

Quantitative Program Analysis

Chris Hankin

joint work with

Alessandra Di Pierro and Herbert Wiklicky

Appsem II Workshop, September 2005

Overview

- Quantitative Program Analysis

Overview

- Quantitative Program Analysis
- Security setting

Overview

- Quantitative Program Analysis
- Security setting
- Formal definitions

Overview

- Quantitative Program Analysis
- Security setting
- Formal definitions
- Probabilistic padding

Overview

- Quantitative Program Analysis
- Security setting
- Formal definitions
- Probabilistic padding
- Algorithm

Overview

- Quantitative Program Analysis
- Security setting
- Formal definitions
- Probabilistic padding
- Algorithm
- Correctness

Quantitative Program Analysis

Classical Abstract Interpretation

To set the scene, imagine some programming language. Its **semantics** identifies some set V of values (like states, closures, double precision reals) and specifies how a program p transforms one value v_1 to another v_2 ; we may write

$$p \vdash v_1 \rightsquigarrow v_2$$

In a similar way, a **program analysis** identifies the set L of properties (like shapes of states, abstract closures, lower and upper bounds for reals) and specifies how a program p transforms one property l_1 to another l_2 ; we may write

$$p \vdash l_1 \triangleright l_2$$

Unlike the semantics, it is customary to require \triangleright to be **deterministic** and thereby define a function; this will allow us to write $f_p(l_1) = l_2$ to mean $p \vdash l_1 \triangleright l_2$.

Correctness relations

Every program analysis should be correct with respect to the semantics. For a class of (so-called first-order) program analyses this is established by directly relating properties to values using a **correctness relation**:

$$R : V \times L \rightarrow \{true, false\}$$

The intention is that $v R l$ formalises our claim that the value v is described by the property l .

To be useful one has to prove that the correctness relation R is preserved under computation: if the relation holds between the initial value and the initial property then it also holds between the final value and the final property. This may be formulated as the implication

$$v_1 R l_1 \wedge p \vdash v_1 \rightsquigarrow v_2 \wedge p \vdash l_1 \triangleright l_2 \Rightarrow v_2 R l_2$$

The theory of Abstract Interpretation comes to life when we augment the set of properties L with a preorder structure and relate this to the correctness relation R . The most common scenario is when $L = (L, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ is a complete lattice with partial ordering \sqsubseteq . We then impose the following relationship between R and L :

$$v R l_1 \wedge l_1 \sqsubseteq l_2 \Rightarrow v R l_2$$

$$(\forall l \in L' \subseteq L : v R l) \Rightarrow v R (\bigsqcup L')$$

Alternatives to this formulation involve

- **Representation functions:**

$$\beta : V \rightarrow L$$

The idea is that β maps a value to the **best** property describing it.

- **Galois connections:** We define (V, α, γ, L) is a Galois connection between the complete lattices (V, \sqsubseteq) and (L, \sqsubseteq) if and only if $\alpha : V \rightarrow L$ and $\gamma : L \rightarrow V$ are monotone functions that satisfy:

$$\gamma \circ \alpha \sqsupseteq \lambda v.v \alpha \circ \gamma \sqsubseteq \lambda l.l$$

Having defined a suitable “set” of properties we then:

- define suitable interpretations of program operations
($\alpha \circ F \circ \gamma$)
- construct efficient implementations (widenings).

We now shift our attention to a setting where the semantic relation carries some quantitative information:

$$p \vdash v_1 \rightsquigarrow_q v_2$$

We could stay with the classical framework of abstract interpretation (cf Monniaux) but we choose a different approach ...

Matrix Representation

For quantitative relations $R \subseteq X \times \mathbb{W} \times X$ define:

$$(\mathbf{M}_R)_{xy} = \begin{cases} w & \text{iff } (x, w, y) \in R \\ 0 & \text{otherwise} \end{cases}$$

Matrix Representation

For quantitative relations $R \subseteq X \times \mathbb{W} \times X$ define:

$$(\mathbf{M}_R)_{xy} = \begin{cases} \sum w & \text{iff } (x, w, y) \in R \\ 0 & \text{otherwise} \end{cases}$$

Matrix Representation

For quantitative relations $R \subseteq X \times \mathbb{W} \times X$ define:

$$(\mathbf{M}_R)_{xy} = \begin{cases} w & \text{iff } (x, w, y) \in R \\ 0 & \text{otherwise} \end{cases}$$

For labelled relations $L \subseteq X \times A \times \mathbb{W} \times X$ define:

$$L|_a = \{(x, w, y) \mid (x, a, w, y) \in L\}$$

$$\mathbf{M}_L = \bigoplus_{a \in A} \mathbf{M}_{L|_a}$$

Interpreting relations as linear operators allows us to measure them. The standard way to measure the “size” of a linear operator is via an **operator norm** which in turn may have its origins in a vector *norm*:

1. $\|\vec{x}\| \geq 0$

2. $\|\vec{x}\| = 0 \leftrightarrow \vec{x} = \vec{o}$

3. $\|\alpha\vec{x}\| = |\alpha|\|\vec{x}\|$

4. $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$

For example: **1**-norm (sum of absolute values), **euclidean** norm (sqrt of the sum of squares of absolute values), **supremum** norm (supremum of absolute values)

Probabilistic Abstract Interpretation

Given two probabilistic domains, \mathcal{C} and \mathcal{D} , a **probabilistic abstract interpretation** is a pair of linear maps, $A : \mathcal{C} \mapsto \mathcal{D}$ and $G : \mathcal{D} \mapsto \mathcal{C}$, between the concrete domain \mathcal{C} and the abstract domain \mathcal{D} , such that G is the **Moore-Penrose pseudo-inverse** of A , and vice versa.

Let \mathcal{C} and \mathcal{D} be two Hilbert spaces and $A : \mathcal{C} \mapsto \mathcal{D}$ a bounded linear map between them. A bounded linear map $A^\dagger = G : \mathcal{D} \mapsto \mathcal{C}$ is the Moore-Penrose pseudo-inverse of A iff

$$A \circ G = P_A \quad \text{and} \quad G \circ A = P_G$$

where P_A and P_G denote orthogonal projections onto the ranges of A and G .

Alternatively, if a is Moore-Penrose invertible, its Moore-Penrose pseudoinverse, a^\dagger satisfies the following:

(i) $aa^\dagger a = a,$

(ii) $a^\dagger aa^\dagger = a^\dagger,$

(iii) $(aa^\dagger)^* = aa^\dagger,$

(iv) $(a^\dagger a)^* = a^\dagger a.$

It is instructive to compare the equations on this and the last slide with the classical setting. For example, if (α, γ) is a **Galois insertion**:

$$\alpha \circ \gamma \circ \alpha = \alpha \text{ and } \gamma \circ \alpha \circ \gamma = \gamma$$

A simple method to construct a probabilistic abstract interpretation is as follows:

Given a linear operator Φ on some vector space \mathcal{V} expressing the probabilistic semantics of a concrete system, and a linear abstraction function $\mathbf{A} : \mathcal{V} \mapsto \mathcal{W}$ from the concrete domain into an abstract domain \mathcal{W} , we compute the (unique) Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of \mathbf{A} . The abstract semantics can then be defined as the linear operator on the abstract domain \mathcal{W} :

$$\Psi = \mathbf{A} \circ \Phi \circ \mathbf{G}.$$

Security and Timing Attacks

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Various insecure information flows might arise in a program.

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Various insecure information flows might arise in a program.

Direct leakage:

$$l := h$$

The value of the high confidential variable h is directly transmitted to the variable l of low confidentiality.

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Various insecure information flows might arise in a program.

Indirect leakage:

if $(h = 1)$ **then** $l := 1$ **else** $(l := 0)$

Secret information can be deduced indirectly from the value of the low variable l .

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Various insecure information flows might arise in a program.

Probabilistic leakage:

$$h := h \bmod 2; (l := h \stackrel{\frac{1}{2}}{\square} \stackrel{\frac{1}{2}}{\square} (l := 0 \stackrel{\frac{1}{2}}{\square} \stackrel{\frac{1}{2}}{\square} l := 1))$$

The final value of l will reveal information about h with probability $\frac{3}{4}$.

Confidentiality

Confidentiality is the property of a system where all information flows are secure.

Various insecure information flows might arise in a program.

Timing leakage:

if $(h = 0)$ **then** (**tick**; $l := 0$) **else** $(l := 0)$

Information about h can be obtained by measuring the running time (steps) of the program execution.

Models of Confidentiality

Most confidentiality properties of programs are specified according to the **non-interference** model:

Models of Confidentiality

Most confidentiality properties of programs are specified according to the **non-interference** model:

Varying the high security inputs to a program has no effect on the **observable behaviour** of the program.

Models of Confidentiality

Most confidentiality properties of programs are specified according to the **non-interference** model:

Varying the high security inputs to a program has no effect on the **observable behaviour** of the program.

Non-interference can be naturally expressed by *semantic models* of program execution.

Models of Confidentiality

Most confidentiality properties of programs are specified according to the **non-interference** model:

Varying the high security inputs to a program has no effect on the **observable behaviour** of the program.

Non-interference can be naturally expressed by *semantic models* of program execution.

The semantics of a process P depends on “high variable” h ; depending on the value of h we have different variants $P(h)$.

The value of h is kept confidential, iff all $P(h)$'s are **behaviourally equivalent**.

Timing Attacks

Timing Attacks are based on measuring the time it takes for a unit to perform operations.

Timing Attacks

Timing Attacks are based on measuring the time it takes for a unit to perform operations.

Paul Kocher showed how encryption keys might be learned via timing attacks.

Timing Attacks

Timing Attacks are based on measuring the time it takes for a unit to perform operations.

Paul Kocher showed how encryption keys might be learned via timing attacks.

```
s := 1;
for (i=0; i<w; i++) {
  if ([k(i)])
    C := (s · M) mod n;
  else
    C := s;
  s := C · C;
}
```

padding

Dealing with time-based interference

Semantics-based analysis of information flow:

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems

Enforcing secure information flow

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis
Checking (Quantifying the level of) security

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Beyond checking: Program transformation

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Beyond checking: Program transformation

- Padding

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Beyond checking: Program transformation

- Padding

Adding delays: The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time.

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Beyond checking: Program transformation

- Padding
- Probabilistic Padding

Dealing with time-based interference

Semantics-based analysis of information flow:

- Security Type Systems
- (Quantitative) Program Analysis

Beyond checking: Program transformation

- Padding
- Probabilistic Padding

Fixed time implementations are likely to be slow. Adding **random delays** can make the attack a bit more difficult, although still possible.

Padding

Agat's algorithm would transform the program for modular exponentiation into the following padded version, which is now secure:

Padding

Agat's algorithm would transform the program for modular exponentiation into the following padded version, which is now secure:

```
s := 1;
for (i=0; i<w; i++) {
  if ([k(i)])
    C := (s · M) mod n;
    [C := s];
  else
    [C := (s · M) mod n];
    C := s;
s := C · C;
}
```

Padding: Problem

Agat's transformation not only eliminates timing leaks from insecure programs but also modifies programs that are already secure.

Padding: Problem

Agat's transformation not only eliminates timing leaks from insecure programs but also modifies programs that are already secure.

Example

```
if ( $h < 0$ ) then  $h := h + 1$  else  $h := h - 1$ ;
```

Padding: Problem

Agat's transformation not only eliminates timing leaks from insecure programs but also modifies programs that are already secure.

Example

if ($h < 0$) **then** $h := h + 1$ **else** $h := h - 1$;

would be translated into a “safe” version of double execution time:

if ($h < 0$) **then** ($h := h + 1$; **skip**) **else** (**skip**; $h := h - 1$);

Padding: Problem

Agat's transformation not only eliminates timing leaks from insecure programs but also modifies programs that are already secure.

Example

```
if ( $h < 0$ ) then  $h := h + 1$  else  $h := h - 1$ ;
```

or even of four times longer, if the typing rule is applied again (transformation is not idempotent):

```
if ( $h < 0$ ) then ( $h := h + 1$ ; skip; skip; skip)  
else (skip; skip; skip;  $h := h - 1$ );
```

Probabilistic Padding

- Padding **probabilistic** programs

Probabilistic Padding

- Padding **probabilistic** programs
- Performing the transformation **probabilistically**, i.e. only with a certain probability.

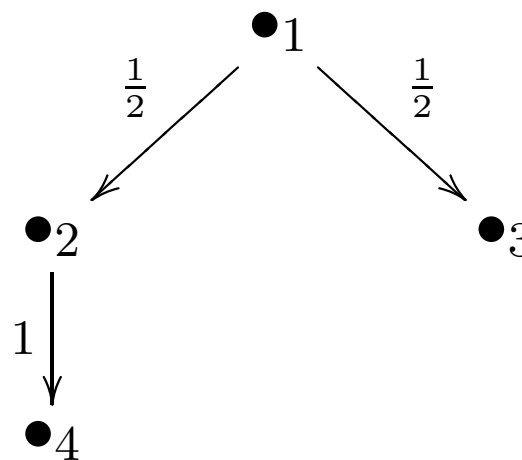
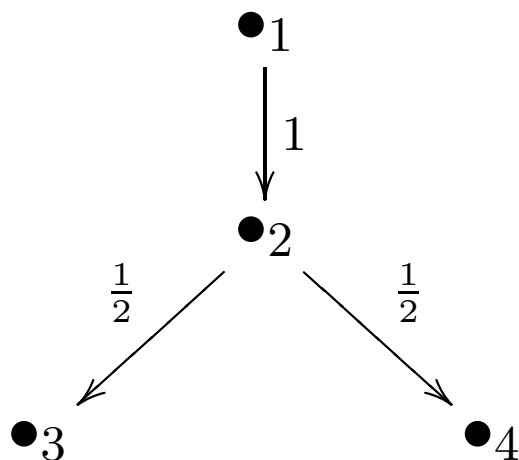
Probabilistic Padding

- Padding **probabilistic** programs
- Performing the transformation **probabilistically**, i.e. only with a certain probability.

```
s := 1;
for (i=0; i<w; i++) {
  if ([k(i)])
    { C := (s · M) mod n } □ pp
    { [C := s]; C := (s · M) mod n; }
  else
    { C := s; } □ pp
    { [C := (s · M) mod n]; C := s; }
  s := C · C;
}
```

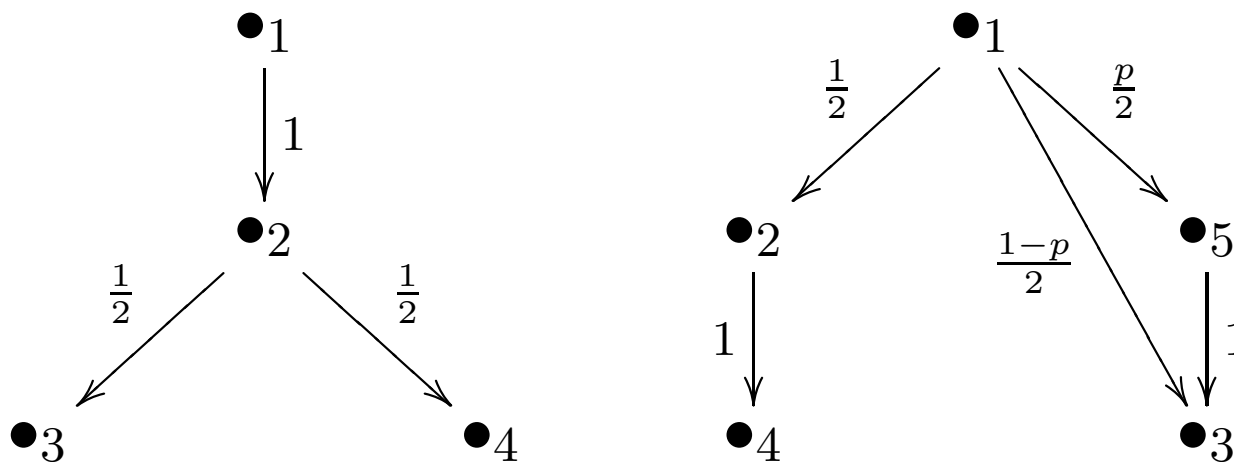
Probabilistic Padding: General setting

Consider the following abstract descriptions of probabilistic programs



Probabilistic Padding: General setting

A probabilistic padding of B would lead to:



Process Algebra

Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

A **probabilistic transition system** (PTS) is a tuple $(S, A, \longrightarrow, \pi_0)$, with:

Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

A **probabilistic transition system** (PTS) is a tuple $(S, A, \longrightarrow, \pi_0)$, with:

- S is a non-empty, finite set of **states**,

Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

A **probabilistic transition system** (PTS) is a tuple $(S, A, \longrightarrow, \pi_0)$, with:

- S is a non-empty, finite set of **states**,
- A is a non-empty, finite set of **actions**,

Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

A **probabilistic transition system** (PTS) is a tuple $(S, A, \longrightarrow, \pi_0)$, with:

- S is a non-empty, finite set of **states**,
- A is a non-empty, finite set of **actions**,
- $\longrightarrow \subseteq S \times \text{Dist}(A \times S)$ is a **transition relation**,

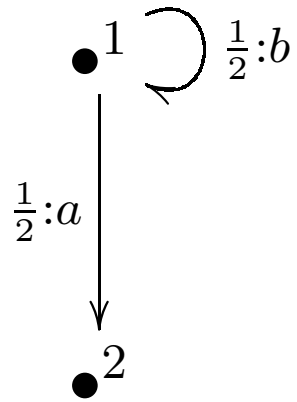
Probabilistic Processes

We adopt Probabilistic Transition System as a basic model.

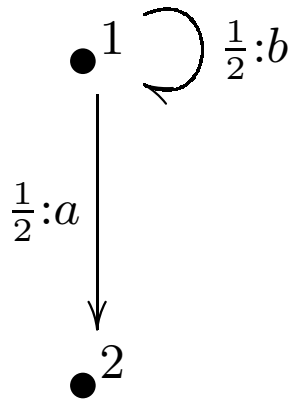
A **probabilistic transition system** (PTS) is a tuple $(S, A, \longrightarrow, \pi_0)$, with:

- S is a non-empty, finite set of **states**,
- A is a non-empty, finite set of **actions**,
- $\longrightarrow \subseteq S \times \text{Dist}(A \times S)$ is a **transition relation**,
- $\pi_0 \in \text{Dist}(S)$ is an **initial distribution** on S .

Example

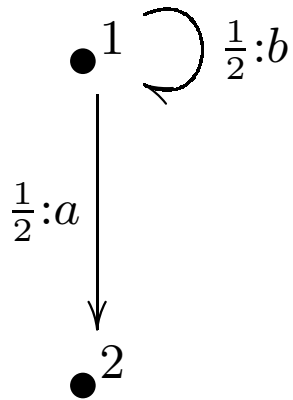


Example



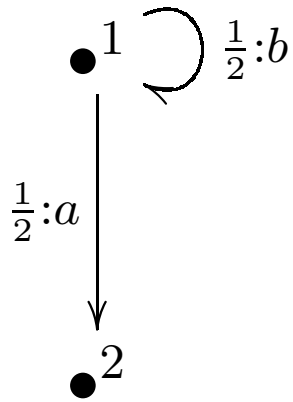
$$\mathbf{M}(A) = \mathbf{M}_a(A) \oplus \mathbf{M}_b(A)$$

Example



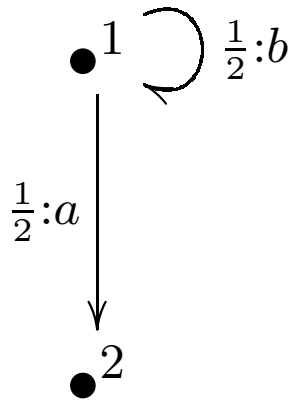
$$\mathbf{M}(A) = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix}$$

Example



$$\mathbf{M}(A) = \left(\begin{array}{c} \left(\begin{array}{cc} 0 & \frac{1}{2} \\ 0 & 0 \end{array} \right) \\ \left(\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array} \right) \end{array} \right) \left(\begin{array}{c} \left(\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array} \right) \\ \left(\begin{array}{cc} \frac{1}{2} & 0 \\ 0 & 0 \end{array} \right) \end{array} \right)$$

Example



$$\mathbf{M}(A) = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

PTS and Markov Chains

We consider finite state acyclic programs represented via *rooted* PTS's.

PTS and Markov Chains

A **rooted** PTS is a PTS $(S, A, \longrightarrow, \pi_0)$, with $\pi_0 = \{\langle s_0, 1 \rangle\}$ (i.e. with a unique initial state $s_0 \in S$).

PTS and Markov Chains

A **rooted** PTS is a PTS $(S, A, \longrightarrow, \pi_0)$, with $\pi_0 = \{\langle s_0, 1 \rangle\}$ (i.e. with a unique initial state $s_0 \in S$).

A rooted PTS is a discrete finite Markov chain labelled with actions.

PTS and Markov Chains

A **rooted** PTS is a PTS $(S, A, \longrightarrow, \pi_0)$, with $\pi_0 = \{\langle s_0, 1 \rangle\}$ (i.e. with a unique initial state $s_0 \in S$).

A rooted PTS is a discrete finite Markov chain labelled with actions.

Theorem[Kemeny & Snell 1960]

A finite Markov chain is **lumpable** with respect to a partition $\{C_1, C_2, \dots, C_m\}$ iff for every pair C_i and C_j , the probability $p_{kC_j} = \sum_{t \in C_j} p_{it}$ of moving in one step from state s_k into set C_j have the same value for every $s_k \in C_i$.

Bisimulation and Lumpability

Given two probabilistic processes A and B , then $A \sim_b B$ iff there exist classification matrices \mathbf{K}_A and \mathbf{K}_B such that:

$$\mathbf{K}_A^\dagger \mathbf{M}(A) \mathbf{K}_A = \mathbf{K}_B^\dagger \mathbf{M}(B) \mathbf{K}_B,$$

Bisimulation and Lumpability

Given two probabilistic processes A and B , then $A \sim_b B$ iff there exist classification matrices \mathbf{K}_A and \mathbf{K}_B such that:

$$\mathbf{K}_A^\dagger \mathbf{M}(A) \mathbf{K}_A = \mathbf{K}_B^\dagger \mathbf{M}(B) \mathbf{K}_B,$$

$A \sim_b B$, iff there exists a **lumping** \mathbf{K} of the process $\mathbf{M}(A) \oplus \mathbf{M}(B)$ of the form

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_A \\ \mathbf{K}_B \end{pmatrix}$$

with \mathbf{K}_A and \mathbf{K}_B fully column-ranked classification operators for A and B .

Bisimulation and Non-interference

Two bisimilar processes are indistinguishable to an external observer and therefore **confined** (no information leakage).

Bisimulation and Non-interference

Two bisimilar processes are indistinguishable to an external observer and therefore **confined** (no information leakage).

Consider an attacker who can observe the timing behaviour of the processes and the final results of their executions, aka **I/O observables**.

Bisimulation and Non-interference

Two bisimilar processes are indistinguishable to an external observer and therefore **confined** (no information leakage).

Consider an attacker who can observe the timing behaviour of the processes and the final results of their executions, aka **I/O observables**.

We describe the timing behaviour of a system abstractly via “ticked” PTS’s. A **ticked** PTS is a labelled transition system with set of labels $A = \{\checkmark\}$, where \checkmark = time of a transition.

Bisimulation and Non-interference

Two bisimilar processes are indistinguishable to an external observer and therefore **confined** (no information leakage).

Consider an attacker who can observe the timing behaviour of the processes and the final results of their executions, aka **I/O observables**.

The ticked version of a PTS A is the ticked PTS \hat{A} obtained by replacing all its labels by \checkmark .

Bisimulation and Non-interference

Two bisimilar processes are indistinguishable to an external observer and therefore **confined** (no information leakage).

Consider an attacker who can observe the timing behaviour of the processes and the final results of their executions, aka **I/O observables**.

The ticked version of a PTS A is the ticked PTS \hat{A} obtained by replacing all its labels by \checkmark .

If A and B are such that $\hat{A} \sim_b \hat{B}$, then they are **confined against timing attacks**.

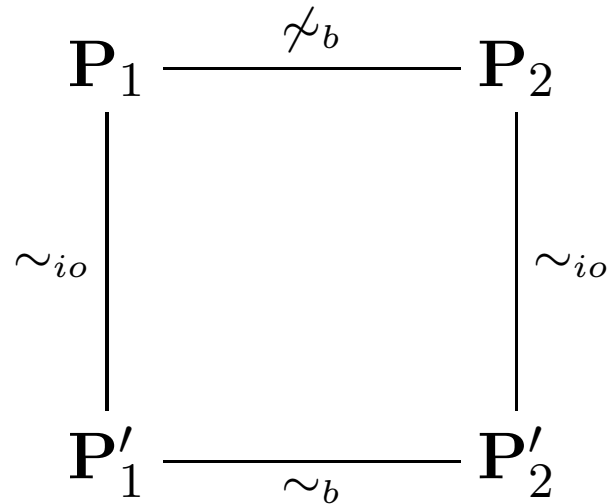
Transformation

A Transformation Algorithm

The algorithm transforms the input system into an I/O-equivalent one which is free from timing leaks.

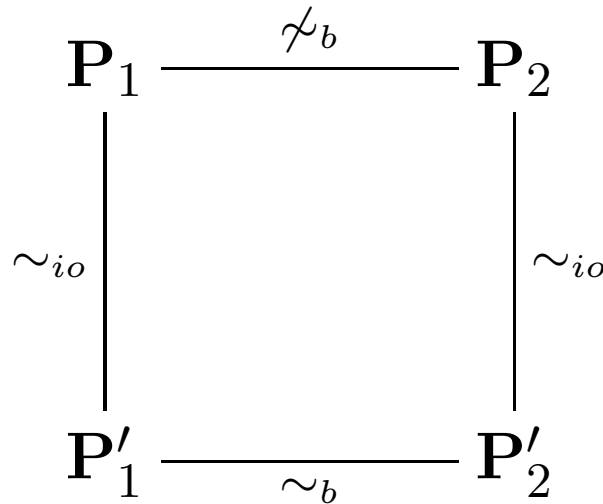
A Transformation Algorithm

The algorithm transforms the input system into an I/O-equivalent one which is free from timing leaks.



A Transformation Algorithm

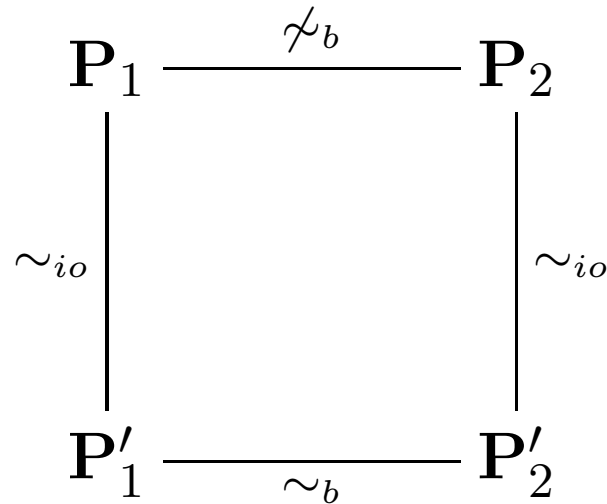
The algorithm transforms the input system into an I/O-equivalent one which is free from timing leaks.



It uses ideas from S. Derisavi, H. Hermanns, W. H. Sanders, *Optimal state-space lumping in Markov chains*, Inf. Proc. Let.87 (6) (2003).

