before (or after) all tasks in *O*-{*T*}

Edge finding: Idea

- Given a set O of tasks and $T \in O$

  find out whether T must execute before (or after) all tasks in $O-\{T\}$

- Can be done in time $O(n^2)$ for n tasks

Edge finding: Idea

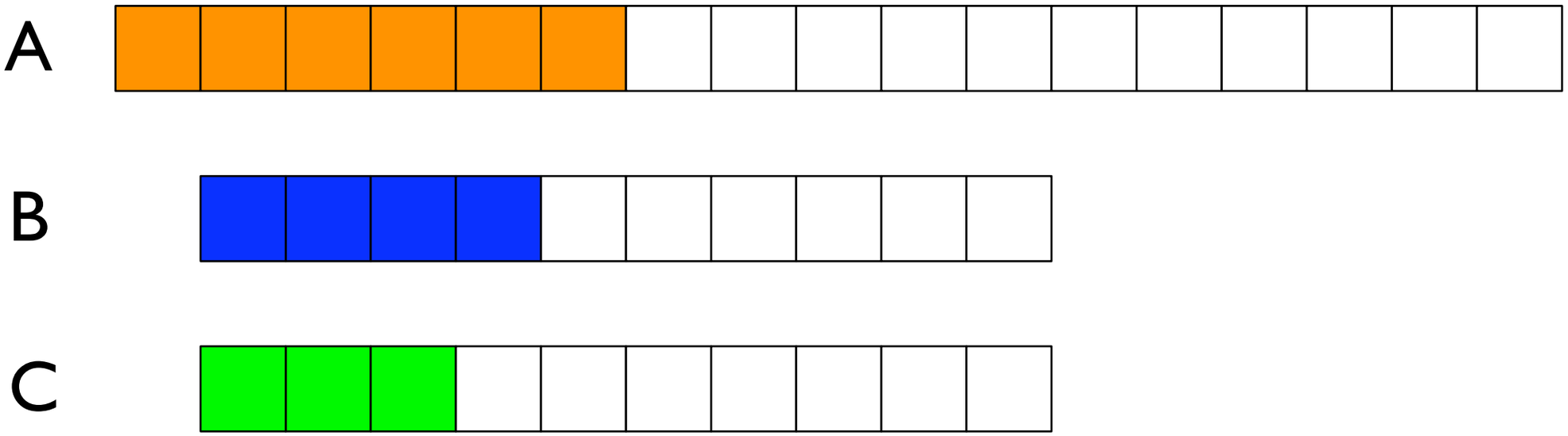
- Given a set *O* of tasks and *T*∈*O*

  find out whether T must execute before (or after) all tasks in *O*-{*T*}

- Can be done in time $O(n^2)$ for *n* tasks

- Dually: not-first, not-last

Example: Edge finding

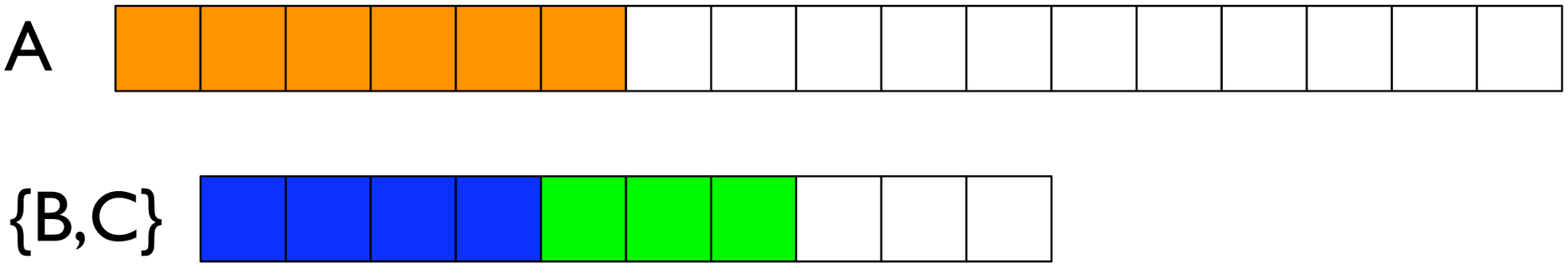


$start(A) \in \{0,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Example: Edge finding

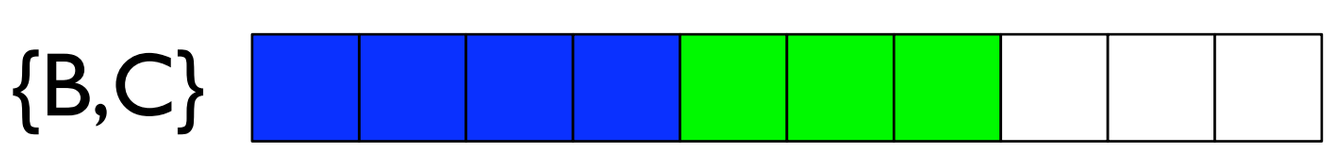


$start(A) \in \{0,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Example: Edge finding



consider {B,C}:
A must be last!

$start(A) \in \{0,\ldots,11\}$ $\quad dur(A) = 6$

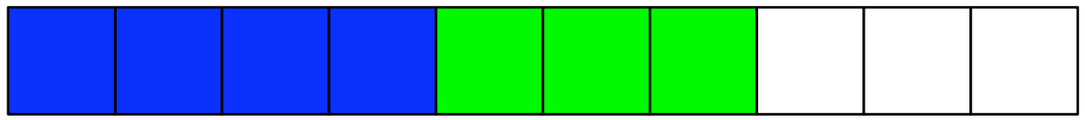
$start(B) \in \{1,\ldots,7\}$ $\quad dur(B) = 4$

$start(C) \in \{1,\ldots,8\}$ $\quad dur(C) = 3$

Example: Edge finding

A

{B,C}



consider {B,C}:
A must be last!

$start(A) \in \{8,\ldots,11\}$ $\quad dur(A) = 6$

$start(B) \in \{1,\ldots,7\}$ $\quad dur(B) = 4$

$start(C) \in \{1,\ldots,8\}$ $\quad dur(C) = 3$

Example: Edge finding

A

B

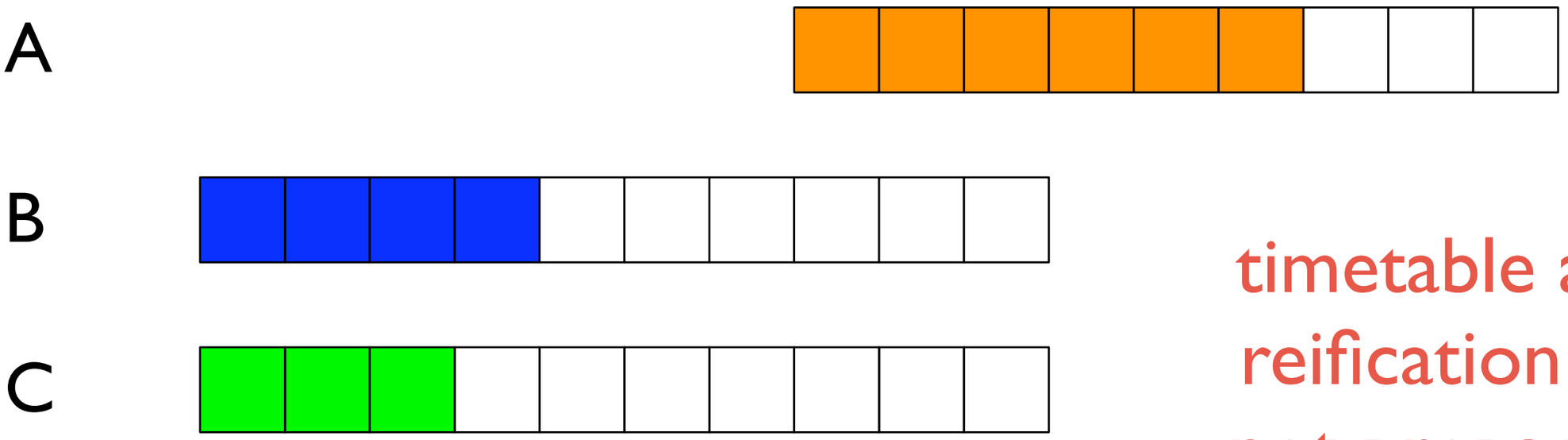
C

$start(A) \in \{8,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Example: Edge finding



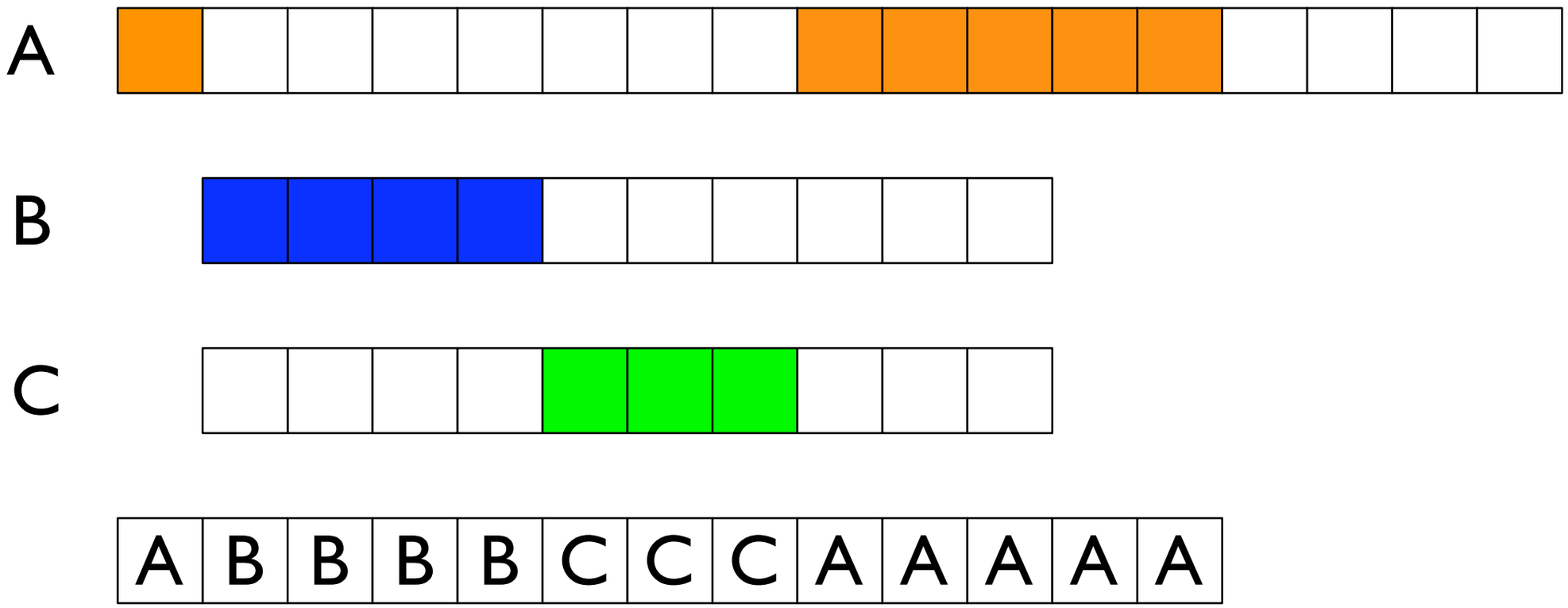
timetable and reification do not propagate anything!

$start(A) \in \{8,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Edge finding: algorithmic idea



Edge finding: algorithmic idea

A	B	B	B	B	C	C	C	A	A	A	A	A

- **For task** A:

$$\mathrm{start}(A) + \mathrm{dur}(A) + \sum_{a \in \{B,C\}} \mathrm{residual}(0, a) > \mathrm{end}(C)$$

where residual(*t,a*) is the residual processing time on the JPS of task *a* at time *t*

- **Consequence:**

A comes after {B,C}

Edge finding: consistency

- **Edge finding computes**

  "the smallest earliest start time for each activity A_i, assuming all other activities are interruptible"

- **Sometimes stronger** than disjunction by reification
- **Sometimes weaker**
- **Often combination is used**

Branching

- **General idea of any heuristic:**

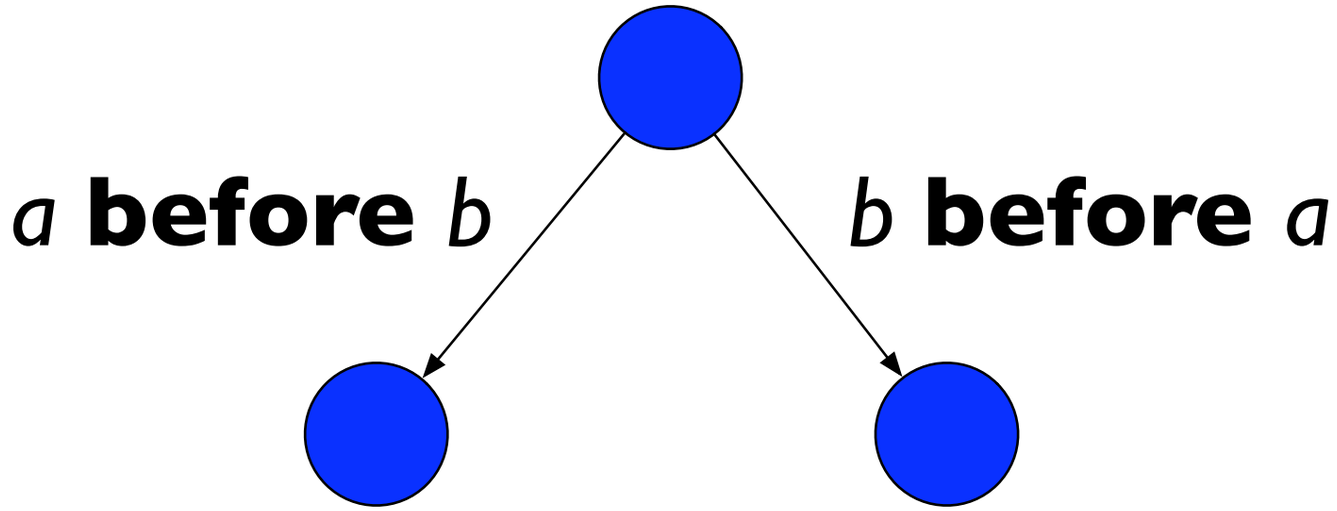
  determine *critical* variables early!

- A critical variable is one that makes a big difference if determined

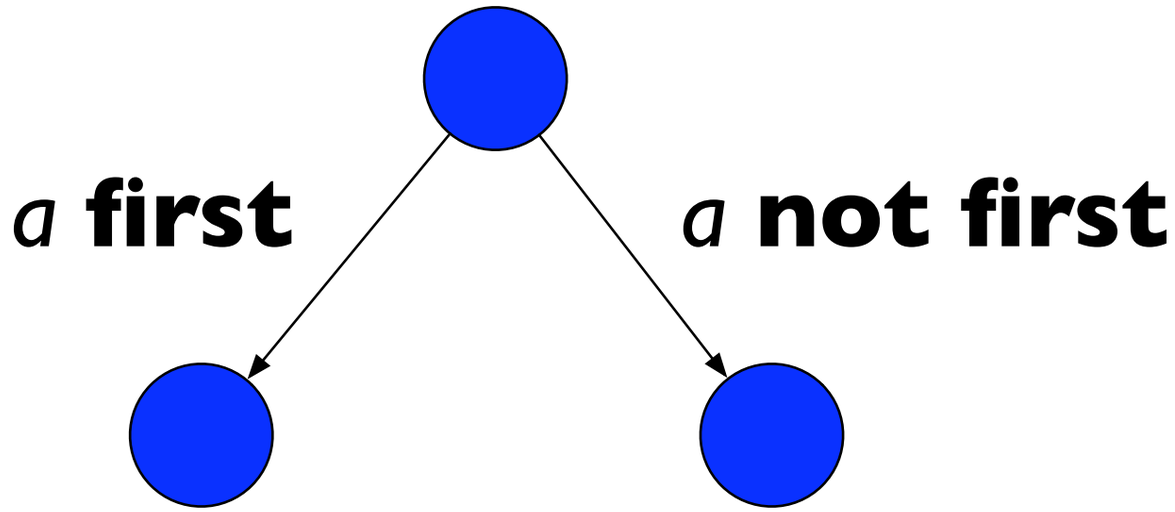
Branching

- **Heuristic:** establish order among tasks
  - which resource to choose
  - guess first task on resource
- **After ordering:** assign start times
  - solution exists, so no branching required
  - after assigning, propagate precedence constraints

Branching

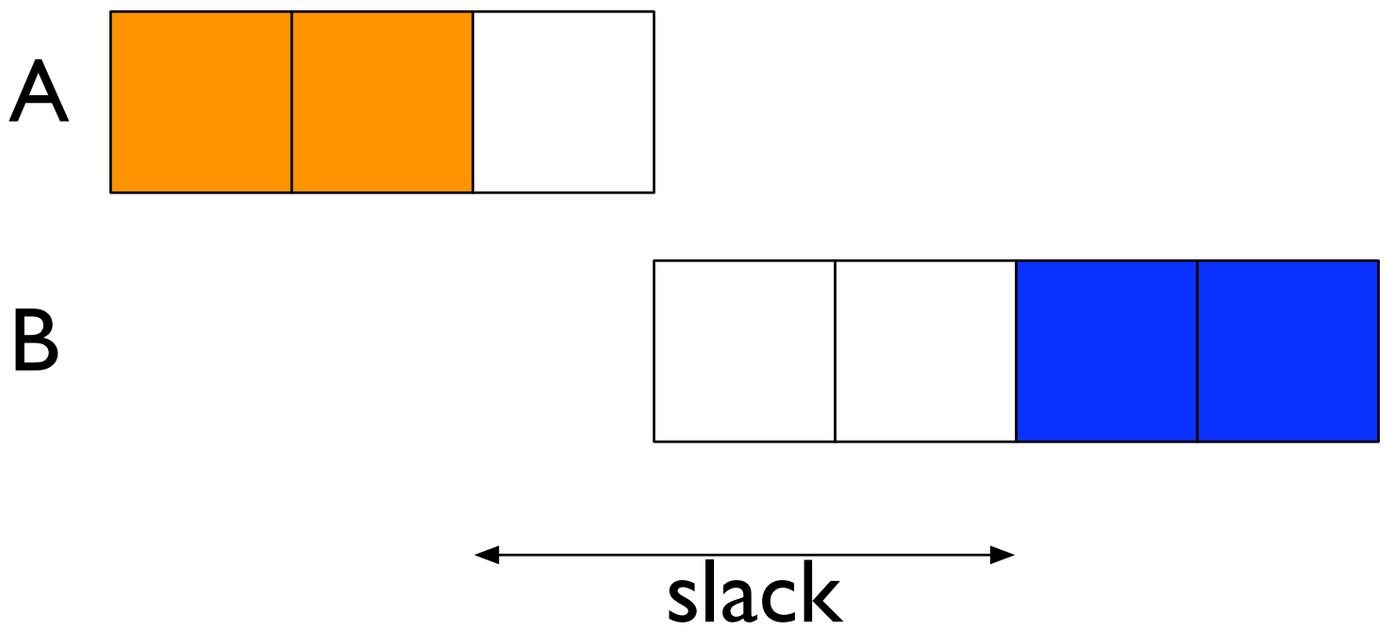


Branching



Branching: Slack

- **How to choose a?**
- **Good heuristic:** minimum slack



Branching and propagation

- **Necessary information:**

  partial order of tasks

- **Cooperate!**

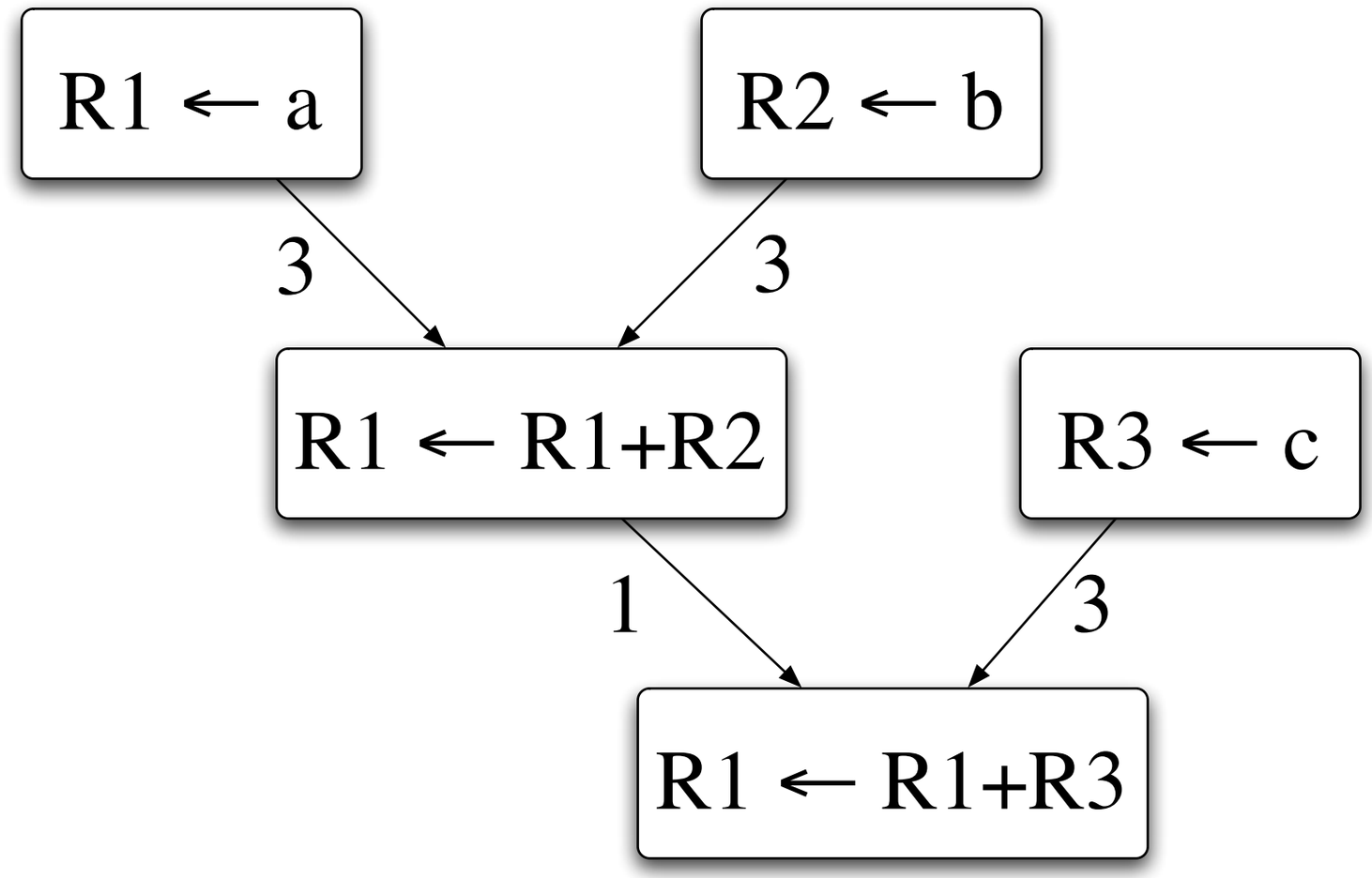
  - share data structures between branching and propagators

Example: Instruction Scheduling

- **Optimize machine code**
- **Goal:** minimum length instruction schedule
- **Important step** for improving performance of object code generated by a complier

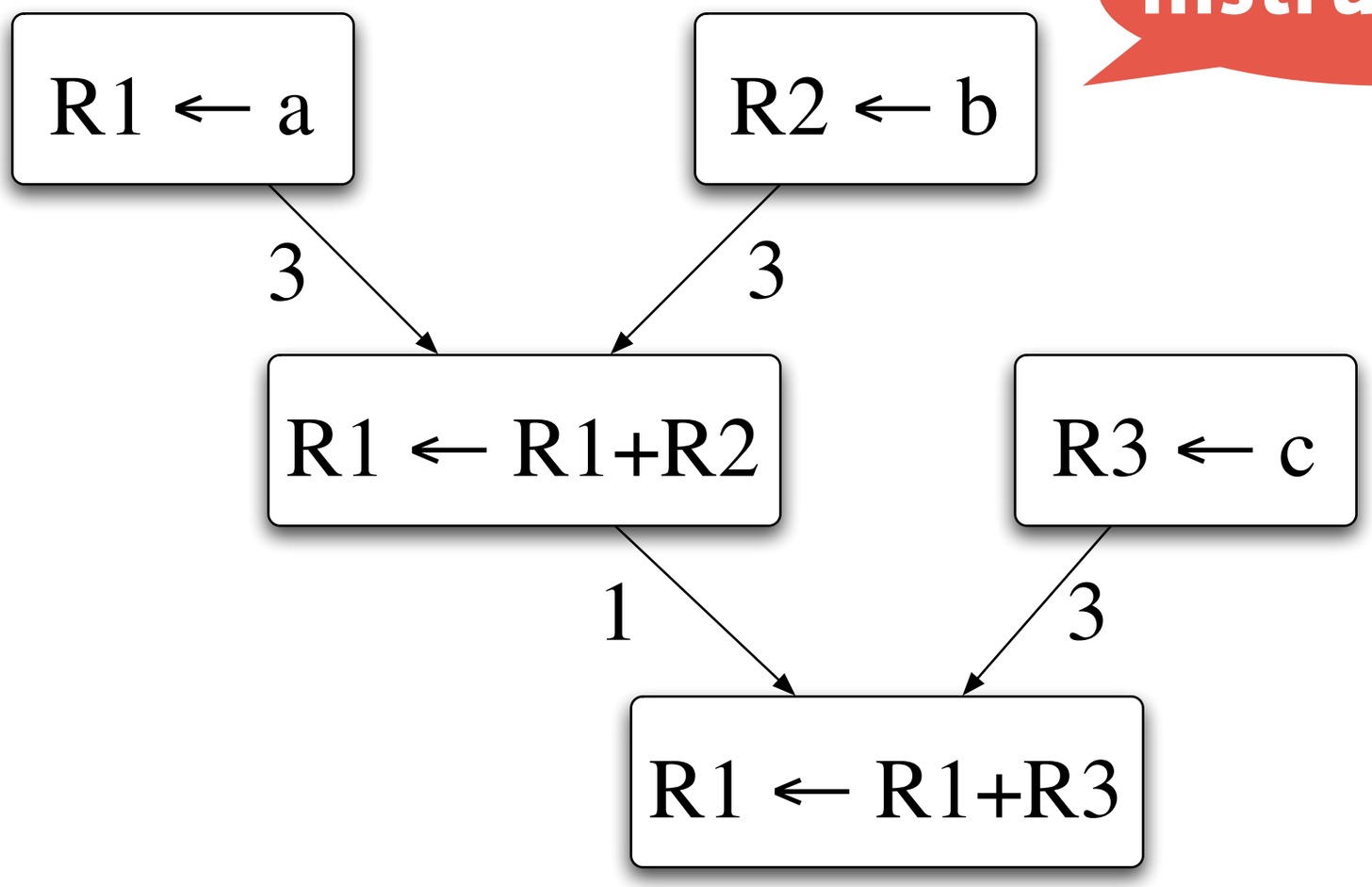
Best paper CP 2001 (Peter van Beek and Kent Wilken)

Example: Instruction Scheduling

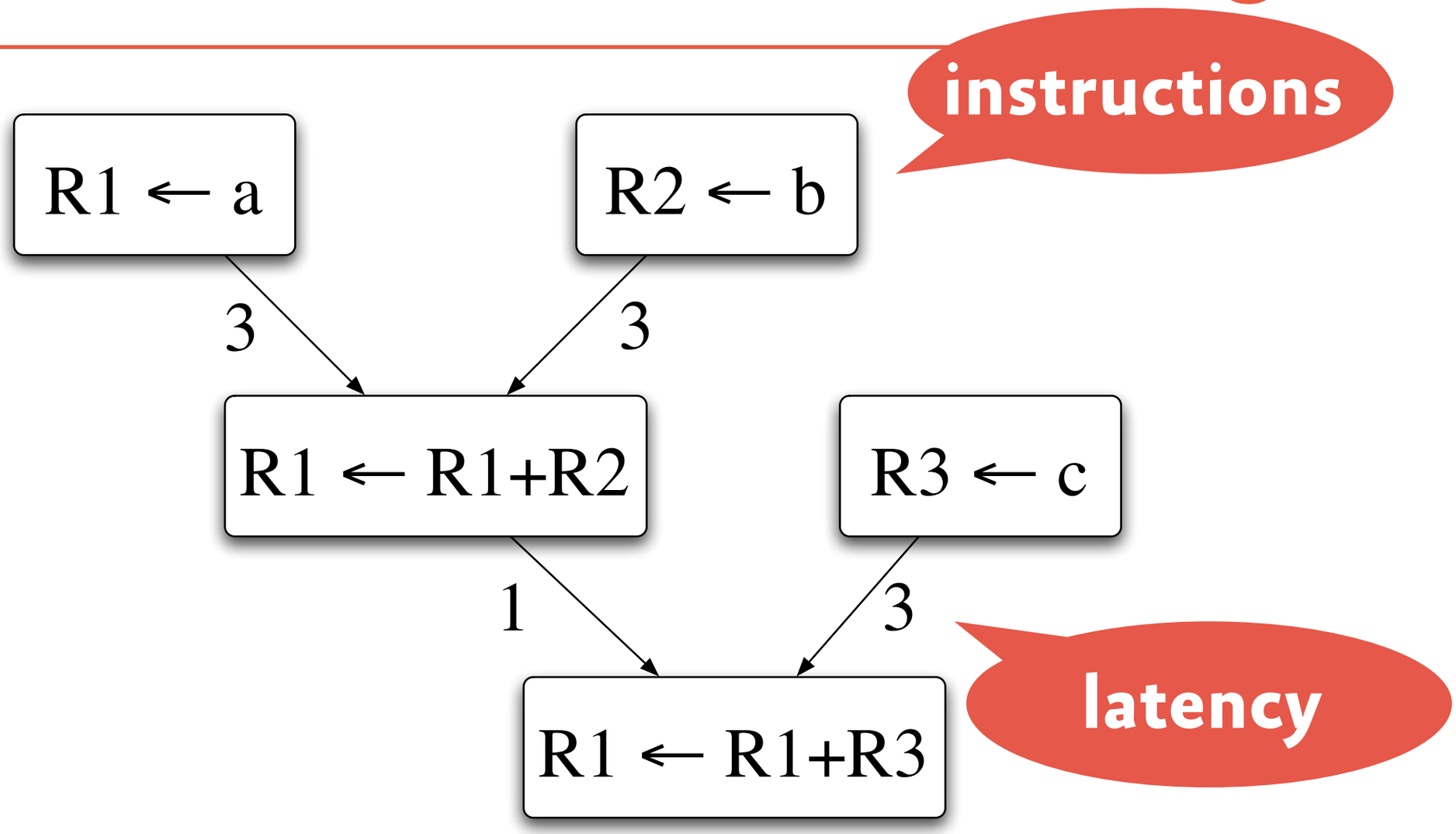


Example: Instruction Scheduling

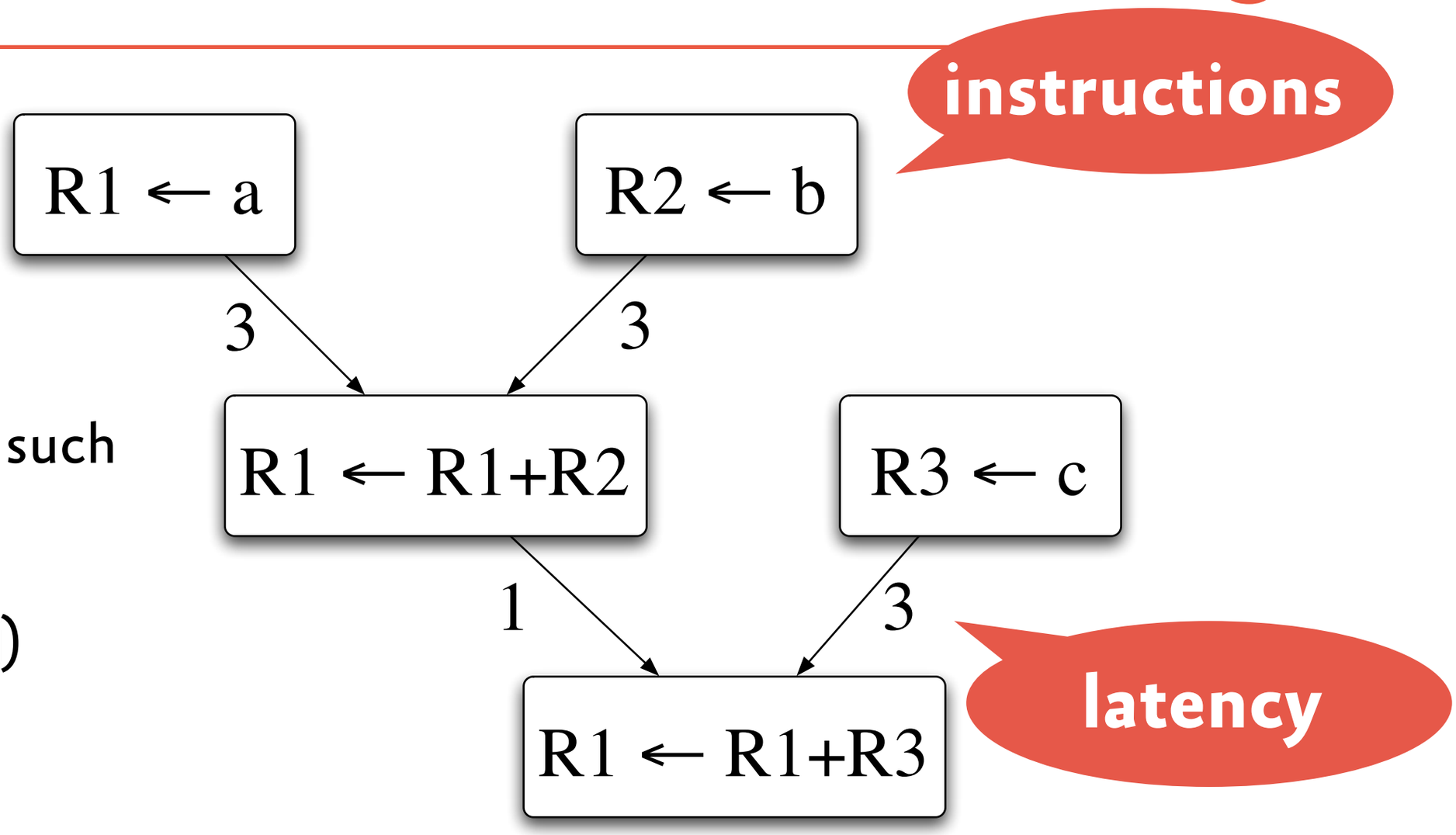
instructions



Example: Instruction Scheduling



Example: Instruction Scheduling



Find **issue time** s(i) such that

1. i≠j implies s(i)≠s(j)

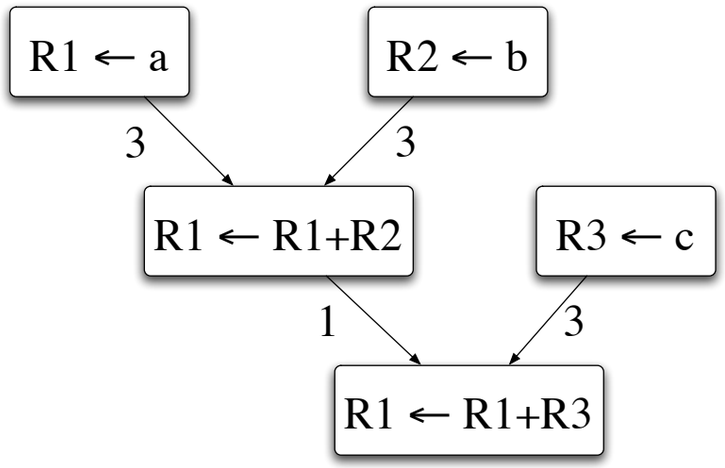
2. s(i) + l(i,j) ≤ s(j)

Minimize max s(i)

Example: Instruction Scheduling

- Constraints:
  all s(i) must be distinct
- Precedence constraints for latency
- Plus special case of edge finding

Example: Instruction Scheduling



Non-optimal:

R1←a
R2←b
nop
nop
R1←R1+R2
R3←c
nop
nop
R1←R1+R3

Optimal:

R1←a
R2←b
R3←c
nop
R1←R1+R2
R1←R1+R3

Results

- **Built into gcc** for experiments
- **Tested on SPEC95 FP** benchmark
- **Large basic blocks** (up to 1000 instructions)
- **Optimally solved**
- **Far better than ILP approach (20 times faster)**

Cumulative Scheduling

- Each resource *R* has a capacity *cap(R)*
- Each task T on R uses amount *use(R)*
- Tasks can overlap but never exceed capacity

Cumulative: Disjunctive Propagation

- For tasks *A* and *B* on resource *R*:

  use(A) + use(B) ≤ cap(R)

  or *start(A) + dur(A) ≤ start(B)*

  or *start(B) + dur(B) ≤ start(A)*

Cumulative

- Techniques from disjunctive scheduling carry over to cumulative scheduling
- Generalized timetabling, edge-finding, not-first/not-last

Geometric Interpretation

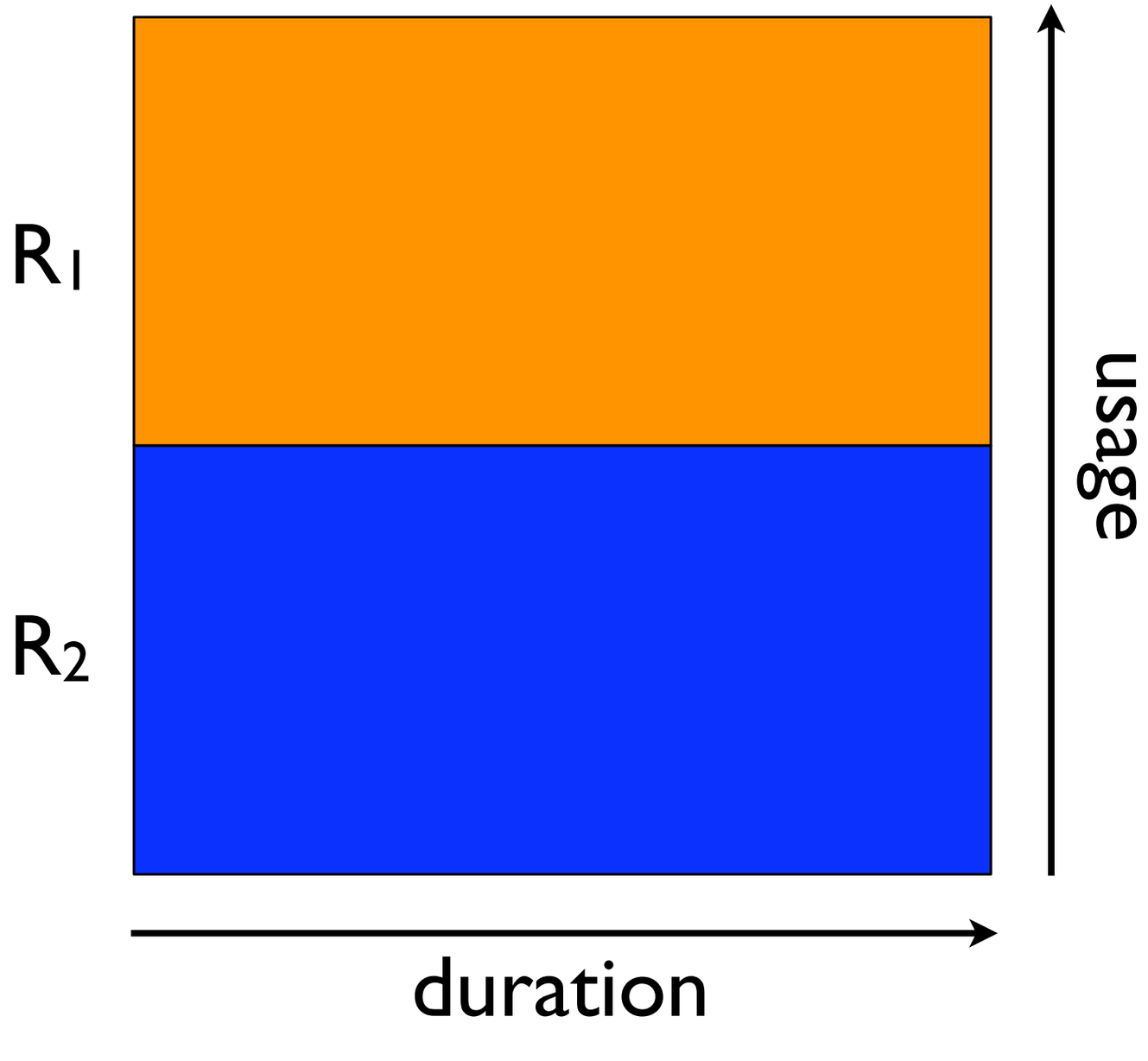
- **Task is rectangle**

  dimension *dur(T)* × *use(T)*

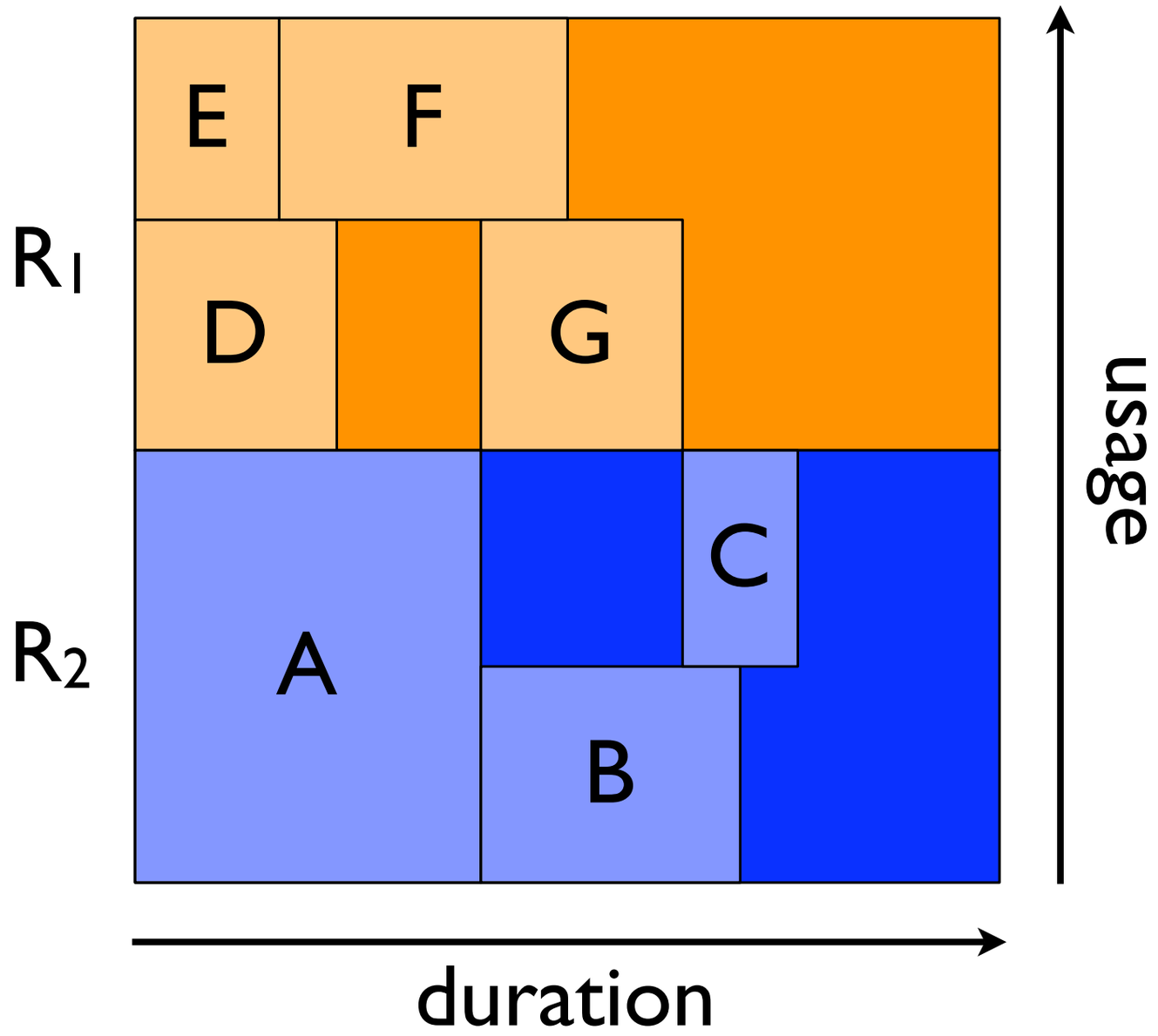
  placed at x-coordinate *start(T)*

- **Resources are rectangles**
  - enclose task-rectangles
  - rectangles never exceed *y*-cordinates

