Make flows small again: revisiting the flow framework

Roland Meyer[1], Thomas Wies[2], Sebastian Wolff[2]

[1] TU Braunschweig, Germany
[2] New York University, USA

[TACAS'23]

Frame Rule

$$\frac{\{\ P\ \}\ com\ \{\ Q\ \}}{\{\ P * F\ \}\ com\ \{\ Q * F\ \}}$$

Frame Rule

$$\frac{\{\ P\ \}\ com\ \{\ Q\ \}}{\{\ P * F\ \}\ com\ \{\ Q * F\ \}}$$

Usage

$$\frac{\{\ x \mapsto 5\ \}\ [x] = 7\ \{\ x \mapsto 7\ \}}{\{\ x \mapsto 5 * F\ \}\ [x] = 7\ \{\ x \mapsto 7 * F\ \}}$$

Frame Rule

$$\frac{\{\ P\ \}\ com\ \{\ Q\ \}}{\{\ P * F\ \}\ com\ \{\ Q * F\ \}}$$

Usage

$$\frac{\dfrac{(small\ axioms)}{\{\ x \mapsto 5\ \}\ [x] = 7\ \{\ x \mapsto 7\ \}}}{\{\ x \mapsto 5 * F\ \}\ [x] = 7\ \{\ x \mapsto 7 * F\ \}}$$

Frame Rule

$$\frac{\{\ P\ \}\ com\ \{\ Q\ \}}{\{\ P * F\ \}\ com\ \{\ Q * F\ \}}$$

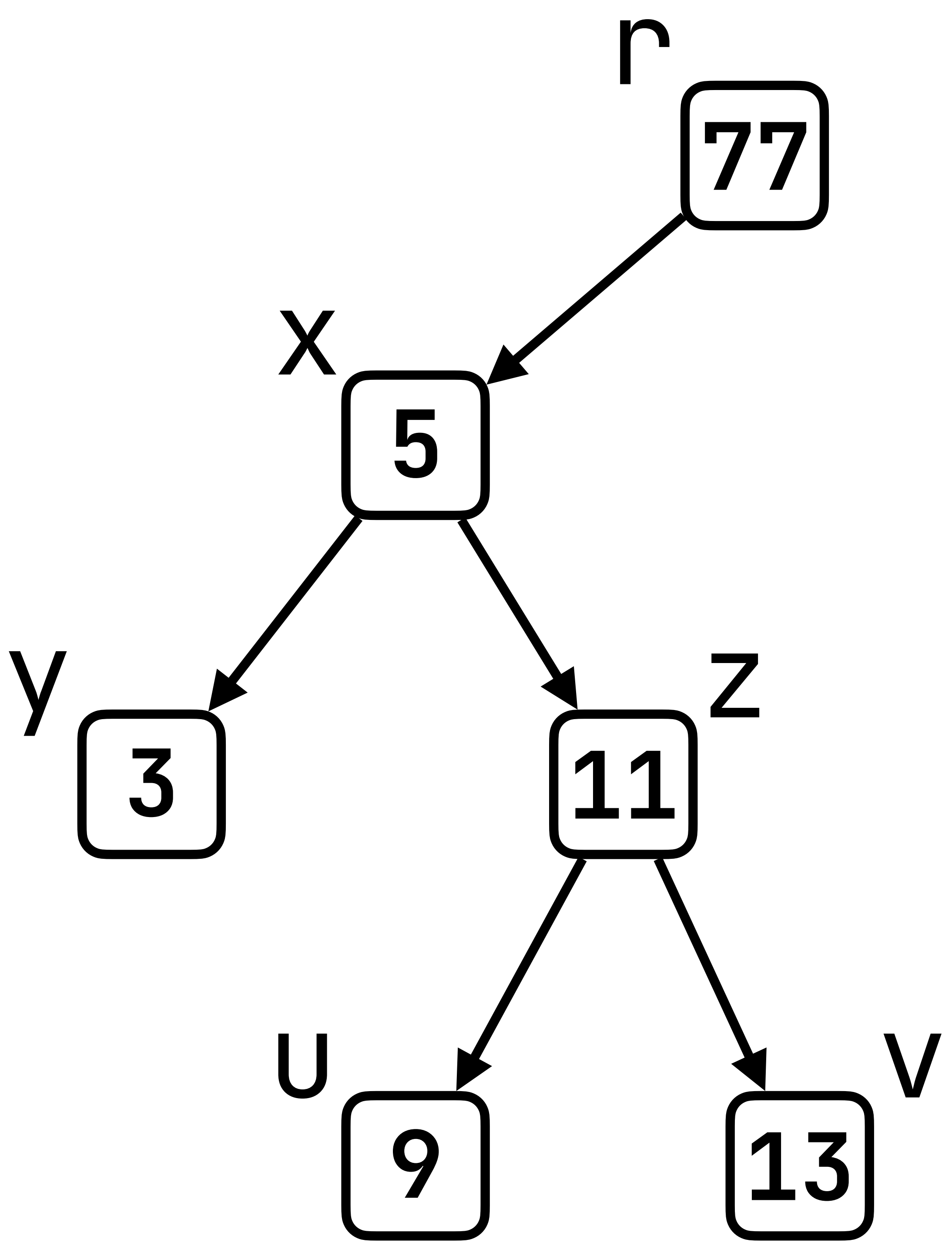
Usage

$$\frac{\dfrac{(small\ axioms)}{\{\ x \mapsto 5\ \}\ [x] = 7\ \{\ x \mapsto 7\ \}}}{\{\ x \mapsto 5 * F\ \}\ [x] = 7\ \{\ x \mapsto 7 * F\ \}}$$

Frame inference is a key challenge for proof automation.

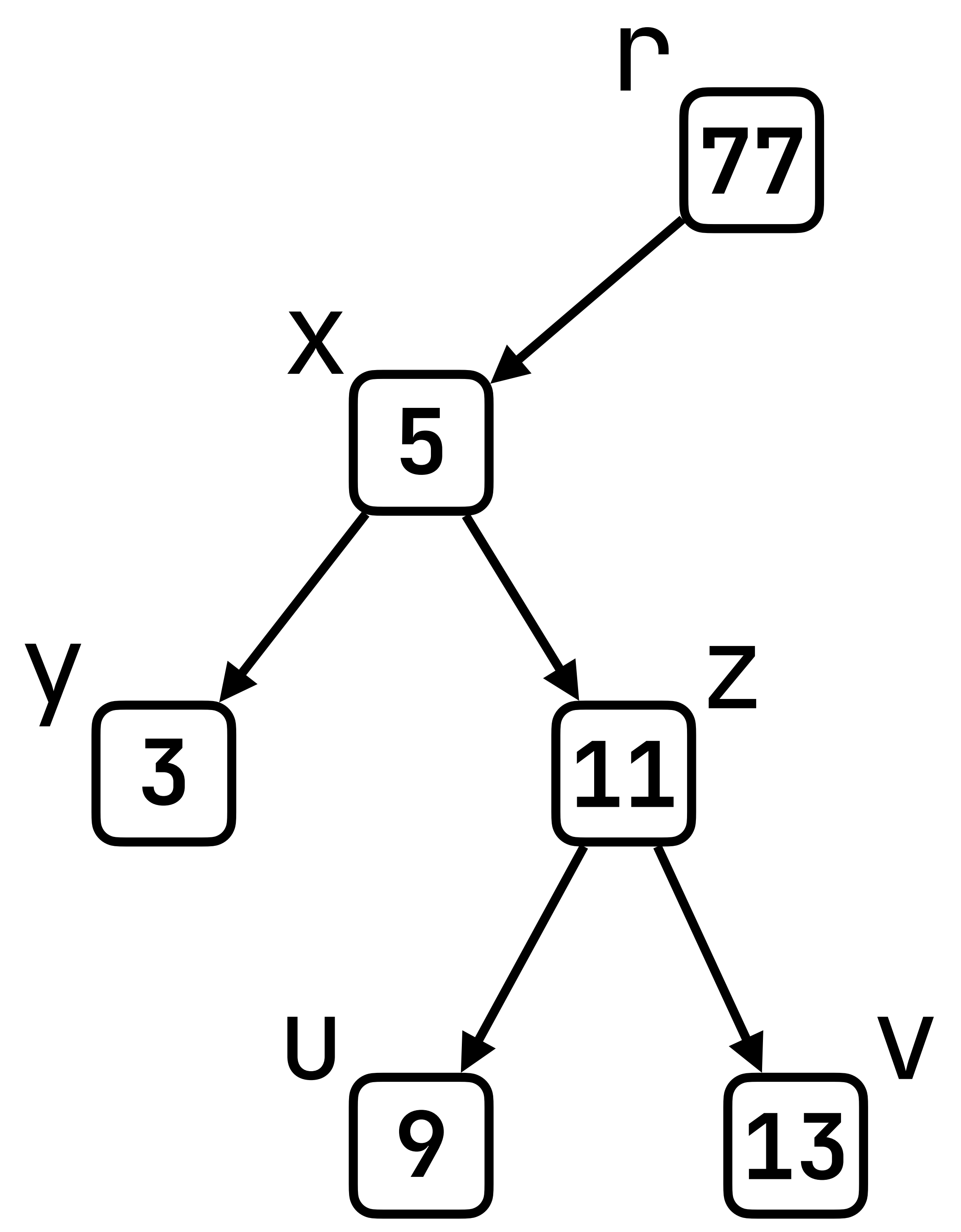
State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \; * \; x \mapsto 5, y, z \; * \; \ldots$



State

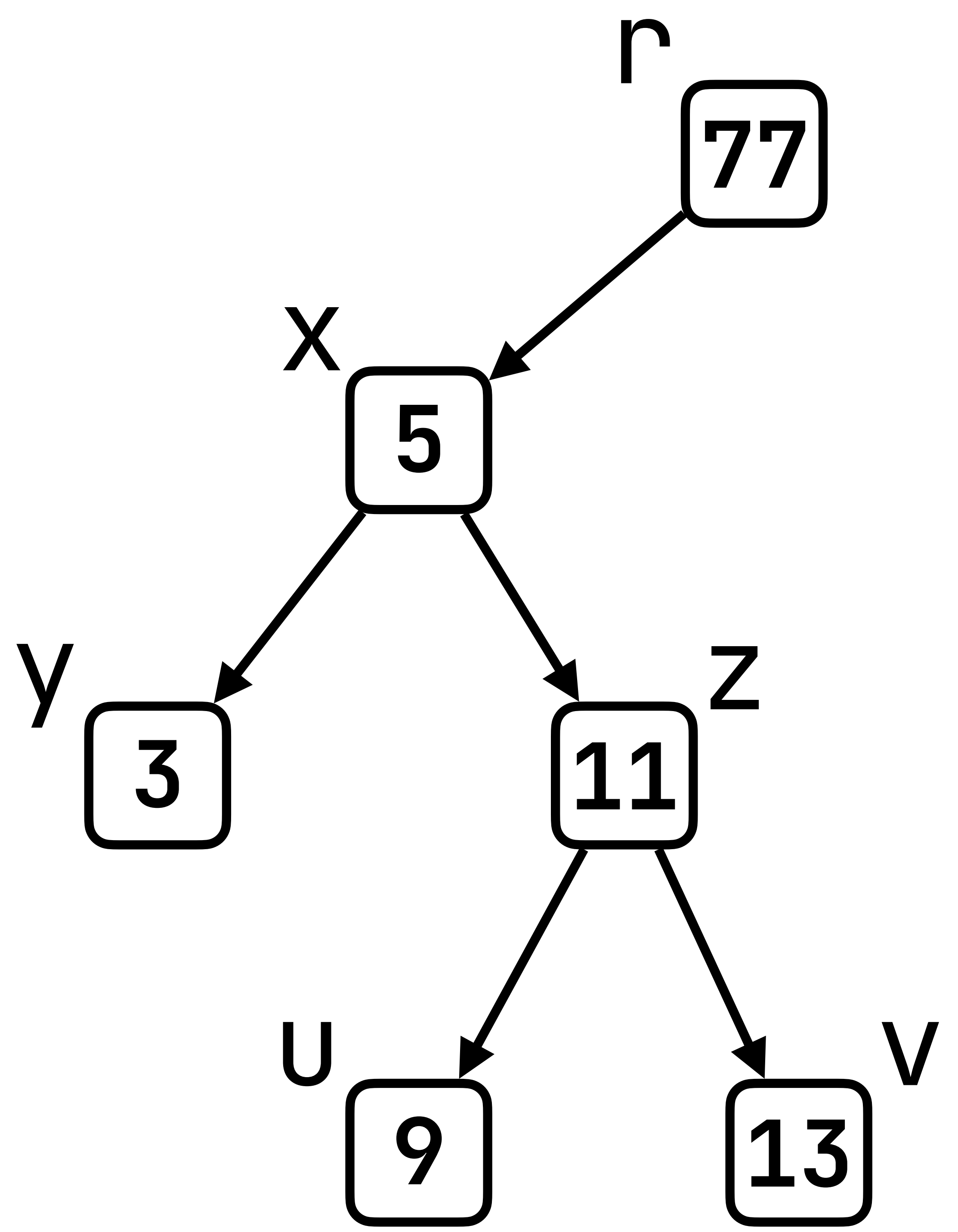
- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \;*\; x \mapsto 5, y, z \;*\; \ldots$
- Ghost state
  - ➡ info for functional correctness



State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \;*\; x \mapsto 5, y, z \;*\; \dots$
- Ghost state
  - ➡ info for functional correctness
  - ➡ e.g. *traversals* (search paths)

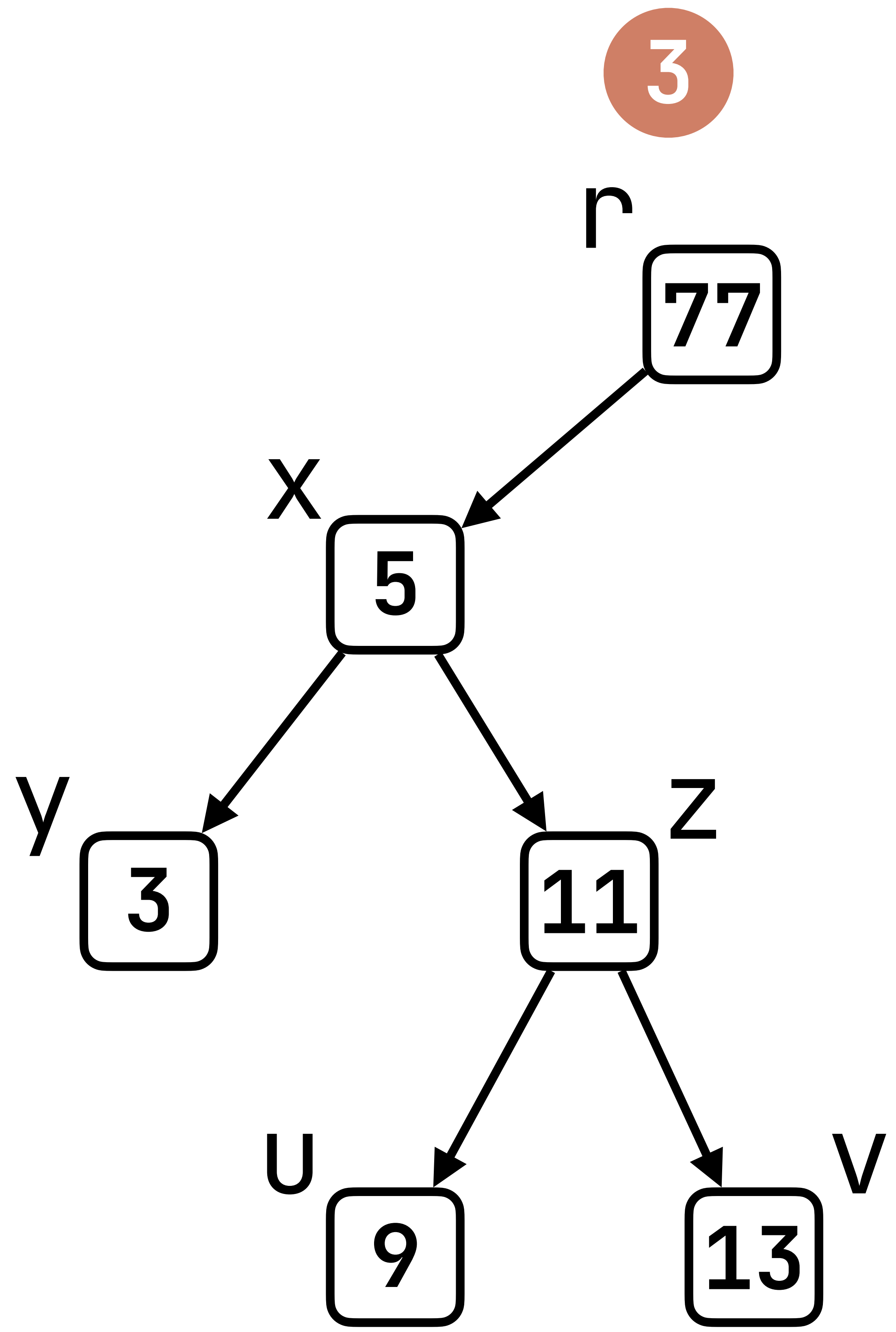
    to show linearizability of data structures



State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \;*\; x \mapsto 5, y, z \;*\; \ldots$
- Ghost state
  - ➡ info for functional correctness
  - ➡ e.g. *traversals* (search paths) 3

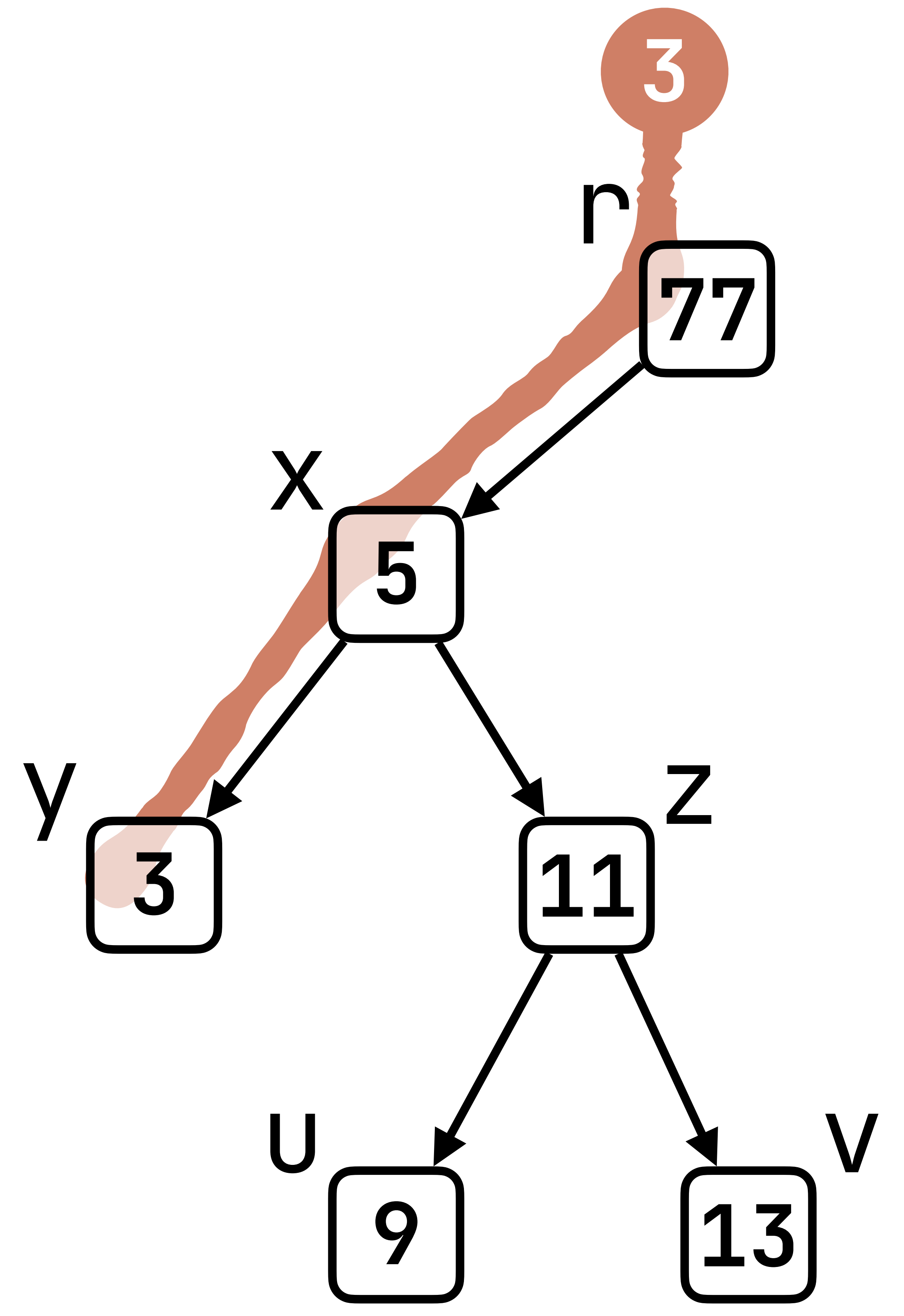
    to show linearizability of data structures



State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \ * \ x \mapsto 5, y, z \ * \ \ldots$
- Ghost state
  - ➡ info for functional correctness
  - ➡ e.g. *traversals* (search paths) 3