A Transformation Algorithm

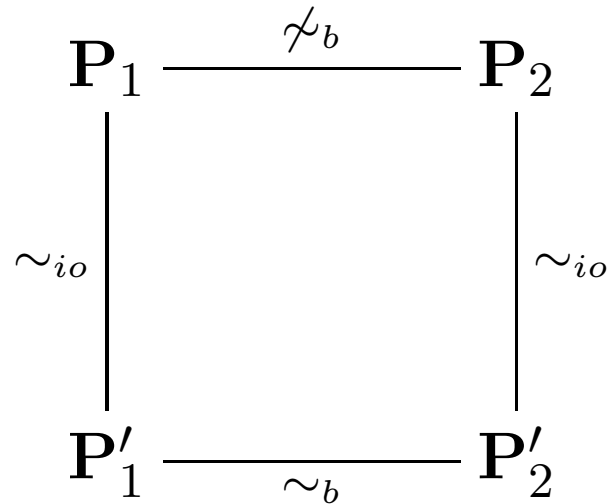
The algorithm transforms the input system into an I/O-equivalent one which is free from timing leaks.



R. Paige, R. Tarjan, *Three partition refinement algorithms*, SIAM Journal of Computation 16 (6) (1987).

A Transformation Algorithm

The algorithm transforms the input system into an I/O-equivalent one which is free from timing leaks.



and the padding techniques from
Johan Agat, *Transforming out timing leaks*, Proc. POPL
2000.

Basic Definitions I

A partition $P = \{B_1, \dots, B_n\}$ of a set of states is **stable** with respect to \xrightarrow{p} iff $pre_q(B_i) \cap B_j = \emptyset$ or $B_j \subseteq pre_q(B_i)$.

Basic Definitions I

A partition $P = \{B_1, \dots, B_n\}$ of a set of states is **stable** with respect to \xrightarrow{p} iff $pre_q(B_i) \cap B_j = \emptyset$ or $B_j \subseteq pre_q(B_i)$.

SPLITTING (B_i, P) refines P by replacing each $B_j \in P$ s.t. $pre_p(B_i) \cap B_j \neq \emptyset$ by $pre_p(B_i) \cap B_j$ and $B_j \setminus pre_p(B_i)$.

The block B_i is called a **splitter**.

Basic Definitions I

A partition $P = \{B_1, \dots, B_n\}$ of a set of states is **stable** with respect to \xrightarrow{p} iff $pre_q(B_i) \cap B_j = \emptyset$ or $B_j \subseteq pre_q(B_i)$.

SPLITTING (B_i, P) refines P by replacing each $B_j \in P$ s.t. $pre_p(B_i) \cap B_j \neq \emptyset$ by $pre_p(B_i) \cap B_j$ and $B_j \setminus pre_p(B_i)$.

The block B_i is called a **splitter**.

The **backward image** of B wrt q is

$$pre_{q, \longrightarrow}(B) = \{s \mid \exists B' \subseteq B : p(s, B') = q\},$$

Basic Definitions I

A partition $P = \{B_1, \dots, B_n\}$ of a set of states is **stable** with respect to \xrightarrow{p} iff $pre_q(B_i) \cap B_j = \emptyset$ or $B_j \subseteq pre_q(B_i)$.

SPLITTING (B_i, P) refines P by replacing each $B_j \in P$ s.t. $pre_p(B_i) \cap B_j \neq \emptyset$ by $pre_p(B_i) \cap B_j$ and $B_j \setminus pre_p(B_i)$.

The block B_i is called a **splitter**.

The **backward image** of B wrt q is

$$pre_{q, \longrightarrow}(B) = \{s \mid \exists B' \subseteq B : p(s, B') = q\},$$

where $p(s, B) = \sum \{q \mid s \xrightarrow{q} t \text{ with } t \in B\}$.

Basic Definitions II

The **height** of a rooted PTS T is the maximal length of a path from the root to a terminal state.

Basic Definitions II

The **height** of a rooted PTS T is the maximal length of a path from the root to a terminal state.

The **n layer cut-off** of a DAG is defined inductively as

$n = 0$: the set of terminal nodes

$n = i + 1$: the set of nodes with only direct links (paths of length one) to nodes in the i layer cut-off.

Basic Definitions II

The **height** of a rooted PTS T is the maximal length of a path from the root to a terminal state.

The **n layer cut-off** of a DAG is defined inductively as

$n = 0$: the set of terminal nodes

$n = i + 1$: the set of nodes with only direct links (paths of length one) to nodes in the i layer cut-off.

The complement of the n layer cut-off is the **n layer top**.

Procedure Lumping

```
procedure LUMPING( $T_1, T_2$ )  
   $n \leftarrow 0$   
   $P, S \leftarrow \{S_1 \cup S_2\}$   
  while  $n \leq \text{HEIGHT}(T_1 \oplus T_2)$  do  
     $TT \leftarrow \text{CUTOFF}(T_1 \oplus T_2, n)$   
     $P \leftarrow \{B \cap TT \mid B \in P\}$   
     $S \leftarrow P \cup \{(S_1 \cup S_2) \setminus TT\}$   
    while  $S \neq \emptyset$  do  
      choose  $B \in S, S \leftarrow S \setminus B$   
       $P \leftarrow \text{SPLITTING}(B, P)$   
    end while  
     $n \leftarrow n + 1$   
  end while  
end procedure
```

Incremental Lumping

```
procedure MAXLUMPING( $T_1, T_2$ )  
   $n \leftarrow 0$   
   $P, S \leftarrow \{S_1 \cup S_2\}$   
  while  $n \leq \text{HEIGHT}(T_1 \oplus T_2)$  do  
     $TT \leftarrow \text{CUTOFF}(T_1 \oplus T_2, n)$   
     $P \leftarrow \{B \cap TT \mid B \in P\}$   
     $S \leftarrow P \cup \{(S_1 \cup S_2) \setminus TT\}$   
    while  $S \neq \emptyset$  do  
      choose  $B \in S, S \leftarrow S \setminus B$   
       $P \leftarrow \text{SPLITTING}(B, P)$   
       $C \leftarrow \{B \in P \mid B \cap S_1 = \emptyset \vee B \cap S_2 = \emptyset\}$   
      if  $C \neq \emptyset$  then return  $C$   
    end while  
     $n \leftarrow n + 1$   
  end while  
end procedure
```

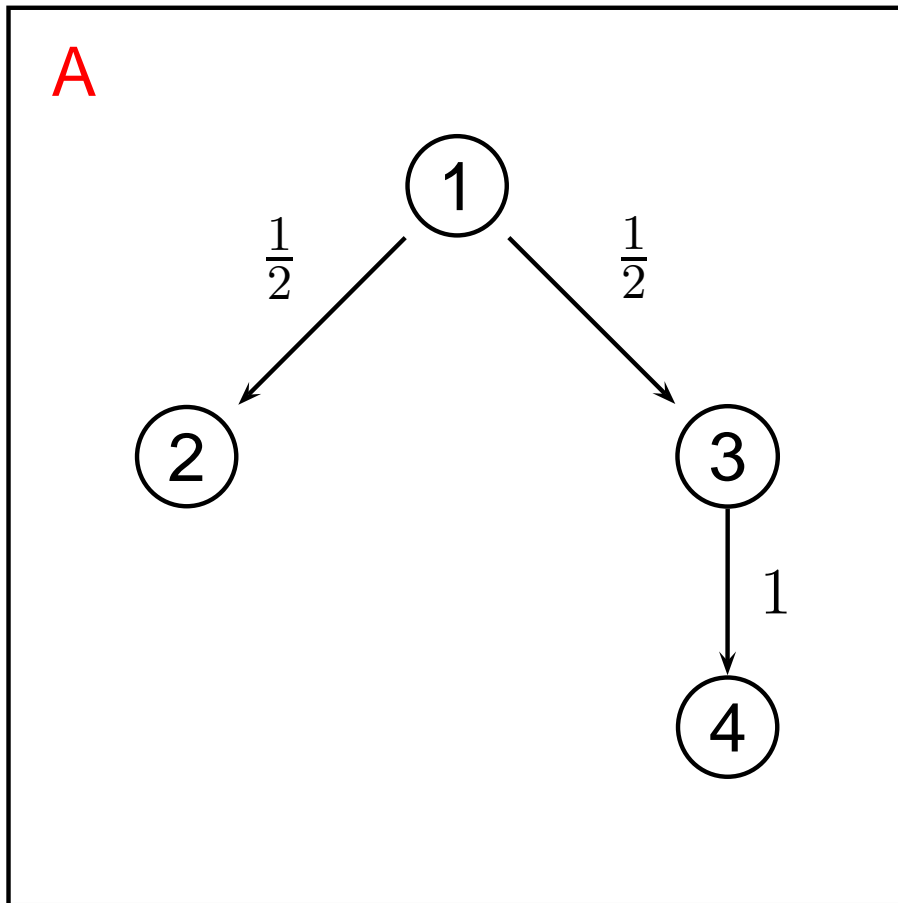
Procedure Padding

```
procedure PADDING( $T_1, T_2$ )  
   $C \leftarrow$  MAXLUMPING( $T_1, T_2$ )  
  repeat  
    if  $C = \{C_i\}_i$  with  $C_i \cap S_1 = \emptyset, \forall C_i$  then  
       $D \leftarrow (\{d\}, \{\sqrt{\phantom{x}}\}, \emptyset, \{\langle d, 1 \rangle\})$   
       $T_1 \leftarrow$  LINKING( $D, d, \sqrt{\phantom{x}}, 1 \cdot pp, T_1$ )  
    else if  $C = \{C_i\}_i$  with  $\forall C_i : C_i \cap S_2 = \emptyset$  then  
       $D \leftarrow (\{d\}, \{\sqrt{\phantom{x}}\}, \emptyset, \{\langle d, 1 \rangle\})$   
       $T_2 \leftarrow$  LINKING( $D, d, \sqrt{\phantom{x}}, 1 \cdot pp, T_2$ )  
    else if  $C = \{C_i\}_i$  with  $C_j \subseteq S_1 \neq \emptyset \neq C_k \subseteq S_2$  then  
      .....  
    end if  
     $C \leftarrow$  MAXLUMPING( $T_1, T_2$ )  
  until  $C = \emptyset$   
  return  $T_1, T_2$   
end procedure
```

Procedure Padding

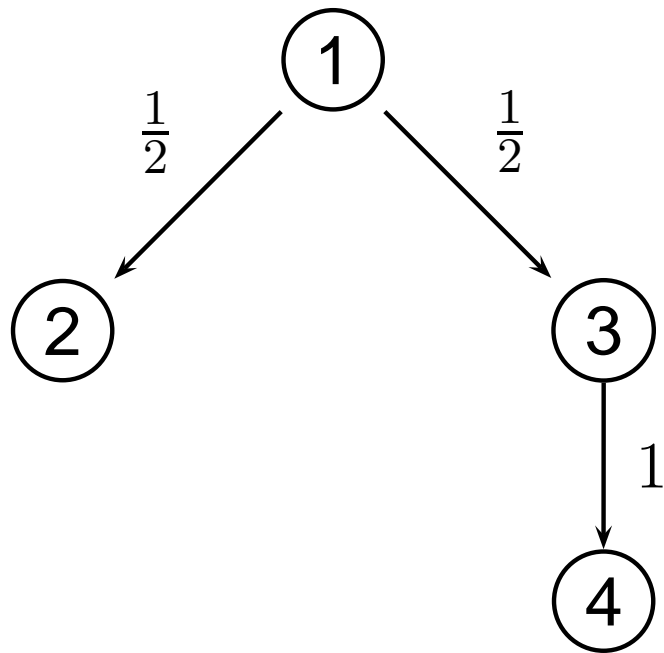
```
procedure PADDING( $T_1, T_2$ )  
   $C \leftarrow$  MAXLUMPING( $T_1, T_2$ )  
  repeat  
    if  $C = \{C_i\}_i$  with  $C_i \cap S_1 = \emptyset, \forall C_i$  then  
      .....  
    else if  $C = \{C_i\}_i$  with  $\forall C_i : C_i \cap S_2 = \emptyset$  then  
      .....  
    else if  $C = \{C_i\}_i$  with  $C_j \subseteq S_1 \neq \emptyset \neq C_k \subseteq S_2$  then  
      choose  $s_1 \in C_k$   
      choose  $s_2 \in C_j$   
      FIXIT( $T_1, s_1, T_2, s_2$ )  
    end if  
     $C \leftarrow$  MAXLUMPING( $T_1, T_2$ )  
  until  $C = \emptyset$   
  return  $T_1, T_2$   
end procedure
```

Lumping

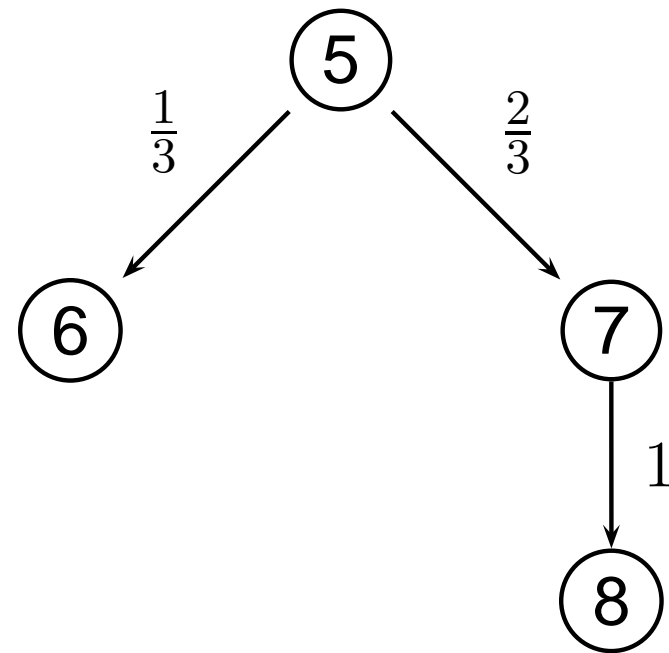


Lumping

A

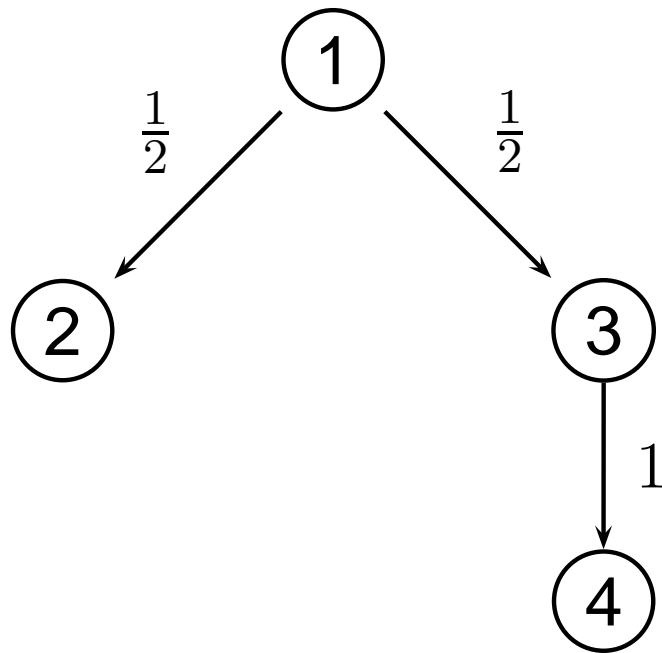


B

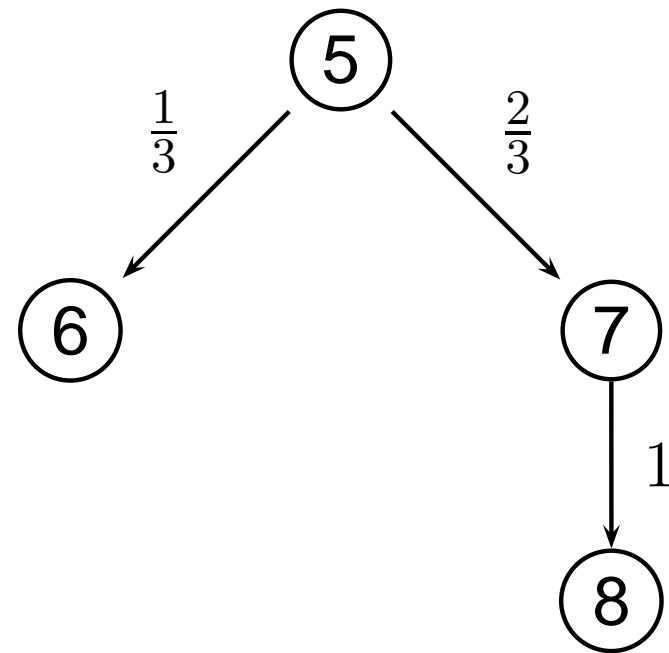


Lumping

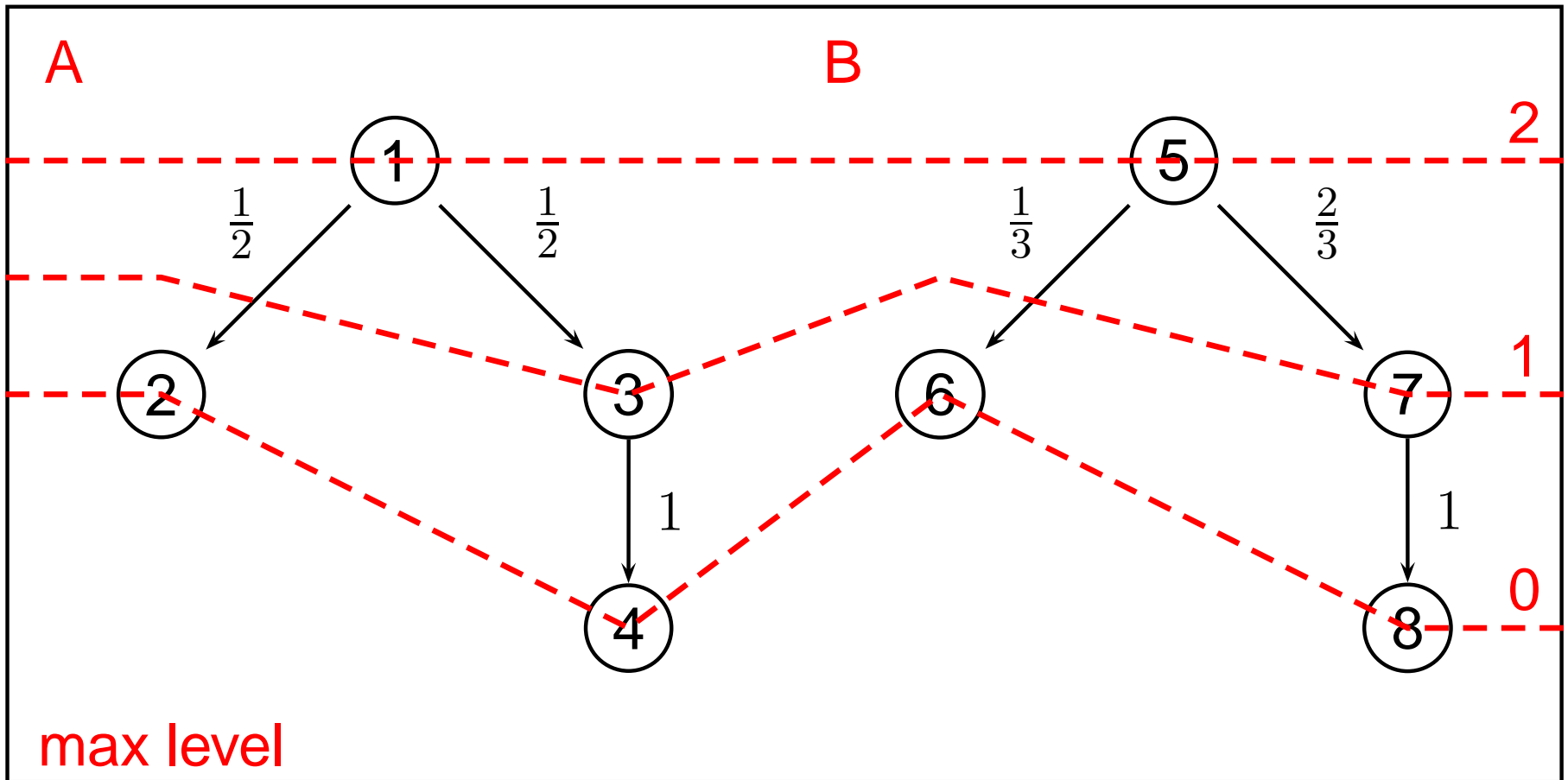
A



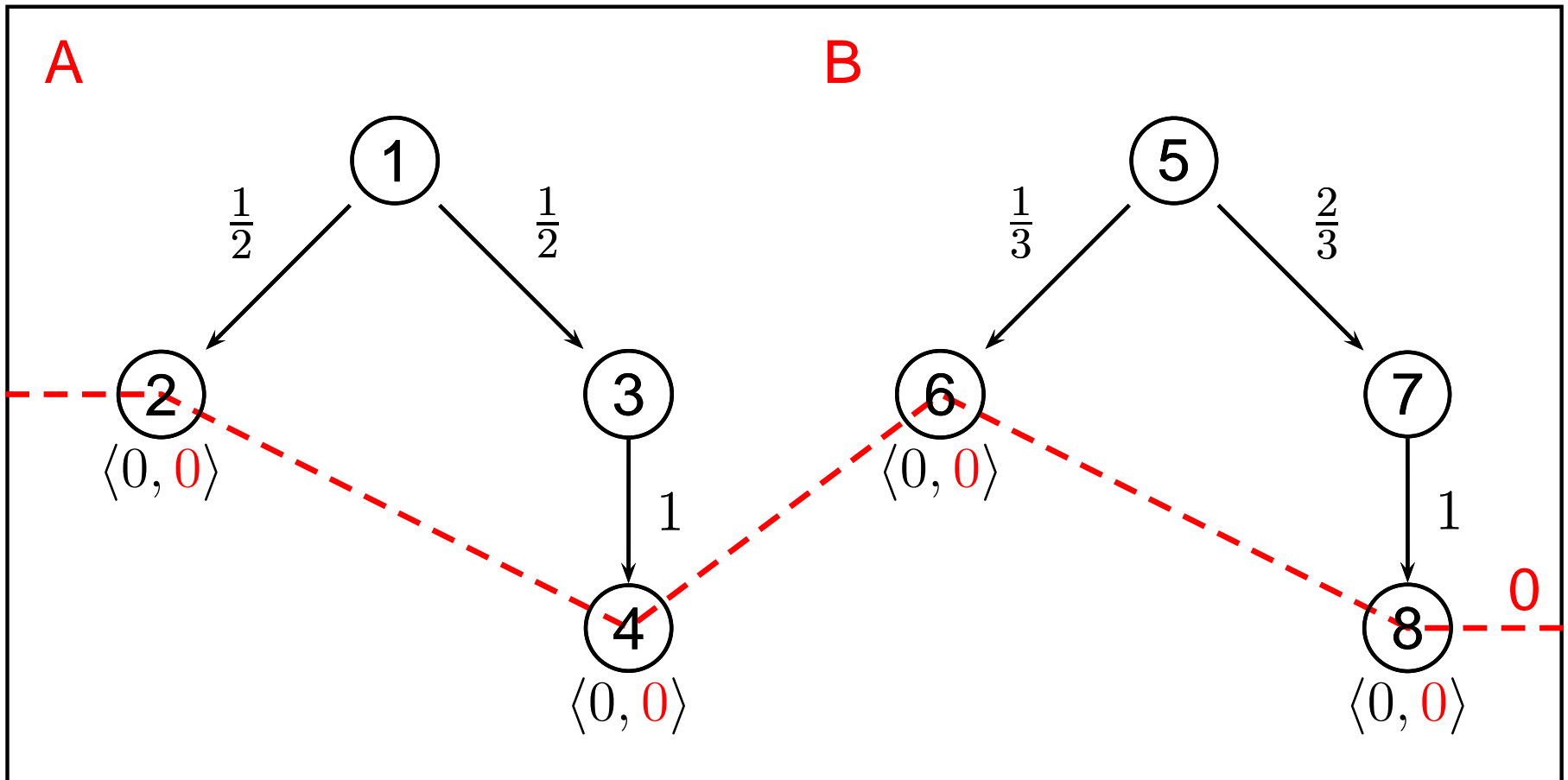
B



Lumping



Lumping

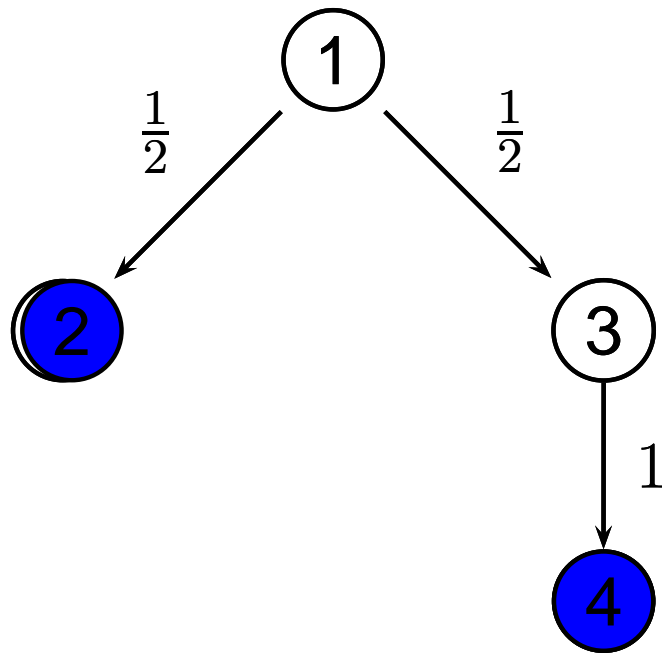


$$P = \{\{2, 4, 6, 8\}\}$$

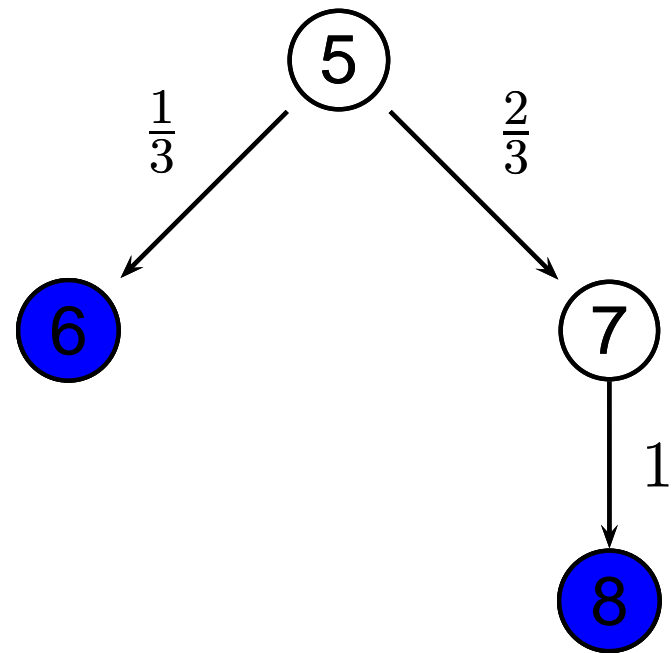
$$S = \{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}$$

