Computing Schedules for Multithreaded Real-Time Programs Using Geometry

Philippe Gerner and Thao Dang

VERIMAG

Introduction

- More and more software in embedded systems
  $\longrightarrow$ multithreaded applications
- Real-time issues
- Behavior of real-time multithreaded programs?
- Schedulers?

Introduction

- More and more software in embedded systems
  $\longrightarrow$ multithreaded applications
- Real-time issues
- Behavior of real-time multithreaded programs?
- Schedulers?

$\longrightarrow$ *Geometric approach*

Plan

I The geometric model

II Adding time

III Exploiting the geometry

Spacial decomposition

Geometric intersection

The geometric model

PV Programs

- Threads sharing resources
- Resources protected by Dijkstra semaphores
- P : Passeren $\longrightarrow$ accessing a resource
- V : Vrÿgeven $\longrightarrow$ releasing a resource

A PV program:

```
a, b : 1-semaphores
thread A = Pa.Pb.Vb.Va
thread B = Pb.Pa.Va.Vb
```

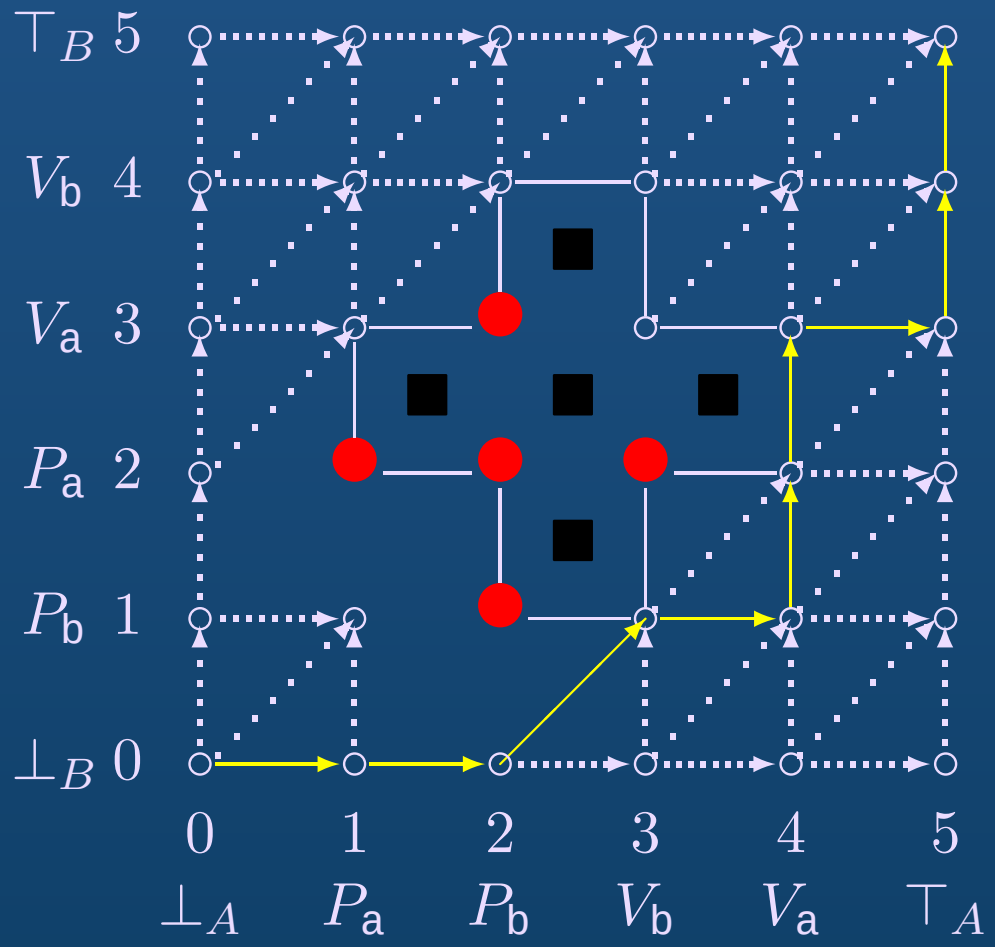
PV Programs: Definitions and Notations

- Set of shared resources $\Re$
- N threads $E_1, \ldots, E_N$.
- Each thread is a total order of events. Each event is associated with an action of locking/unlocking a resource (e.g. P_r, V_r)
- Each thread E_i contains at least two events: the *start event* $\bot_{E_i}$ *and the* end event $\top_{E_i}$
- The running together of the N threads is $\mathcal{E} = \prod_{i=1,\ldots,N} E_i$.

The Swiss flag example

forbidden states, string

A string s with $\bot_s = \bot$ and $\top_s = \top$ is called a **schedule**



Adding Time

Task durations

- Task: code between two consecutive events of a thread
- Duration of a task: worst-case execution time (WCET) of the task
- We are interested in the worst case response time (WCRT) of a schedule

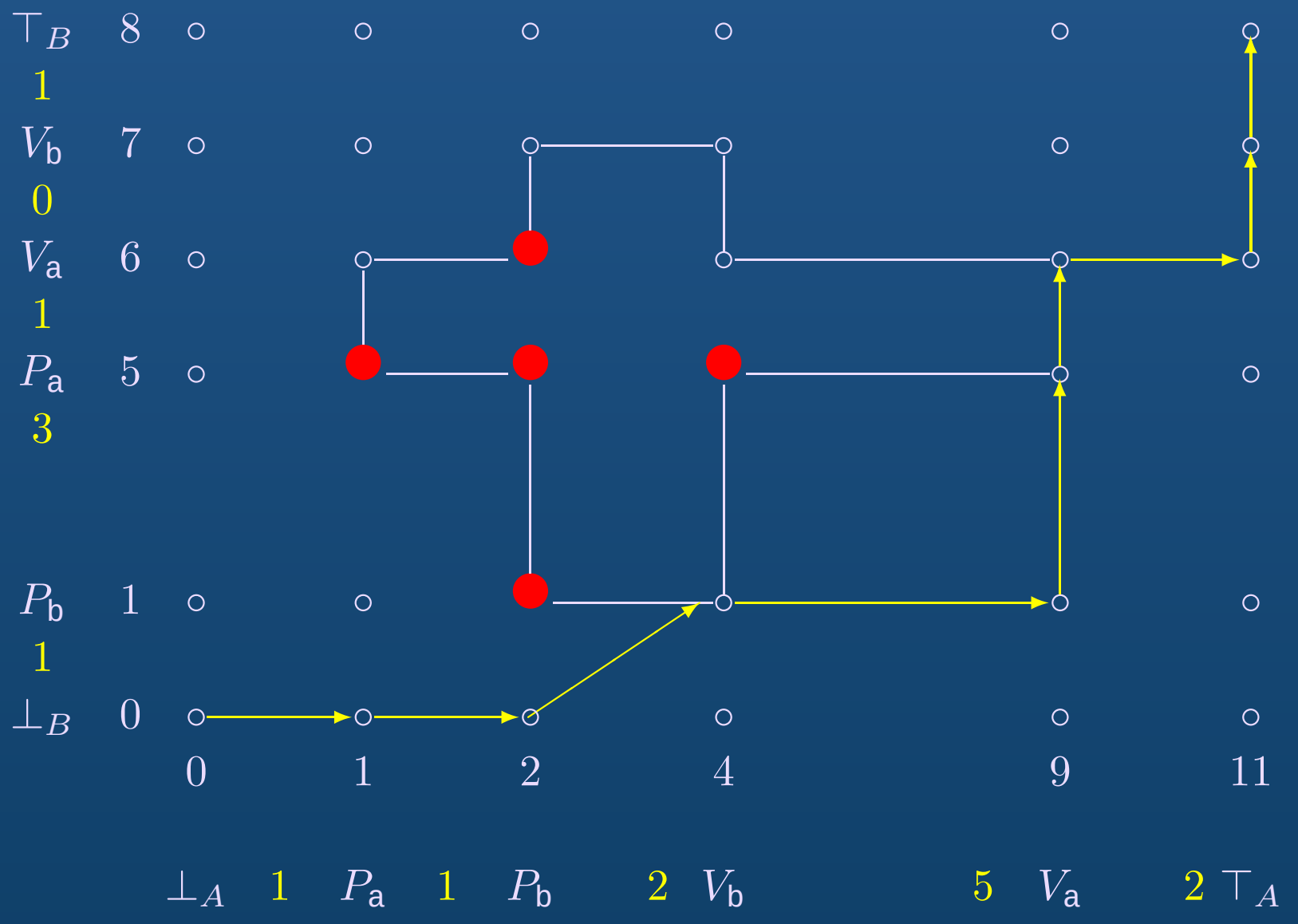
a, b : 1-semaphores

```
thread A = 1.Pa.1.Pb.2.Vb.5.Va.2
thread B = 1.Pb.3.Pa.1.Va.0.Vb.1
```

Scaling PV Diagrams $\Rightarrow$ Timed PV Diagrams

- Task duration $\rightarrow$ Euclidean distance between event coordinates
- Task duration $= 0$ $\rightarrow$ distance $\alpha > 0$ for distinguishing the events

The Timed Swiss Flag



3D Example: Three Philosophers

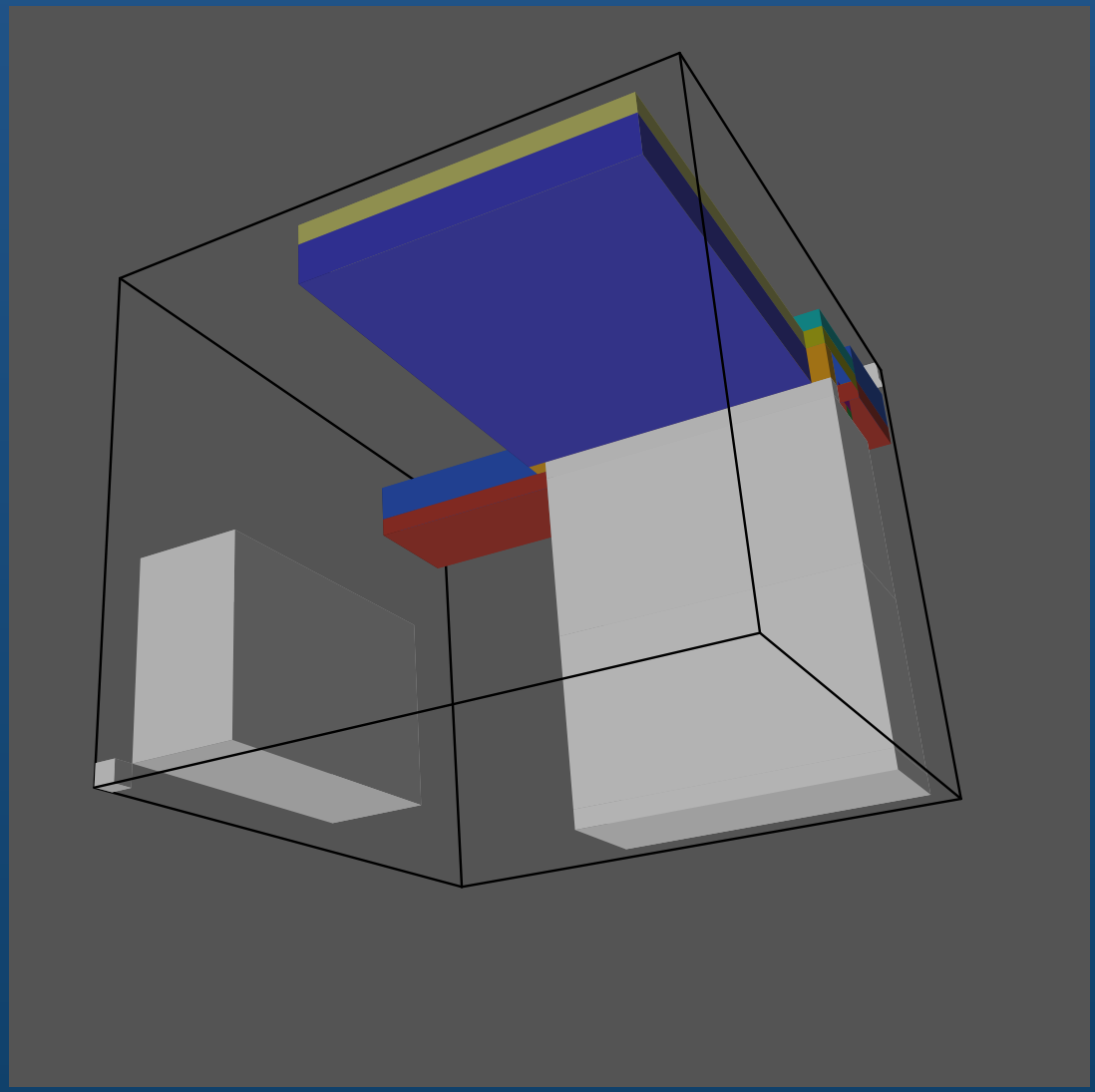
The timed PV program:

```
room  : 2-semaphore
a, b, c : 1-semaphores
thread A = 0.P room .5.V room .4.Pa.0.Pb.15.Va.0.Vb.0
thread B = 0.P room.14.V room .6.Pb.0.Pc .5.Vb.0.Vc.0
thread C = 0.P room .9.V room .9.Pc.0.Pa .2.Vc.0.Va.0
```

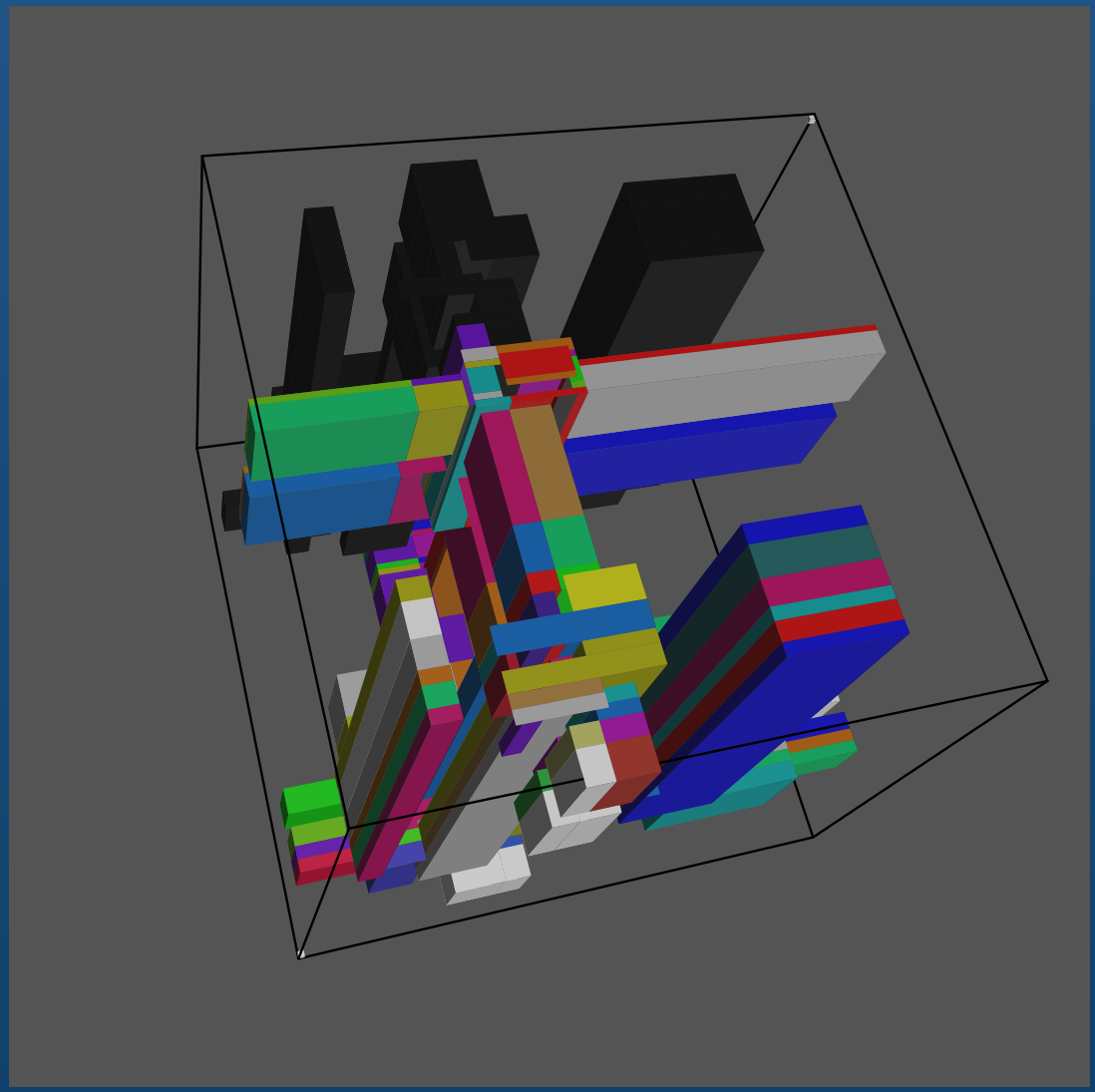
Timed PV diagram:

3 threads $\rightarrow$ 3 dimensions

The Timed PV Diagram



Three Philosophers, Sharing More Resources

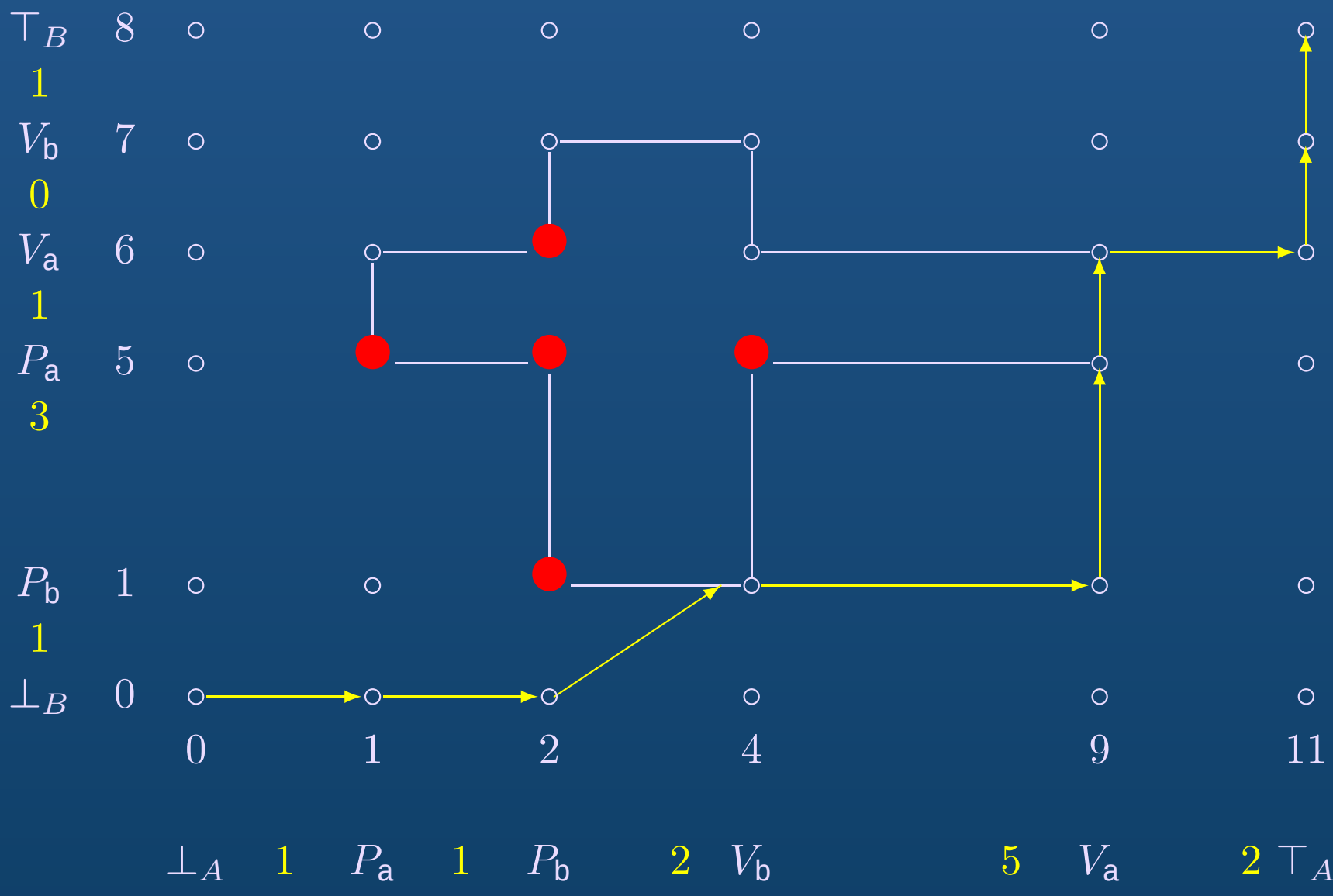


Worst Case Response Time of a Schedule

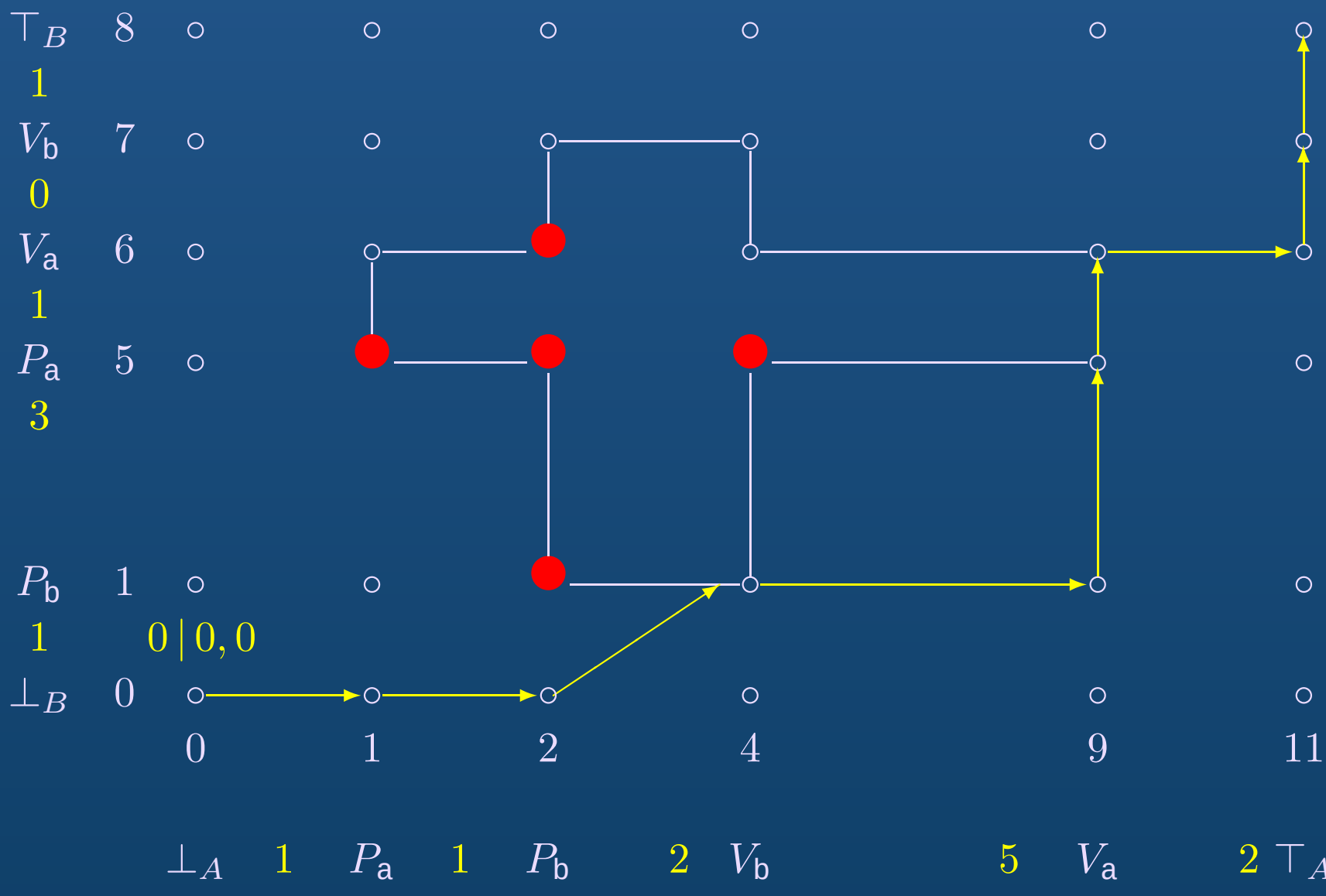
String duration defined by the following algorithm:

- N **local clocks** — one per thread
- One **global clock**
- Global clock at $\perp : 0$; at $\top$: WCRT
- For the threads that have a new event:
  1. update their local clocks
  2. update the global clock with the max of local clock values
  3. *update the local clocks* with the value of the global clock

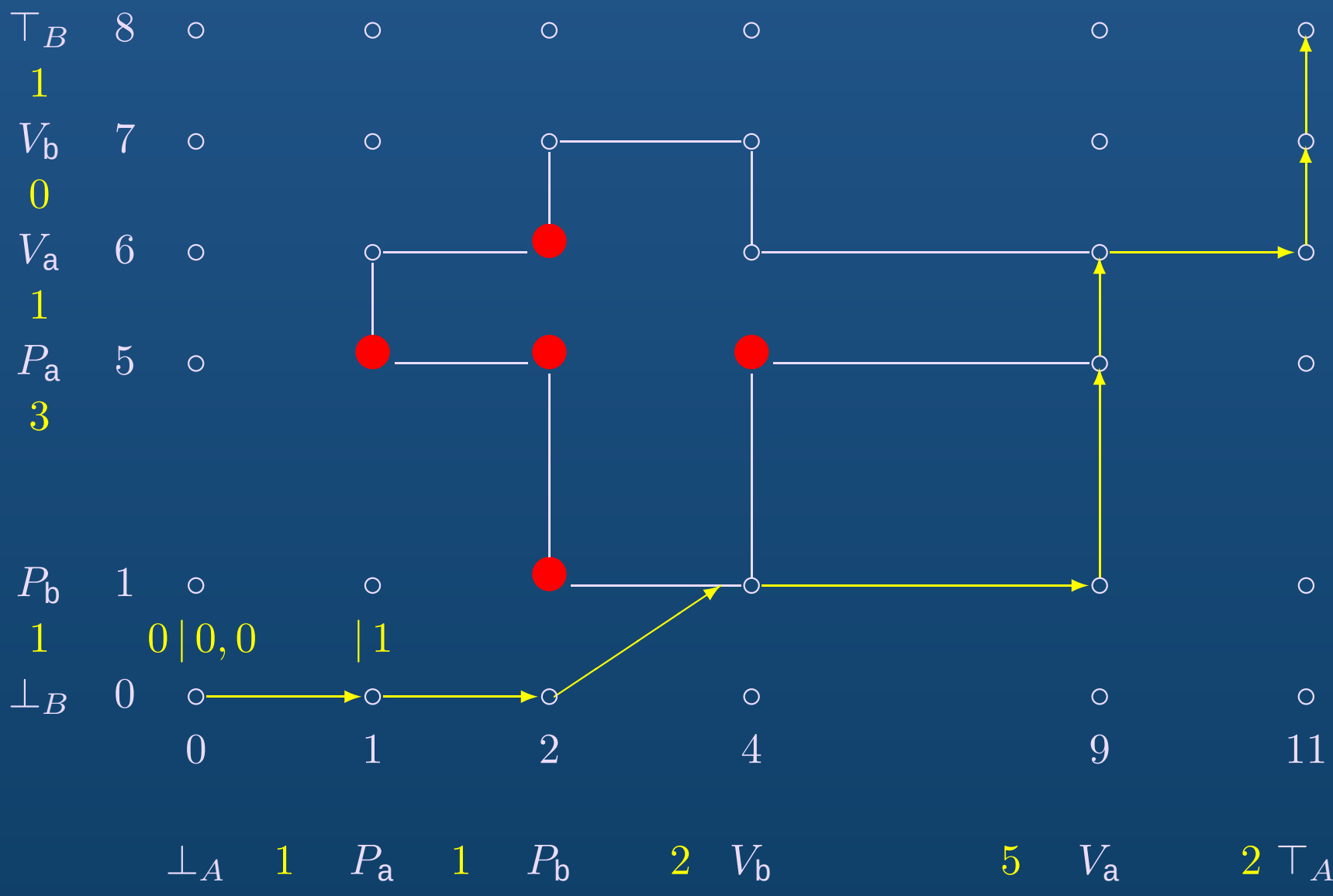
Example: Timed Swiss Flag



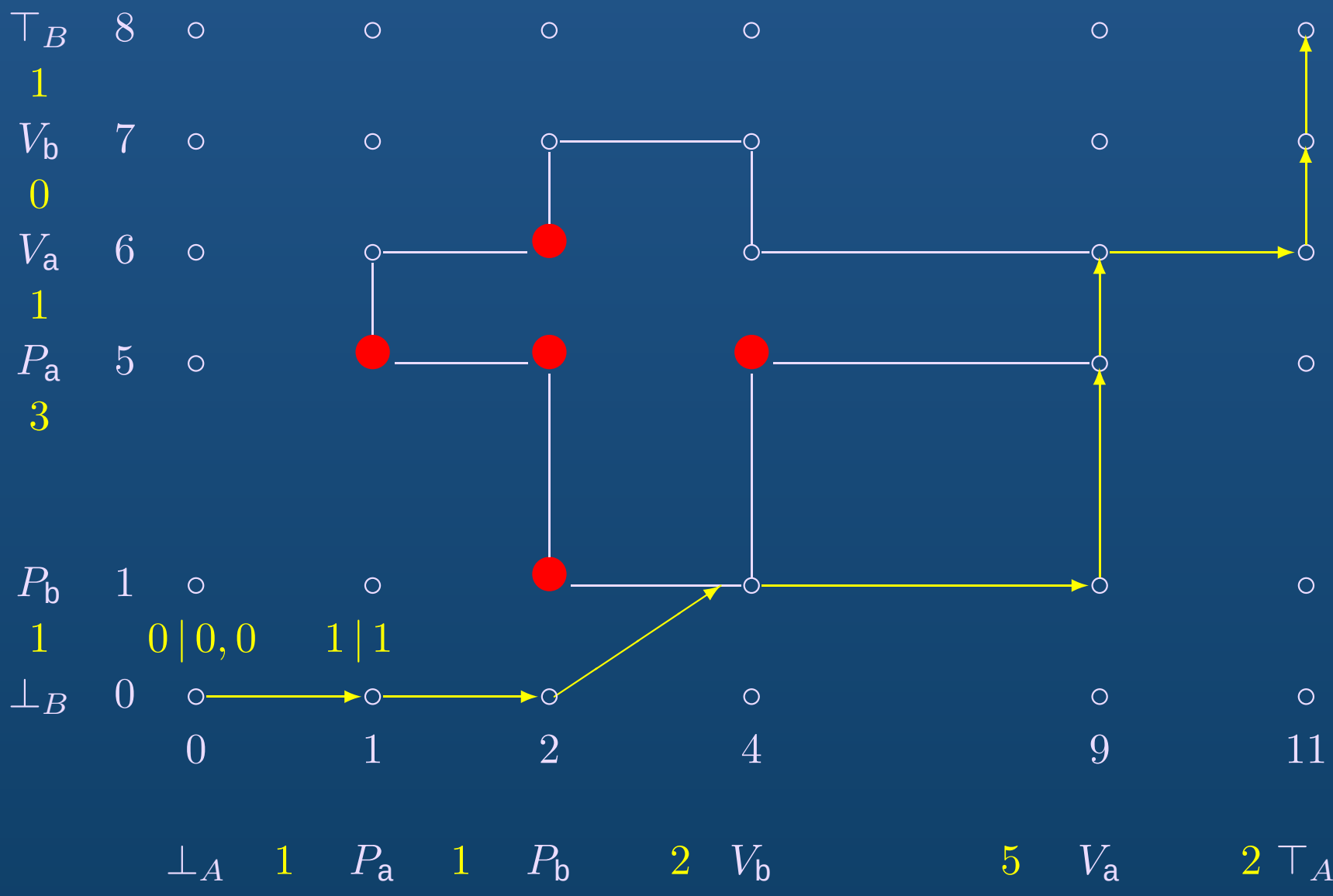
Example: Timed Swiss Flag



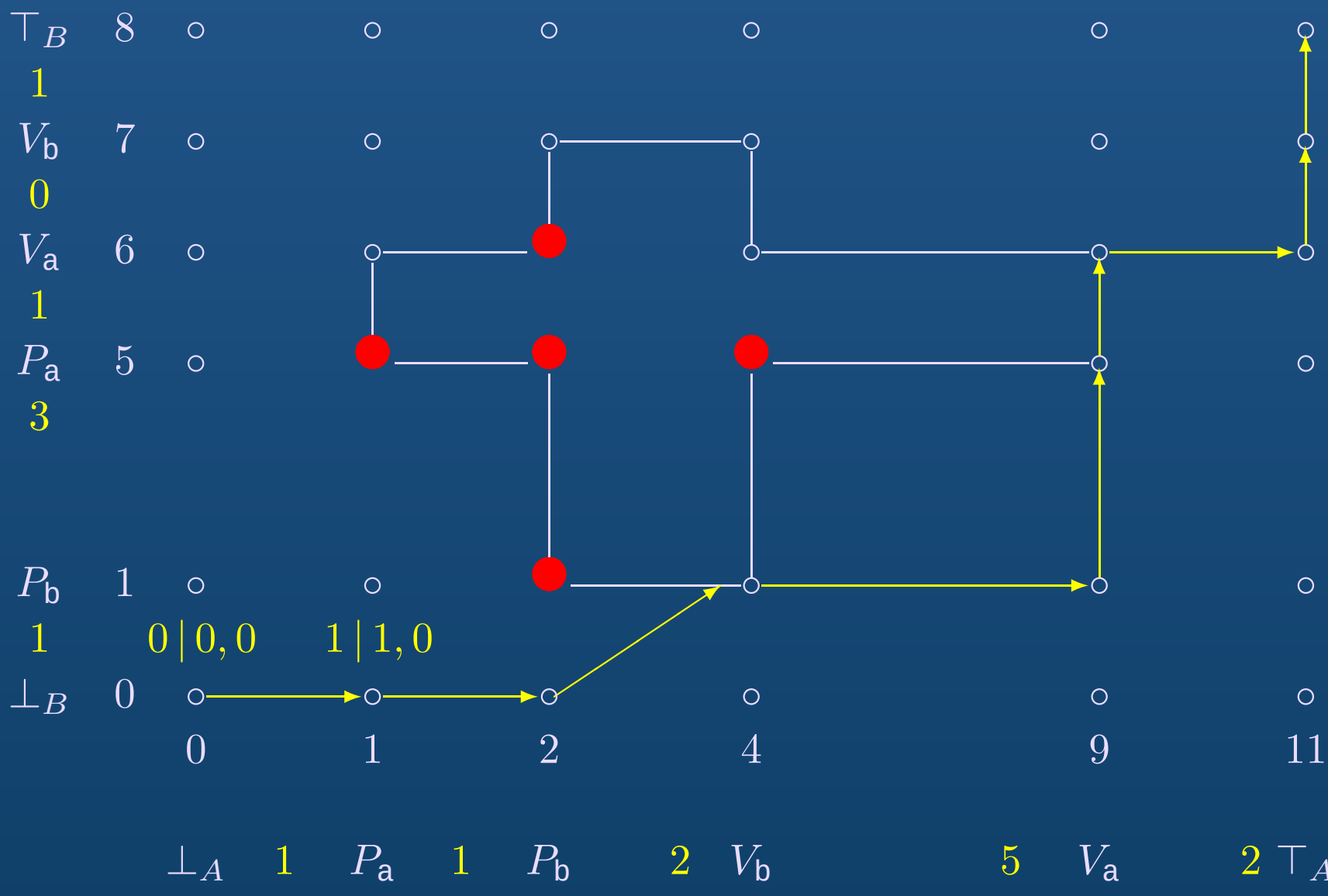
Example: Timed Swiss Flag