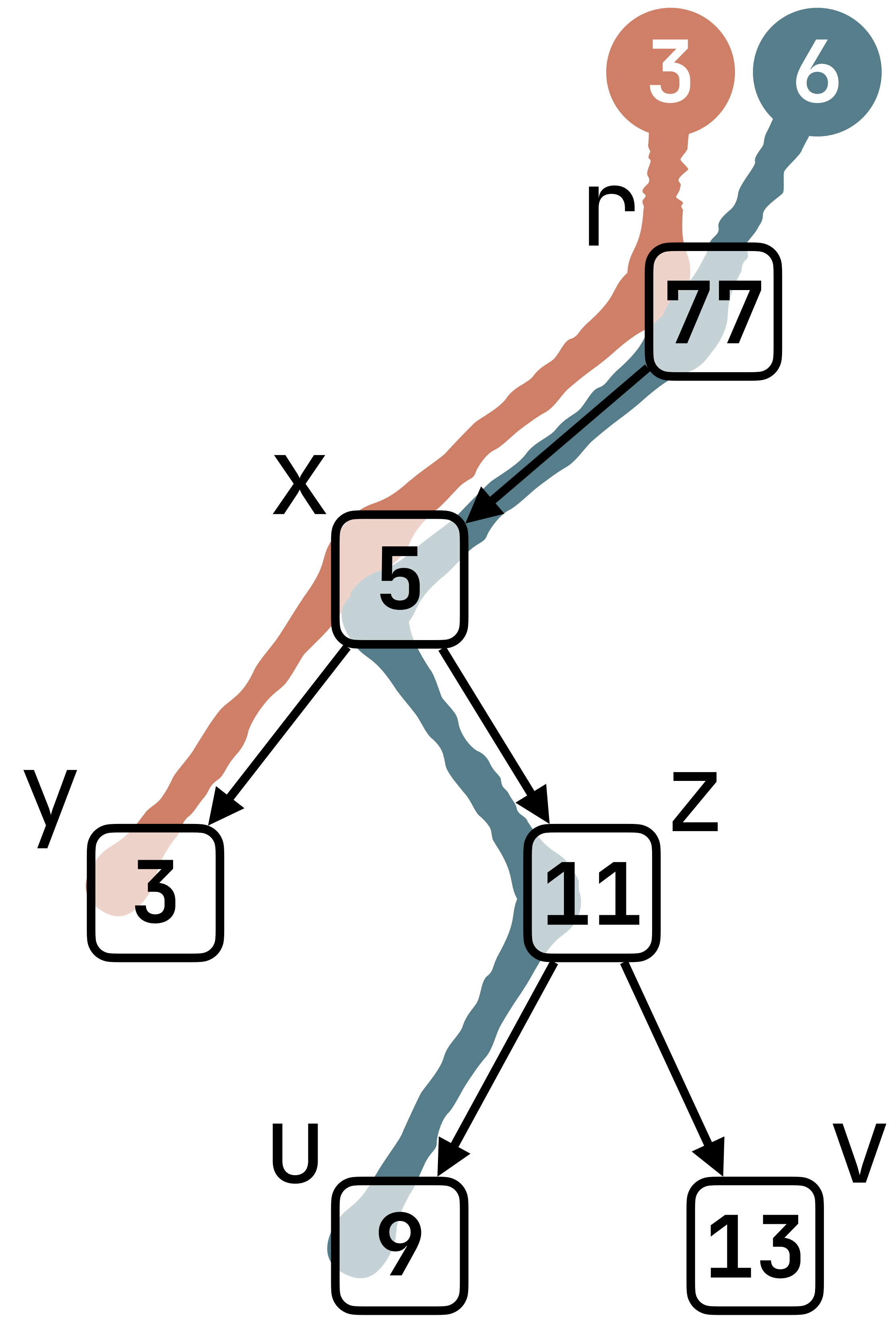
    to show linearizability of data structures



State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \perp \; * \; x \mapsto 5, y, z \; * \; \ldots$
- Ghost state
  - ➡ info for functional correctness
  - ➡ e.g. *traversals* (search paths) 3 6

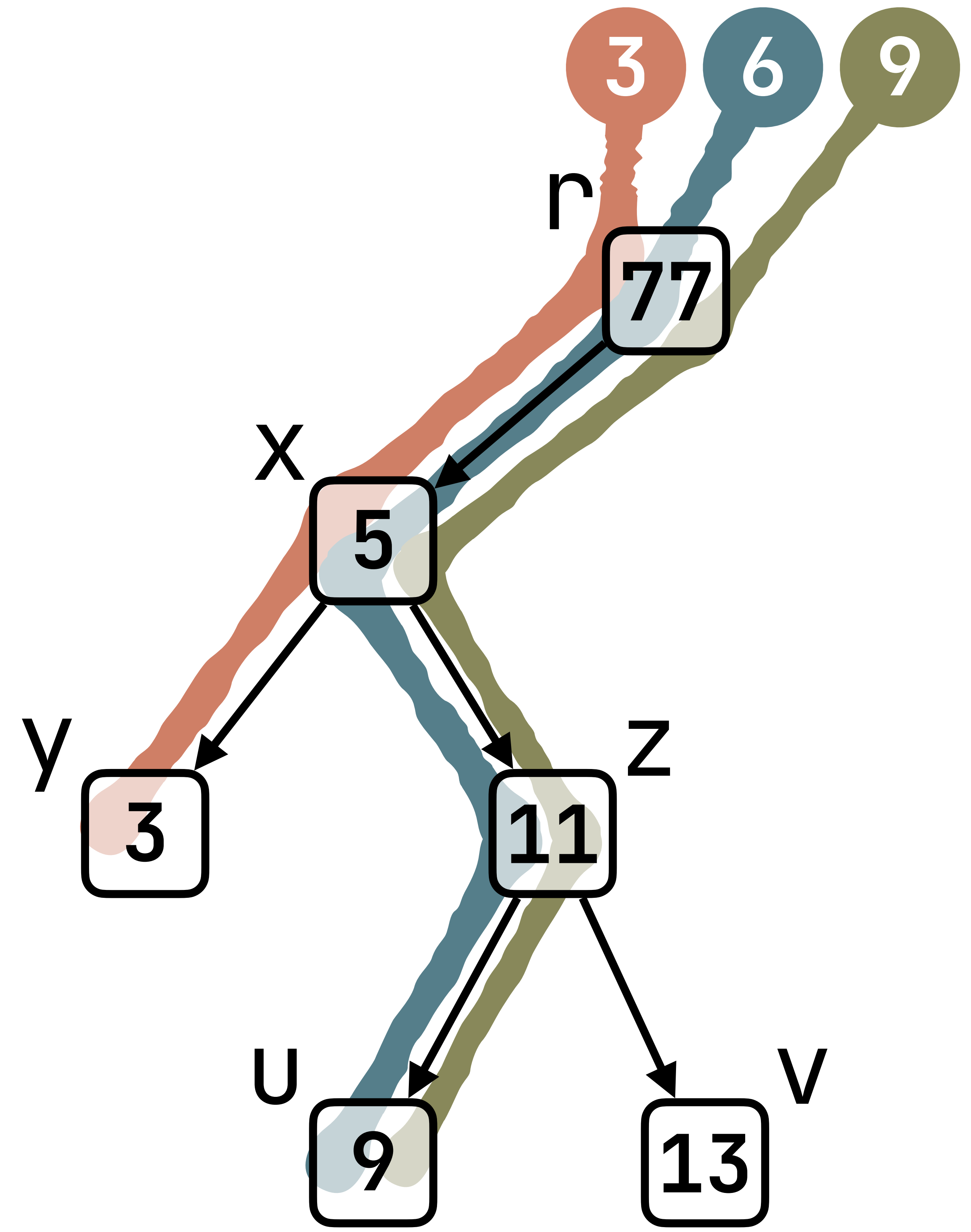
    to show linearizability of data structures



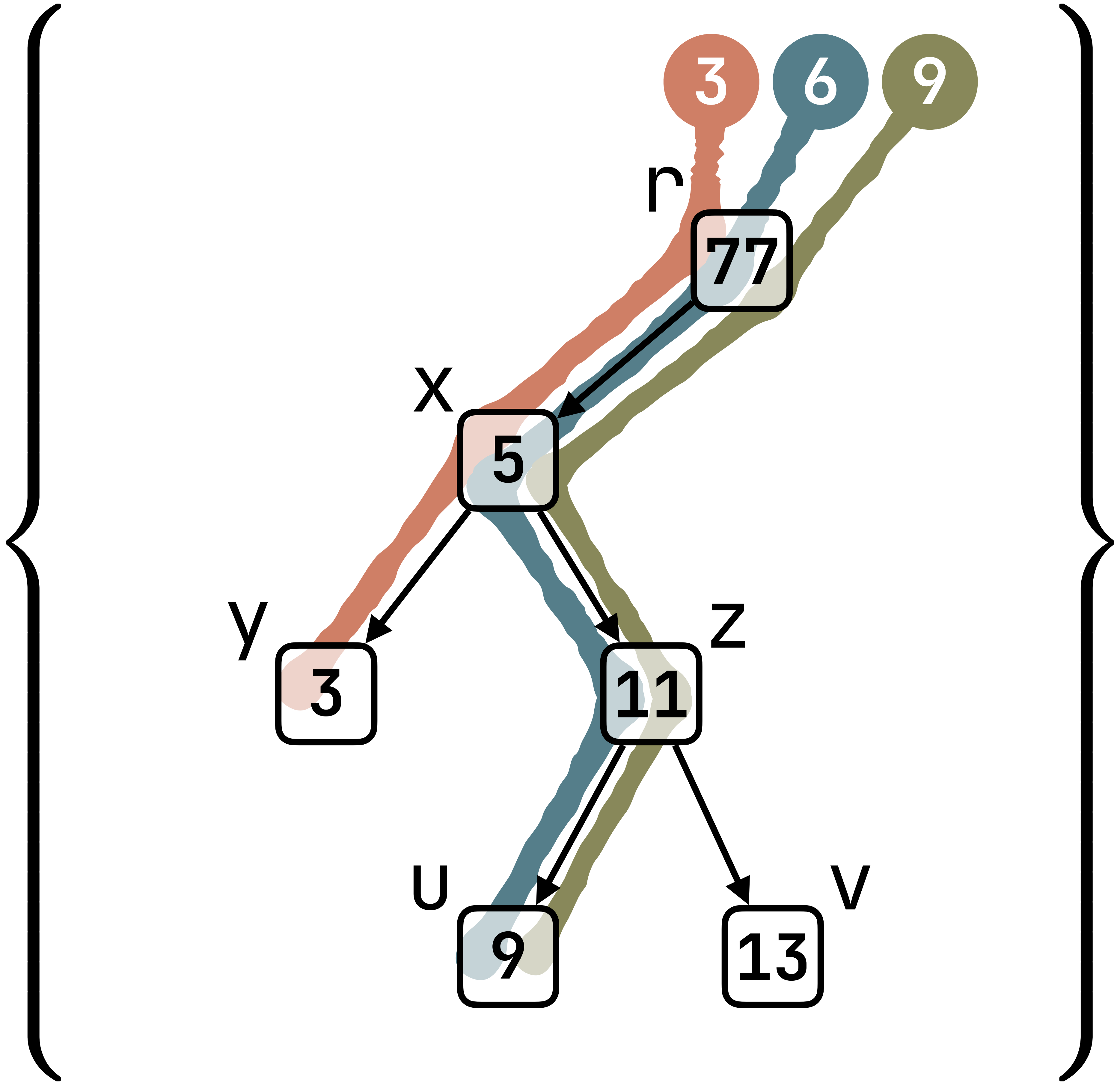
State

- Physical state
  - ➡ *heap graph*
  - ➡ e.g. $r \mapsto 77, x, \bot \;*\; x \mapsto 5, y, z \;*\; \ldots$
- Ghost state
  - ➡ info for functional correctness
  - ➡ e.g. *traversals* (search paths) 3 6 9