Lumping

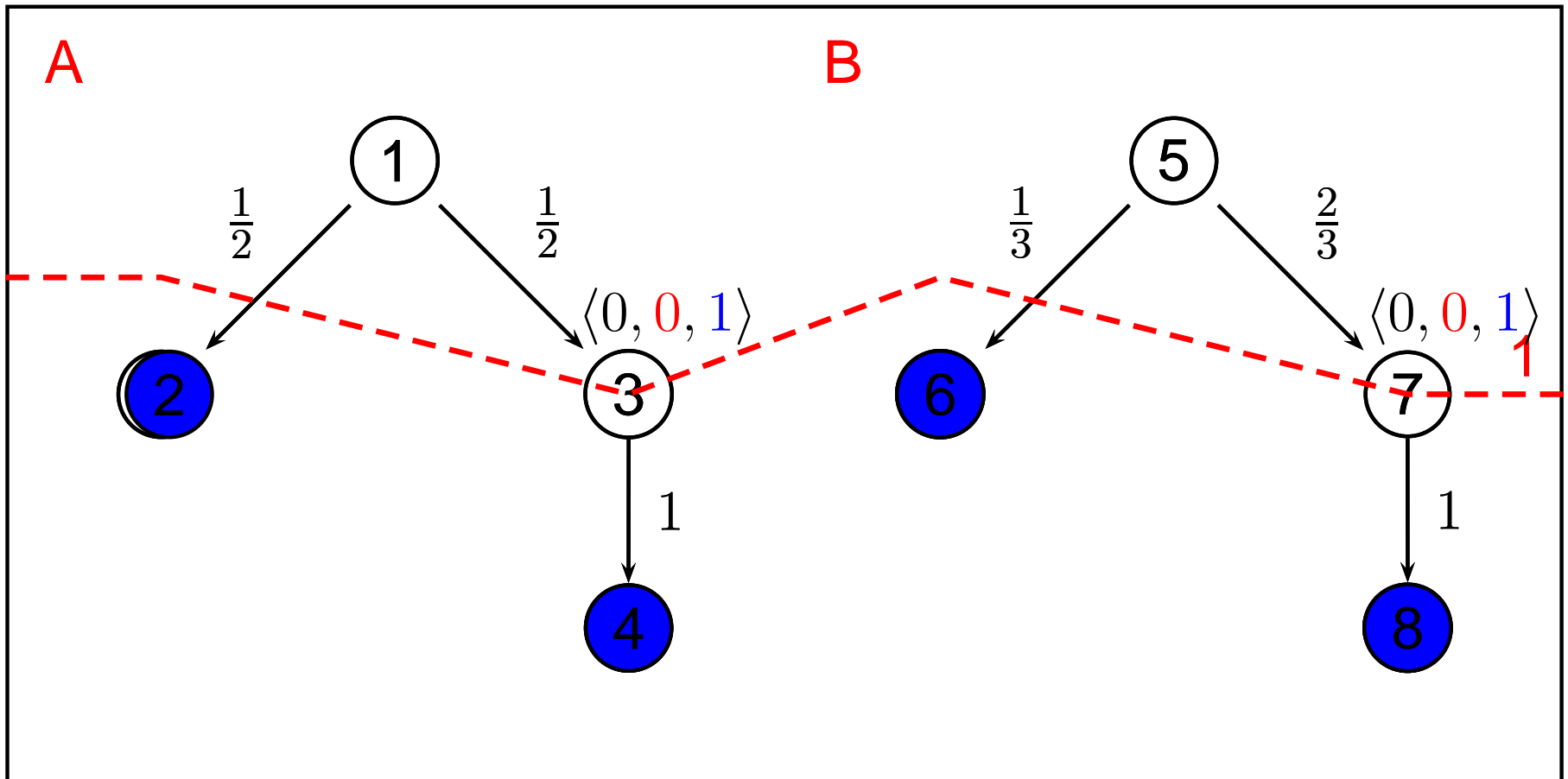
A



B



Lumping

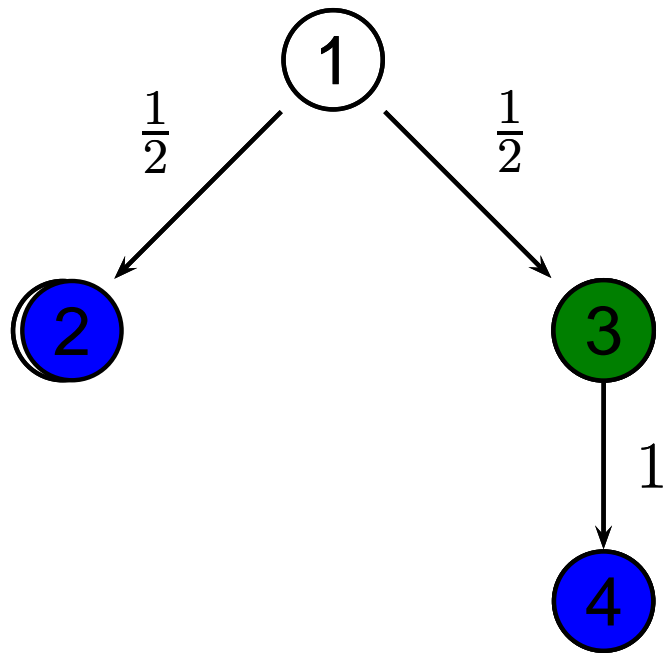


$$P = \{\{3, 7\}, \{2, 4, 6, 8\}\}$$

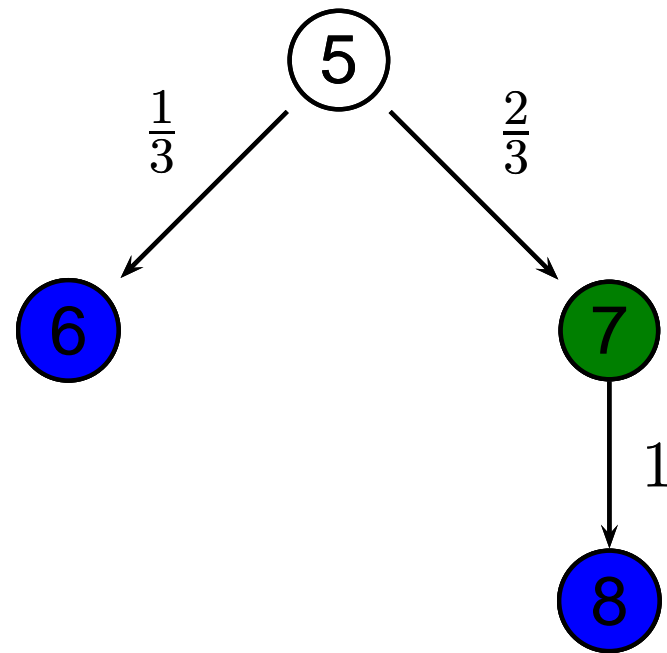
$$S = \{\{1, 5\}, \{3, 7\}, \{2, 4, 6, 8\}\}$$

Lumping

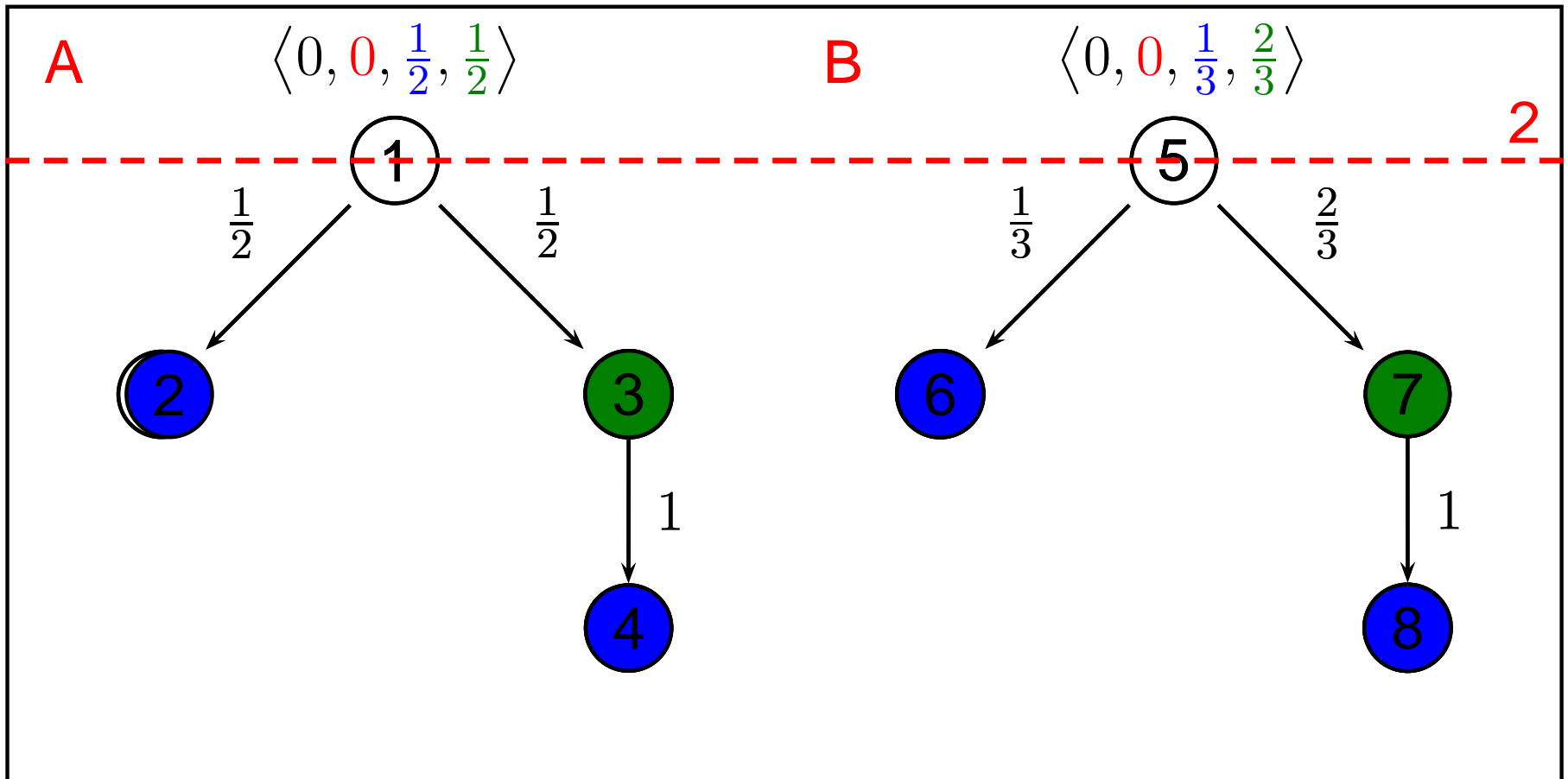
A



B



Lumping

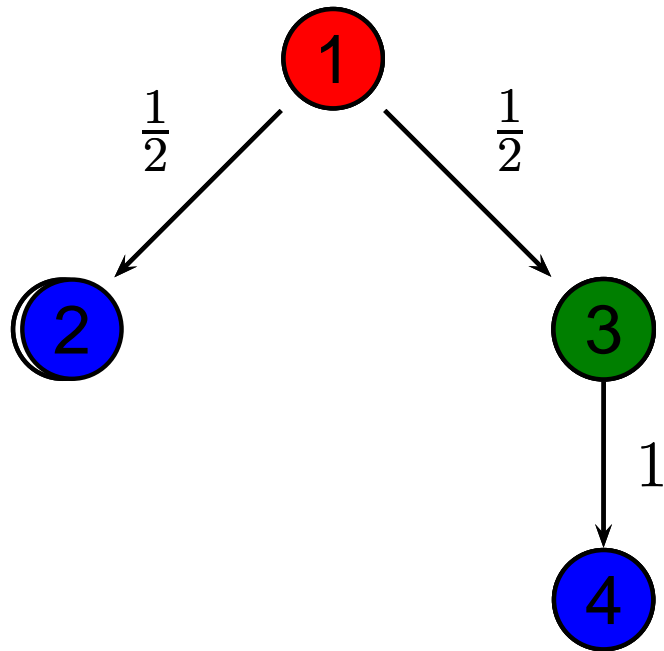


$$P = \{\{3, 7\}, \{2, 4, 6, 8\}\}$$

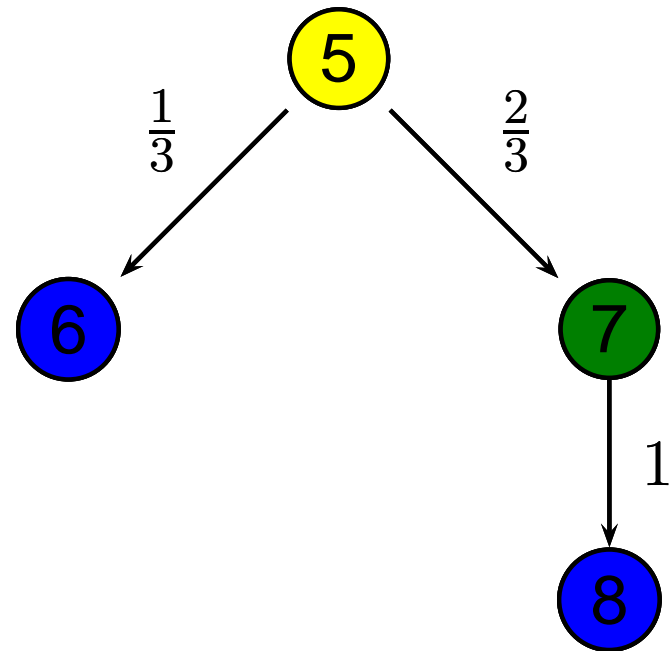
$$S = \{\{1, 5\}, \{3, 7\}, \{2, 4, 6, 8\}\}$$