Geometric Interpretation

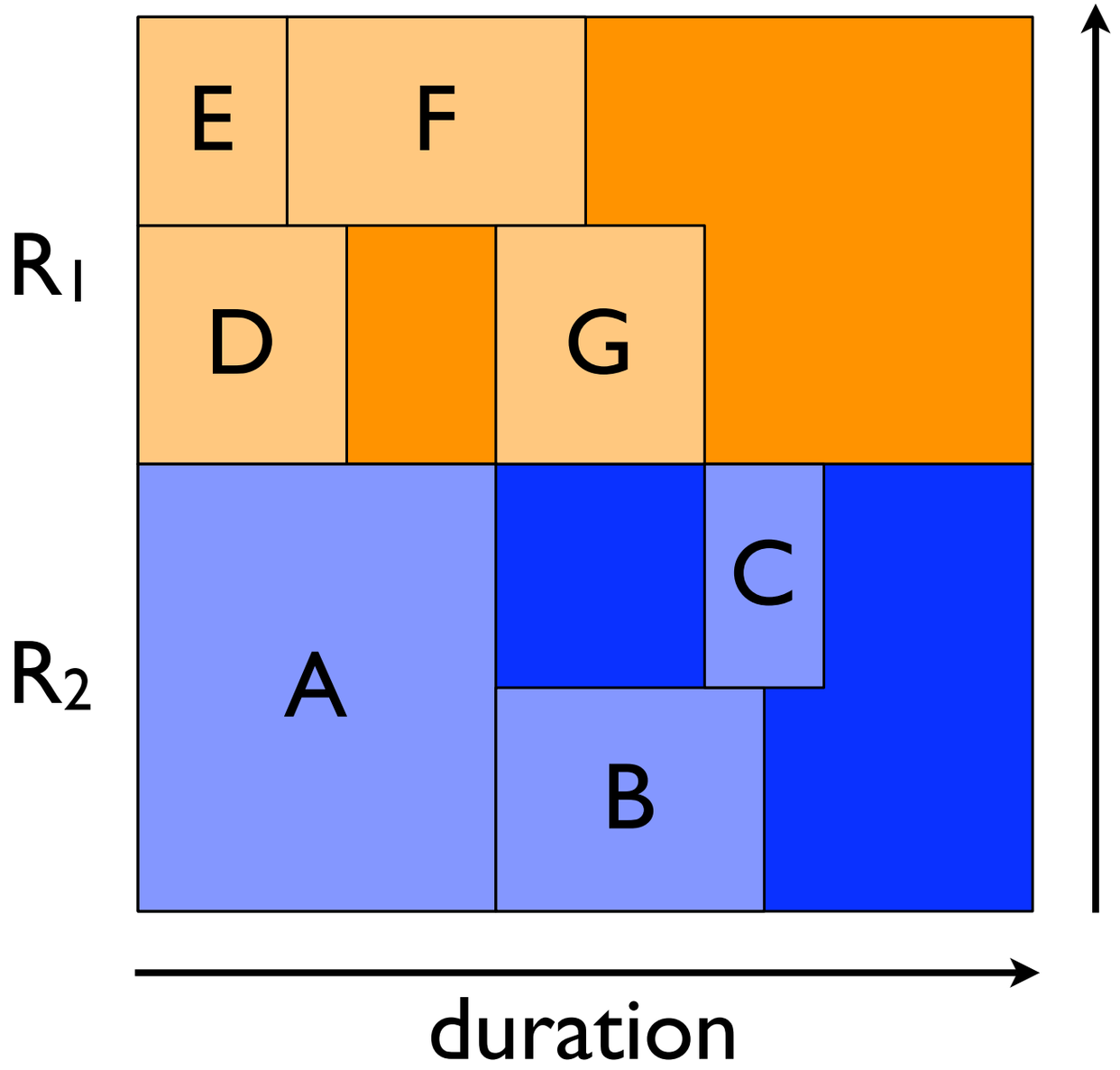


Geometric Interpretation



Geometric Interpretation

Packing squares into rectangles.



The job-shop problem

- *m* machines, *n* jobs
- each job has *m* tasks
  - each task has a fixed duration
  - each task is assigned to a different machine
- tasks in one job must be processed in order
- **always satisfiable!**
- but: **minimize total completion time**

Job-shop example

job1 = (m_2, 1) (m_0, 3) (m_1, 6) (m_3, 7) (m_5, 3) (m_4, 6)

job2 = (m_1, 8) (m_2, 5) (m_4,10) (m_5,10) (m_0,10) (m_3, 4)

job3 = (m_2, 5) (m_3, 4) (m_5, 8) (m_0, 9) (m_1, 1) (m_4, 7)

job4 = (m_1, 5) (m_0, 5) (m_2, 5) (m_3, 3) (m_4, 8) (m_5, 9)

job5 = (m_2, 9) (m_1, 3) (m_4, 5) (m_5, 4) (m_0, 3) (m_3, 1)

job6 = (m_1, 3) (m_3, 3) (m_5, 9) (m_0,10) (m_4, 4) (m_2, 1)

Job-shop example

job1 = (m_2, 1) (m_0, 3) (m_1, 6) (m_3, 7) (m_5, 3) (m_4, 6)

job2 = (m_1, 8) (m_2, 5) (m_4,10) (m_5,10) (m_0,10) (m_3, 4)

job3 = (m_2, 5) (m_3, 4) (m_5, 8) (m_0, 9) (m_1, 1) (m_4, 7)

job4 = (m_1, 5) (m_0, 5) (m_2, 5) (m_3, 3) (m_4, 8) (m_5, 9)

job5 = (m_2, 9) (m_1, 3) (m_4, 5) (m_5, 4) (m_0, 3) (m_3, 1)

job6 = (m_1, 3) (m_3, 3) (m_5, 9) (m_0,10) (m_4, 4) (m_2, 1)

disjoint tasks on one machine

Job-shop example

job1 = (m_2, 1) (m_0, 3) (m_1, 6) (m_3, 7) (m_5, 3) (m_4, 6)

job2 = (m_1, 8) (m_2, 5) (m_4,10) (m_5,10) (m_0,10) (m_3, 4)

job3 = (m_2, 5) (m_3, 4) (m_5, 8) (m_0, 9) (m_1, 1) (m_4, 7)

job4 = (m_1, 5) (m_0, 5) (m_2, 5) (m_3, 3) (m_4, 8) (m_5, 9)

job5 = (m_2, 9) (m_1, 3) (m_4, 5) (m_5, 4) (m_0, 3) (m_3, 1)

job6 = (m_1, 3) (m_3, 3) (m_5, 9) (m_0,10) (m_4, 4) (m_2, 1)

disjoint tasks on one machine

serial tasks in one job

Job-shop example

job1 = (m_2, 1) (m_0, 3) (m_1, 6) (m_3, 7) (m_5, 3) (m_4, 6)

job2 = (m_1, 8) (m_2, 5) (m_4,10) (m_5,10) (m_0,10) (m_3, 4)

job3 = (m_2, 5) (m_3, 4) (m_5, 8) (m_0, 9) (m_1, 1) (m_4, 7)

job4 = (m_1, 5) (m_0, 5) (m_2, 5) (m_3, 3) (m_4, 8) (m_5, 9)

job5 = (m_2, 9) (m_1, 3) (m_4, 5) (m_5, 4) (m_0, 3) (m_3, 1)

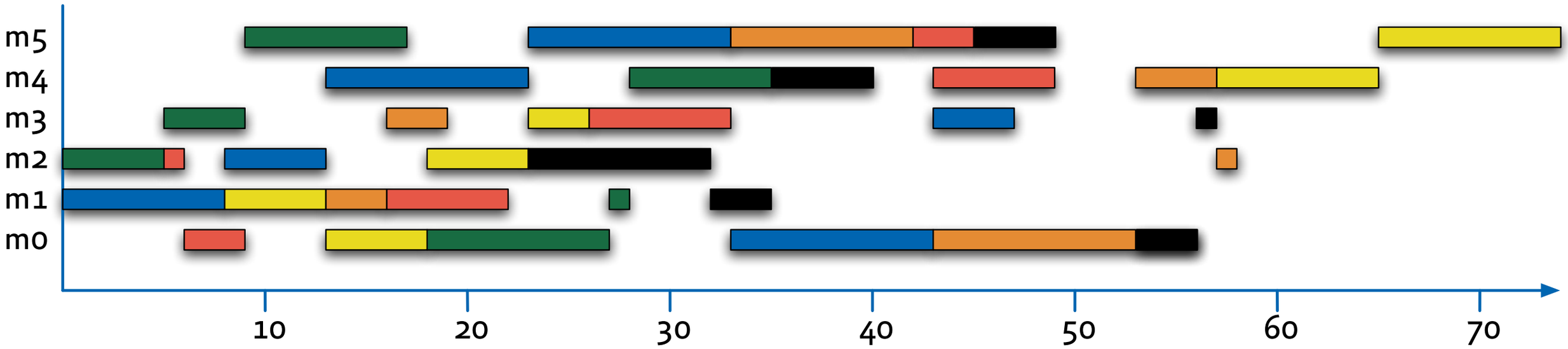
job6 = (m_1, 3) (m_3, 3) (m_5, 9) (m_0,10) (m_4, 4) (m_2, 1)

disjoint tasks on one machine

minimize completion time

serial tasks in one job

Example: optimal solution



job 1
job 2
job 3
job 4
job 5
job 6

earliest completion time: 74

The job-shop problem

- **Hard problem!**
- **6x6 instance solvable using Gecode**
  - disjunction by reification
  - normal branching
- **Classic 10x10 instance not solvable using Gecode!**
  - specialized propagators (edge-finding) and branchings needed

Soft Constraints

- **Some problems are over-constrained:**
  - college time-tabling involving students' choices
  - aligning people on a photo (with preferences of whom to stand next to)
- **No solution satisfying all constraints!**
- **Find solution that violates few constraints**
- **Make constraints *soft***

Soft Constraints

- **Several approaches:**
  - "**soft-as-hard**": introduce additional variables that model violation

    distinct(x_1,x_2,…,x_n, *v*)

    where *v* is the number of violated pairs, use branch-and-bound to minimize *v*
  - **local search** techniques
  - **bucket elimination**

Literature

- Baptiste, Le Pape, Nuijten. *Constraint-based Scheduling*. Kluwer, 2001.
- Van Beek, Wilken. *Fast Optimal Instruction Scheduling for Single-issue Processors with Arbitrary Latencies*. In: CP 2001, Springer-Verlag.

Rostering

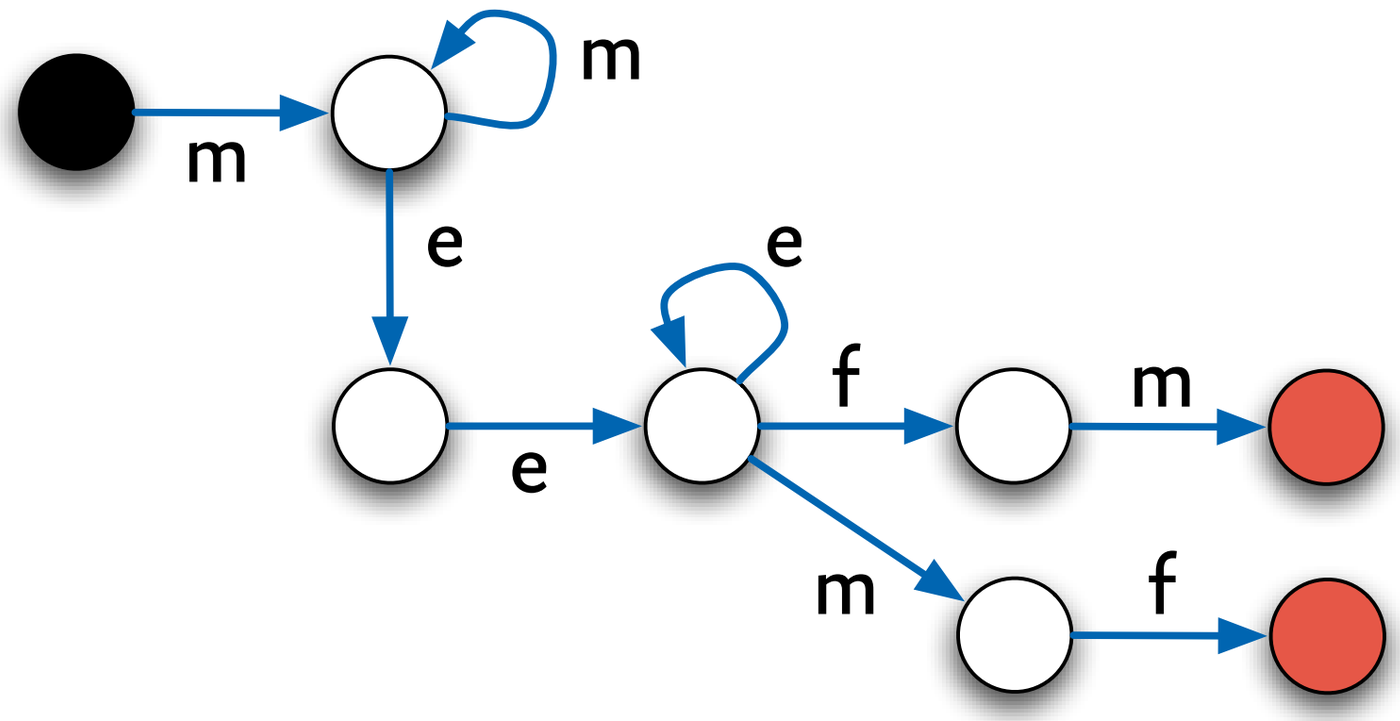
Example: shifts in a hospital

- **Assign nurses to shifts**
- **Constraints:**
  - all shifts taken by a nurse
  - balance different shift types
  - side constraints (extra days off, public holidays)
- **Expressible as finite automata!**

Regular expressions and finite automata

Reminder:

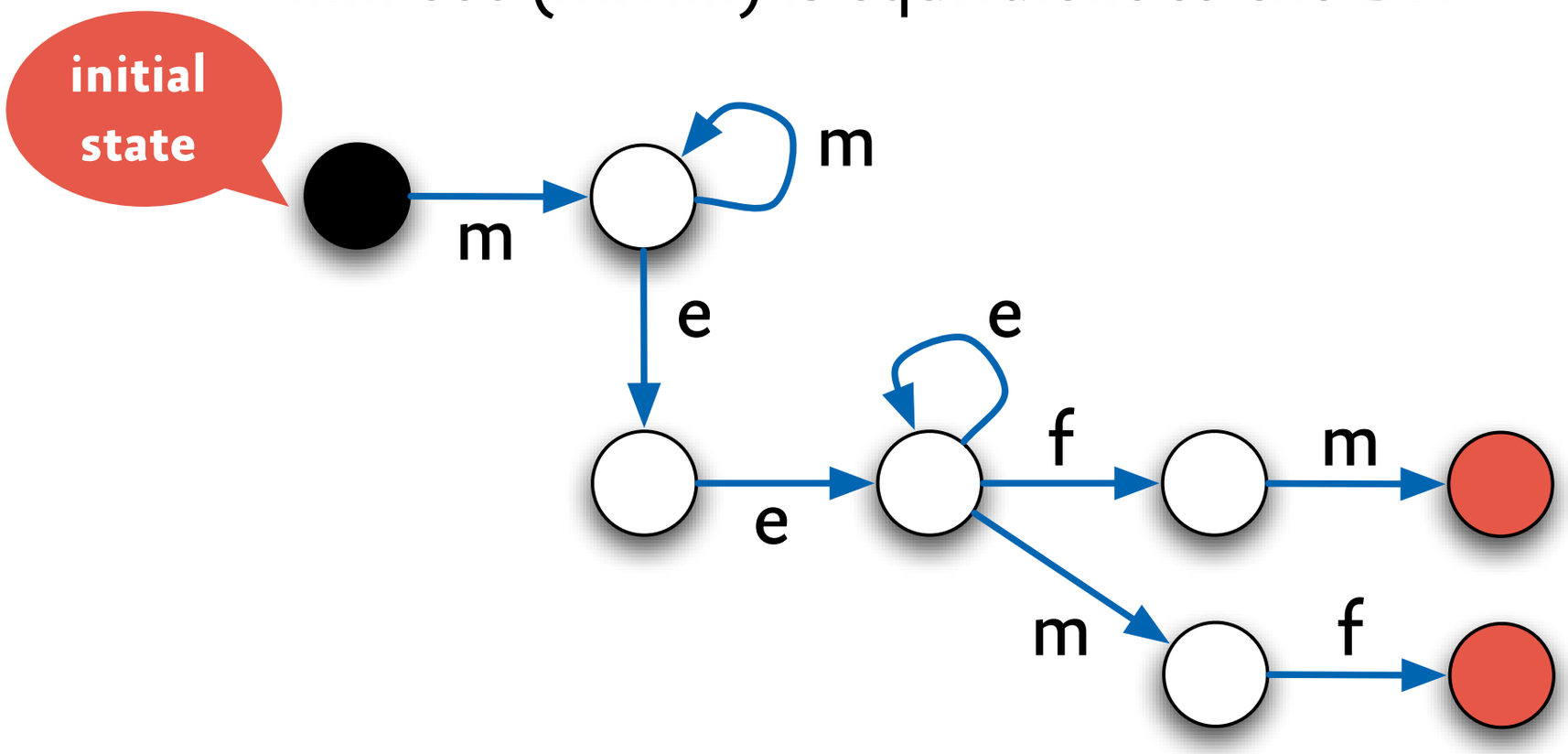
$mm^*eee^*(fm+mf)$ is equivalent to the DFA



Regular expressions and finite automata

Reminder:

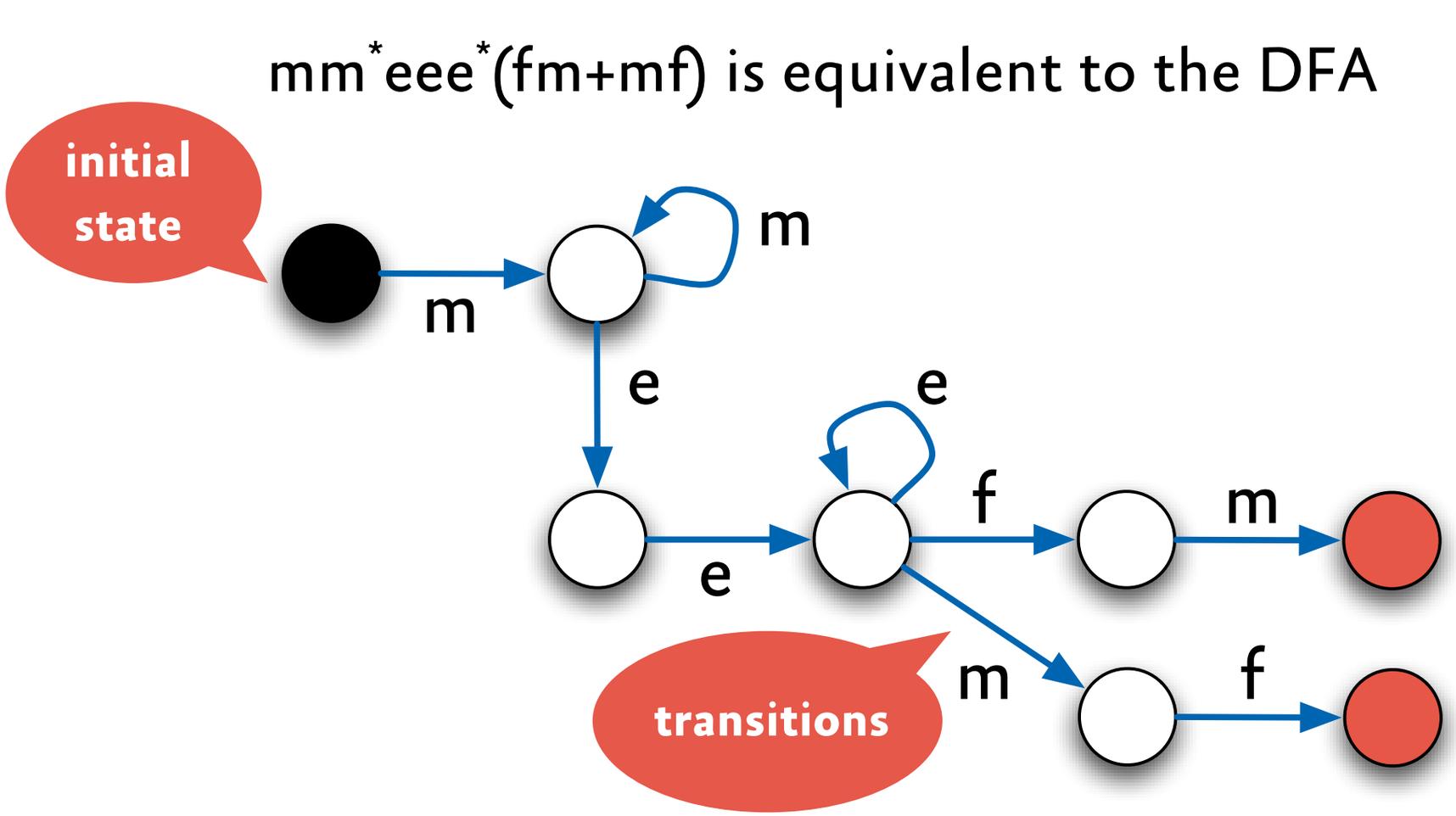
mm*eee*(fm+mf) is equivalent to the DFA



Regular expressions and finite automata

Reminder:

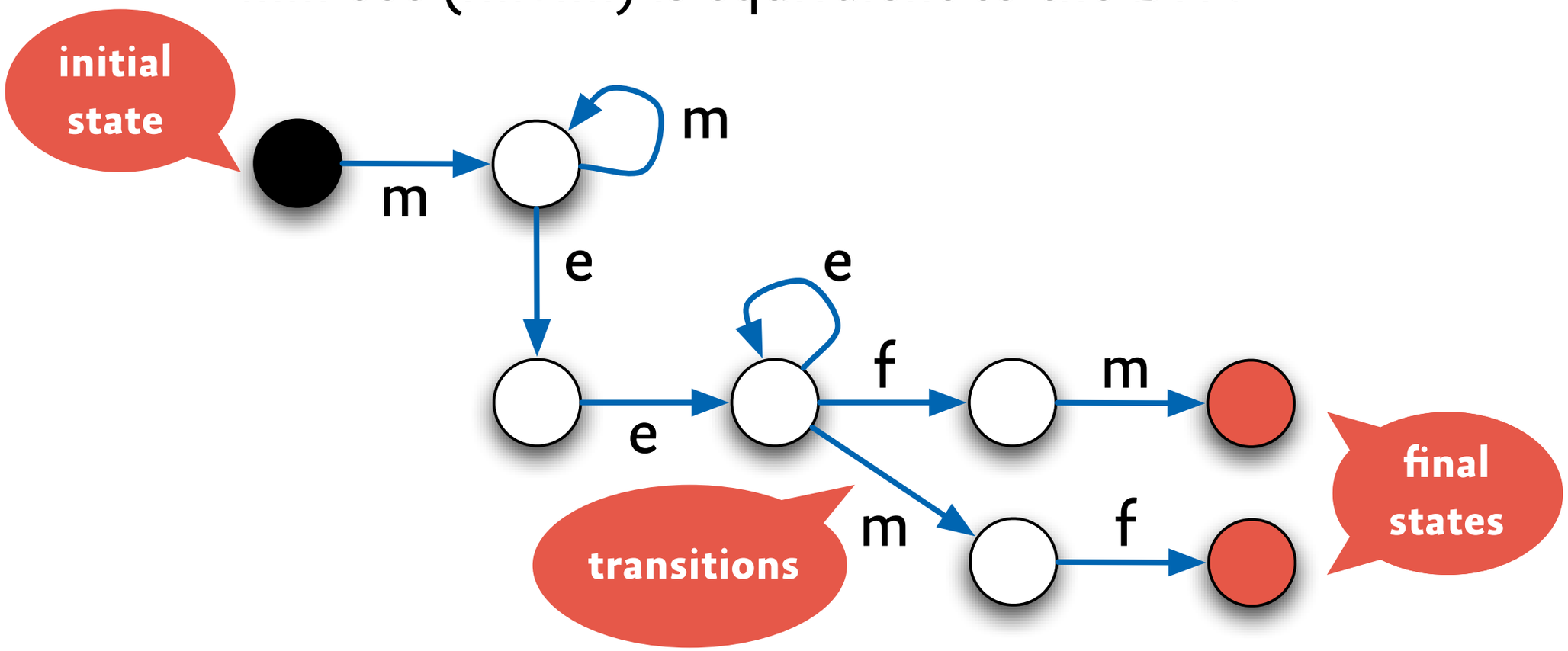
$mm^*eee^*(fm+mf)$ is equivalent to the DFA



Regular expressions and finite automata

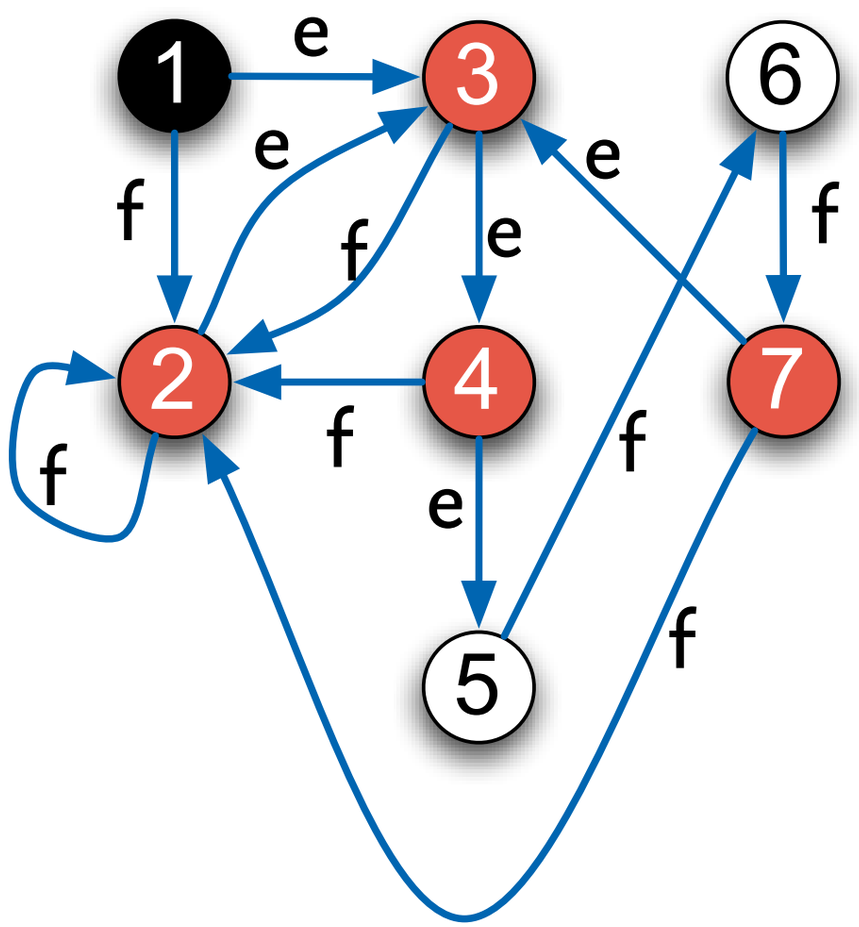
Reminder:

mm*eee*(fm+mf) is equivalent to the DFA



Typical automata for rostering

"after 3 night shifts, at least two days off"



patterned stretch constraint

The regular language membership constraint

- **Given:**
  - sequence of variables $\langle x_1,\ldots,x_n\rangle$
  - regular expression *e* over the alphabet $\mathbb{N}$
- **Constraint:**

  regular($\langle x_1,\ldots,x_n\rangle$, *e*) holds iff

  the word $x_1x_2\ldots x_n$ (a string in $\mathbb{N}^n$) is in the language L(*e*)

The regular language membership constraint

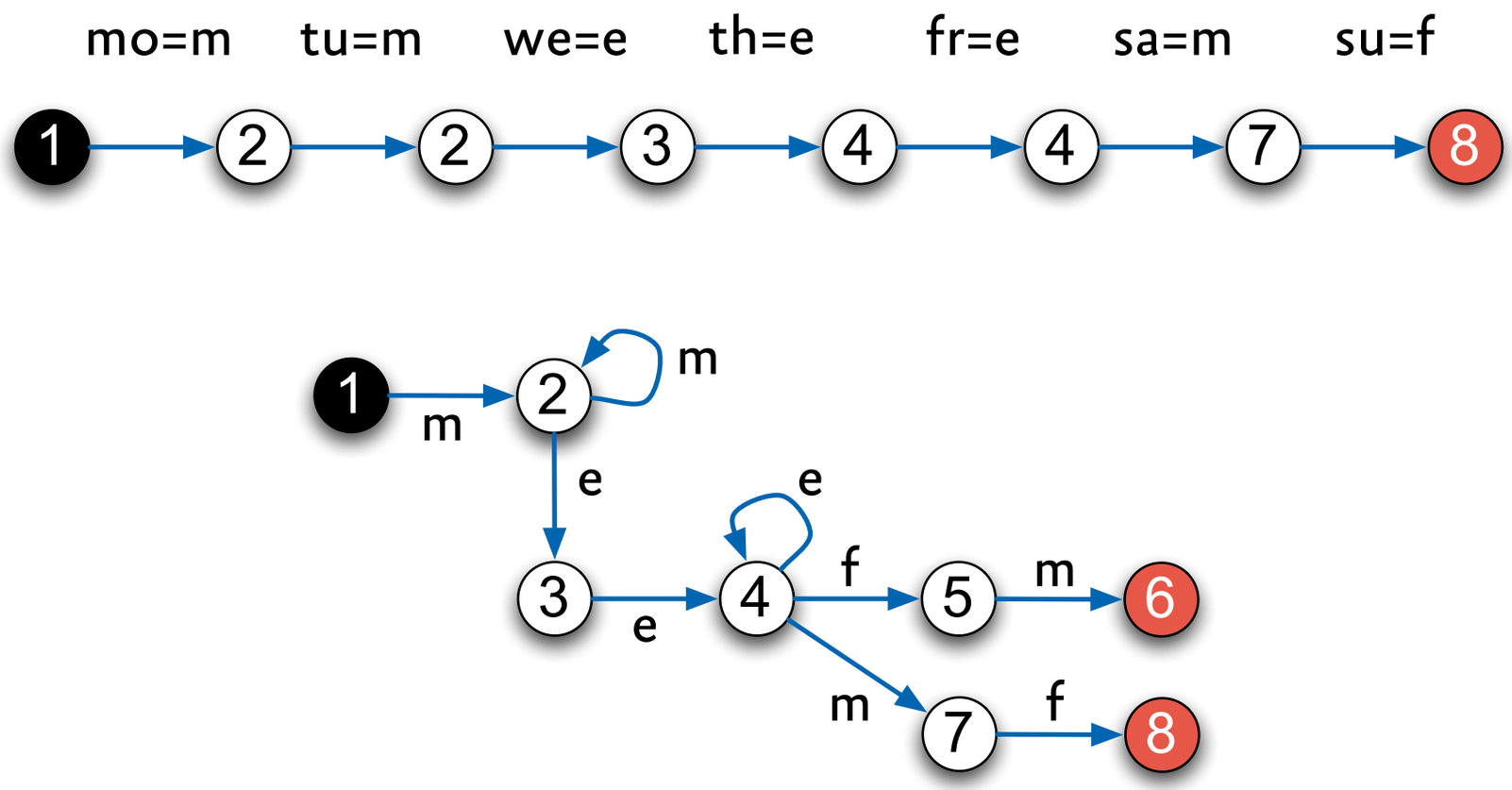
- **Example:**
  - variables *mo,tu,we,th,fr,sa,su*
  - shift types *m(orning)=1, e(vening)=2, f(ree)=3*
  - regular expression

    $mm^{*}eee^{*}(fm+mf)$

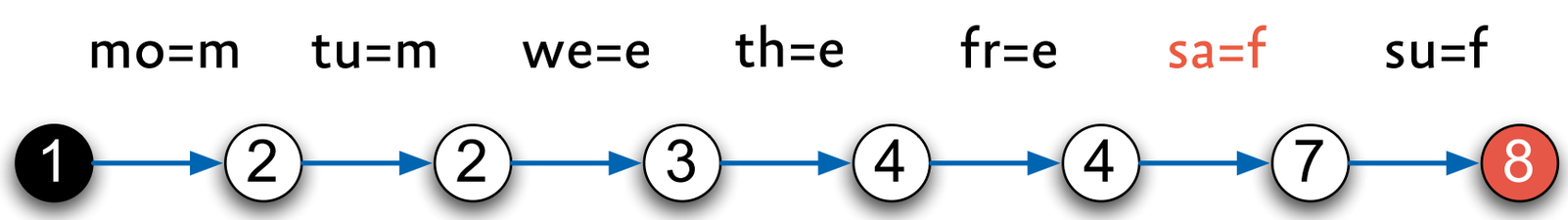
    produces the language (**modulo length 7**):

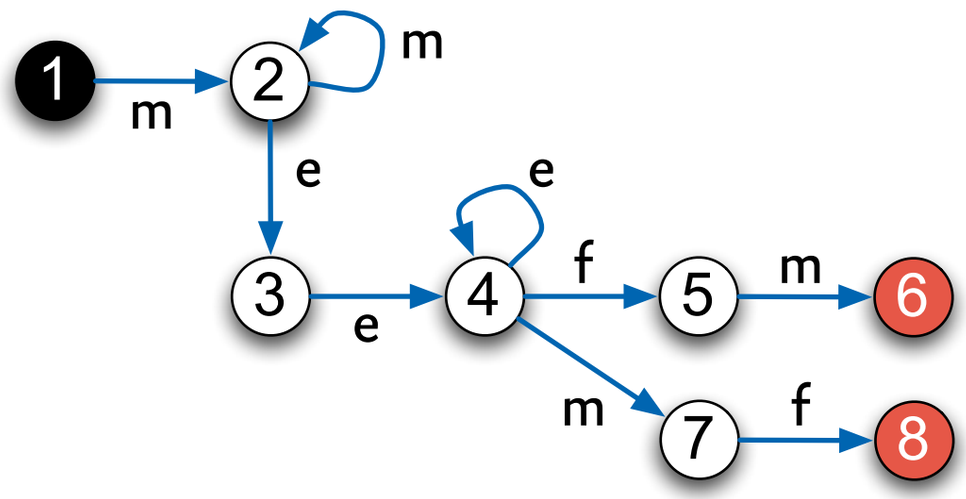
    { mmeeefm, mmmeefm, meeeefm,
      mmeeemf, mmmeemf, meeeemf }

Checking the regular constraint



Checking the regular constraint



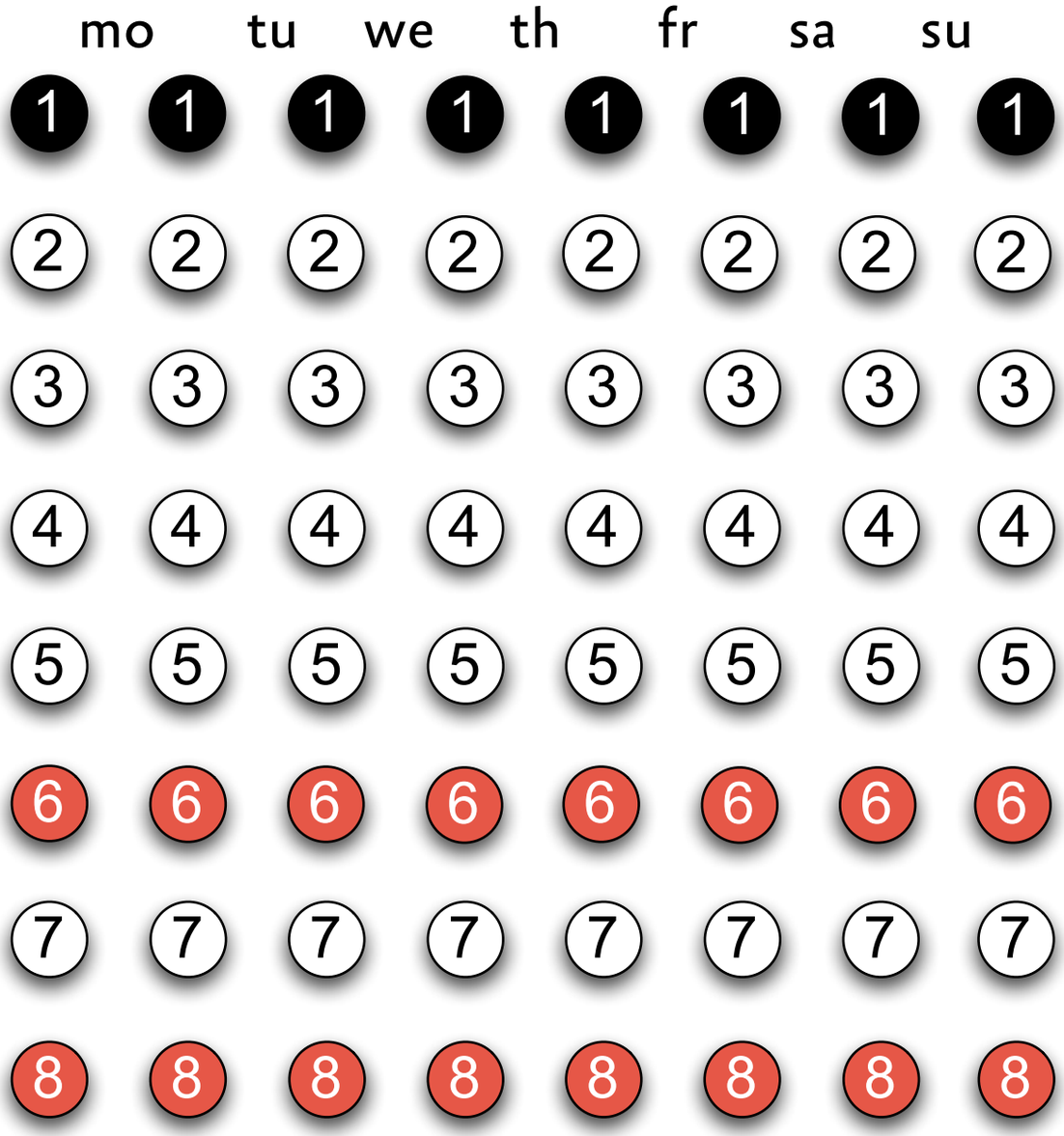


Checking the regular constraint

mo=m tu=m we=e th=e fr=e sa=f su=f

failure!