    to show linearizability of data structures



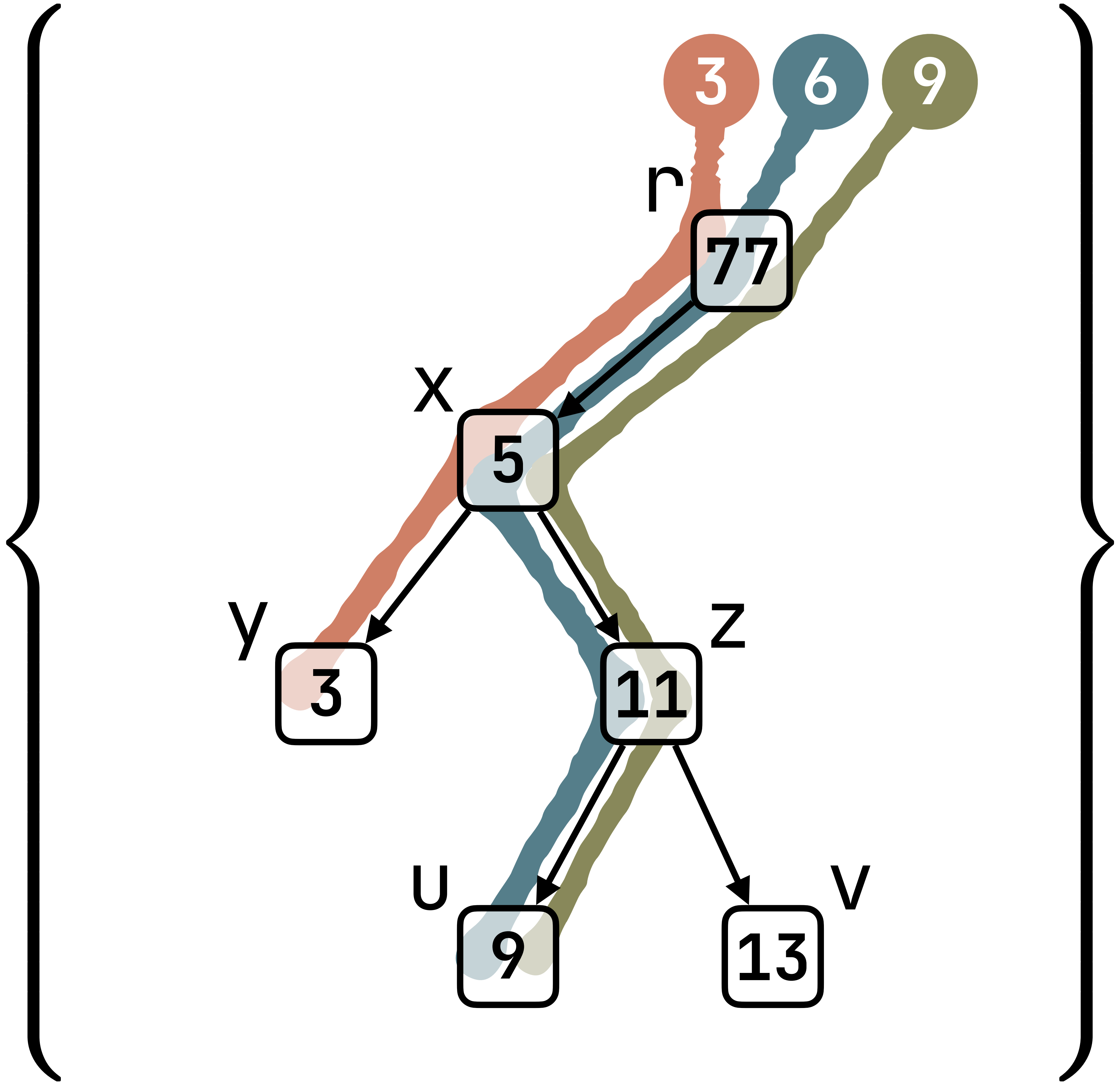
Goal: Frame Inference



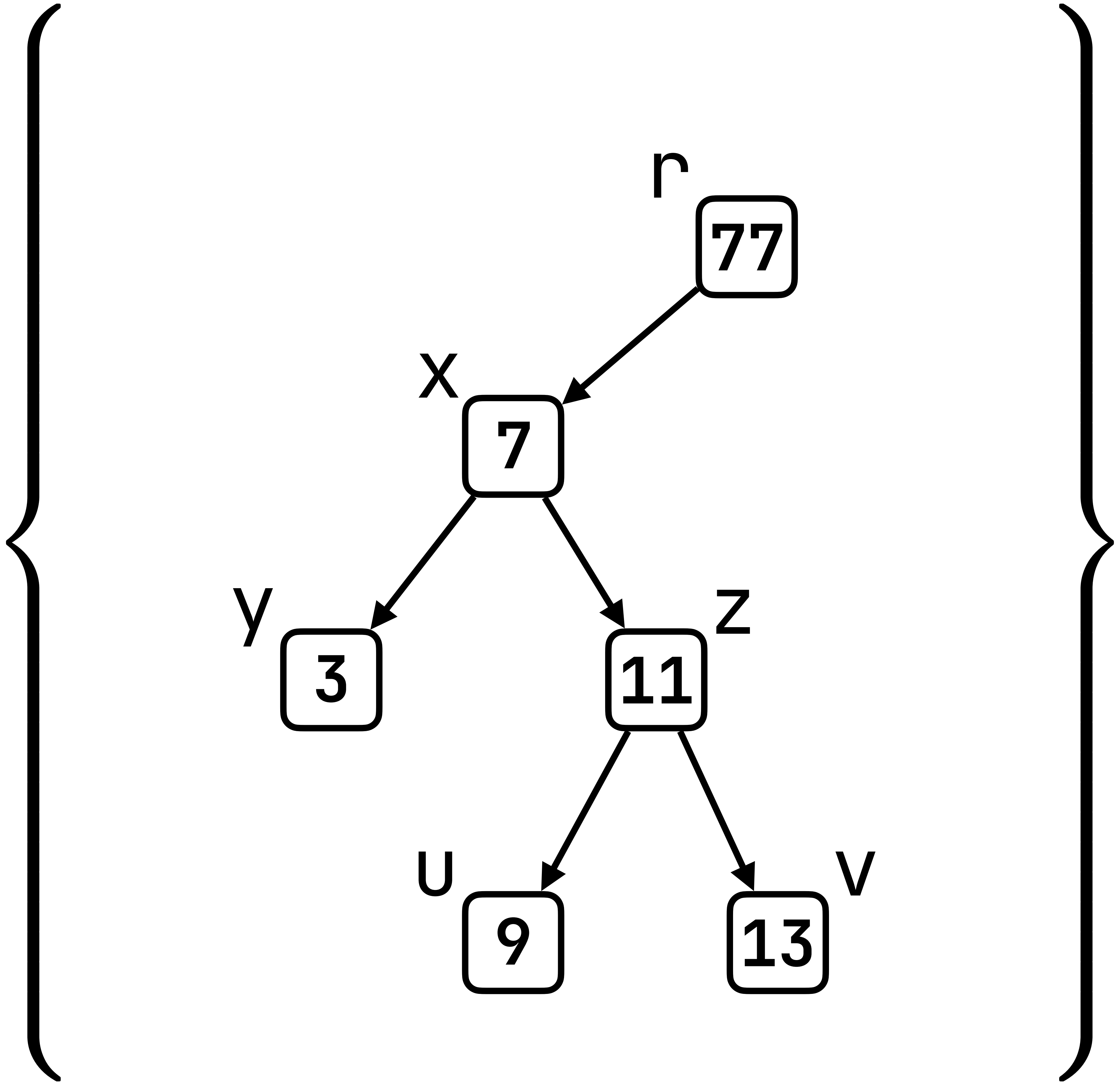
$$x.key = 7$$

$\{ \quad \}$

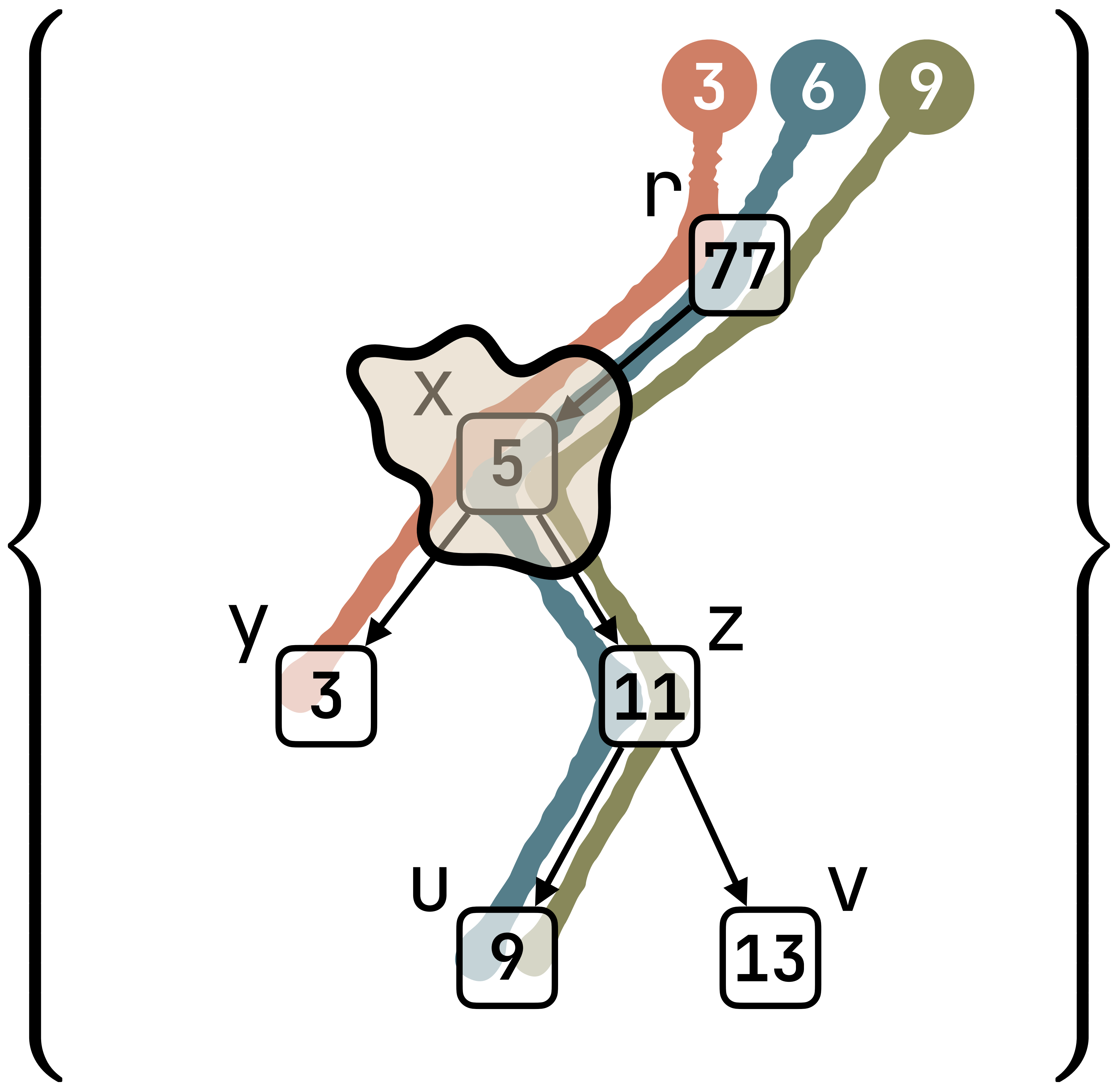
Goal: Frame Inference



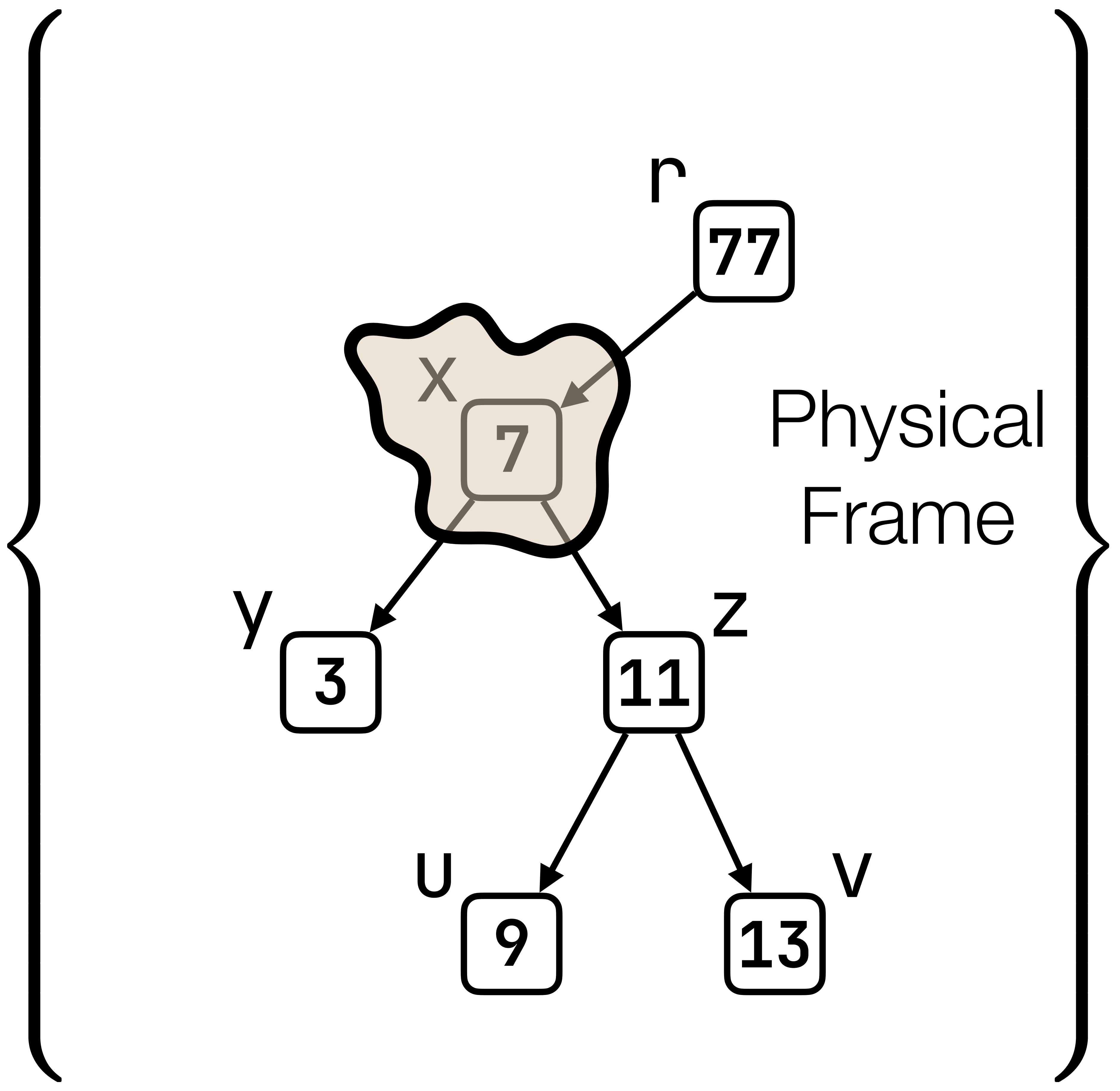
$x.key = 7$



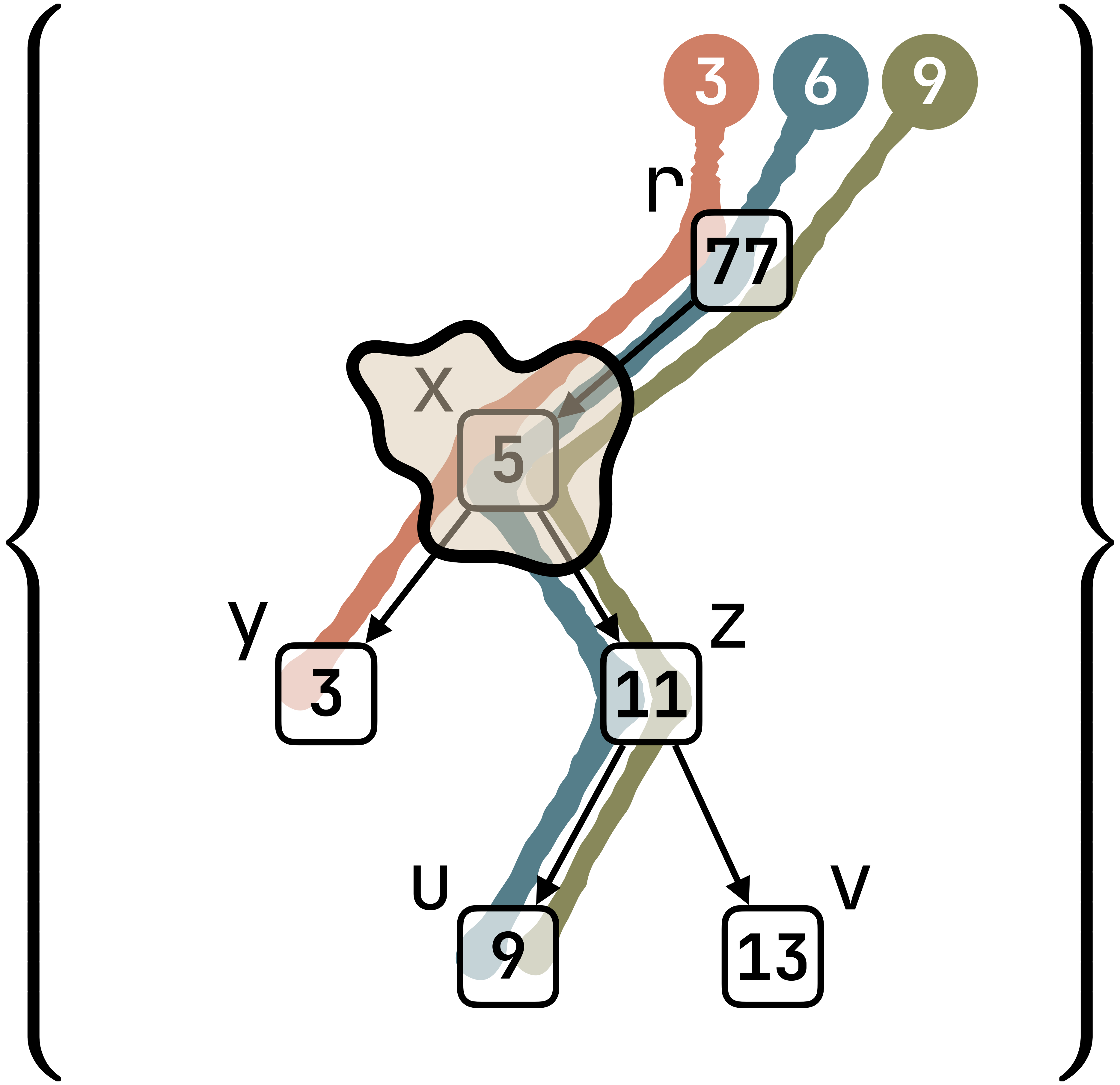
Goal: Frame Inference



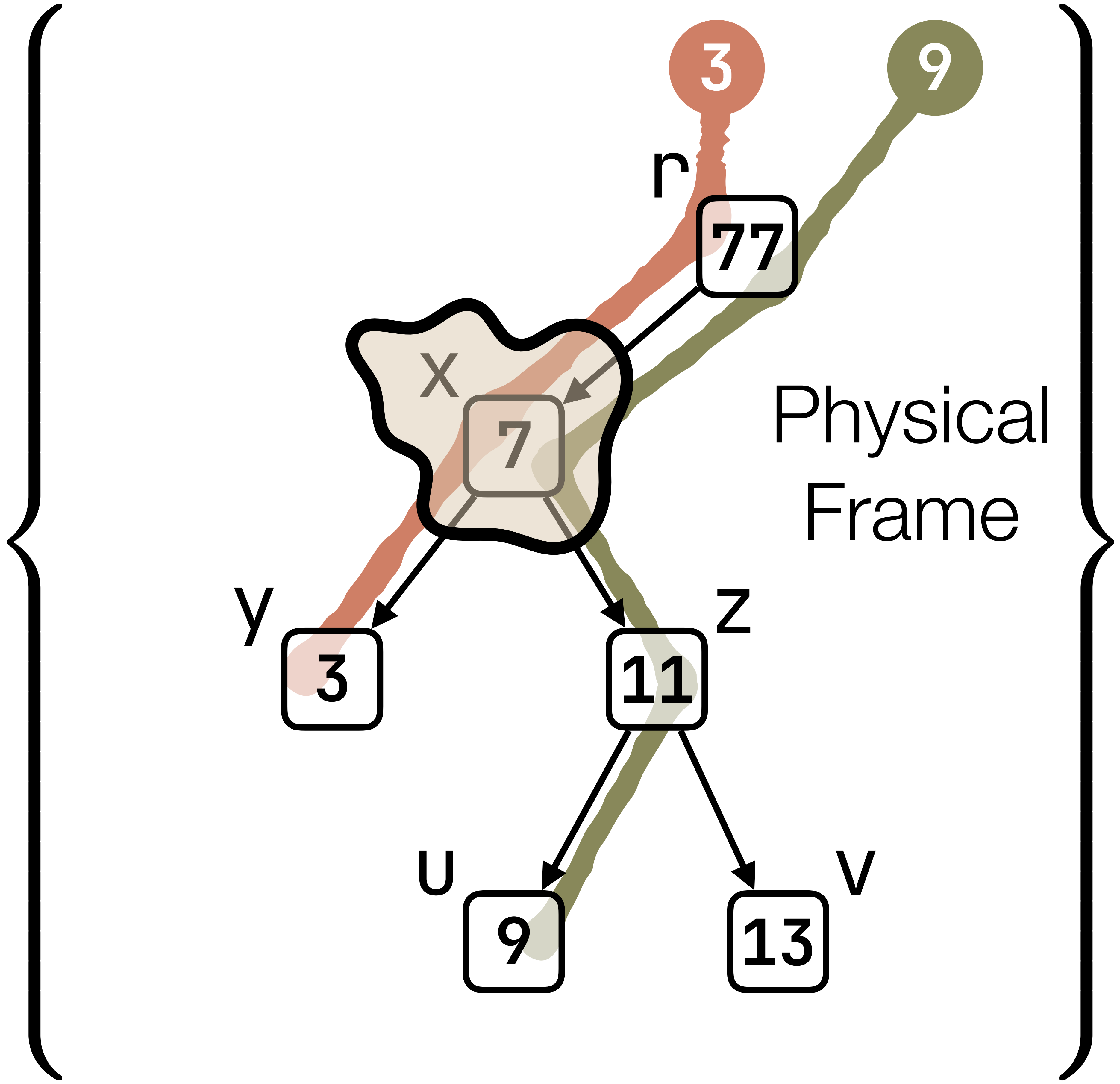
$$x.key = 7$$



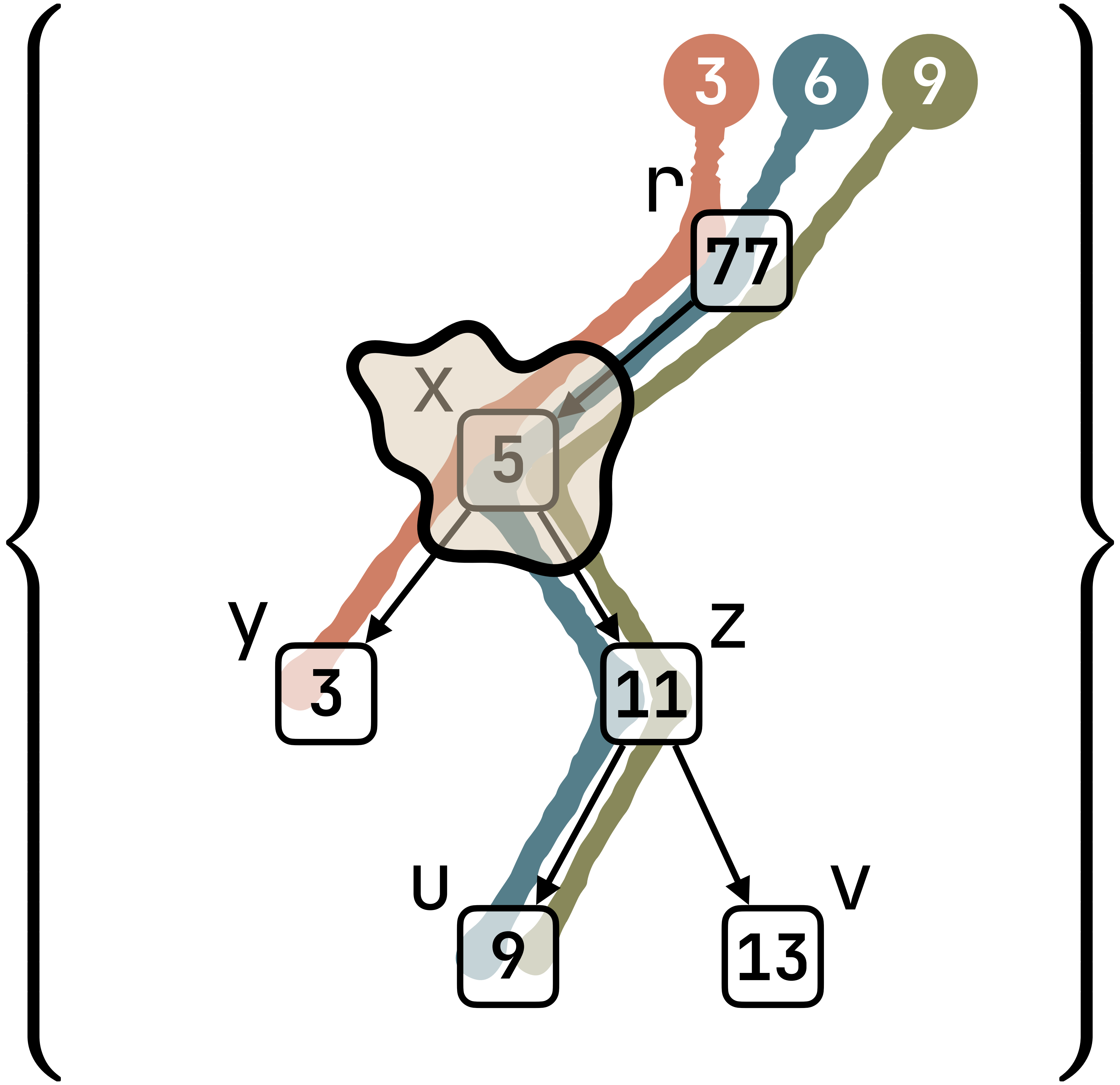
Goal: Frame Inference



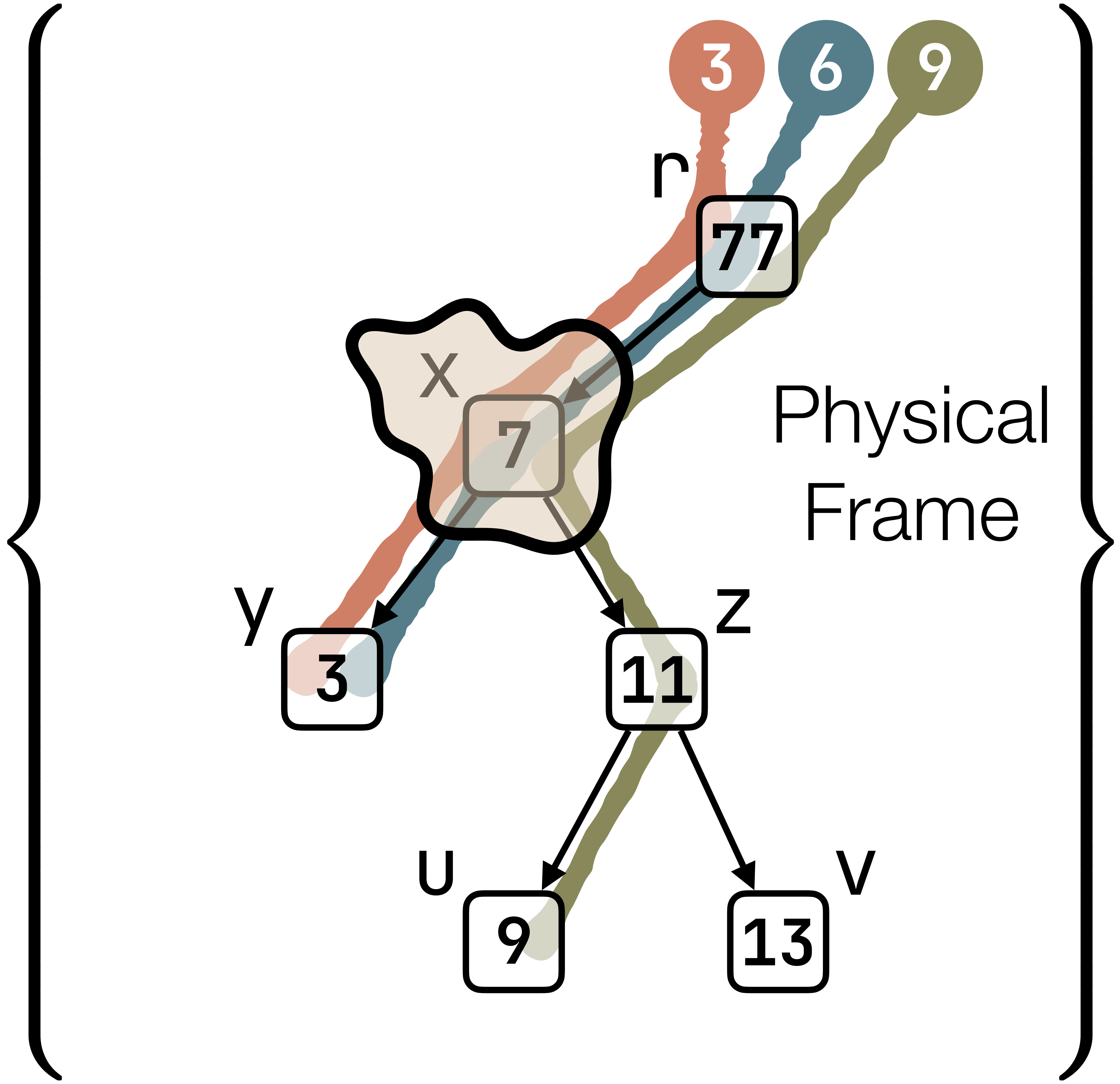
$$x.key = 7$$



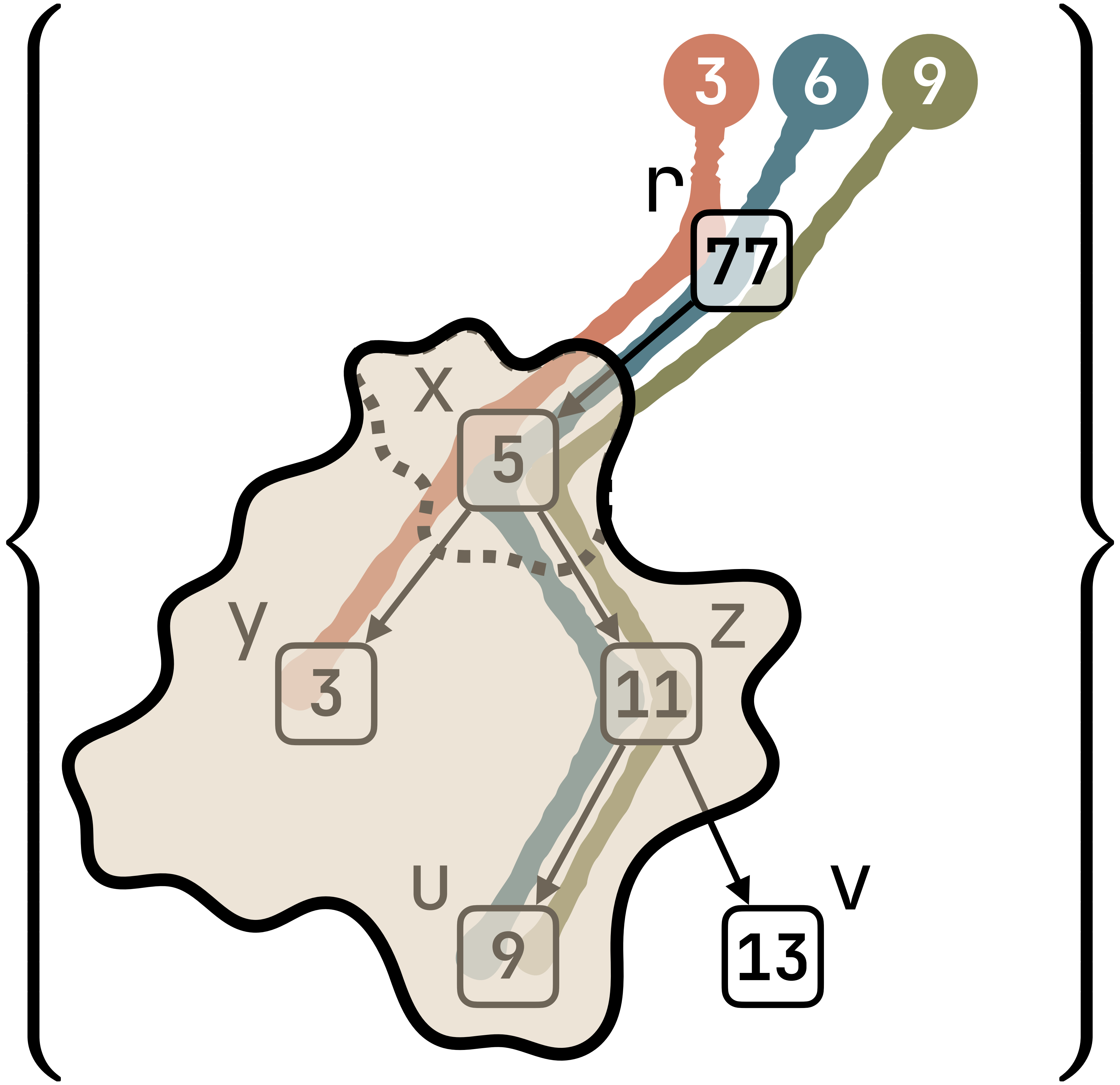
Goal: Frame Inference



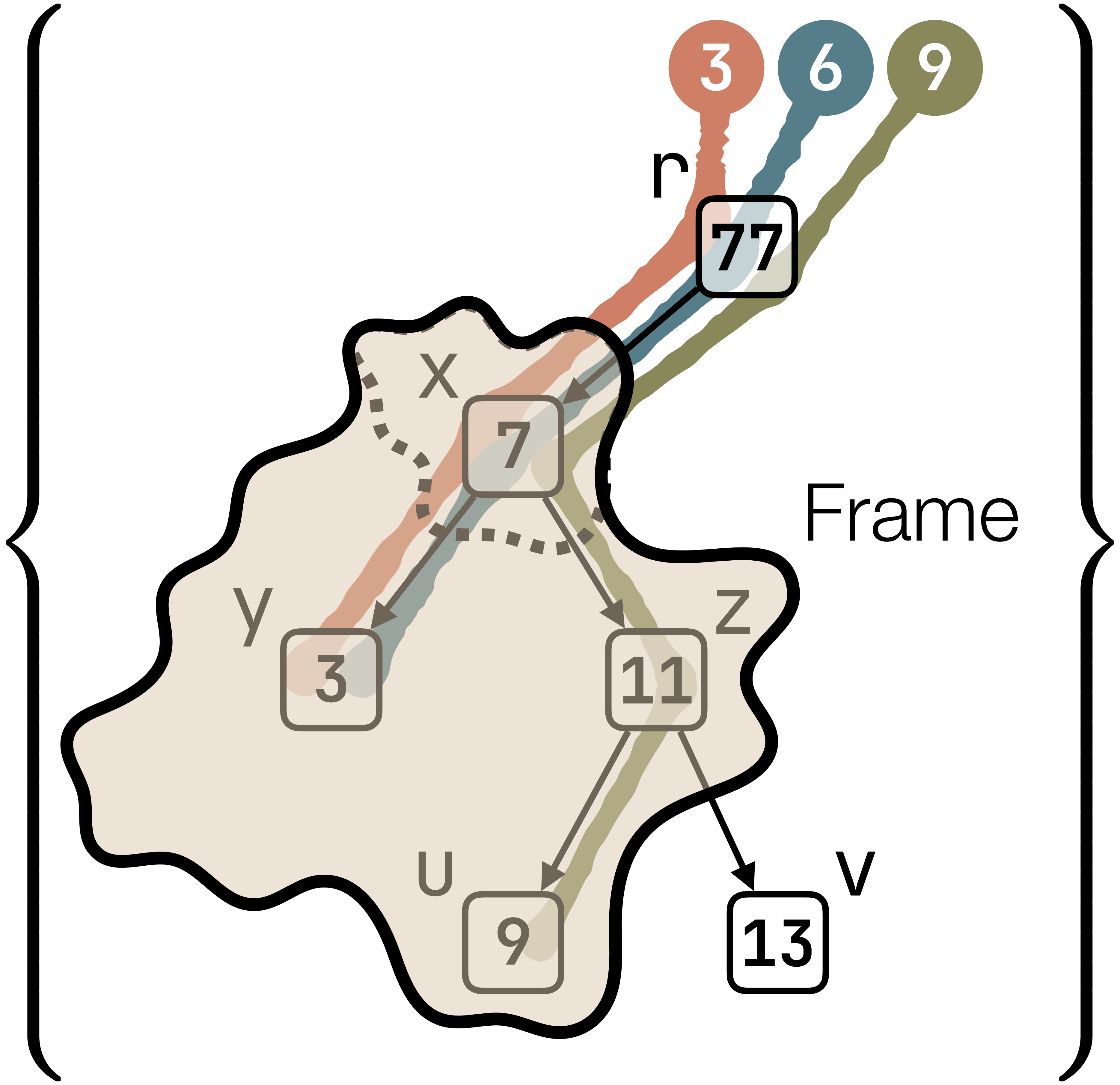
$$x.key = 7$$



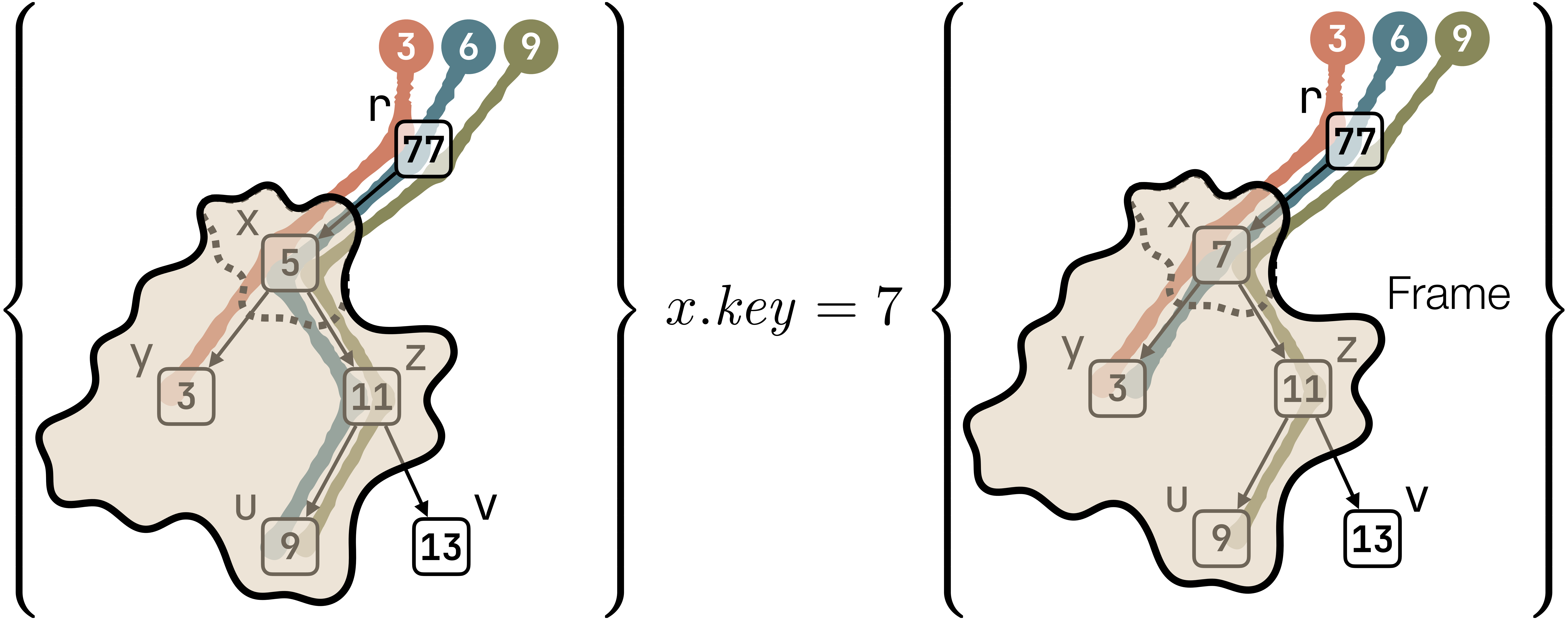
Goal: Frame Inference



$$x.key = 7$$

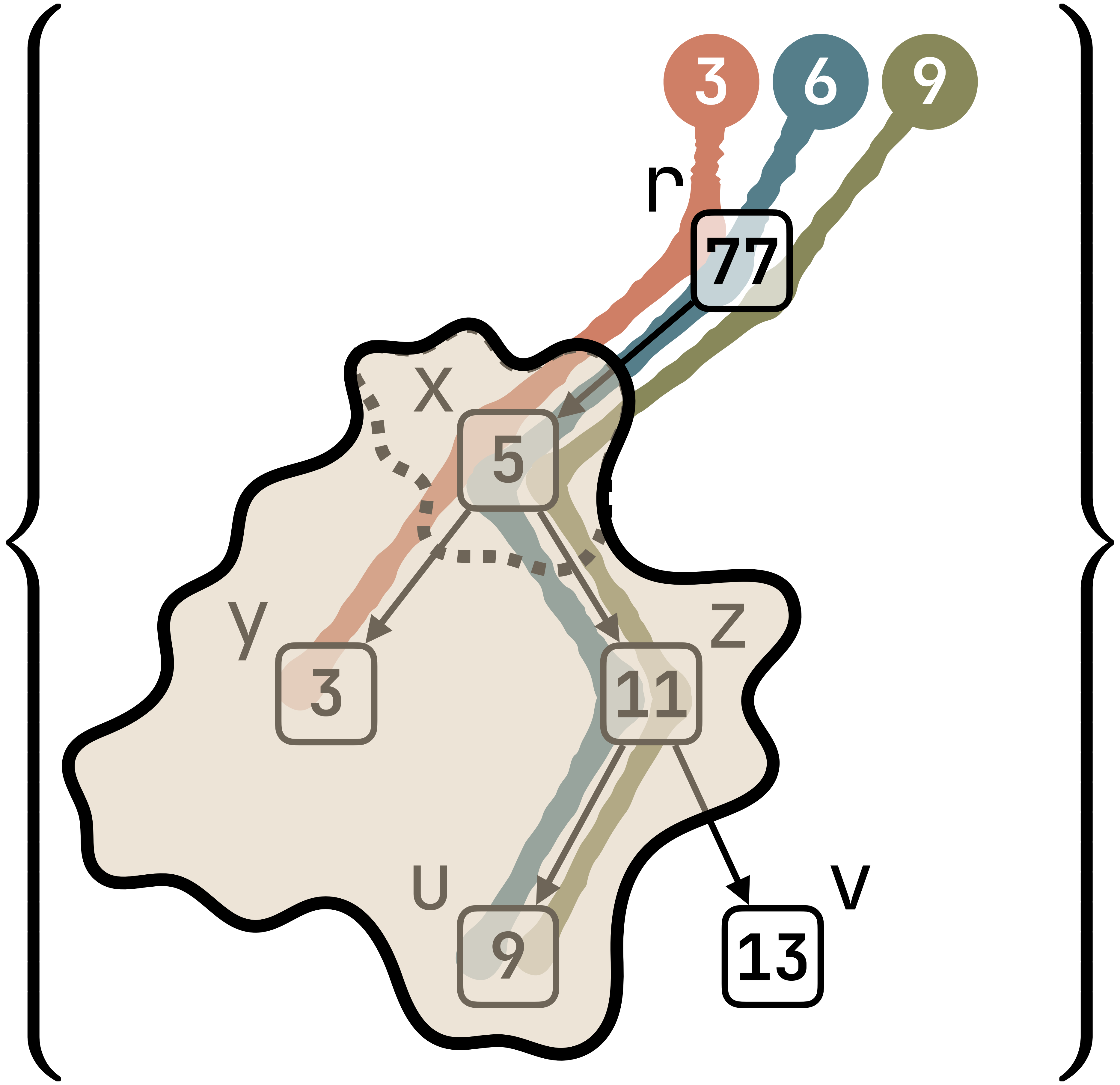


Goal: Frame Inference

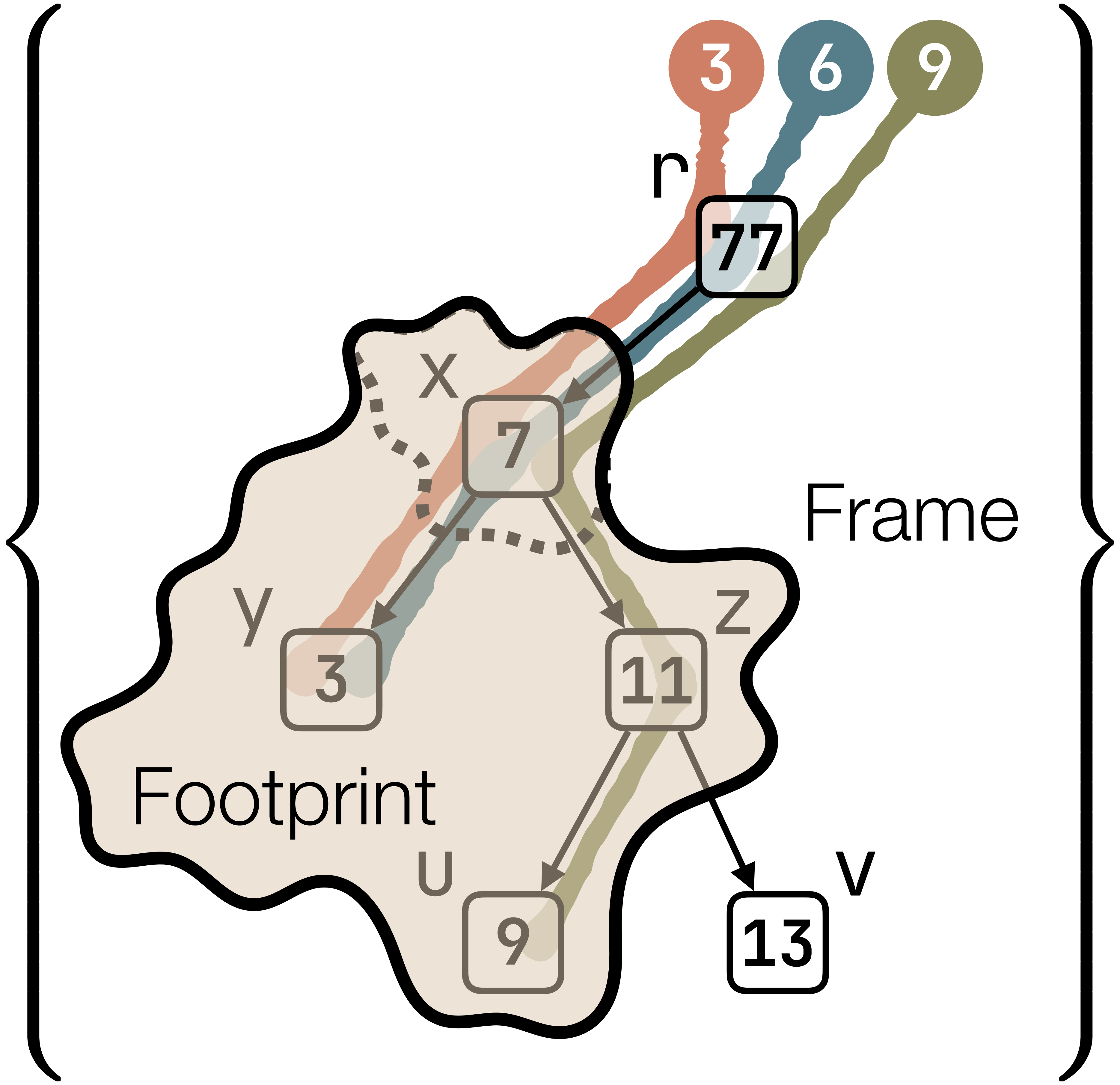


Goal: ***automatically*** find frame.

Goal: Frame Inference



$$x.key = 7$$



Goal: ***automatically*** find ***footprint***.

Flow Framework

- Ghost state for heap graphs
- Inspired by data-flow analysis
- Formalizes inductive heap invariants

Frame Inference

- Separation & flows
- Frame-preserving updates