Lumping

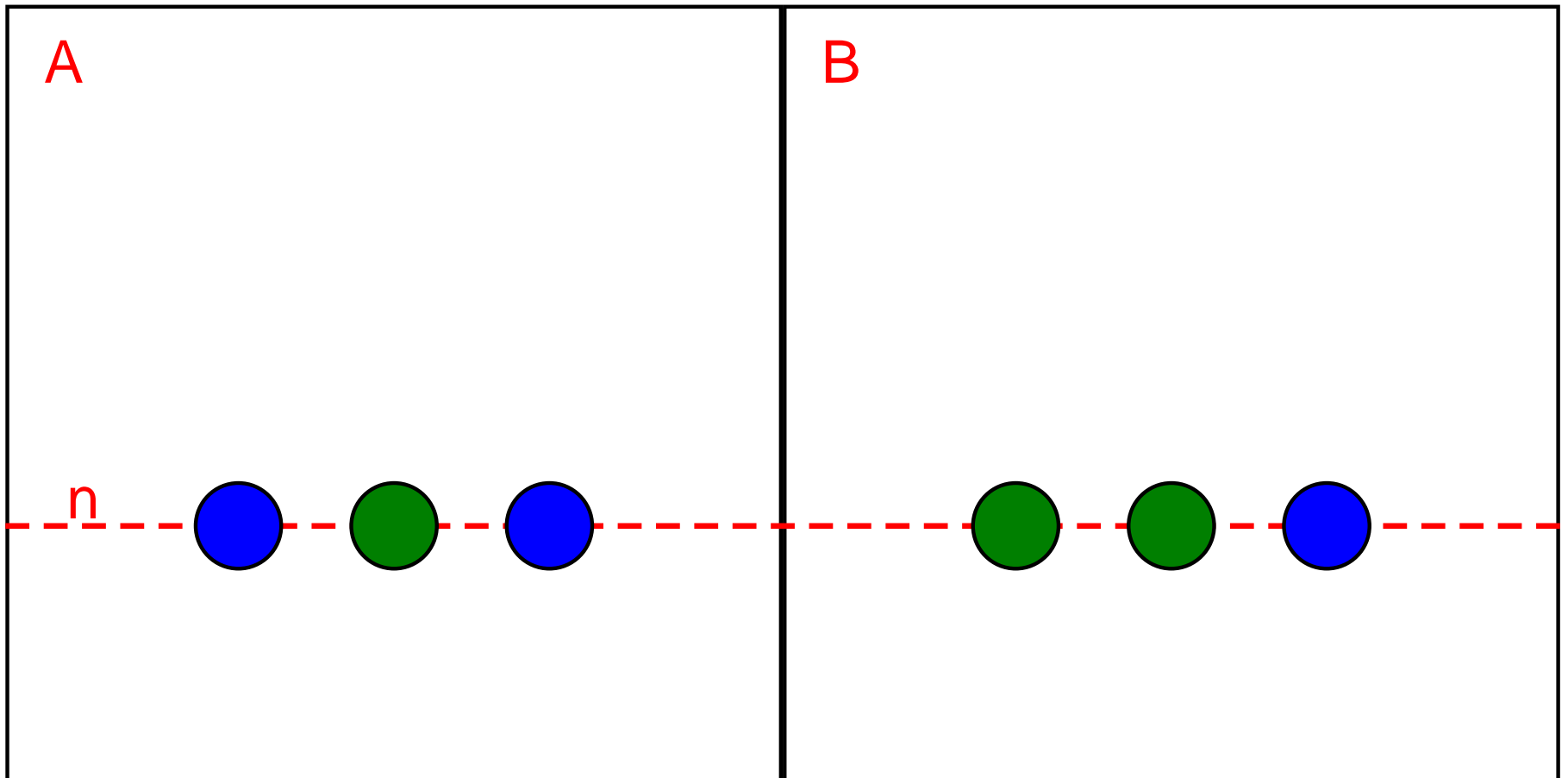
A



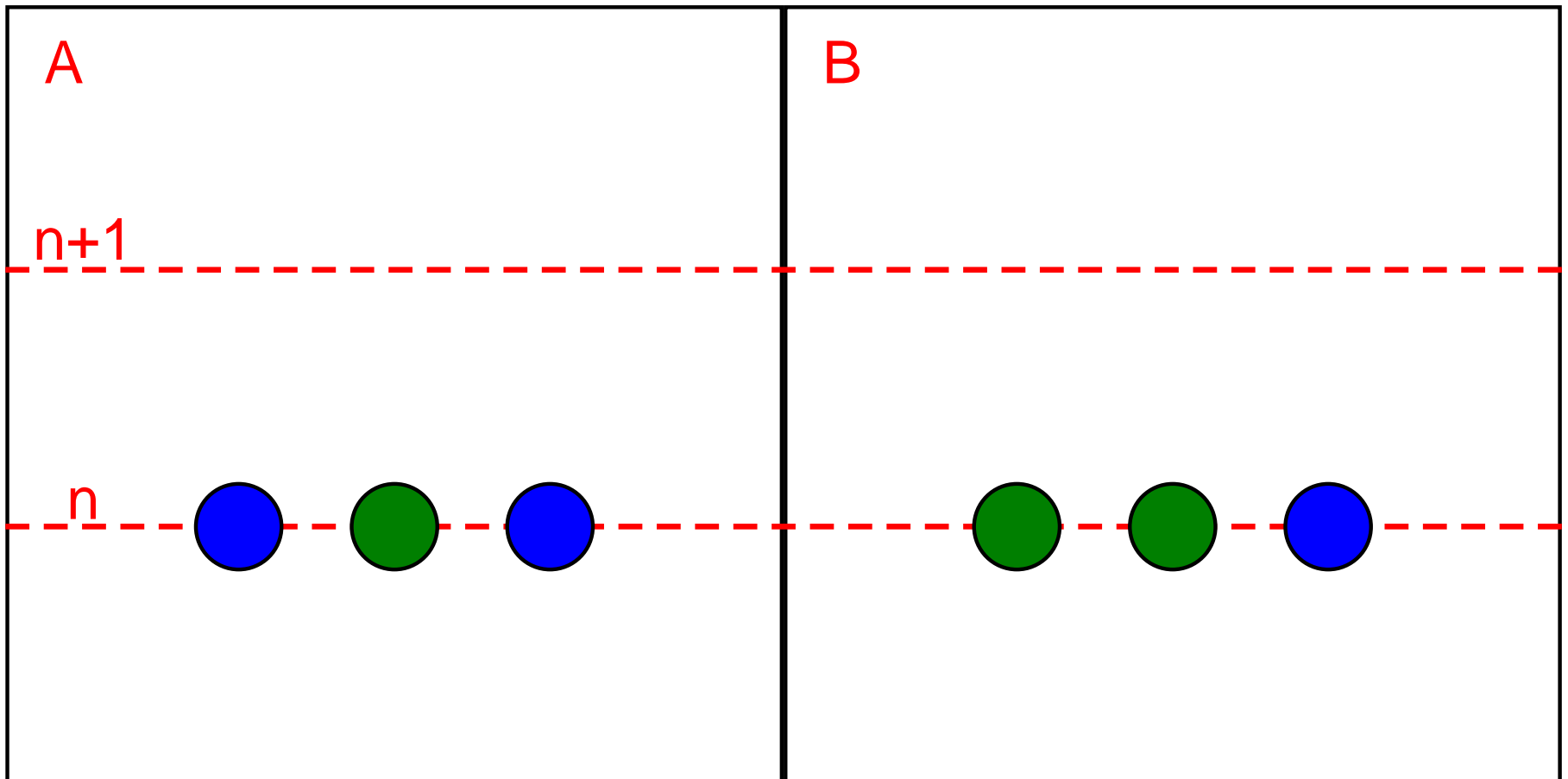
B



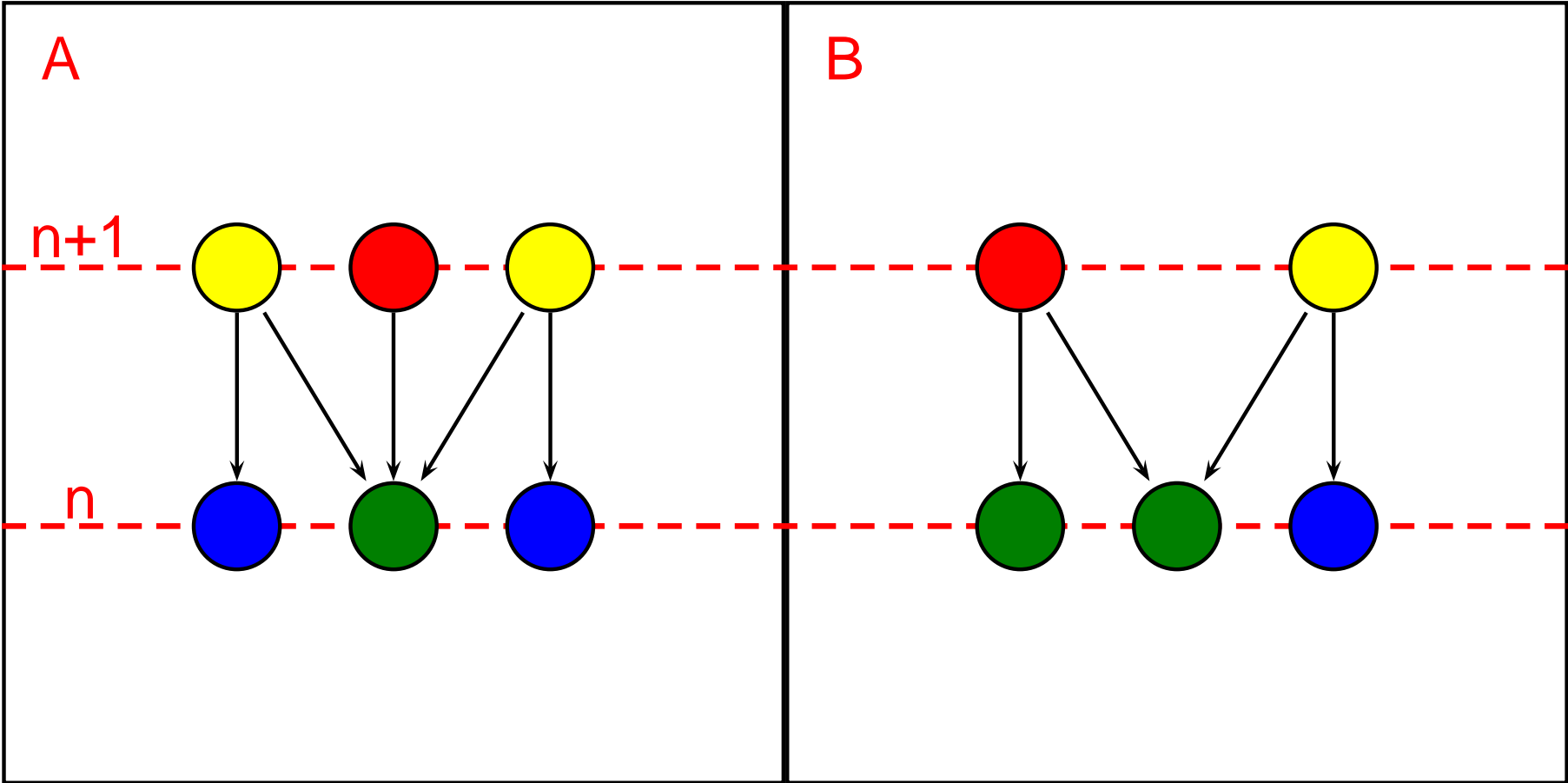
Critical Classes



Critical Classes

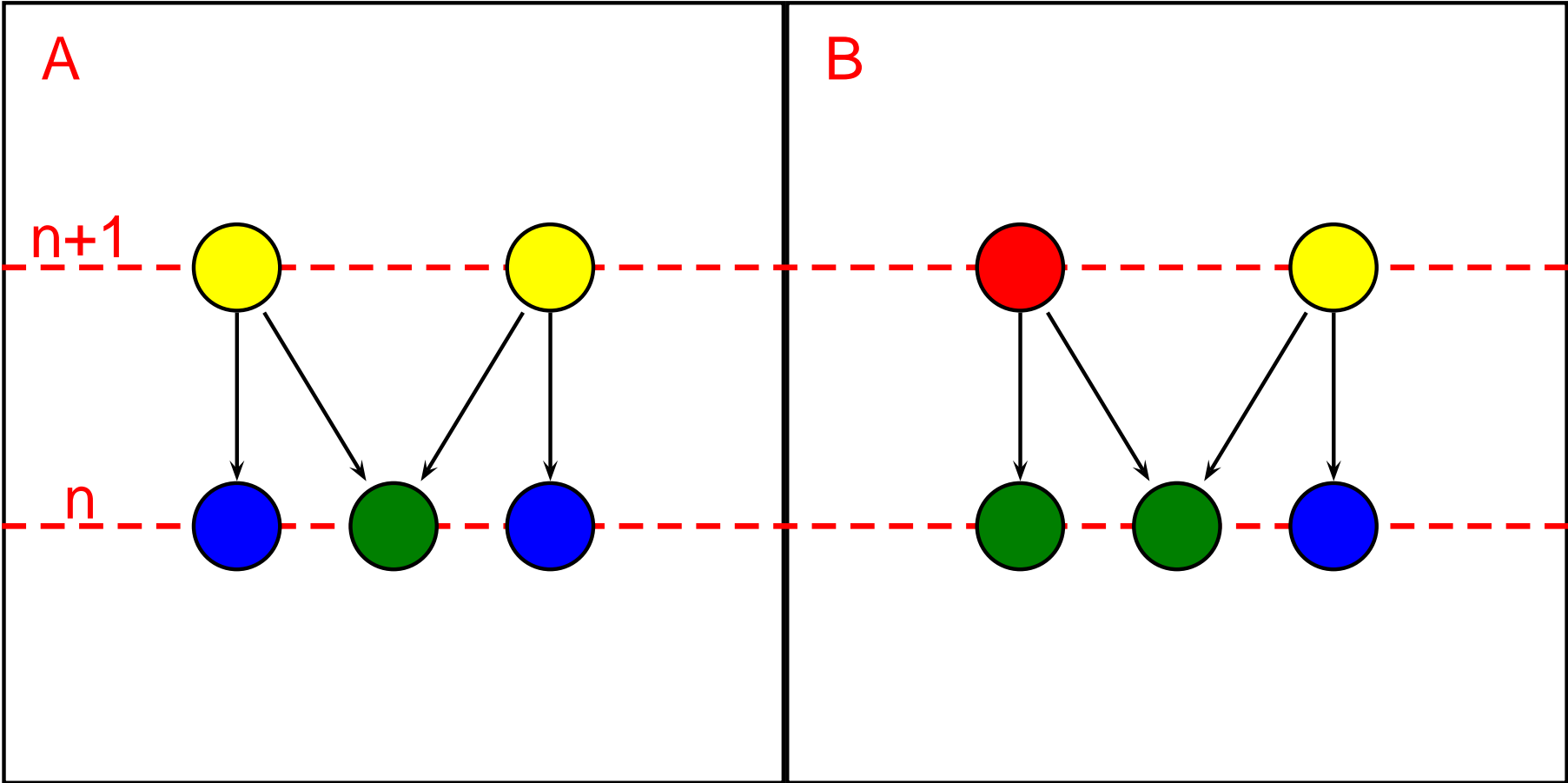


Critical Classes



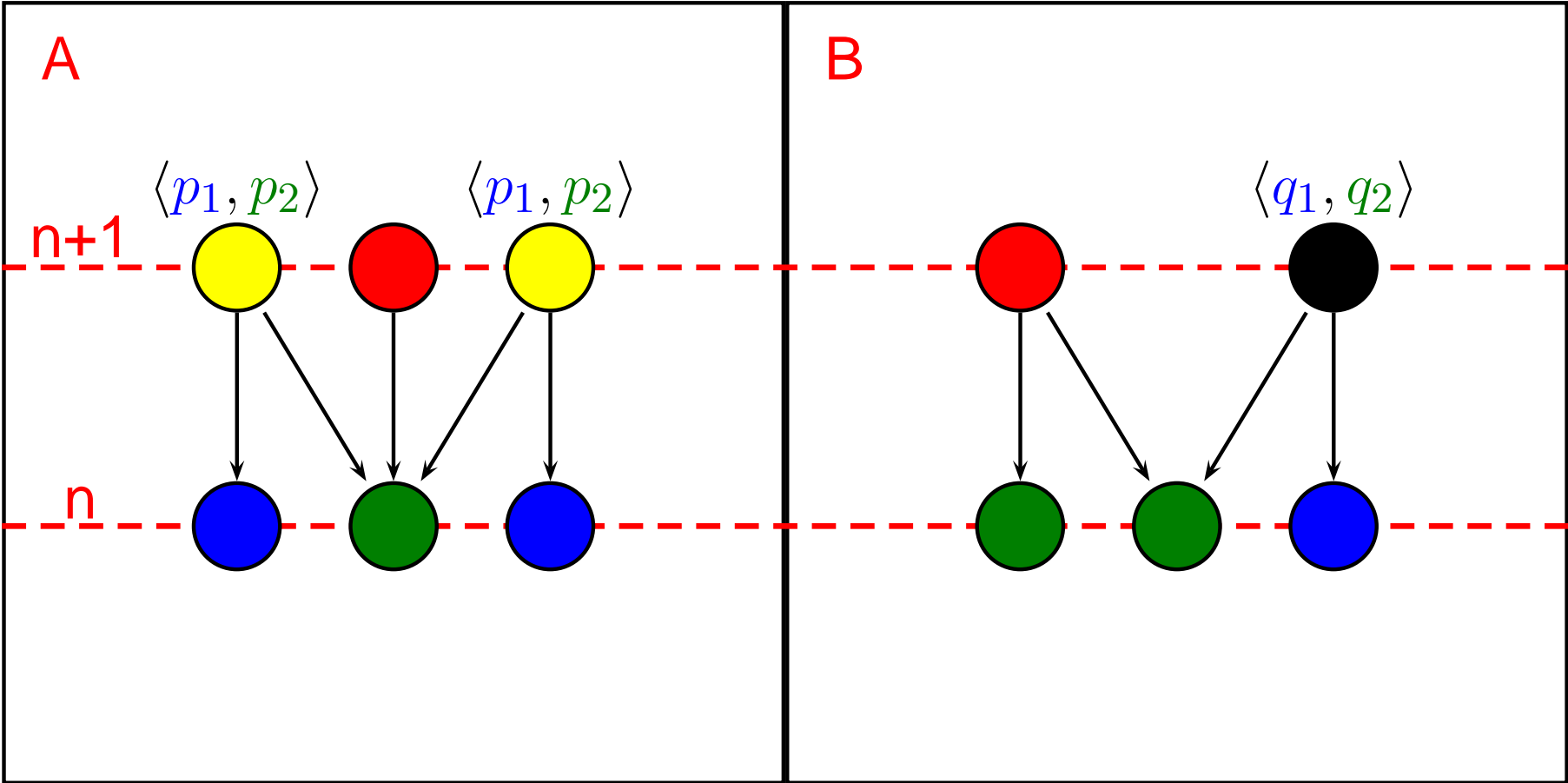
Nothing To Do

Critical Classes



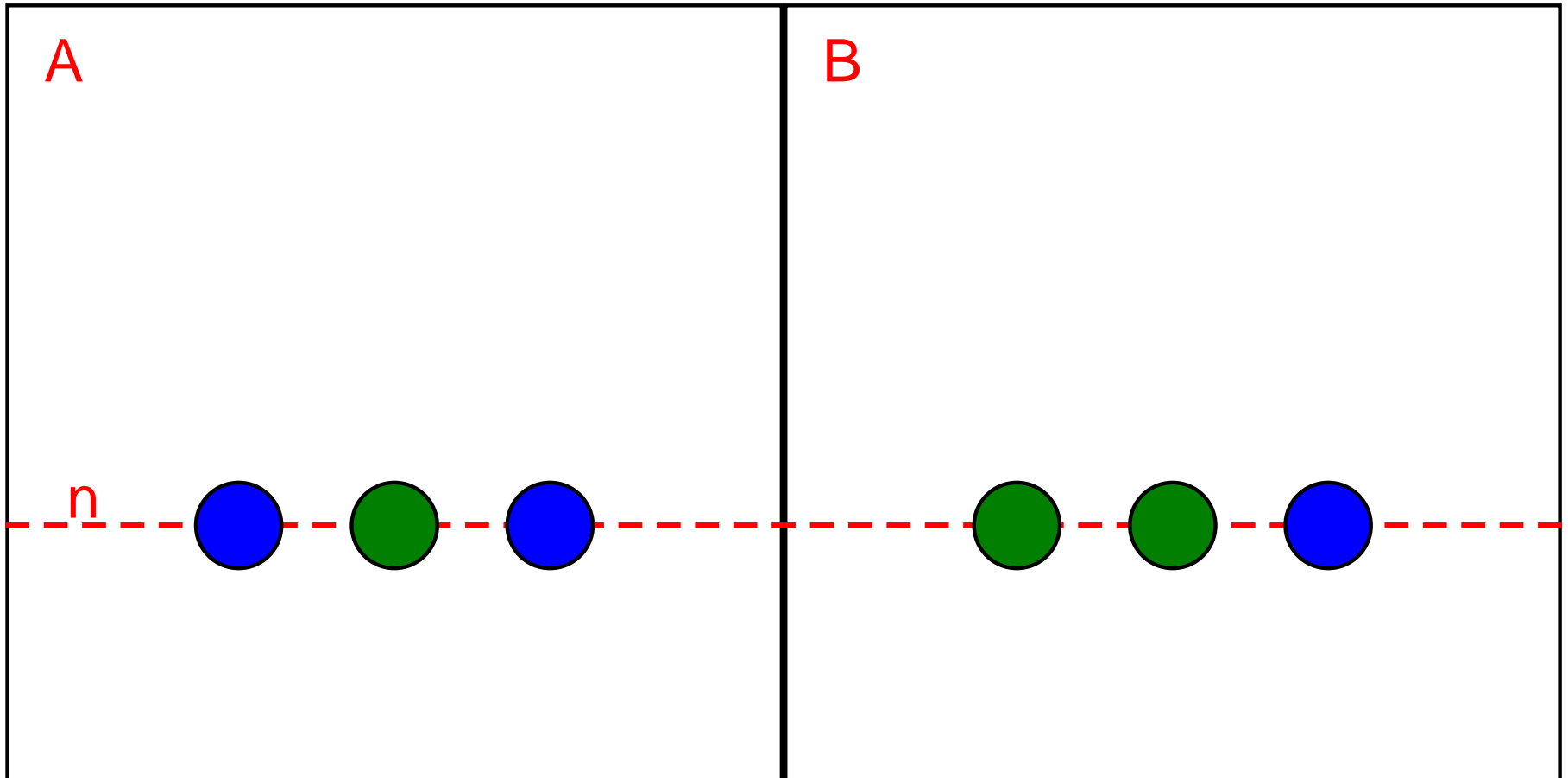
Extension

Critical Classes

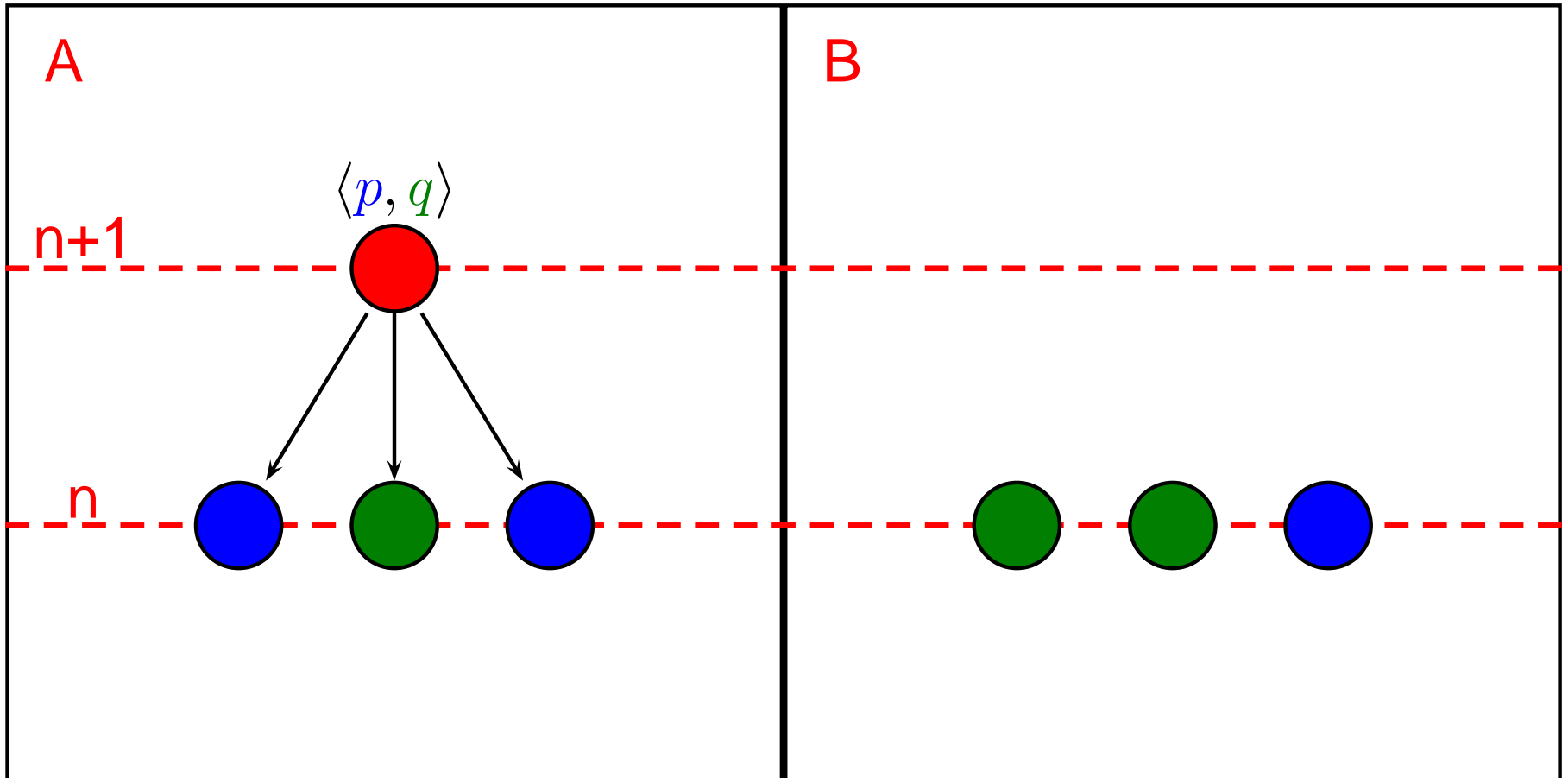


Redistribution

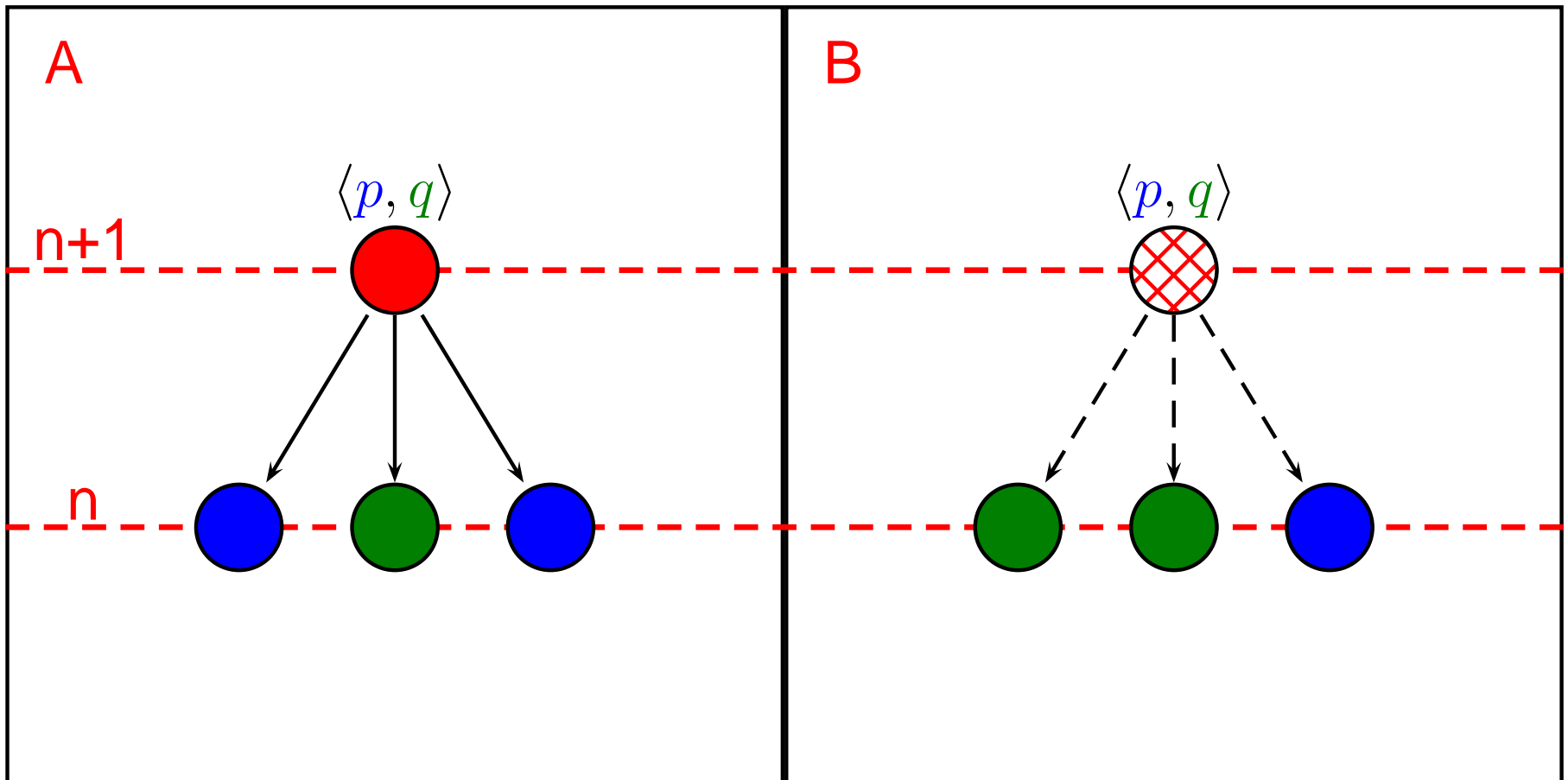
Extension



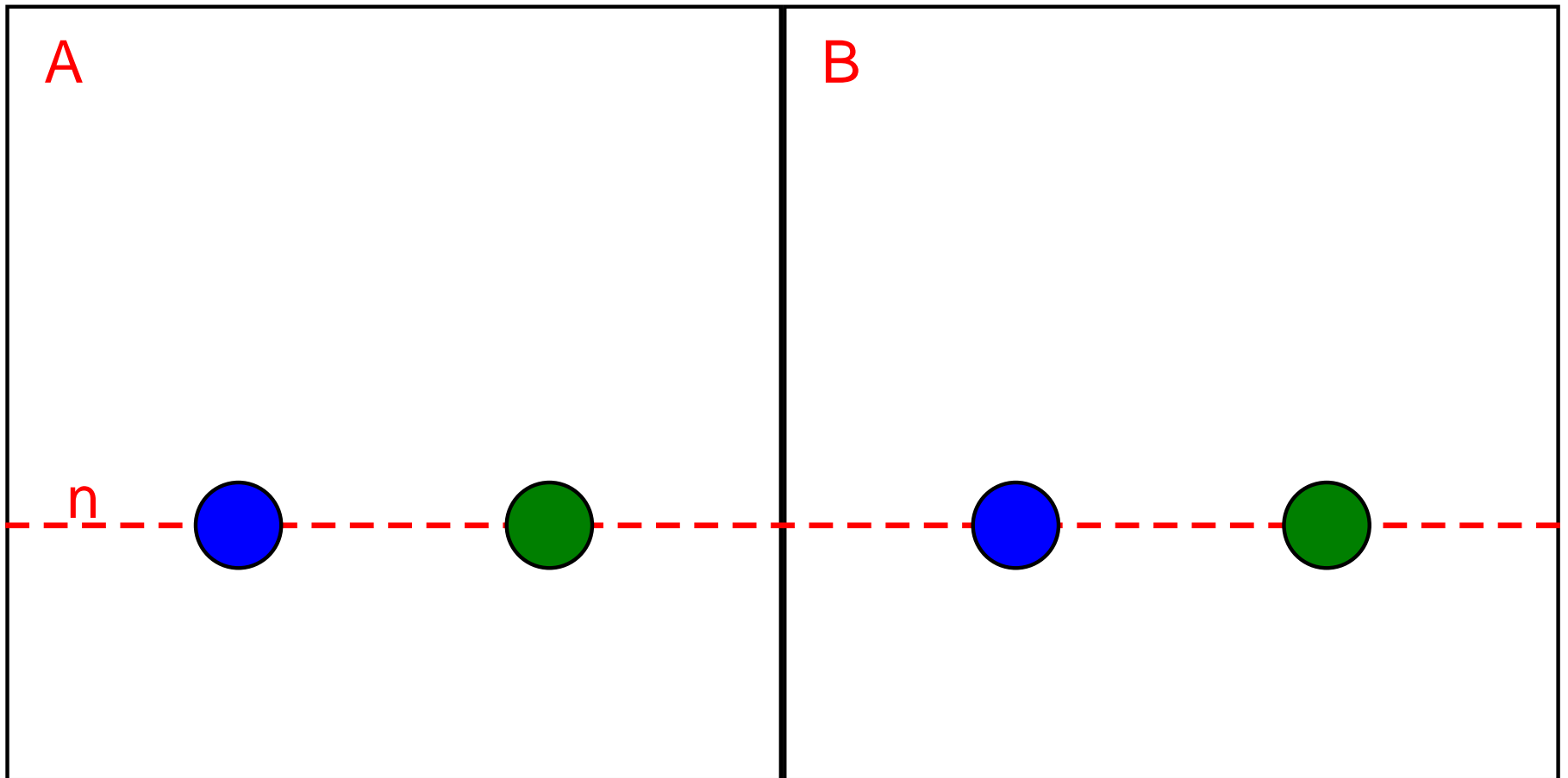
Extension



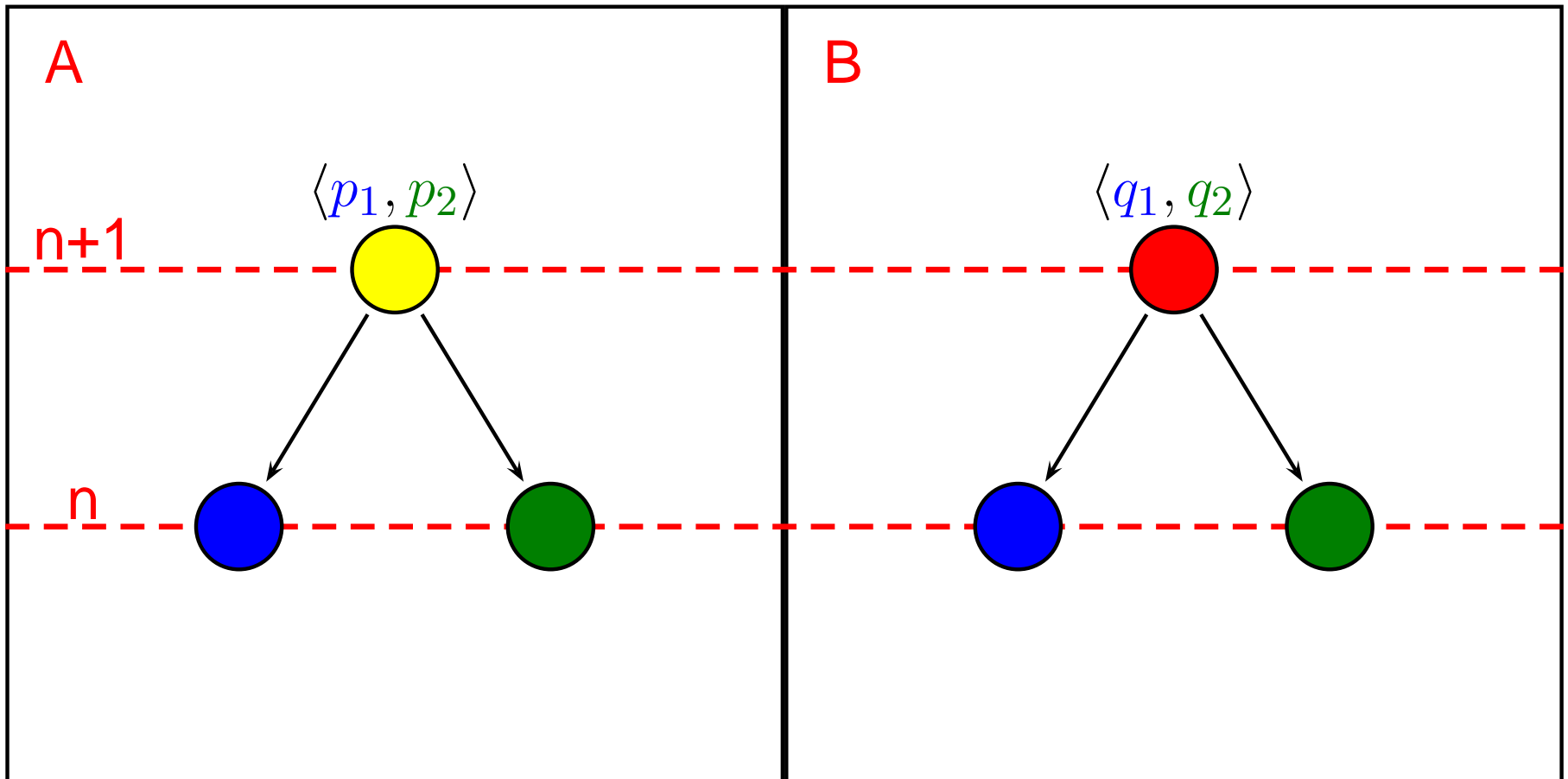
Extension



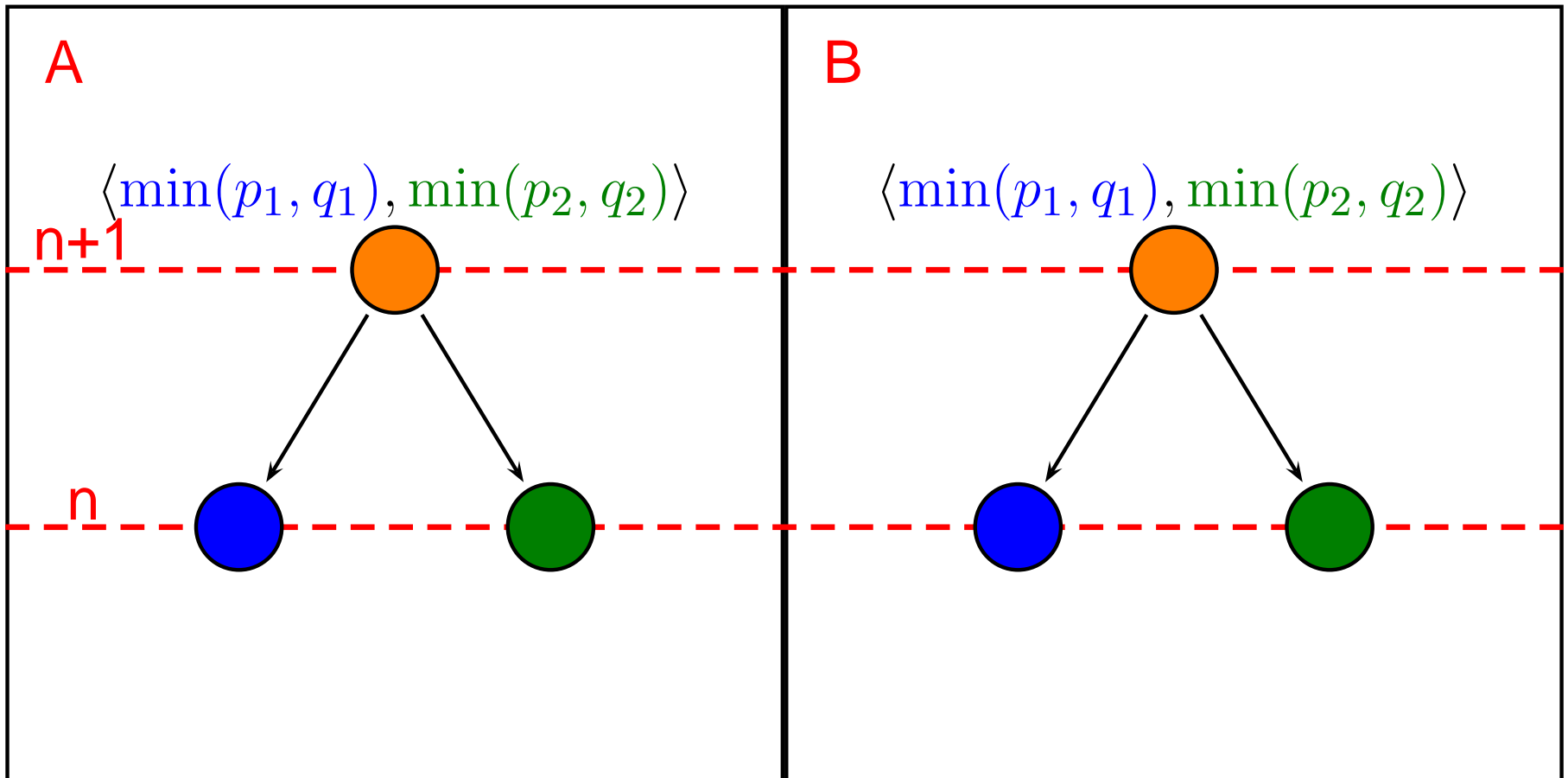