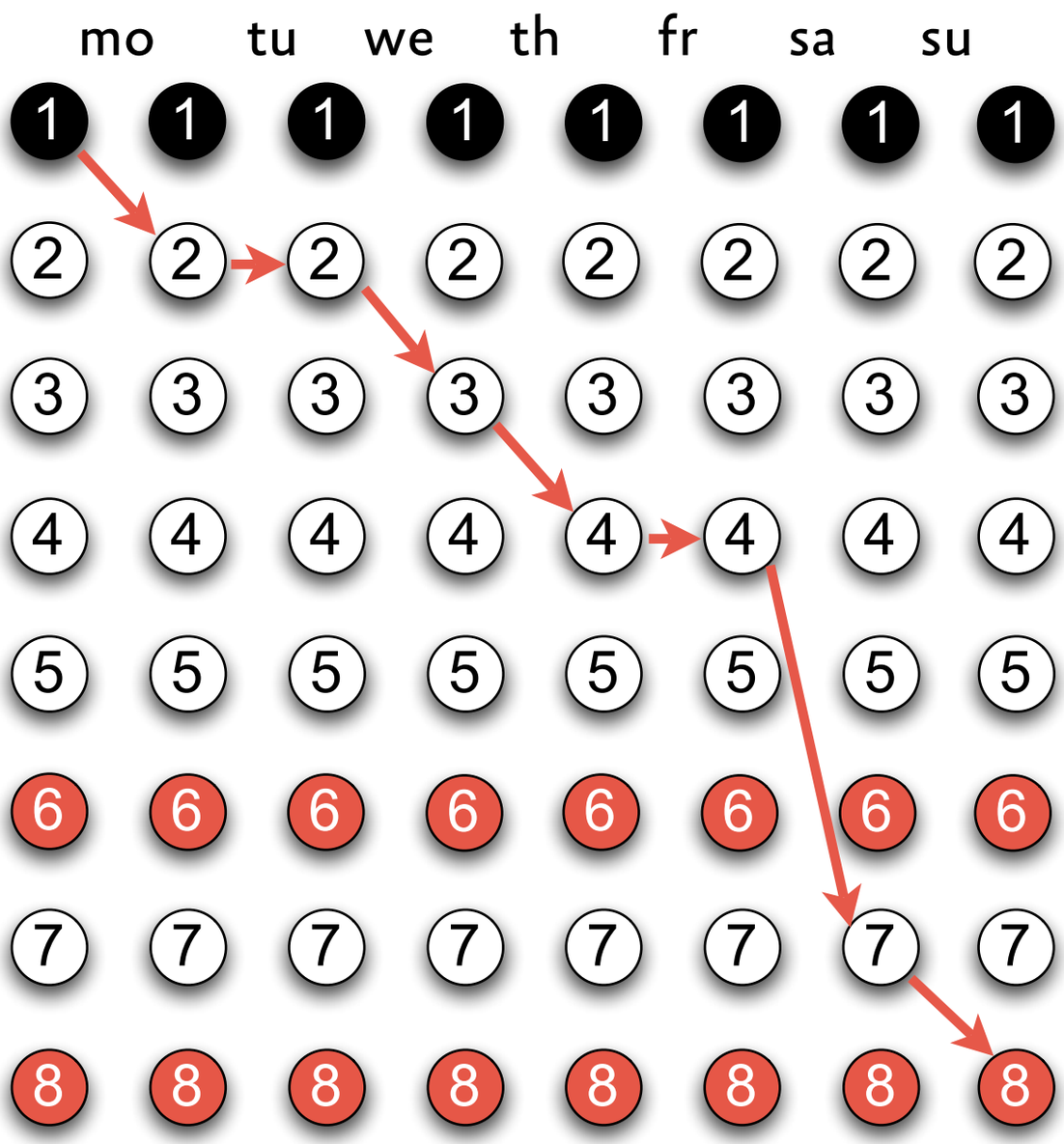
Checking the regular constraint



layered graph

- n+1 rows, |Q| columns
- column = "layer"
- arcs only between consecutive layers
- arc (i-1,j) → (i,k): transition possible from Q_j to Q_k using symbol from x_i

Checking the regular constraint



layered graph

mo ∈ {m}

tu ∈ {m}

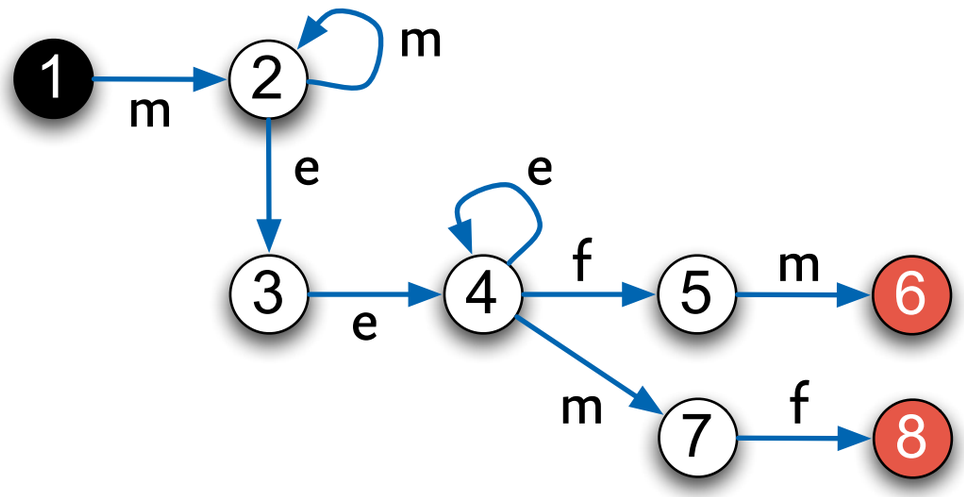
we ∈ {e}

th ∈ {e}

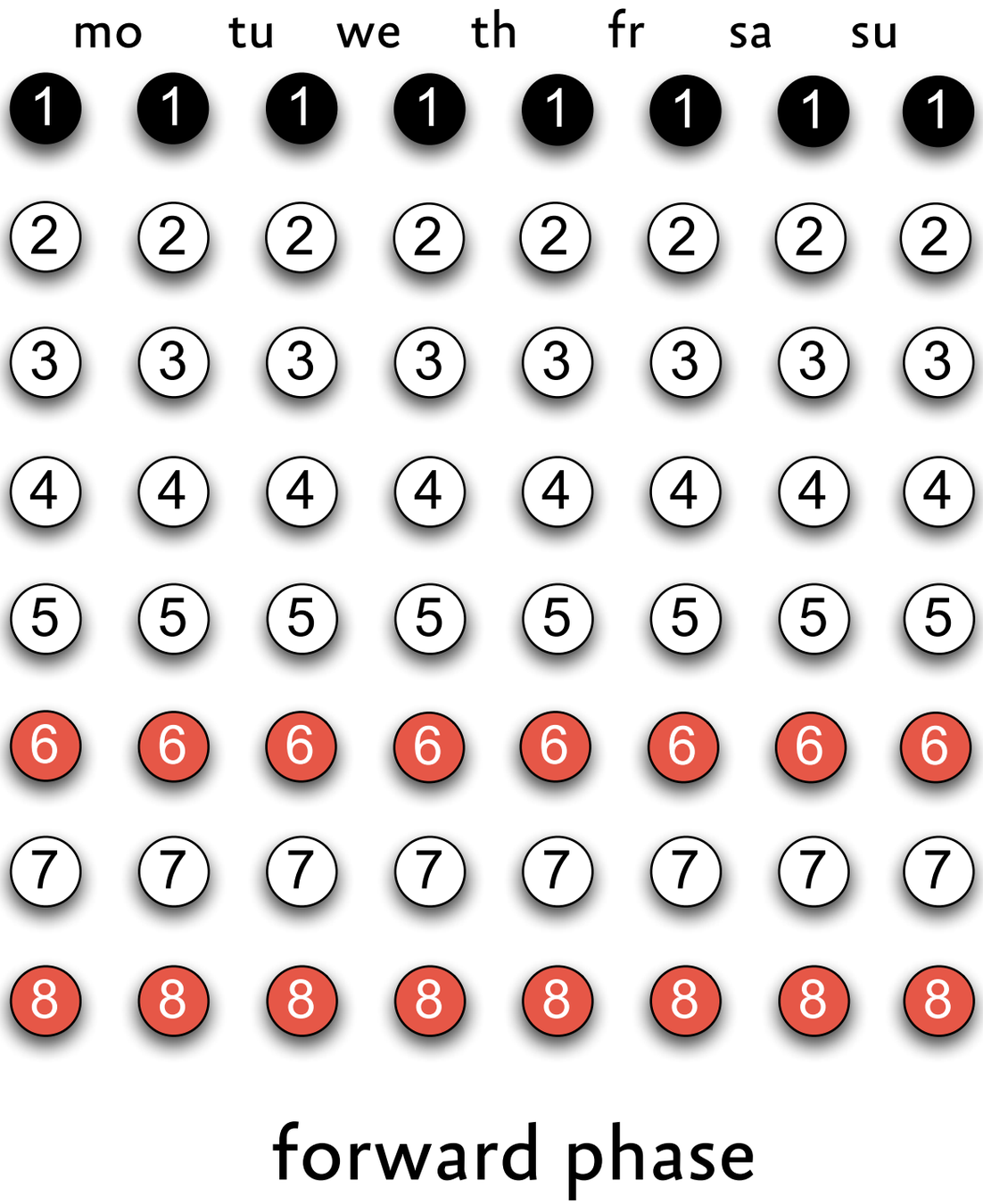
fr ∈ {e}

sa ∈ {m}

su ∈ {f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

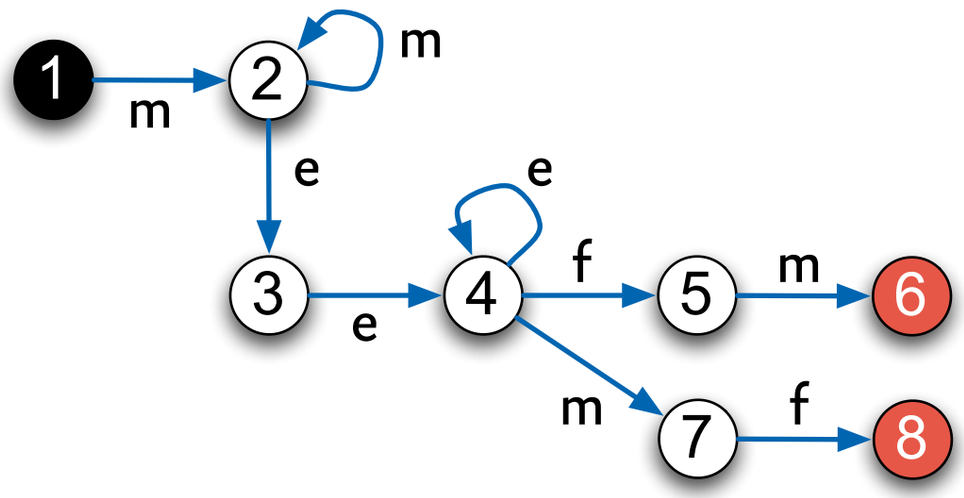
we ∈ {m,e,f}

th ∈ {m,e,f}

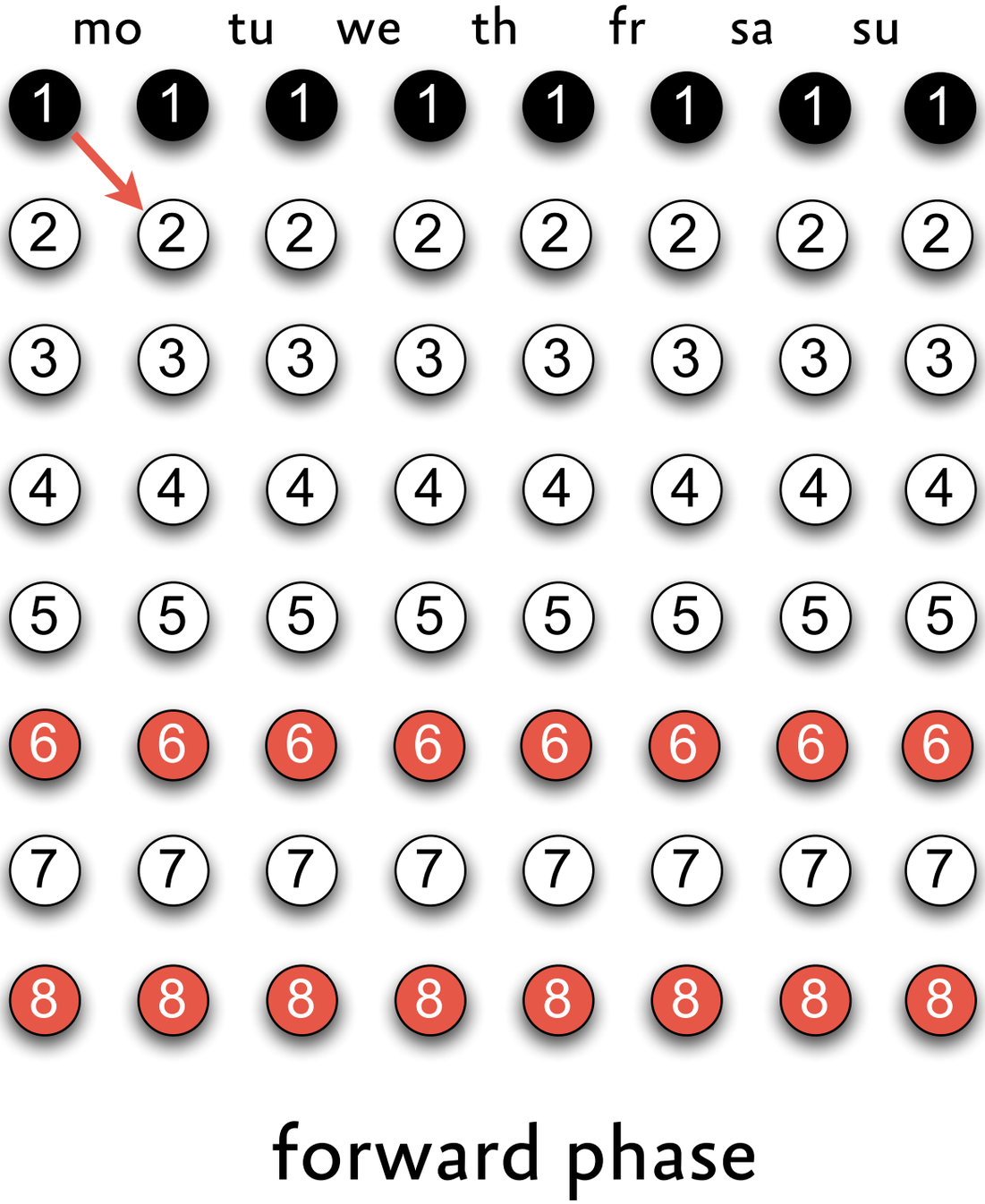
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

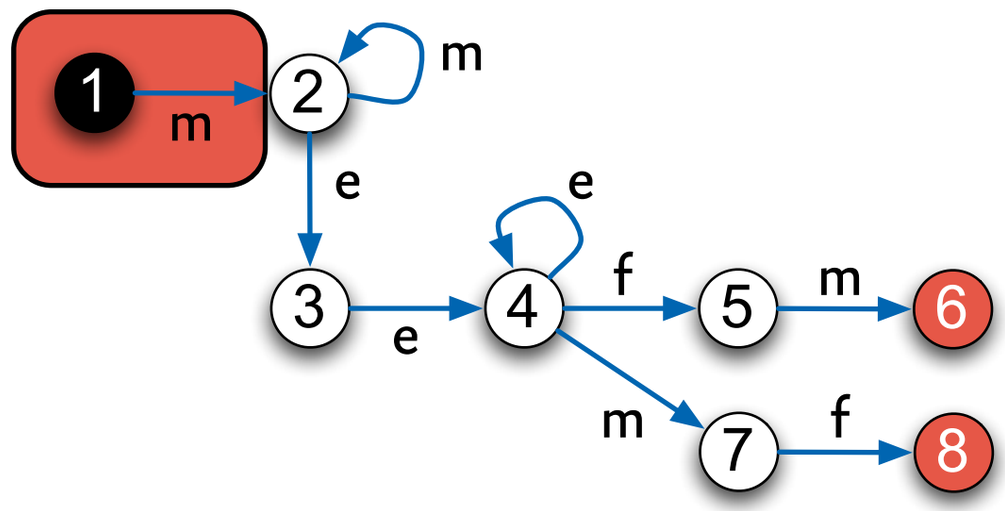
we ∈ {m,e,f}

th ∈ {m,e,f}

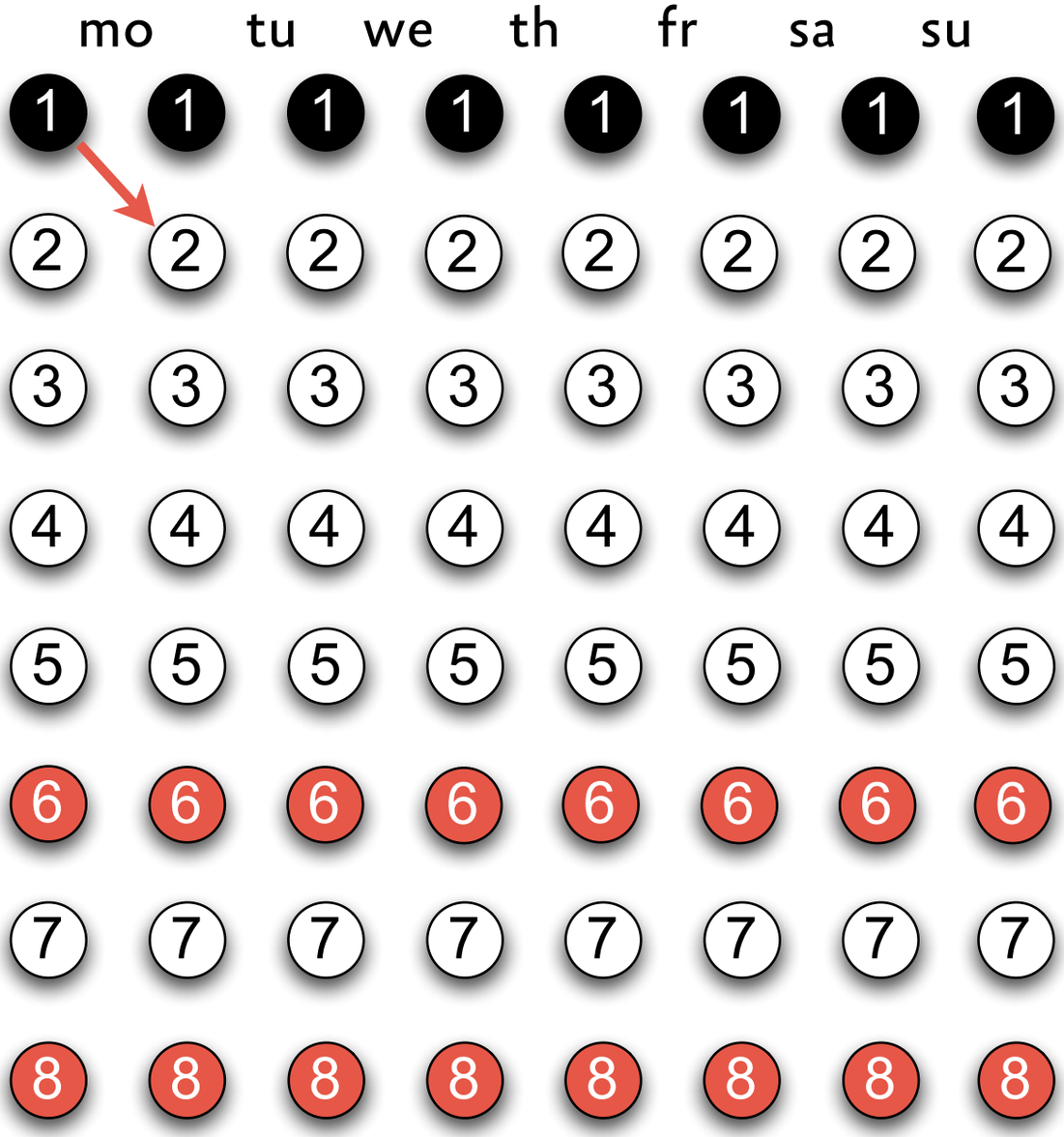
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

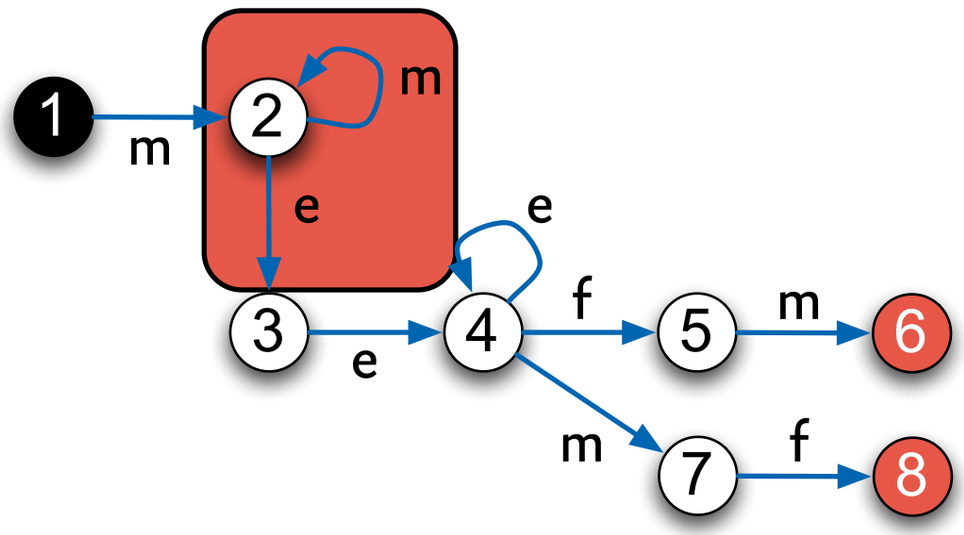
we ∈ {m,e,f}

th ∈ {m,e,f}

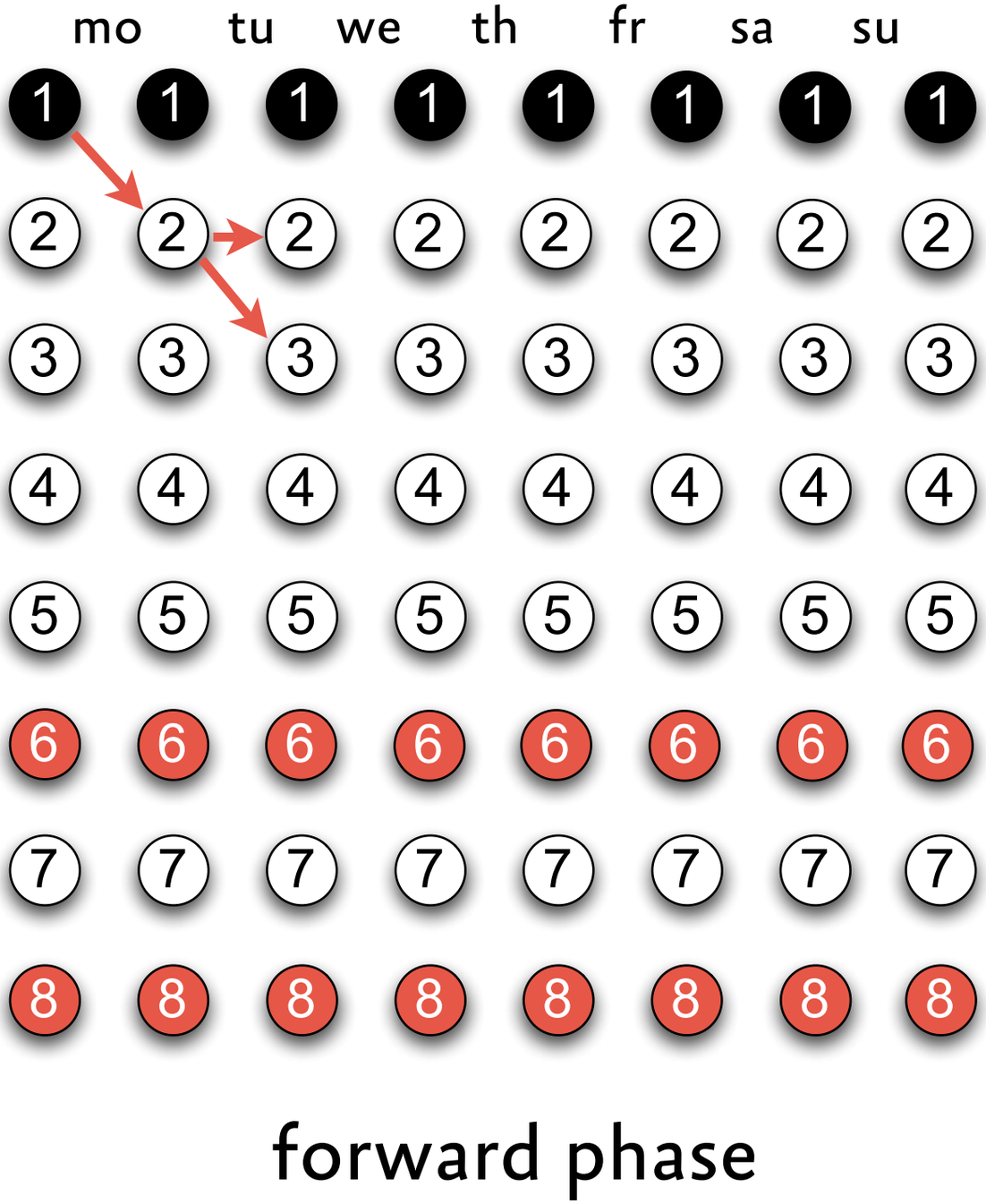
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

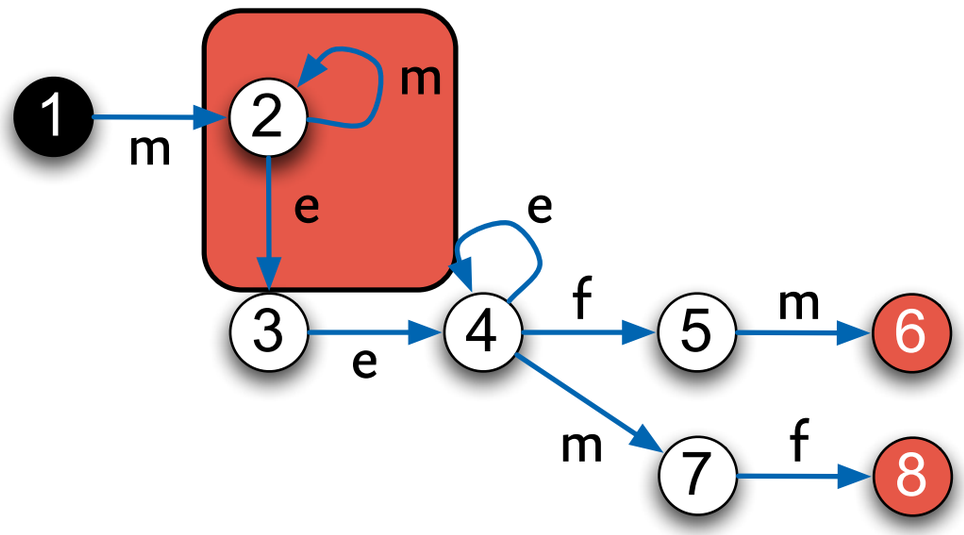
we ∈ {m,e,f}

th ∈ {m,e,f}

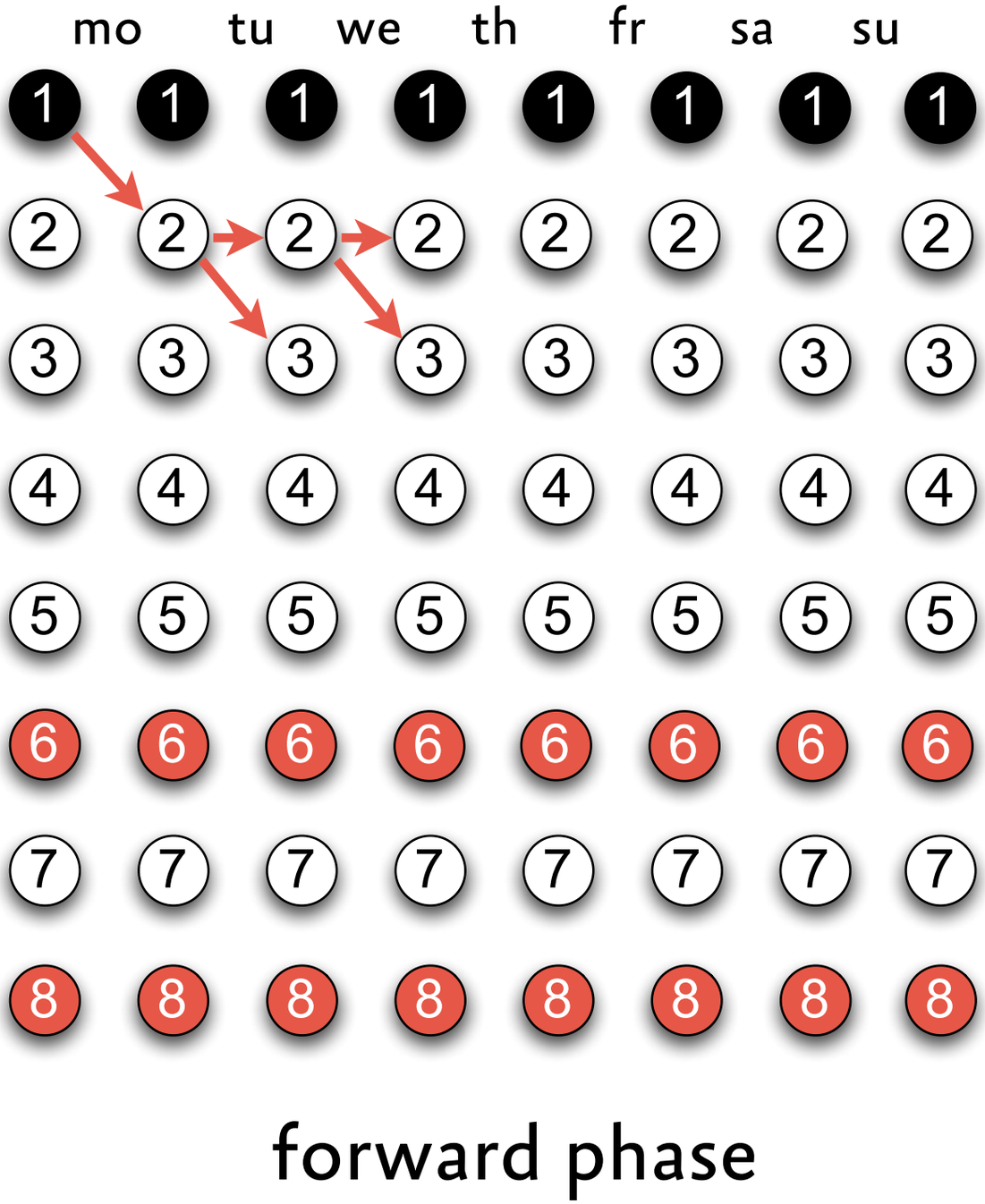
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

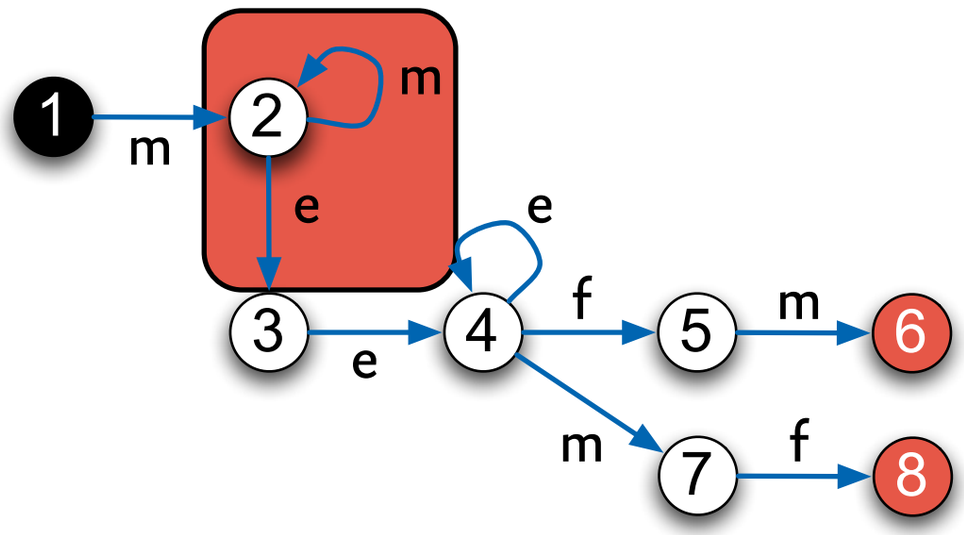
we ∈ {m,e,f}

th ∈ {m,e,f}

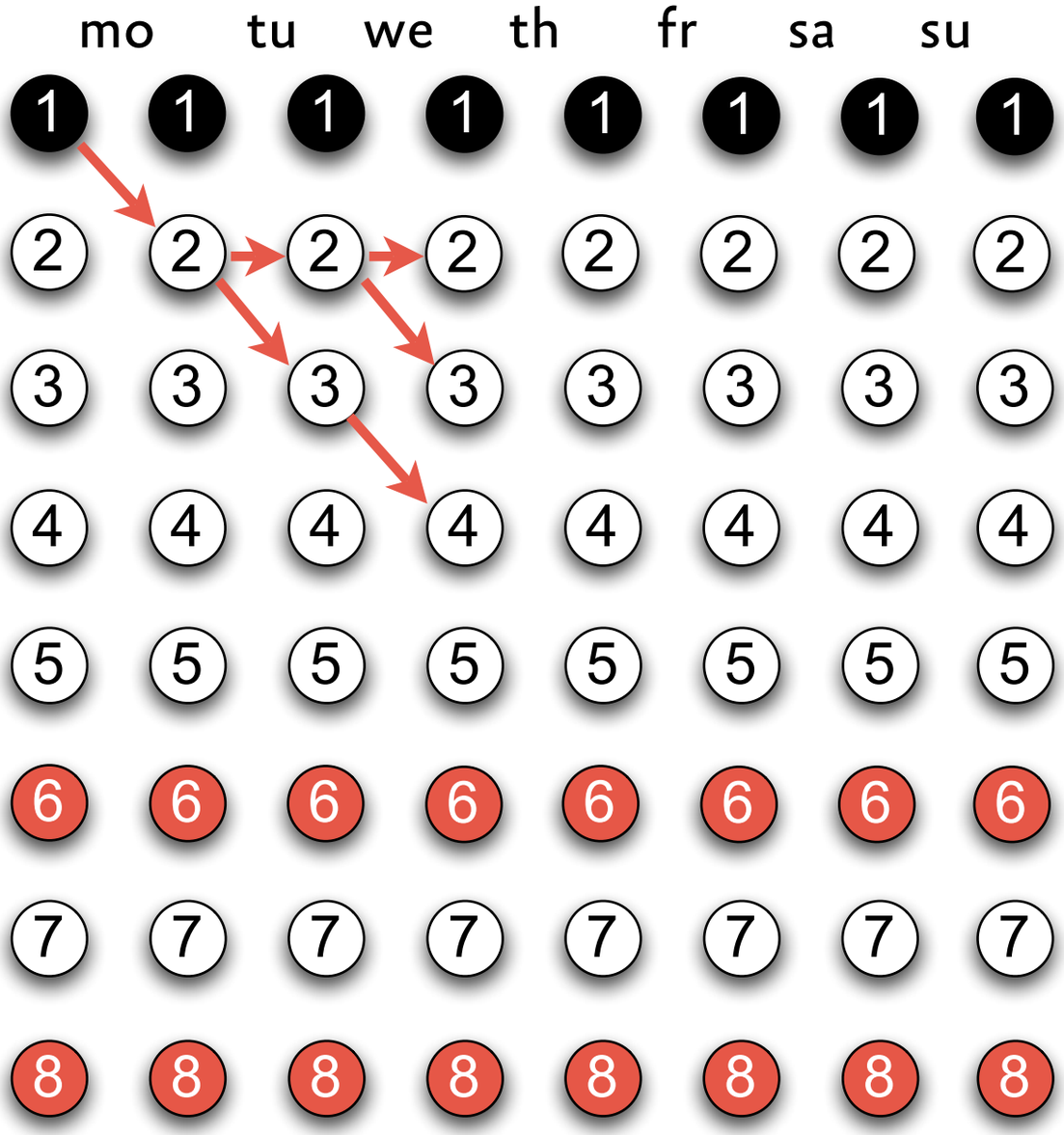
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

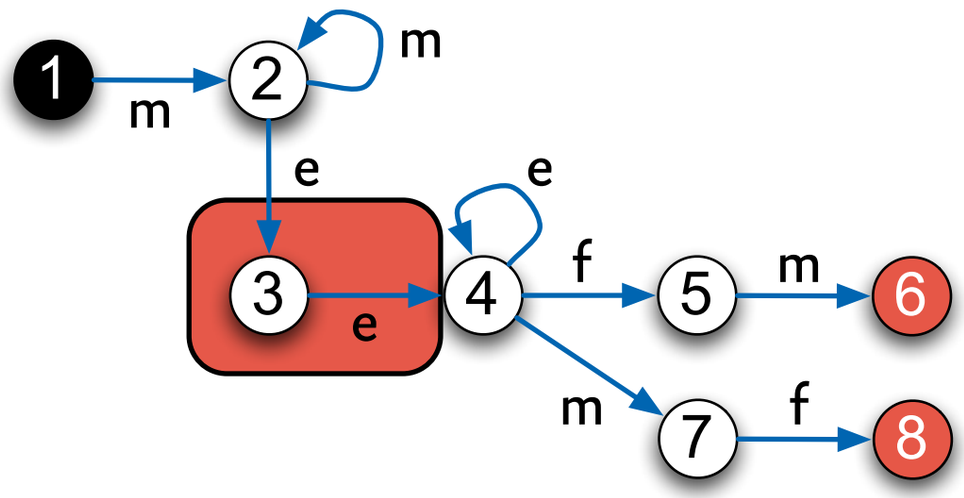
we ∈ {m,e,f}

th ∈ {m,e,f}

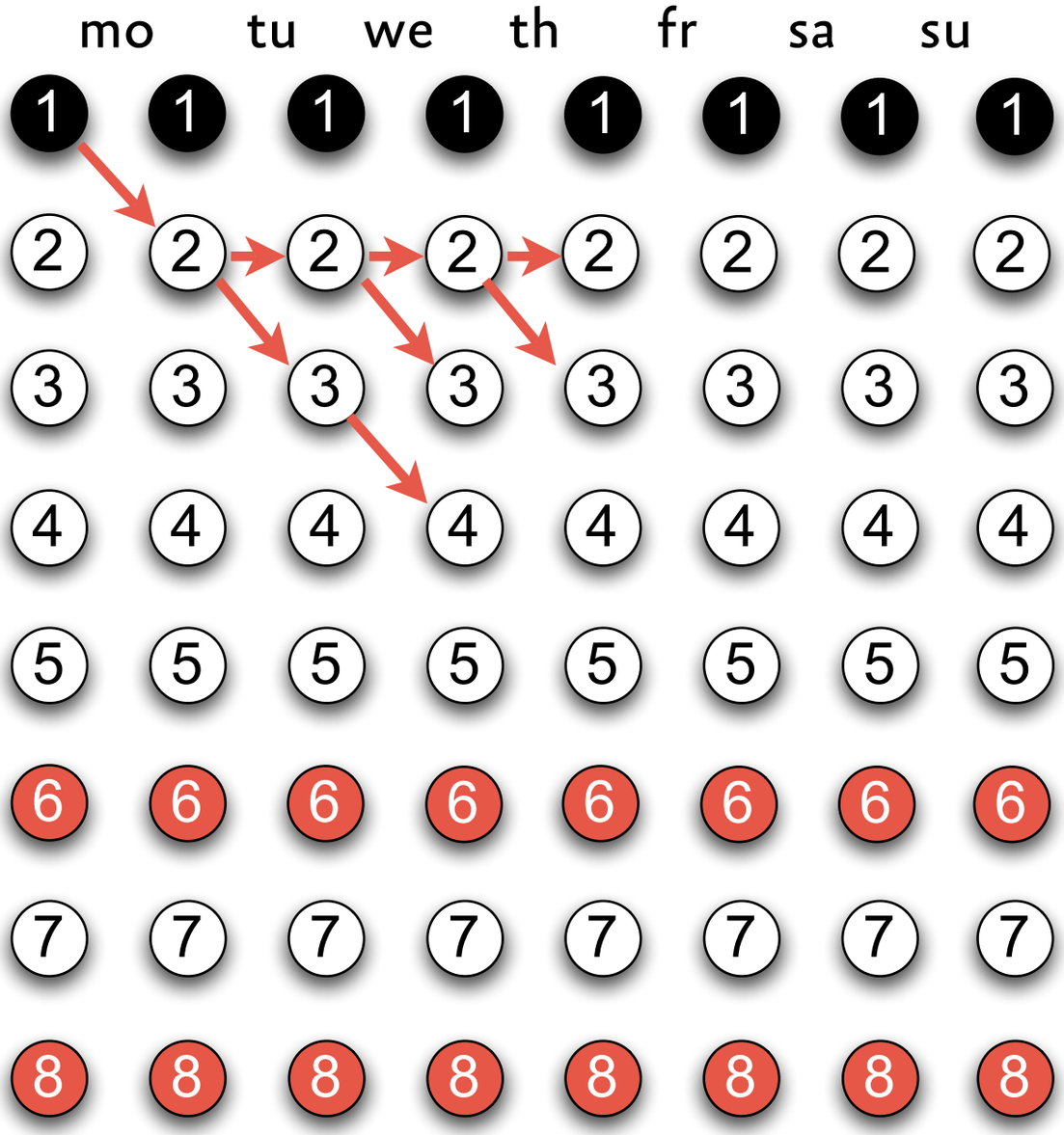
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