- Finding footprints algorithmically

Comparing Footprints

- Check if update is frame-preserving
- Efficient checks for general graphs

Flow Framework

- Ghost state for heap graphs
- Inspired by data-flow analysis
- Formalizes inductive heap invariants

Frame Inference

- Separation & flows
- Frame-preserving updates
- Finding footprints algorithmically

Comparing Footprints

- Check if update is frame-preserving
- Efficient checks for general graphs

Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

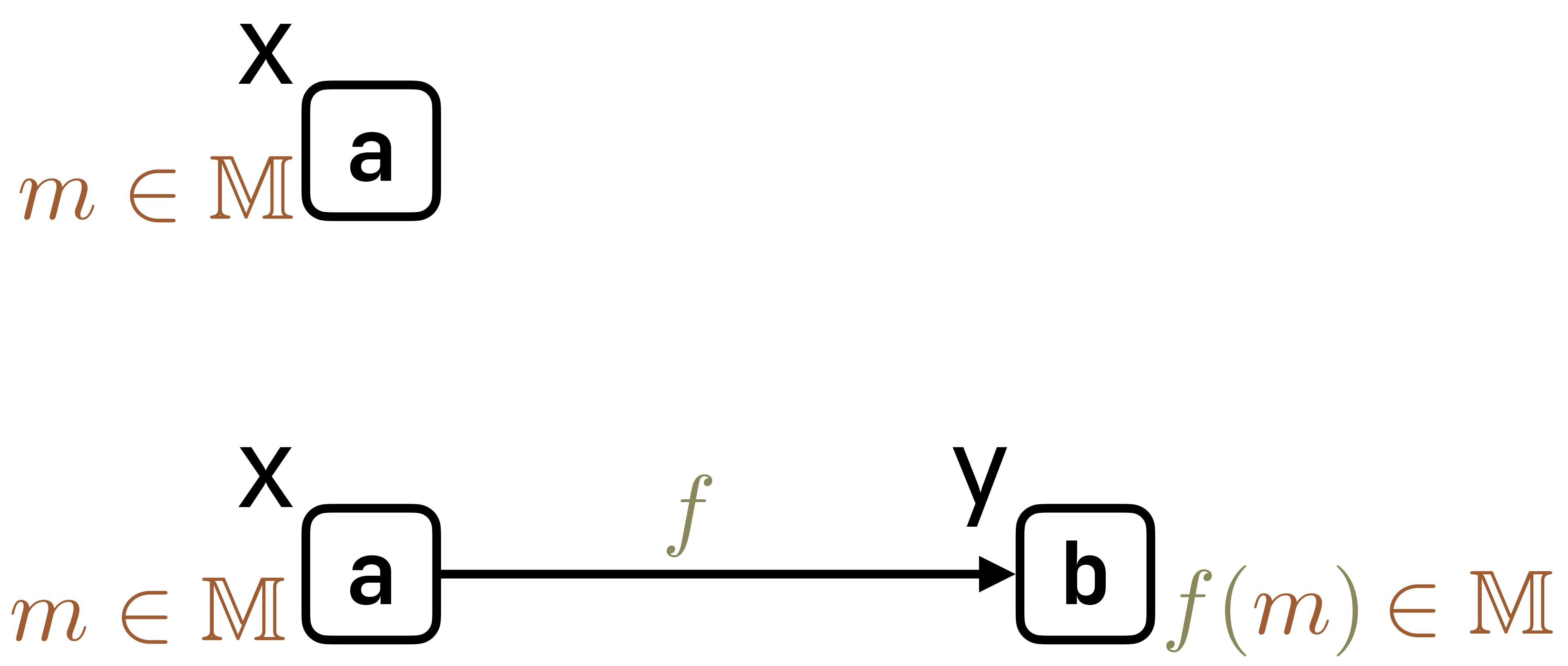
- Flow values from
  commutative monoid $(\mathbb{M}, +, 0)$

Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from
  commutative monoid $(\mathbb{M}, +, 0)$

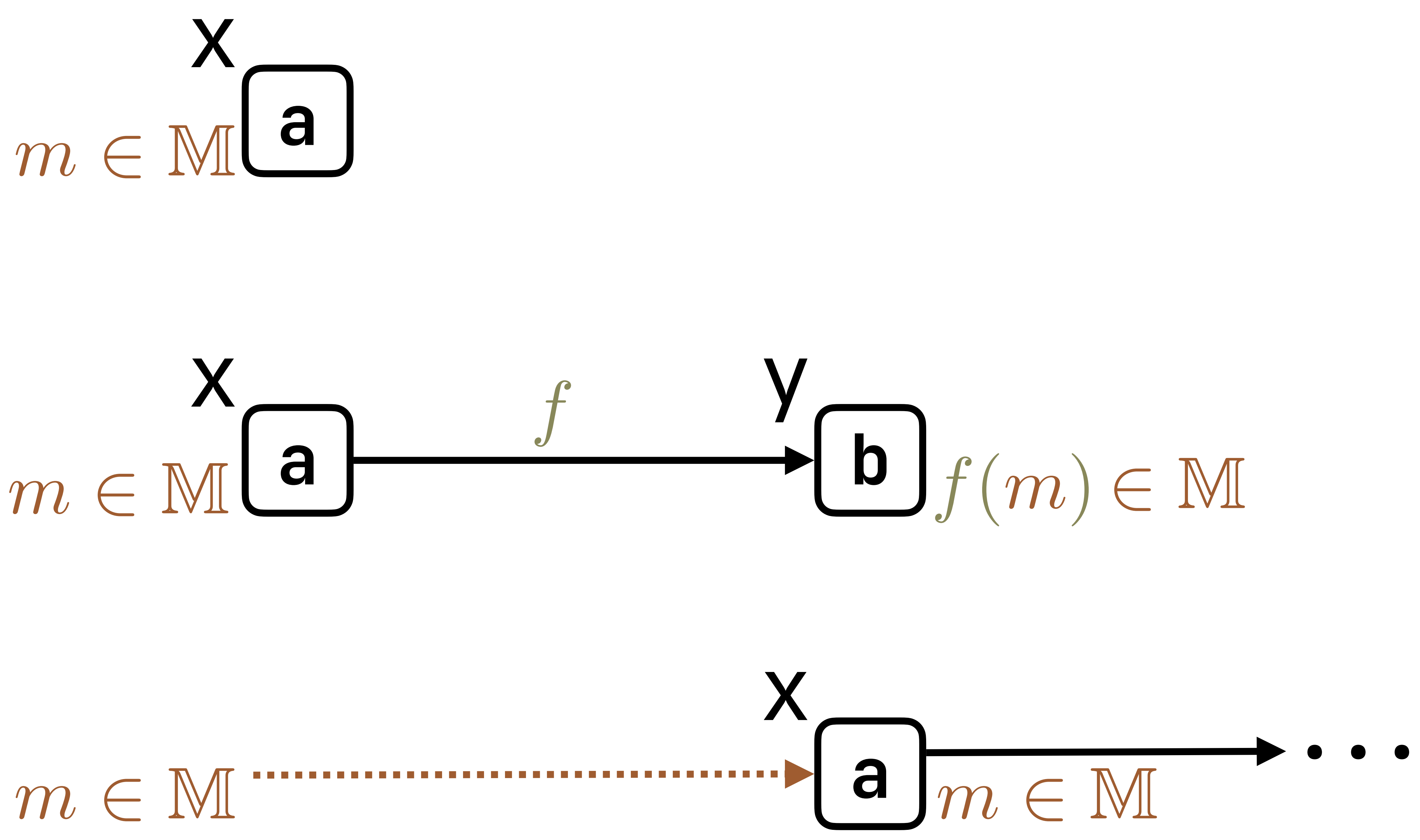
- Flow propagation via
  continuous edge functions



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

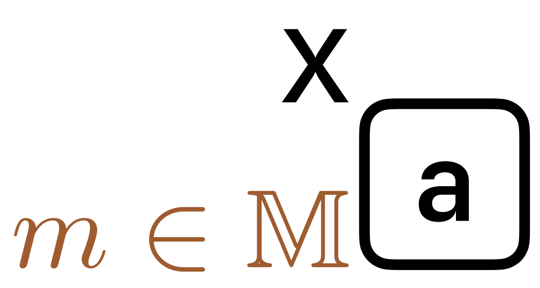
- Flow values from
  commutative monoid $(\mathbb{M}, +, 0)$
- Flow propagation via
  continuous edge functions
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

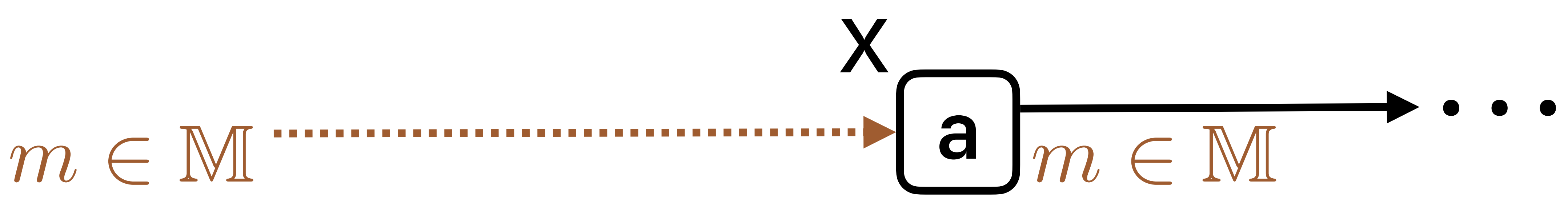
— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from
  commutative monoid $(\mathbb{M}, +, 0)$