Redistribution



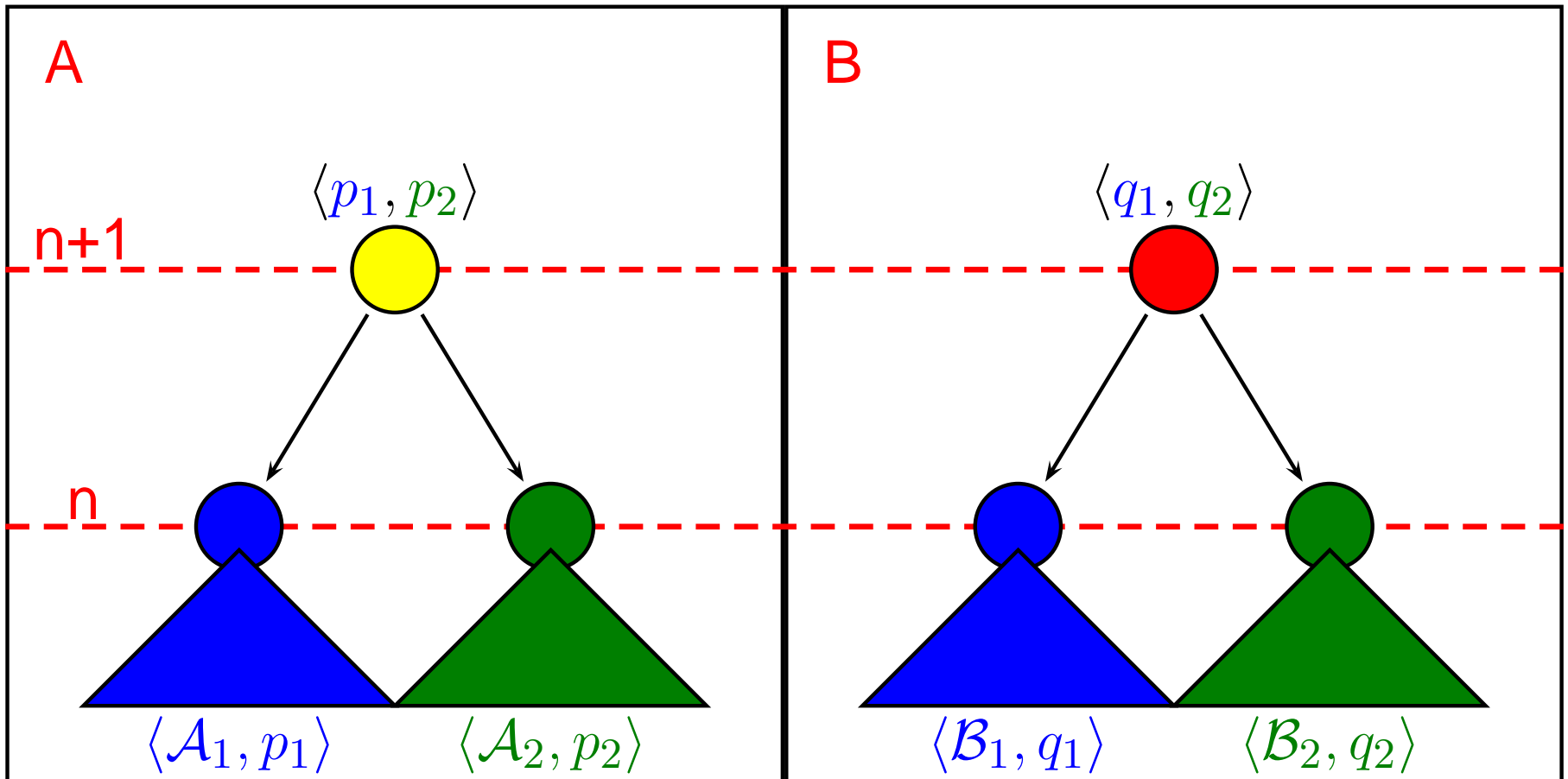
Redistribution



Redistribution

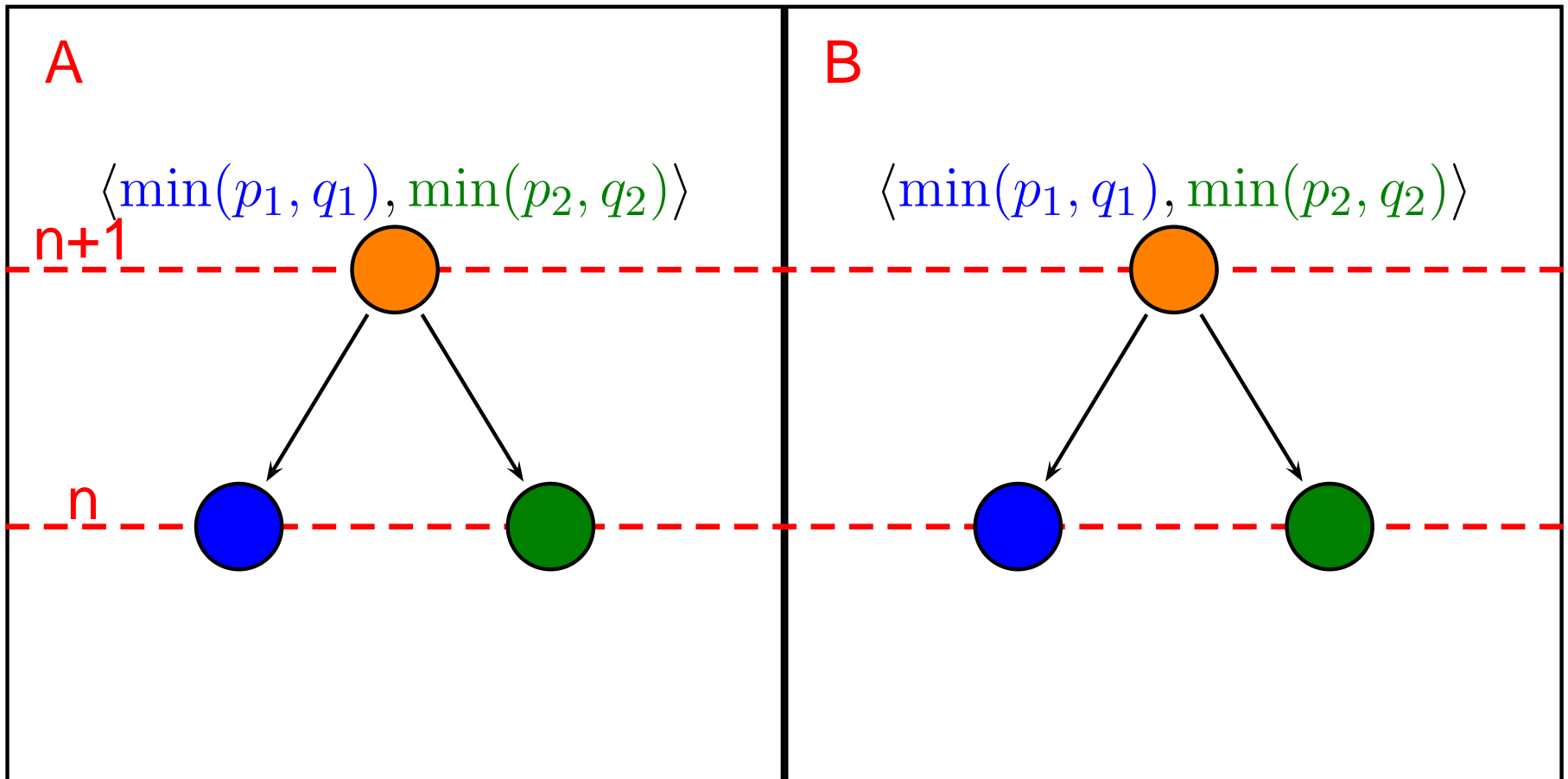


Redistribution



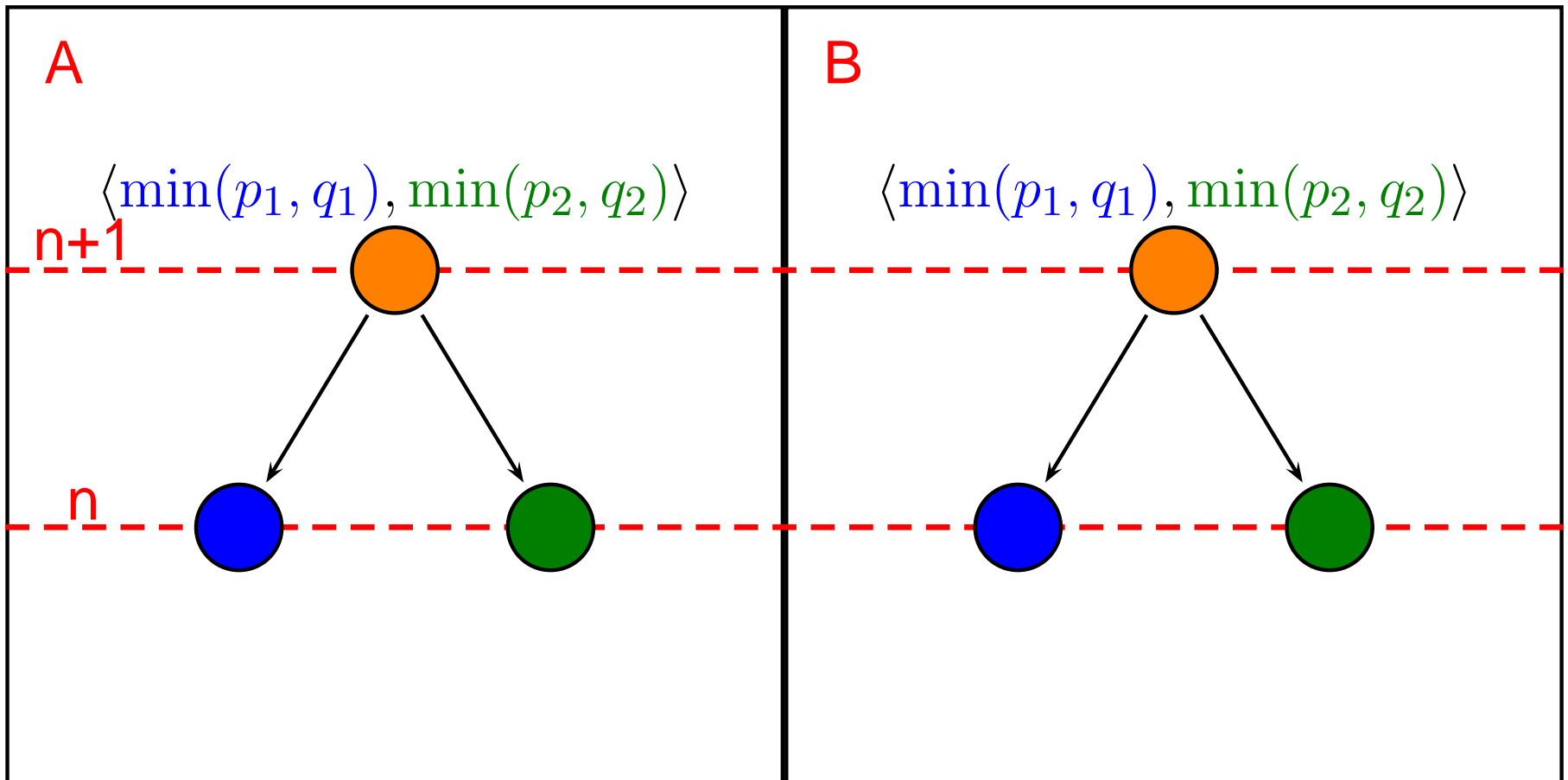
Consider Observables

Redistribution



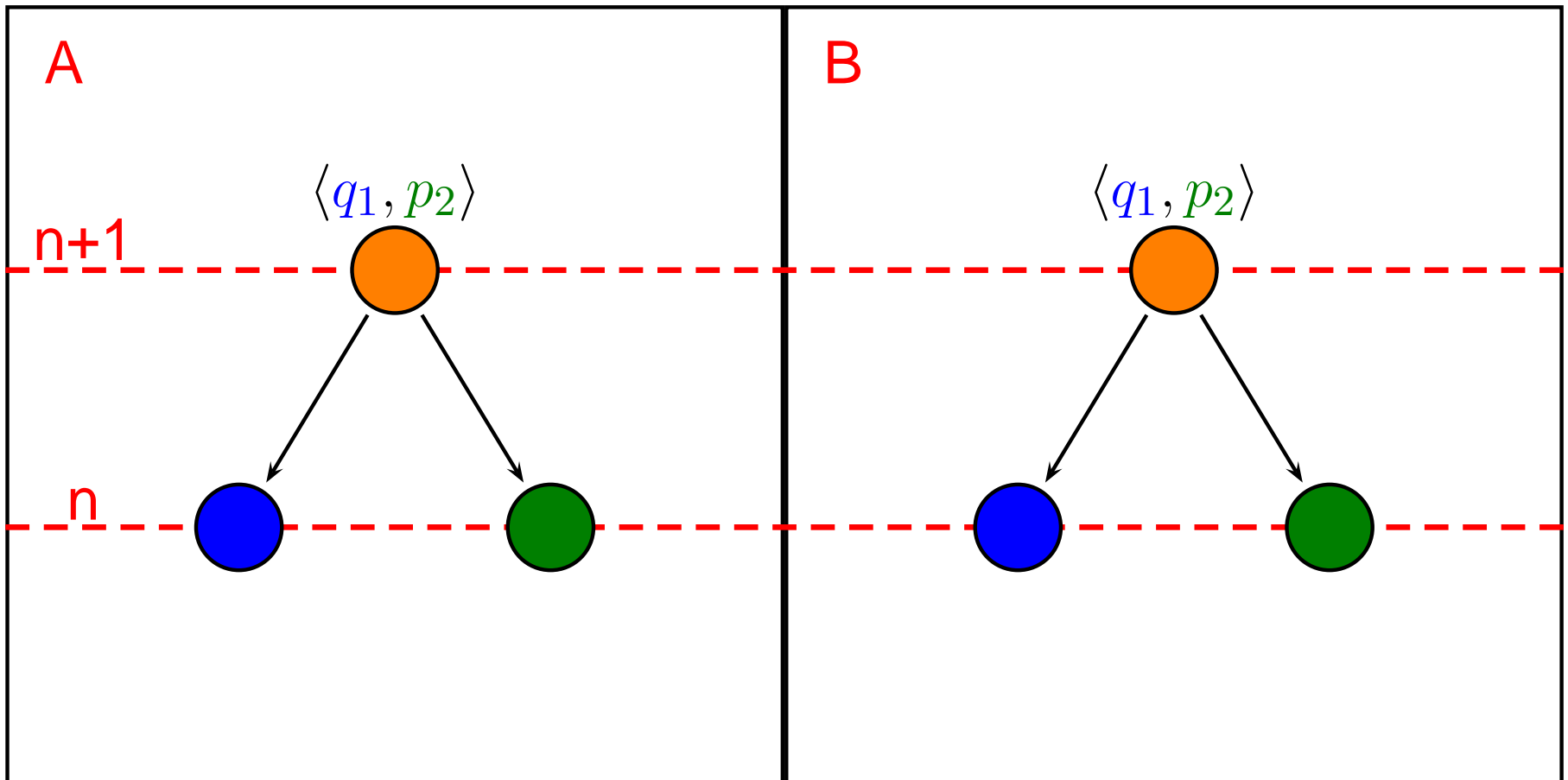
Consider Observables

Redistribution



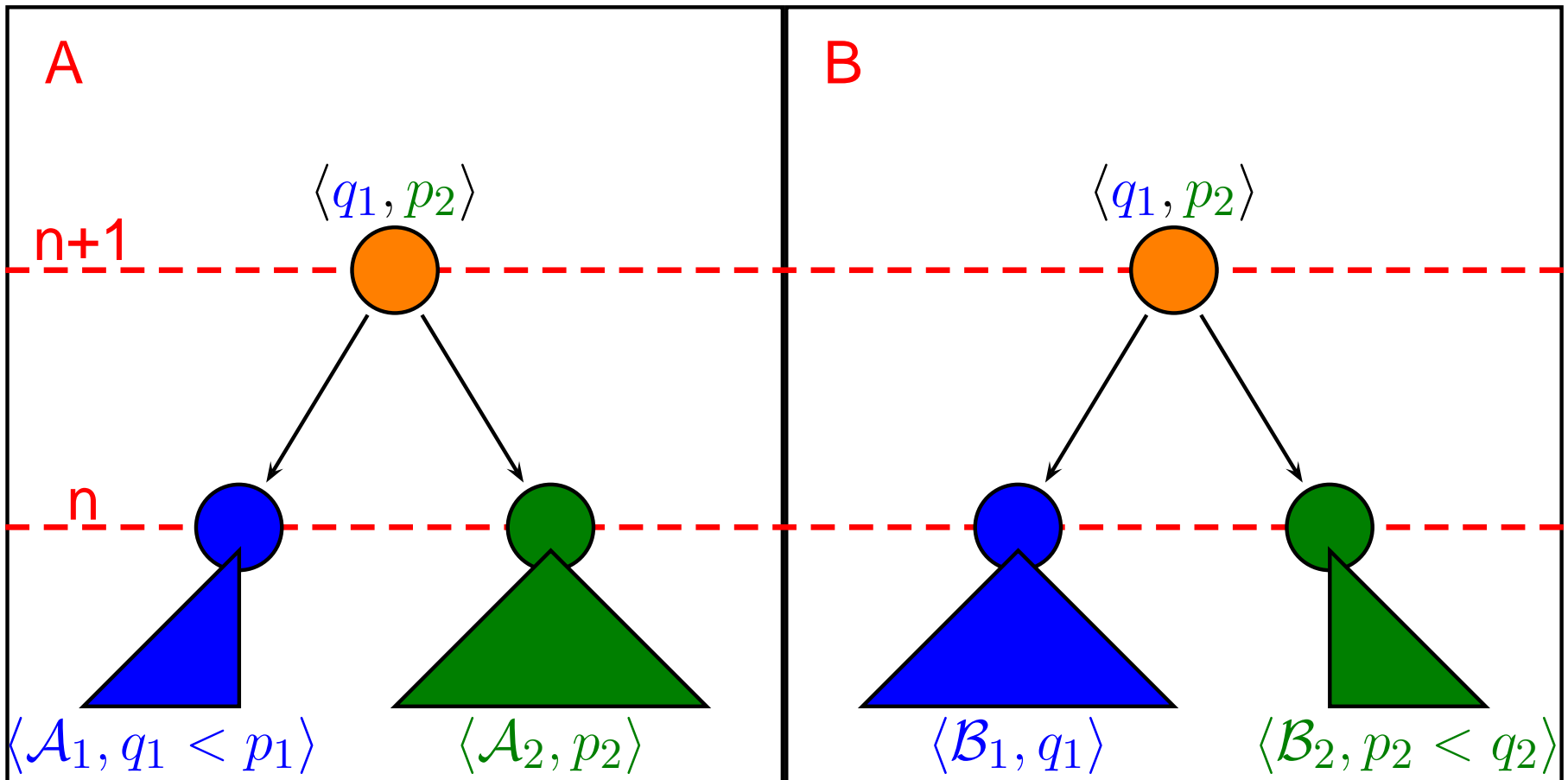
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Redistribution



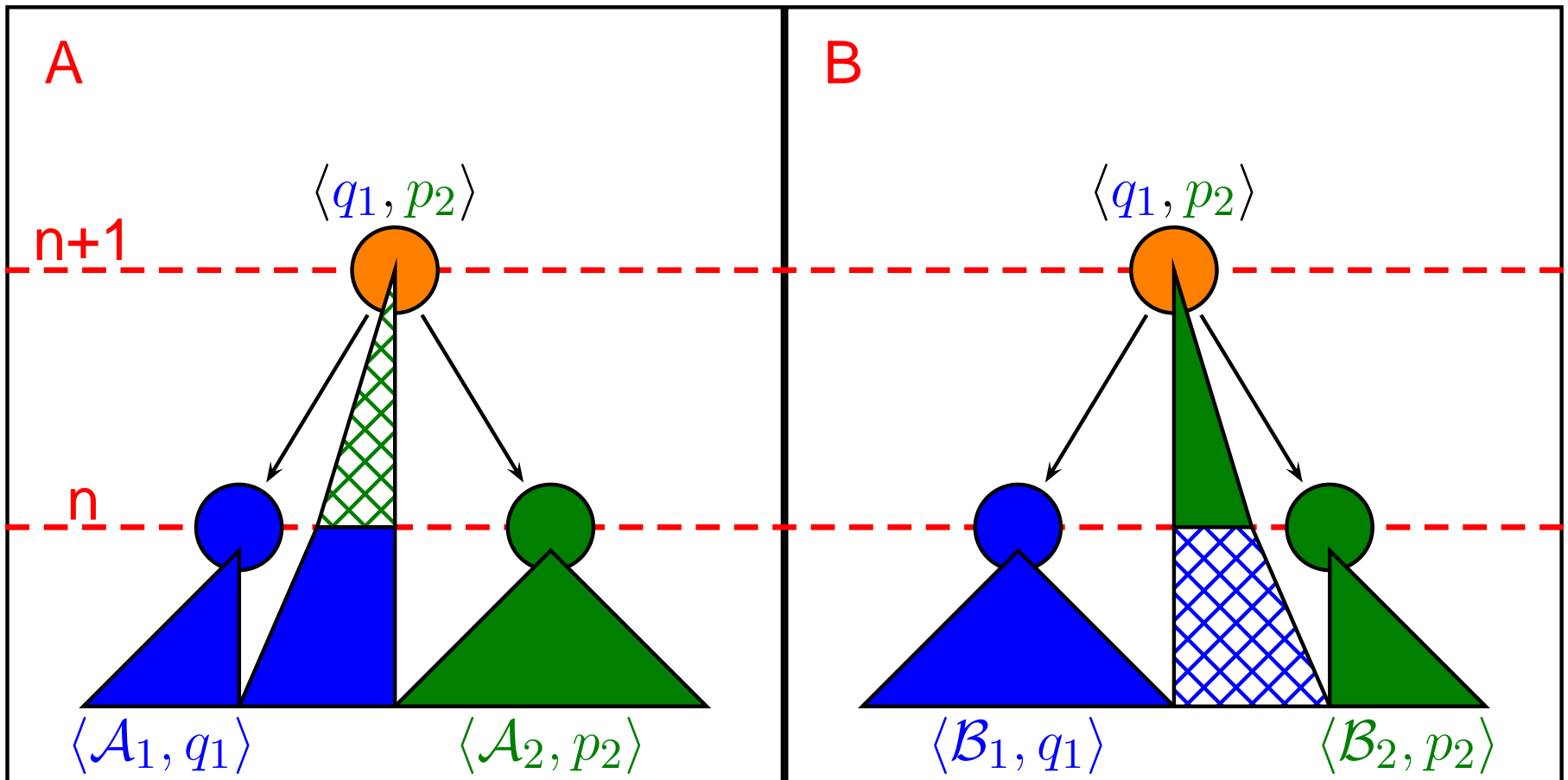
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Redistribution



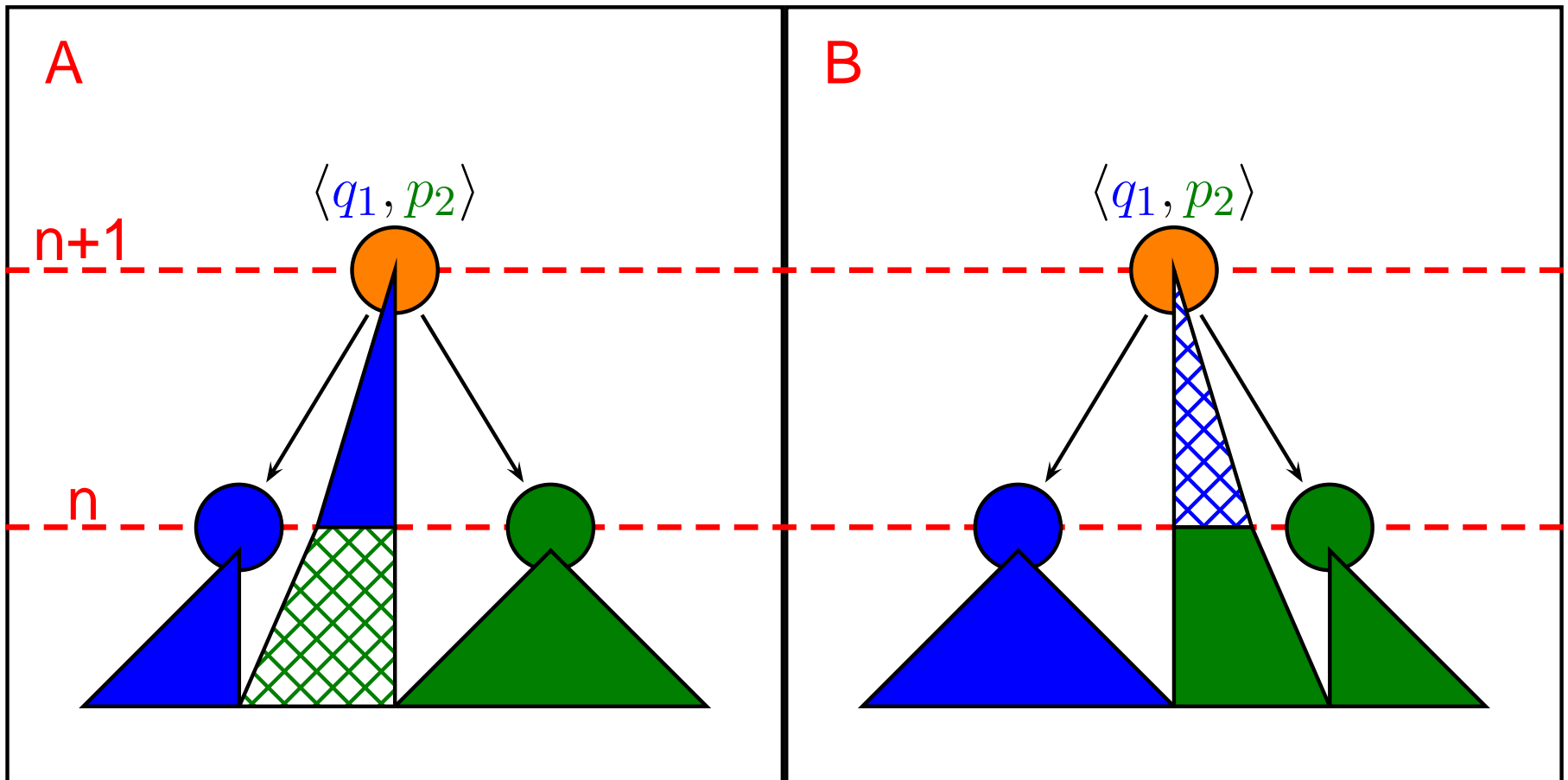
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Redistribution