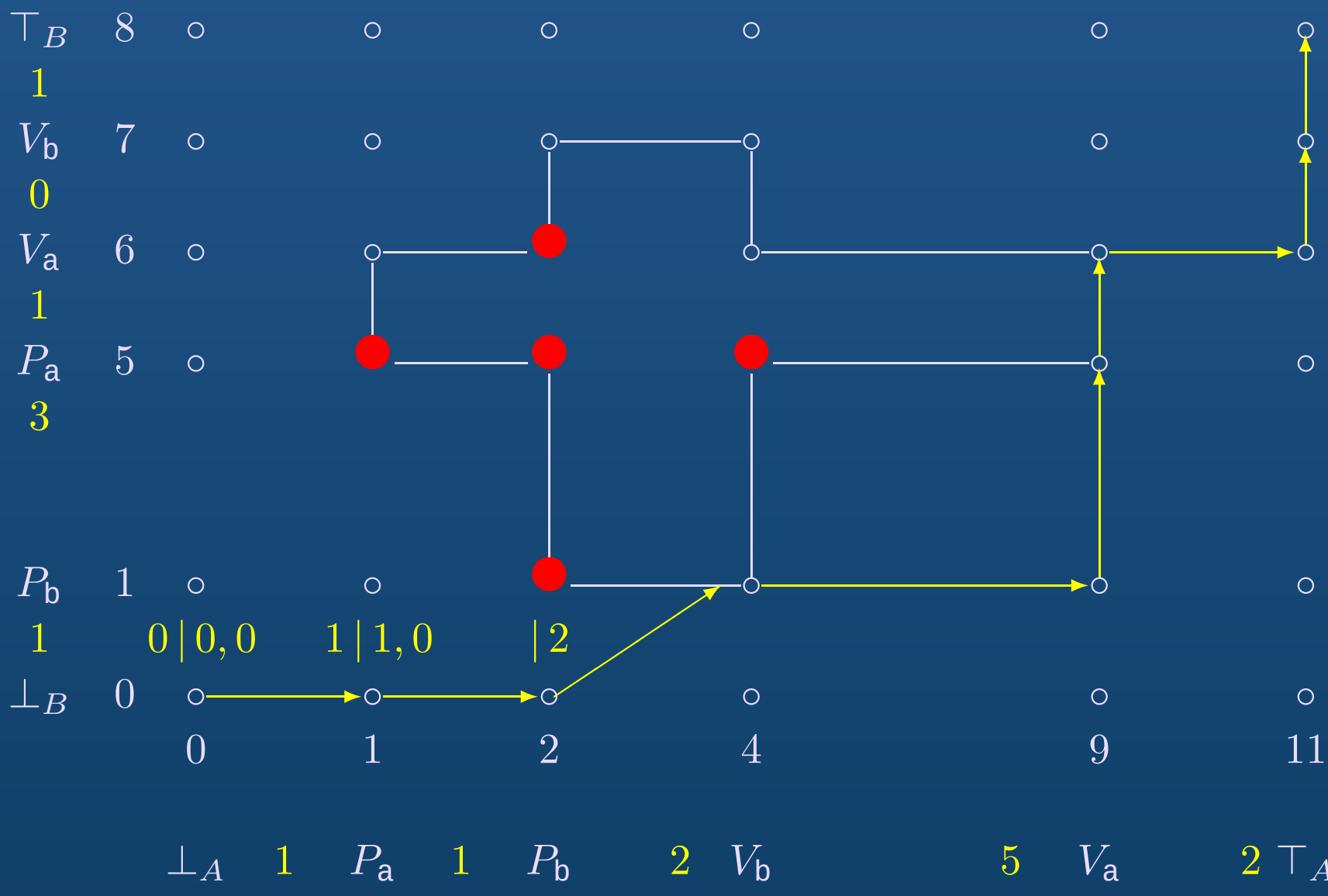
Example: Timed Swiss Flag



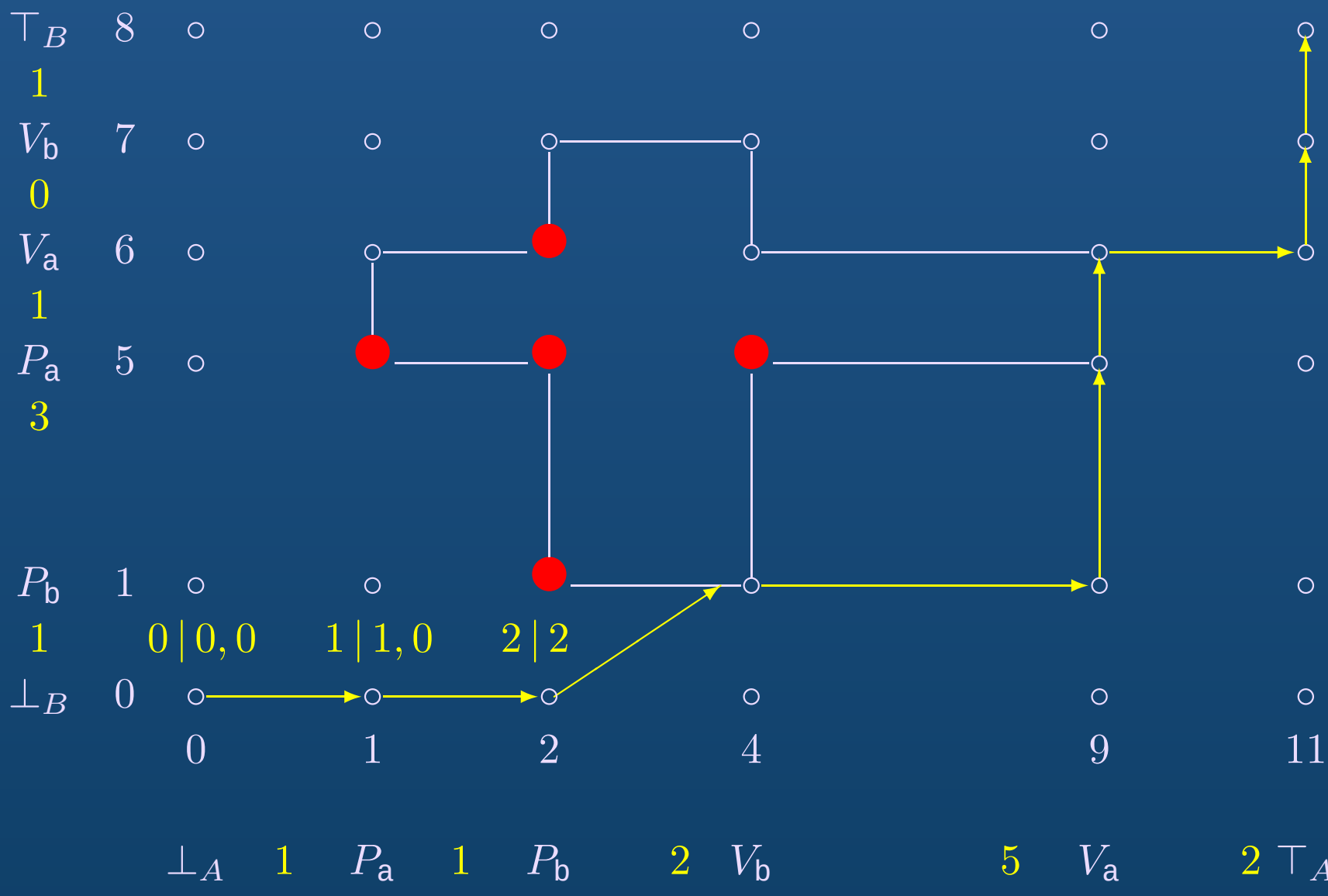
Example: Timed Swiss Flag



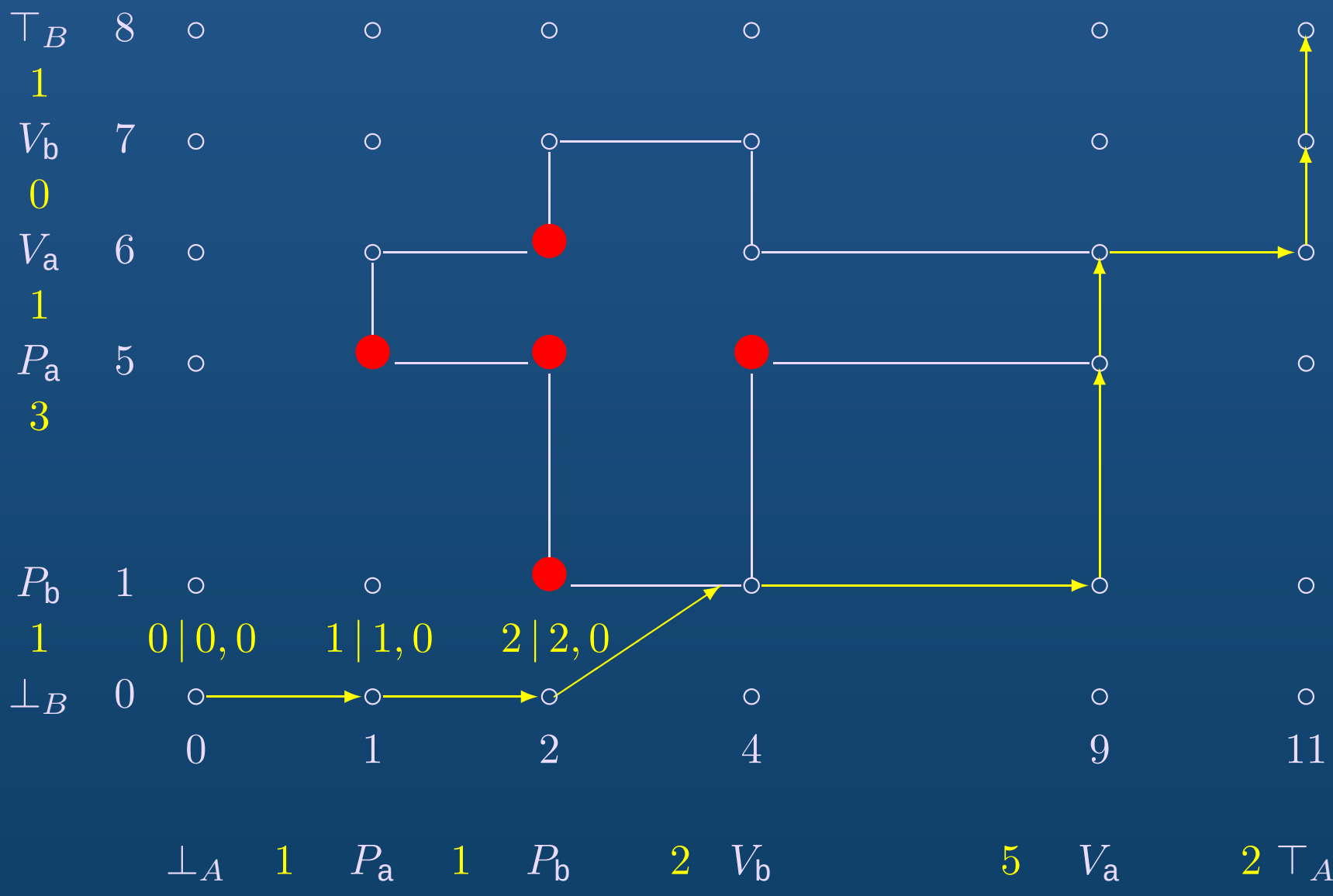
Example: Timed Swiss Flag



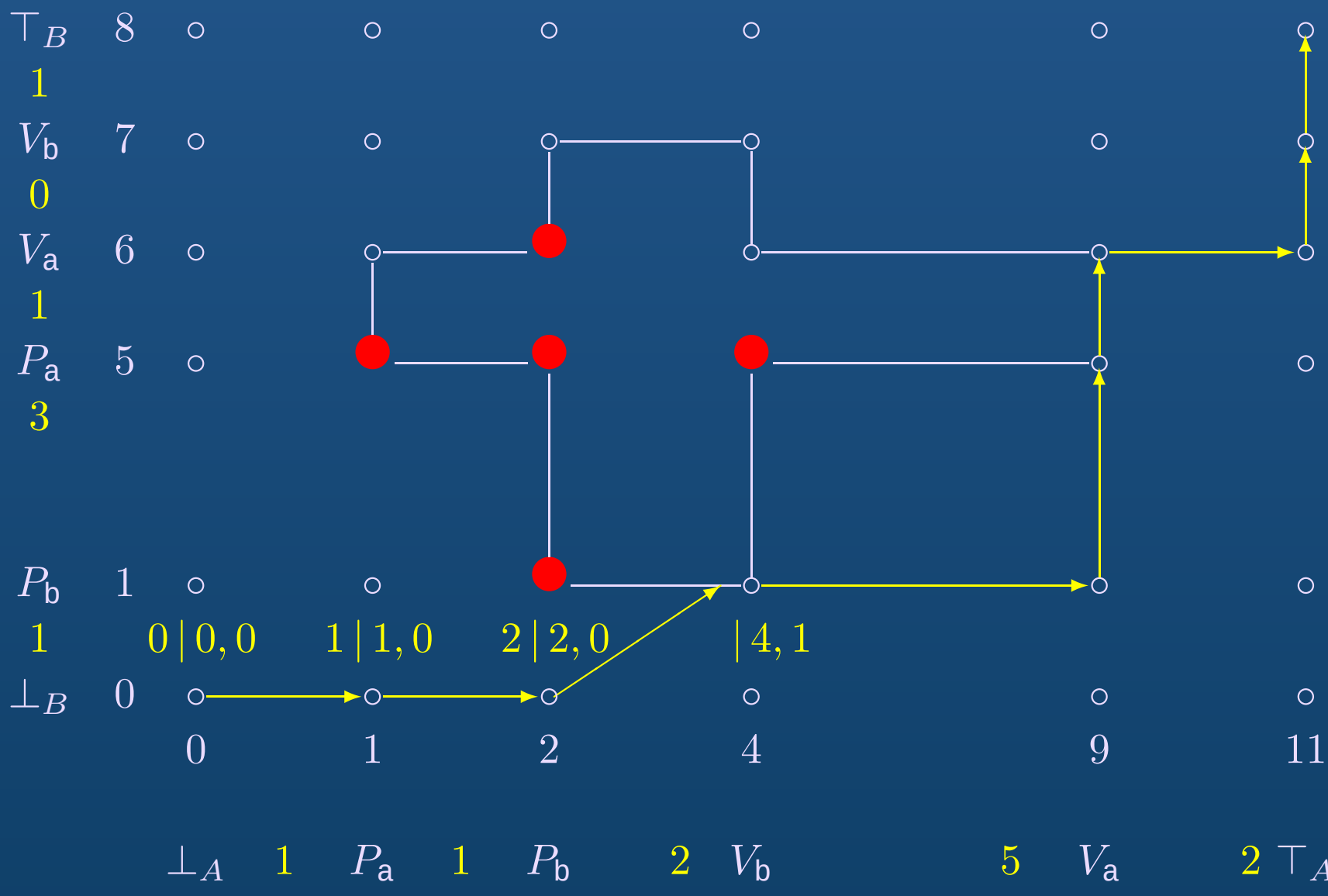
Example: Timed Swiss Flag



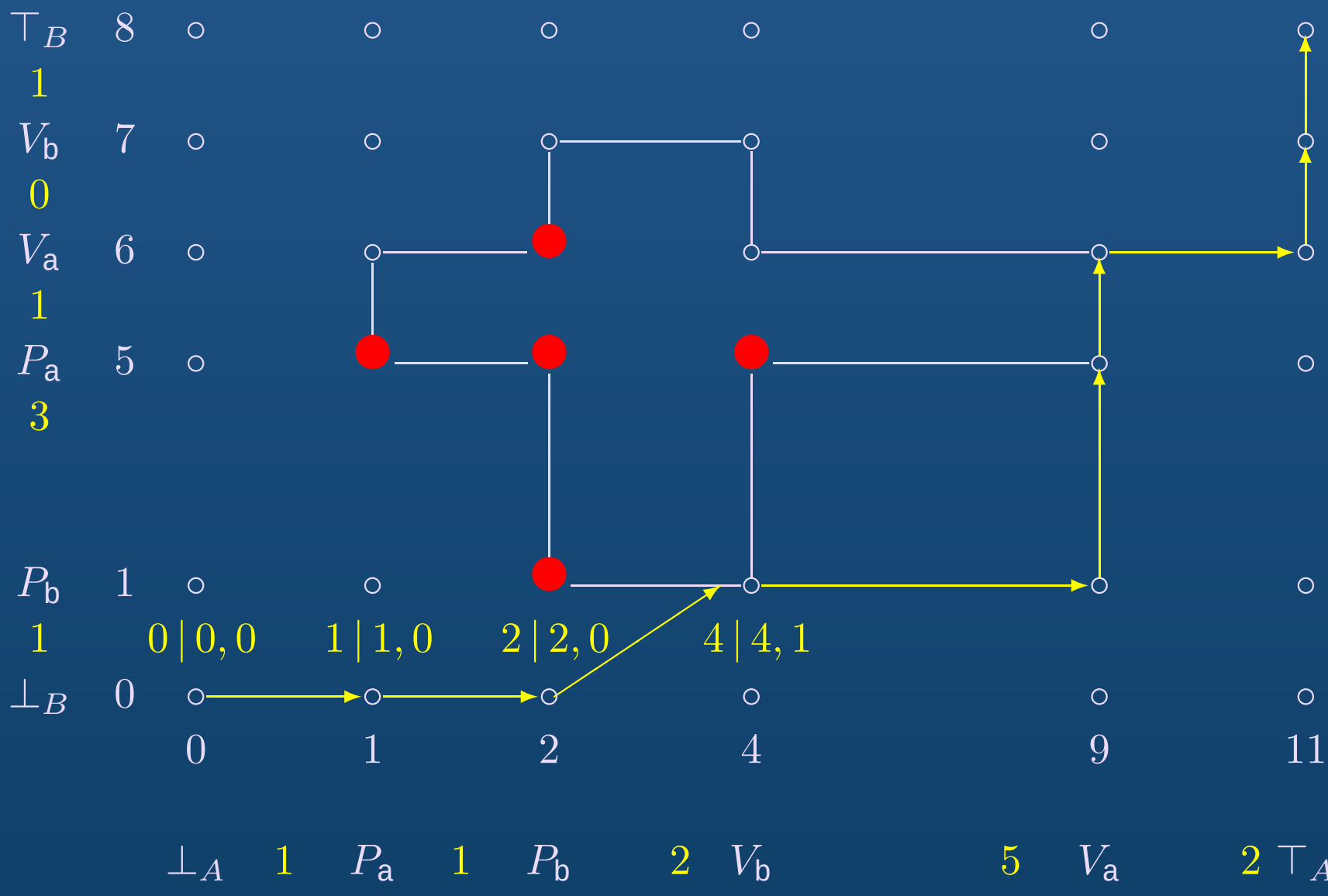
Example: Timed Swiss Flag



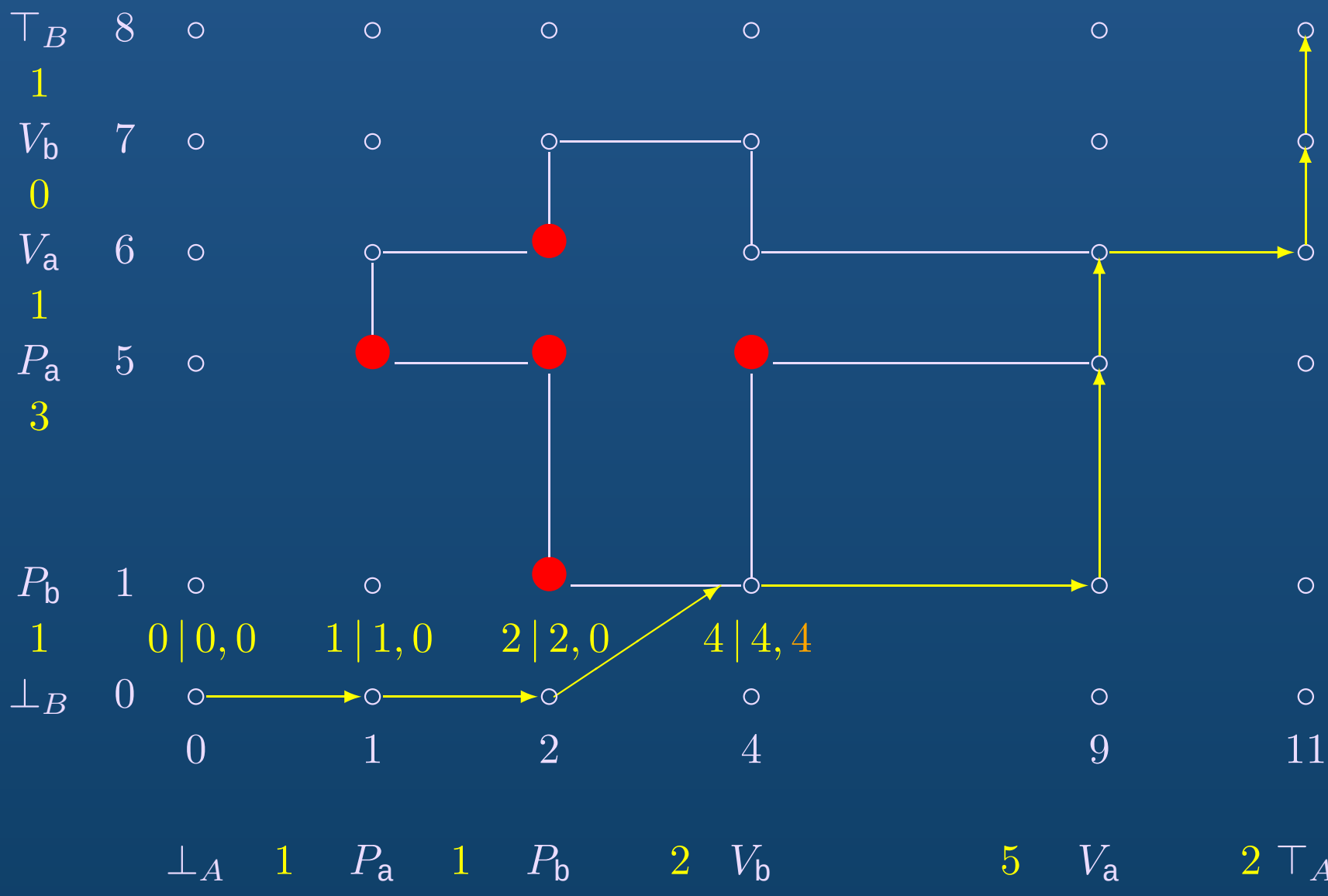
Example: Timed Swiss Flag



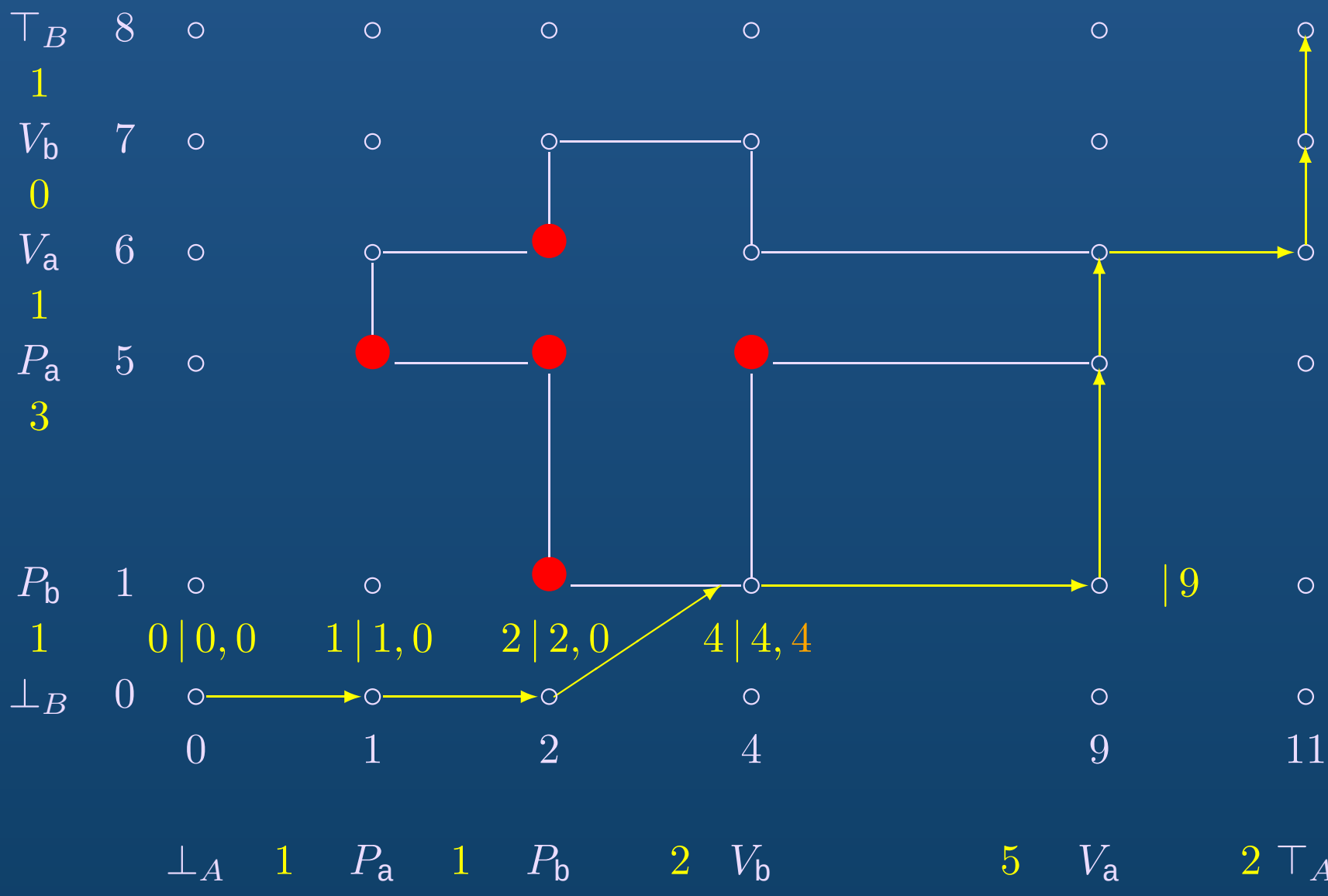
Example: Timed Swiss Flag



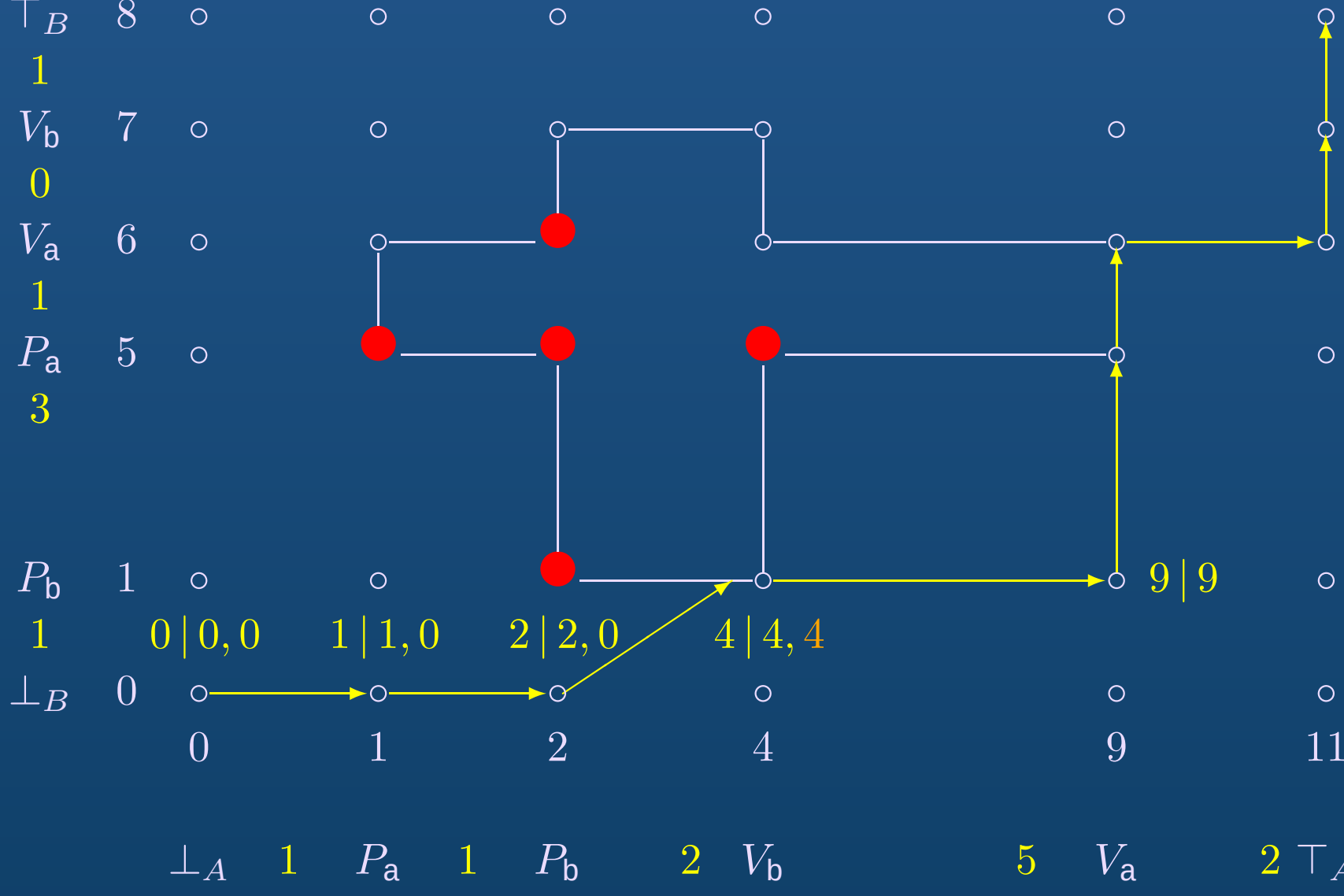