- Flow propagation via
  continuous edge functions

- ***The flow***:
  <u>least</u> fixed point, wrt. initial value



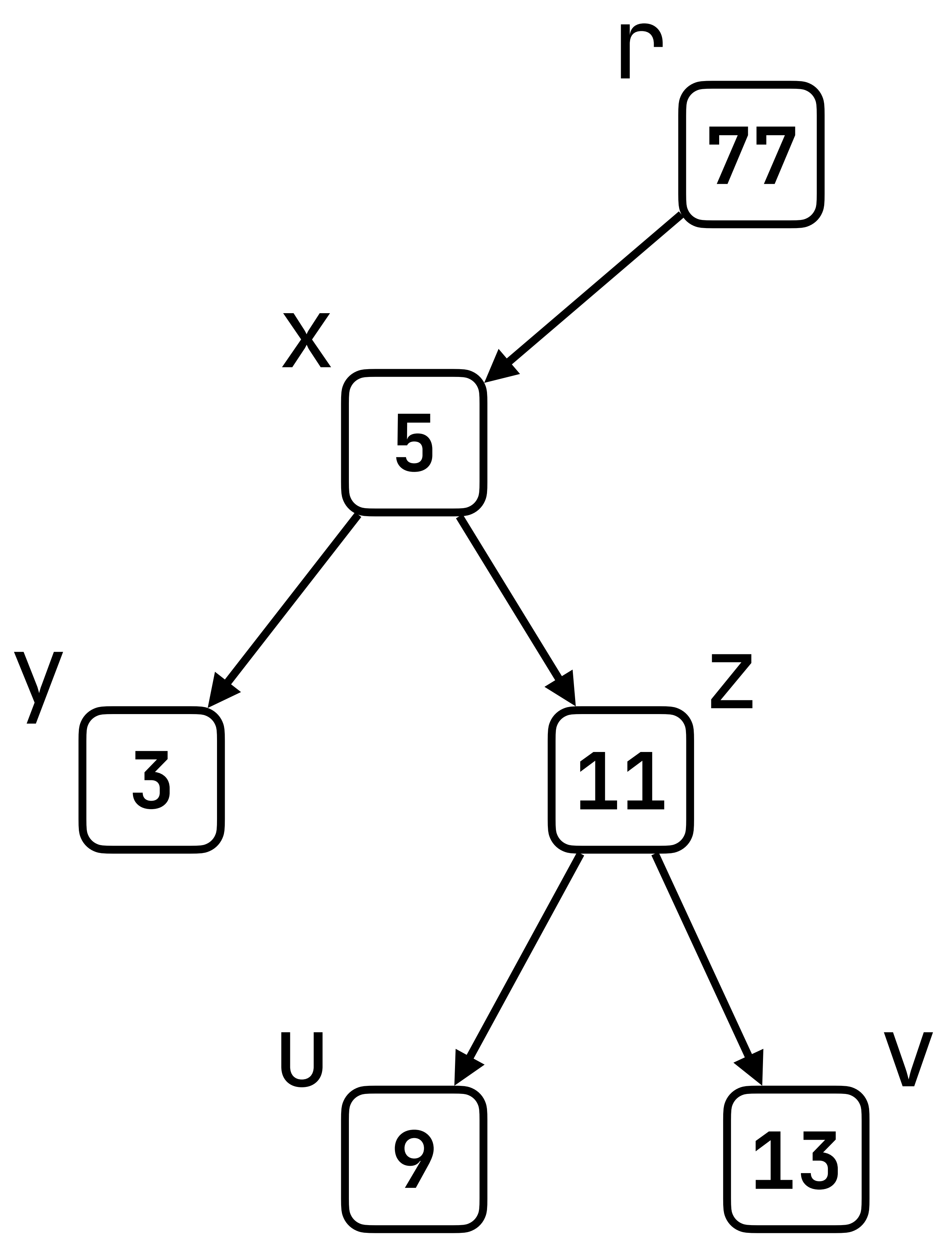
NEW

Exists if: $\leq$ is ω-cpo and $+,\sup$ commute

Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

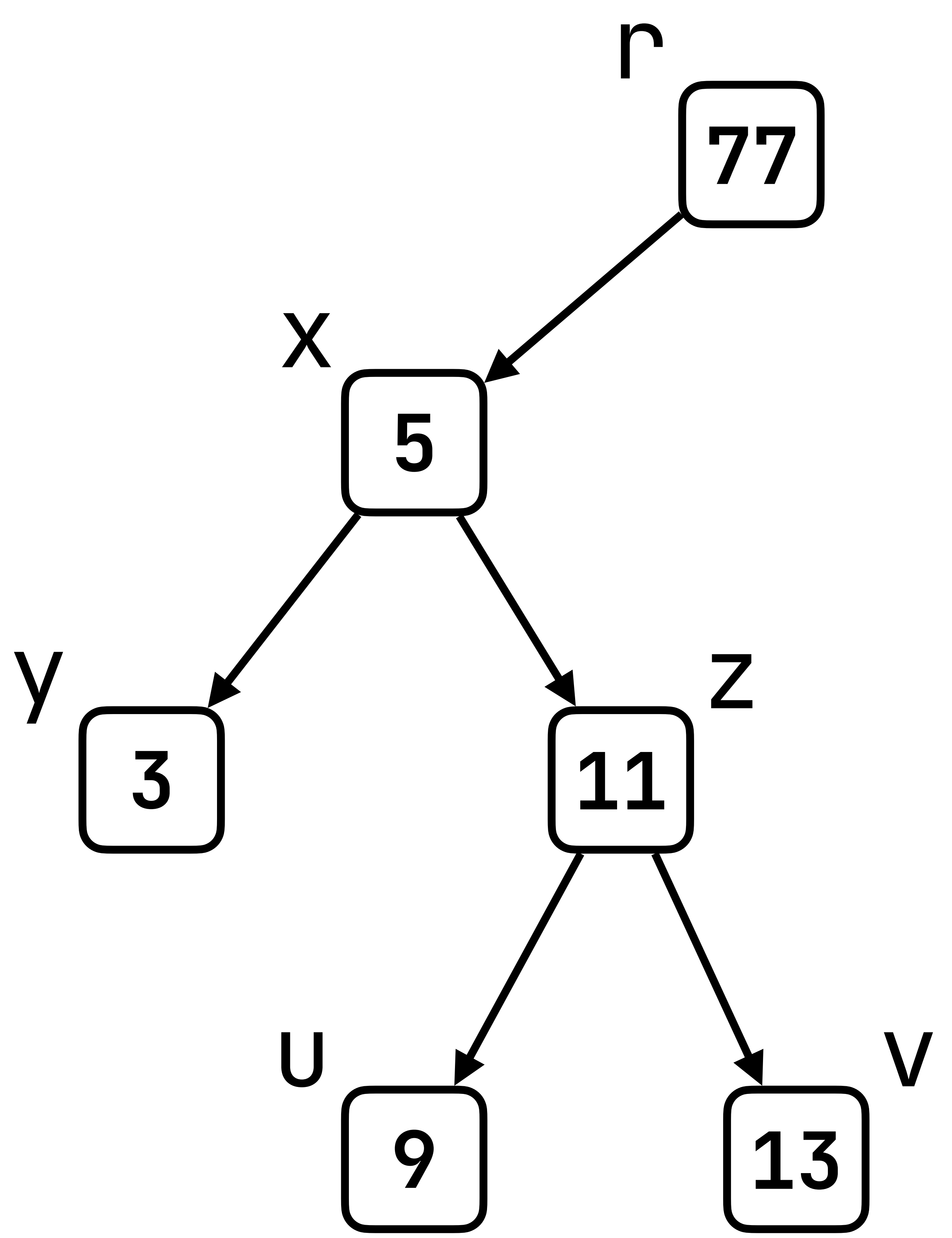
- Flow values from
- Flow propagation via
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \varnothing)$
- Flow propagation via
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

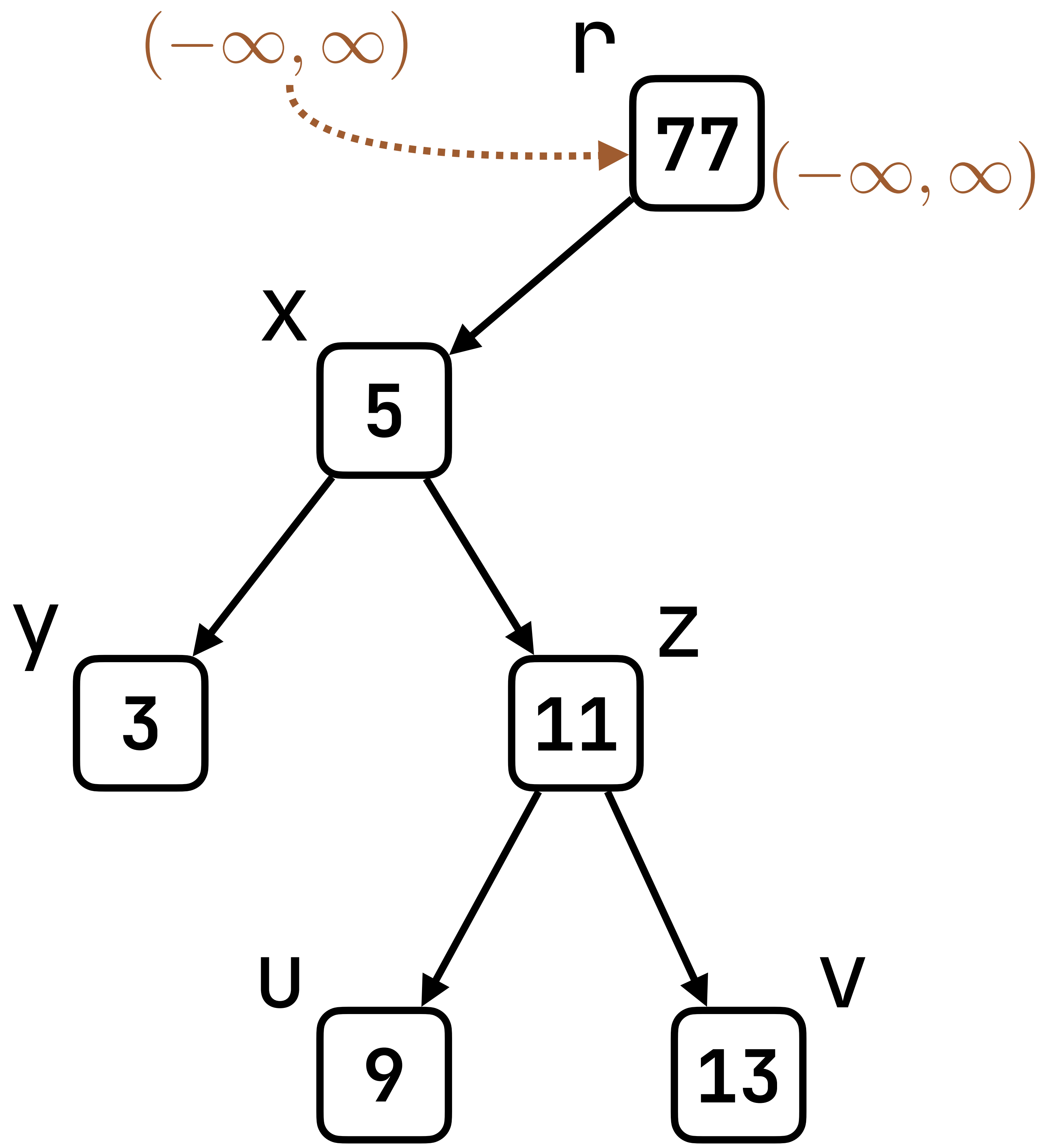
- Flow values from

  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \varnothing)$

- Flow propagation via

- ***The flow***:

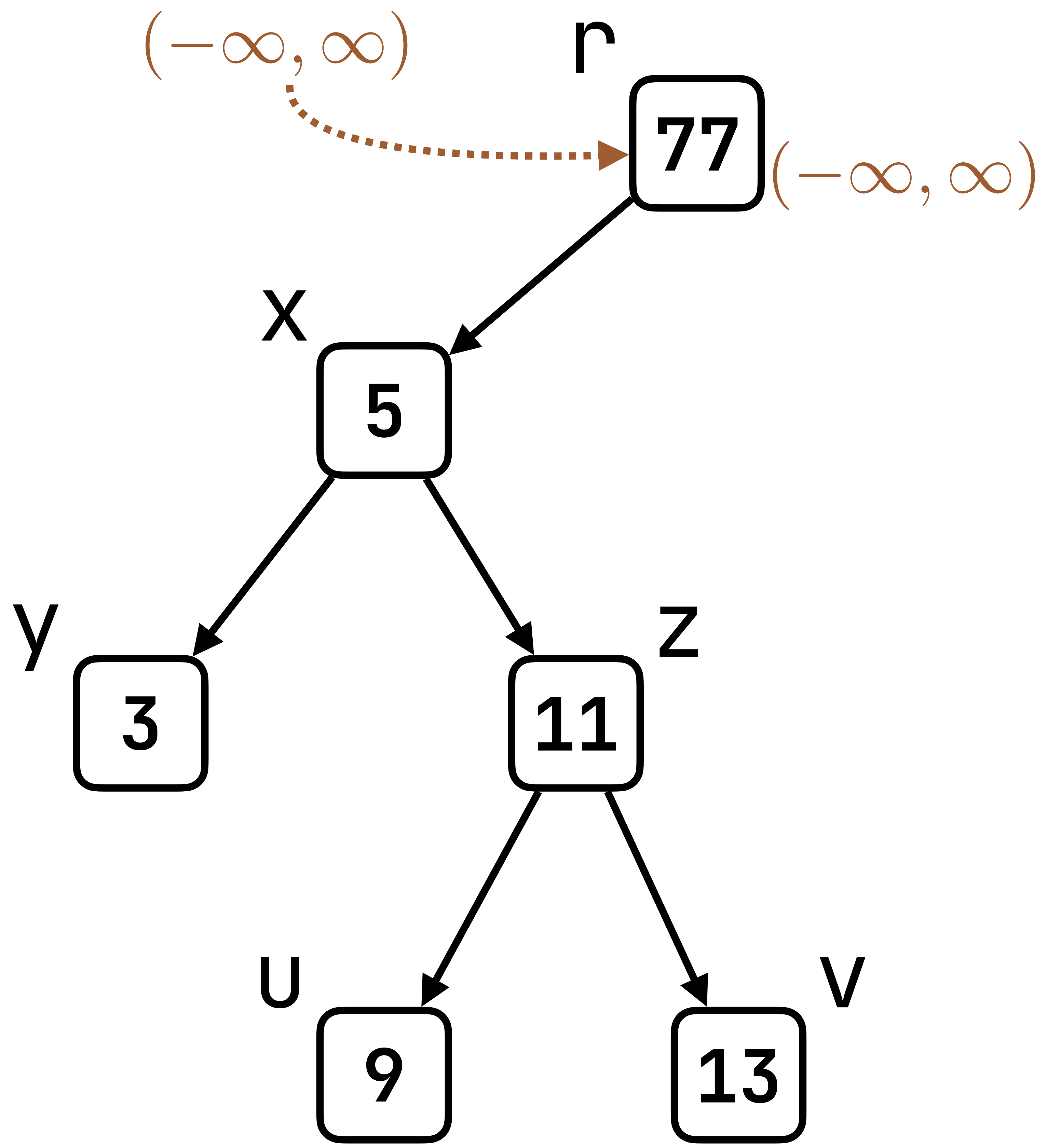
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

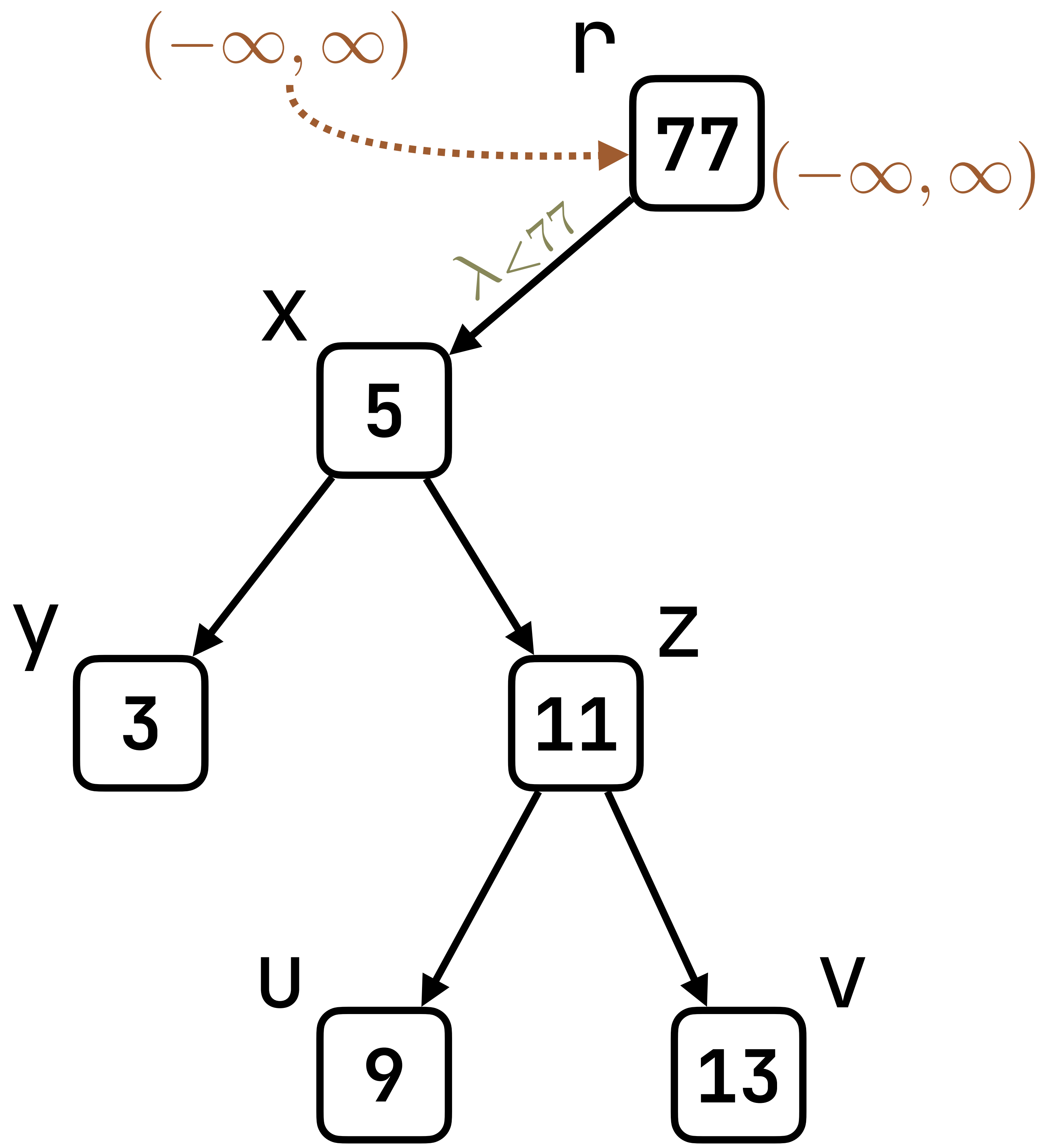
- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \varnothing)$
- Flow propagation via
  $$\lambda_{<k} \; := \; \lambda m.\, m \cap (-\infty, k)$$
  $$\lambda_{>k} \; := \; \lambda m.\, m \cap (k, \infty)$$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

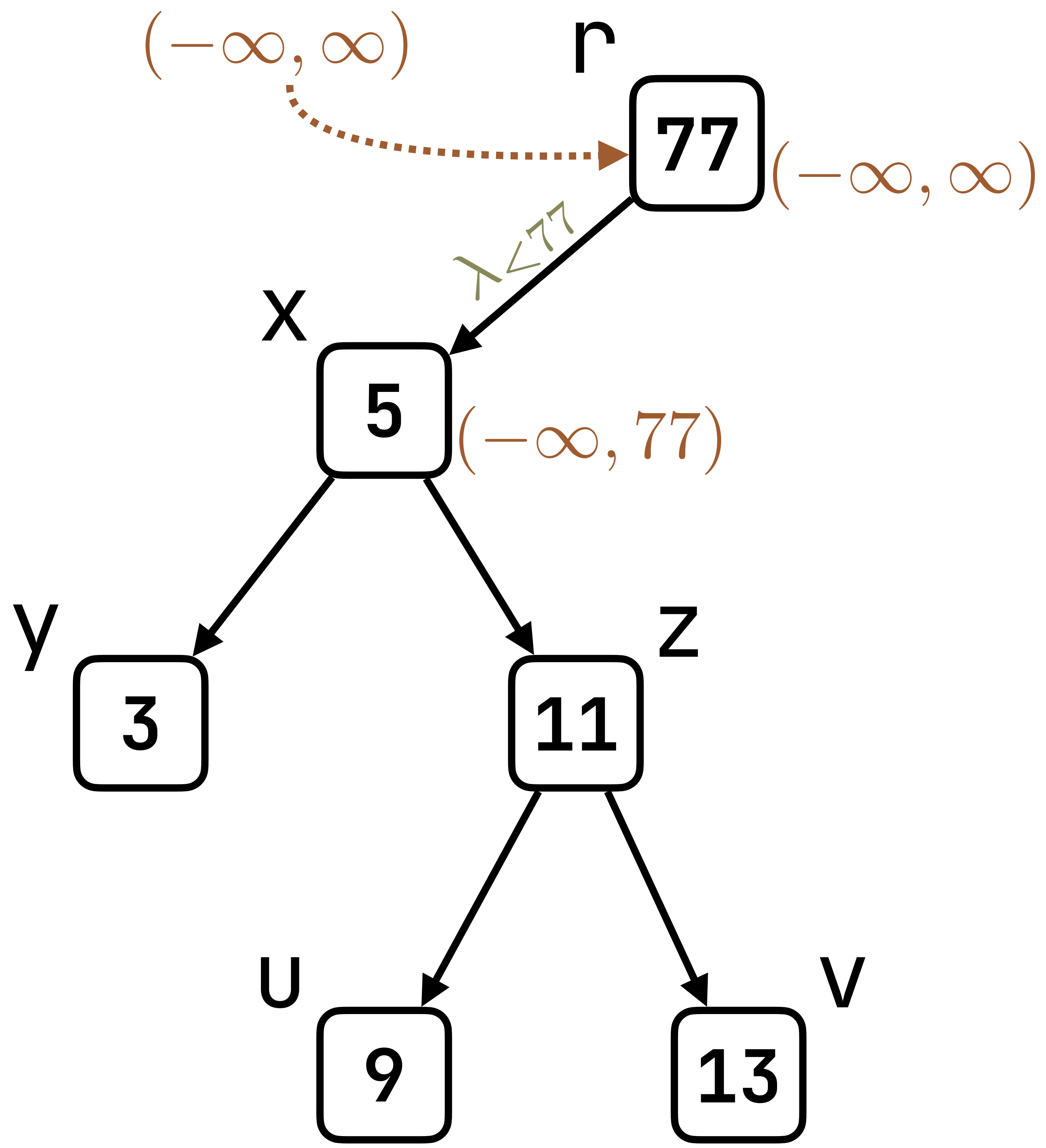
- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \varnothing)$
- Flow propagation via
  $\lambda_{<k} := \lambda m.\, m \cap (-\infty, k)$
  $\lambda_{>k} := \lambda m.\, m \cap (k, \infty)$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