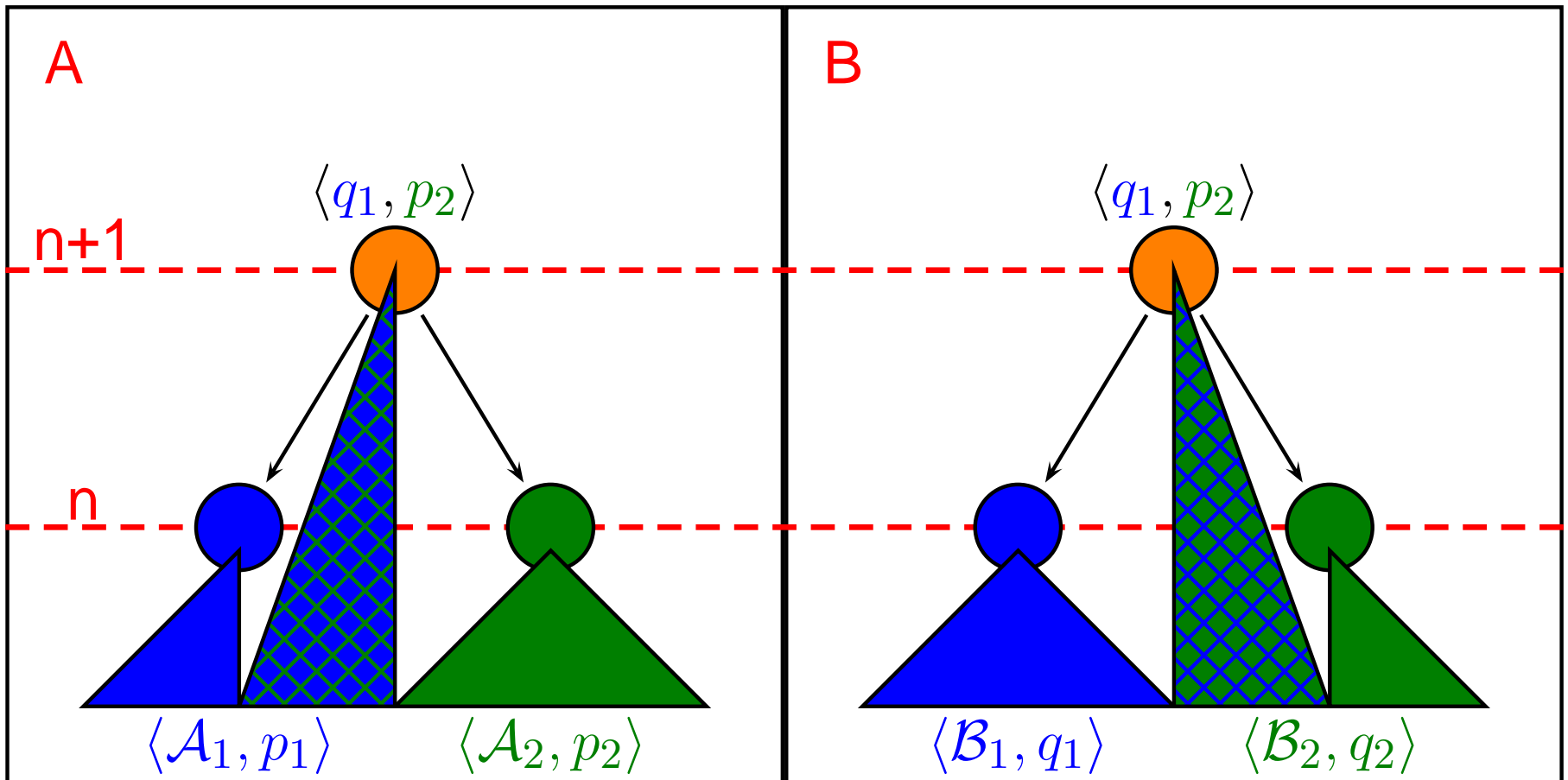
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Redistribution



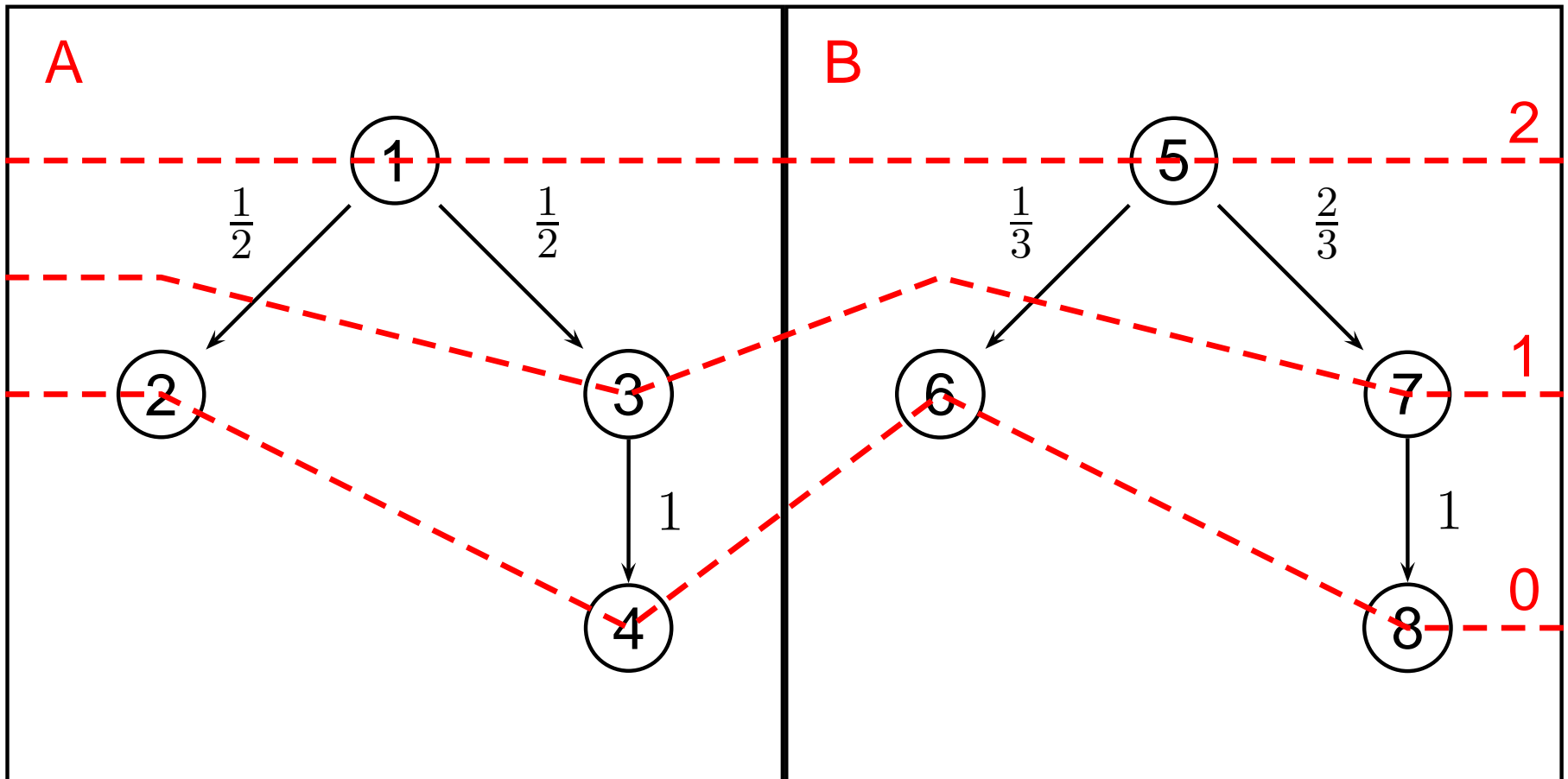
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Redistribution