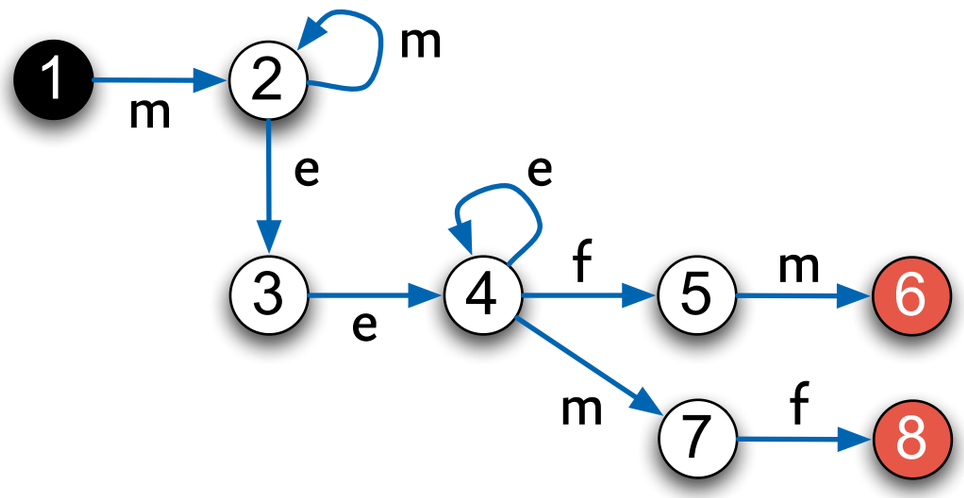
we ∈ {m,e,f}

th ∈ {m,e,f}

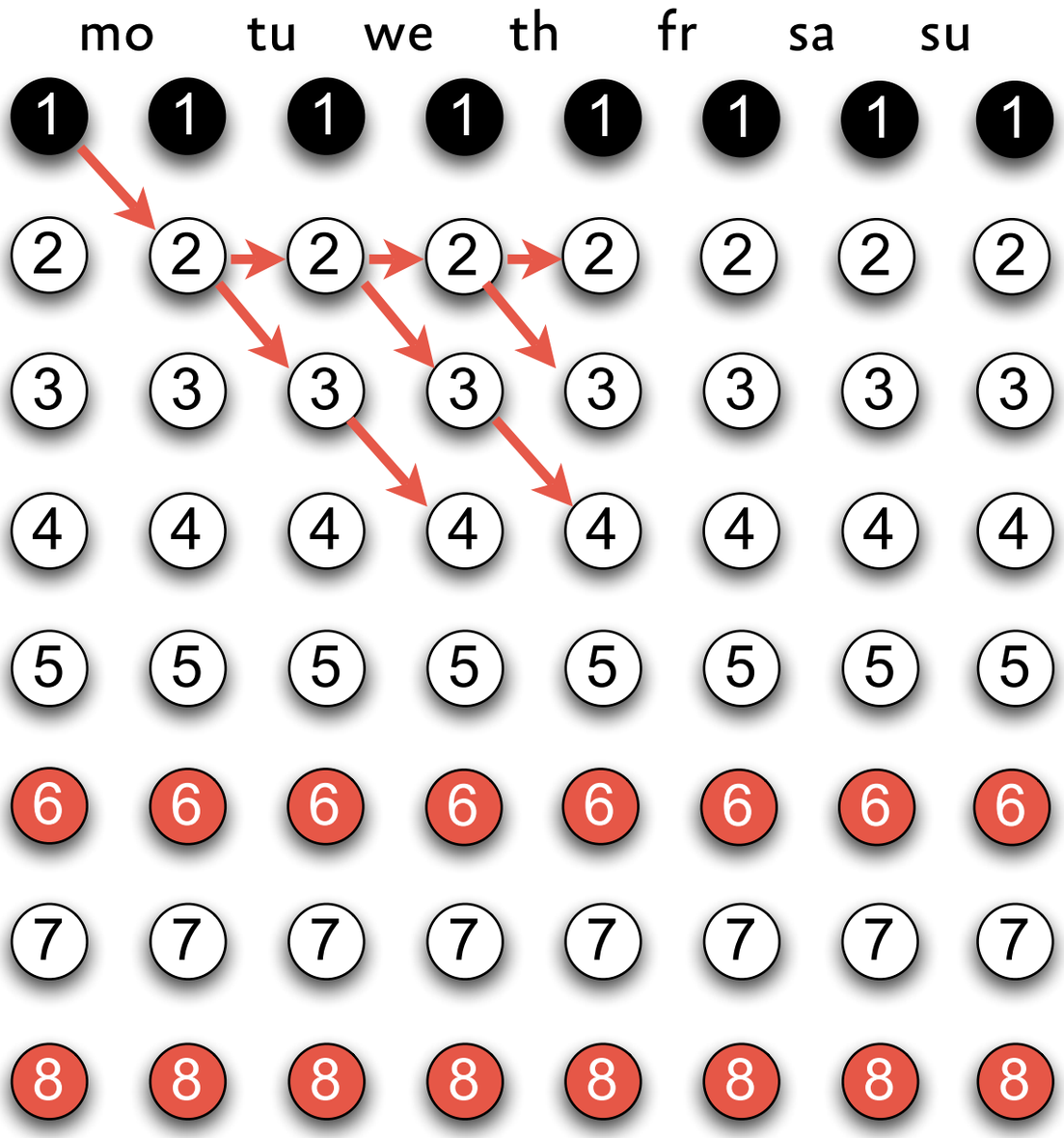
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

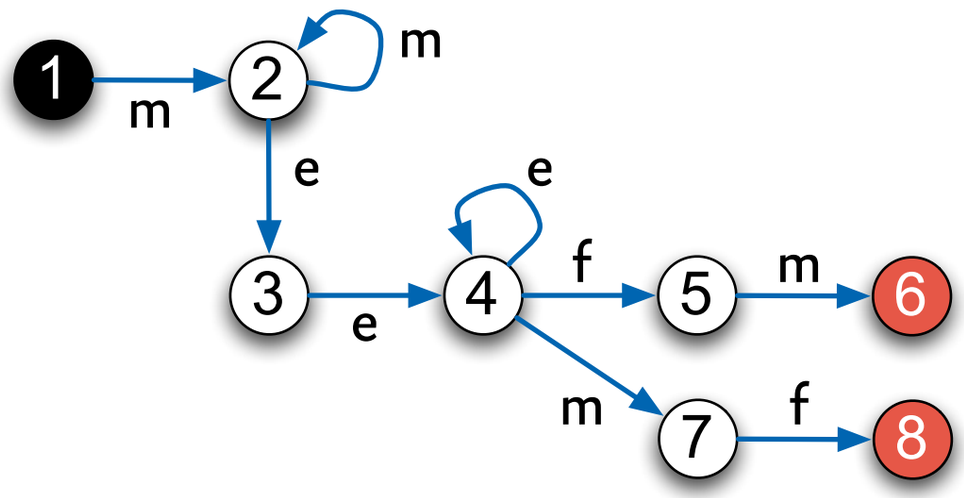
we ∈ {m,e,f}

th ∈ {m,e,f}

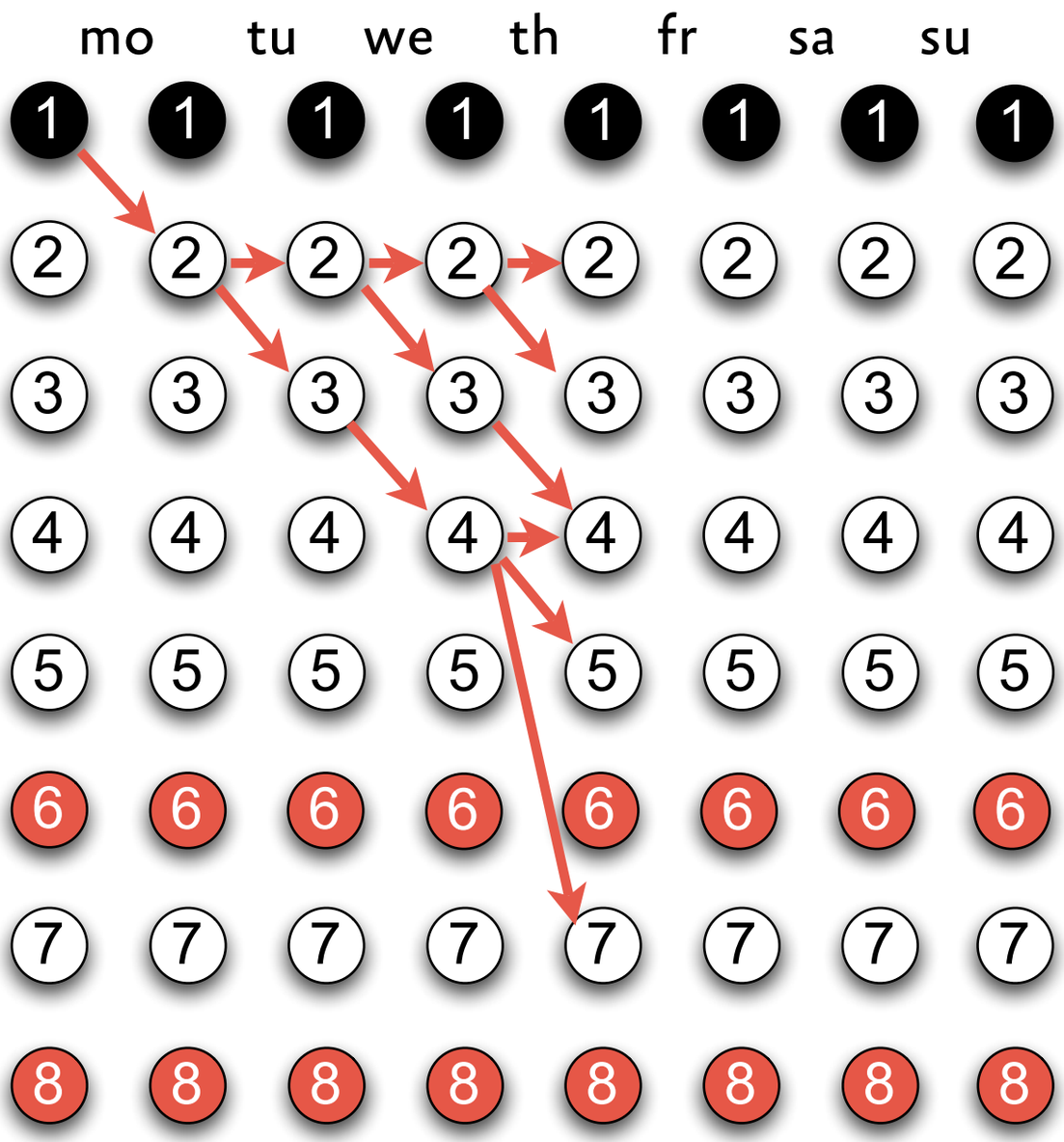
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

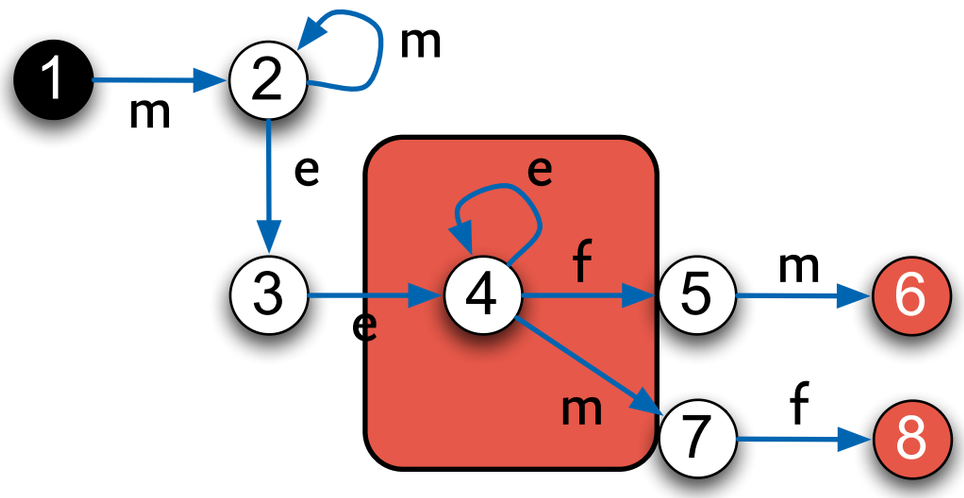
we ∈ {m,e,f}

th ∈ {m,e,f}

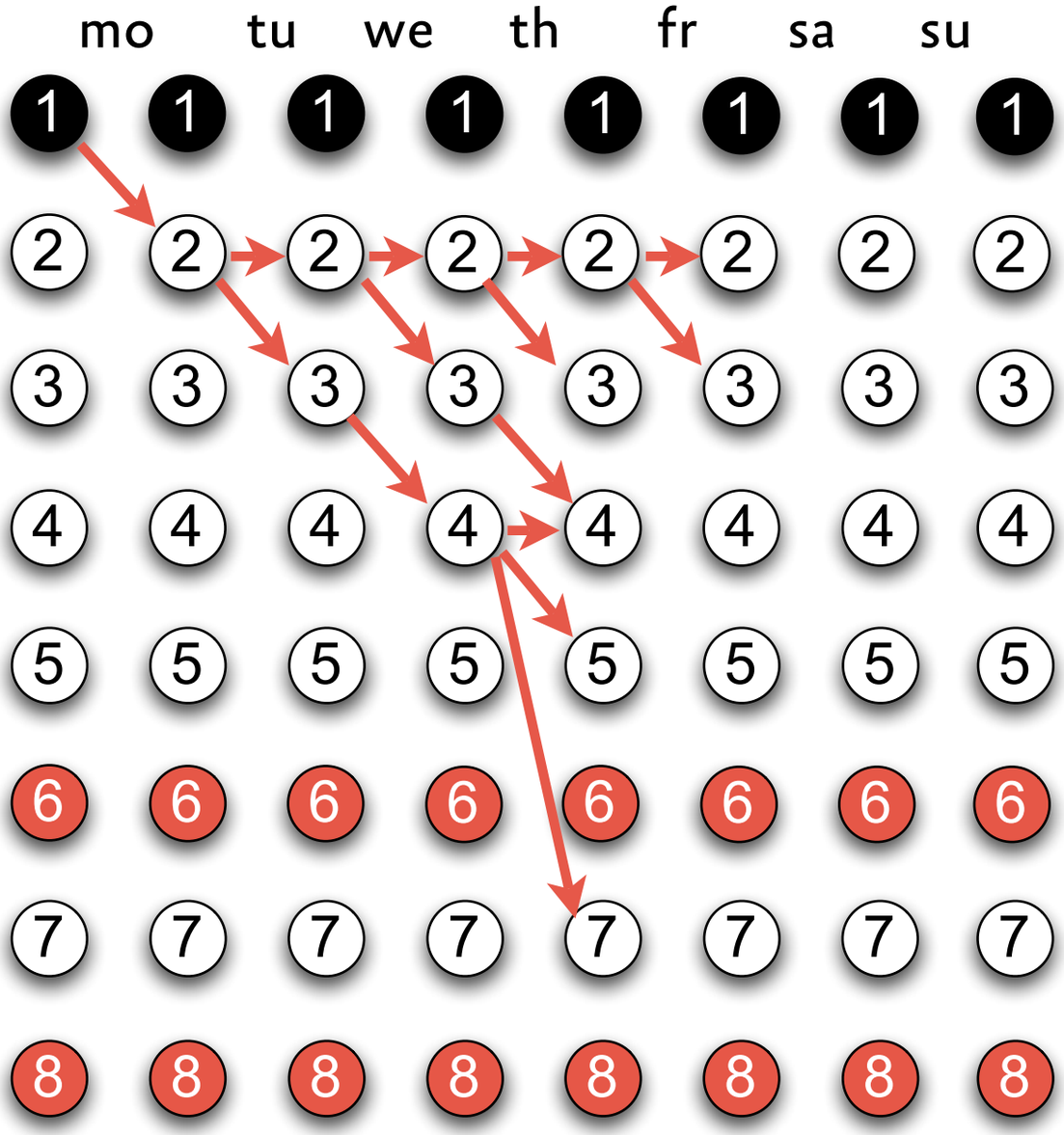
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

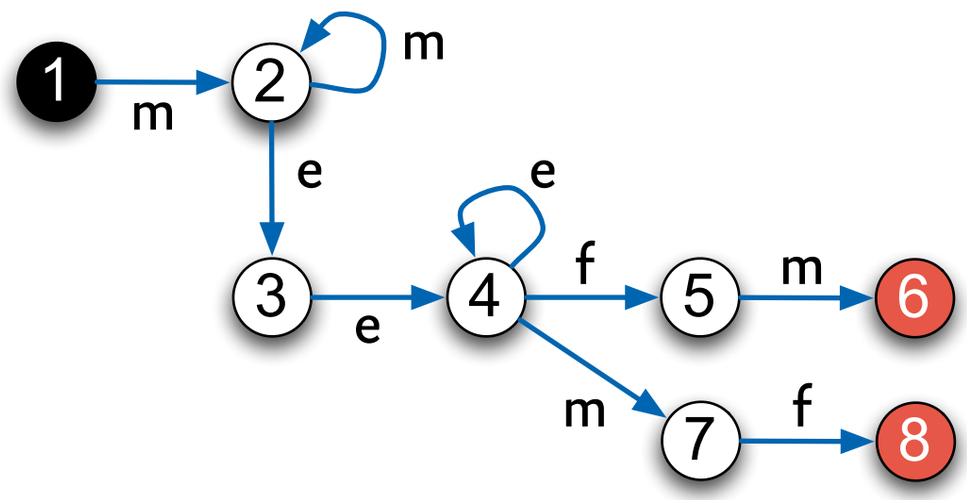
we ∈ {m,e,f}

th ∈ {m,e,f}

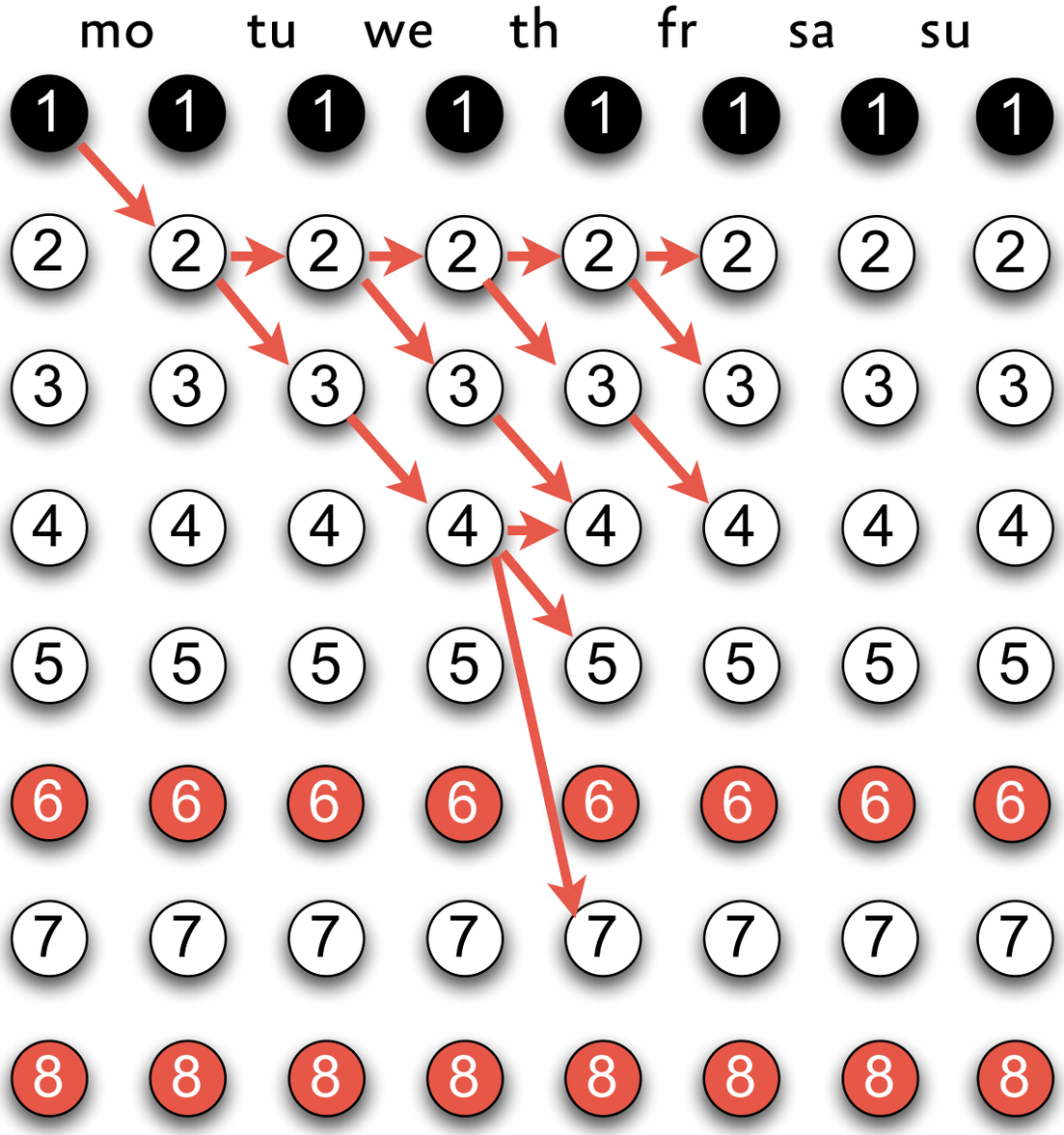
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

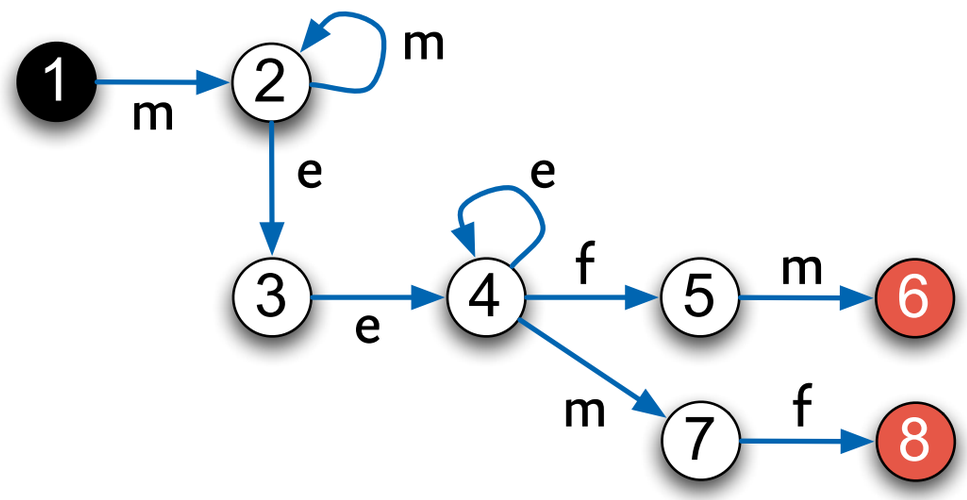
we ∈ {m,e,f}

th ∈ {m,e,f}

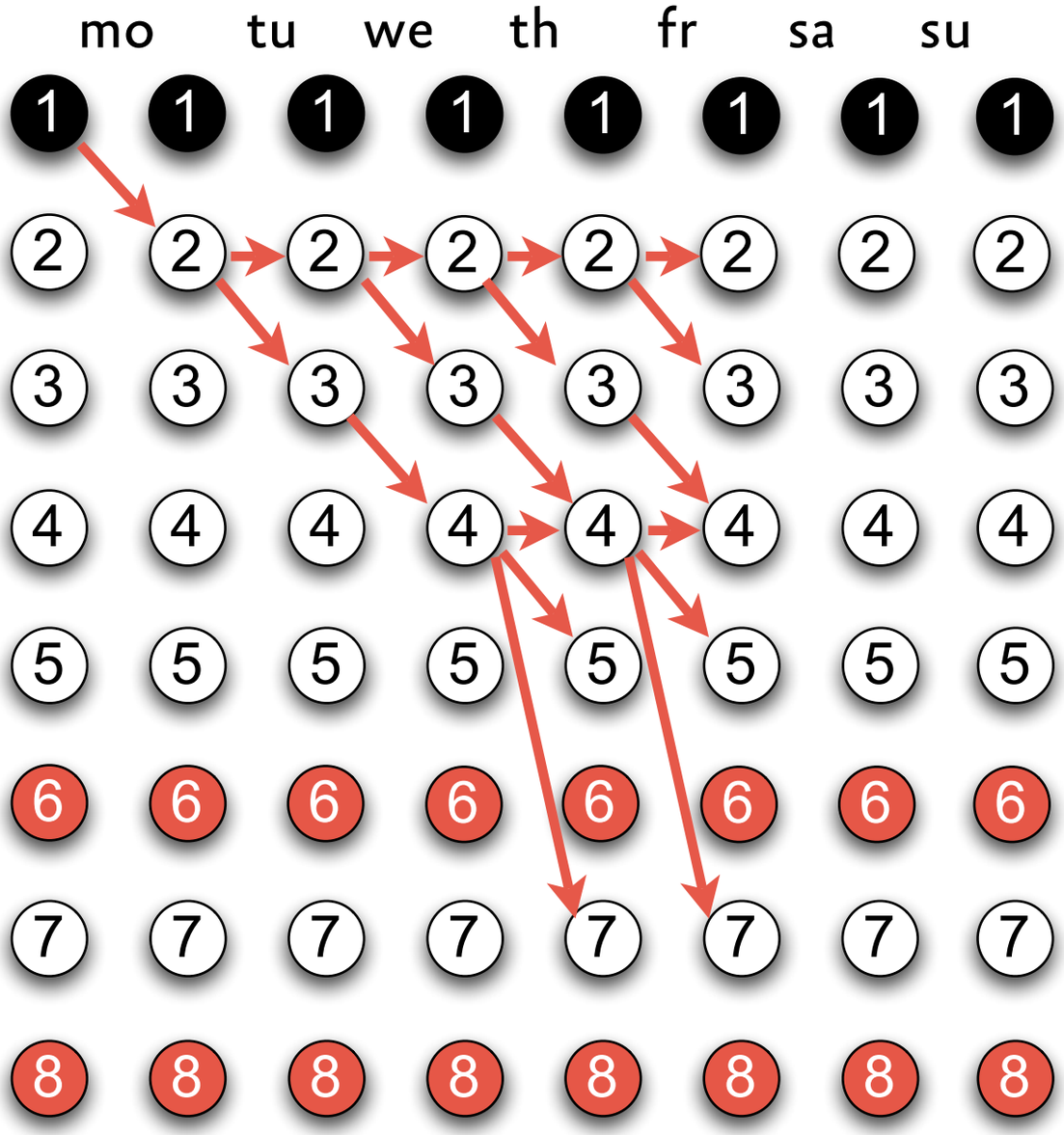
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

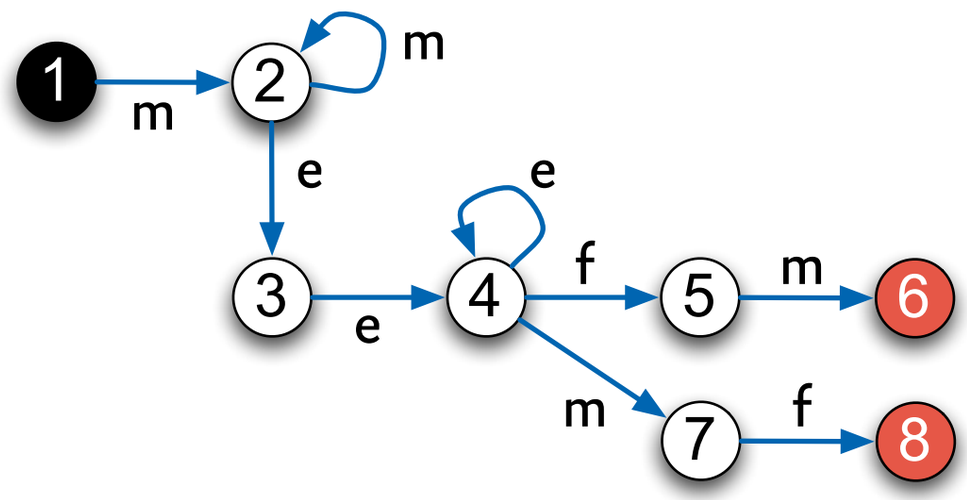
we ∈ {m,e,f}

th ∈ {m,e,f}

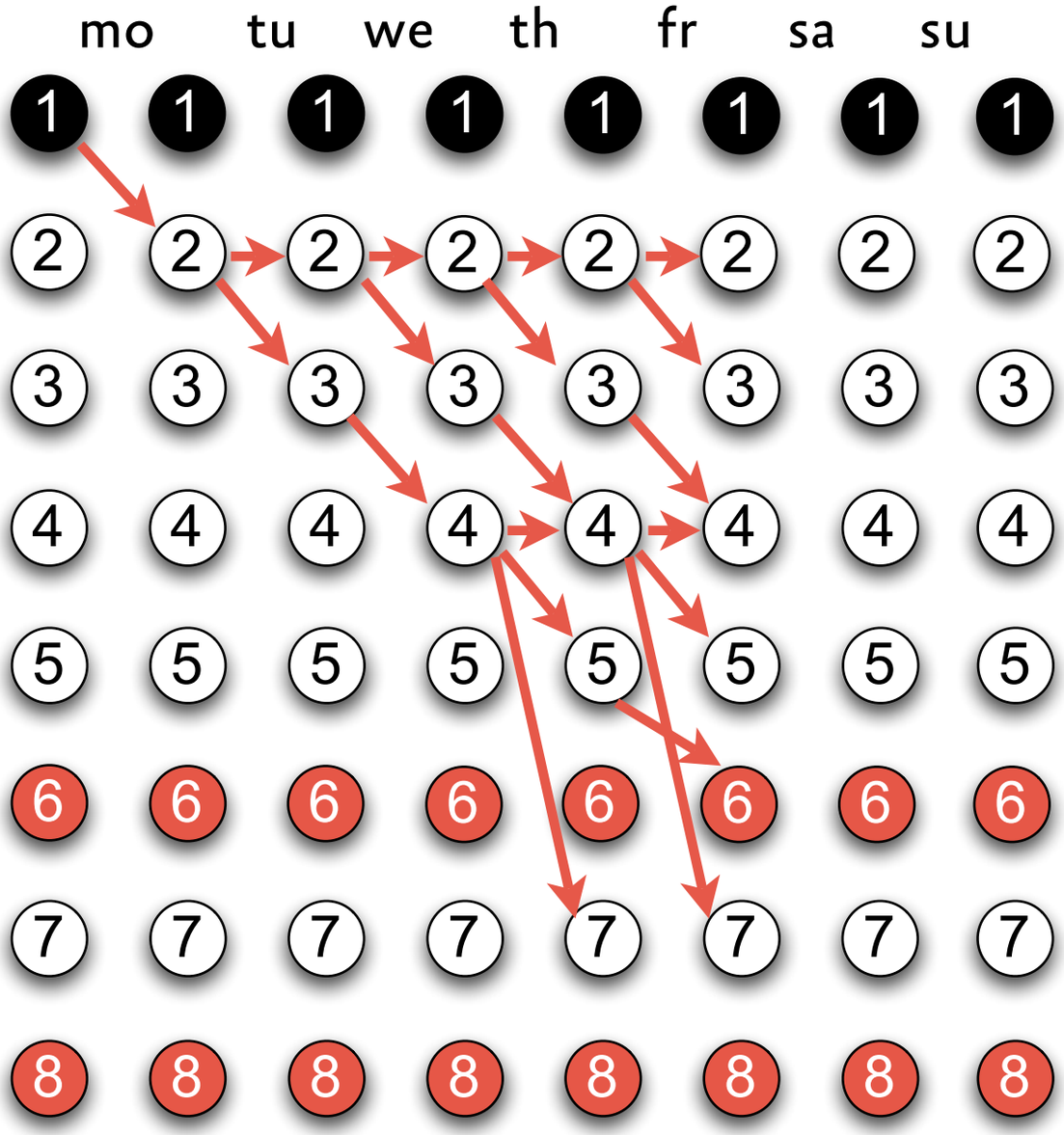
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

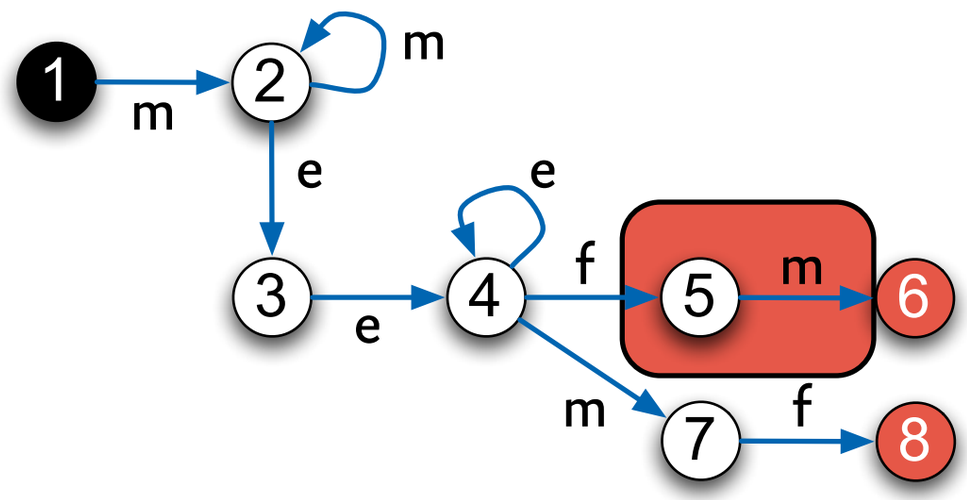
we ∈ {m,e,f}

th ∈ {m,e,f}

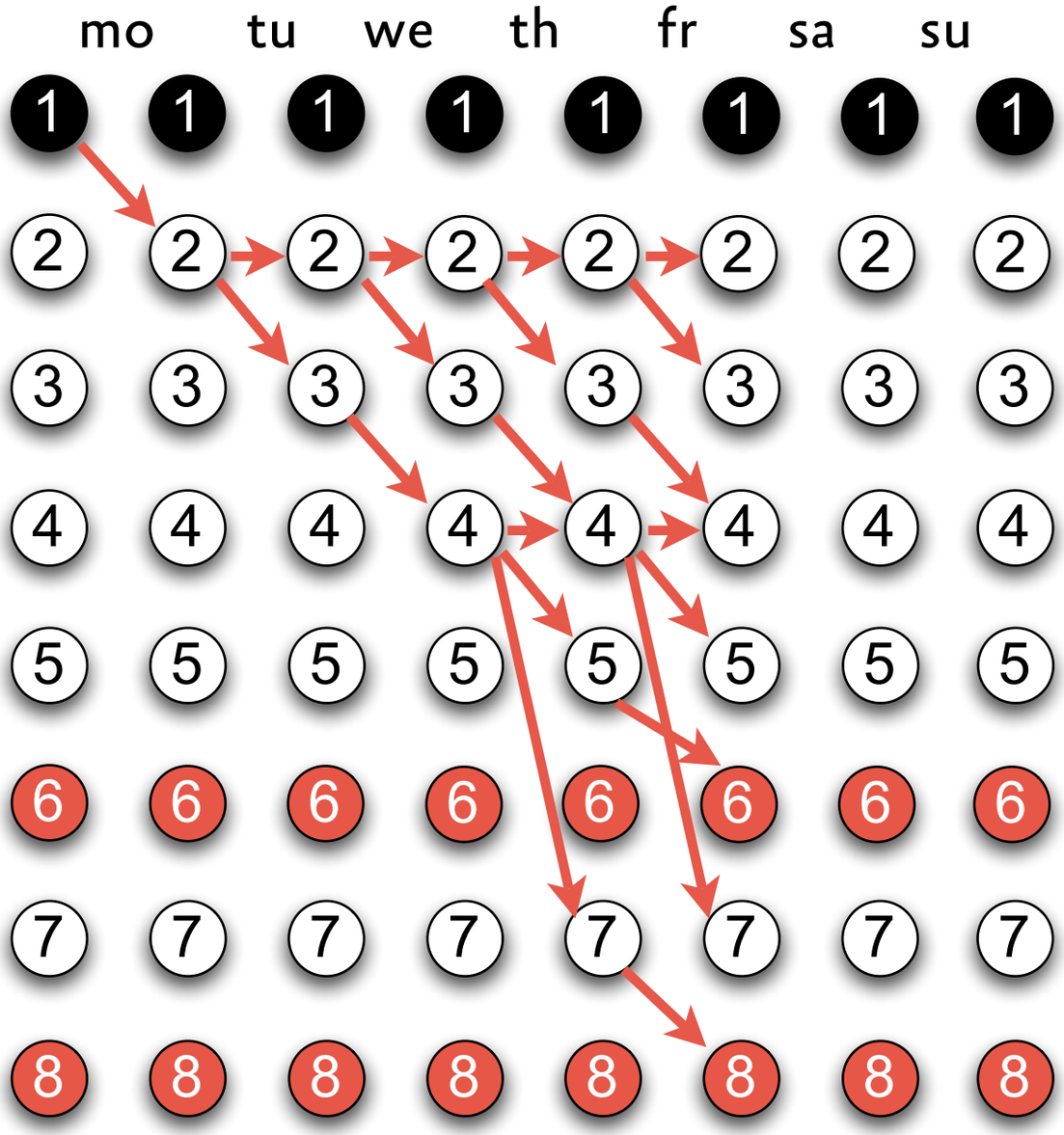
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

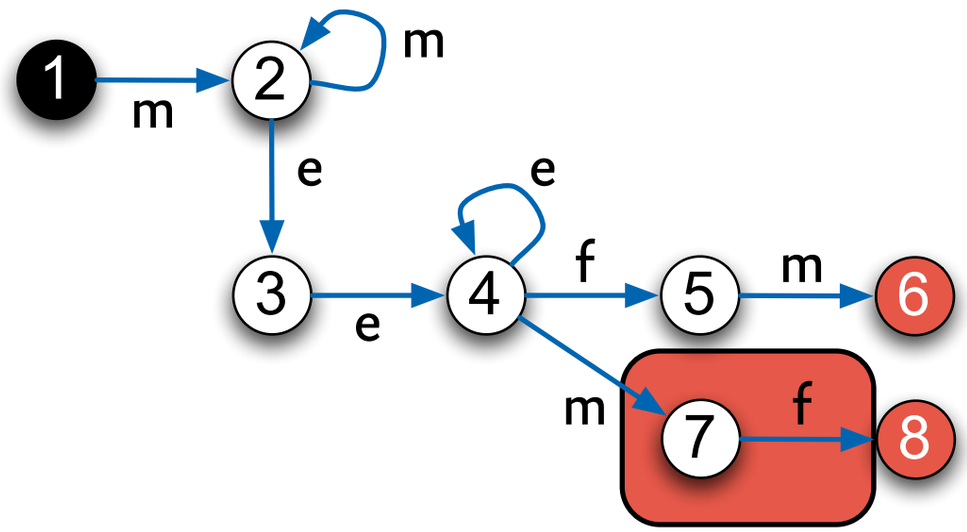
we ∈ {m,e,f}

th ∈ {m,e,f}

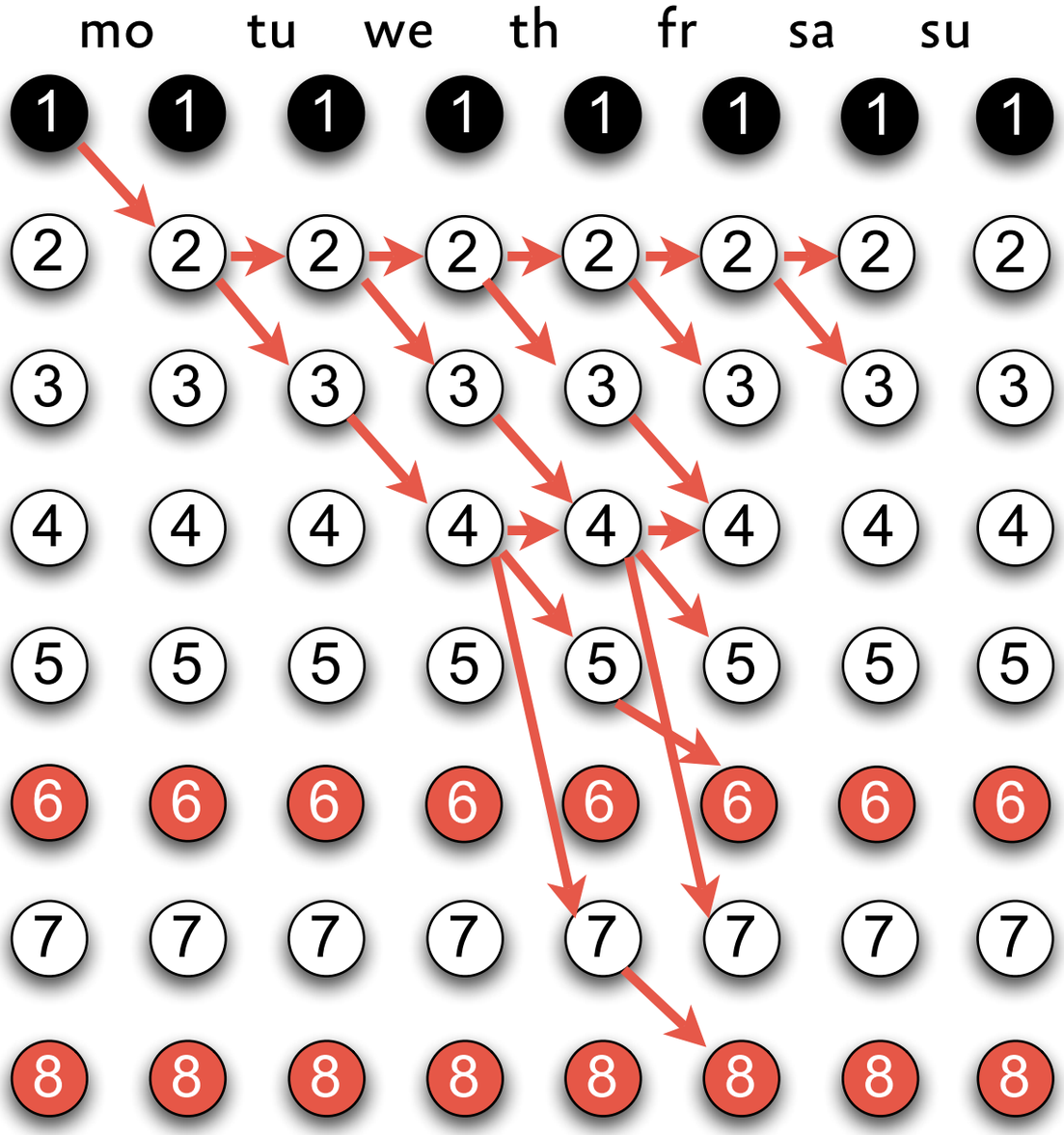
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