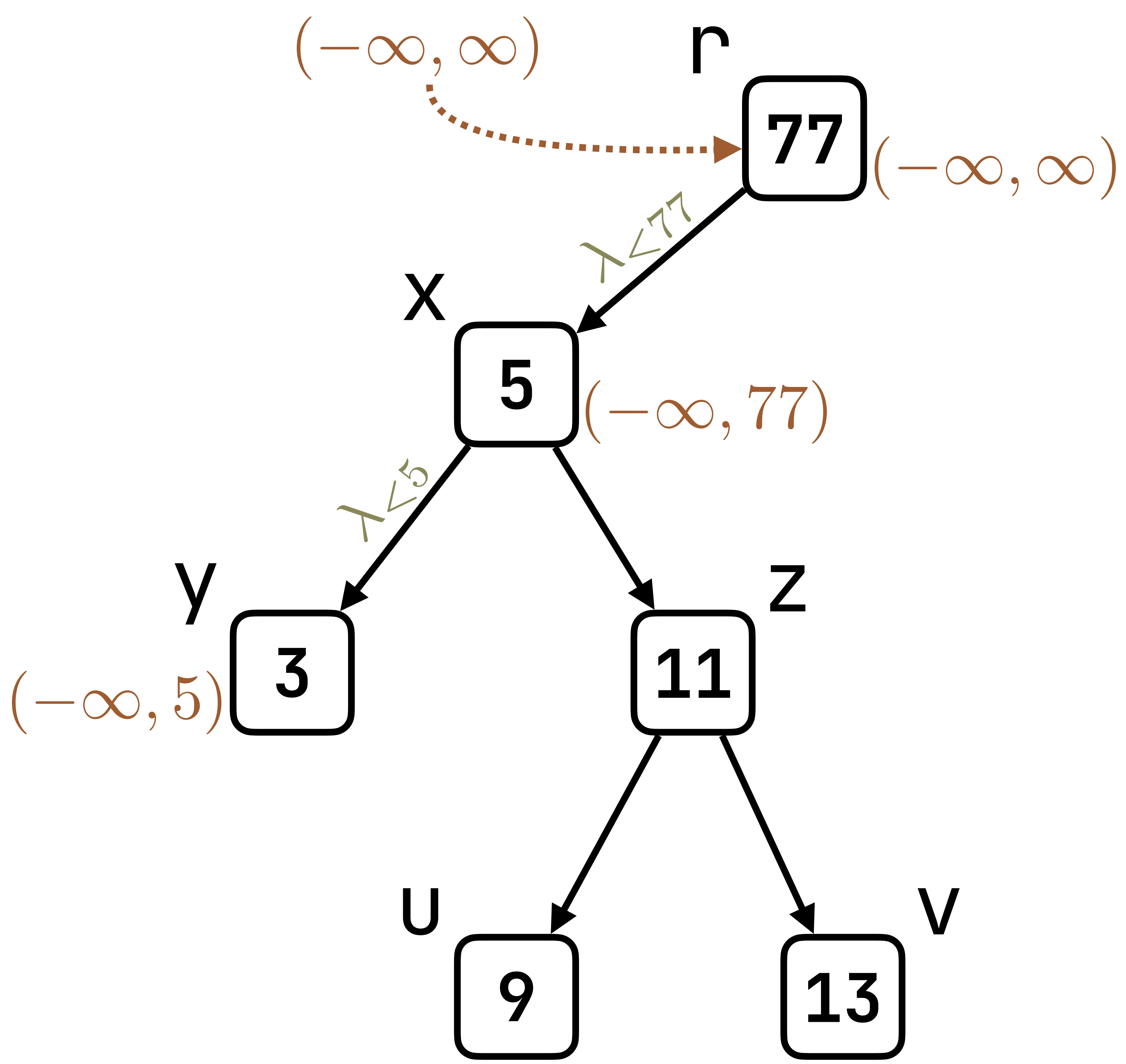
- Flow values from
  search path monoid $(2^{\mathbb{Z}\cup\{-\infty,\infty\}}, \cup, \varnothing)$
- Flow propagation via
  $\lambda_{<k} \;:=\; \lambda m.\, m \cap (-\infty, k)$
  $\lambda_{>k} \;:=\; \lambda m.\, m \cap (k, \infty)$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

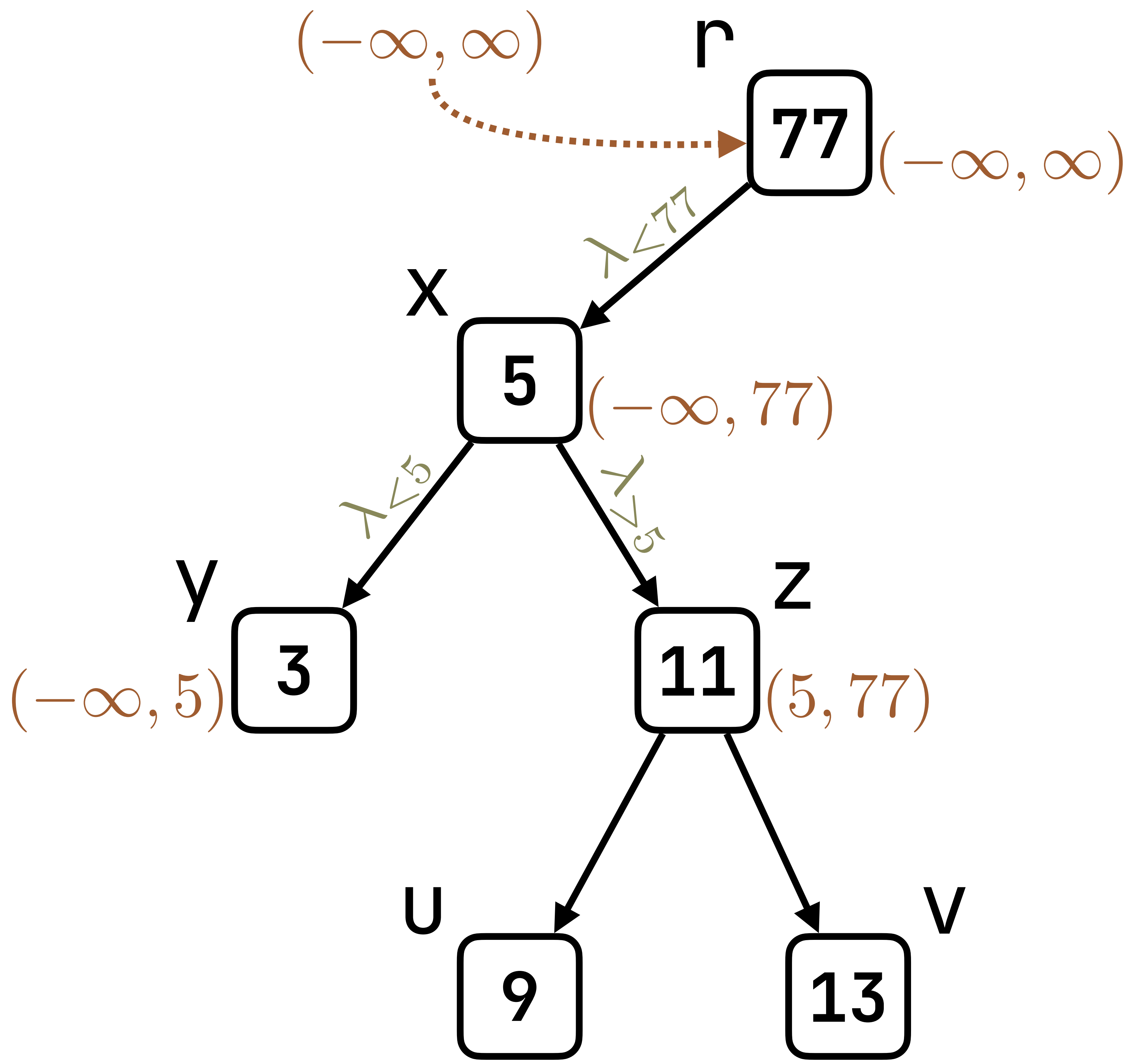
- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \varnothing)$
- Flow propagation via
  $$\lambda_{<k} \;:=\; \lambda m.\, m \cap (-\infty, k)$$
  $$\lambda_{>k} \;:=\; \lambda m.\, m \cap (k, \infty)$$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from
  search path monoid $(2^{\mathbb{Z}\cup\{-\infty,\infty\}}, \cup, \emptyset)$
- Flow propagation via
  $$\lambda_{<k} := \lambda m.\, m \cap (-\infty, k)$$
  $$\lambda_{>k} := \lambda m.\, m \cap (k, \infty)$$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from

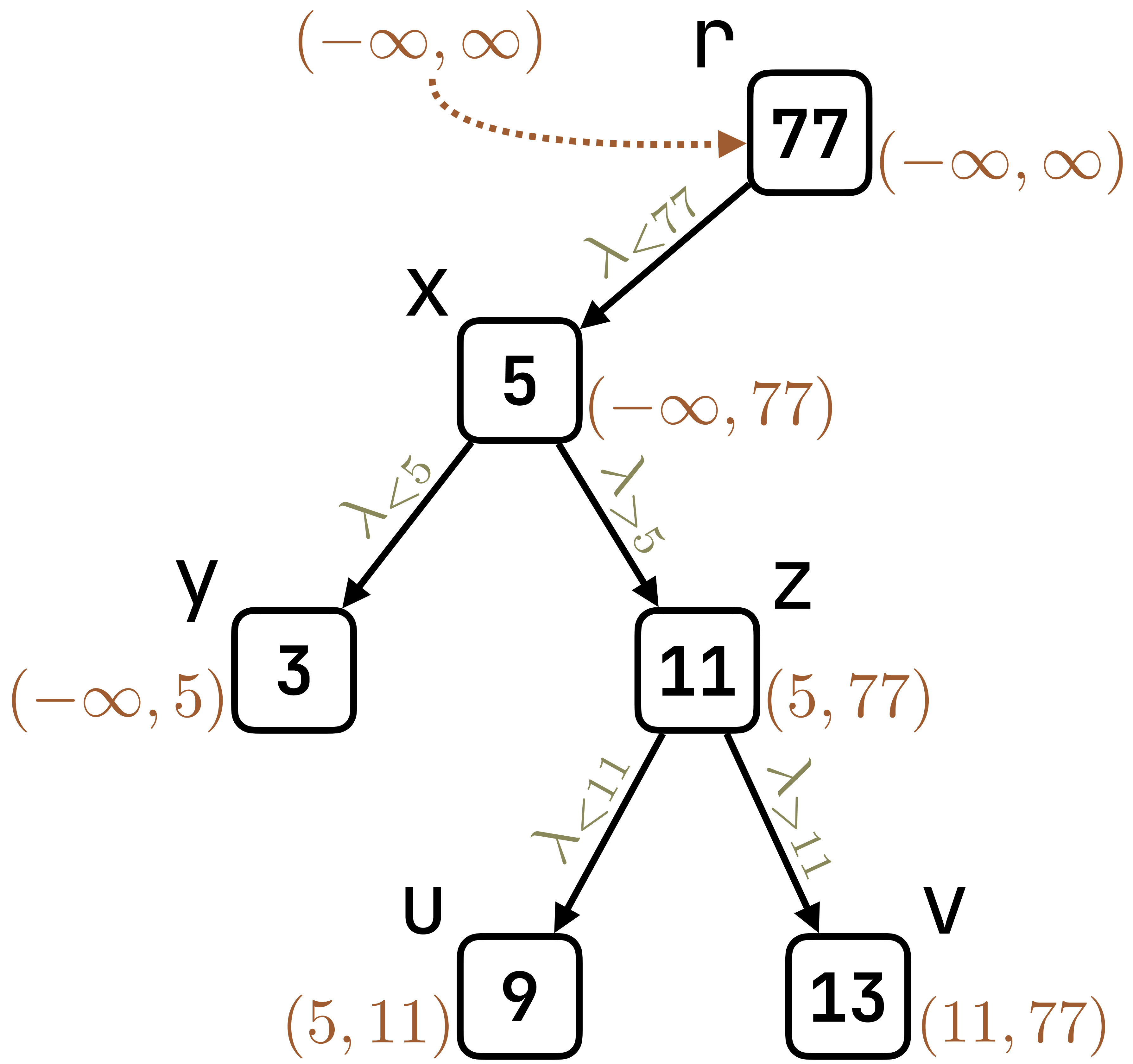
  search path monoid $(2^{\mathbb{Z}\cup\{-\infty,\infty\}}, \cup, \emptyset)$

- Flow propagation via

  $$\lambda_{<k} \;:=\; \lambda m.\, m \cap (-\infty, k)$$
  $$\lambda_{>k} \;:=\; \lambda m.\, m \cap (k, \infty)$$

- ***The flow***:

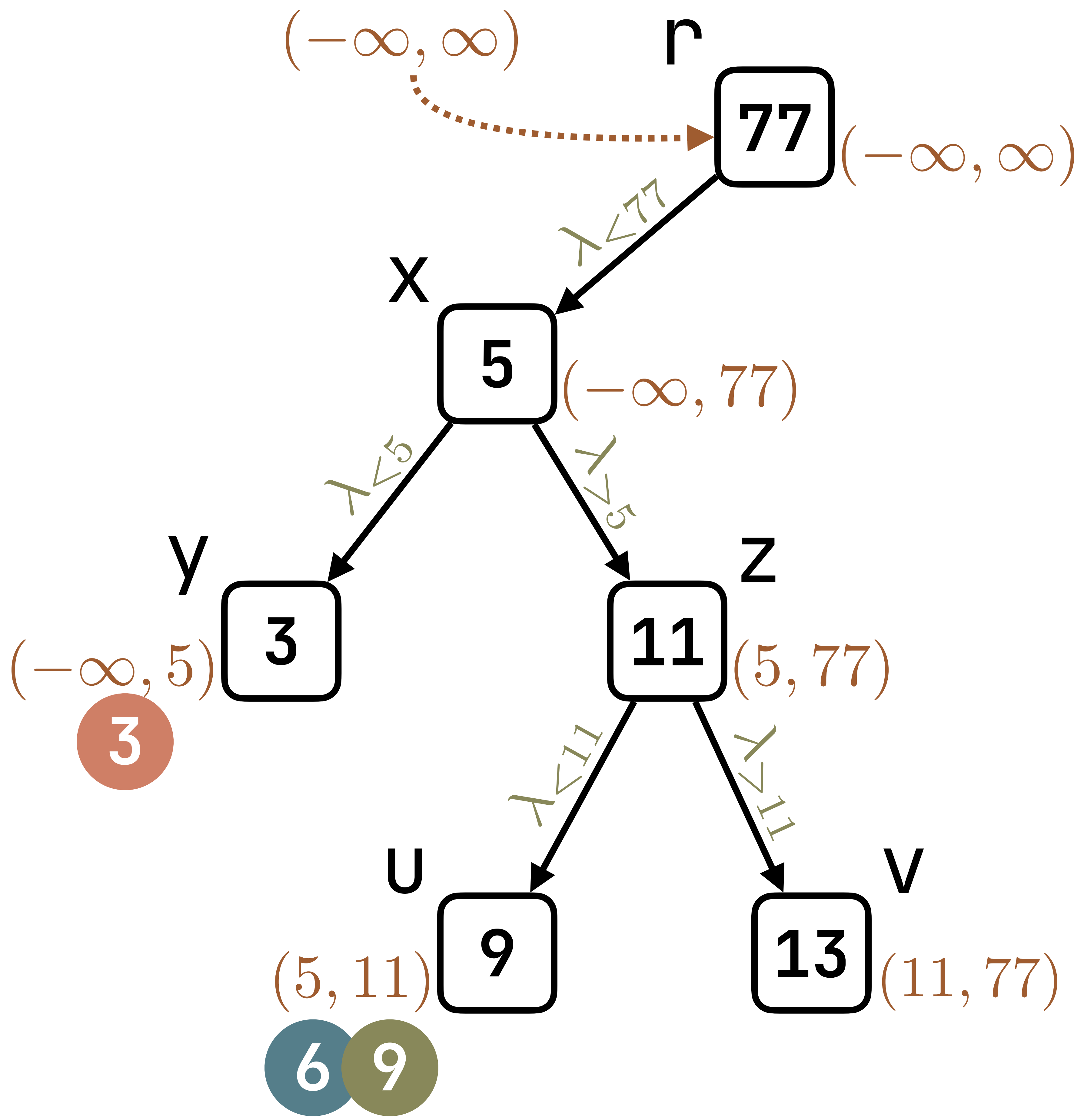
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

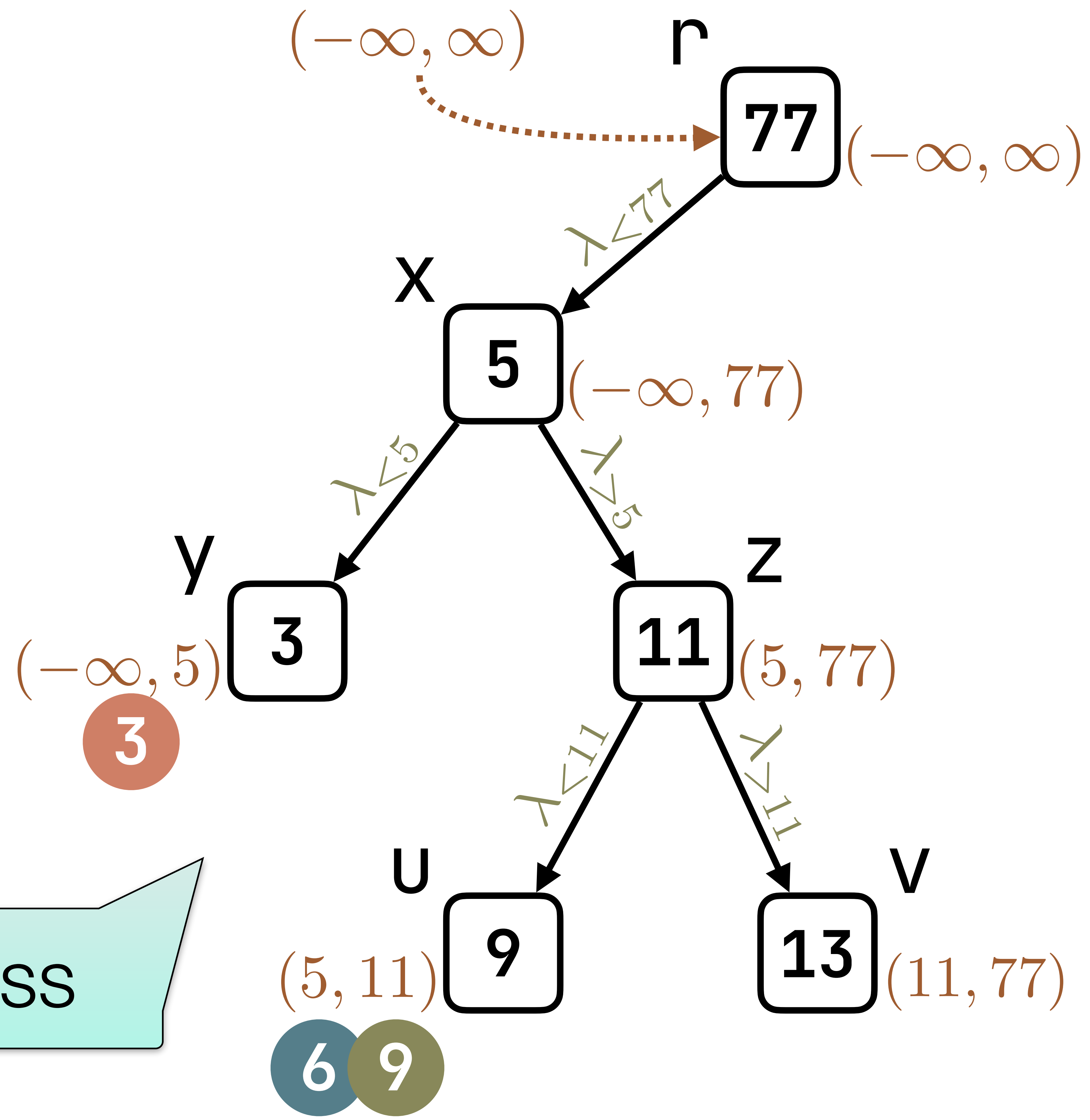
- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \emptyset)$
- Flow propagation via
  $\lambda_{<k} \ := \ \lambda m.\, m \cap (-\infty, k)$
  $\lambda_{>k} \ := \ \lambda m.\, m \cap (k, \infty)$
- ***The flow***:
  least fixed point, wrt. initial value



Flow Framework

— Augment heap graph with ghost state in a dataflow-like fashion —

- Flow values from
  search path monoid $(2^{\mathbb{Z} \cup \{-\infty, \infty\}}, \cup, \emptyset)$
- Flow propagation via
  $$\lambda_{<k} \;:=\; \lambda m.\, m \cap (-\infty, k)$$
  $$\lambda_{>k} \;:=\; \lambda m.\, m \cap (k, \infty)$$
- ***The flow***:
  least fixed point, wrt. initial value



Sufficient information for functional correctness

Flow Framework

- Ghost state for heap graphs
- Inspired by data-flow analysis
- Formalizes inductive heap invariants

Frame Inference

- Separation & flows
- Frame-preserving updates
- Finding footprints algorithmically