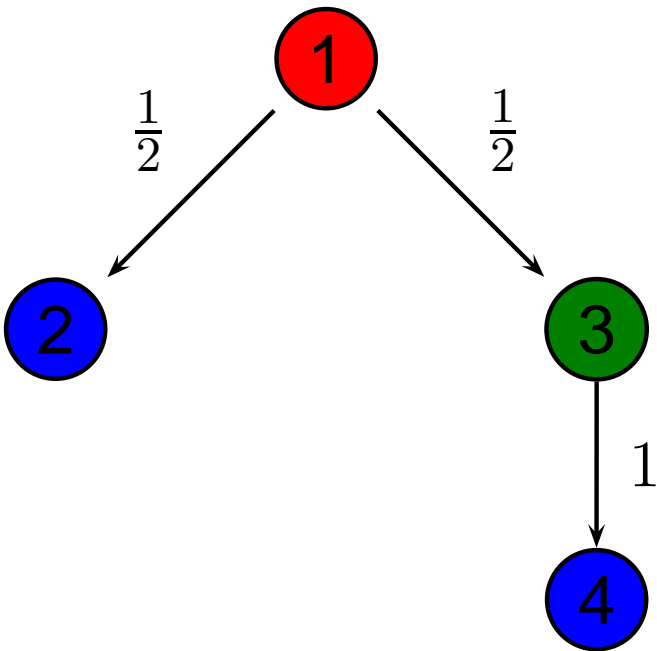
w.l.o.g. $p_1 > q_1$ and $p_2 < q_2$

Padding: Redistribution

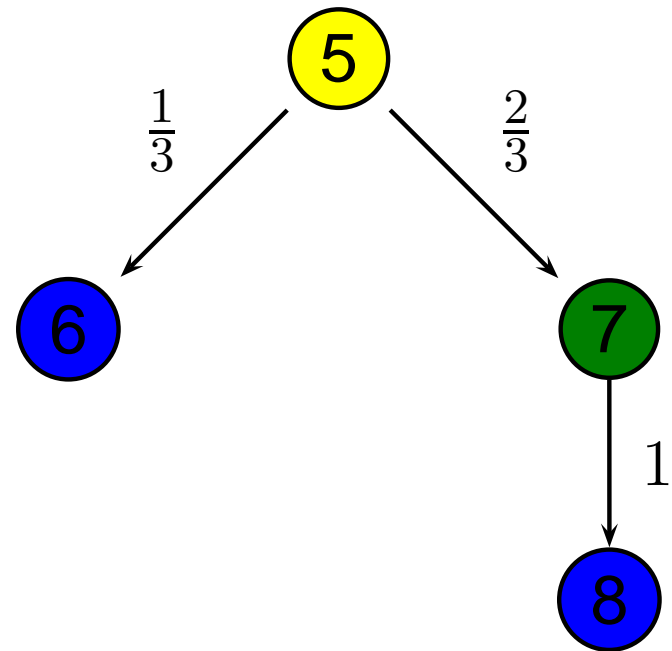


Padding: Redistribution

A

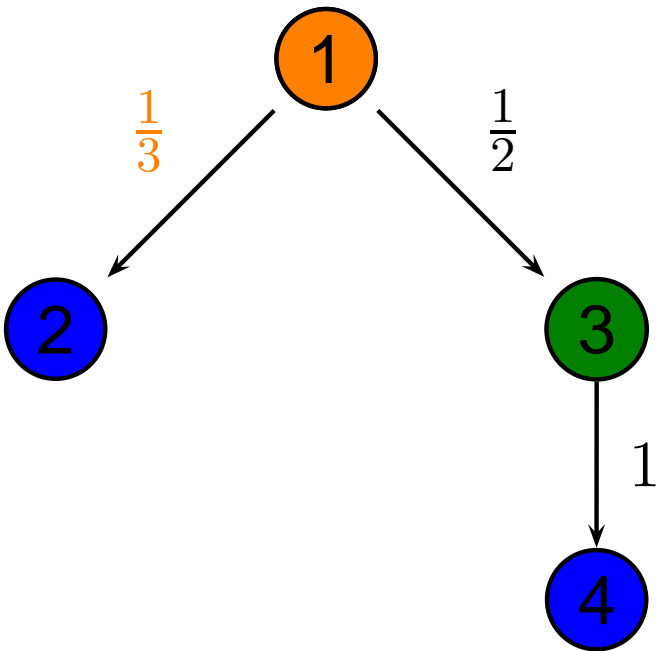


B

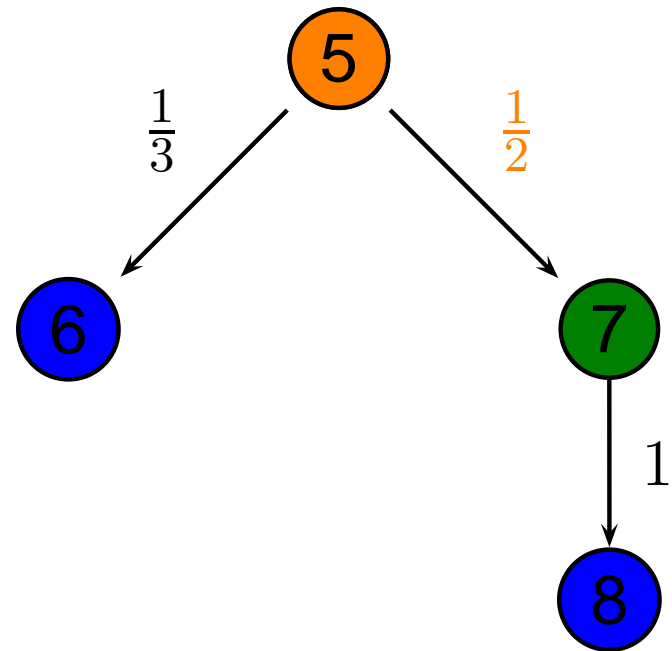


Padding: Redistribution

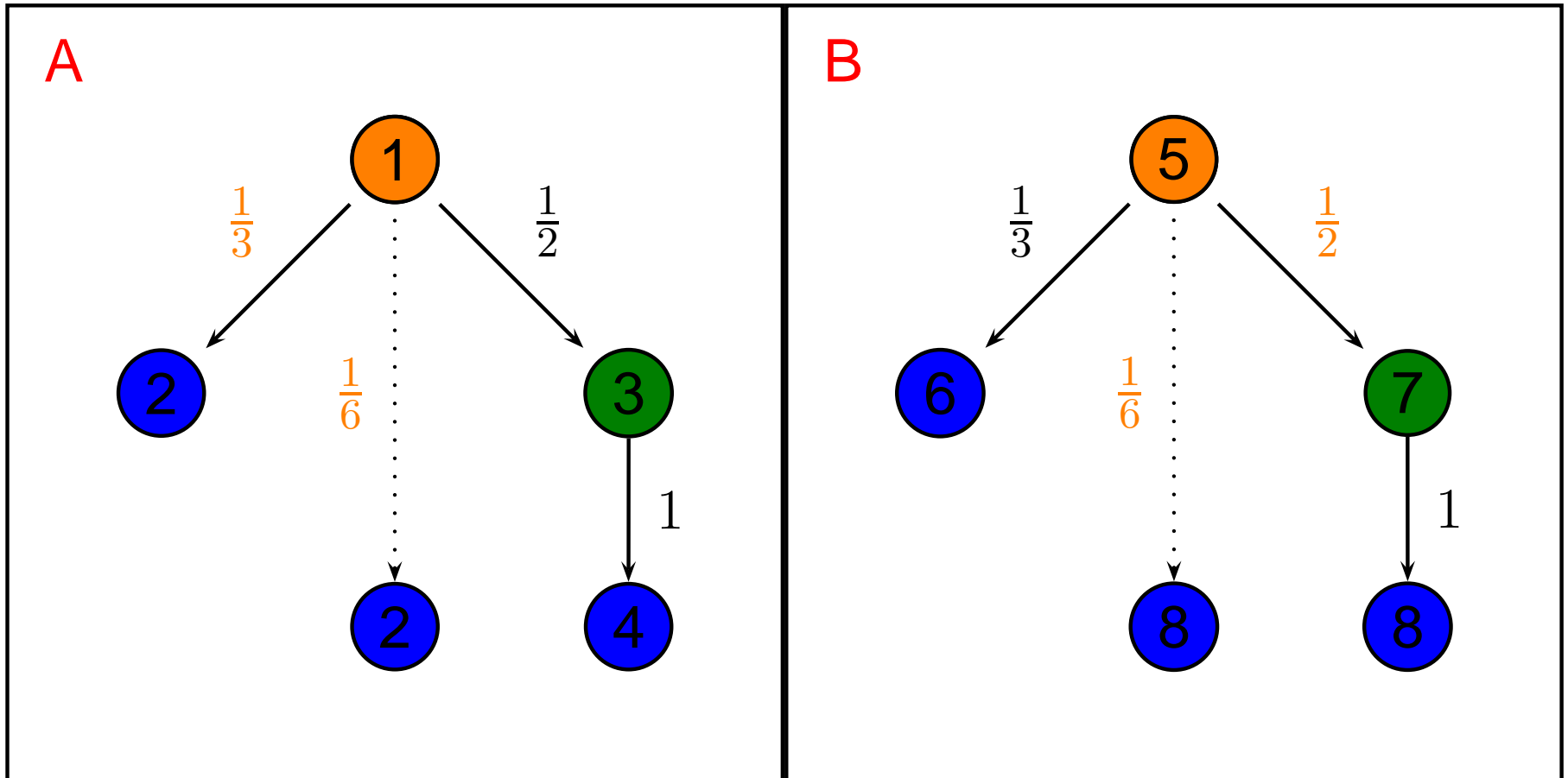
A



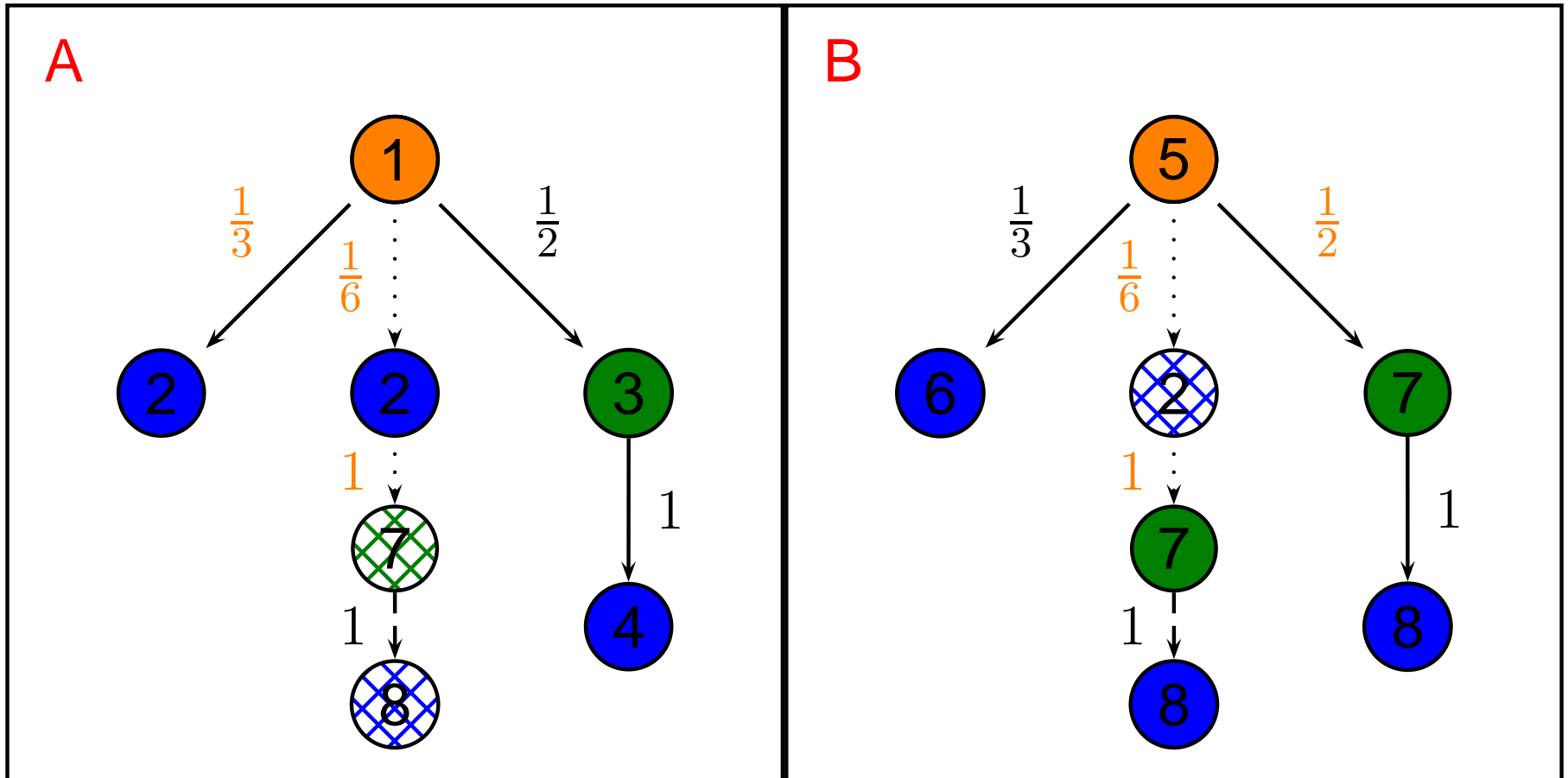
B



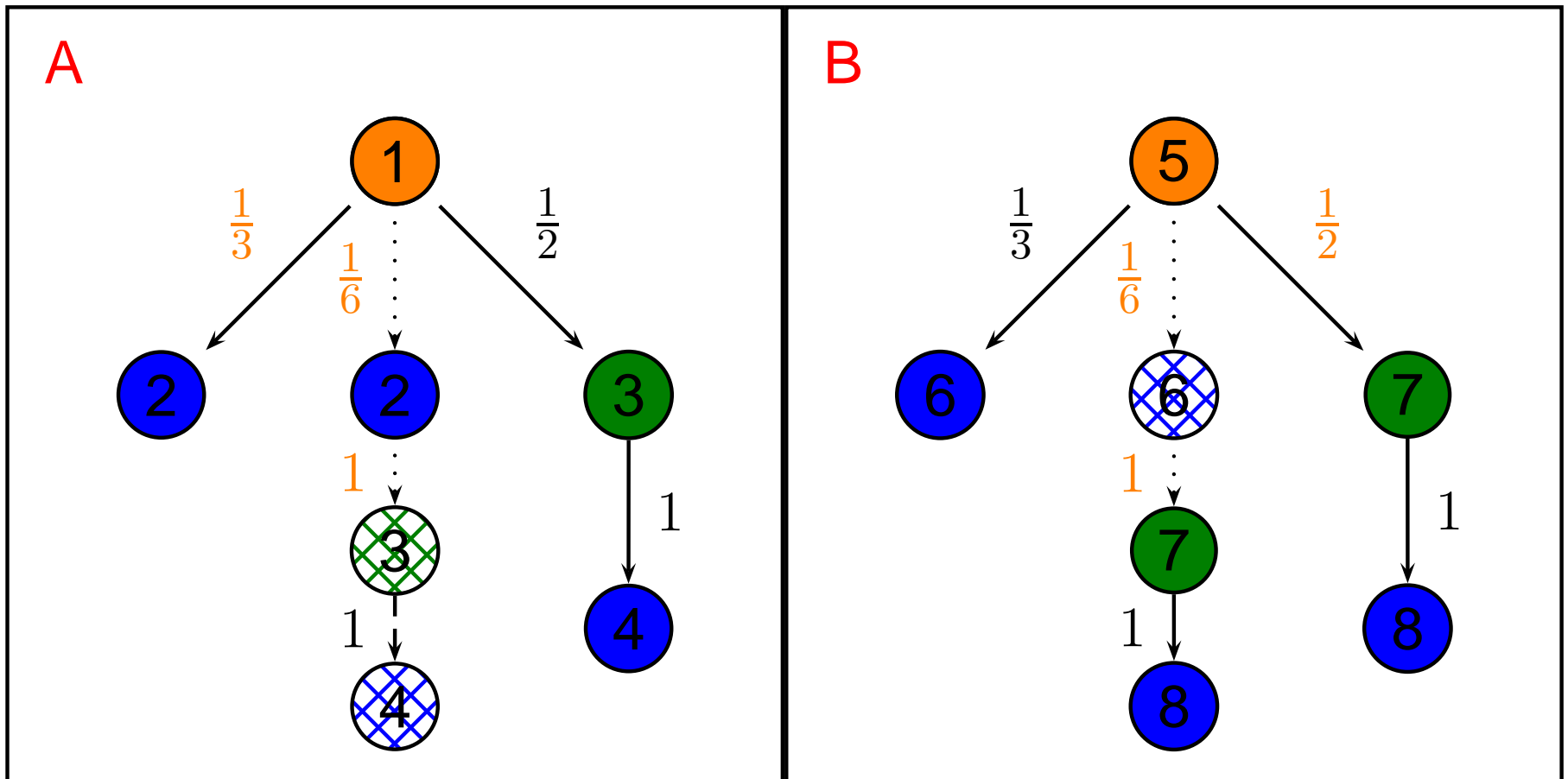
Padding: Redistribution



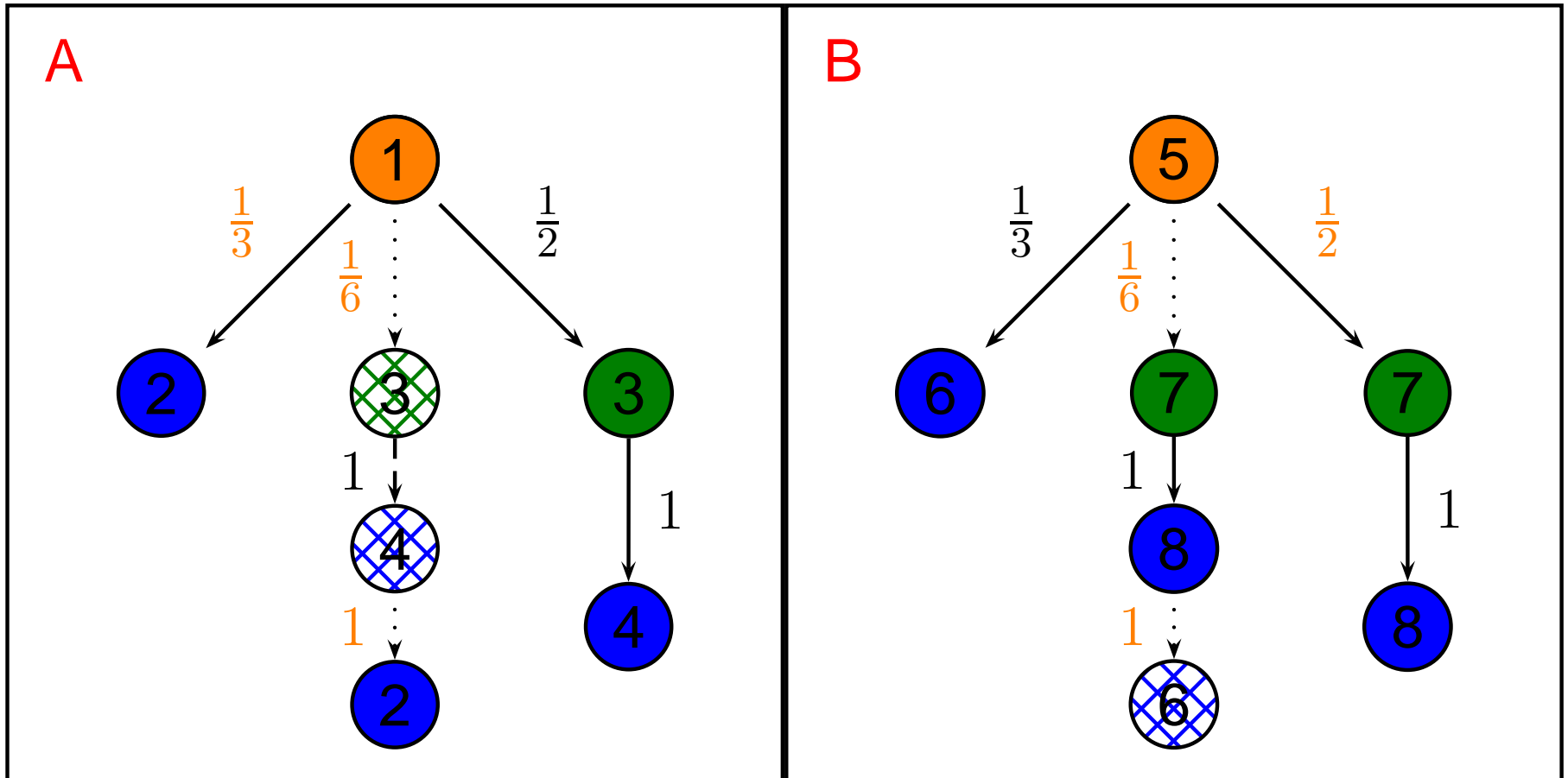
Padding: Redistribution



Padding: Redistribution

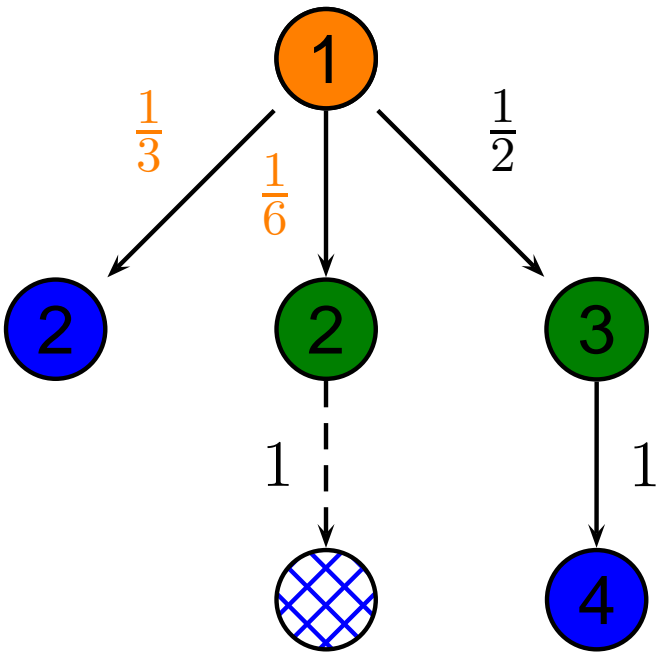


Padding: Redistribution

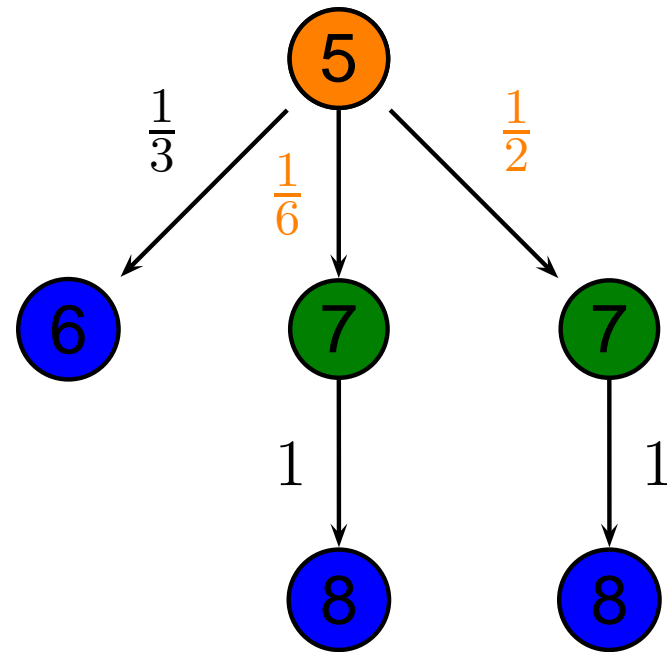


Padding: Redistribution

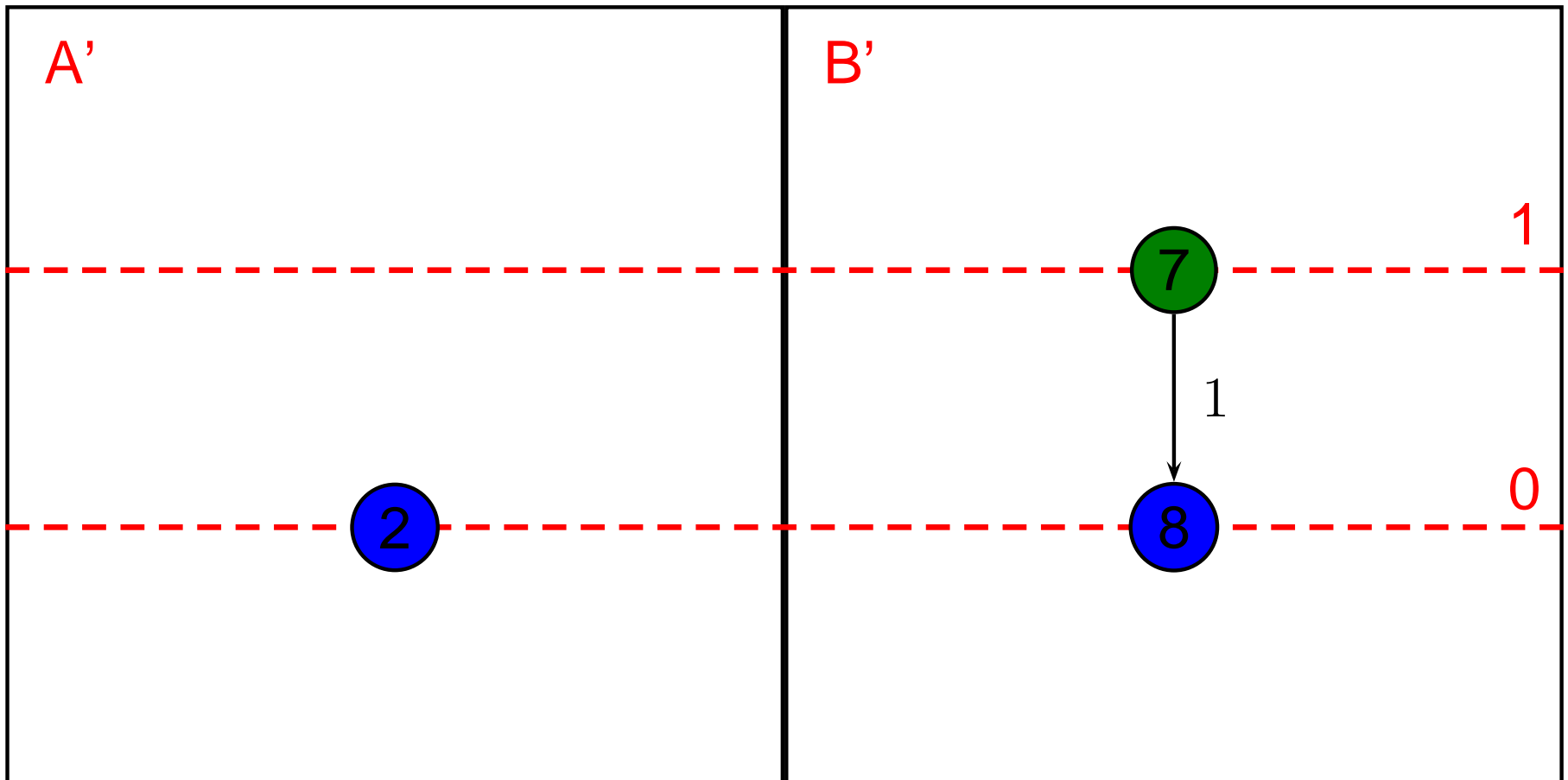
A



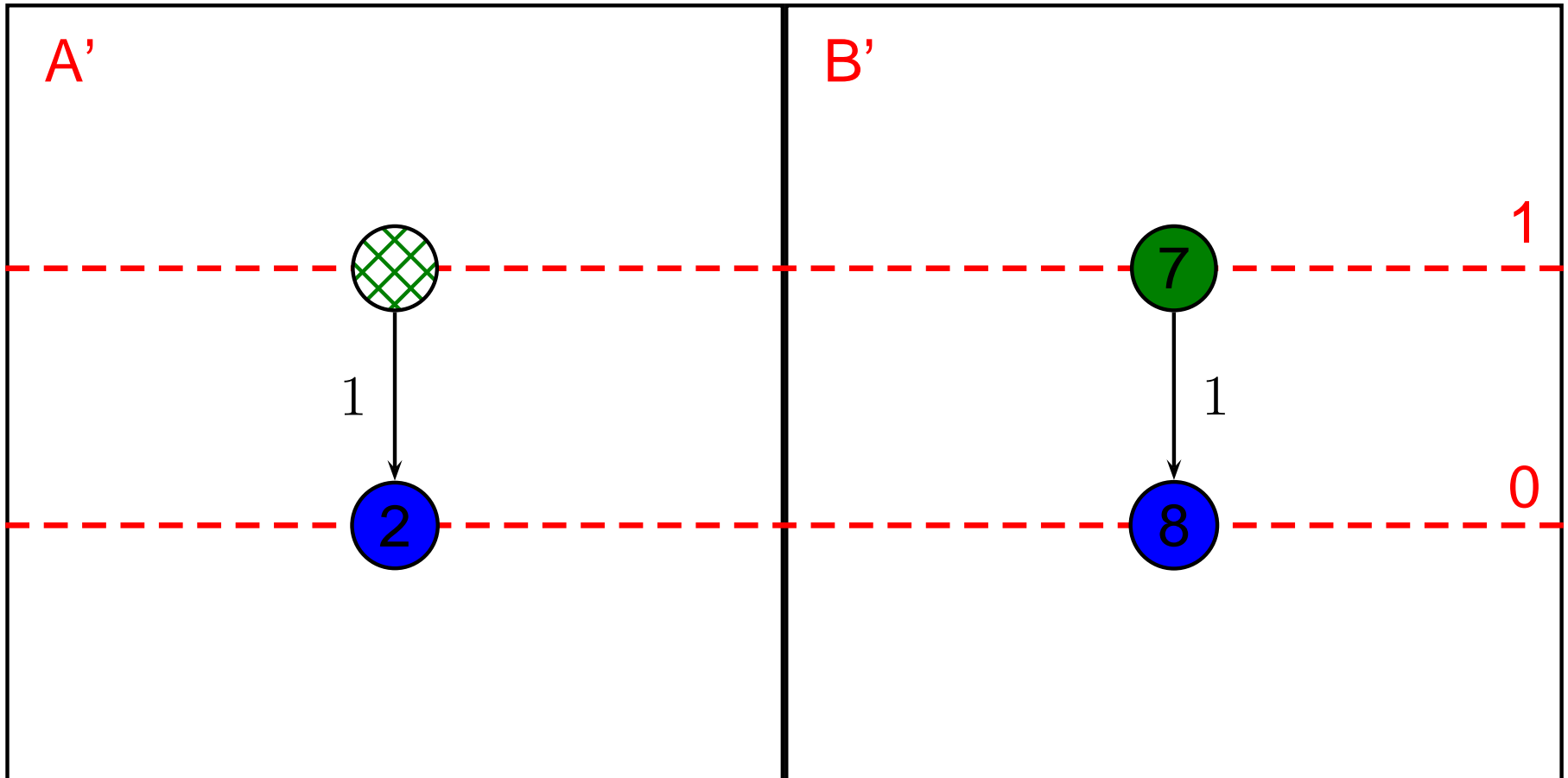
B



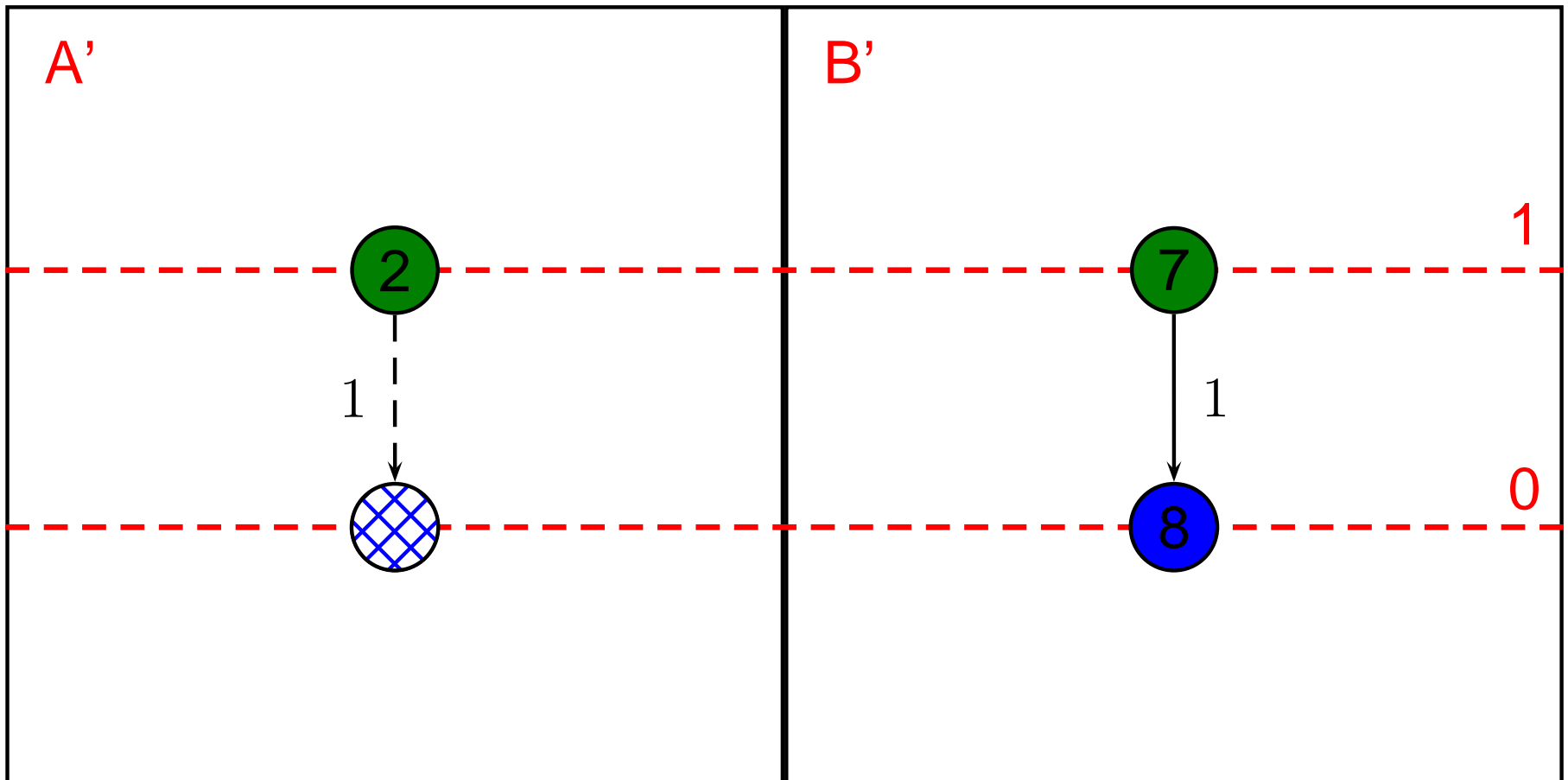
Sub-Padding: Extension



Sub-Padding: Extension

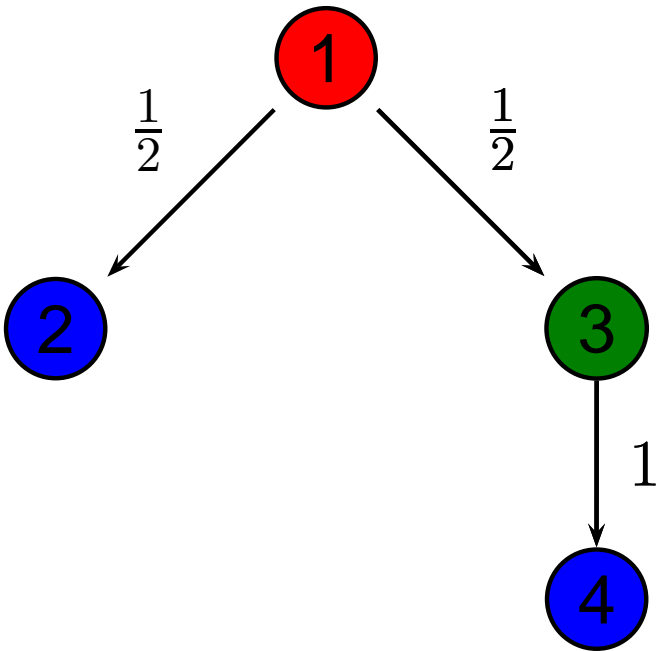


Sub-Padding: Extension

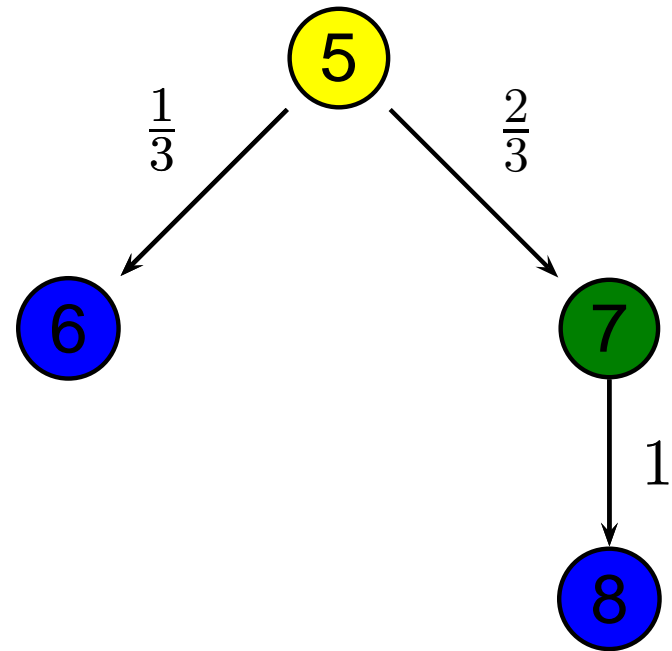


Probabilistic Padding

A

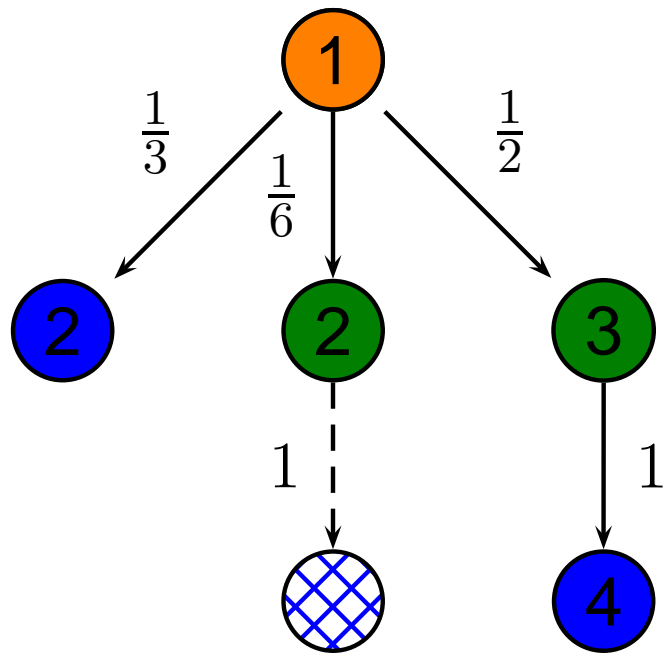


B

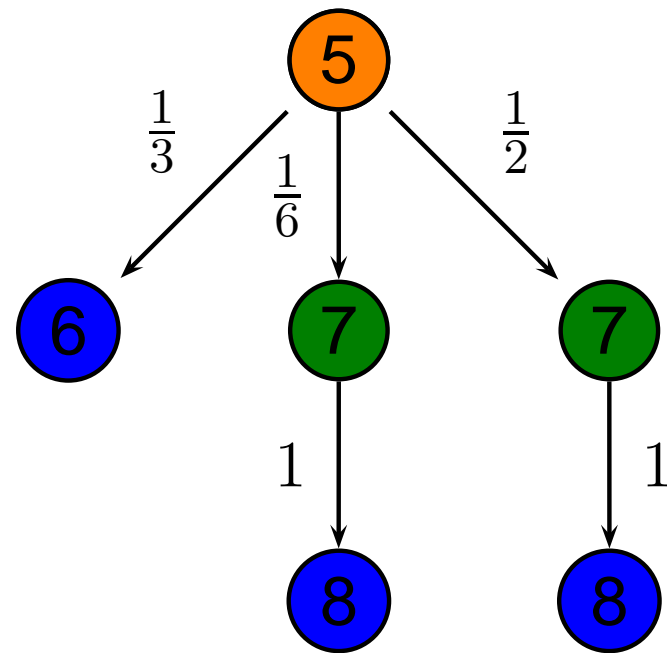


Probabilistic Padding

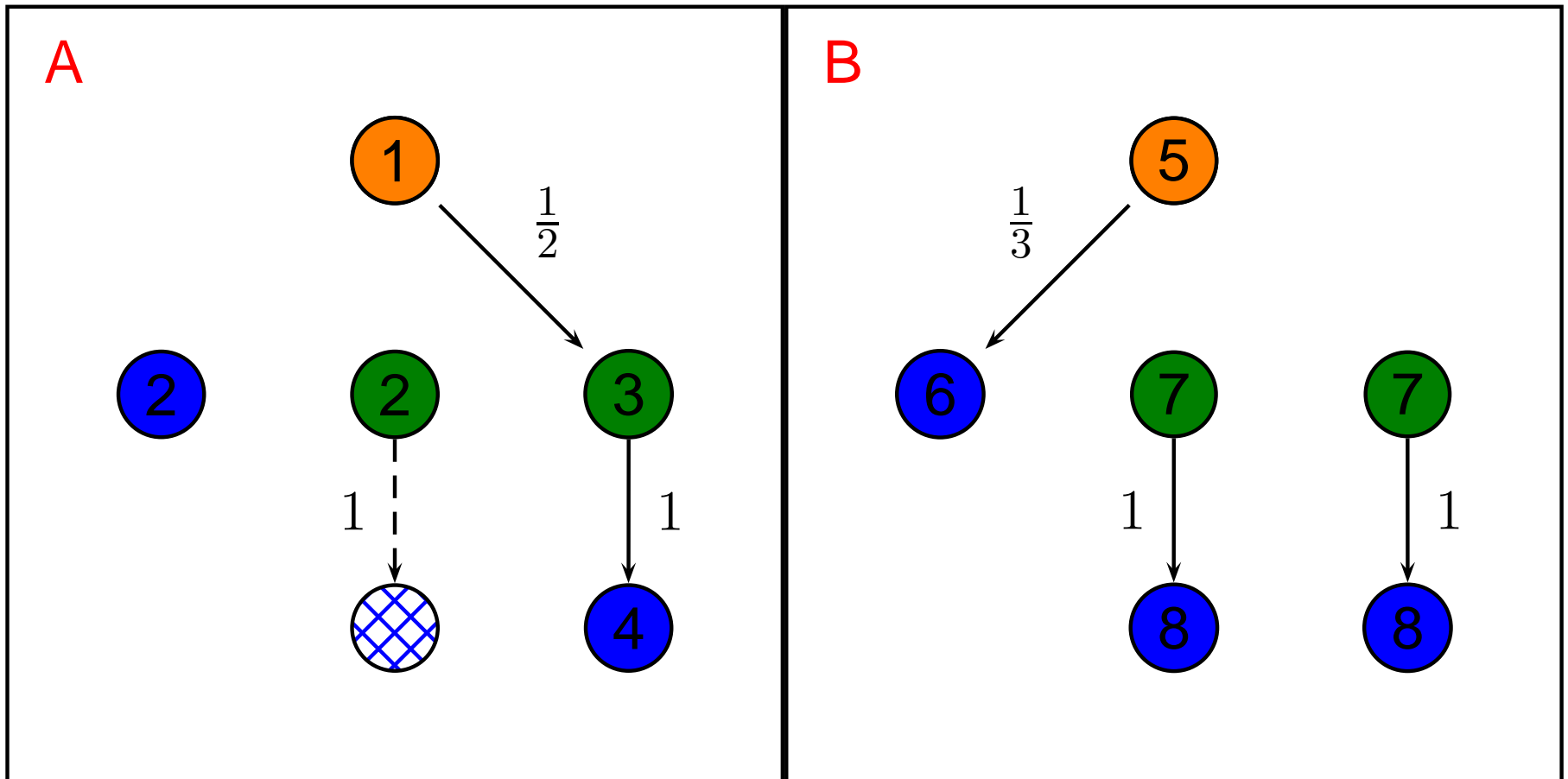
A



B

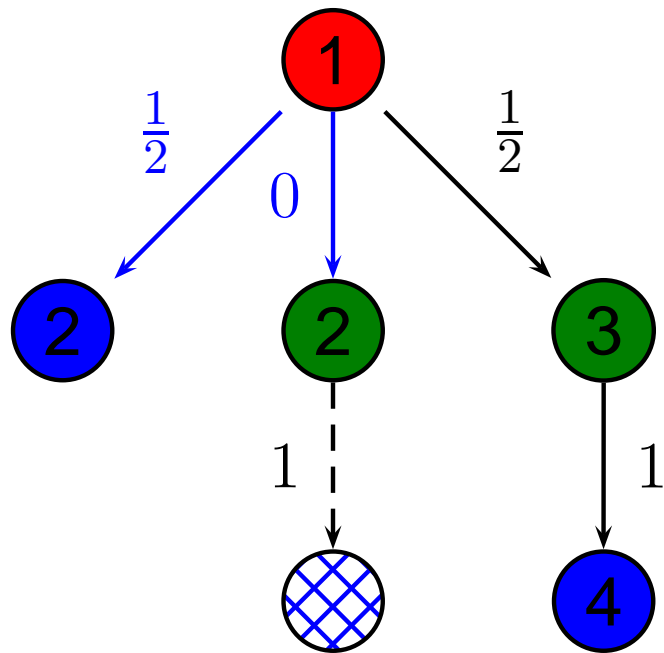


Probabilistic Padding

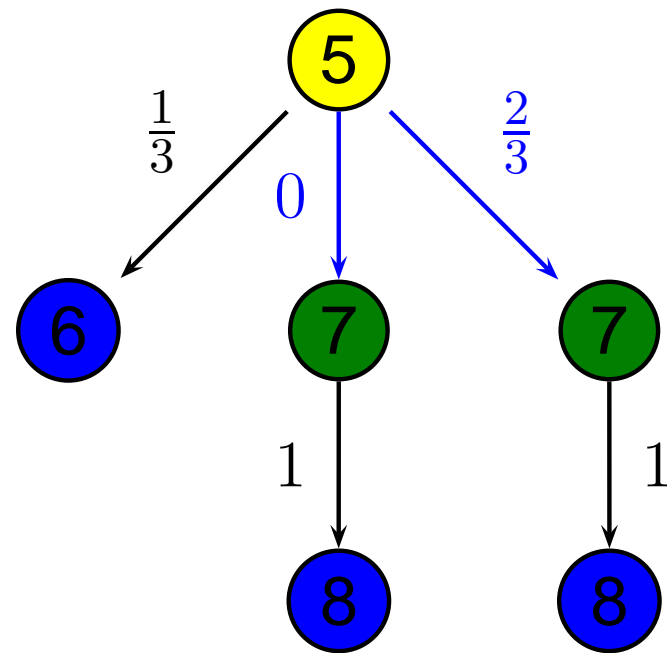


Probabilistic Padding

A

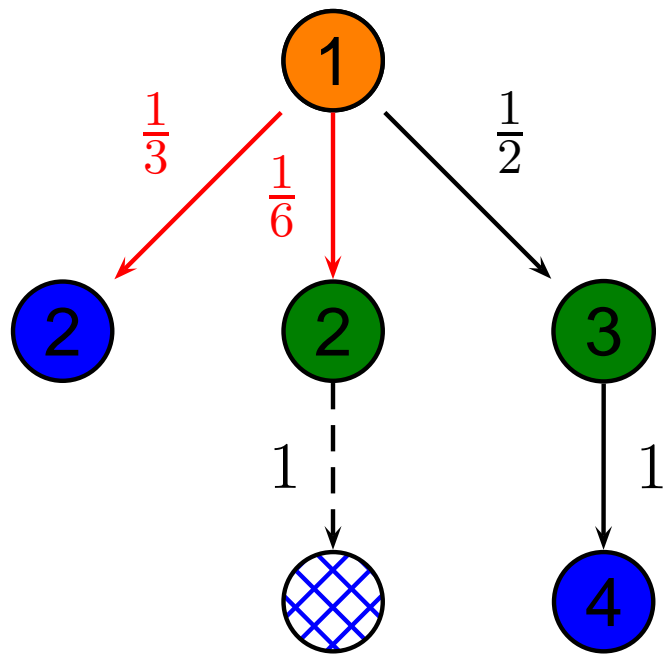


B

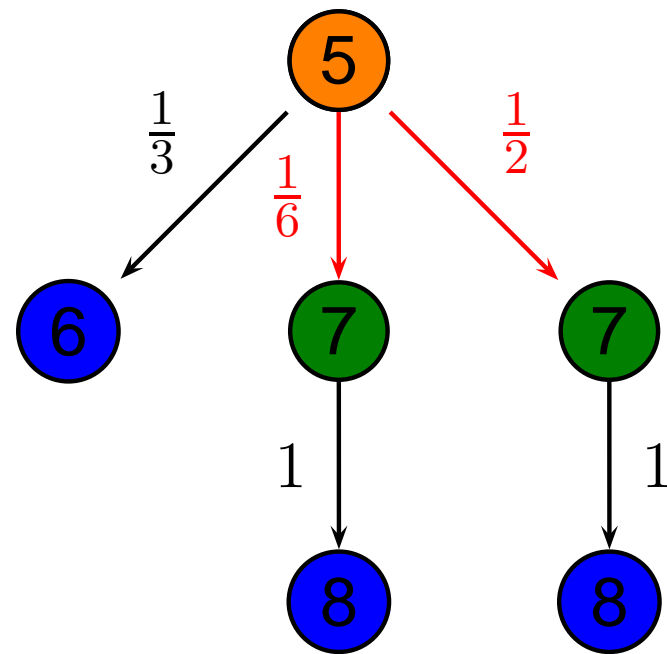


Probabilistic Padding

A

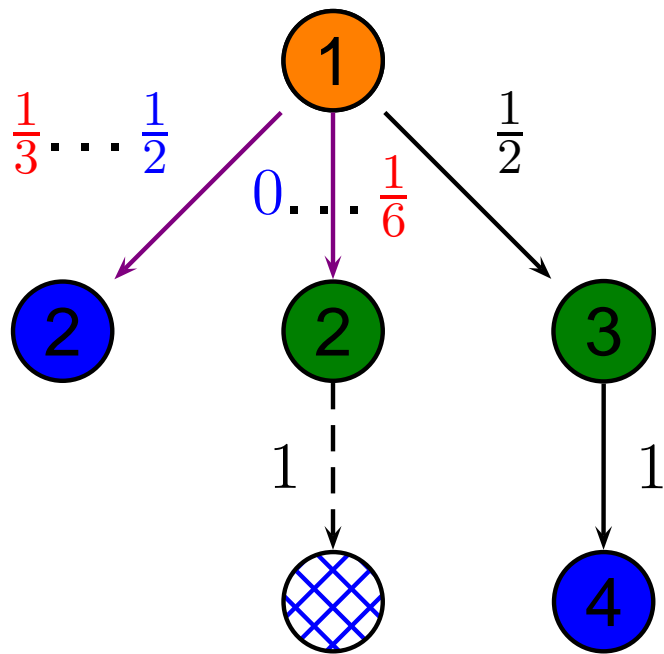


B

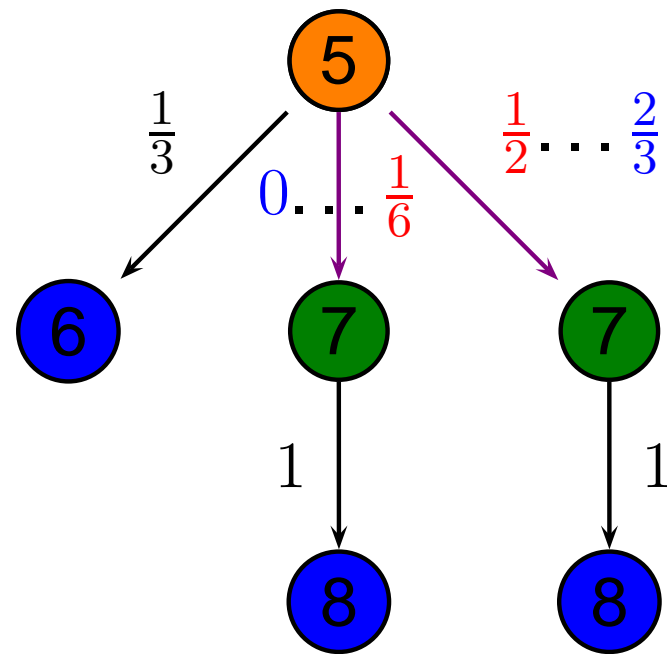


Probabilistic Padding

A

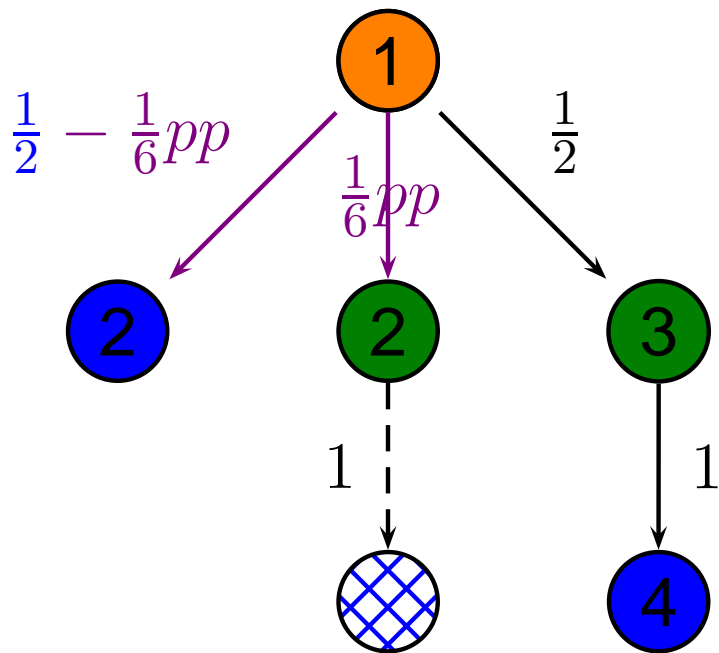


B

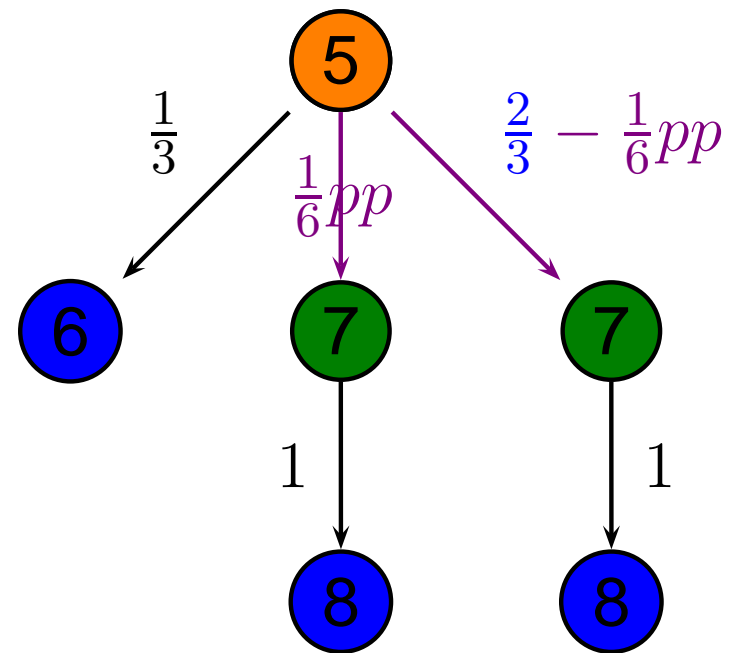


Probabilistic Padding

A



B



Correctness

Theorem

Given rooted PTS's T_1 and T_2 then $\text{PADDING}(T_1, T_2)$ with $pp = 1$ returns rooted PTS's T'_1 and T'_2 such that $T'_1 \sim_b T'_2$, $T'_1 \sim_{io} T_1$, and $T'_2 \sim_{io} T_2$.

Correctness

Theorem

Given rooted PTS's T_1 and T_2 then $\text{PADDING}(T_1, T_2)$ with $pp = 1$ returns rooted PTS's T'_1 and T'_2 such that $T'_1 \sim_b T'_2$, $T'_1 \sim_{io} T_1$, and $T'_2 \sim_{io} T_2$.

Proposition

For all $1 \leq n \leq \text{HEIGHT}(T_1 \oplus T_2)$ the partition (of the n layer cut off) constructed by procedure $\text{MAXLUMPING}(T_1, T_2)$ is stable.

Trading Security for Additional Costs

The transformation algorithm introduces time leak fixes only with a certain probability whenever they appear to be necessary. The resulting processes are therefore not always **perfectly secure**.

Trading Security for Additional Costs

The transformation algorithm introduces time leak fixes only with a certain probability whenever they appear to be necessary. The resulting processes are therefore not always **perfectly secure**.

Provide an **estimate** of their **vulnerability**.

Trading Security for Additional Costs

The transformation algorithm introduces time leak fixes only with a certain probability whenever they appear to be necessary. The resulting processes are therefore not always **perfectly secure**.

Provide an **estimate** of their **vulnerability**.

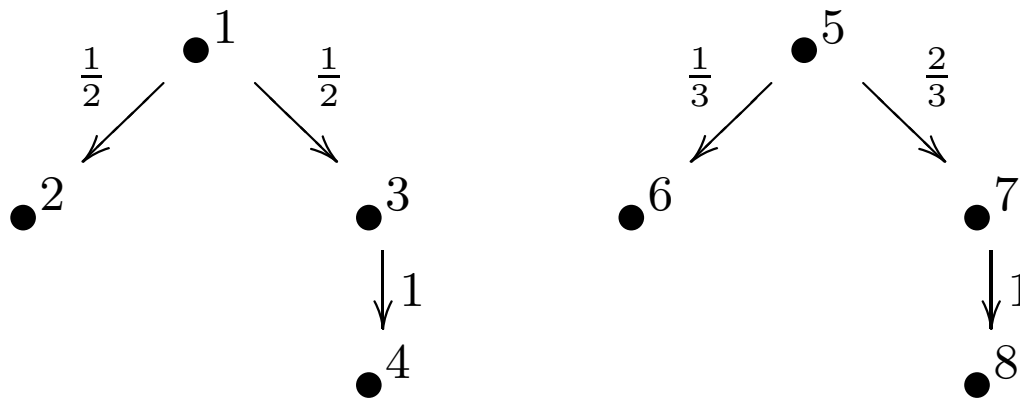
Optimise the trade-off between vulnerability against timing attacks and **additional costs** such as running time, system size, etc.

Example

Consider the following two processes A and B

Example

Consider the following two processes A and B



Example

Consider the following two processes A and B

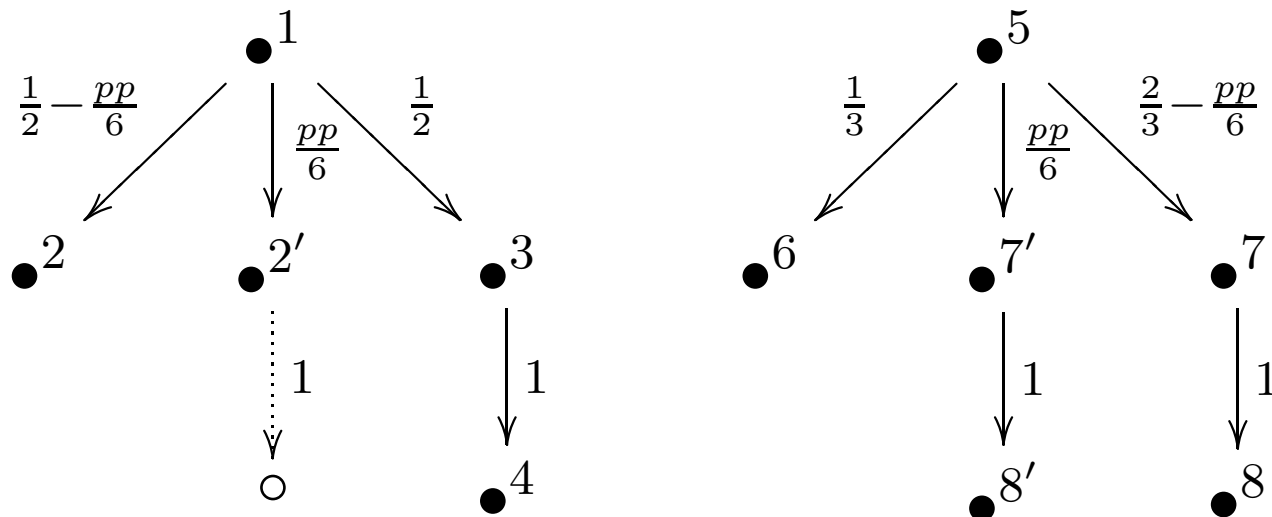
$$\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Example ctd.

Our algorithm produces the probabilistic padded versions of A and B :

Example ctd.