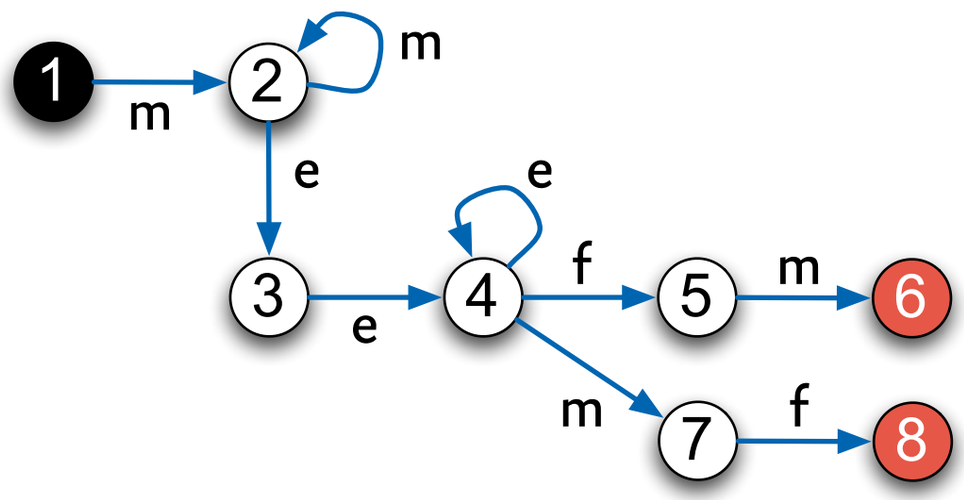
we ∈ {m,e,f}

th ∈ {m,e,f}

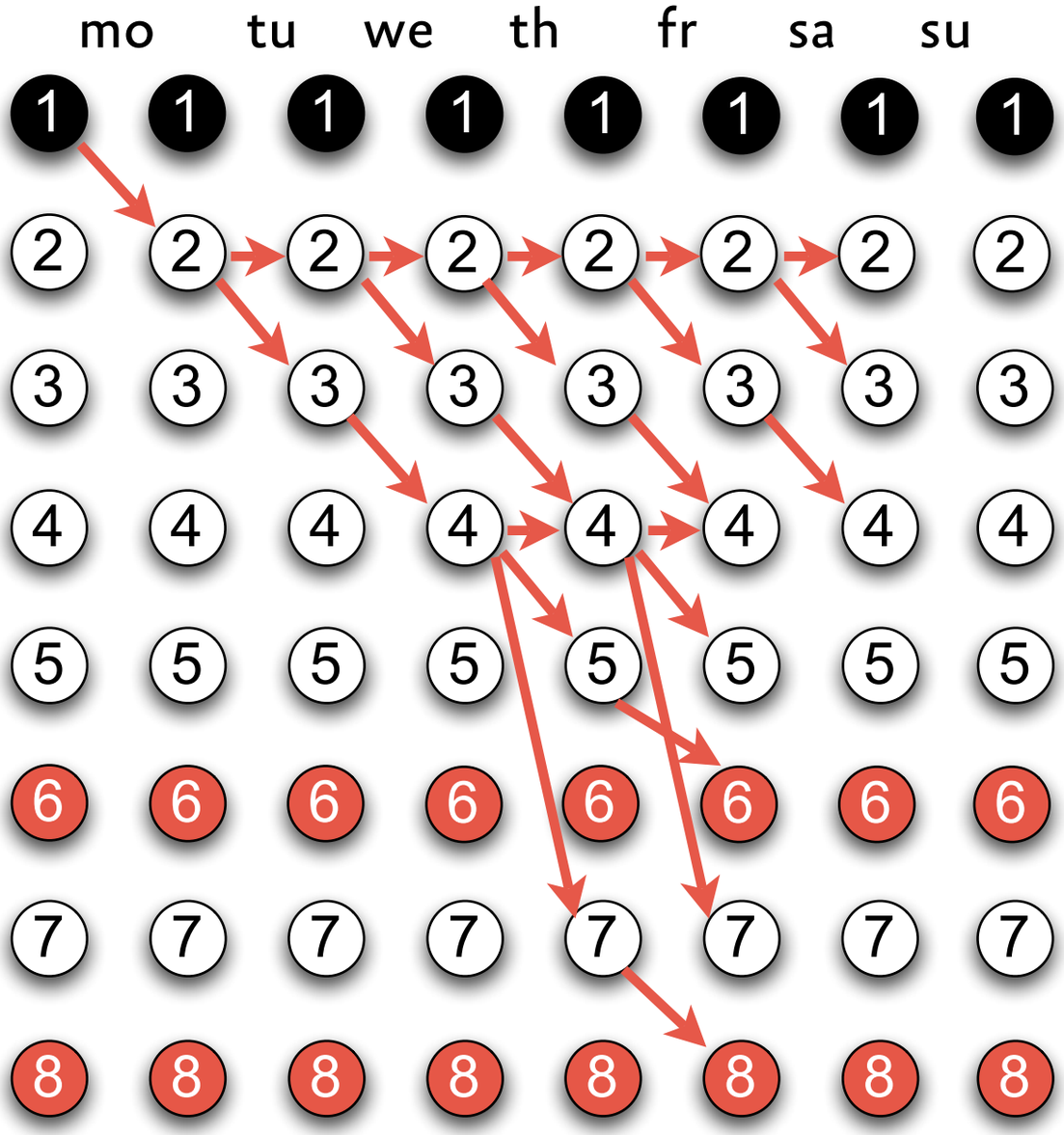
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

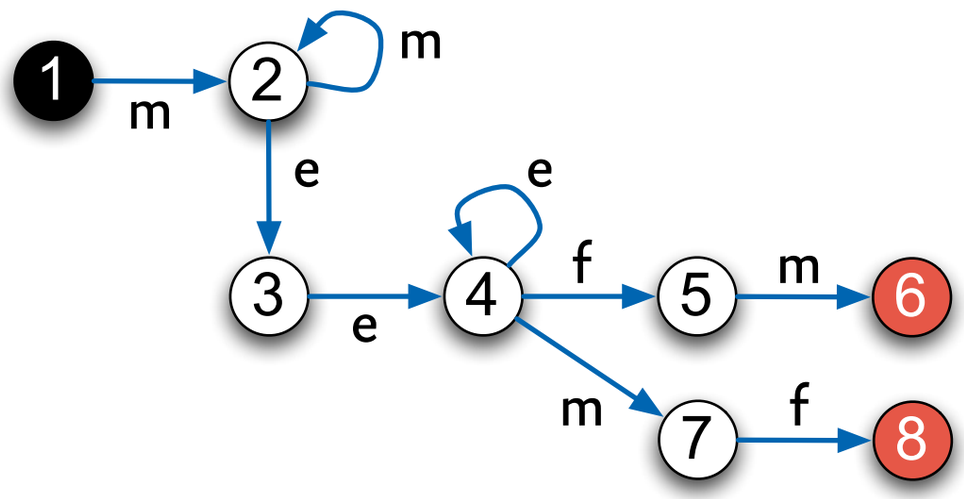
we ∈ {m,e,f}

th ∈ {m,e,f}

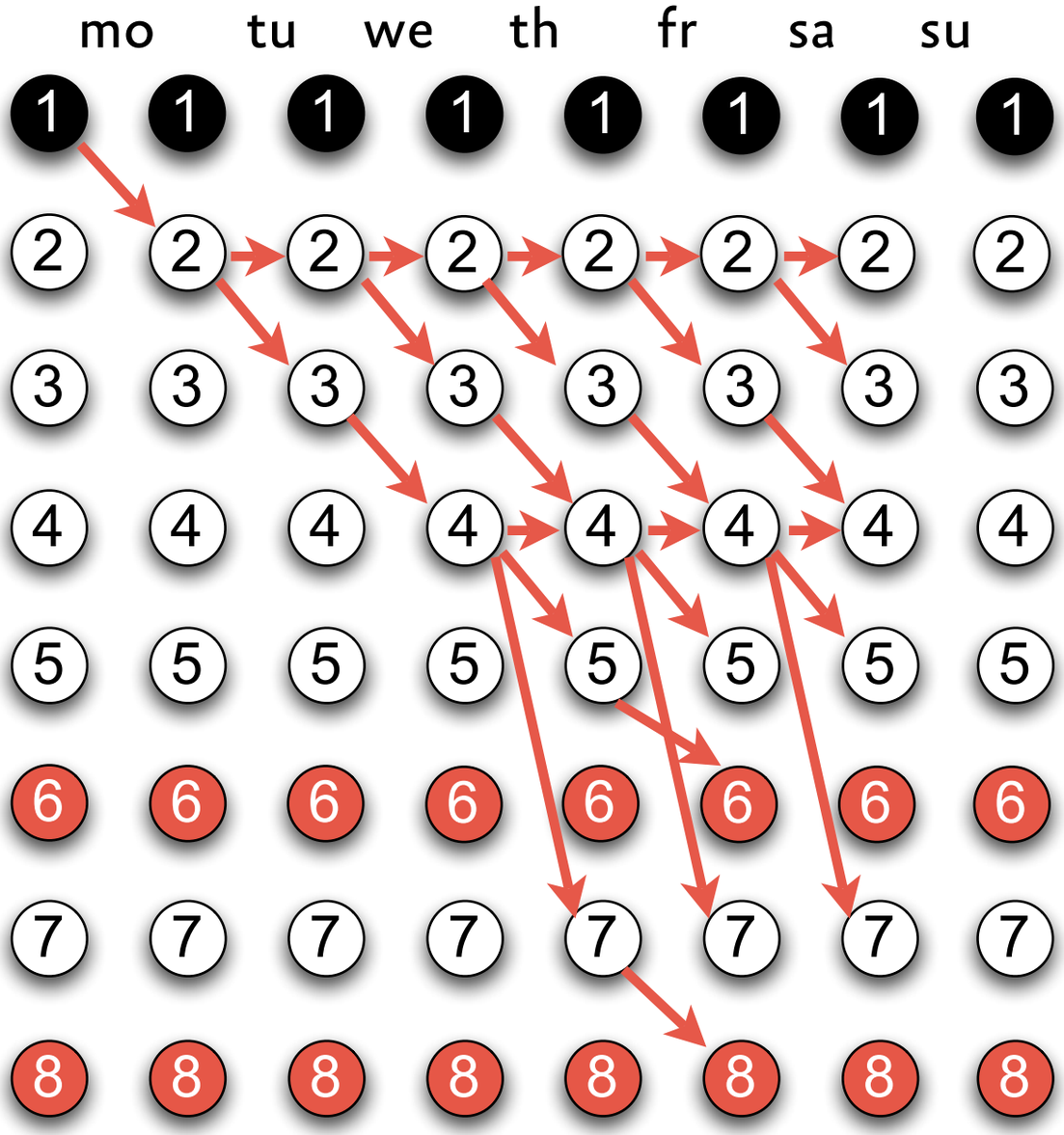
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

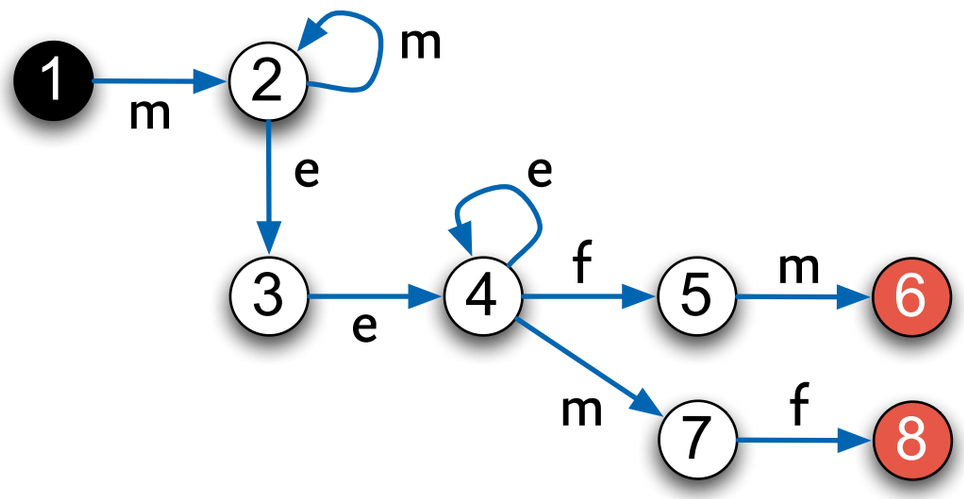
we ∈ {m,e,f}

th ∈ {m,e,f}

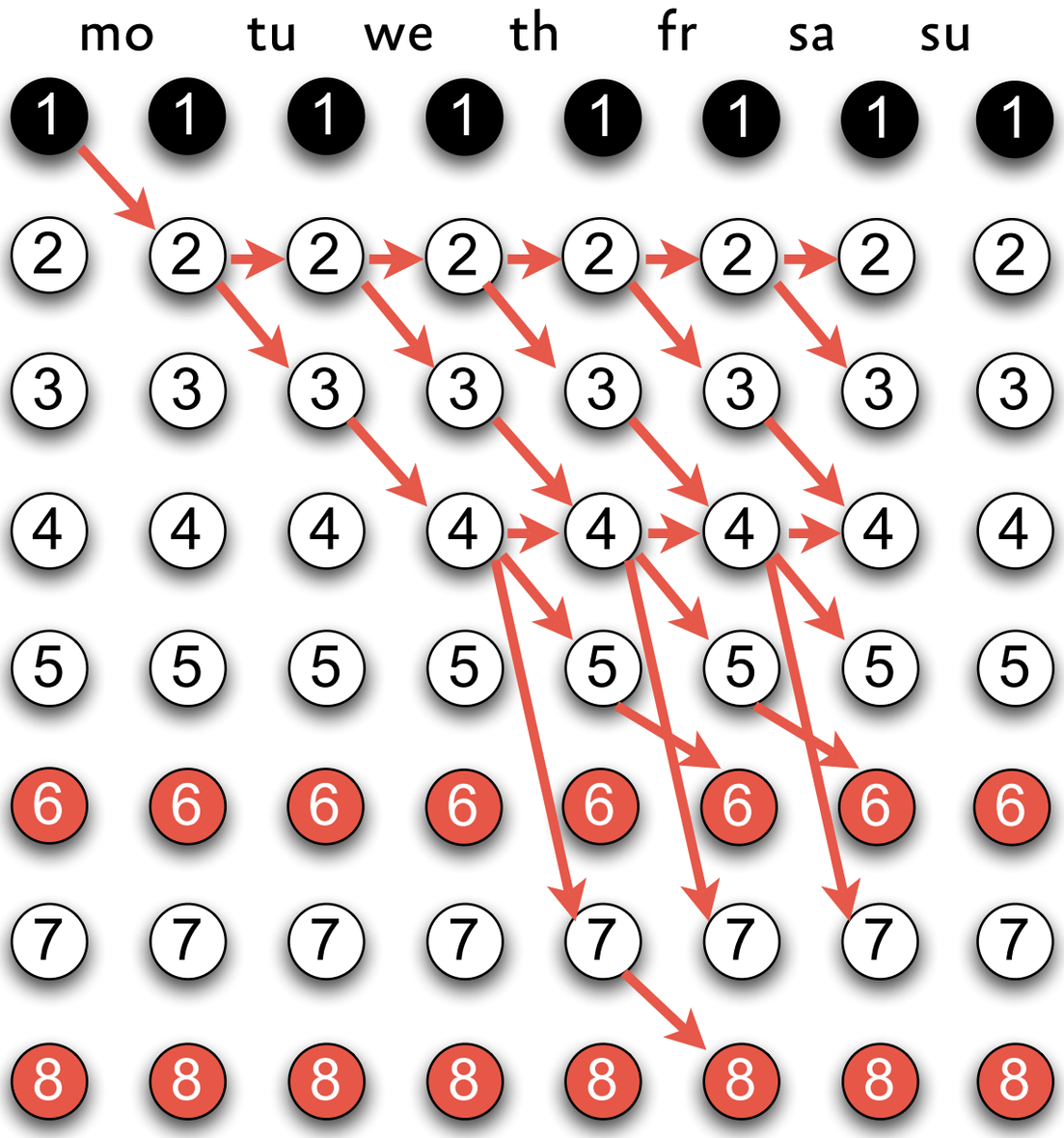
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

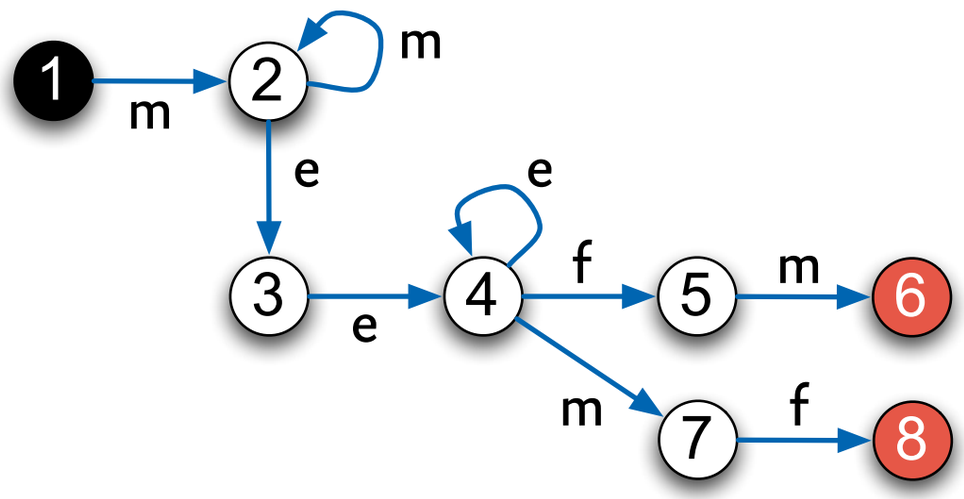
we ∈ {m,e,f}

th ∈ {m,e,f}

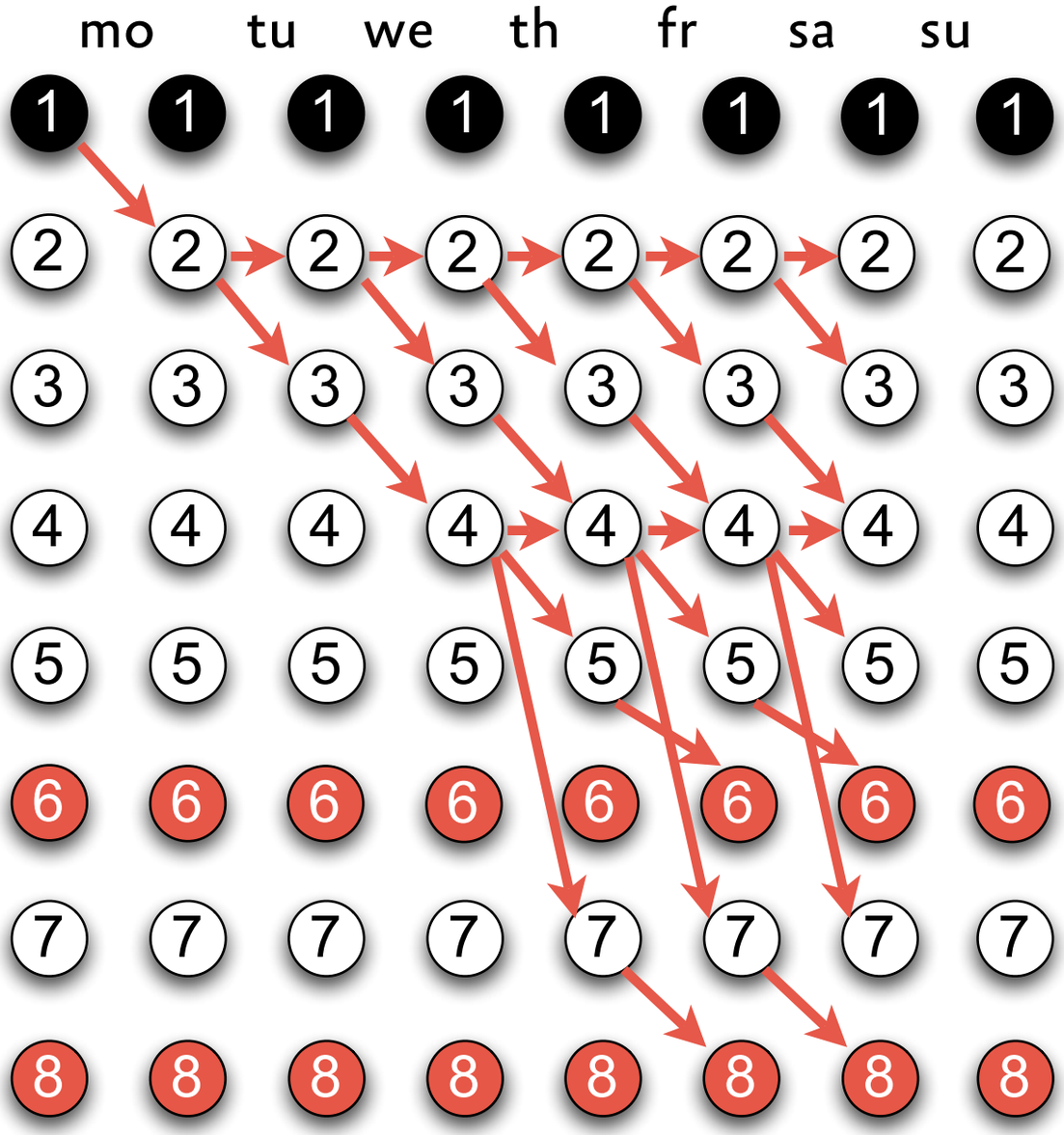
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

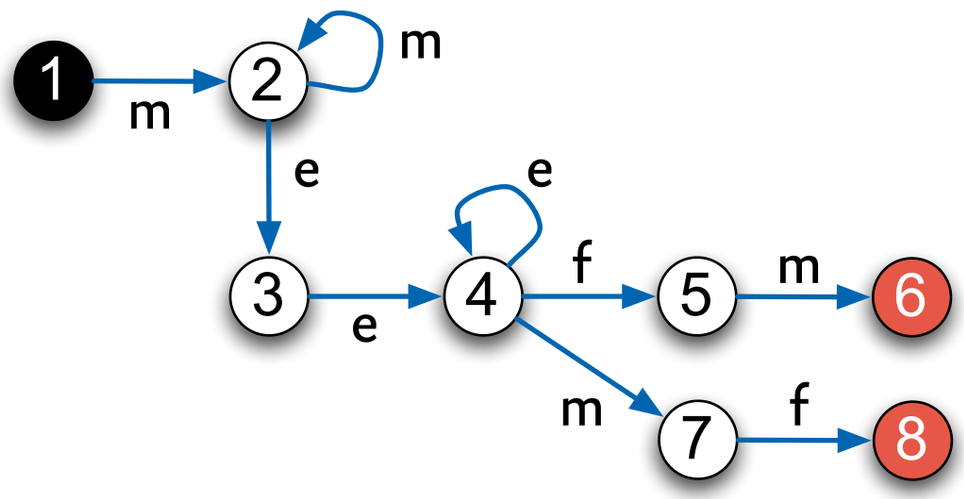
we ∈ {m,e,f}

th ∈ {m,e,f}

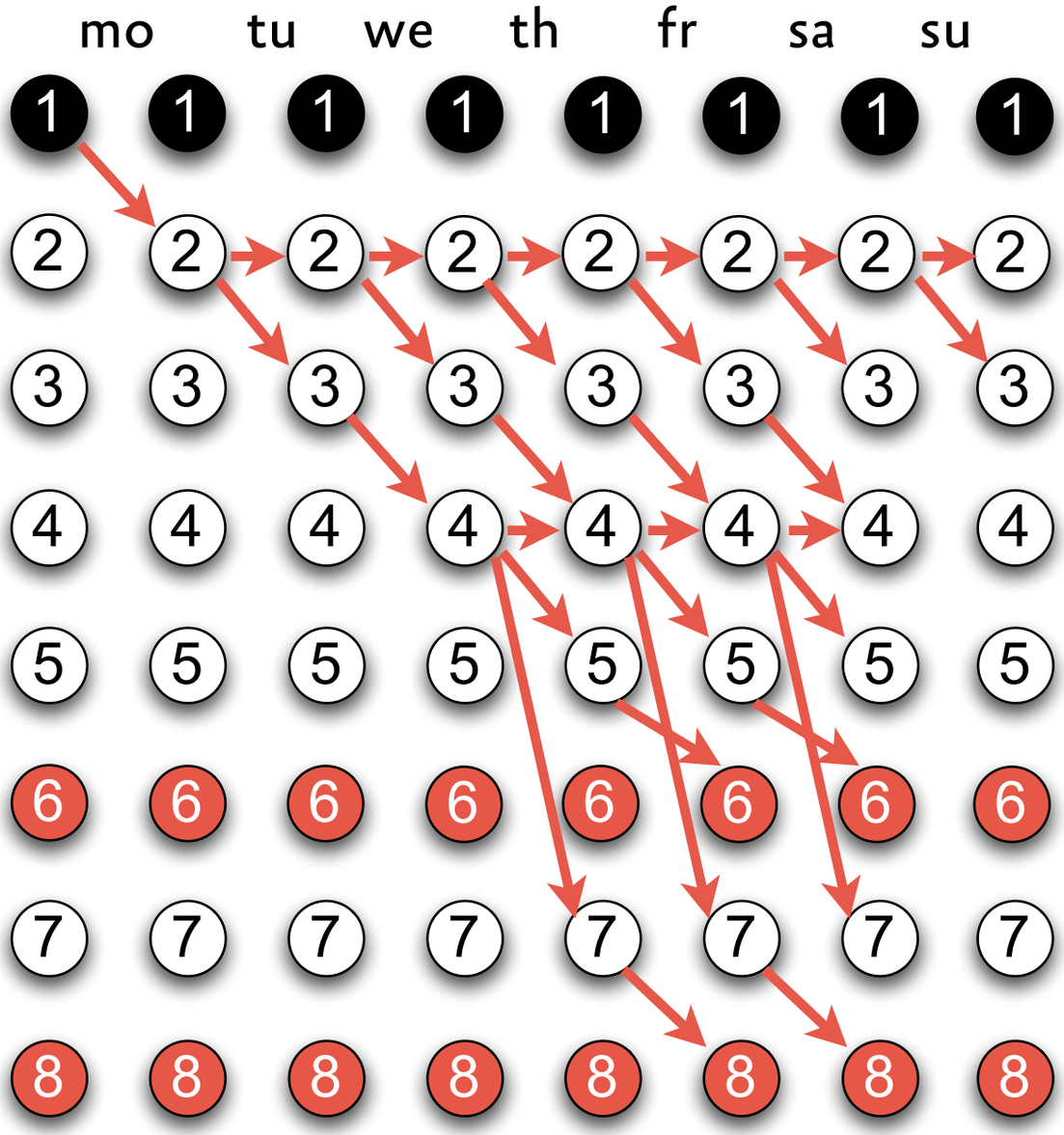
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

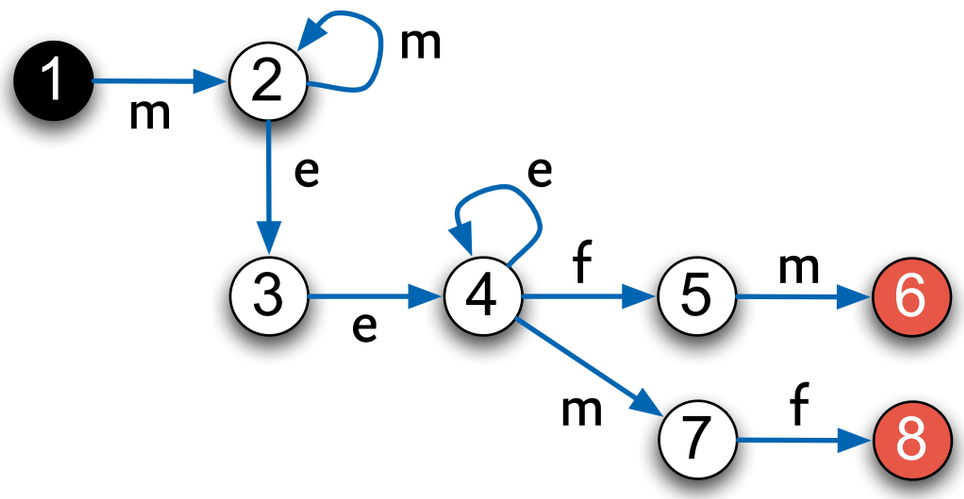
we ∈ {m,e,f}

th ∈ {m,e,f}

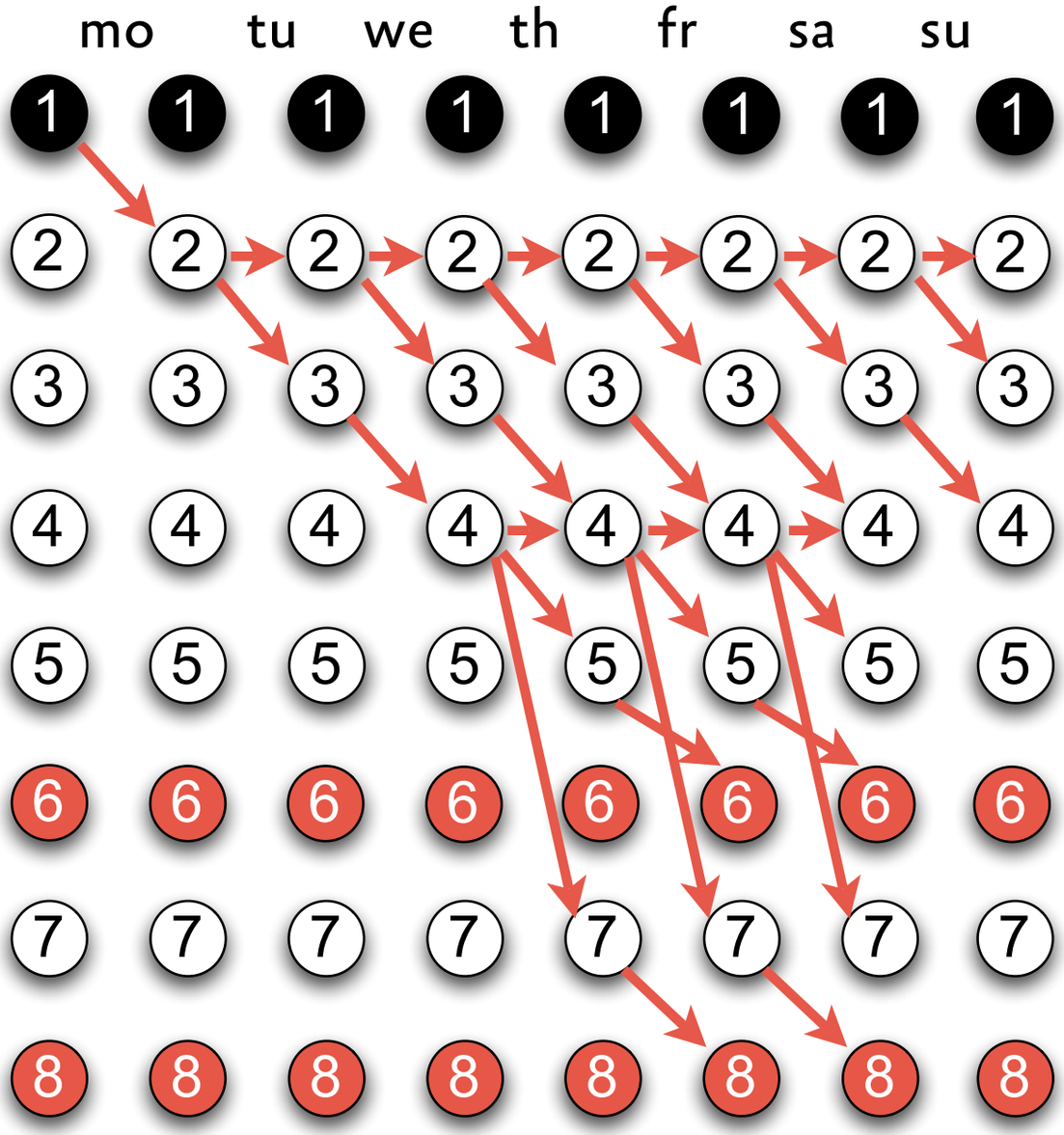
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

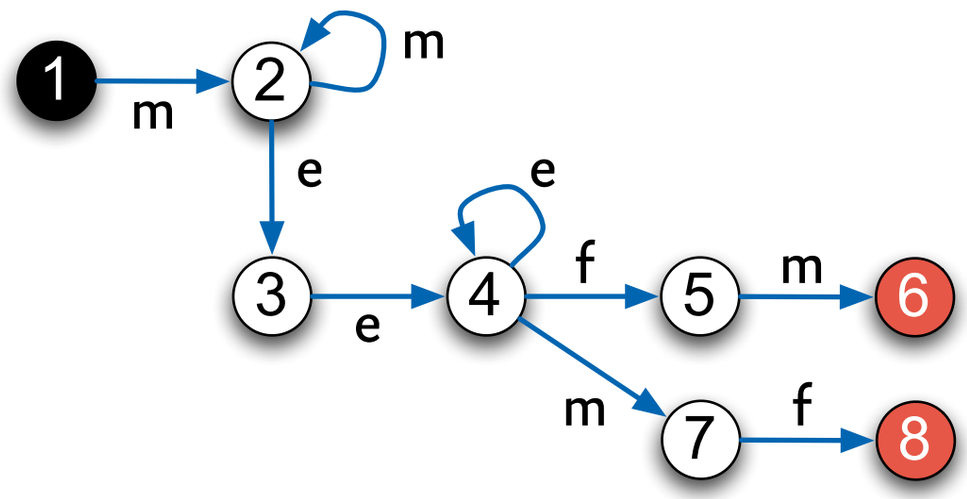
we ∈ {m,e,f}

th ∈ {m,e,f}

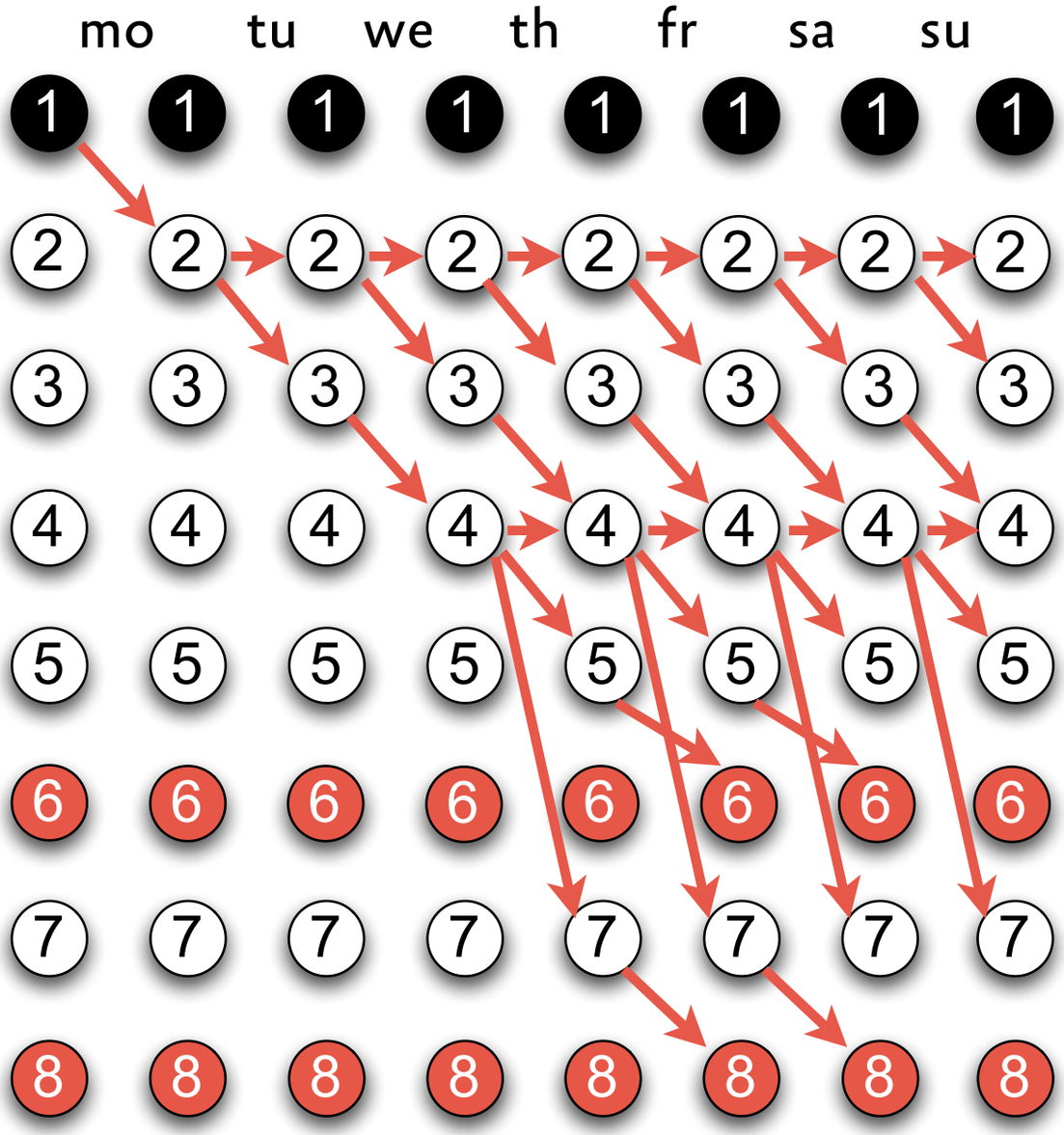
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

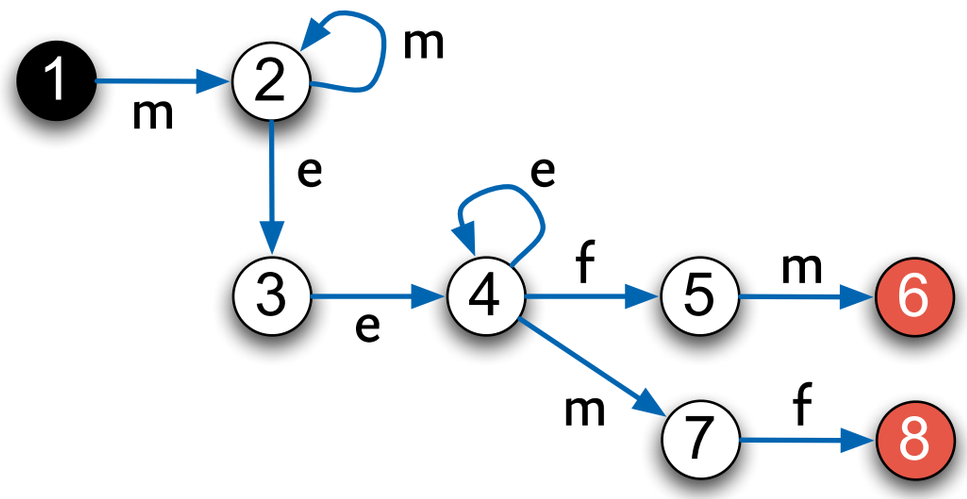
we ∈ {m,e,f}

th ∈ {m,e,f}

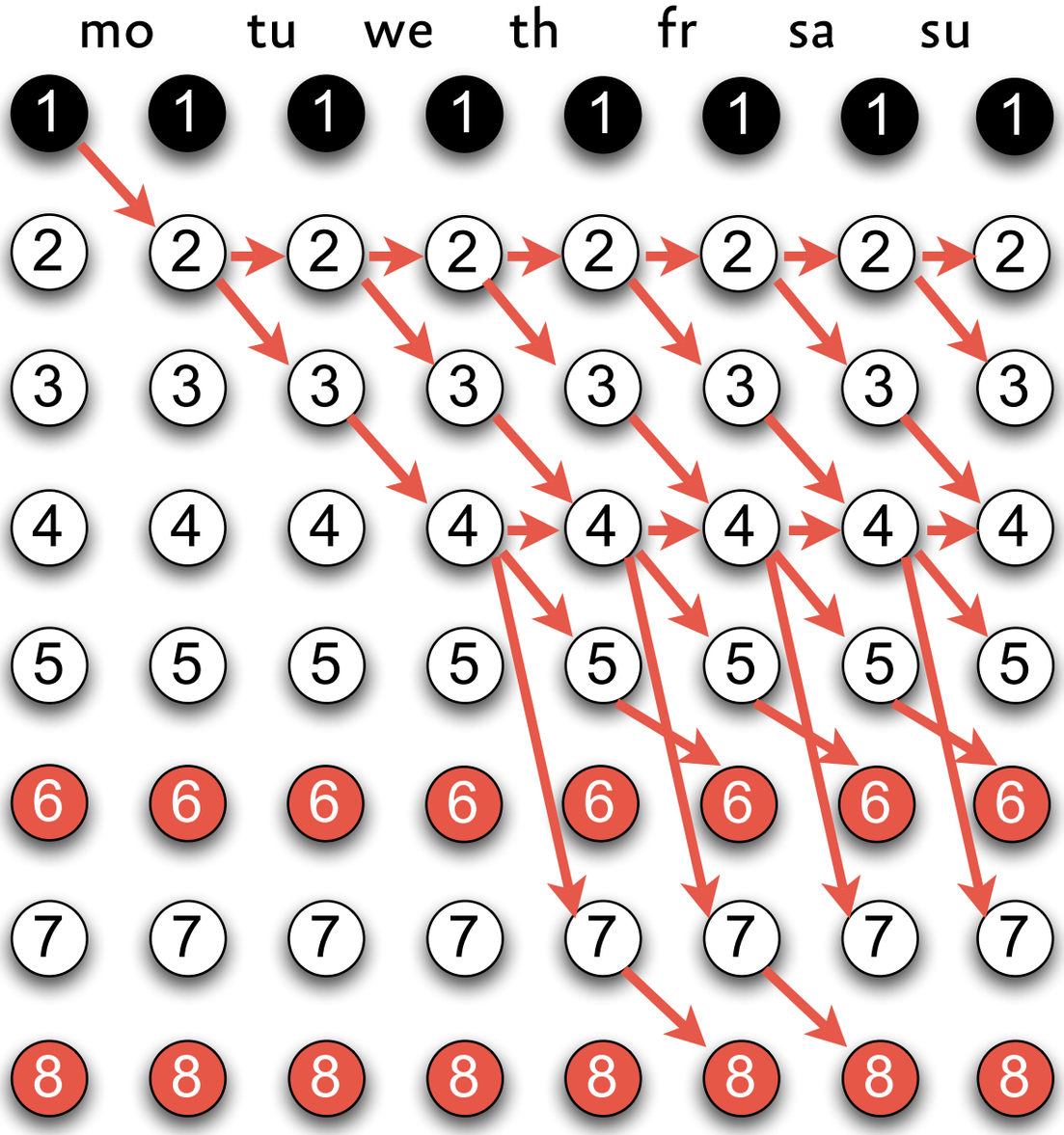
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