Comparing Footprints

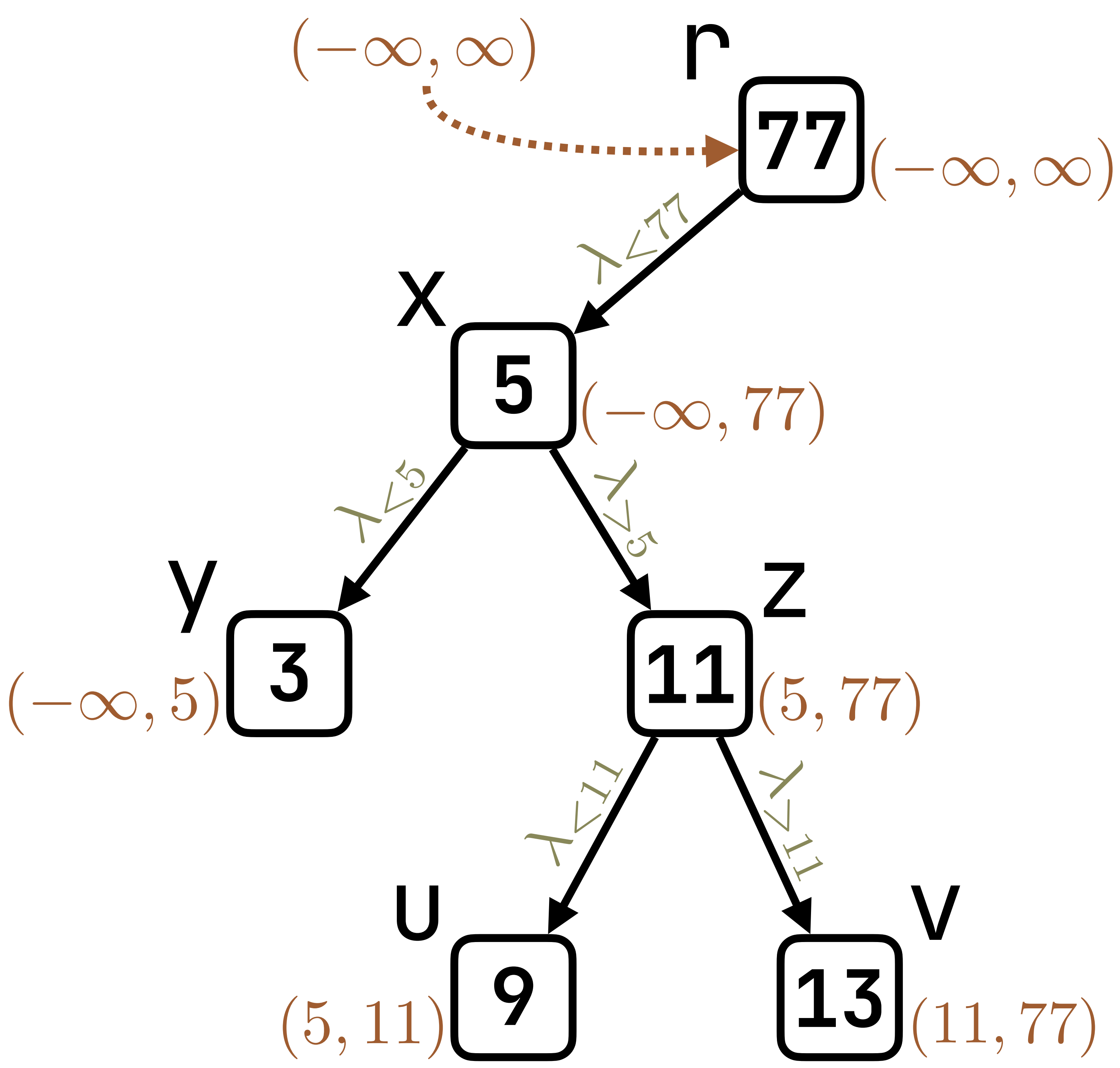
- Check if update is frame-preserving
- Efficient checks for general graphs

Separation

- Flows graphs form a separation algebra
    - ➡ framing = "cutting the graph & flow"

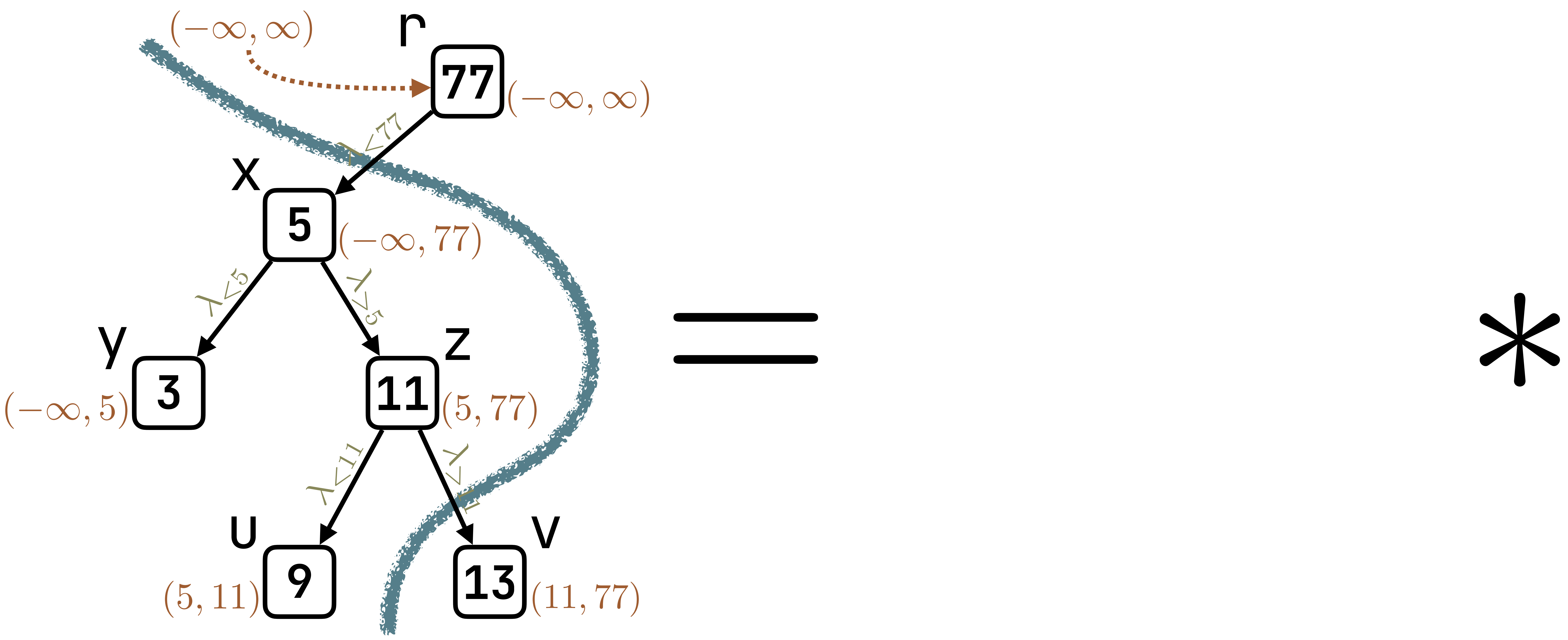
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



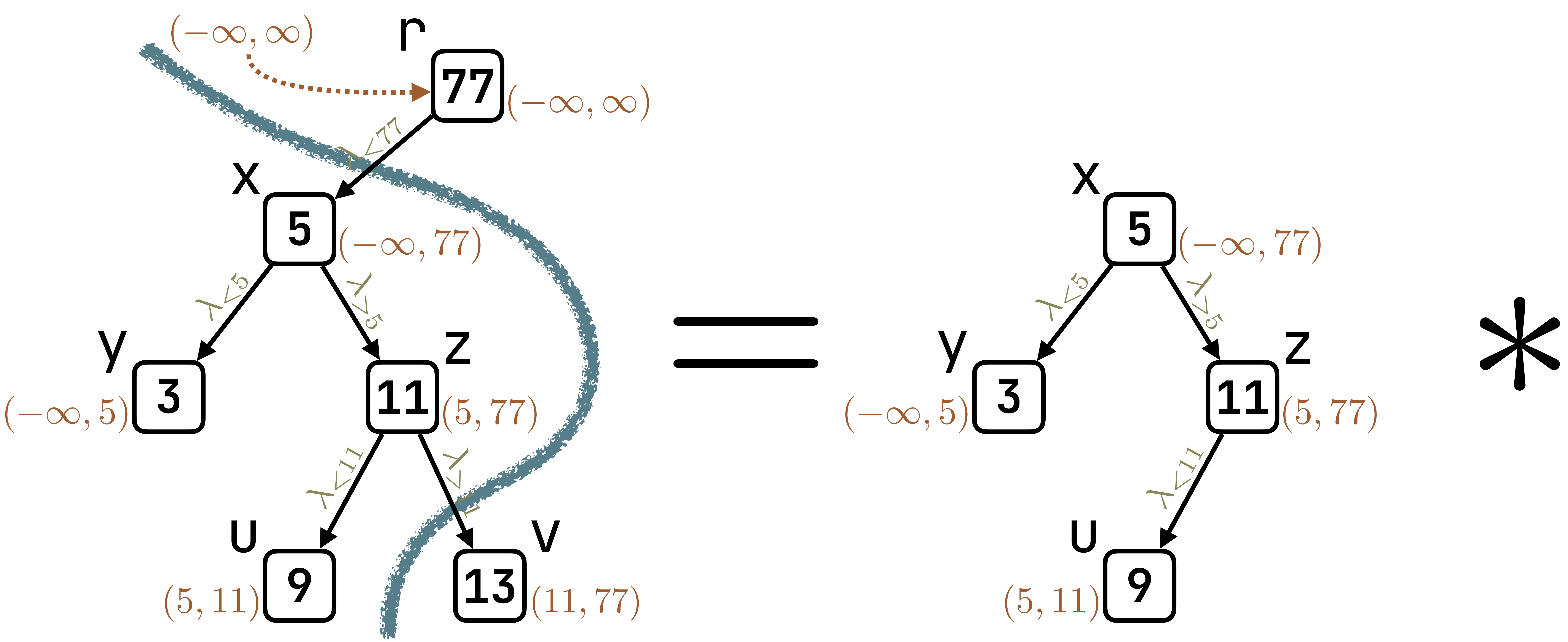
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



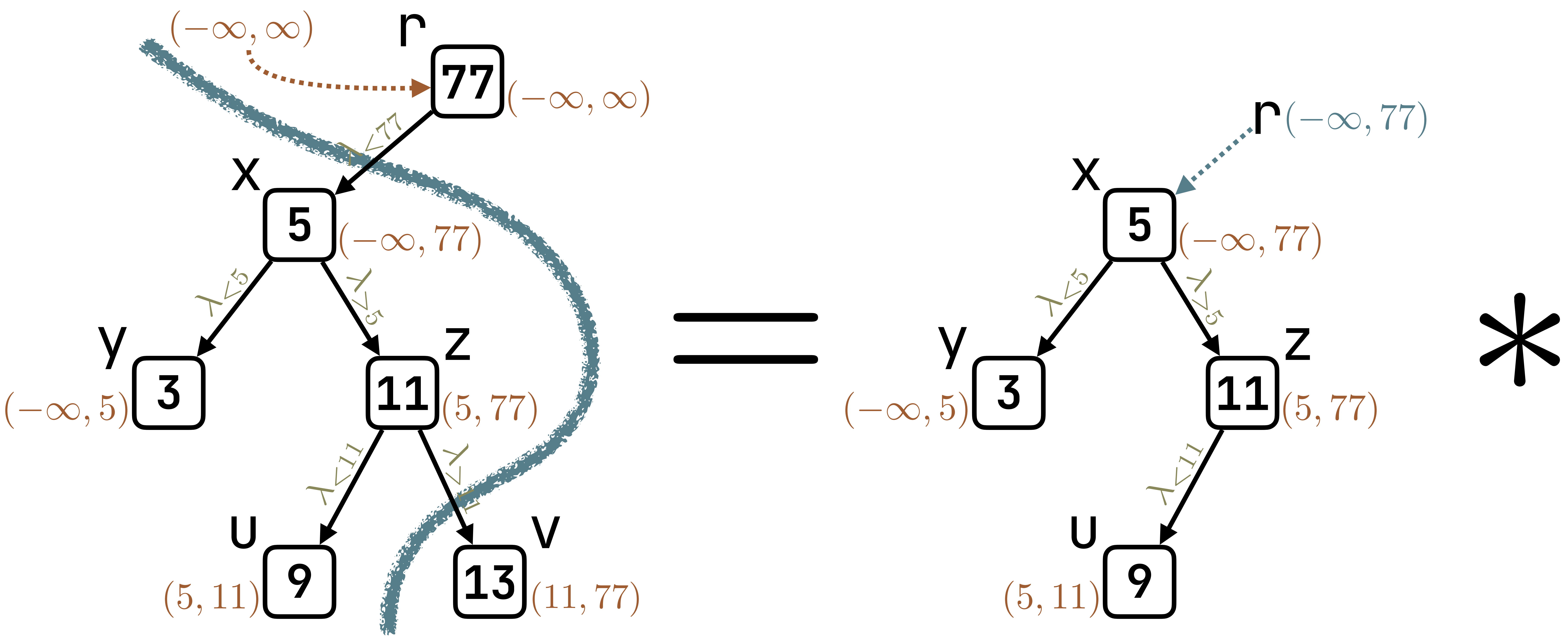
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



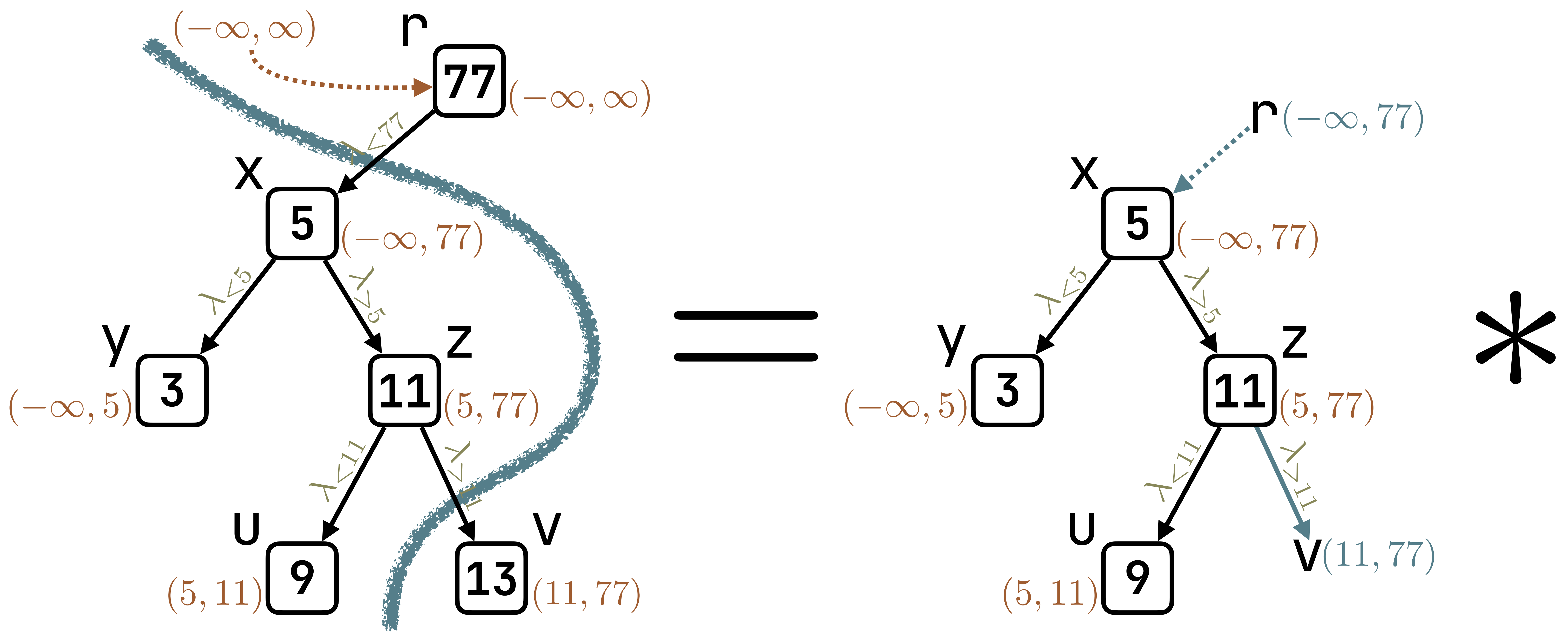
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



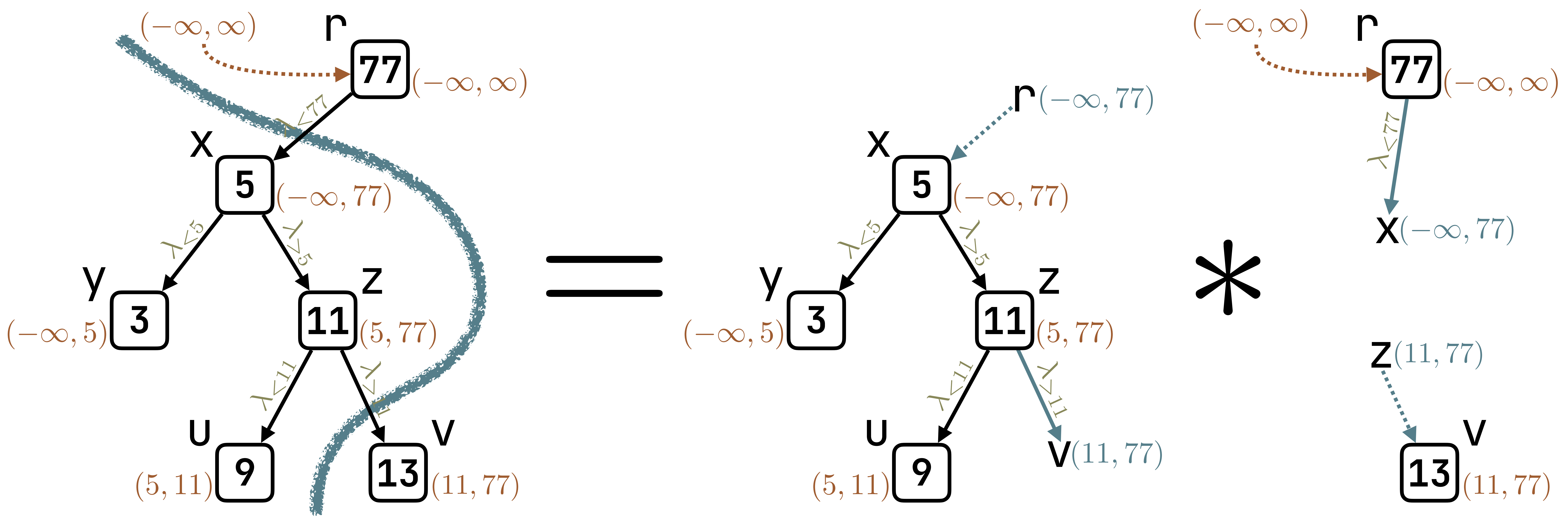
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



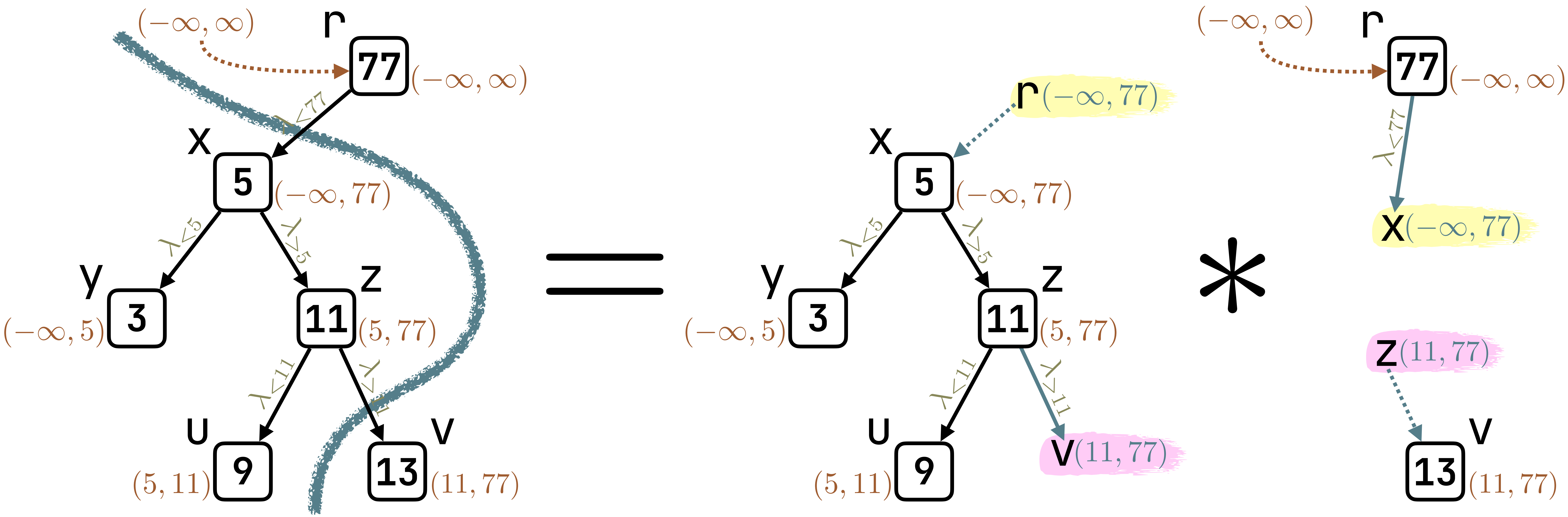
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



Separation

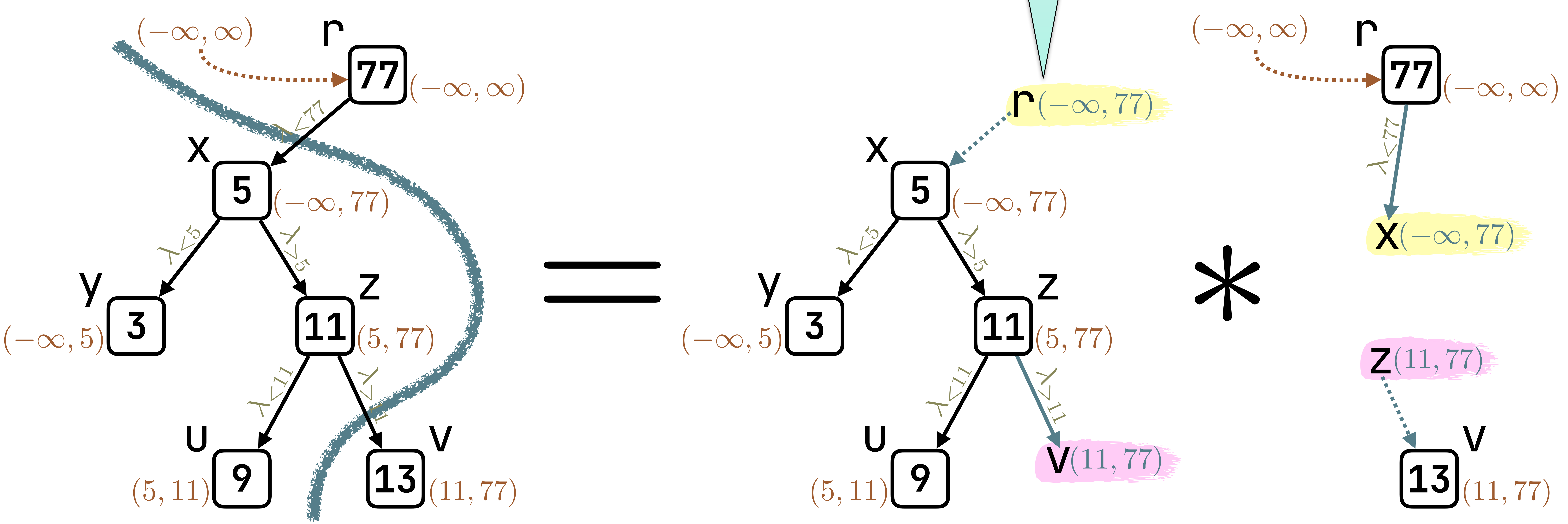
- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"



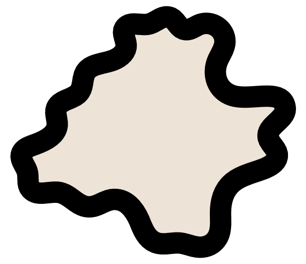
Separation

- Flows graphs form a separation algebra
  - ➡ framing = "cutting the graph & flow"

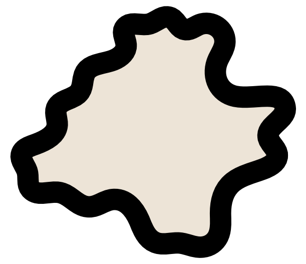
Composition $*$ defined only if inflow & outflow match.