Our algorithm produces the probabilistic padded versions of A and B :



Example ctd.

Our algorithm produces the probabilistic padded versions of A and B :

$$\mathbf{A}(pp) = \begin{pmatrix} 0 & \frac{1}{2} & -\frac{pp}{6} & \frac{1}{2} & 0 & \frac{pp}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example ctd.

Our algorithm produces the probabilistic padded versions of A and B :

$$\mathbf{B}(pp) = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} & -\frac{pp}{6} & 0 & \frac{pp}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example ctd.

Our algorithm produces the probabilistic padded versions of A and B :

$$\mathbf{A}(1) = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{B}(1) = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Costs estimates

We can compute a measure of vulnerability by:

$$\varepsilon(pp) = \|\mathbf{K}_A^\dagger \mathbf{A}(pp) \mathbf{K}_A - \mathbf{K}_B^\dagger \mathbf{B}(pp) \mathbf{K}_B\|.$$

Costs estimates

We can compute a measure of vulnerability by:

$$\varepsilon(pp) = \|\mathbf{K}_A^\dagger \mathbf{A}(pp) \mathbf{K}_A - \mathbf{K}_B^\dagger \mathbf{B}(pp) \mathbf{K}_B\|.$$

We can also compute the change in the average running time, $\Delta_A(pp)$ and $\Delta_B(pp)$, of $A(pp)$ and $B(pp)$ wrt $A(0) = A$ and $B(0) = B$.

Costs estimates

We can compute a measure of vulnerability by:

$$\varepsilon(pp) = \|\mathbf{K}_A^\dagger \mathbf{A}(pp) \mathbf{K}_A - \mathbf{K}_B^\dagger \mathbf{B}(pp) \mathbf{K}_B\|.$$

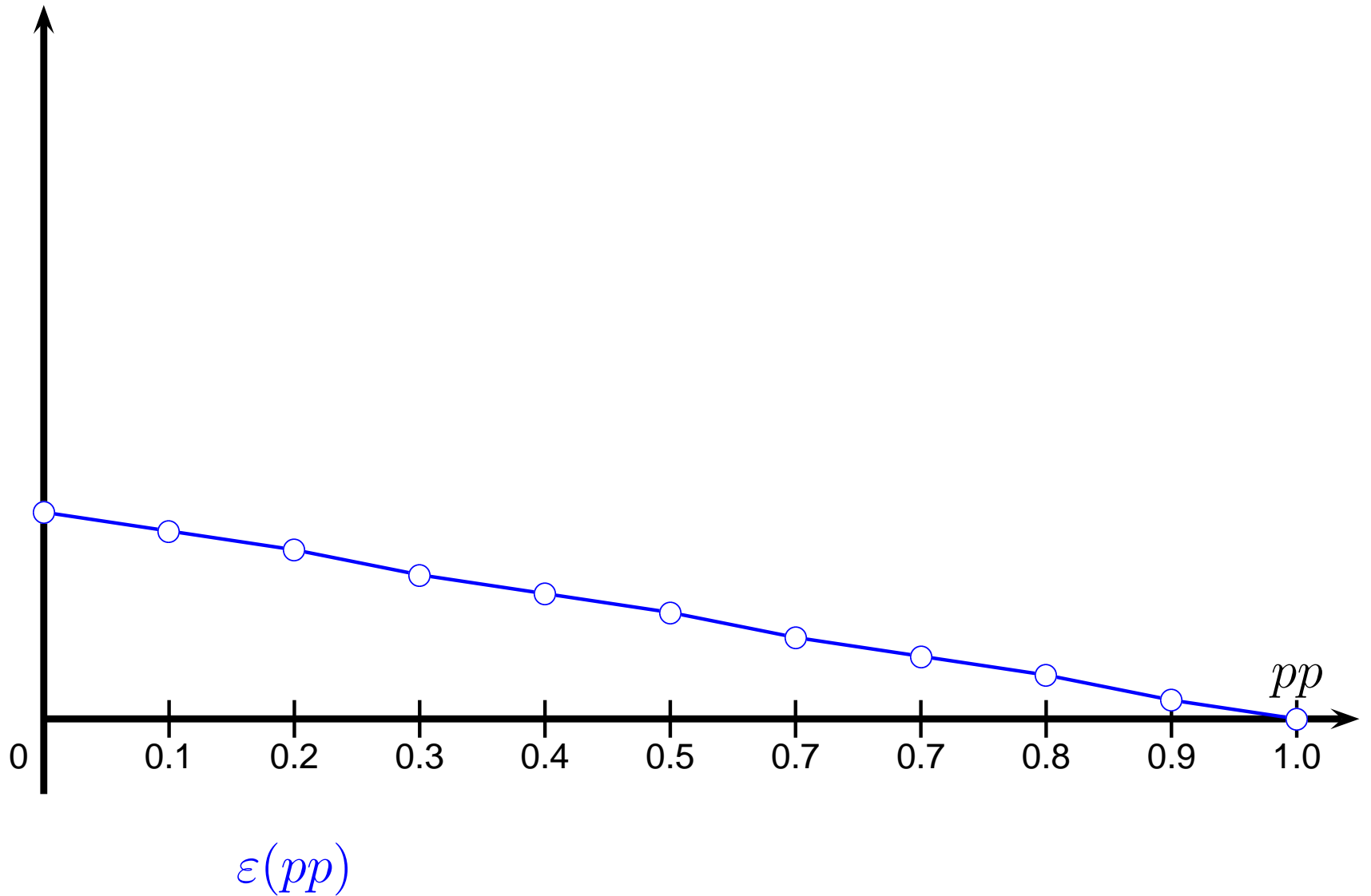
We can also compute the change in the average running time, $\Delta_A(pp)$ and $\Delta_B(pp)$, of $A(pp)$ and $B(pp)$ wrt $A(0) = A$ and $B(0) = B$.

The tradeoff between these two costs can be analysed by defining an objective function, such as e.g.

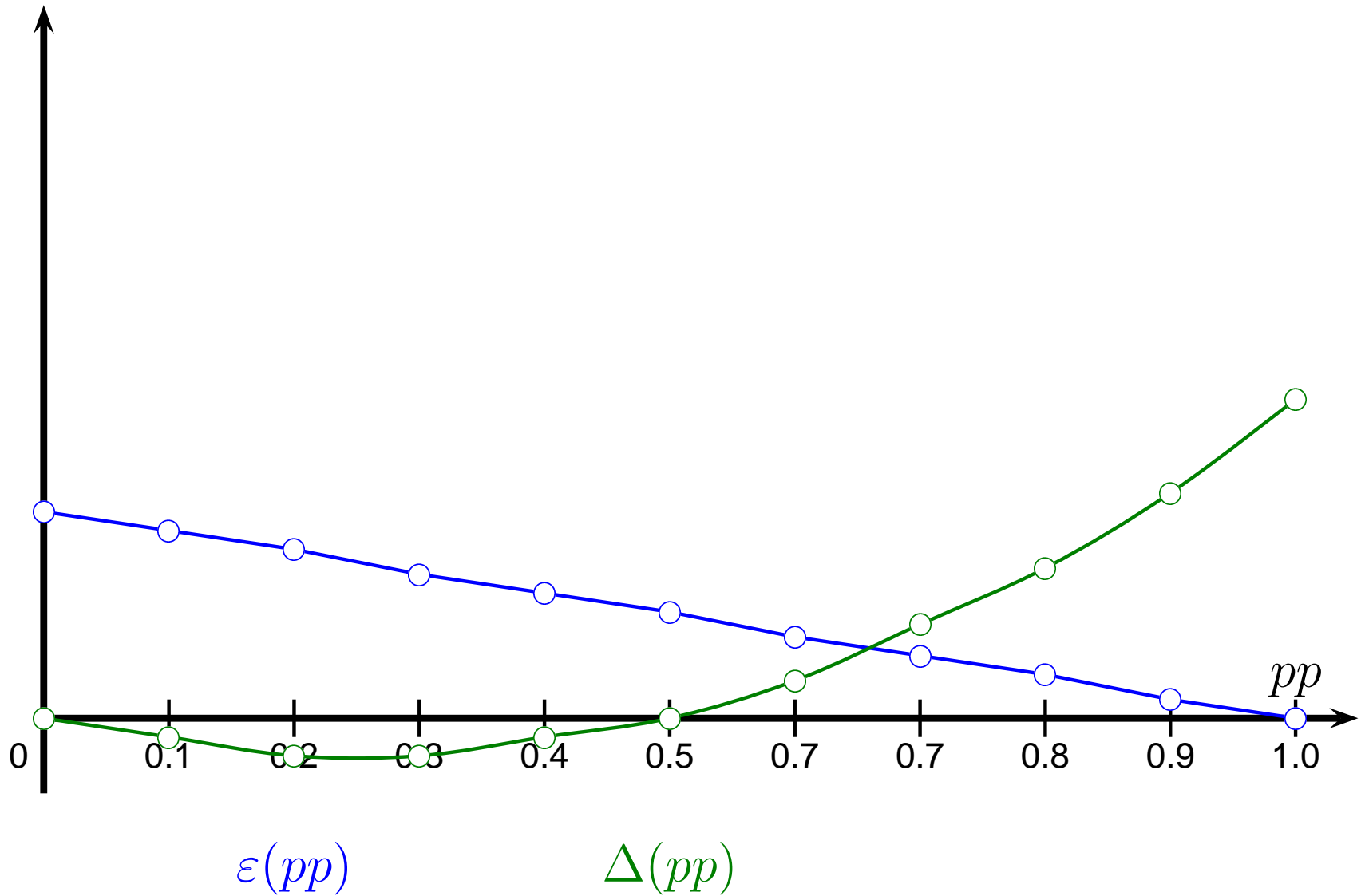
$$f(pp) = \varepsilon(pp) + 100 \cdot \Delta(pp),$$

with $\Delta(pp) = \Delta_A(pp) + \Delta_B(pp)$.

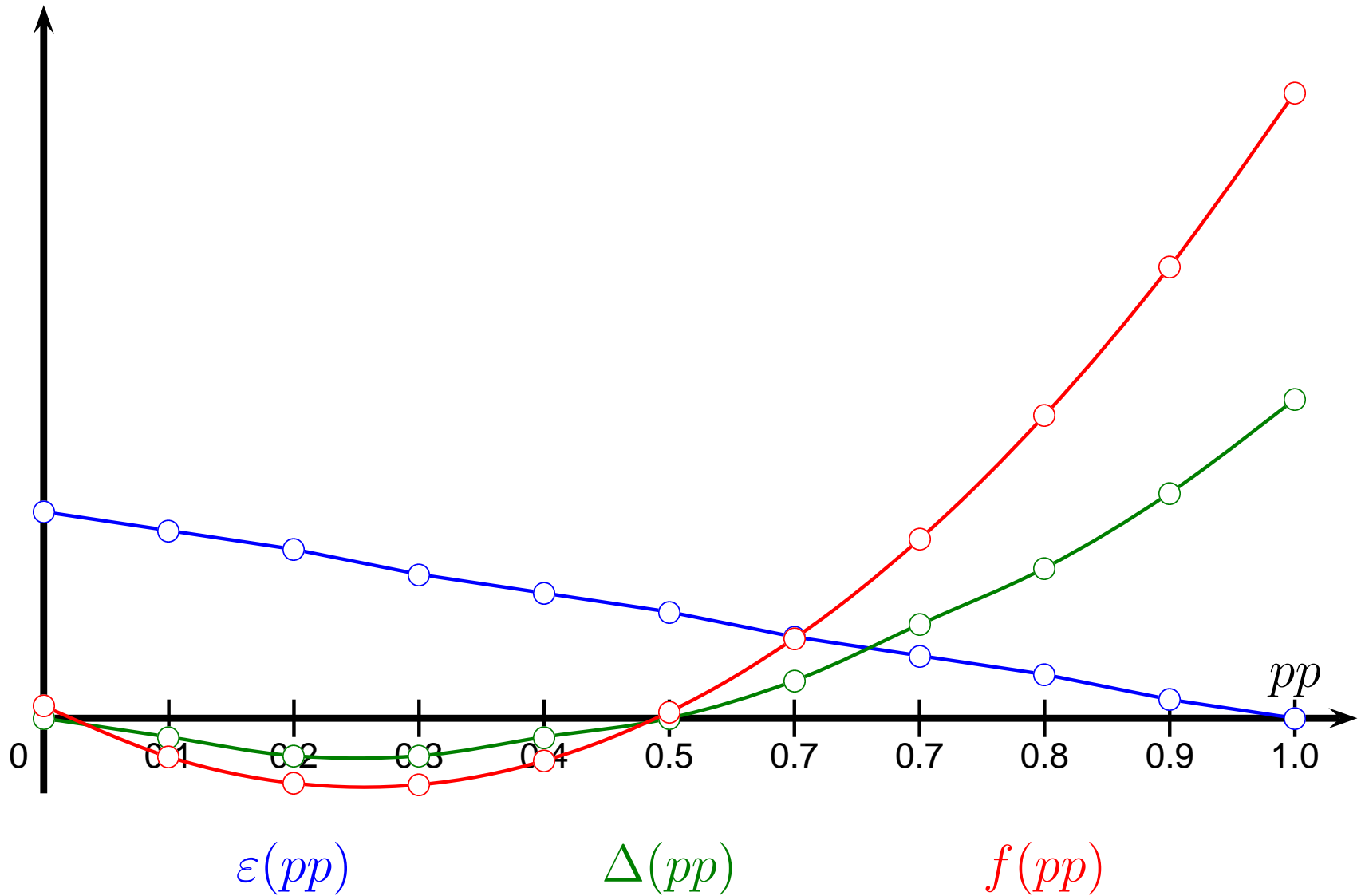
Optimisation of Performance



Optimisation of Performance



Optimisation of Performance



Conclusion/Further Work

- Countermeasures against timing attacks.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.
- Attempt to minimise the running time overhead.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.
- Attempt to minimise the running time overhead.
- Apply patches randomly.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.
- Attempt to minimise the running time overhead.
- Apply patches randomly.

- Decision theoretic analysis.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.
- Attempt to minimise the running time overhead.
- Apply patches randomly.

- Decision theoretic analysis.
Padding only if the expected time leak exceeds a certain threshold.

Conclusion/Further Work

- Countermeasures against timing attacks.
 - Process algebraic setting.
 - Attempt to minimise the running time overhead.
 - Apply patches randomly.
-
- Decision theoretic analysis.
 - Estimation and closed expressions for ε and $\varepsilon(pp)$.

Conclusion/Further Work

- Countermeasures against timing attacks.
 - Process algebraic setting.
 - Attempt to minimise the running time overhead.
 - Apply patches randomly.
-
- Decision theoretic analysis.
 - Estimation and closed expressions for ε and $\varepsilon(pp)$.
 - Non-linear Optimisation.

Conclusion/Further Work

- Countermeasures against timing attacks.
- Process algebraic setting.
- Attempt to minimise the running time overhead.
- Apply patches randomly.

- Decision theoretic analysis.
- Estimation and closed expressions for ε and $\varepsilon(pp)$.
- Non-linear Optimisation.
-