Example: Timed Swiss Flag



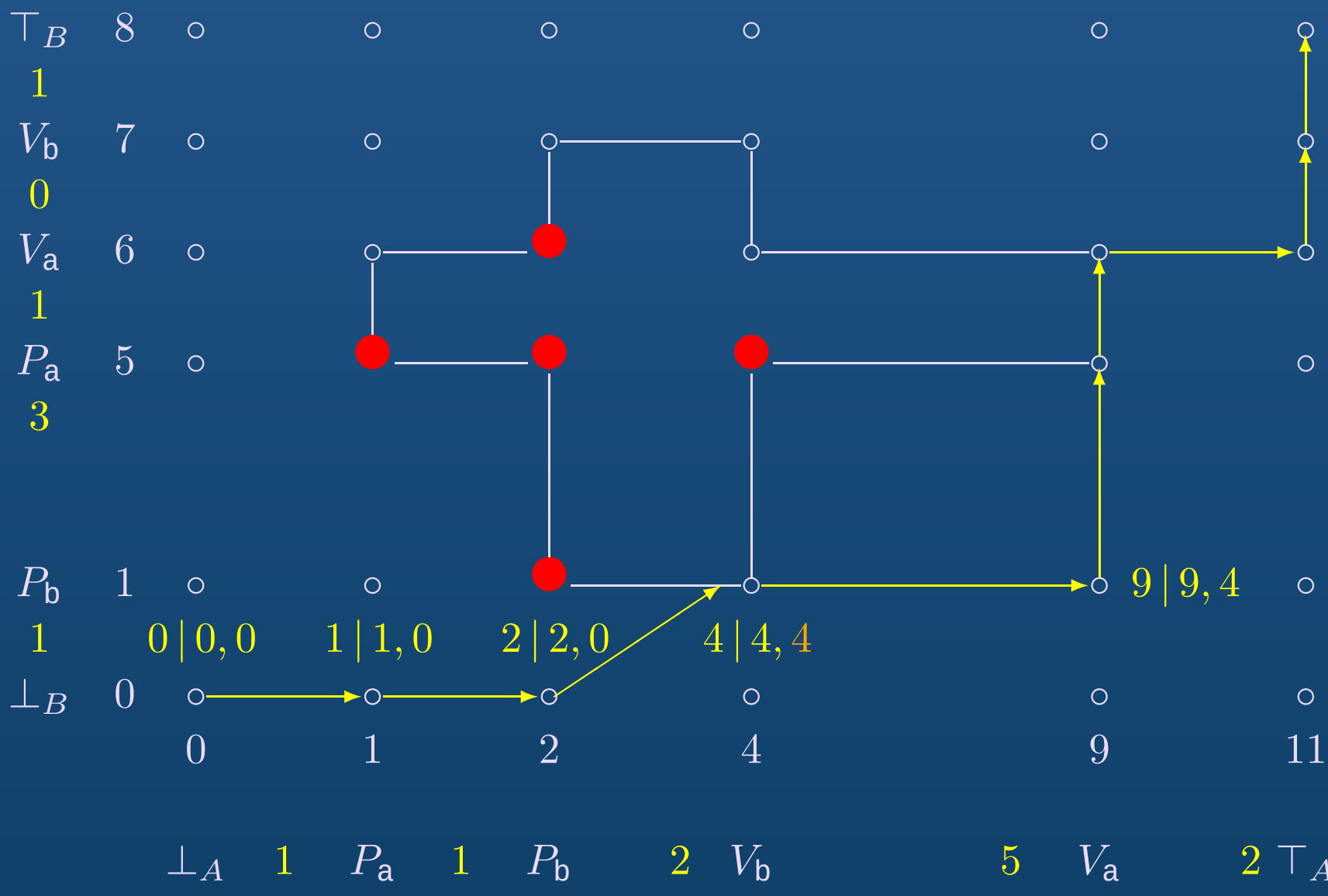
Example: Timed Swiss Flag



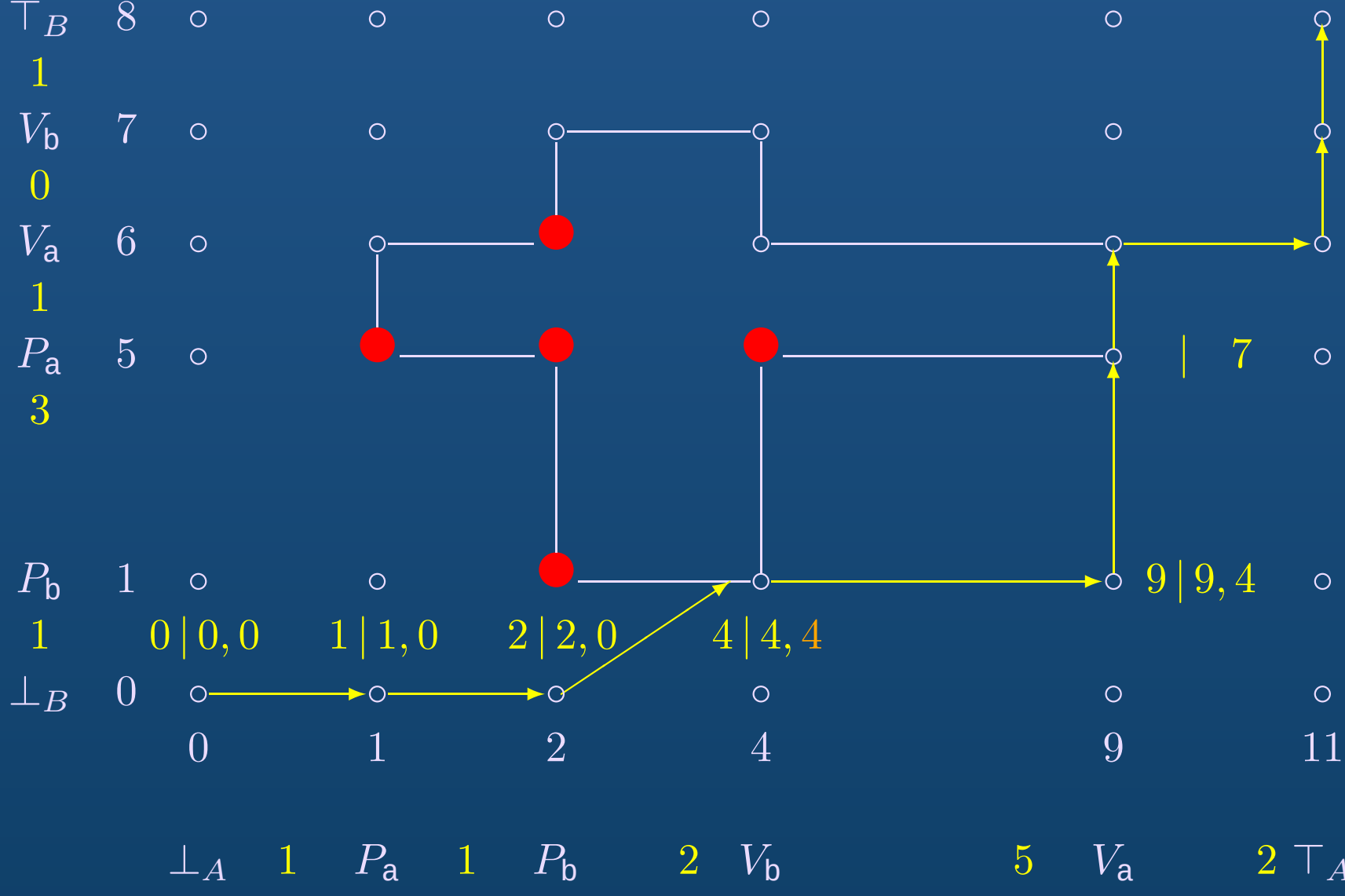
Example: Timed Swiss Flag



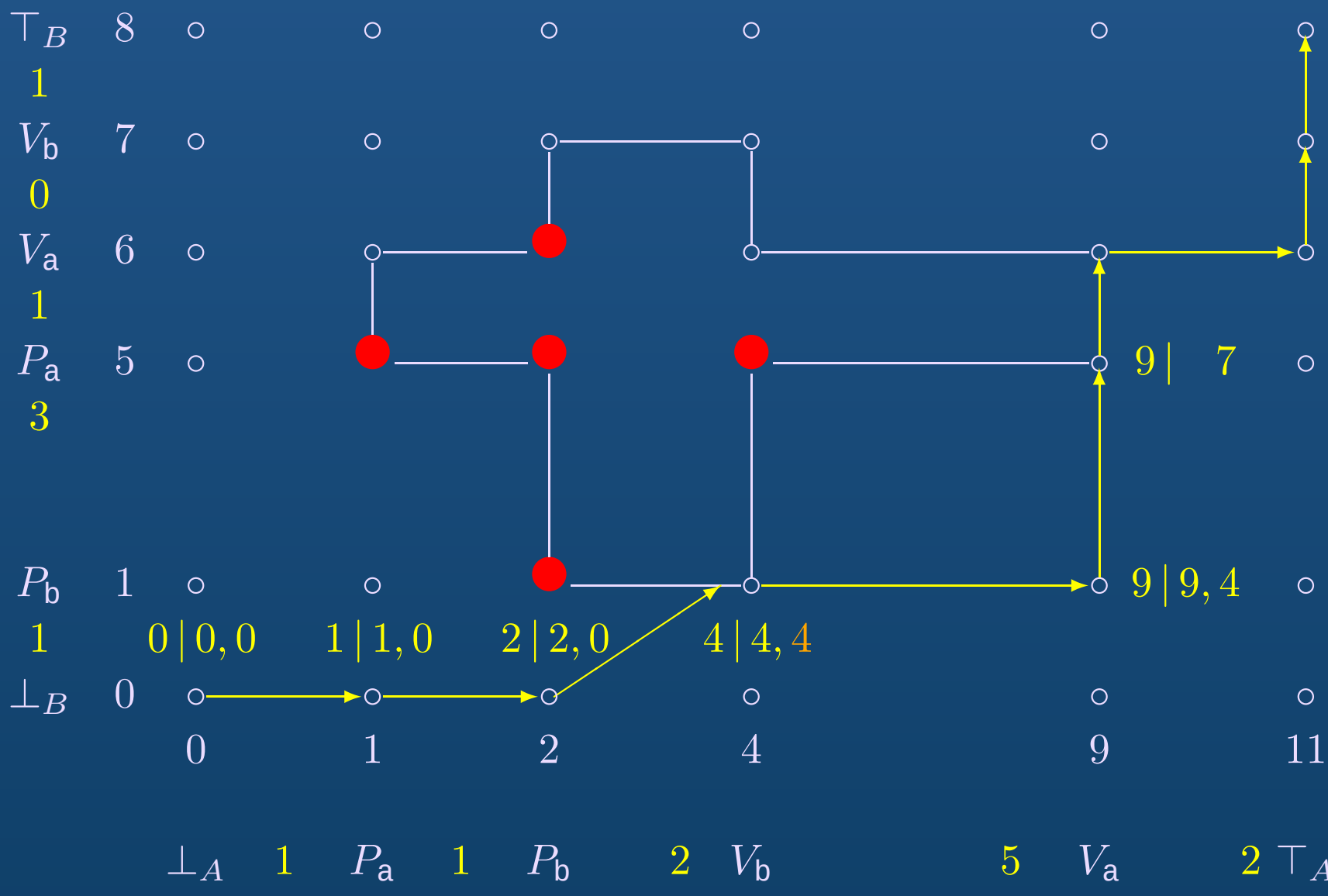
Example: Timed Swiss Flag



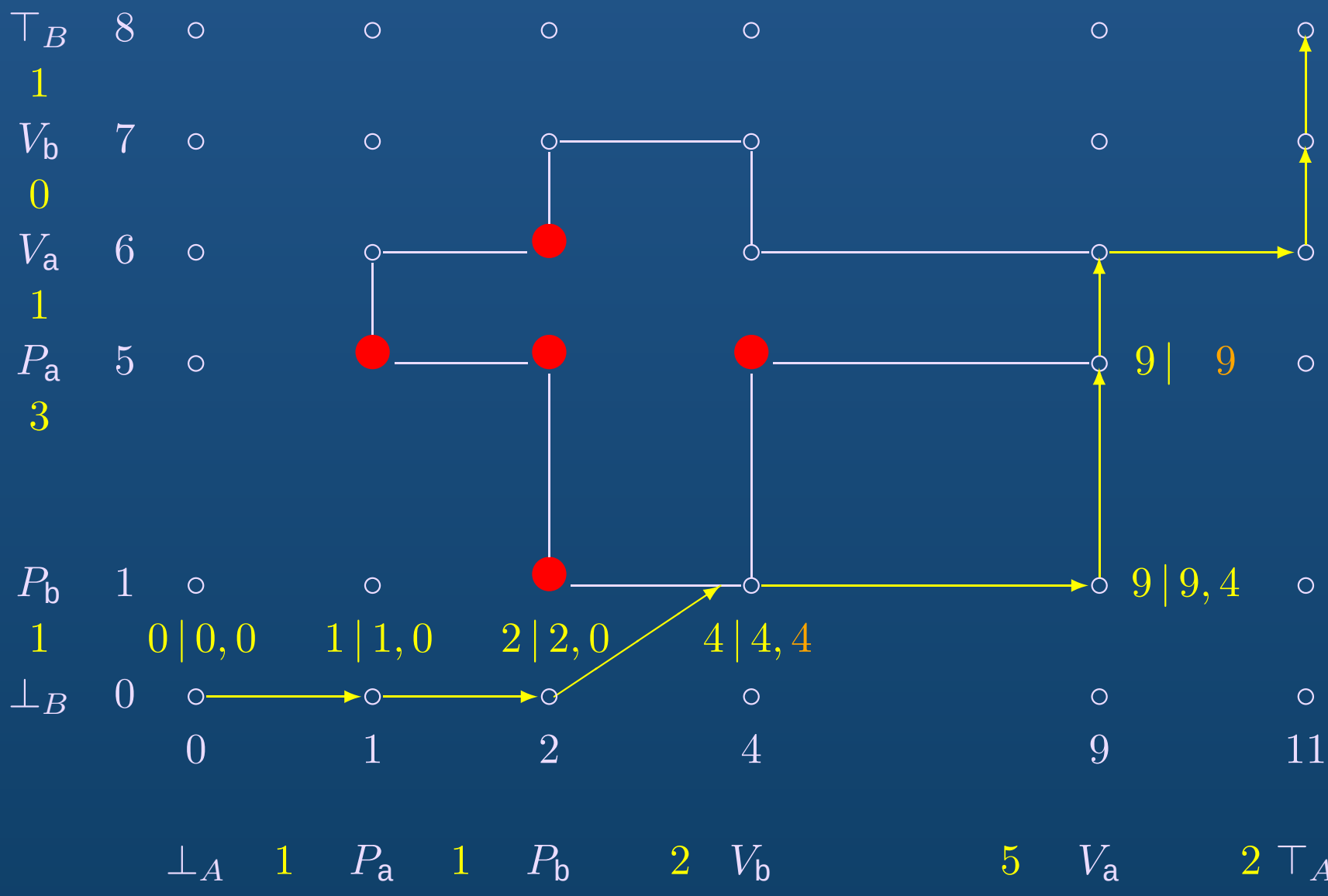
Example: Timed Swiss Flag



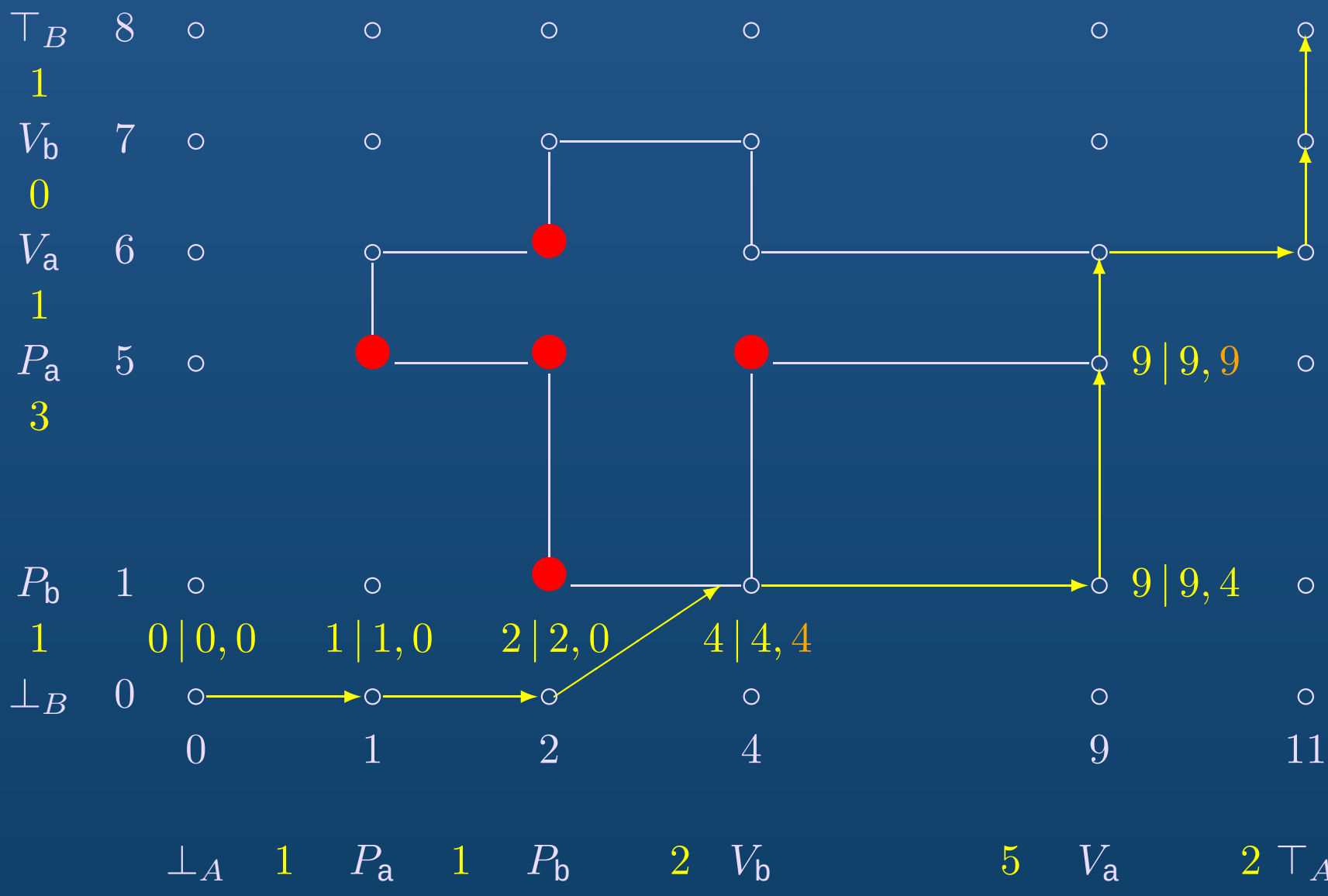
Example: Timed Swiss Flag



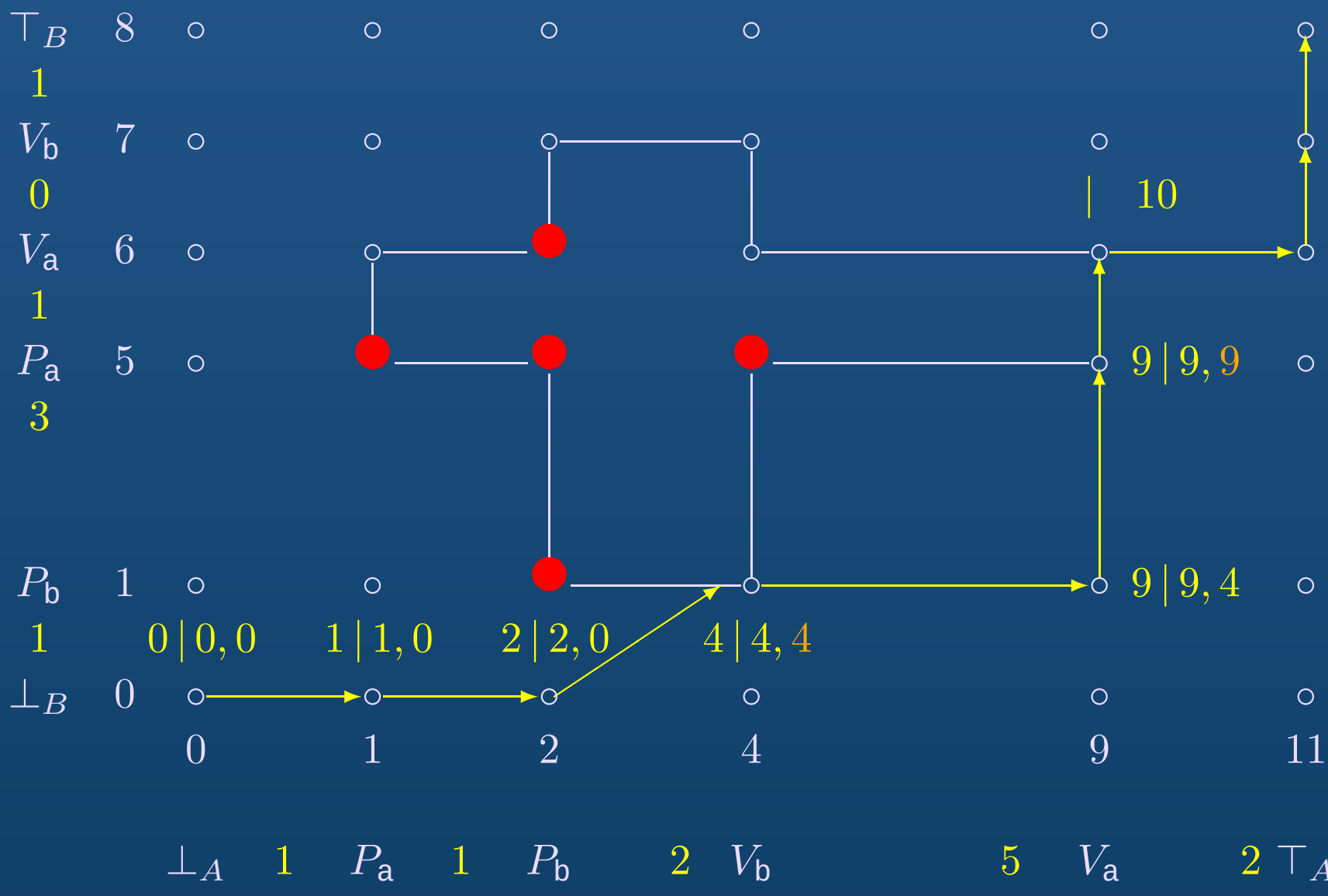
Example: Timed Swiss Flag



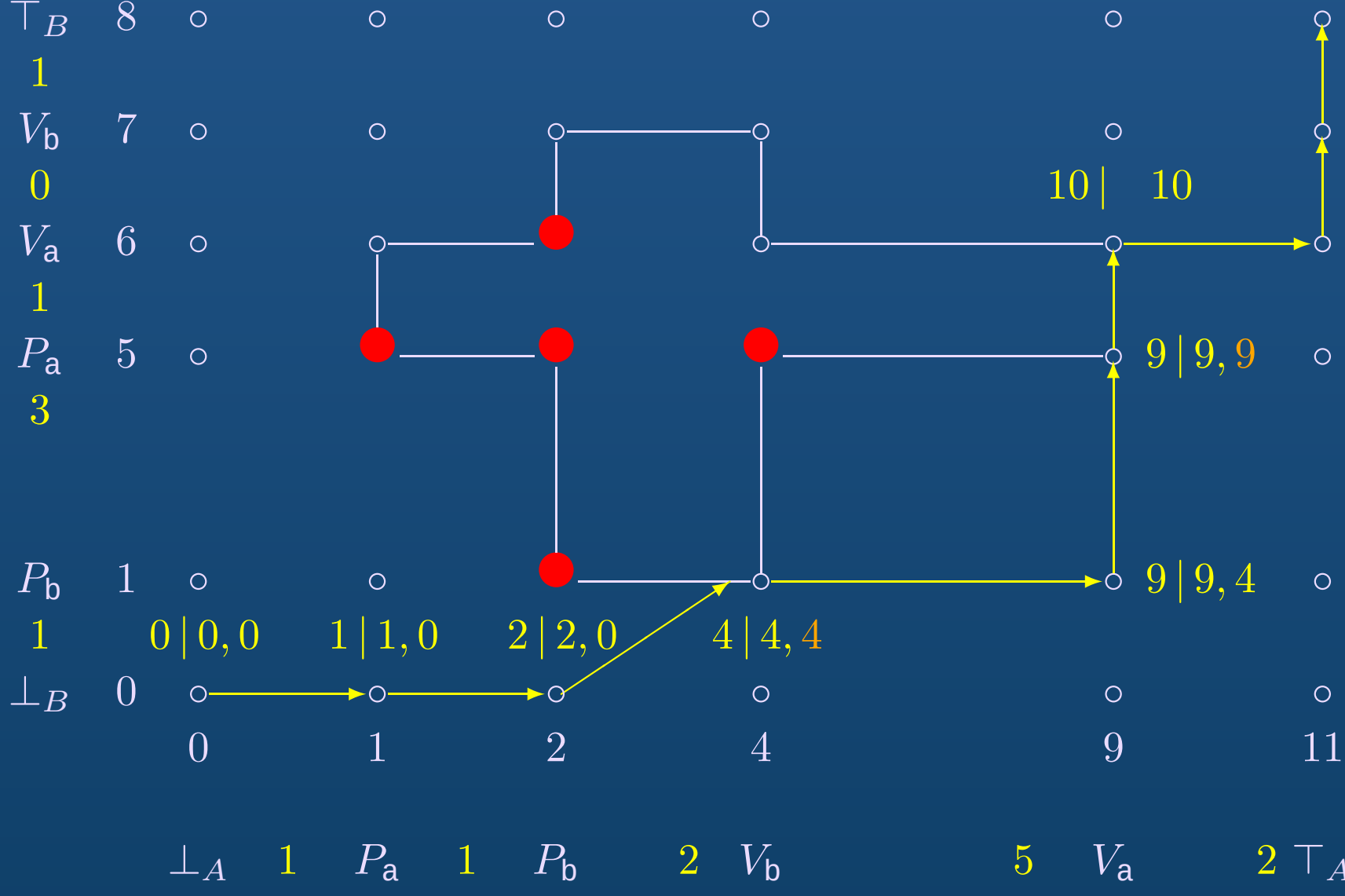
Example: Timed Swiss Flag