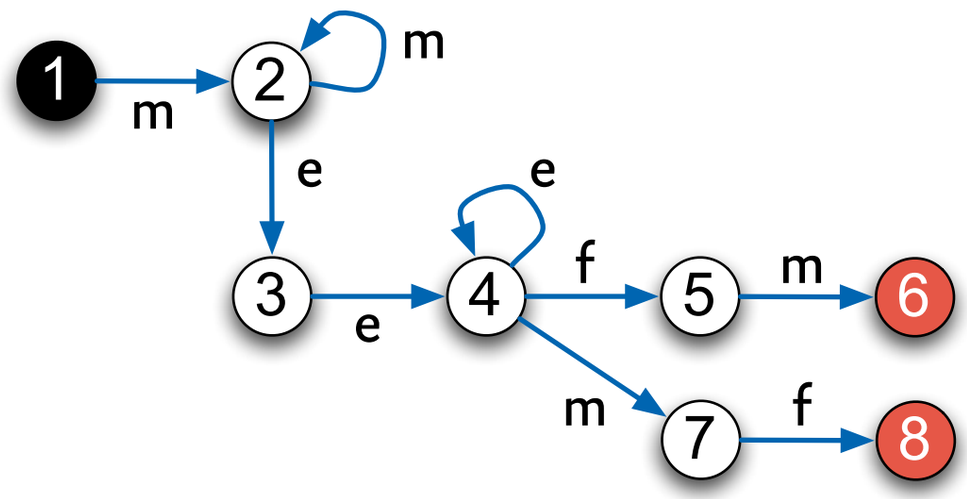
we ∈ {m,e,f}

th ∈ {m,e,f}

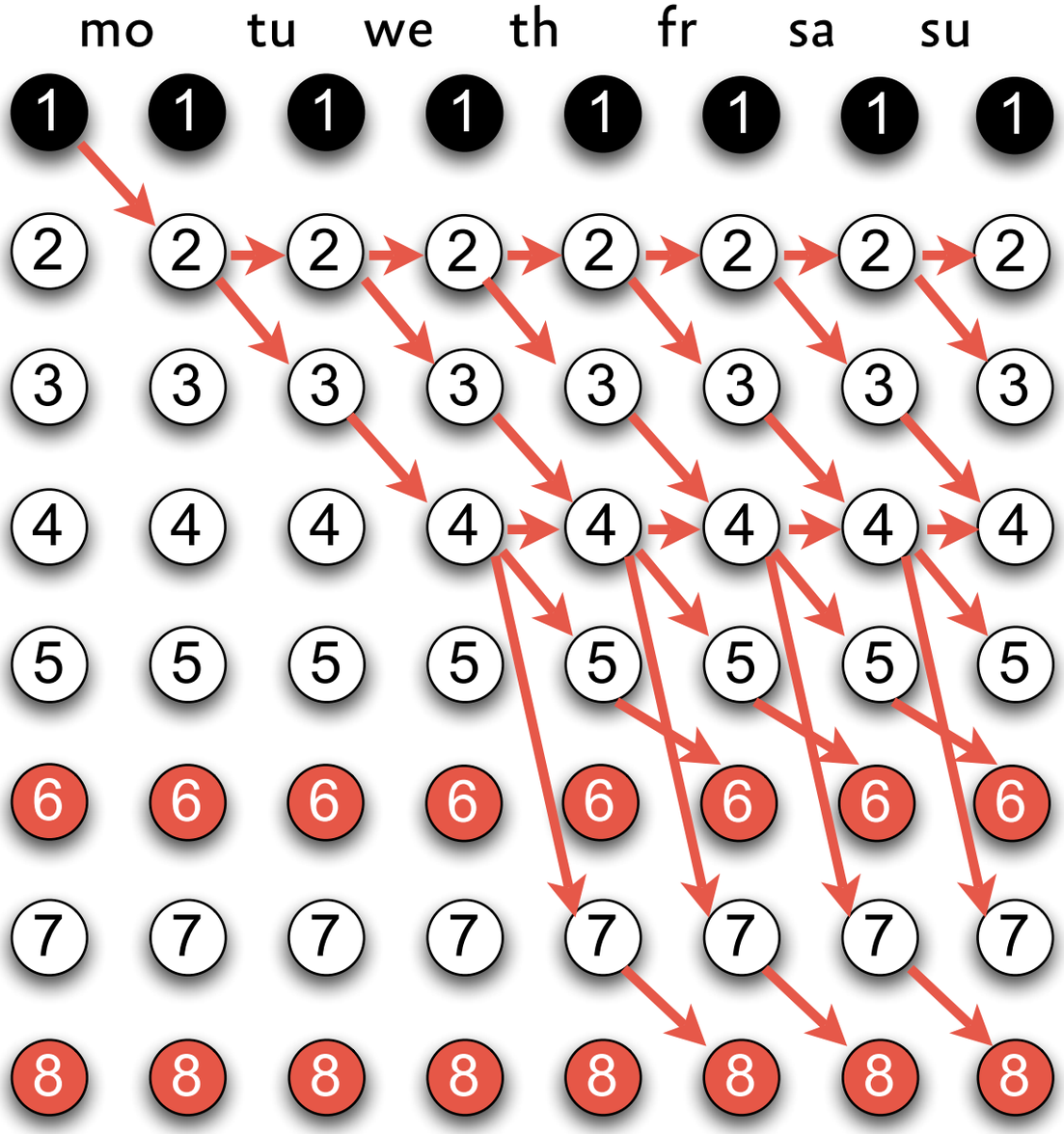
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



forward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

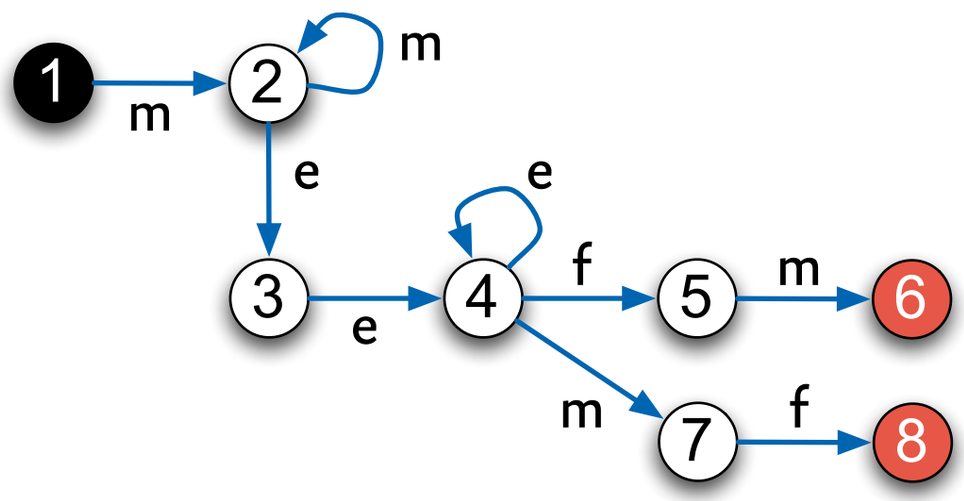
we ∈ {m,e,f}

th ∈ {m,e,f}

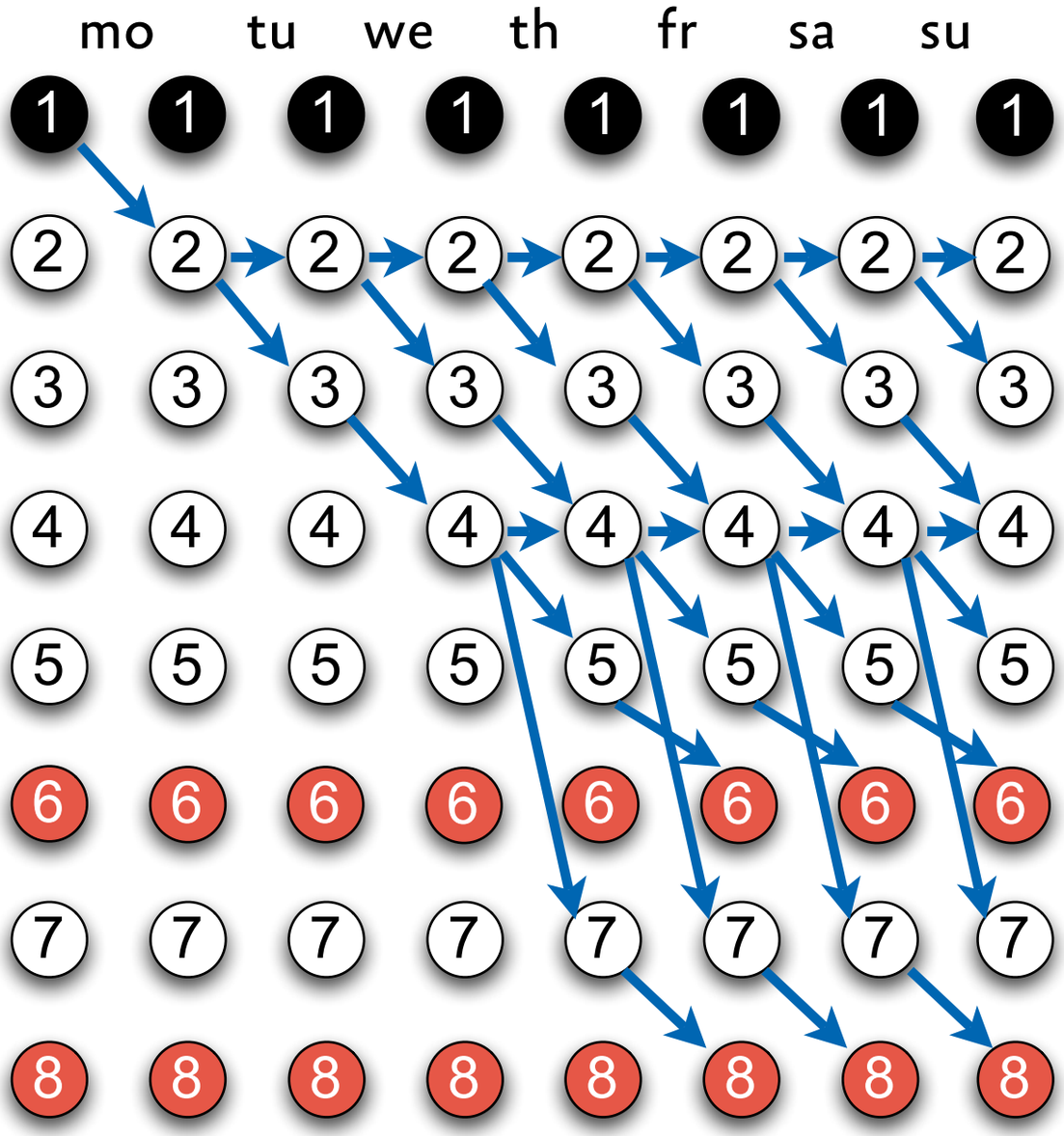
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



mo ∈ {m,e,f}

tu ∈ {m,e,f}

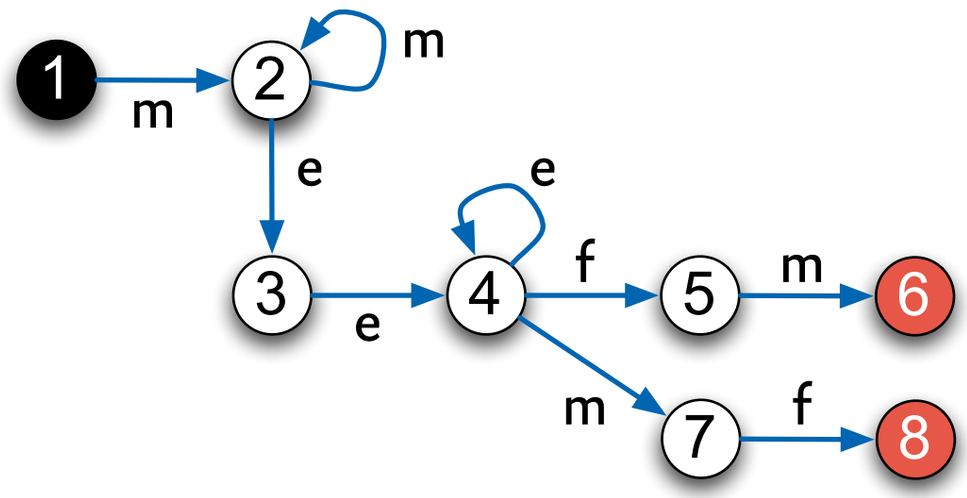
we ∈ {m,e,f}

th ∈ {m,e,f}

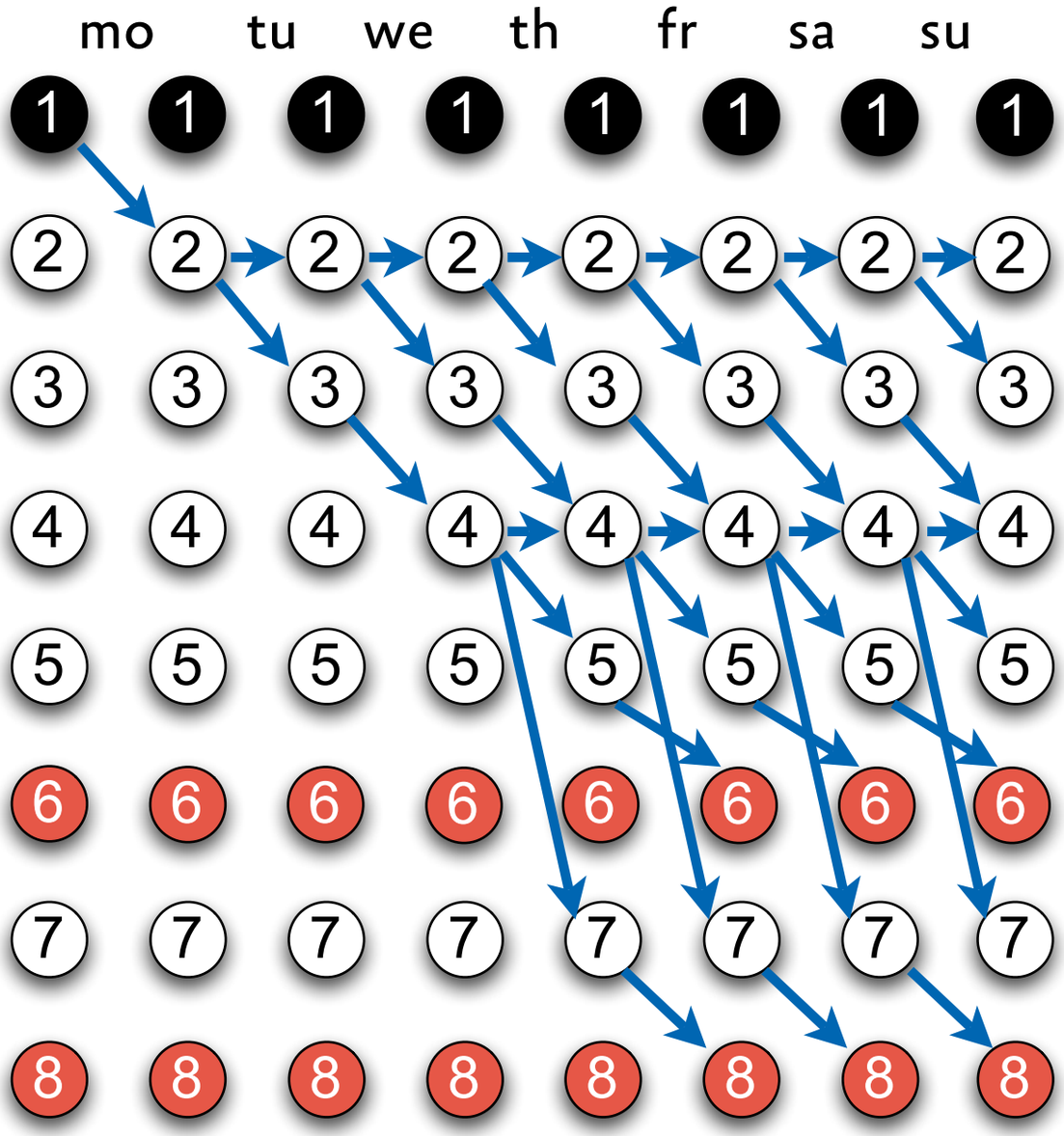
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

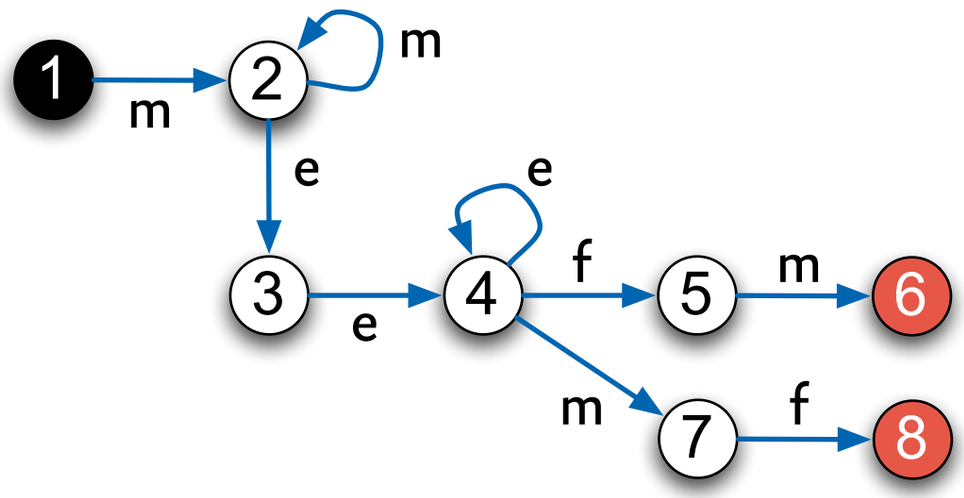
we ∈ {m,e,f}

th ∈ {m,e,f}

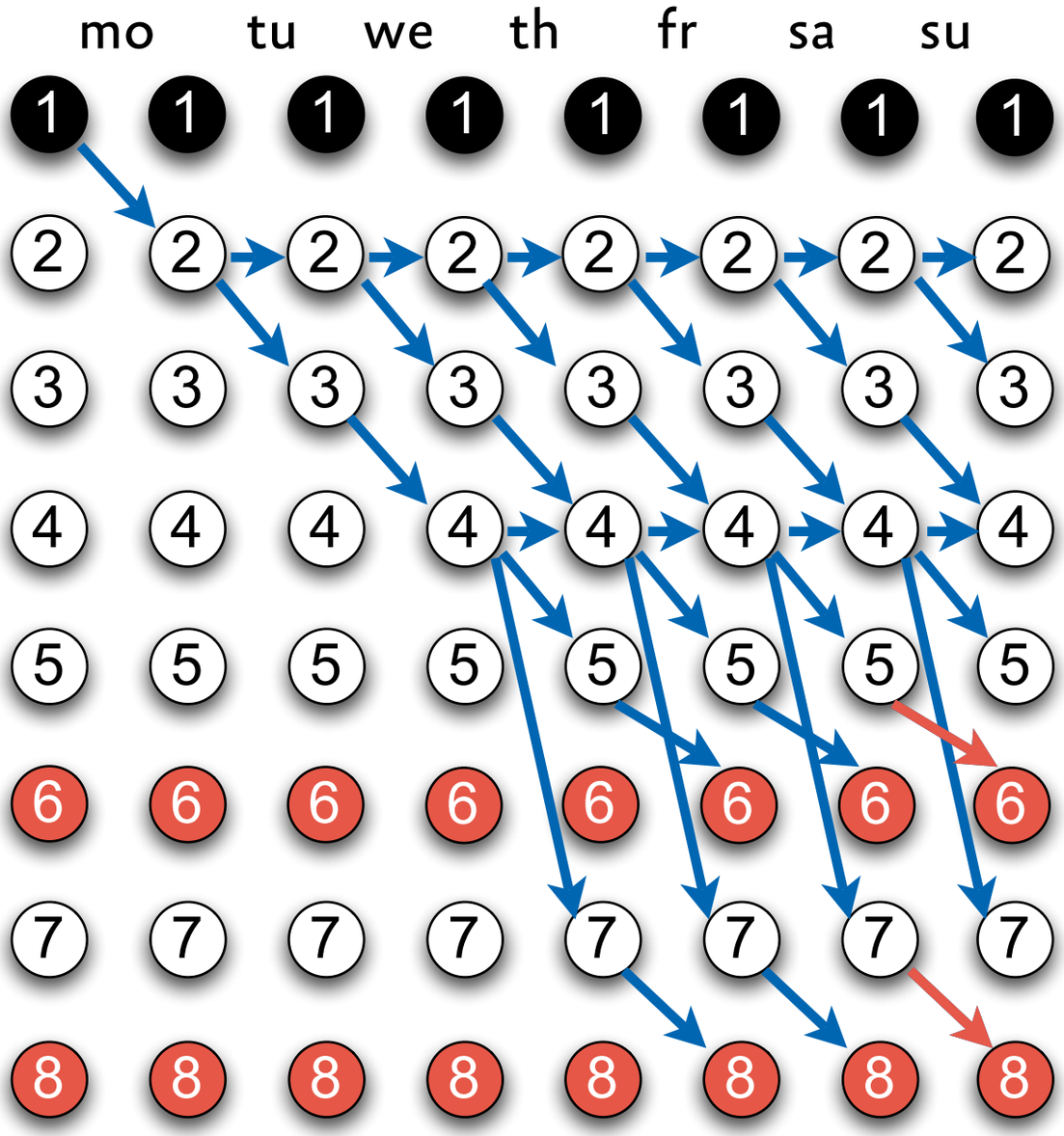
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

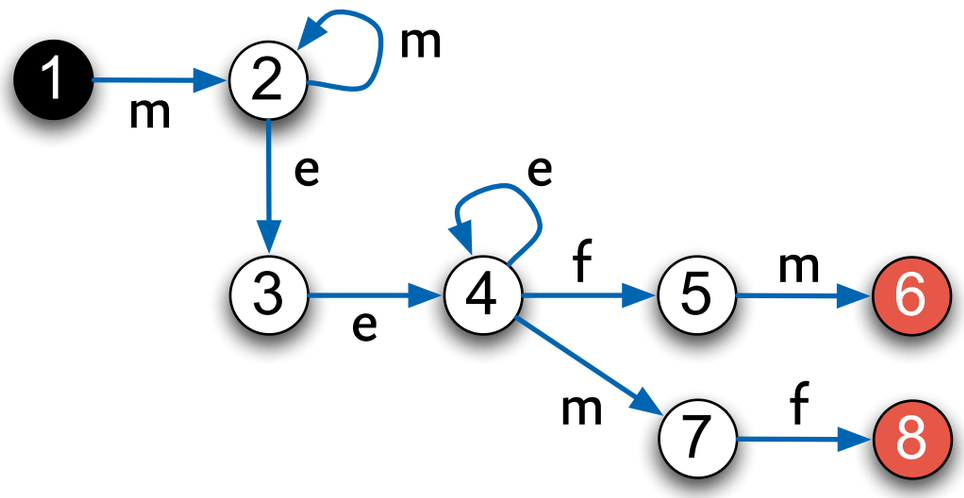
we ∈ {m,e,f}

th ∈ {m,e,f}

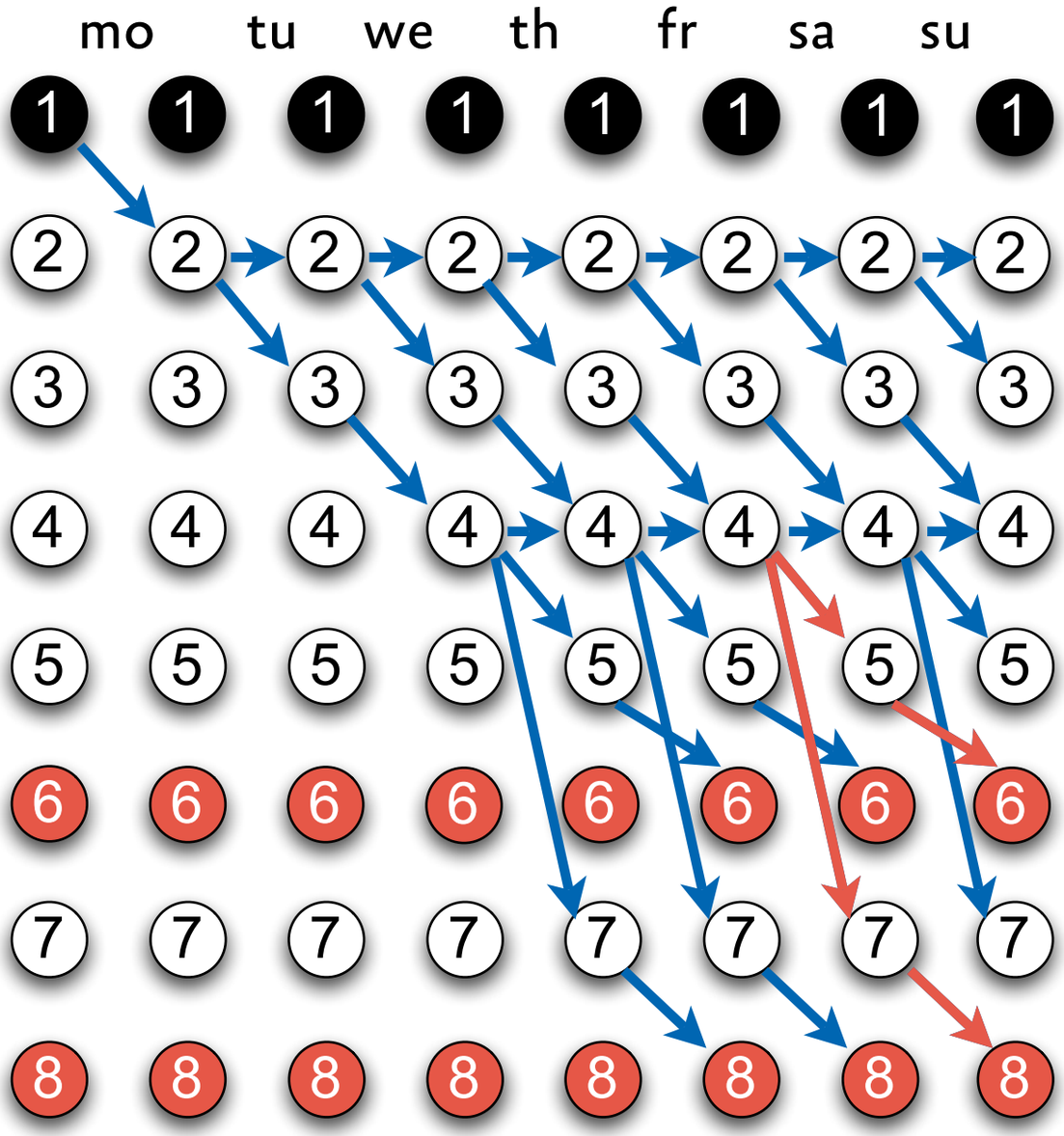
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

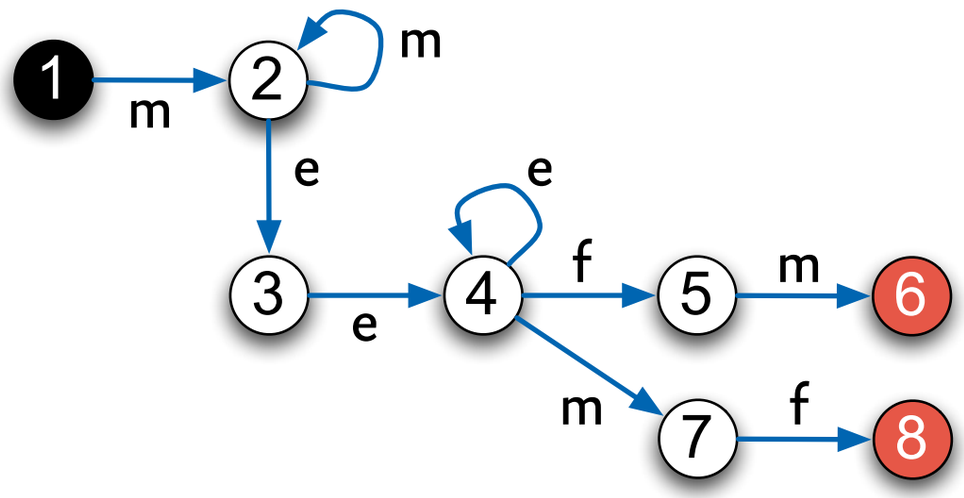
we ∈ {m,e,f}

th ∈ {m,e,f}

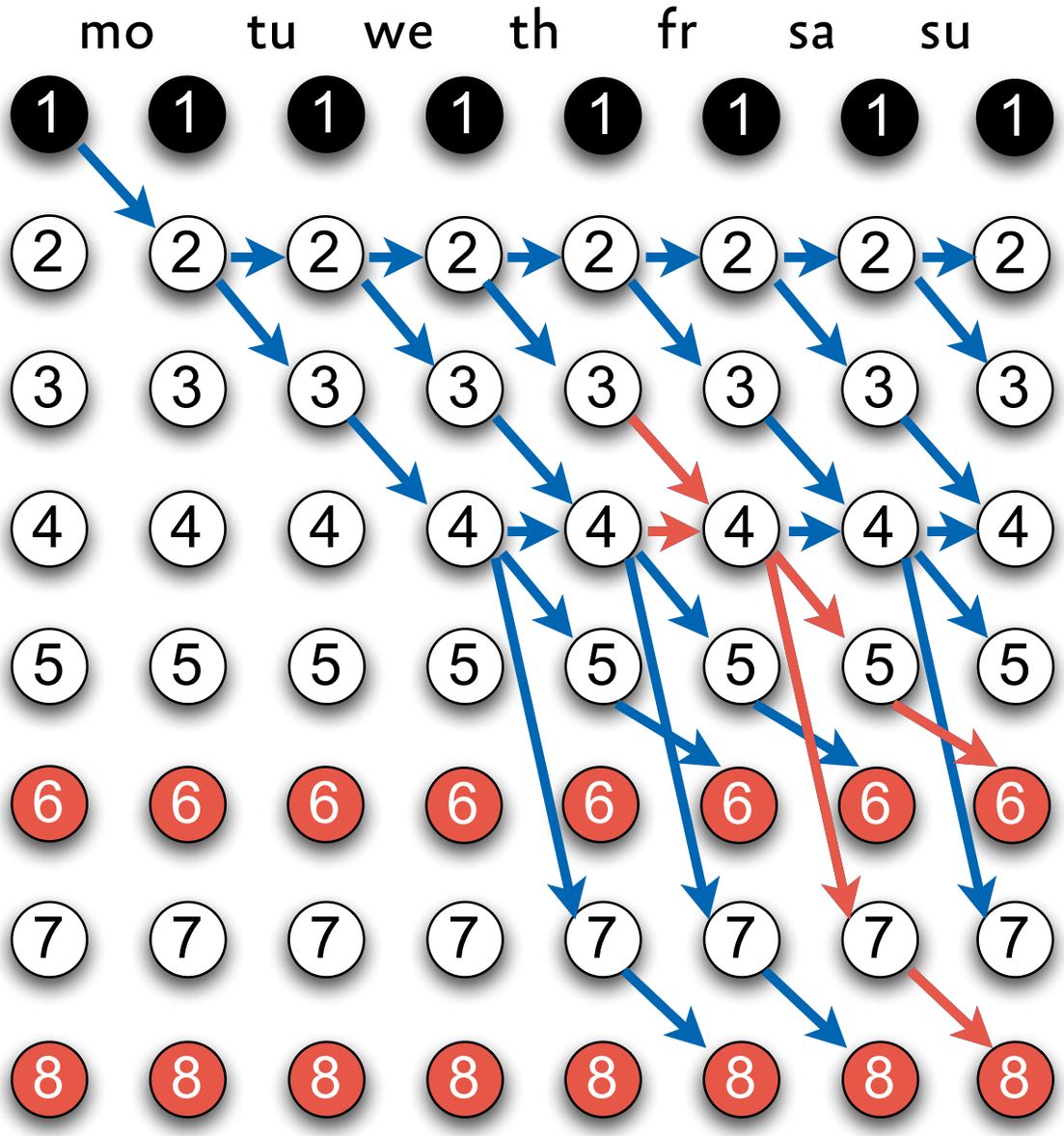
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

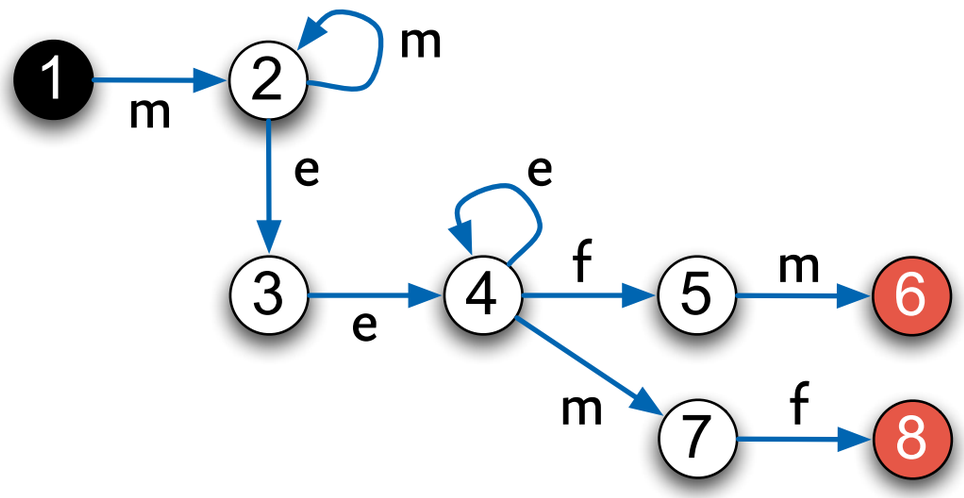
we ∈ {m,e,f}

th ∈ {m,e,f}

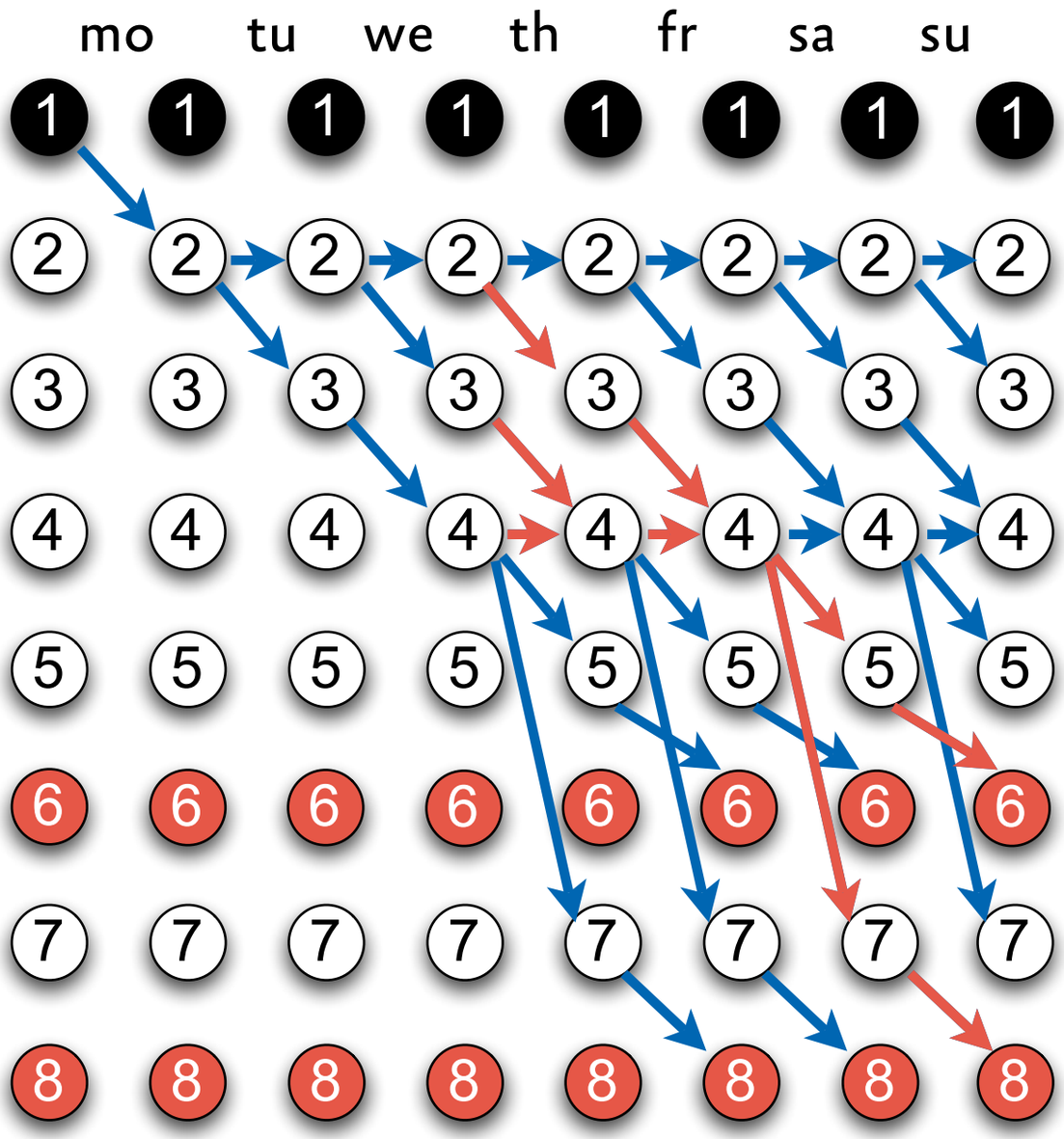
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

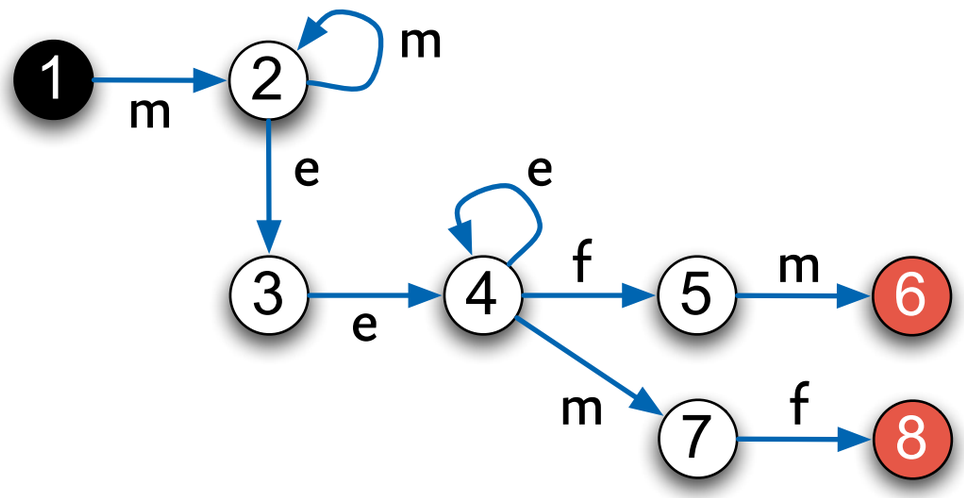
we ∈ {m,e,f}

th ∈ {m,e,f}

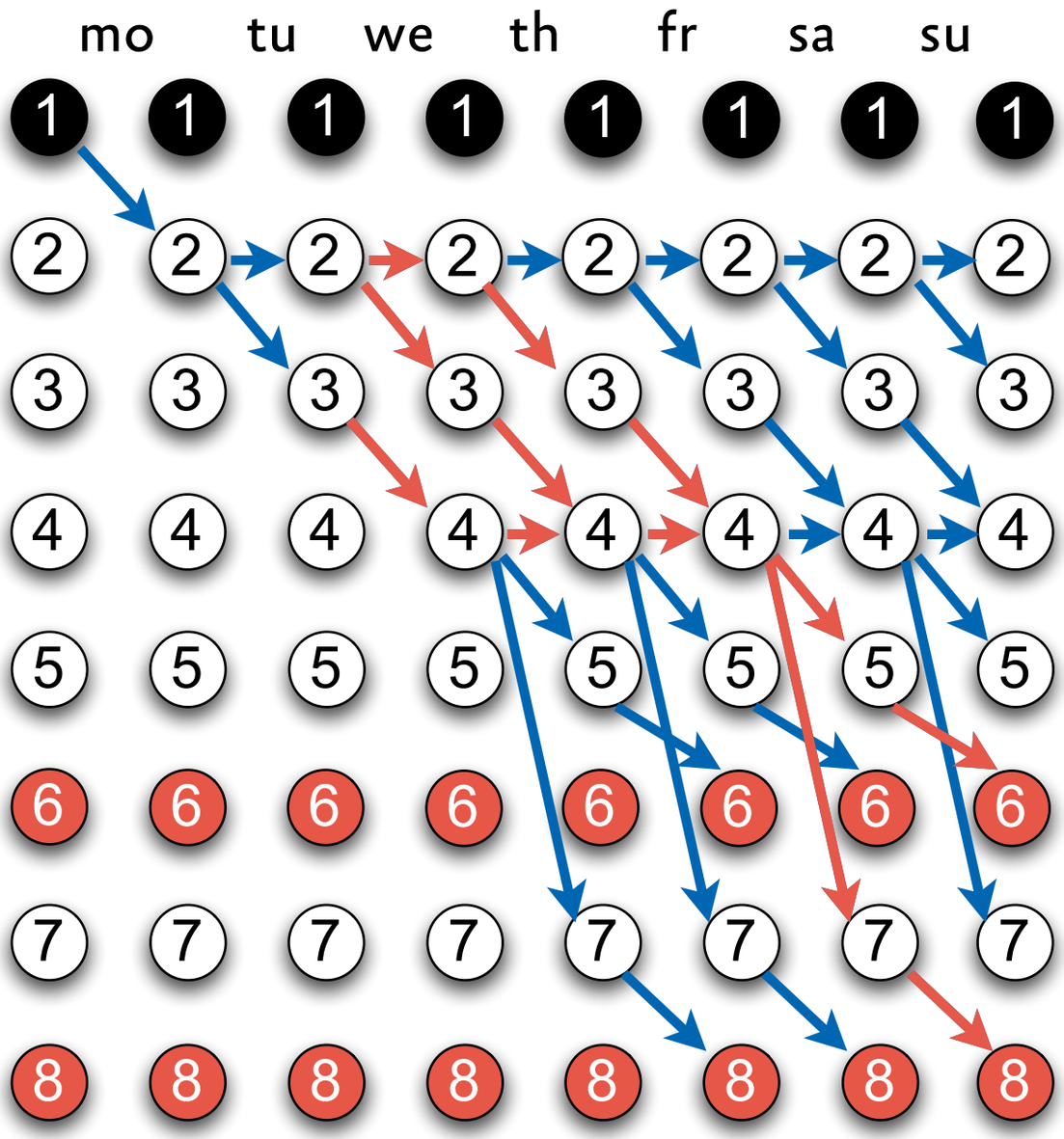
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

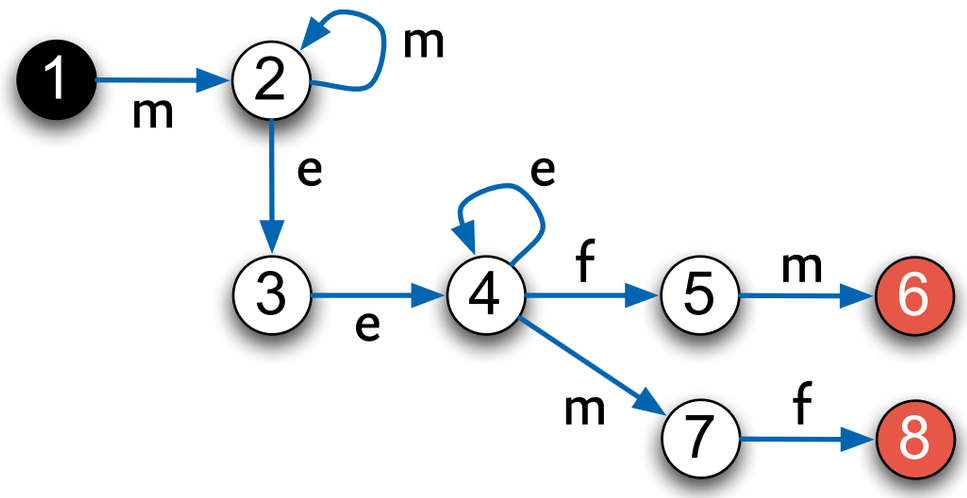
we ∈ {m,e,f}

th ∈ {m,e,f}

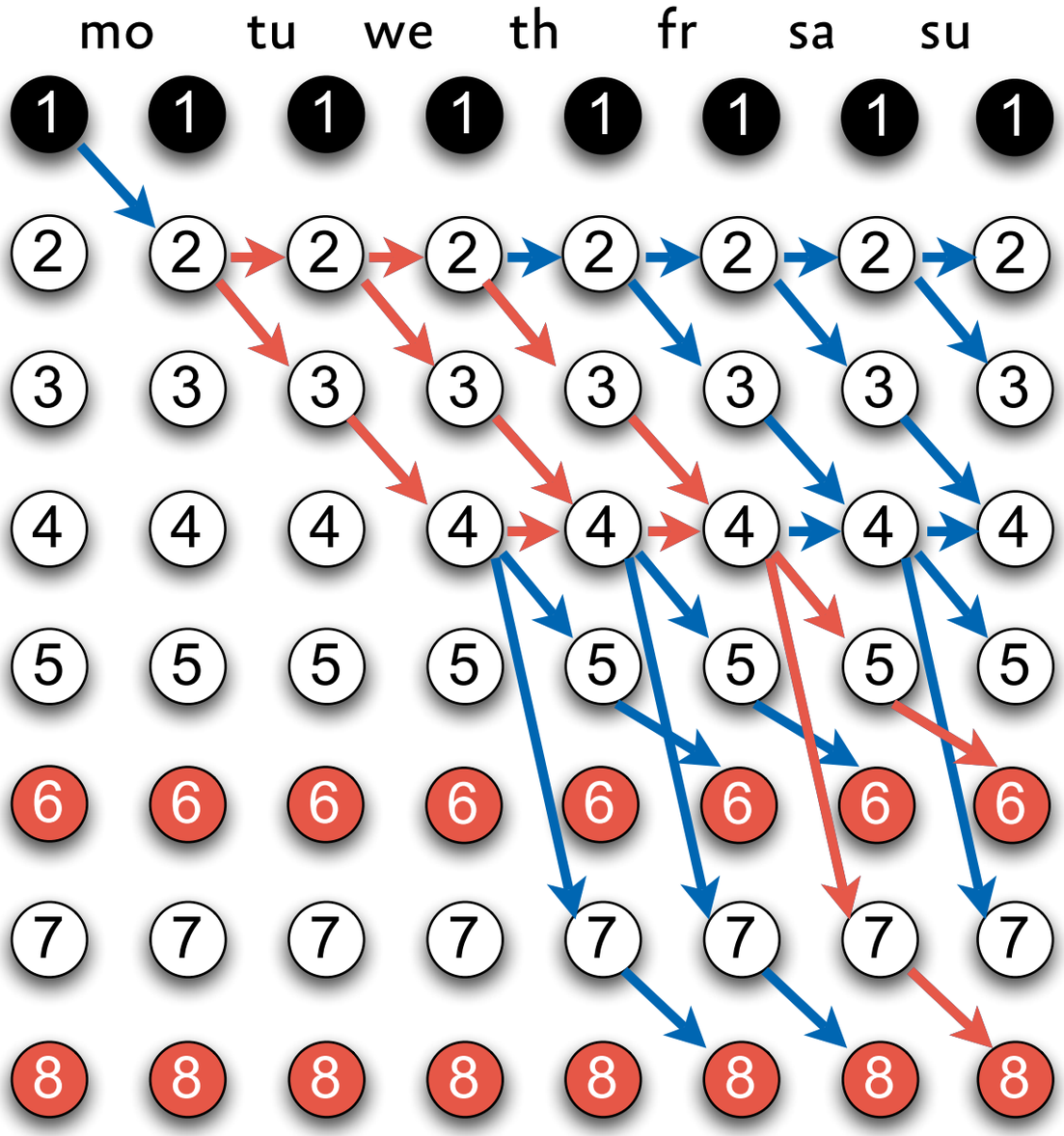
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

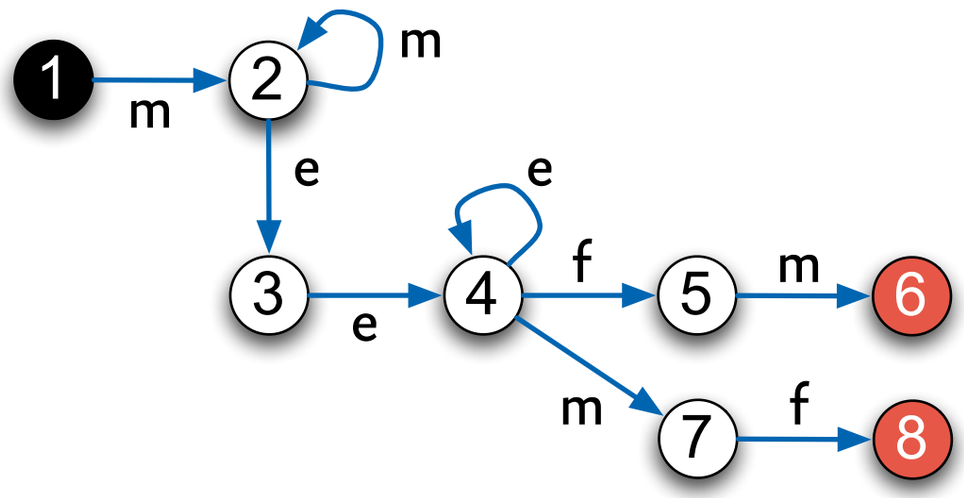
we ∈ {m,e,f}

th ∈ {m,e,f}

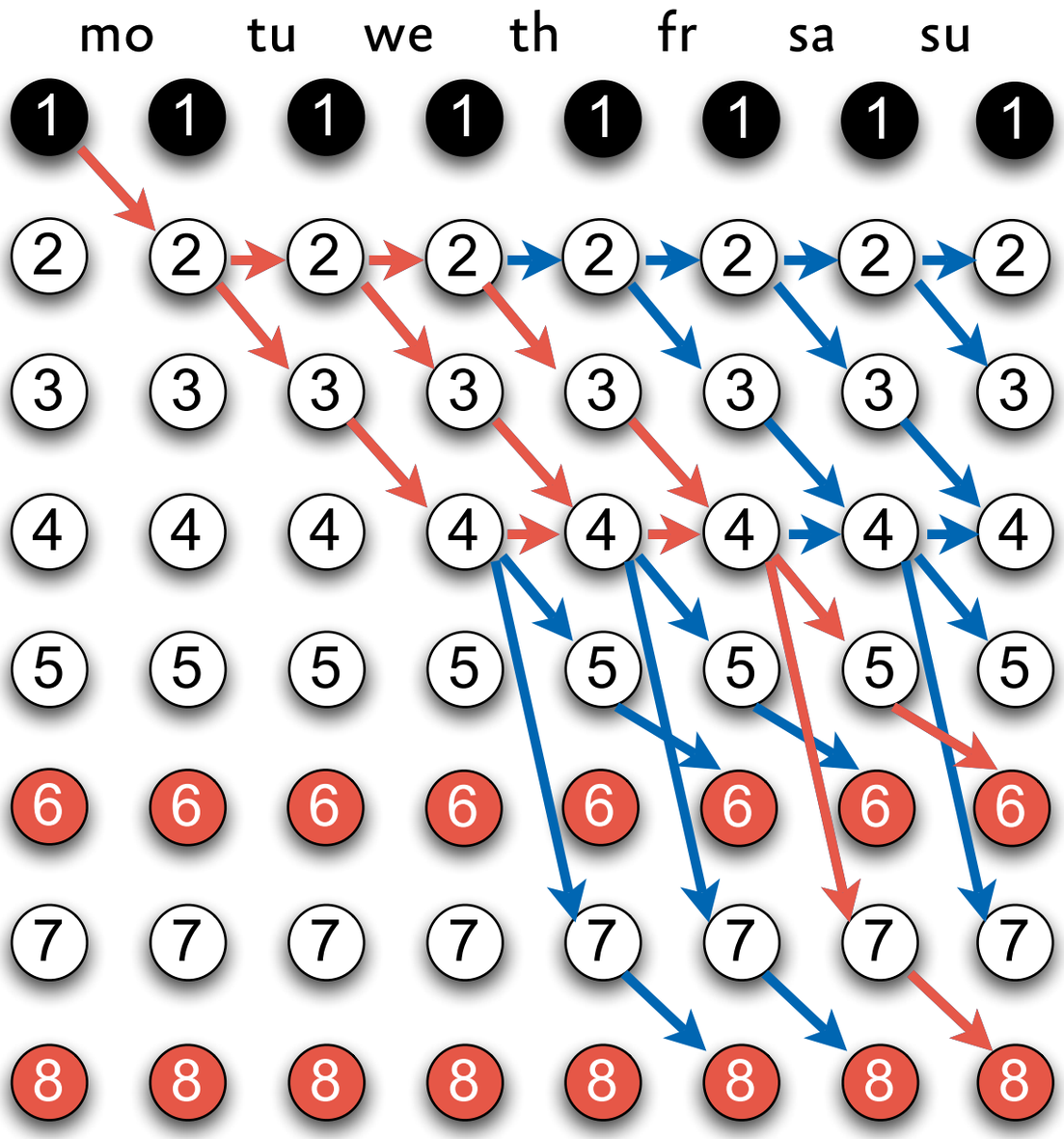
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



backward phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

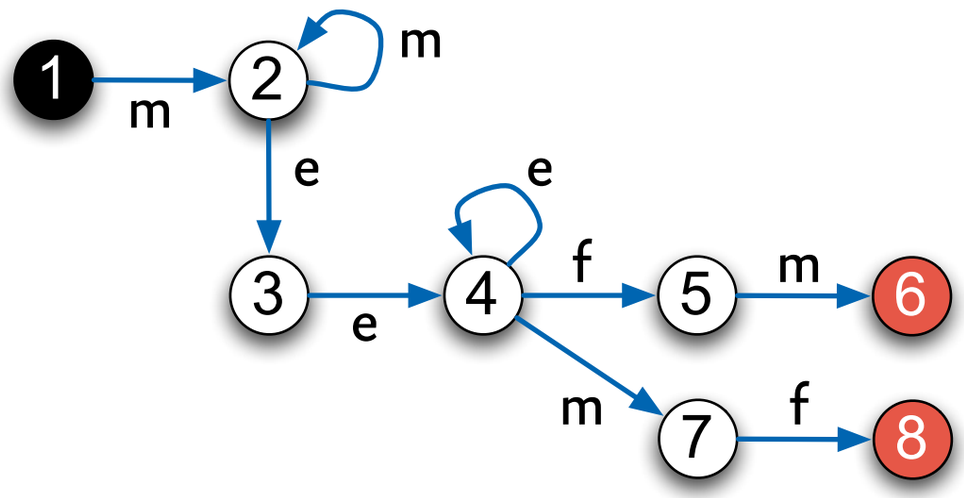
we ∈ {m,e,f}

th ∈ {m,e,f}

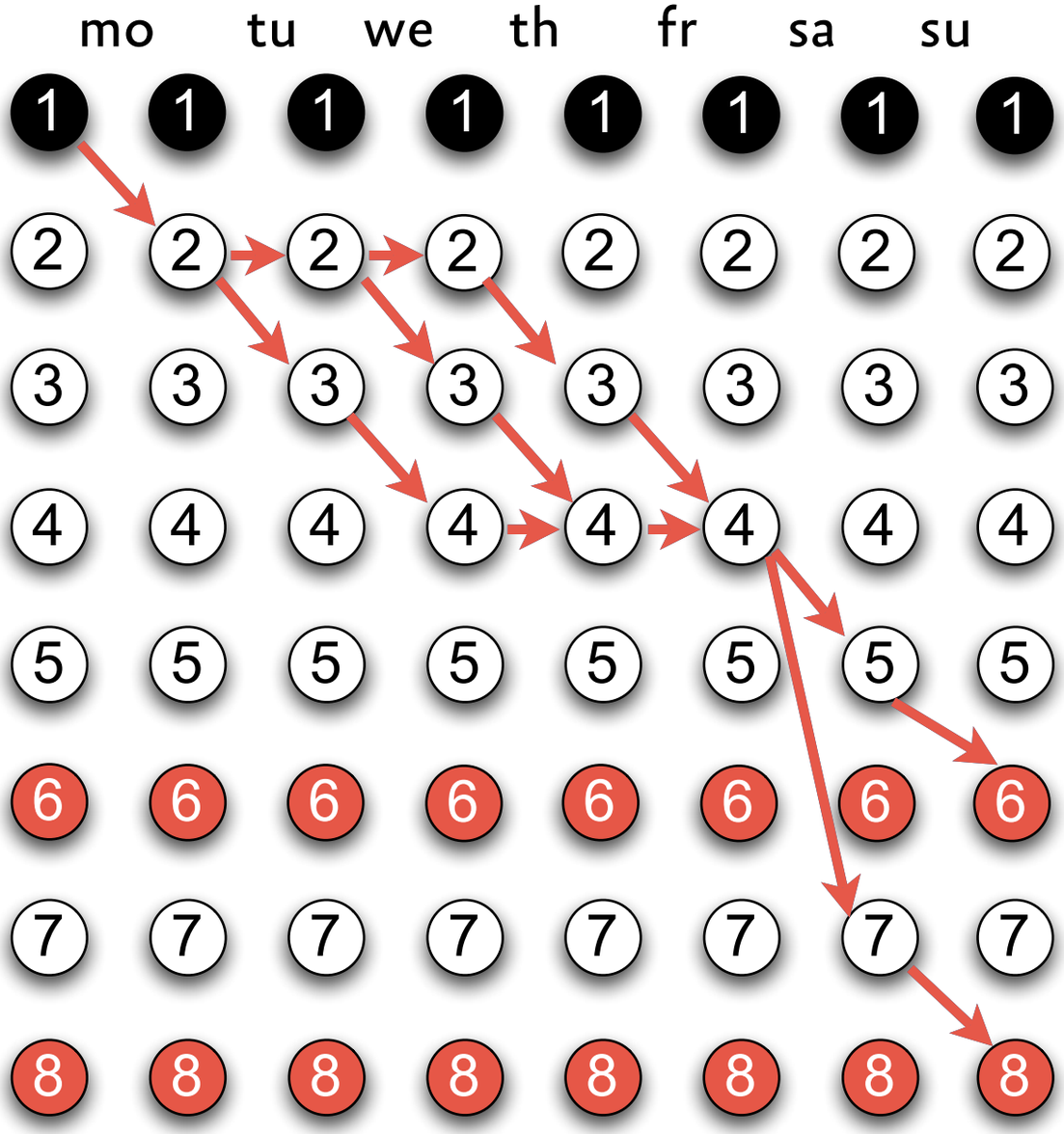
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



update phase

mo ∈ {m,e,f}

tu ∈ {m,e,f}

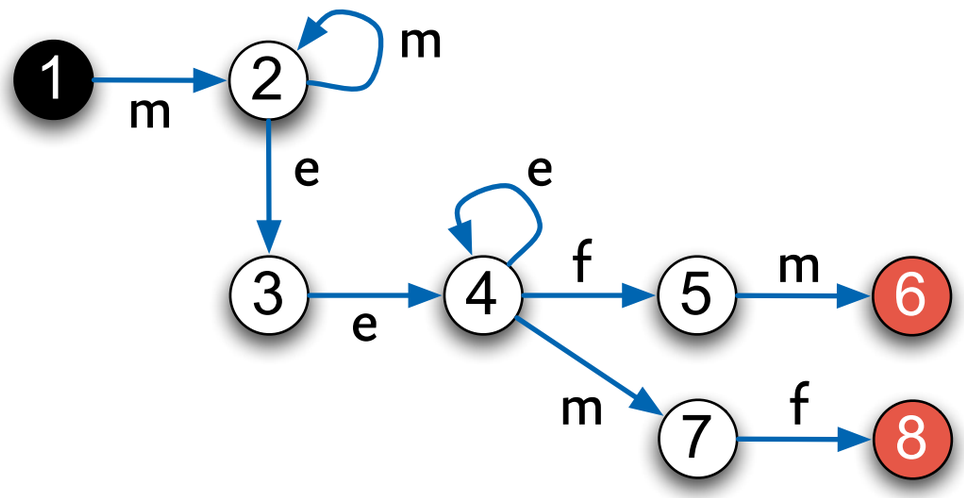
we ∈ {m,e,f}

th ∈ {m,e,f}

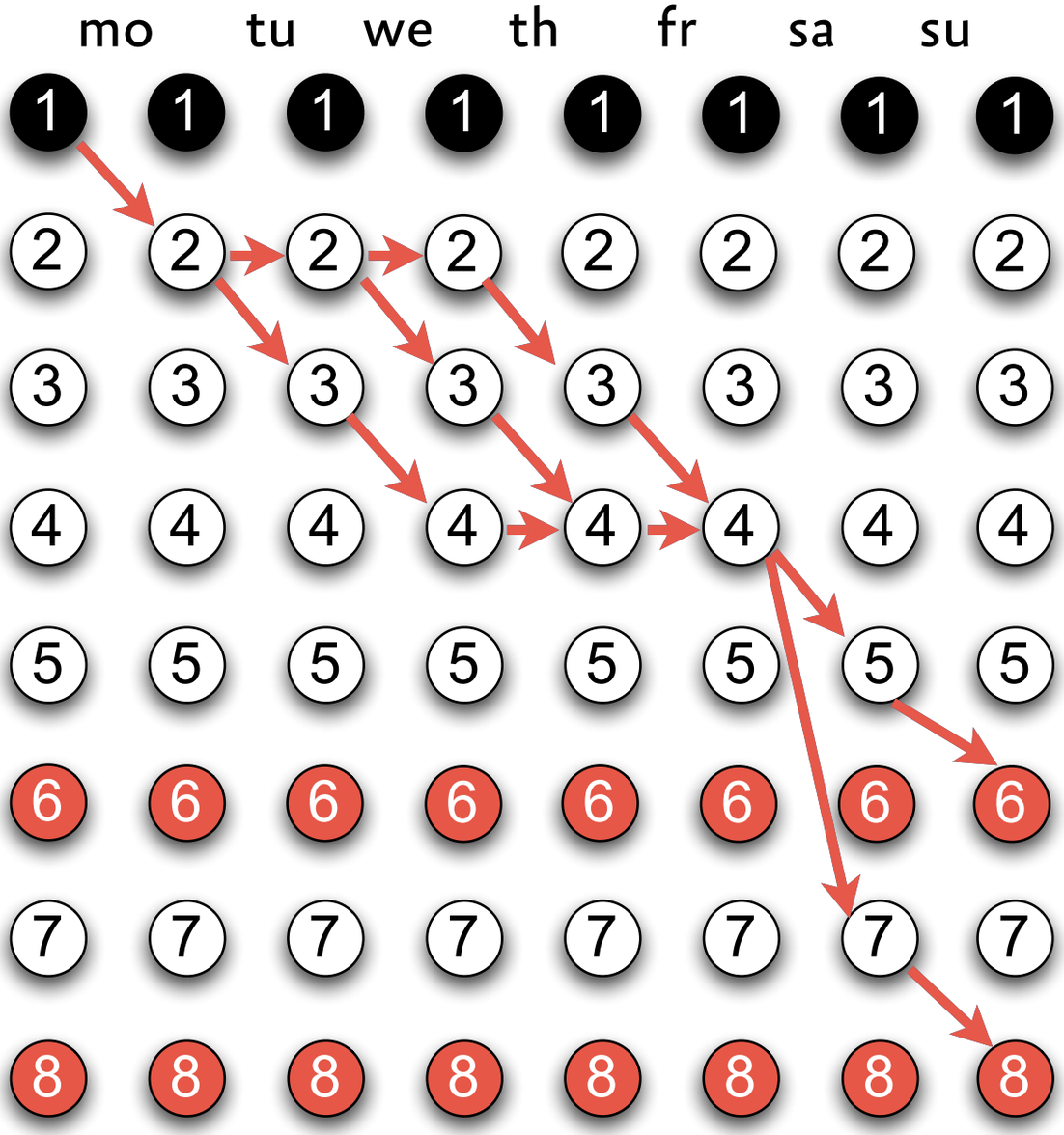
fr ∈ {m,e,f}

sa ∈ {m,e,f}

su ∈ {m,e,f}



Propagating *regular*



update phase

mo ∈ {m}

tu ∈ {m,e}

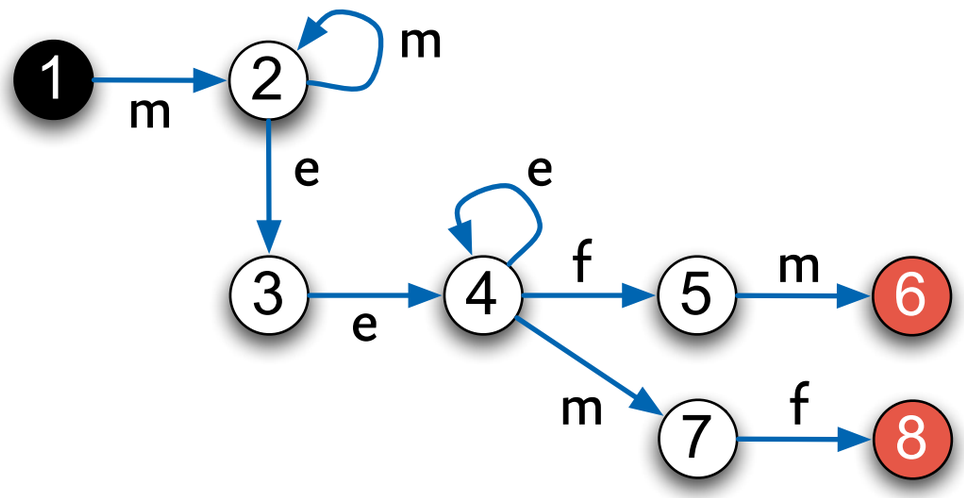
we ∈ {m,e}

th ∈ {e}

fr ∈ {e}

sa ∈ {m,f}

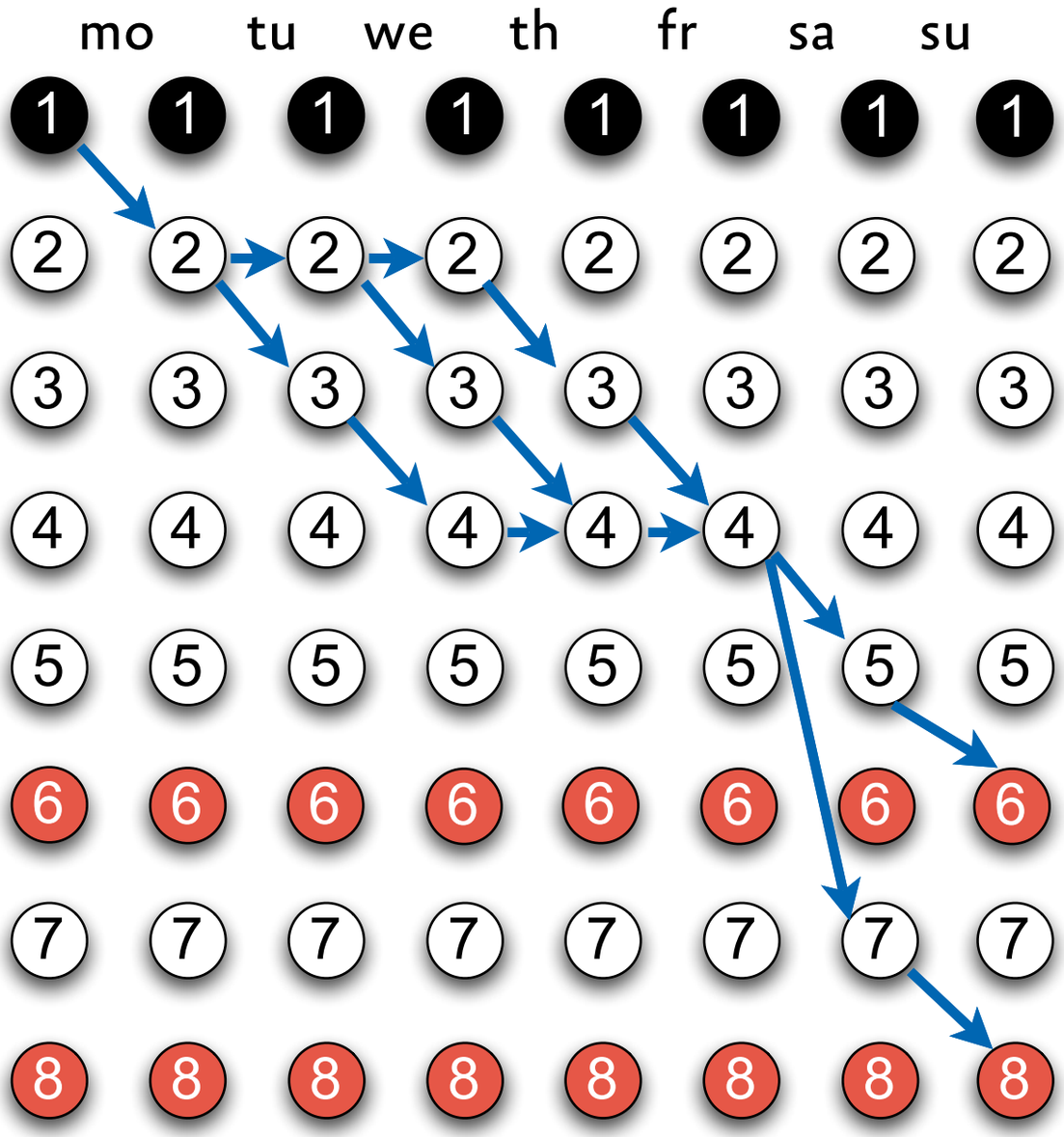
su ∈ {m,f}



Incrementality

- **keep layered graph between invocations**
- **if *v* is removed from x_i:**
  - delete all arcs in layer *i* corresponding to a transition with label *v*
  - propagate changes forward and backward

Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

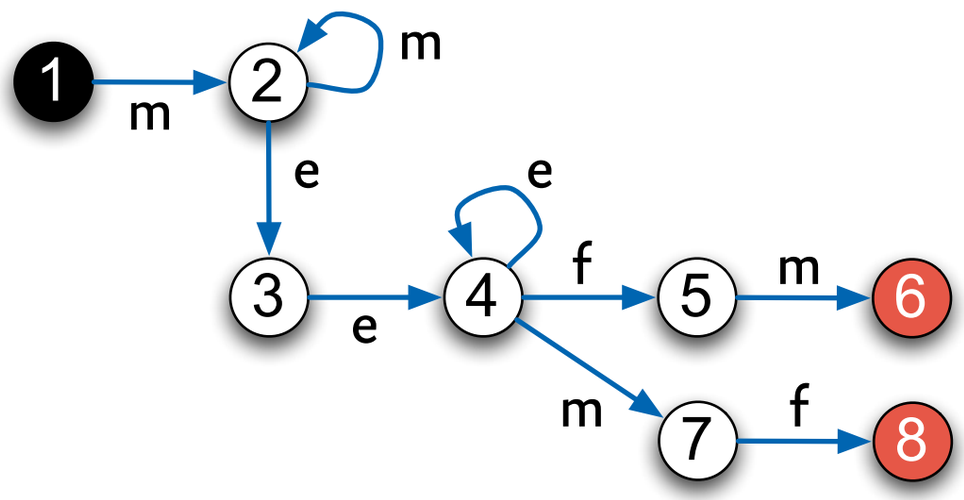
we ∈ {m,e}

th ∈ {e}

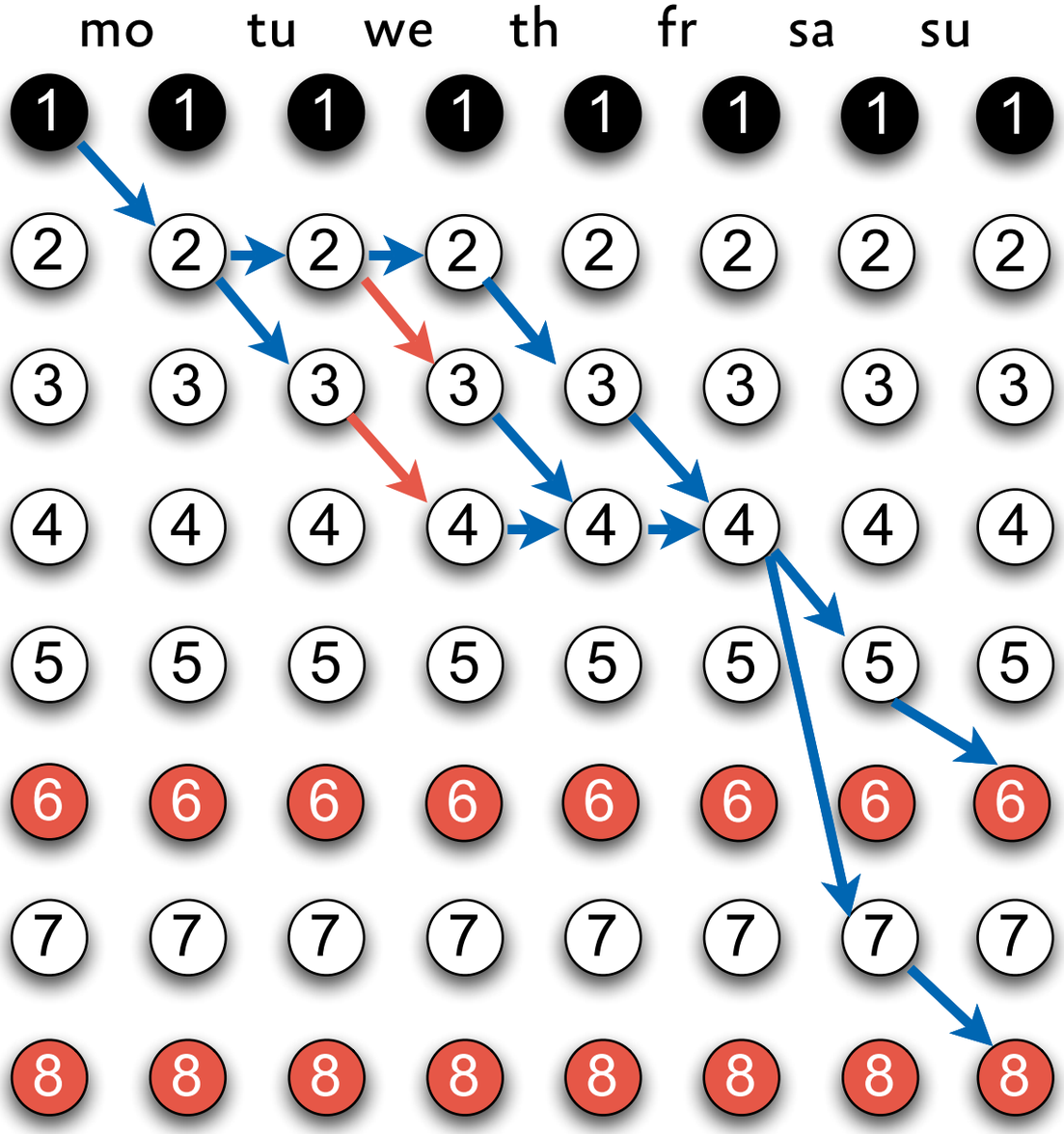
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

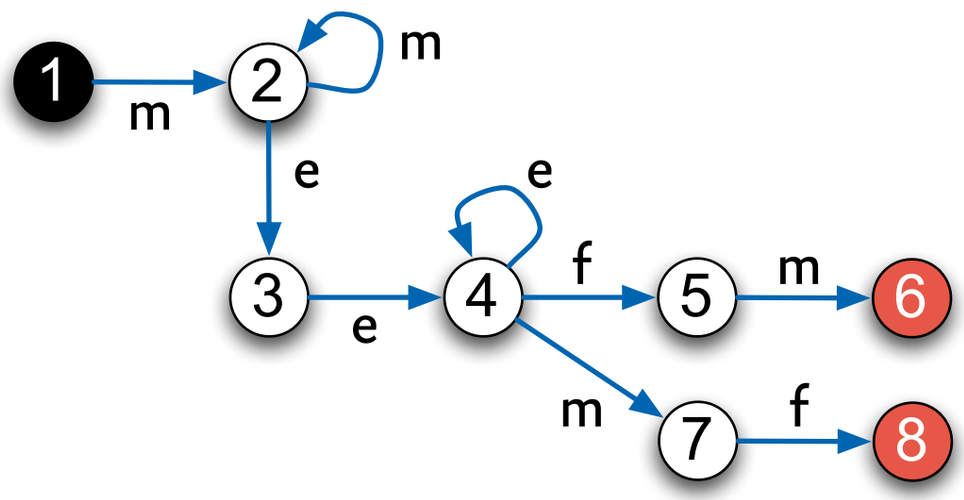
we ∈ {m,e}

th ∈ {e}

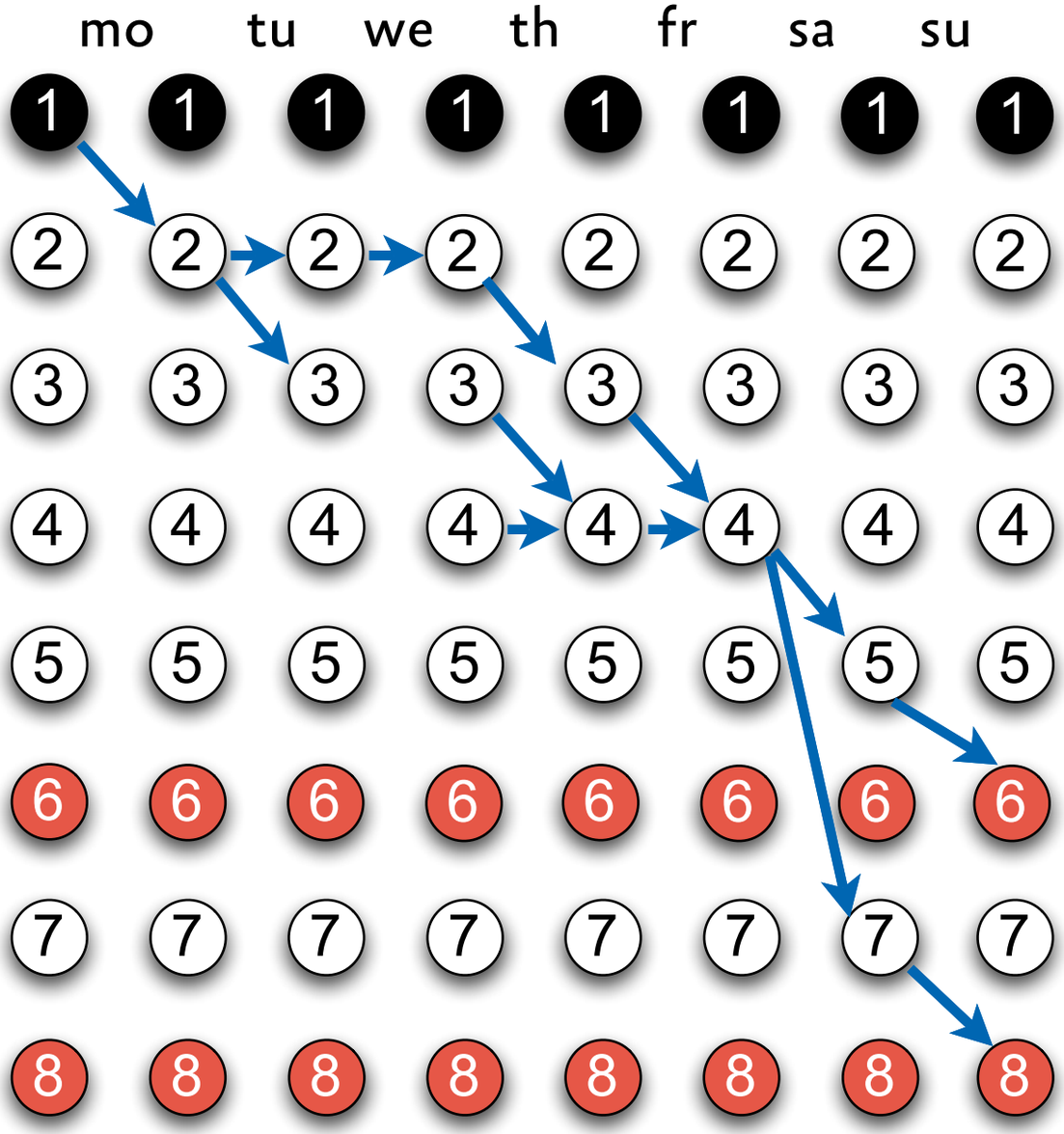
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

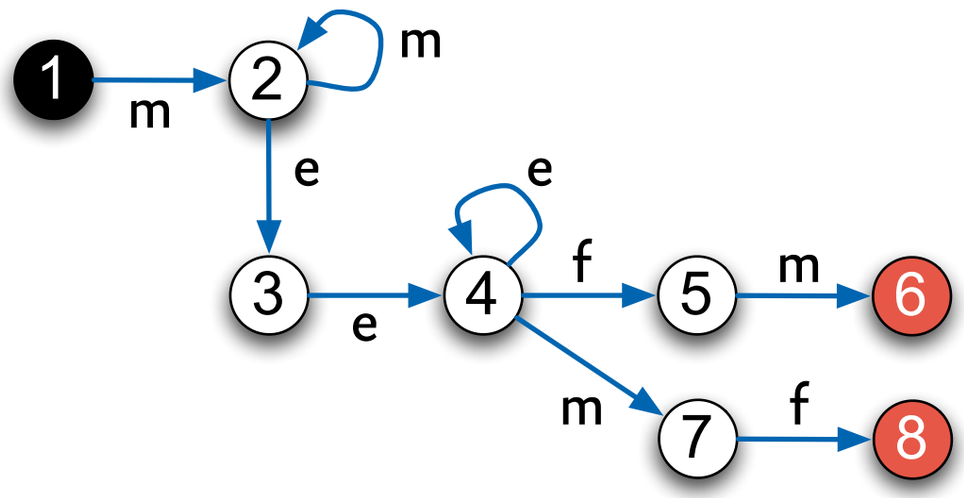
we ∈ {m,e}

th ∈ {e}

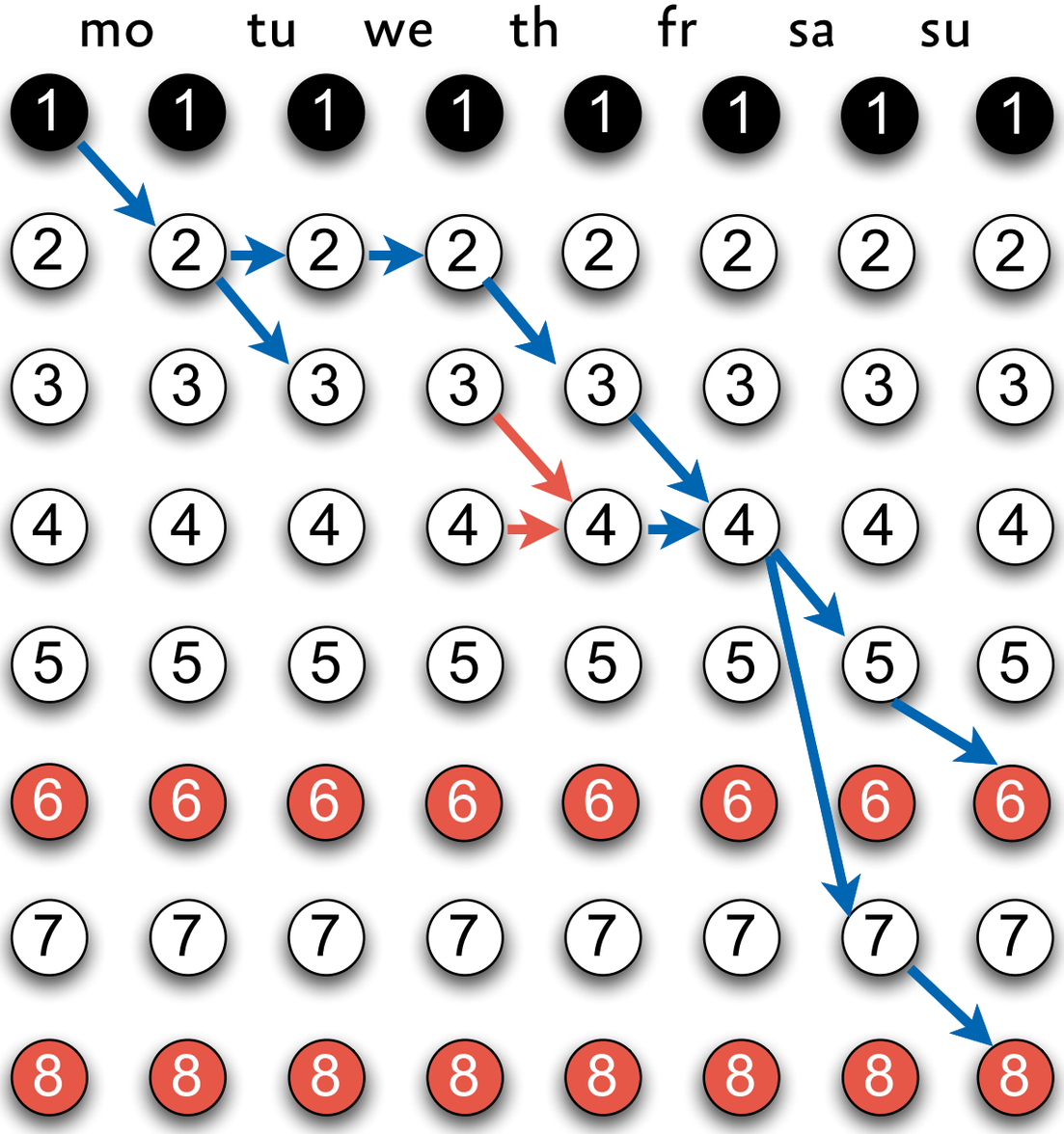
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