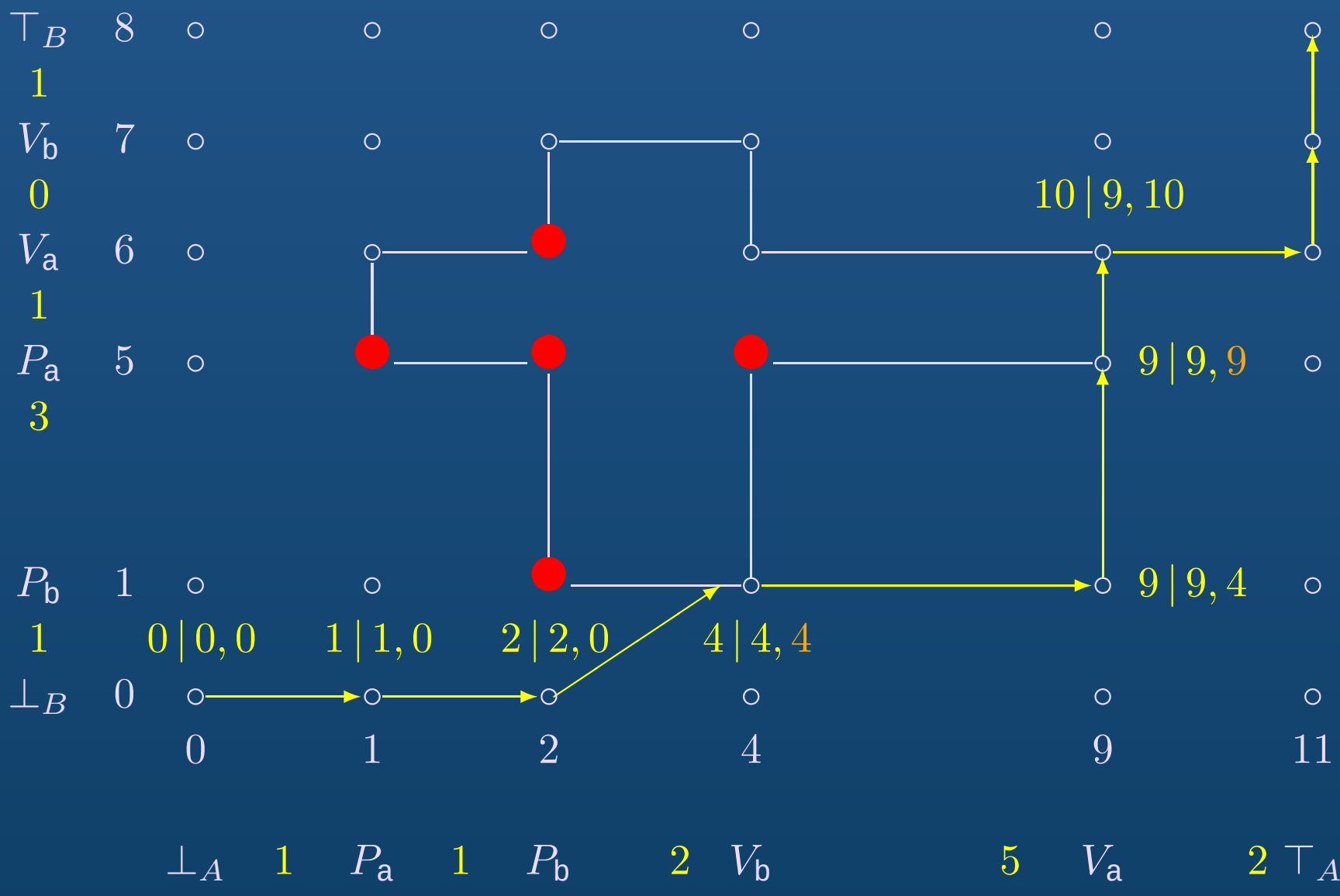
Example: Timed Swiss Flag



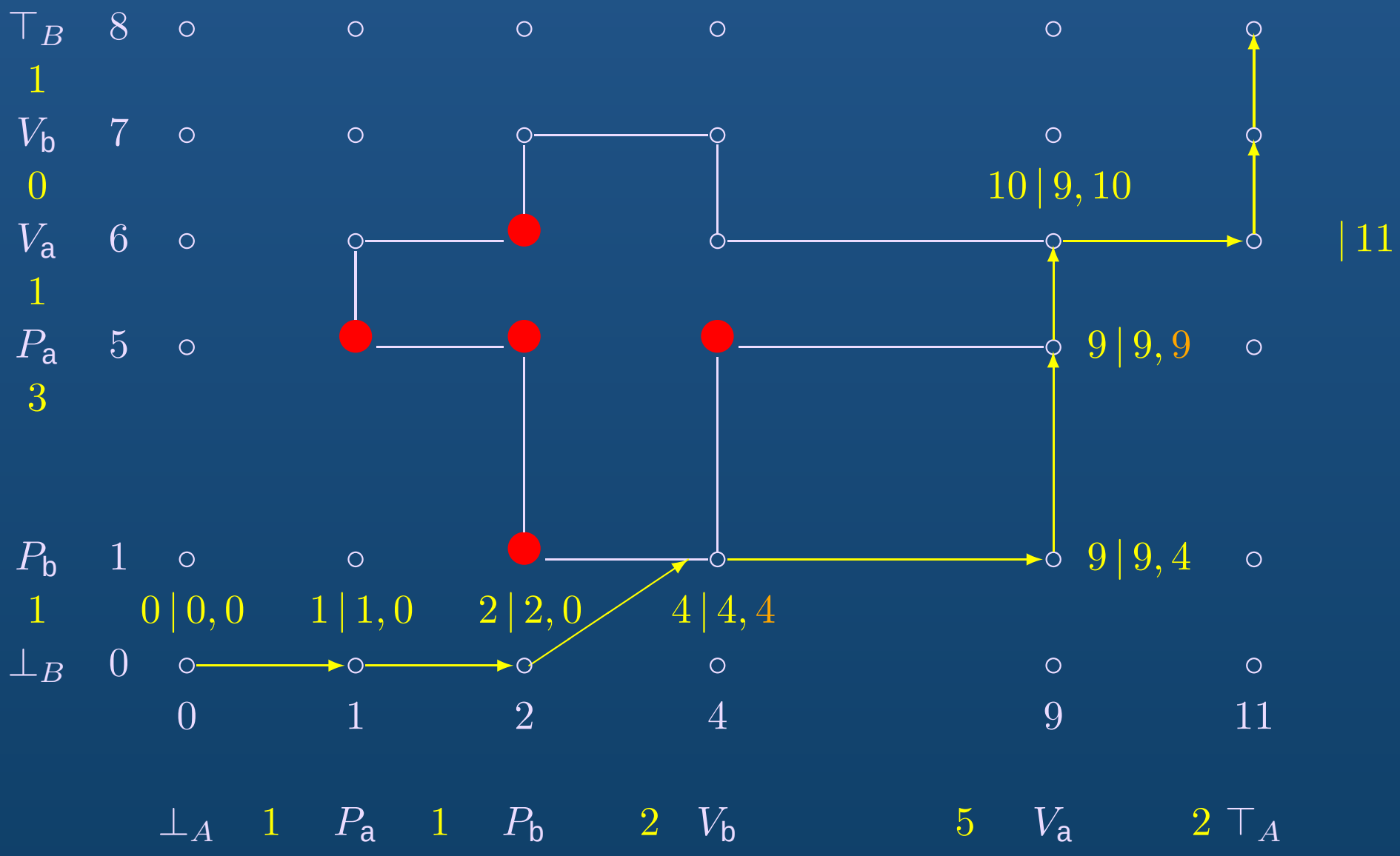
Example: Timed Swiss Flag



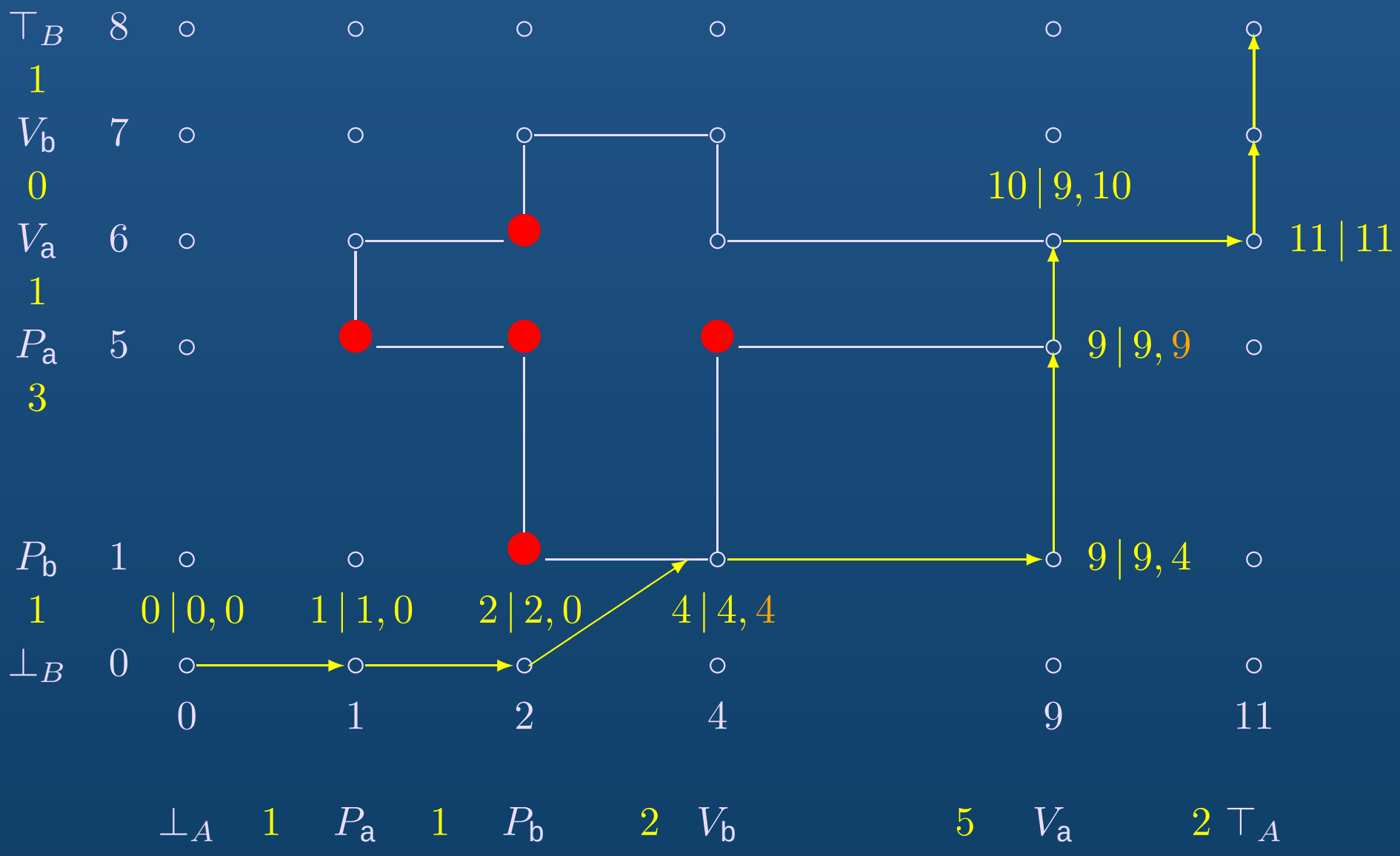
Example: Timed Swiss Flag



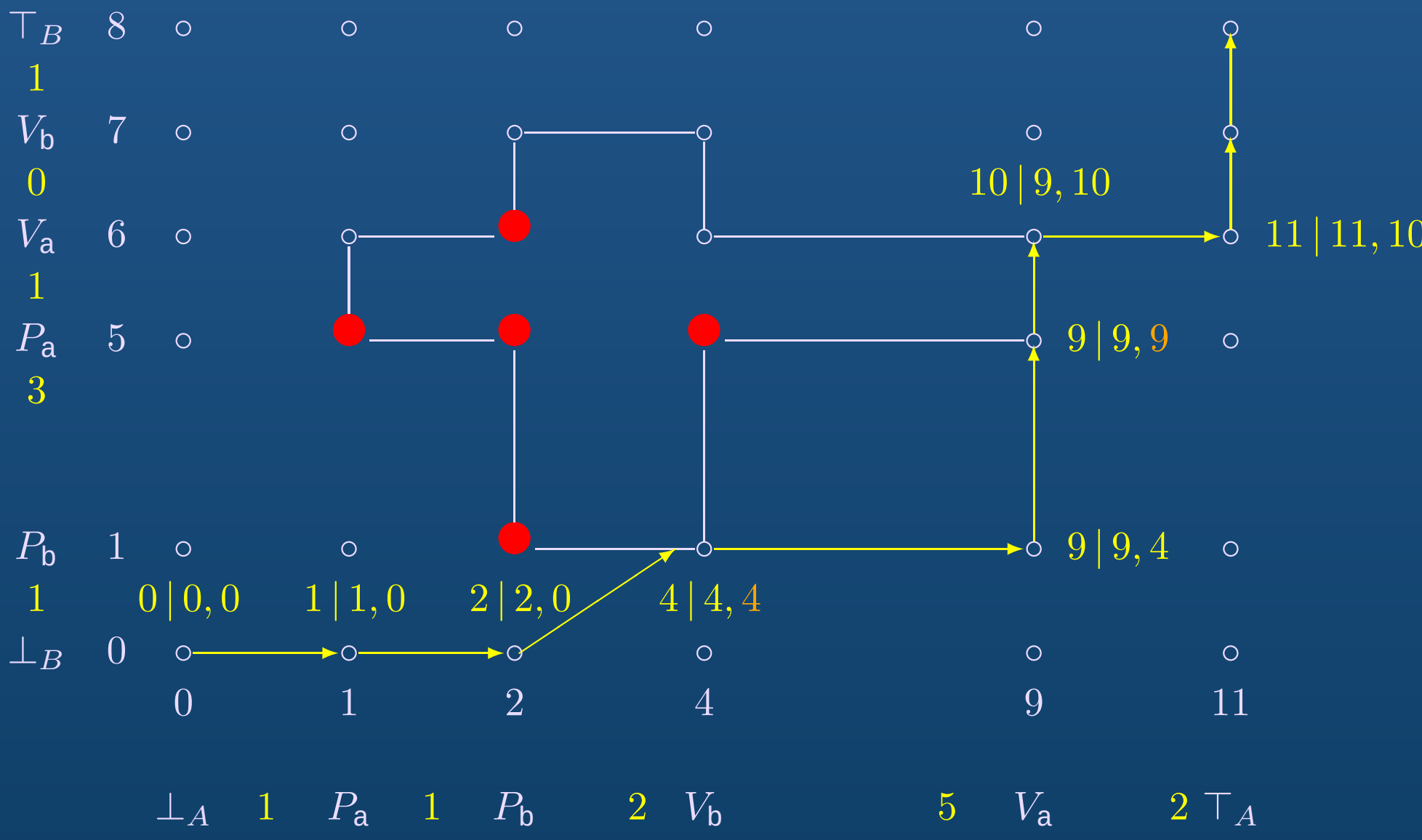
Example: Timed Swiss Flag



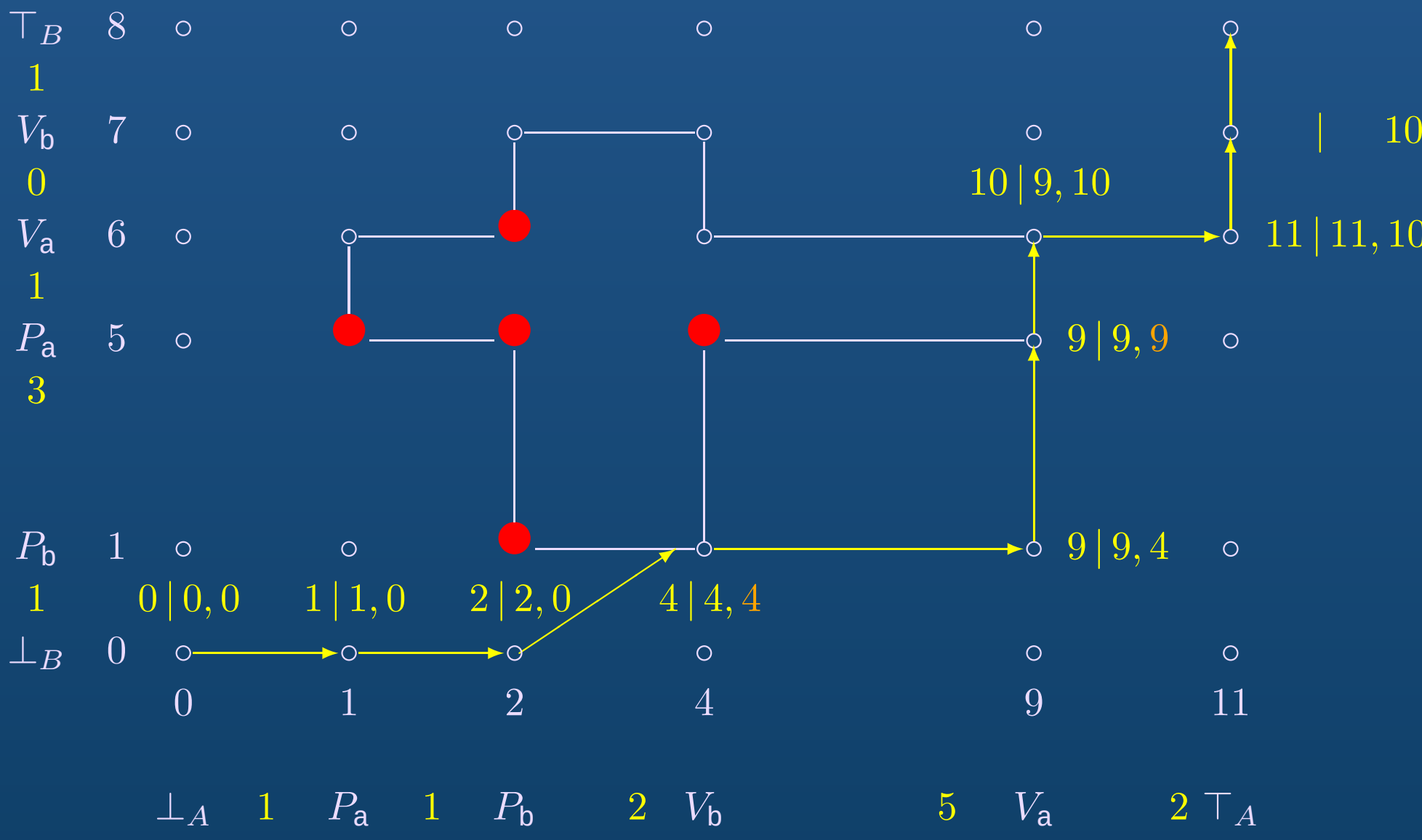
Example: Timed Swiss Flag



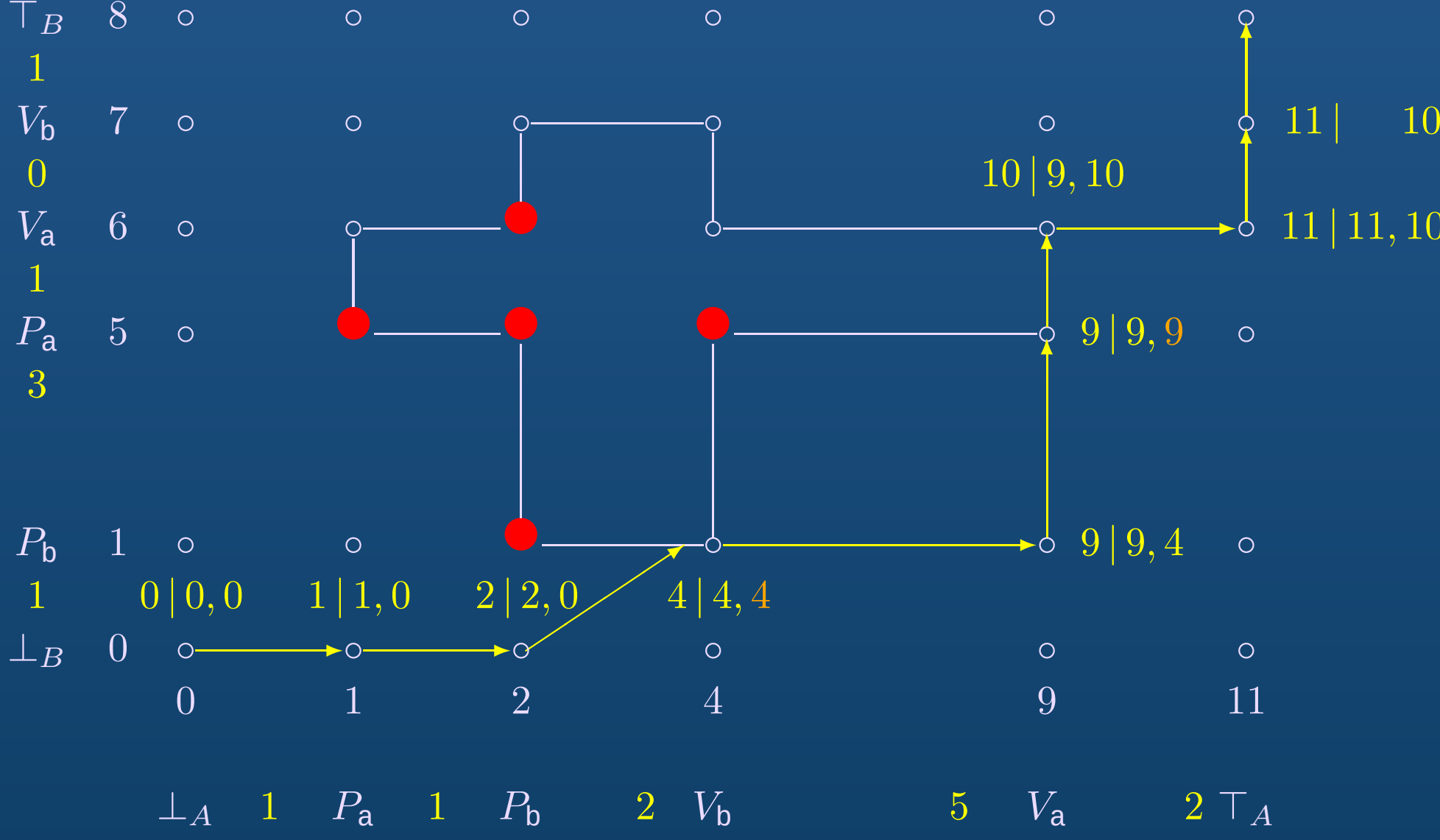
Example: Timed Swiss Flag



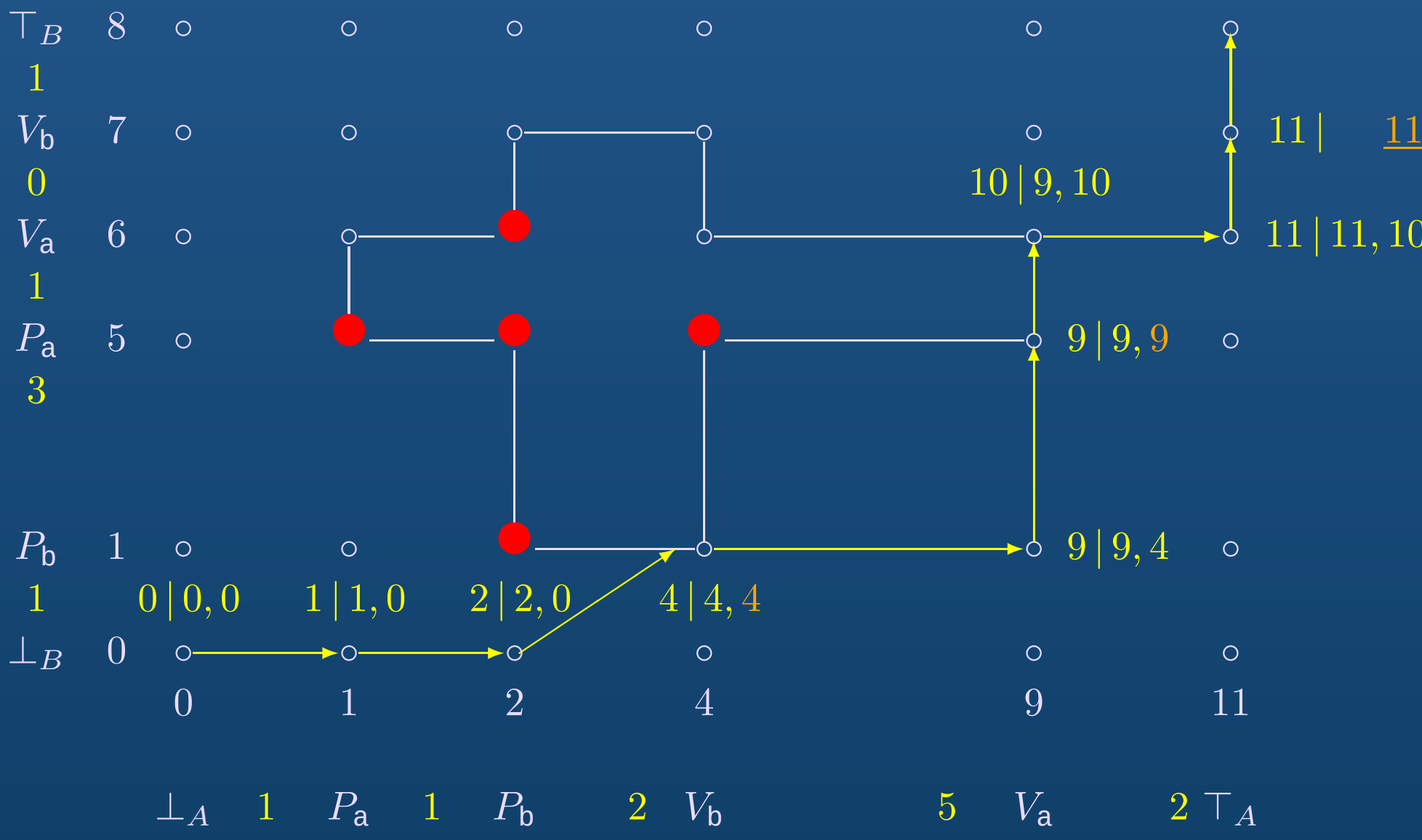
Example: Timed Swiss Flag



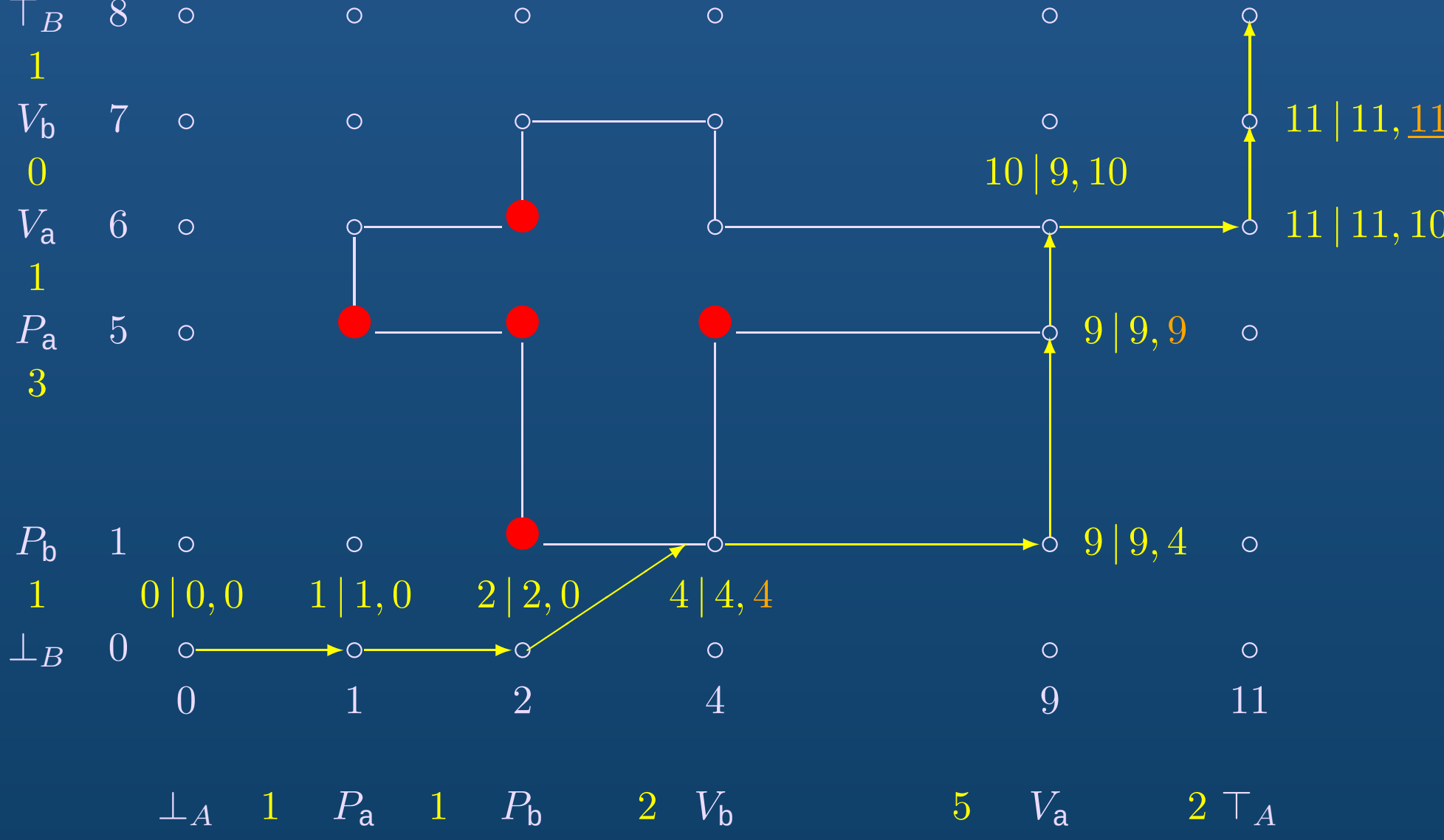