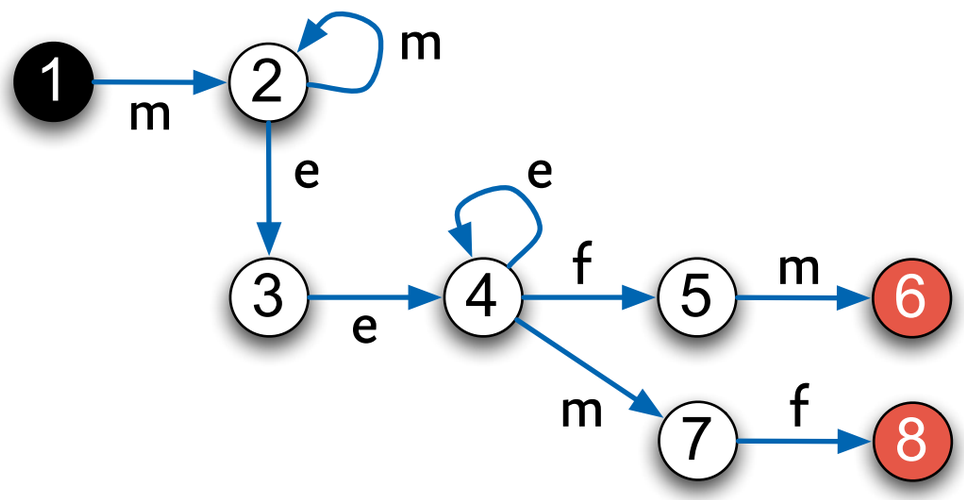
we ∈ {m,e}

th ∈ {e}

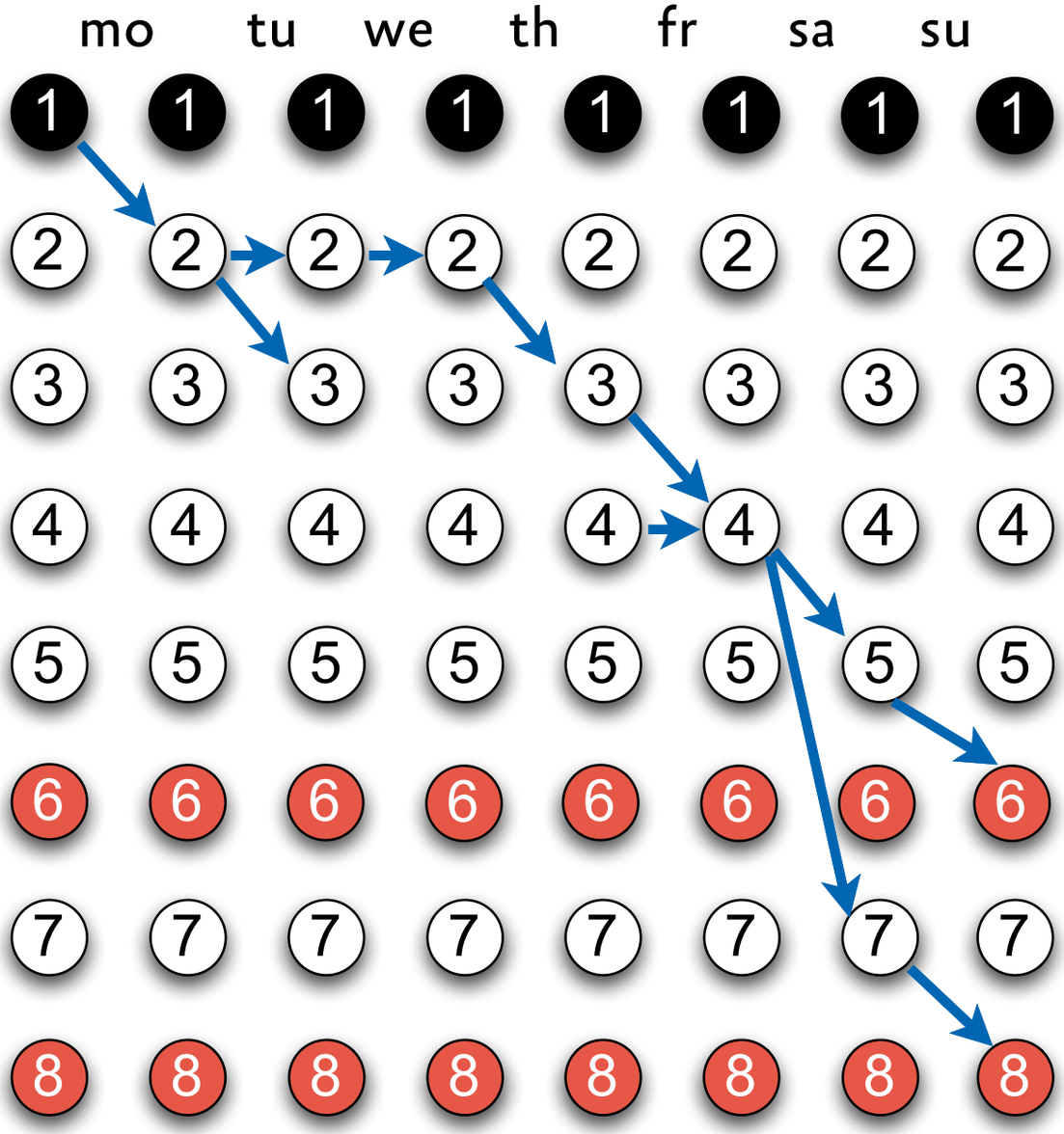
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

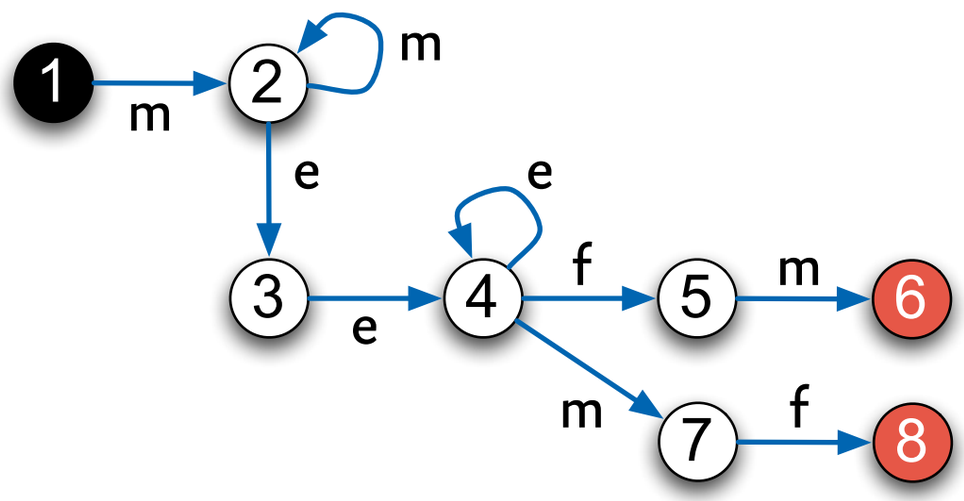
we ∈ {m,e}

th ∈ {e}

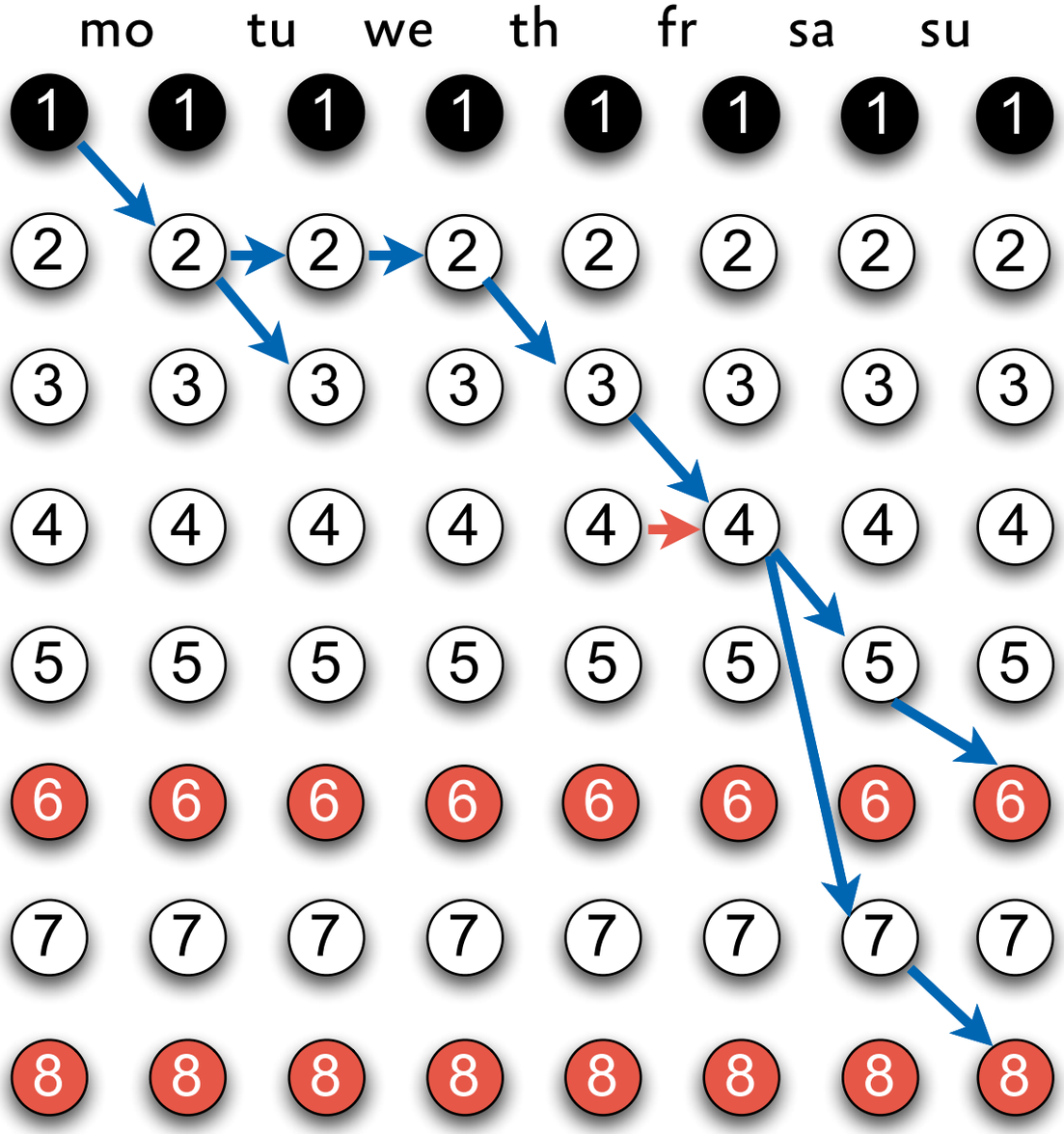
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

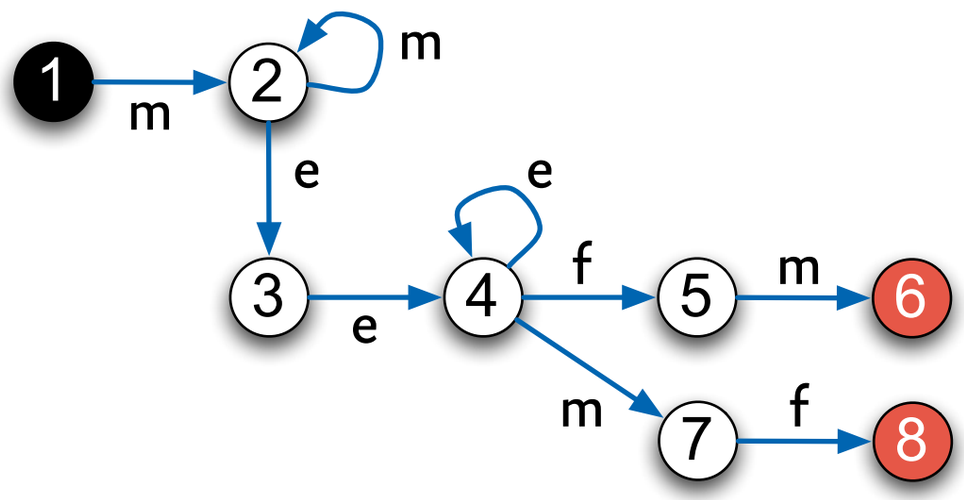
we ∈ {m,e}

th ∈ {e}

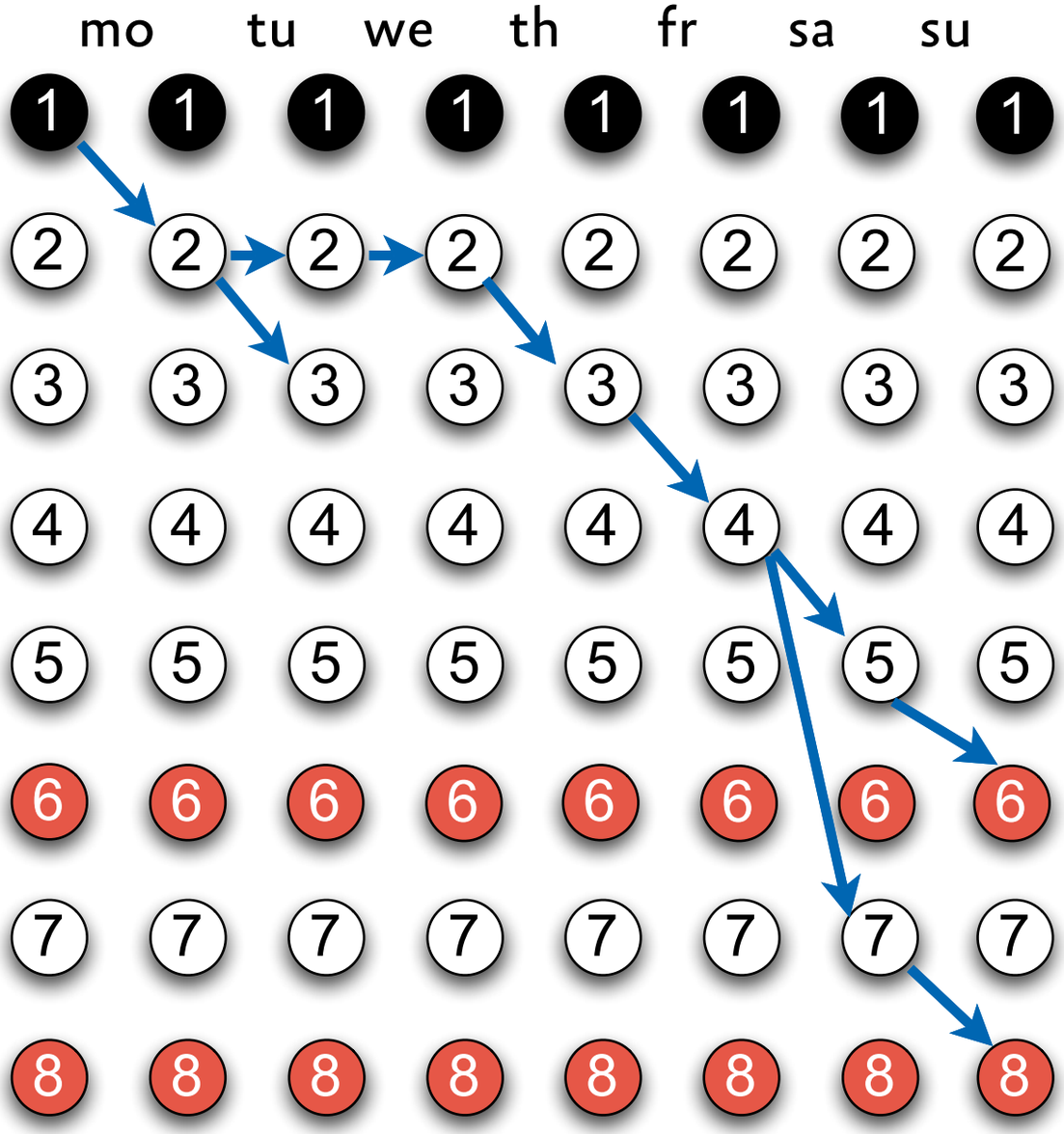
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

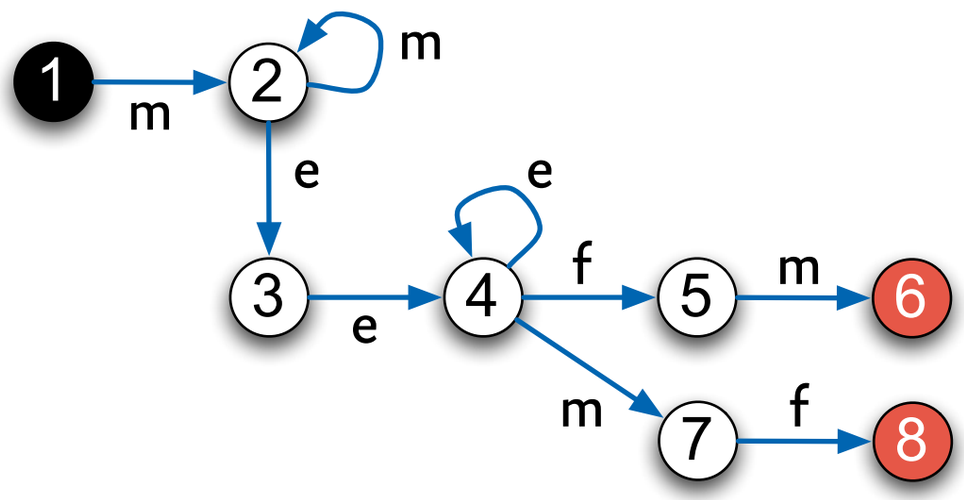
we ∈ {m,e}

th ∈ {e}

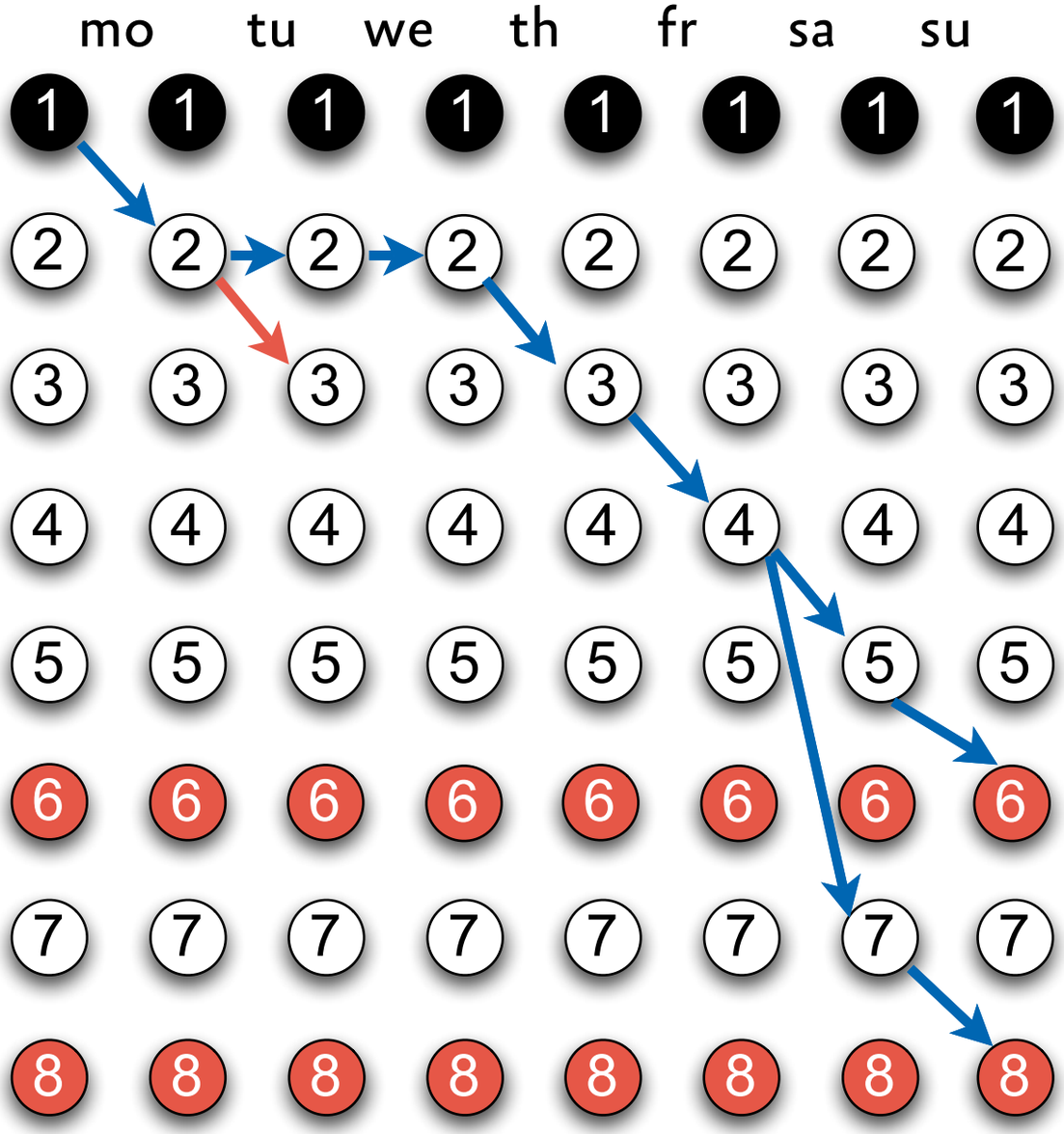
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

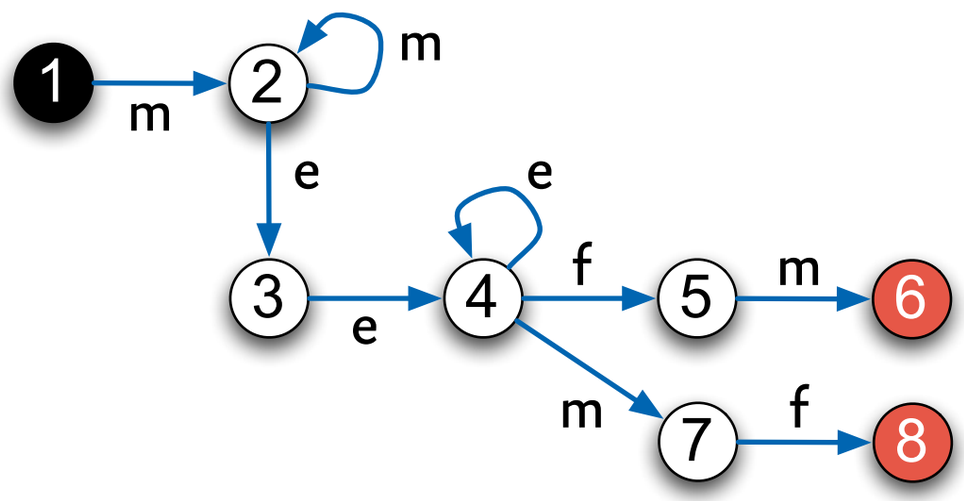
we ∈ {m,e}

th ∈ {e}

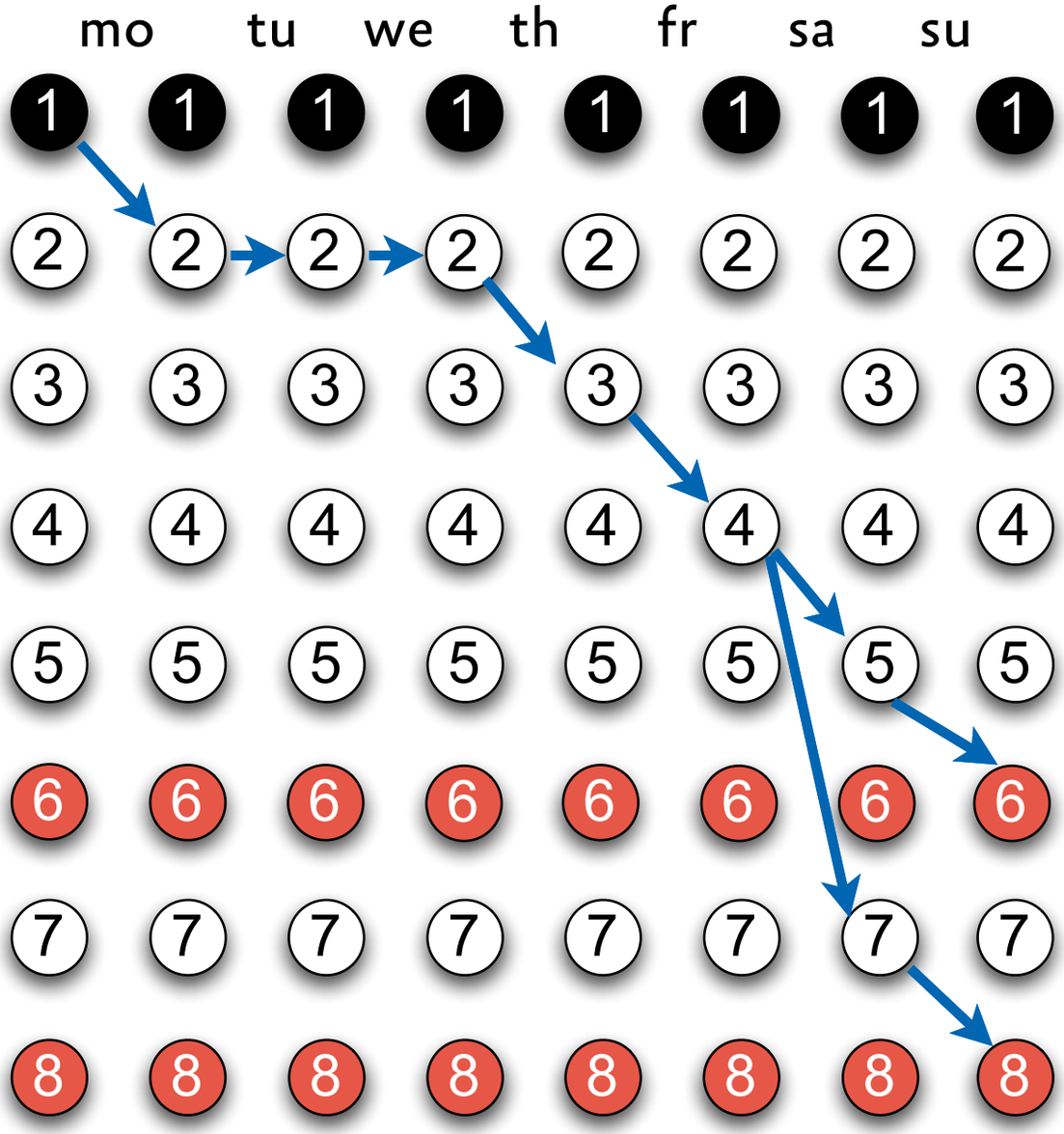
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

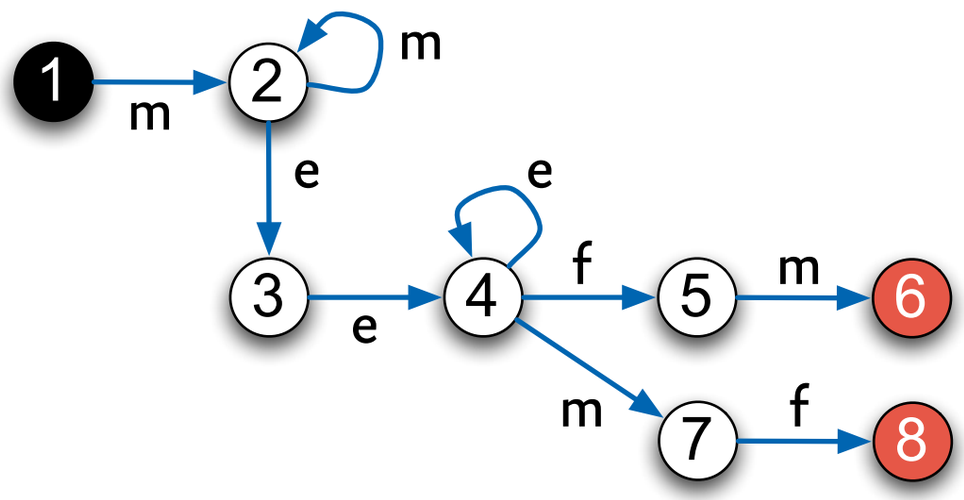
we ∈ {m,e}

th ∈ {e}

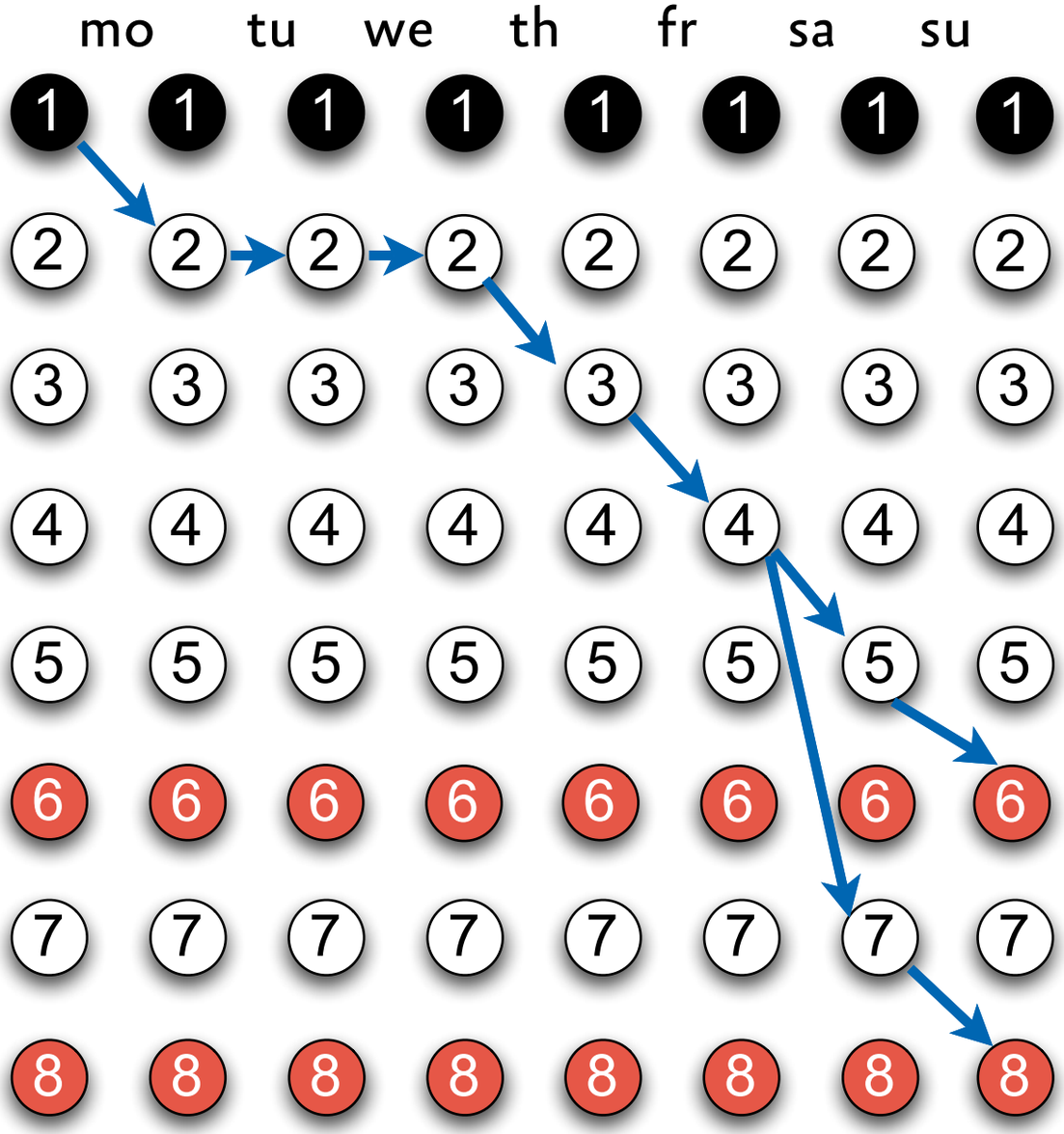
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m,e}

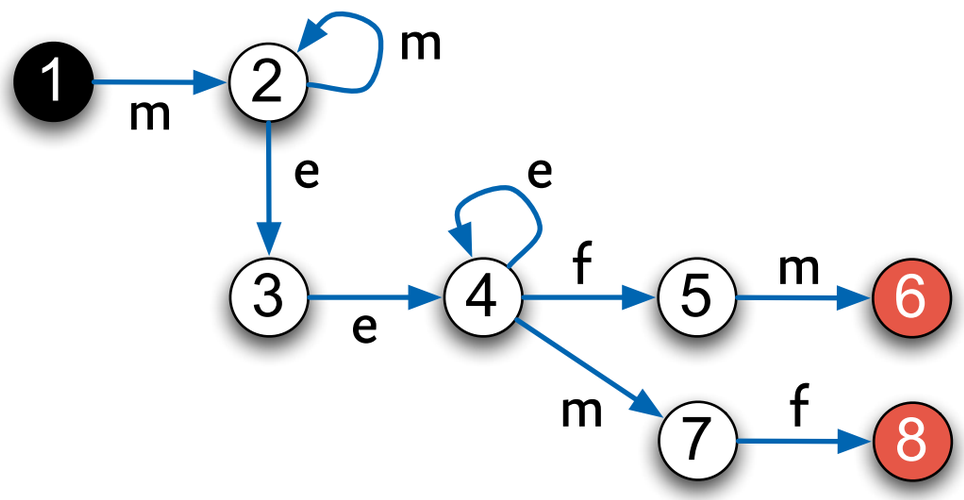
we ∈ {m,e}

th ∈ {e}

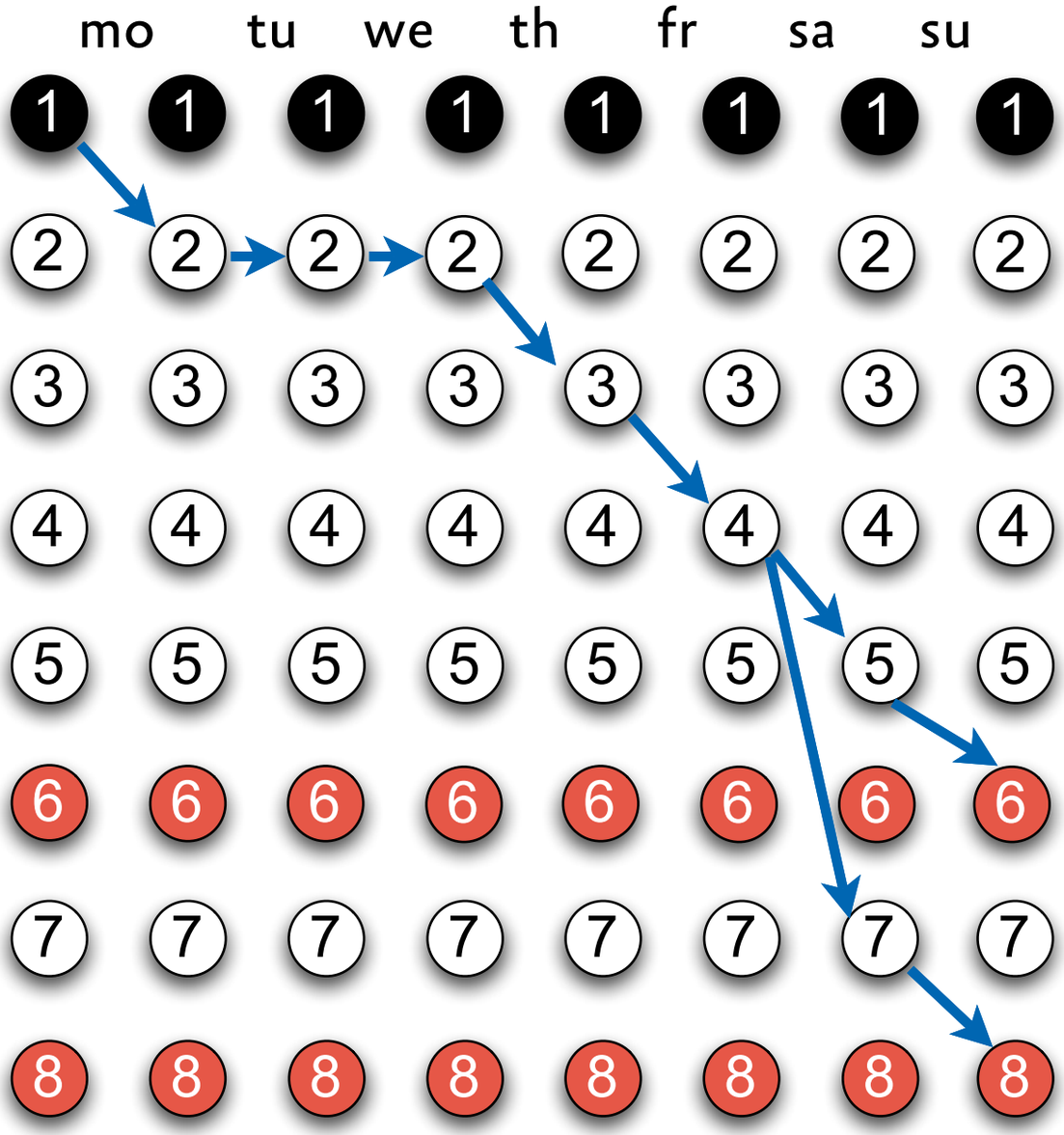
fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Propagating *regular*



removing a value

mo ∈ {m}

tu ∈ {m}

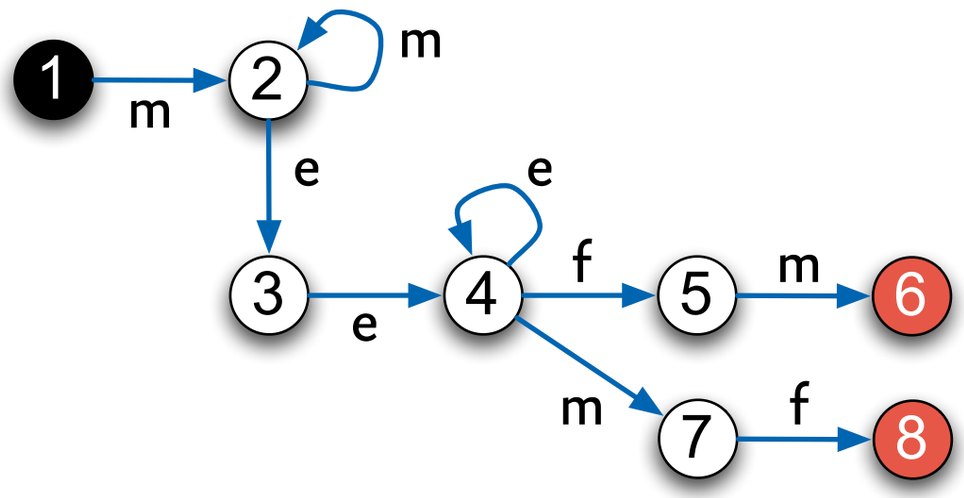
we ∈ {m}

th ∈ {e}

fr ∈ {e}

sa ∈ {m,f}

su ∈ {m,f}



Consistency achieved by *regular*

- **propagating regular achieves domain consistency:**
  - all arcs correspond to values in a domain and valid transitions
  - every arc is reachable from the initial state
  - every arc lies on a path to a final state
  - consequence: every value has support

Runtime analysis

- **Initial setup:**

  $O(nm|Q|)$ with n=#variables, m=domain size, $|Q|$=#states

- **Incremental propagation:**

  constant time per removed arc

  (provided we use clever data structures for the graph)

- **Linear in** $|Q|$: minimize the automaton!

Other applications of *regular*

- **encode ad-hoc extensional constraints**

  $c_{x,y}$ = { (1,3), (1,4), (5,8), (7,3) } is encoded as

  regular([x,y], (13)+(14)+(58)+(73))

- **encode puzzles**

  e.g. solitaire battleships, placing tiles on a board,...

- **sometimes infeasible!**

  e.g. *distinct* results in exponential size automaton!

Similar ideas

- **Easy extension:** circular patterns (x_n followed by x_1)
- **Extension to context-free languages**
  - grammar contraints
- **More complex automata**
  - counters lead to smaller automata
- **Alternative implementation**
  - decomposition into smaller constraints, without hindering propagation

Literature

- **Recommended:**

  Gilles Pesant: *A regular language membership constraint*.

  In: CP 2004, LNCS 3258, Springer Verlag.

  Well-written, and includes implementation details.

Summary: Scheduling

- **Hard, real-life problems**
- **Model similar to temporal relations**

  but: interested in actual solution + optimization
- **Global constraints:** timetable, edge-finding, not-first/not-last
- **Specialized branching**

Summary: Rostering

- **Hard, real-life problems**
- **Useful propagator:** regular

Thank you.