Example: Timed Swiss Flag



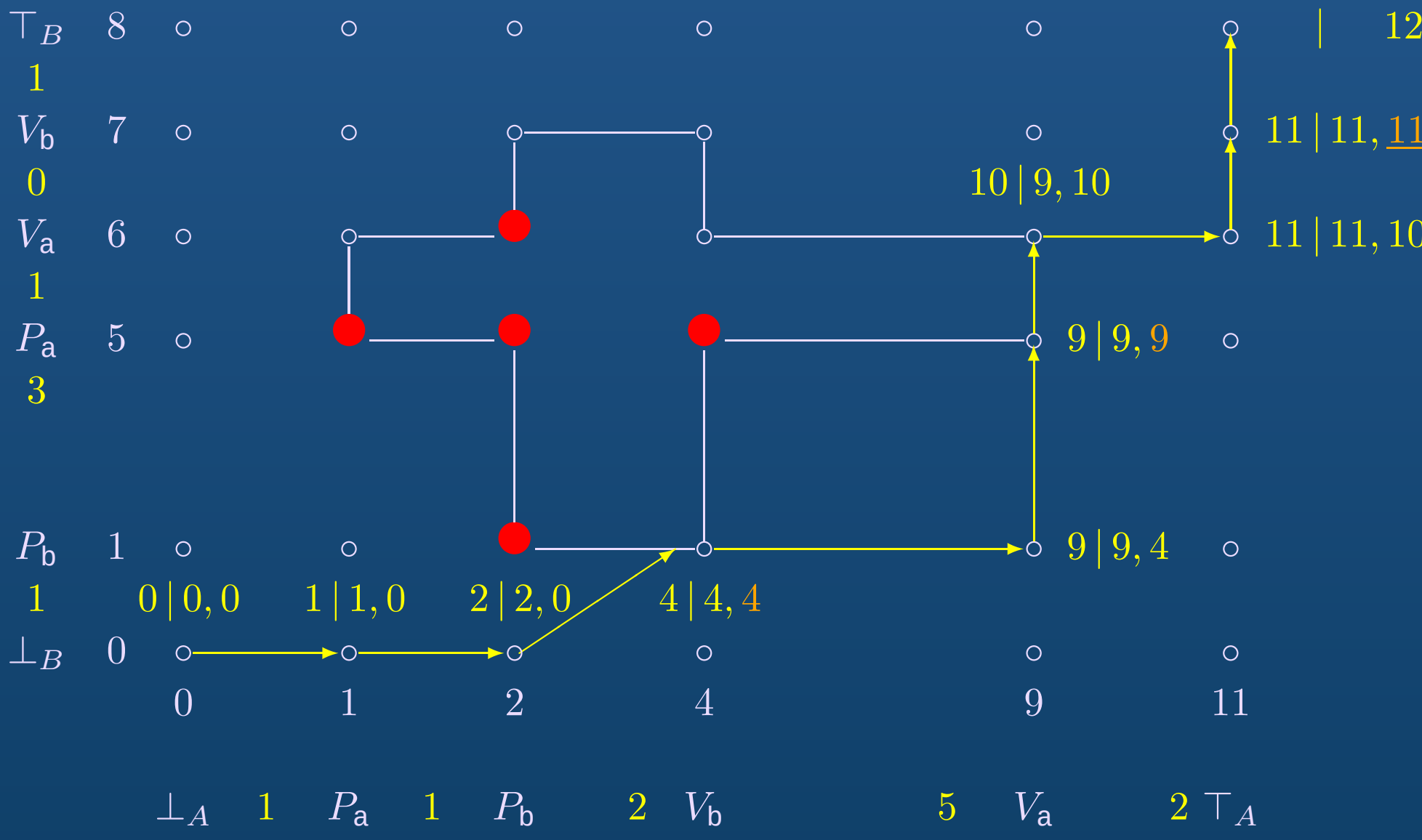
Example: Timed Swiss Flag



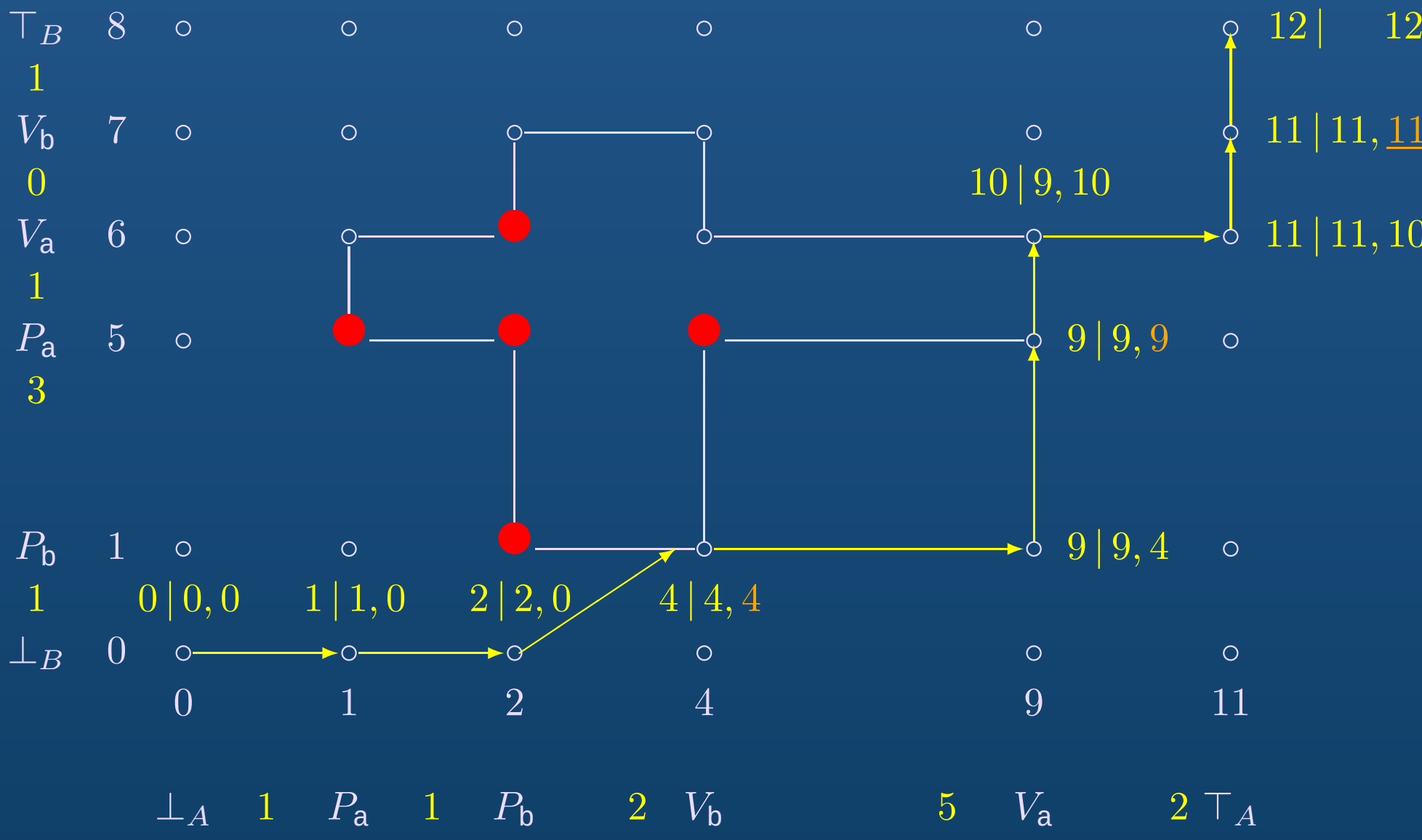
Example: Timed Swiss Flag



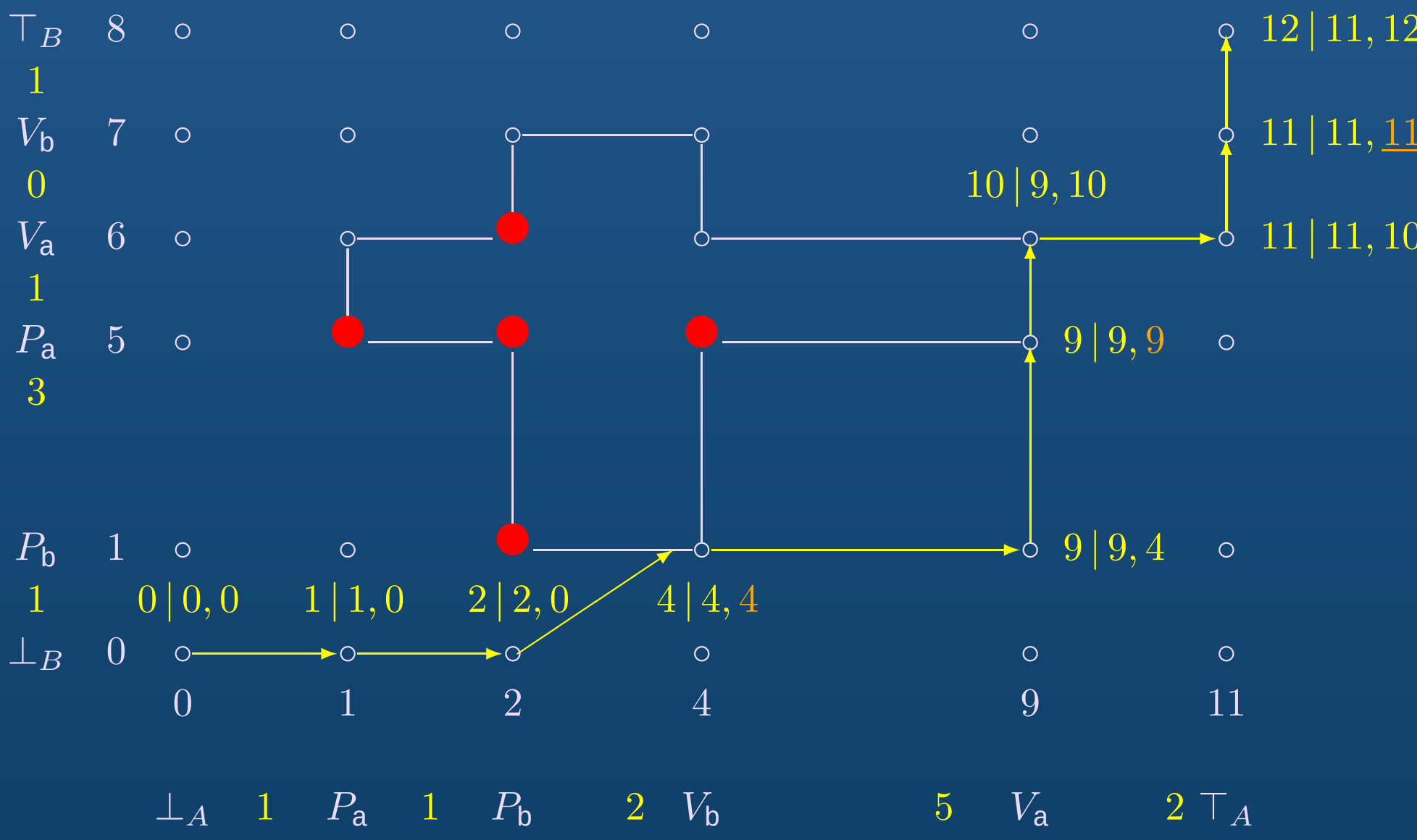
Example: Timed Swiss Flag



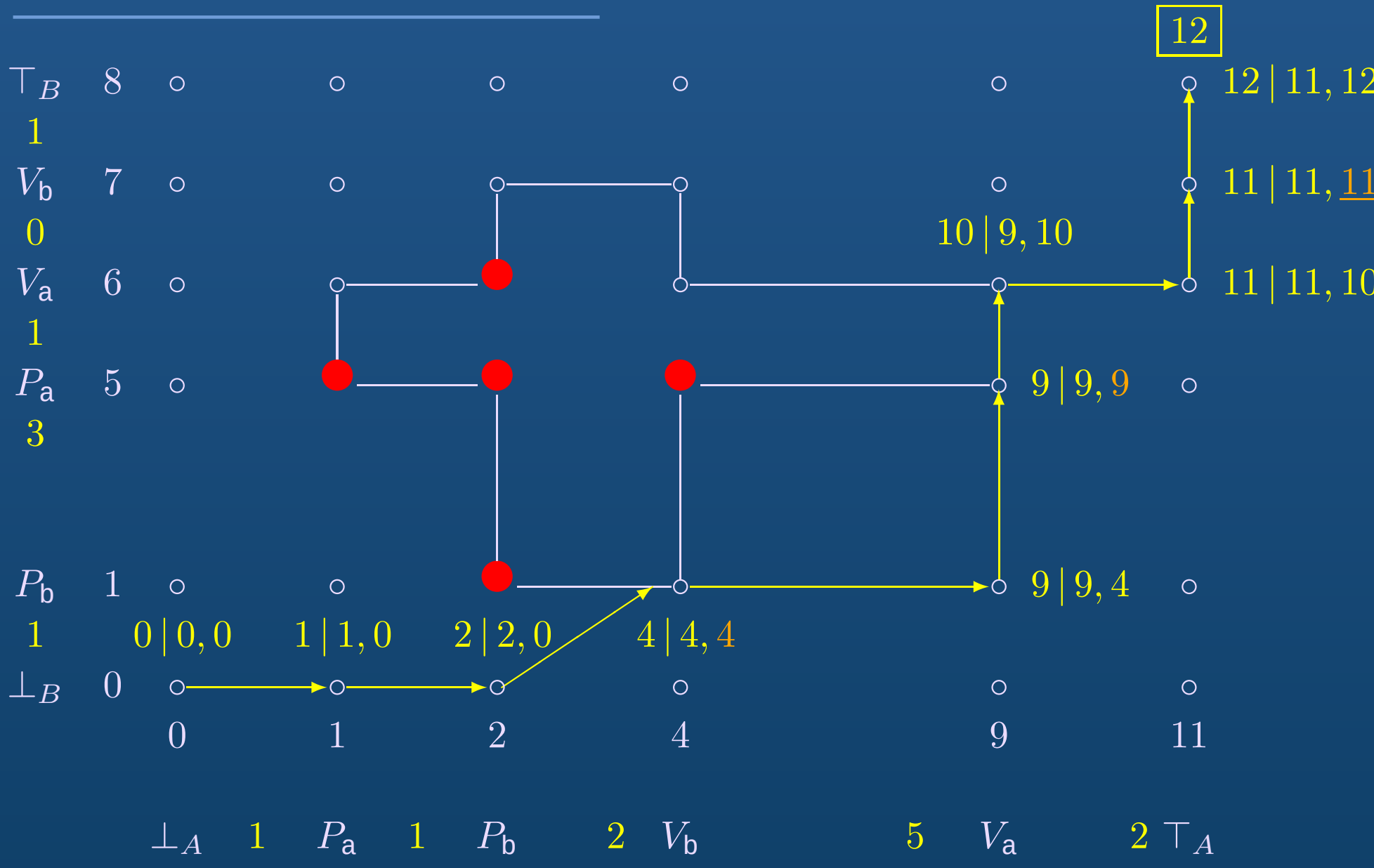
Example: Timed Swiss Flag



Example: Timed Swiss Flag



Example: Timed Swiss Flag



Computing Efficient Schedules

Goal

- Compute efficient schedules w.r.t. WCRT
  and deadlock-free
- Exploiting geometric properties of timed PV diagrams

Problematics

- Large number of schedules
- Computing the WCRT: history-dependent

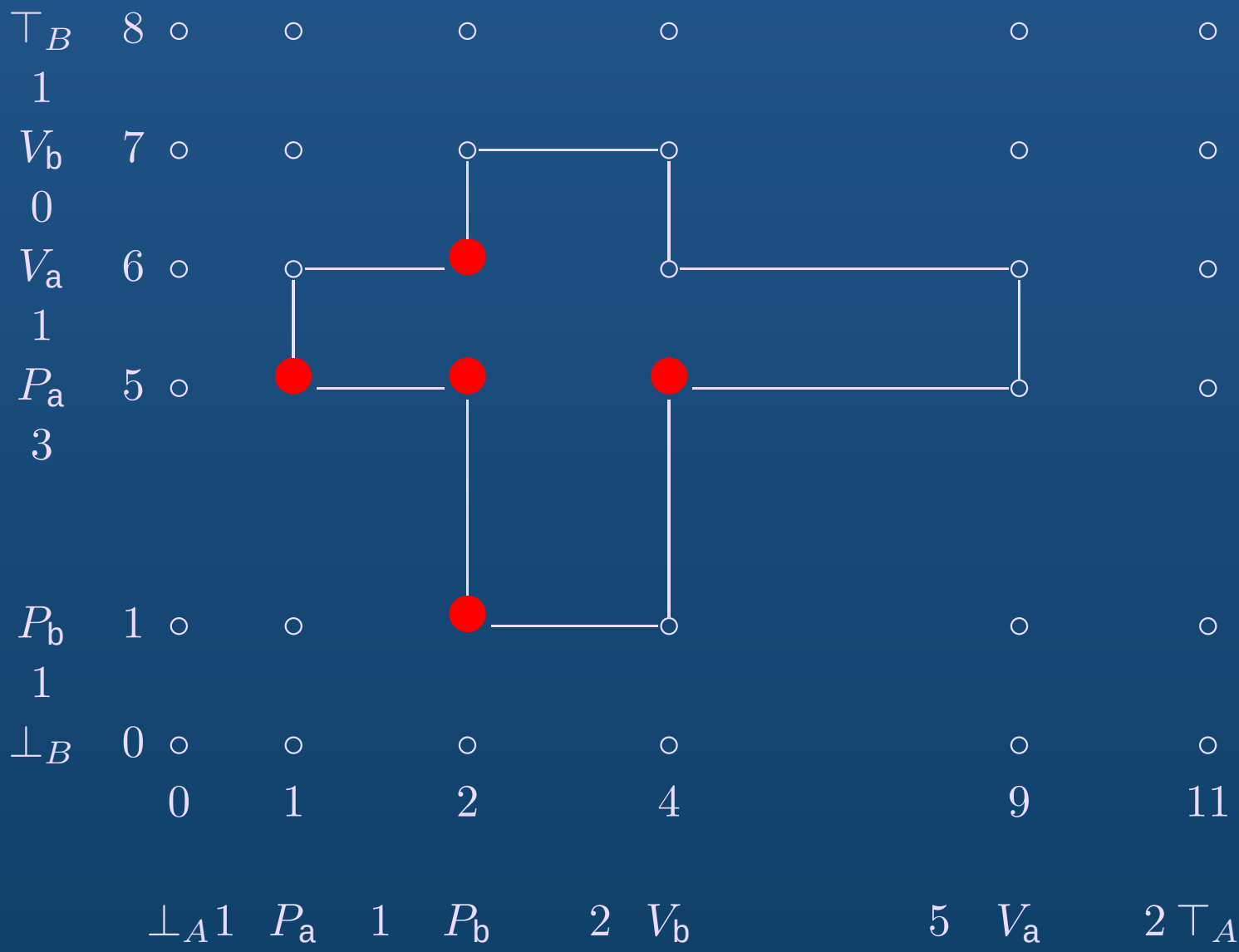
Some definitions

- Potential exchange point ϵ
  - where a resource can be exchanged
  - called critical if $accessing(r, \epsilon) = limit(r)$
- Distance between two states: length of the longest side of $box(\epsilon, \epsilon')$
- Bow

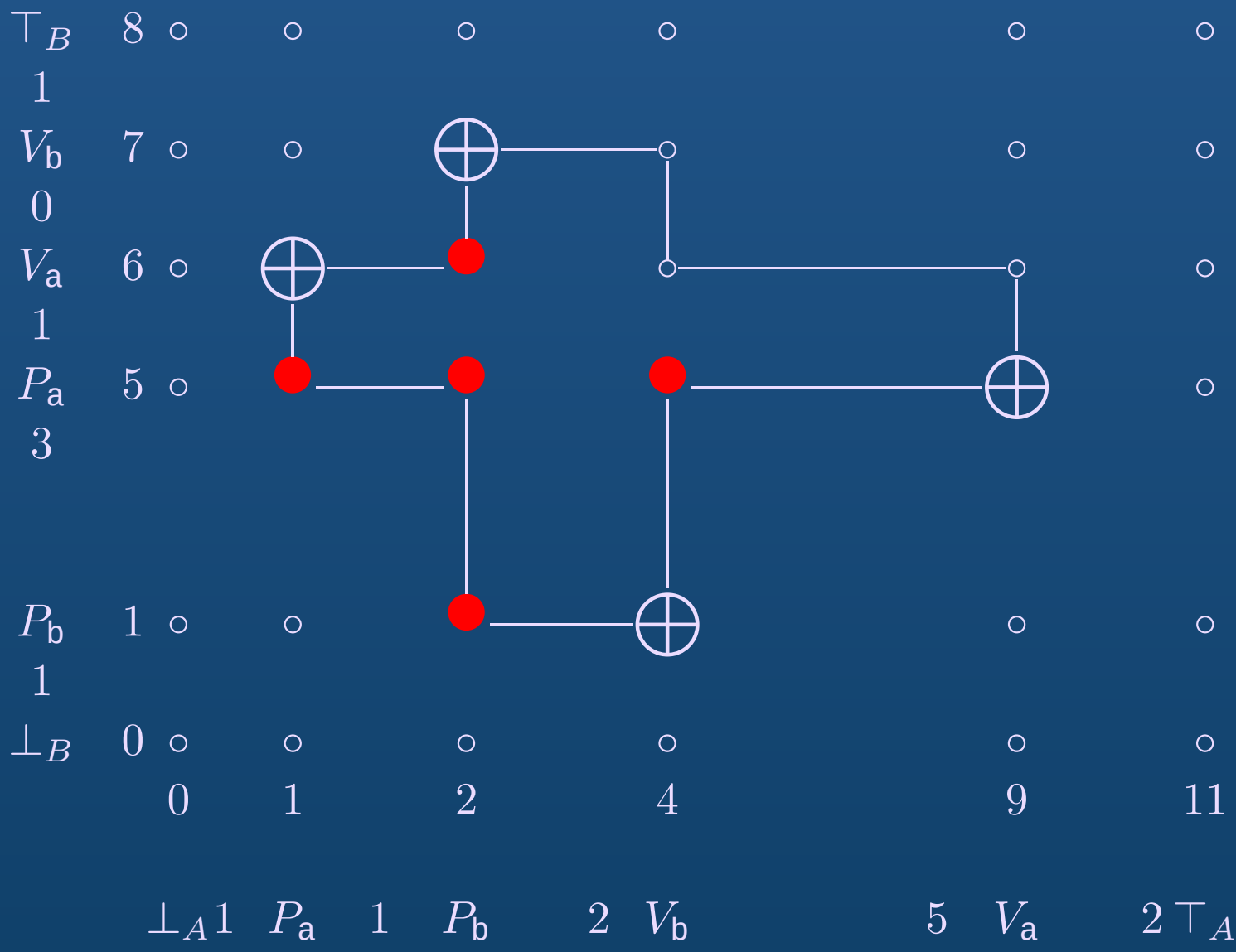
  an arrow from state ϵ to state ϵ'

  the duration of the shortest string from ϵ to state ϵ' is equal to the distance between ϵ and ϵ'

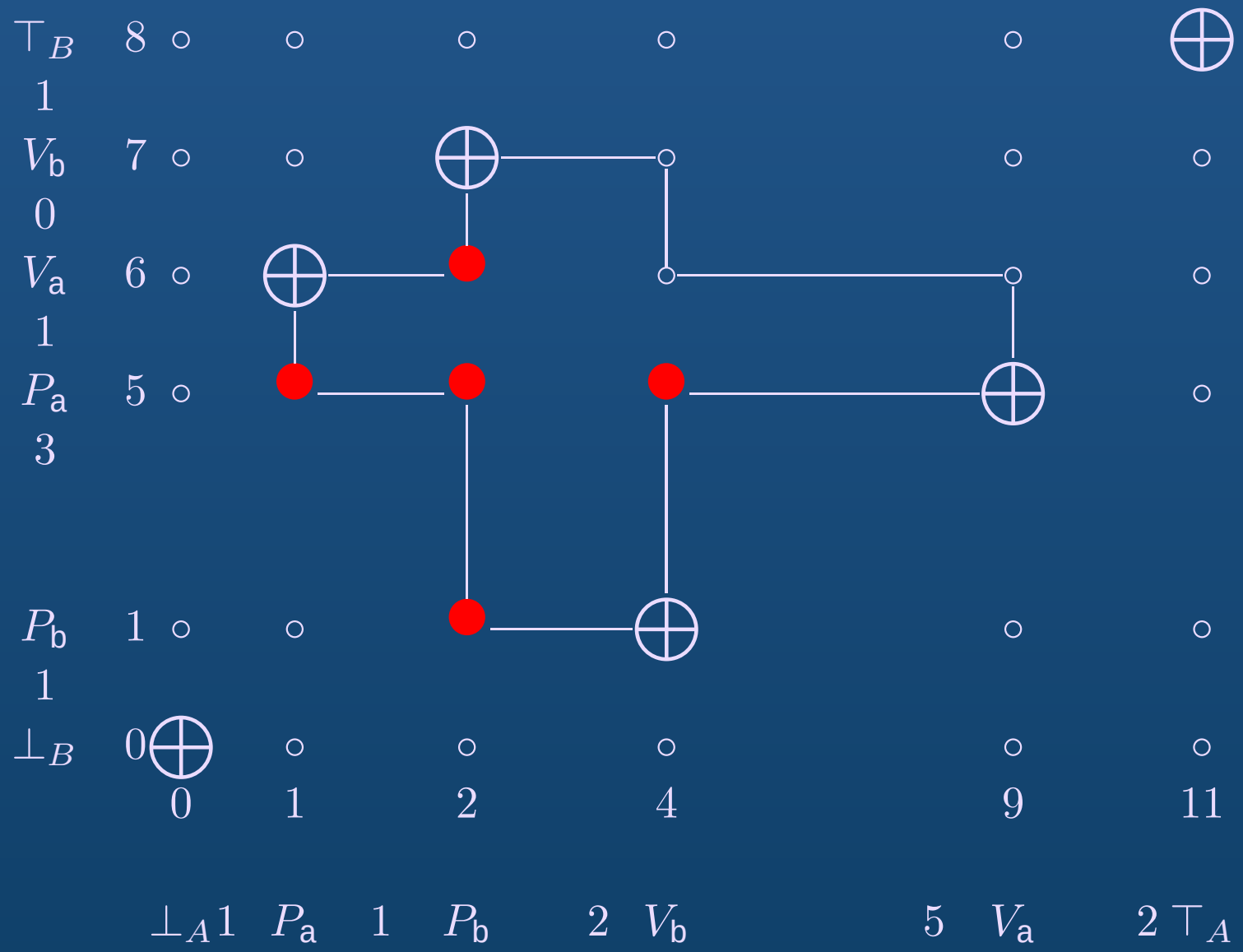
Illustration



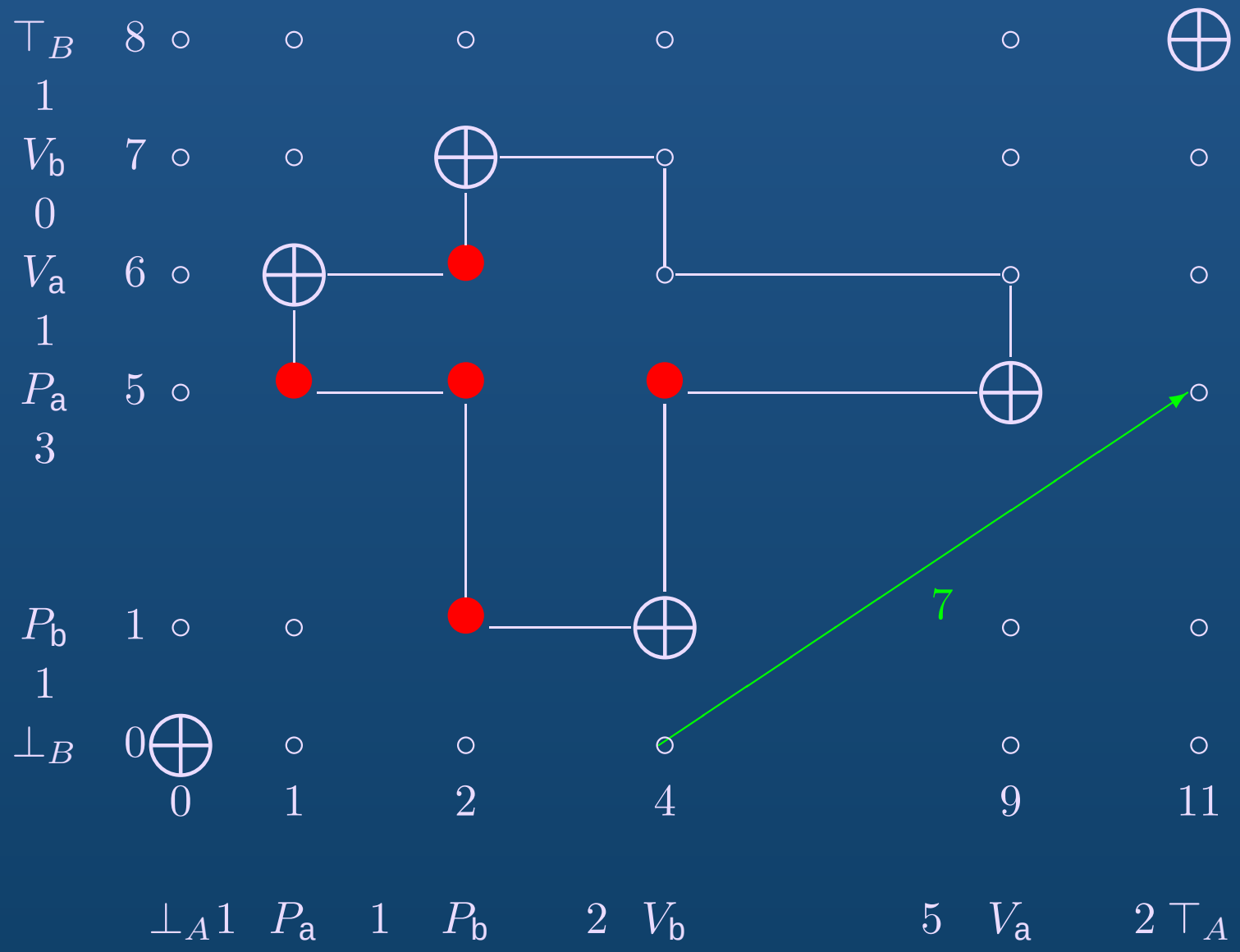
Illustration



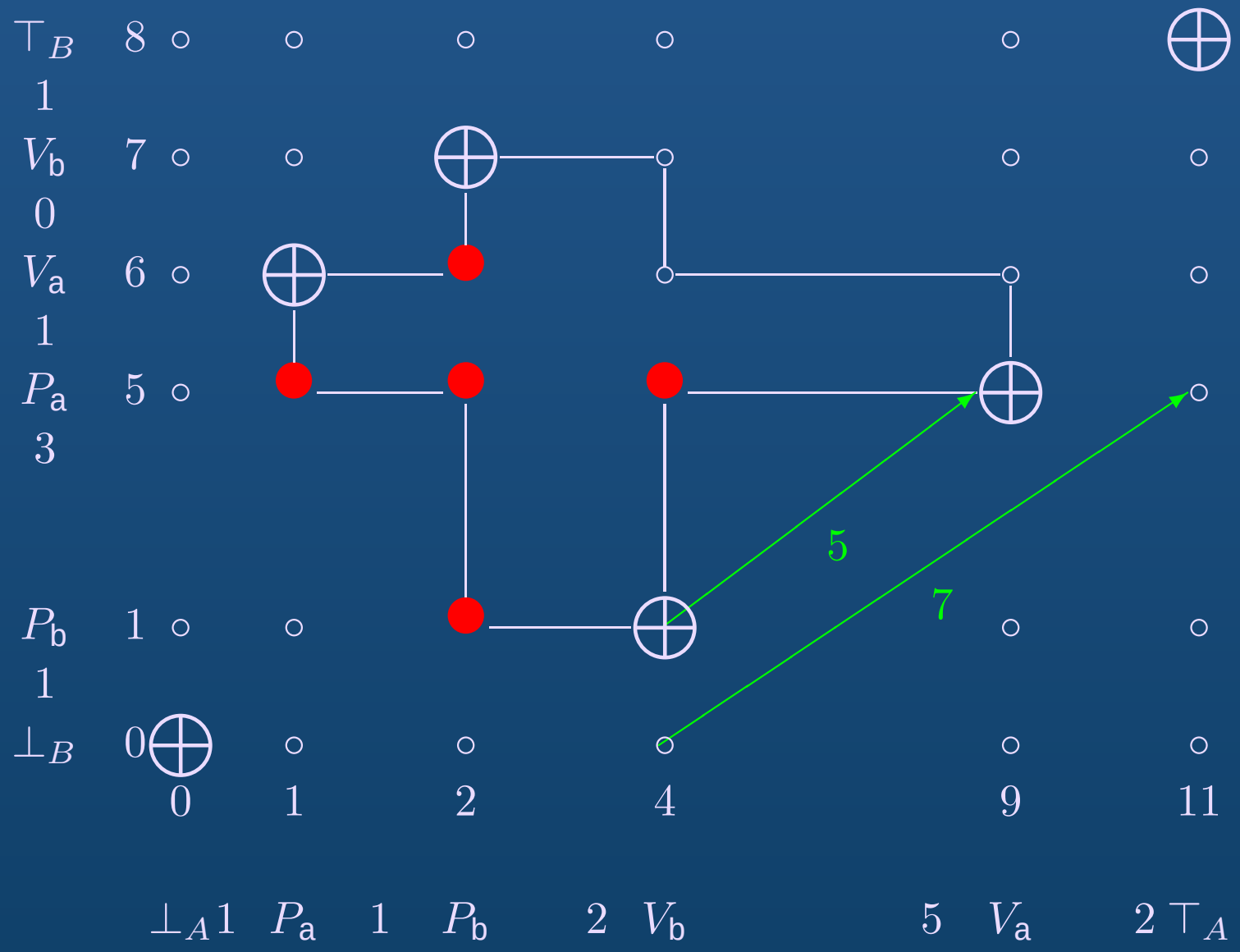
Illustration



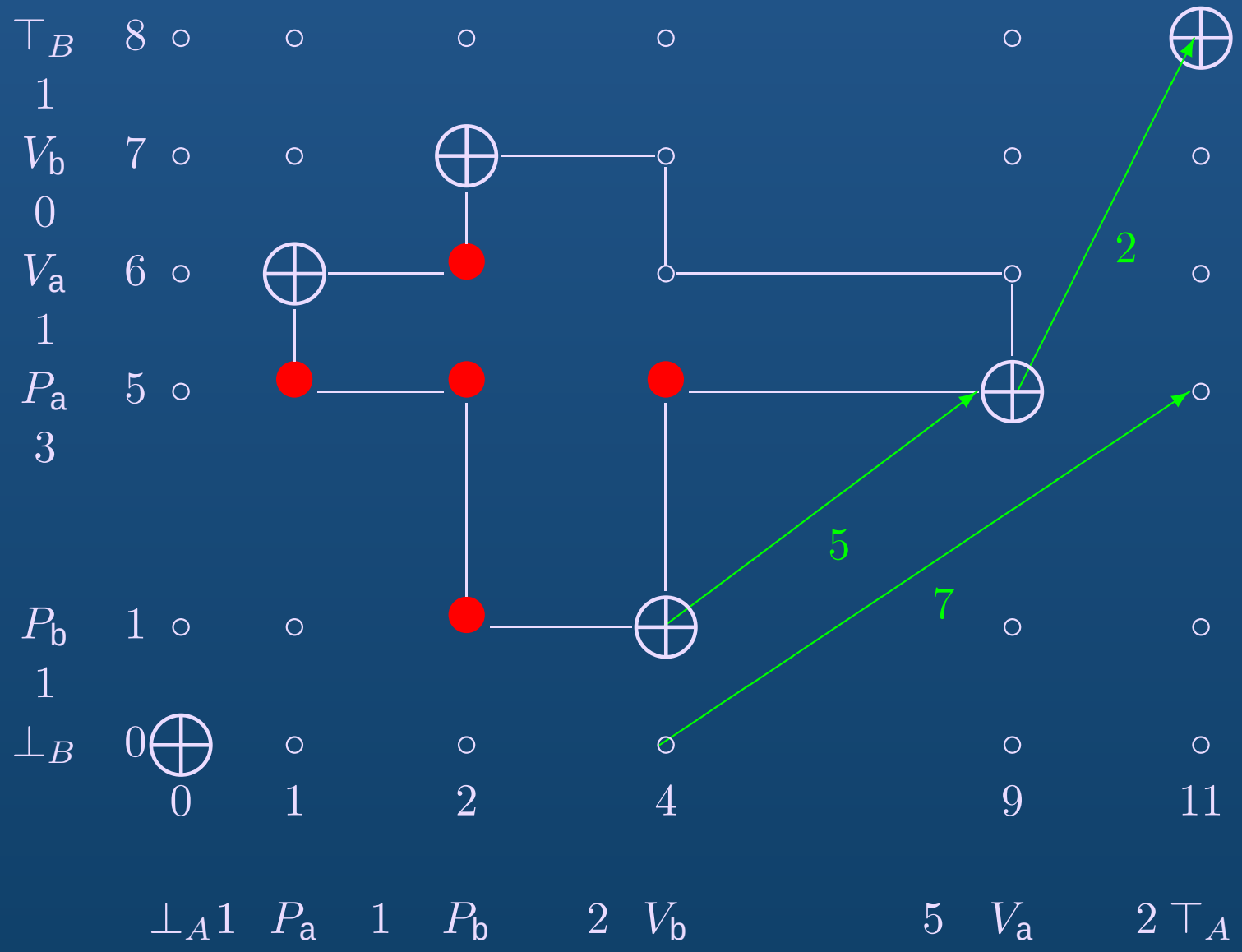
Illustration



Illustration



Illustration



Abstraction of Eager Schedules

Abstraction graph:

- nodes = *potential exchange point*
- edges = *bows*
- edge $\langle \epsilon, \epsilon' \rangle$ weighted with distance between ϵ and ϵ'

Property of the Abstraction Graph

Theorem

The duration of a quickest schedule
is the length of a shortest path from $\bot$ *to* $\top$
in the abstraction graph.

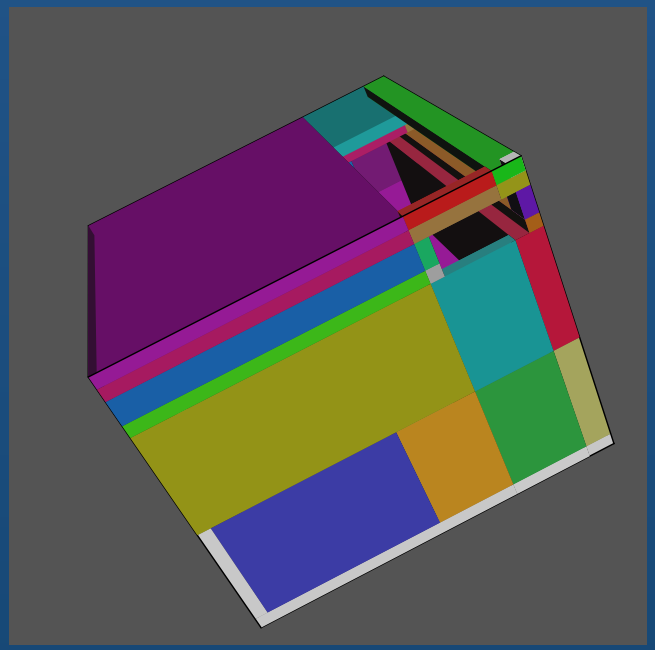
Computation of the length of a path: *history-free*

Computing an Efficient Schedule

1. Construct the abstraction graph:
   (a) Compute the potential exchange points: by computing the forbidden boxes (potential exchange points are vertices of the forbidden boxes)
   (b) Find the bows between the potential exchange points $\rightarrow$ *using a spacial decomposition into boxes*
2. Compute a shortest path from $\bot$ to $\top$
3. From the shortest path, construct a concrete (eager) schedule

$\rightarrow$ Deadlock-free efficient schedule, not necessarily optimal but with reasonable computation time

Decomposition into Allowed Boxes



- No forbidden element inside an allowed box $\Longrightarrow$ its diagonal corresponds to a bow
- Using *Orthogonal polyhedra [Bournez99]*
- Decomposition algorithm based on a greedy strategy

Computing Efficient Schedules using Geometric Intersection

Exact Scaling Geometrization

- The diagram is scaled according to the task durations (we do not yet consider tasks with duration 0)
- The product of threads $\mathcal{E}$ form a non-uniform grid over $\mathcal{B} = [0, c(\top_1)] \times \cdots \times [0, c(\top_N)]$ in $\mathbb{R}^N$
- A state $\epsilon \in \mathcal{E}$ is mapped to $\overline{\epsilon}$ which is a grid point
- A forbidden state is mapped to a forbidden box
- A string is mapped to a sequence of grid points
- To exploit continuous geometric properties, we define the geometrization of an arc $\langle \epsilon, \epsilon' \rangle$ as the directed line segment s from vertex $\overline{\epsilon}$ to vertex $\overline{\epsilon'}$

Bows and Geometric Intersection - Property II

Let P_F forbidden polyhedron

Theorem 1

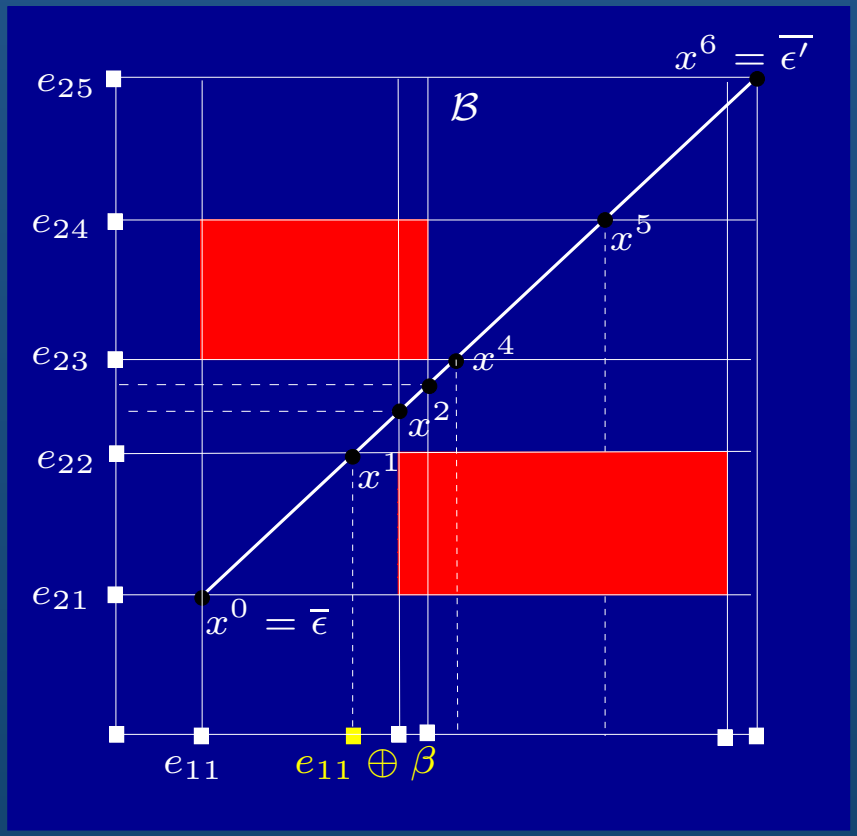
Given an arrow $\langle \epsilon, \epsilon' \rangle$,

if $\overline{\langle \epsilon, \epsilon' \rangle} \cap P_F = \emptyset$, then $\langle \epsilon, \epsilon' \rangle$ is a bow.

Idea of Proof

Construct a concrete feasible string based on $\overline{\langle \epsilon, \epsilon' \rangle}$ whose duration is the distance between ϵ and $\epsilon' \Rightarrow$ witness string

Bows and Geometric Intersection - Property I (contd)

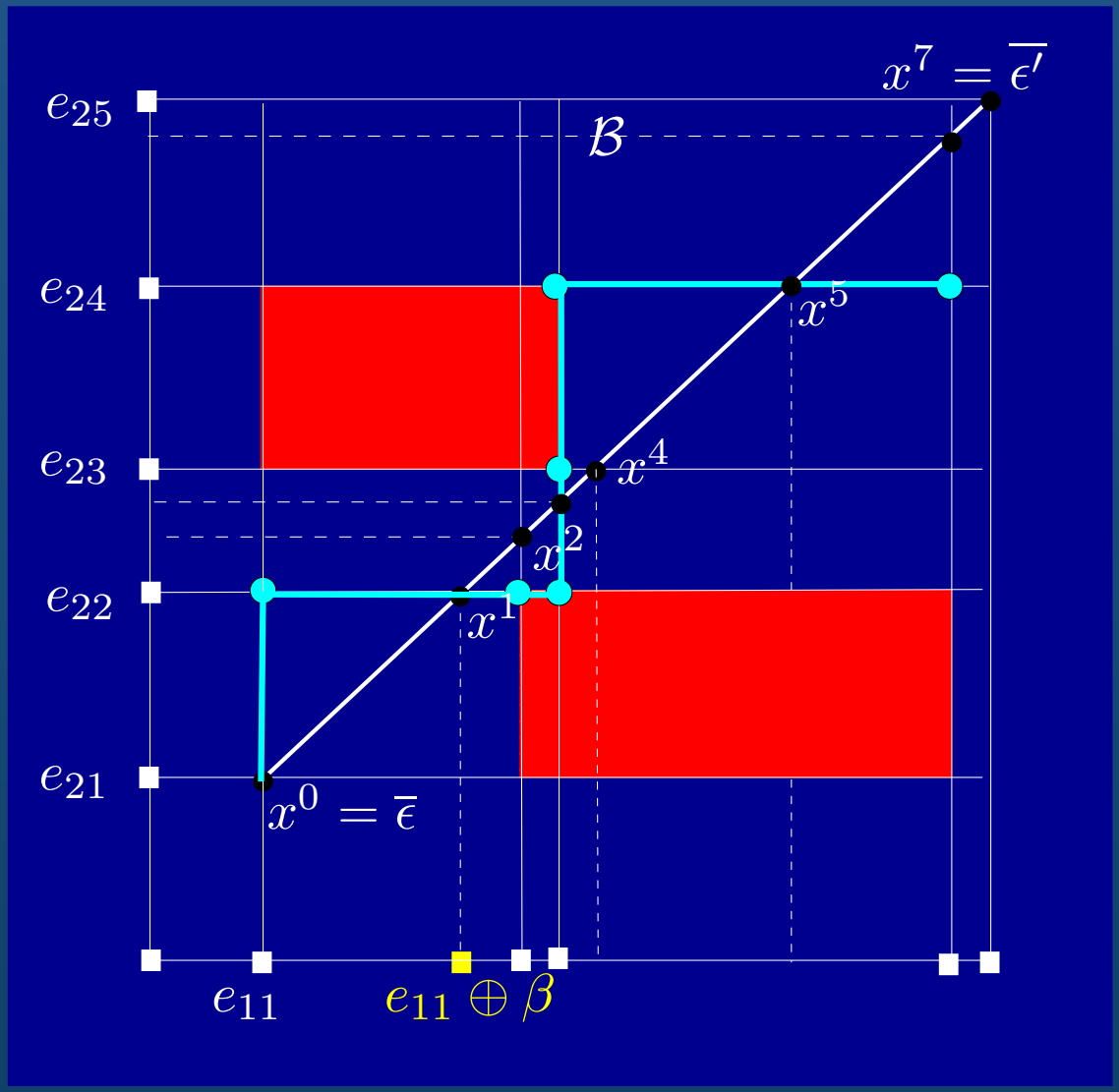


$\{x^j\}$: clipping of $\overline{\langle \epsilon, \epsilon' \rangle}$ with the grid. Any point in $\mathcal{B}$ is mapped to an extended event (e.g. $e \oplus \beta$). Mapping $\{x^j\}$ to a sequence of extended events $\Rightarrow$ concrete string s

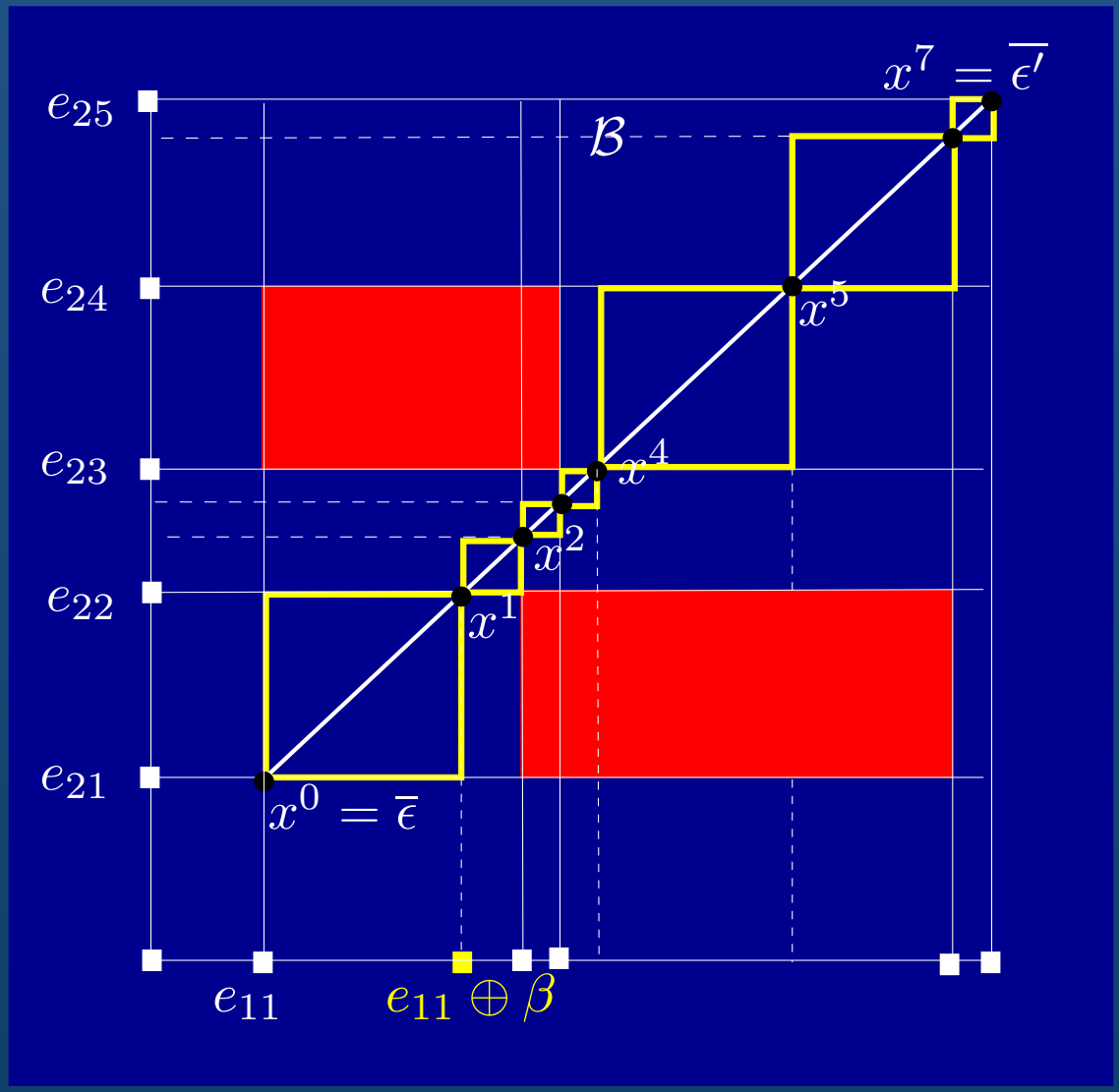
Proof of Theorem 1

- Prove that the concrete string s is feasible (defining $reset(s)$ and $reset(s)$ is feasible since all the boxes $box(x^j, x^{j+1})$ do not contain any forbidden points)
- Prove that duration of s = distance between ϵ and ϵ' (for all $box(x^j, x^{j+1})$ the axis with the longest side length is the same).

Proof of Resource Conflict Free



Proof of Duration



Bows and Geometric Intersection - Property II

Let P_F forbidden polyhedron

Theorem 2

Given an arrow $\langle \epsilon, \epsilon' \rangle$,

if it is a bow and $\overline{\langle \epsilon, \epsilon' \rangle} \cap P_F \neq \emptyset$, then there exists a sequence of bows $\langle \epsilon, \epsilon_1 \rangle, \langle \epsilon_1, \epsilon_2 \rangle, \ldots, \langle \epsilon_n, \epsilon' \rangle$ with empty intersection with P_F such that

$$d(\langle \epsilon, \epsilon' \rangle) = d(\langle \epsilon, \epsilon_1 \rangle) + d(\langle \epsilon_1, \epsilon_2 \rangle) + \ldots + d(\langle \epsilon_n, \epsilon' \rangle).$$

$\Rightarrow$ Combining the two theorems, it suffices to consider line segments with empty interesection with the forbidden boxes.

Algorithm for Computing Optimal Schedules

- Consider the set D of connected boxes that intersects with the diagonal of $\mathcal{B}$.
- Optimal schedule should be as closest as possible to the boundary of D
- Using ray tracing to determine bows.

Conclusion

Undergoing Work

- Experimentations
- Comparison with timed automata techniques
- Deadlines

Related works

- Éric Goubault

  Transitions take time, 1996

  $\rightarrow$ ◦ Continuous setting

  ◦ WCET tied to the geometry

  $\Longrightarrow$ WCET $= 0$ not possible

- Ulrich Fahrenberg

  The geometry of timed PV programs, 2003

  $\rightarrow$ One time dimension added for each clock

Dijkstra's approach

- Edger W. Dijkstra

  Co-operating sequential processes, 1968

  $\rightarrow$ **introducing the geometry**

  Untimed

- L. Fajstrup, E. Goubault and M. Raussen

  Detecting deadlocks in concurrent systems, 1998

  $\rightarrow$ **exploiting the geometry**

  Untimed