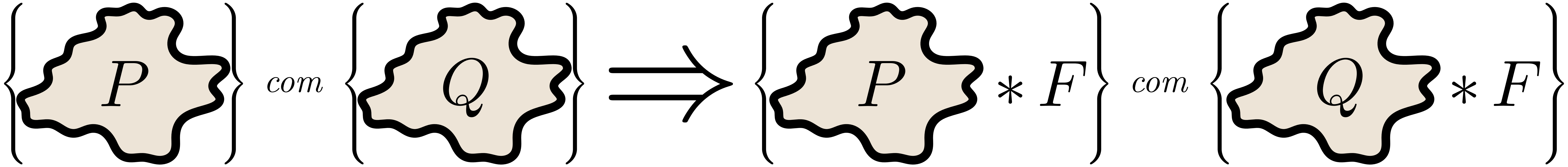


Frame-preserving Updates

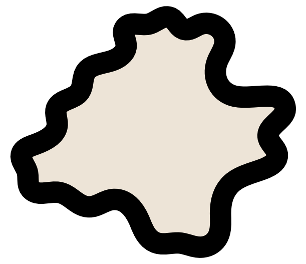
- Footprint
  - the region  affected by an update

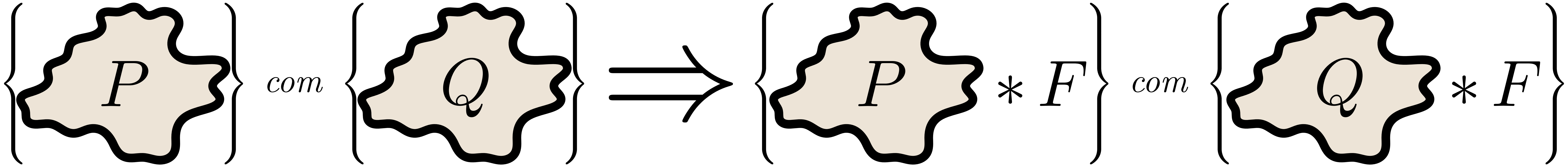
Frame-preserving Updates

- Footprint
  - the region  affected by an update
  - must be frame-preserving (not affect the frame):

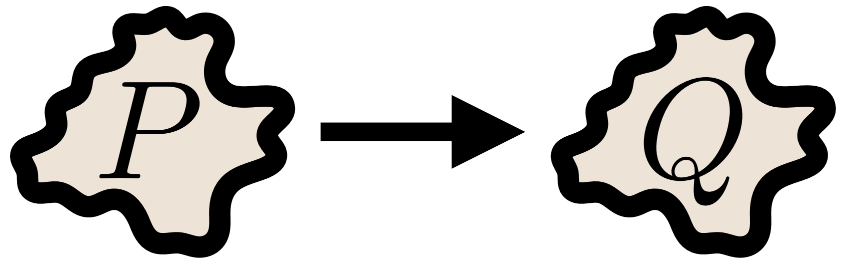


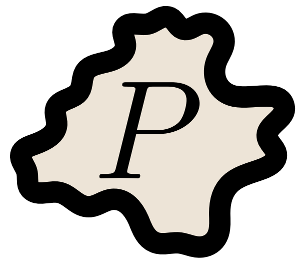
Frame-preserving Updates

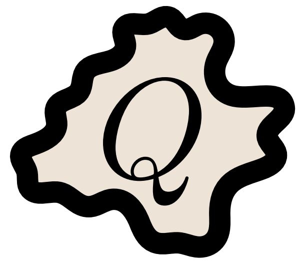
- Footprint
  - the region  affected by an update
  - must be frame-preserving (not affect the frame):



- Theorem:

  Update 

  is frame-preserving if 

  and 

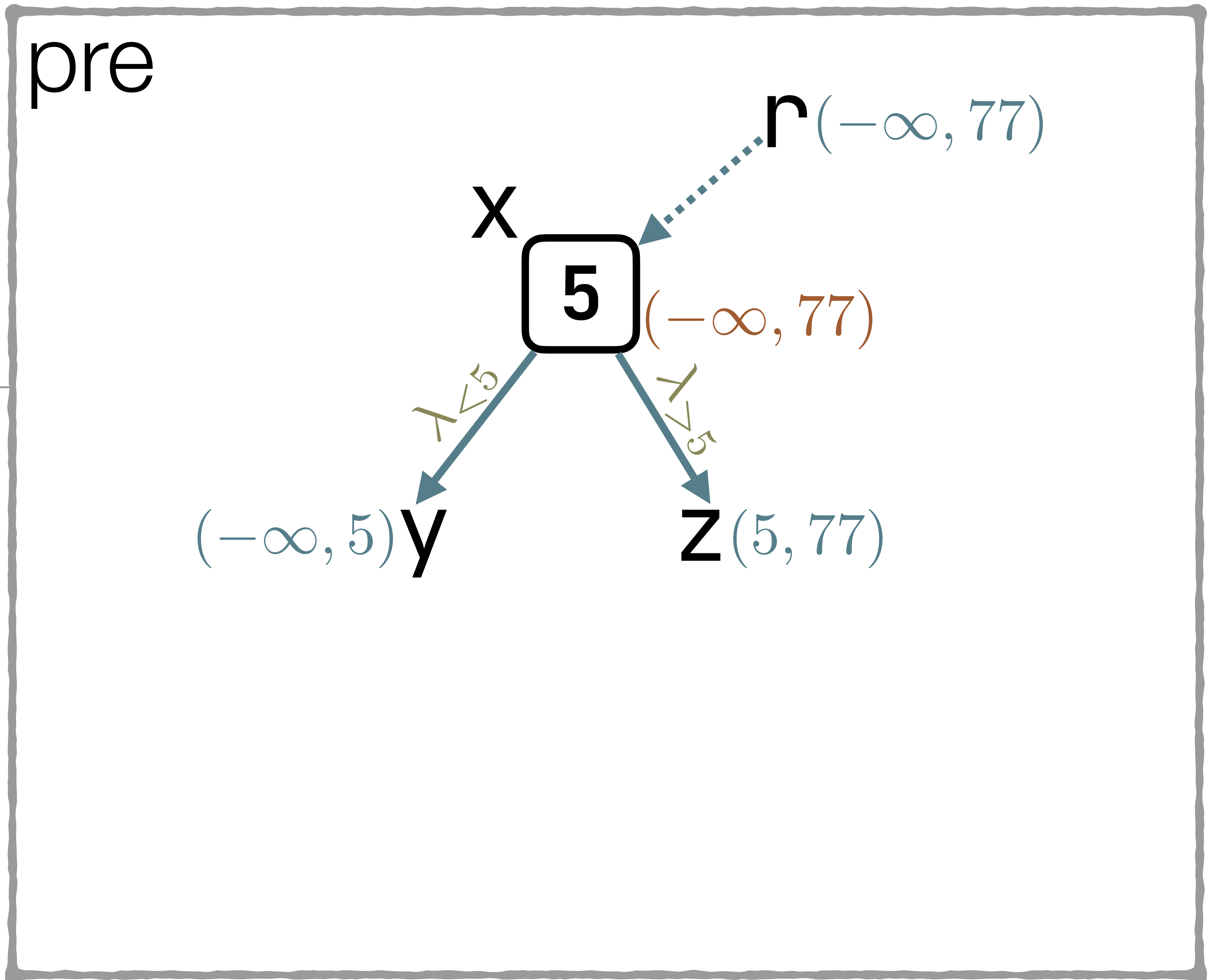
  have the same outflow, *for all inflows*.

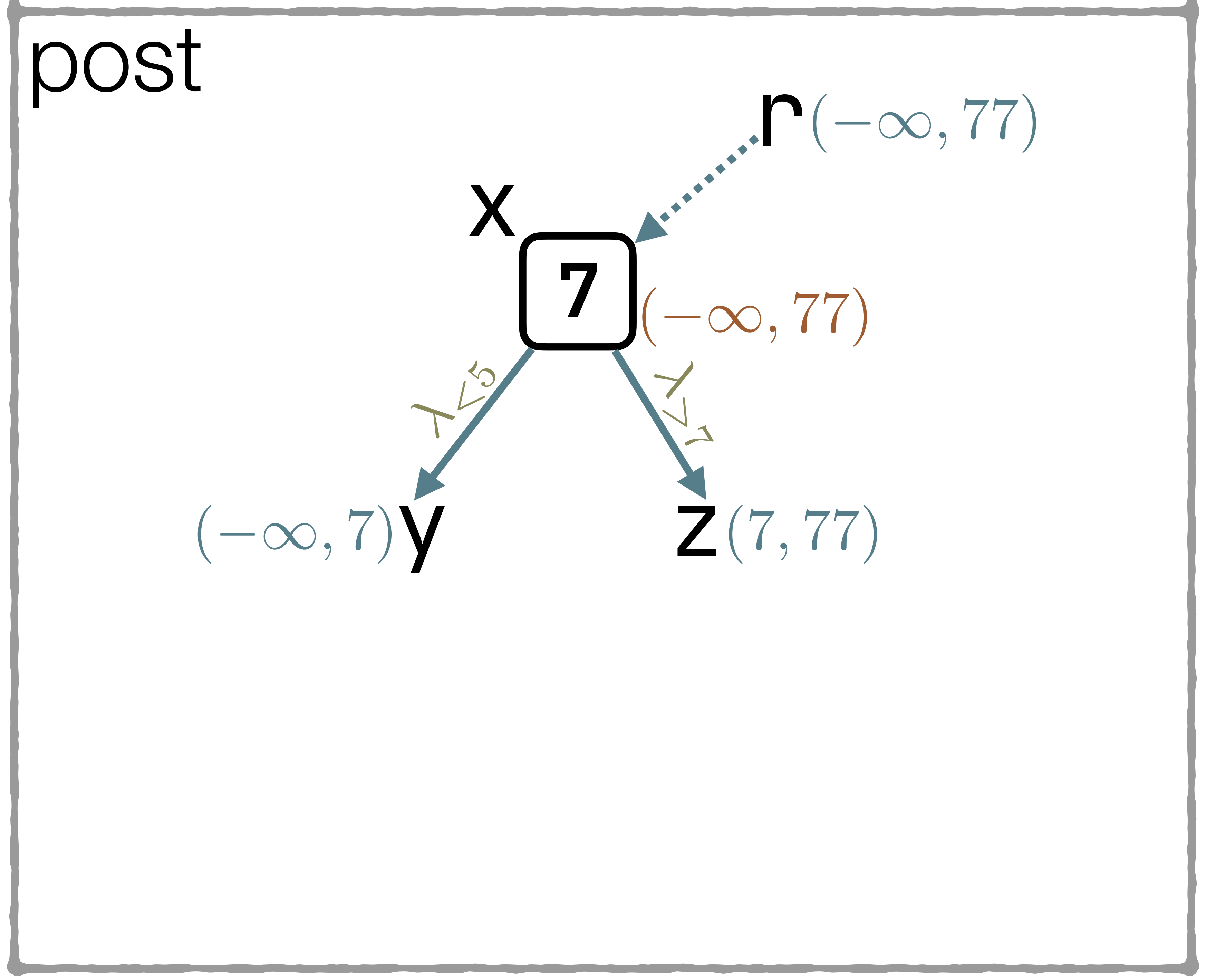
Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point

Finding Footprints

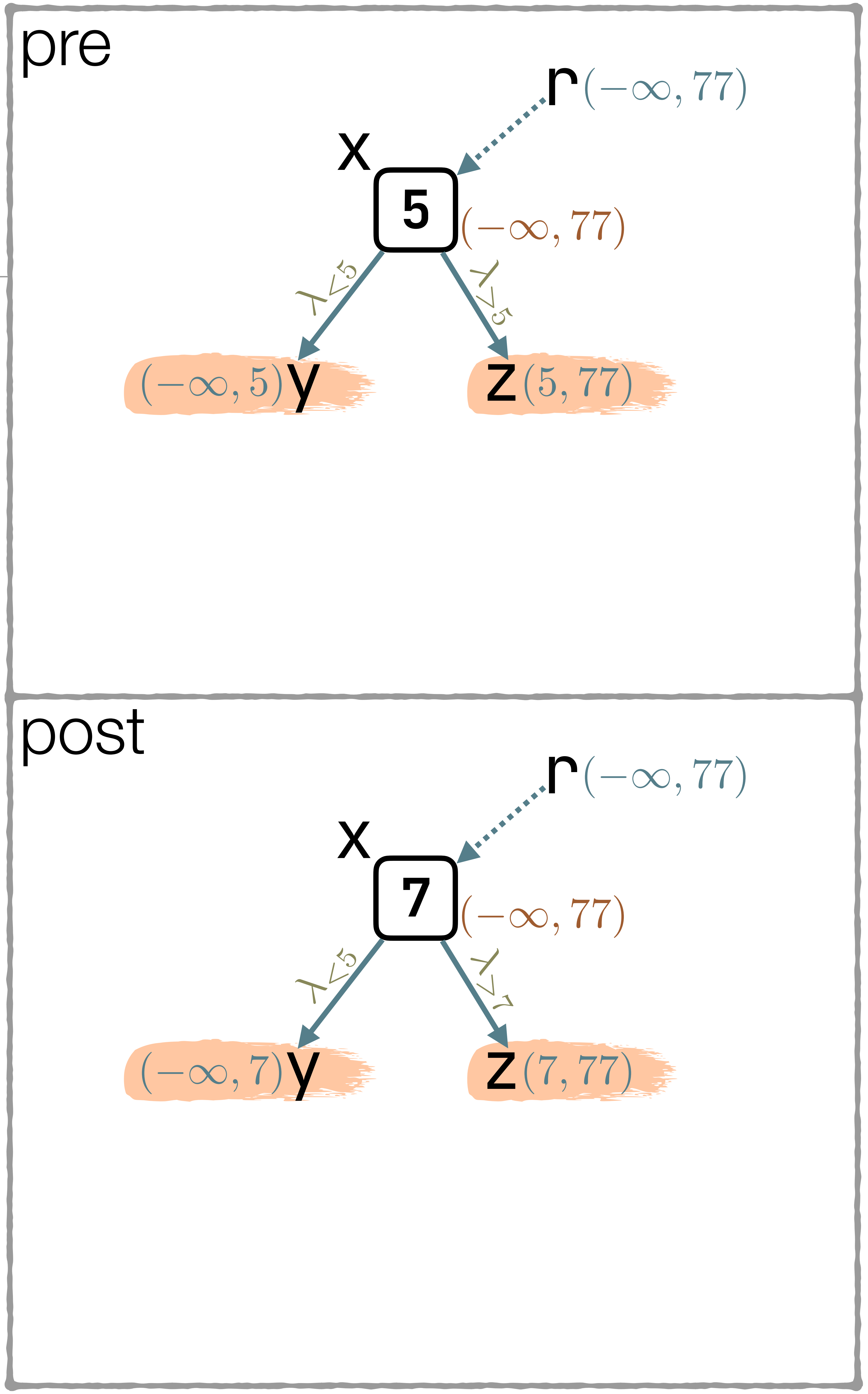
- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point





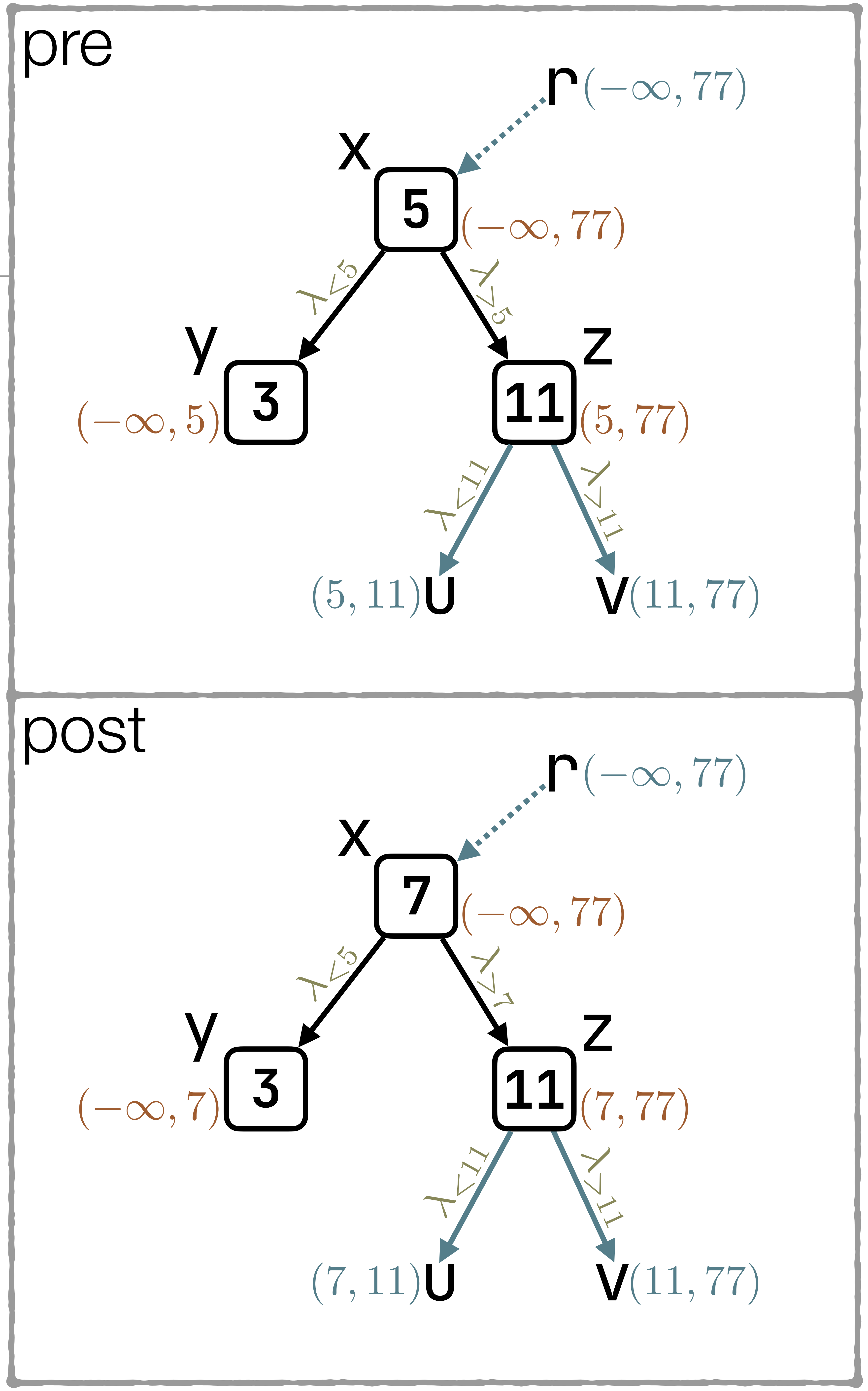
Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point



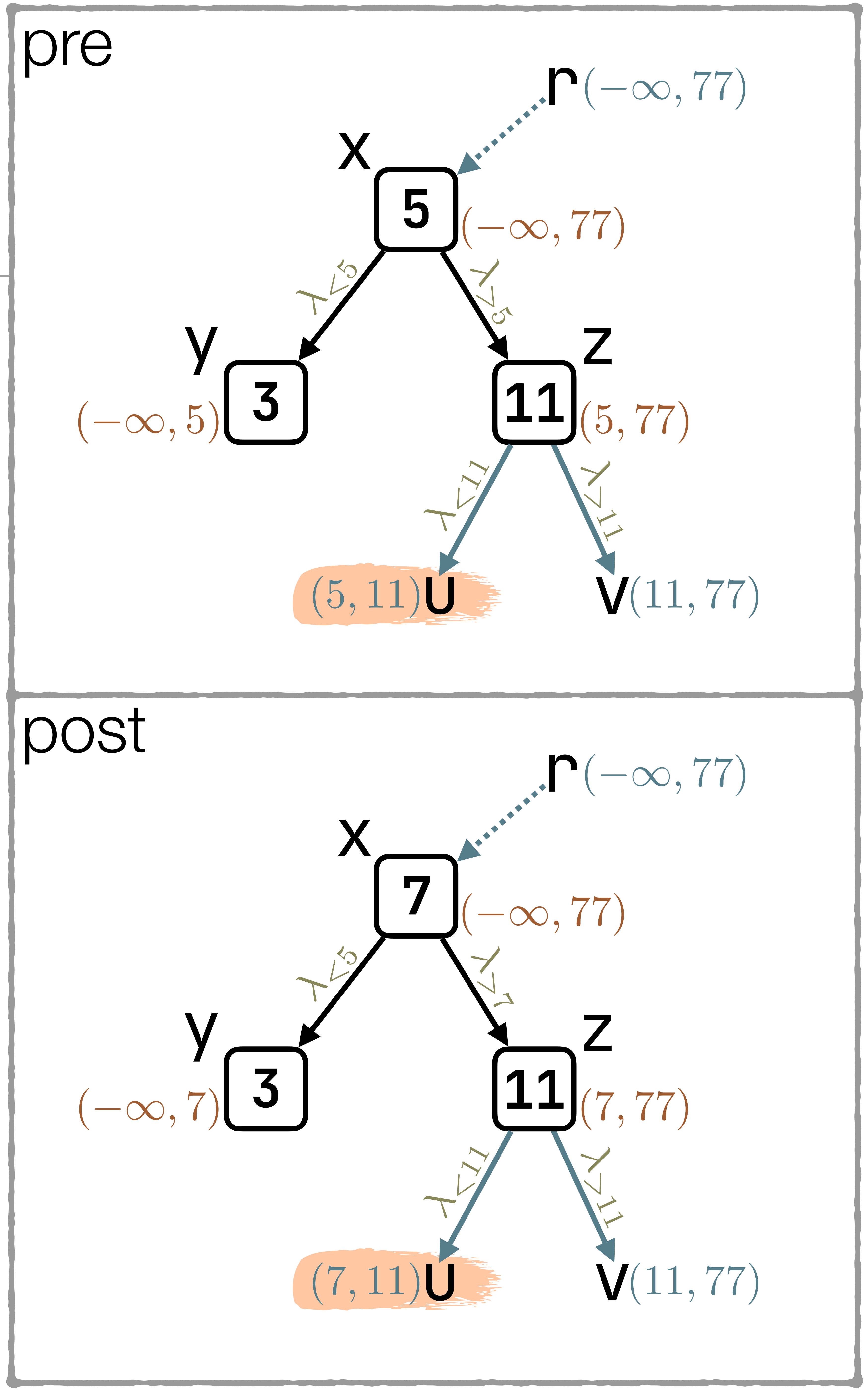
Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point



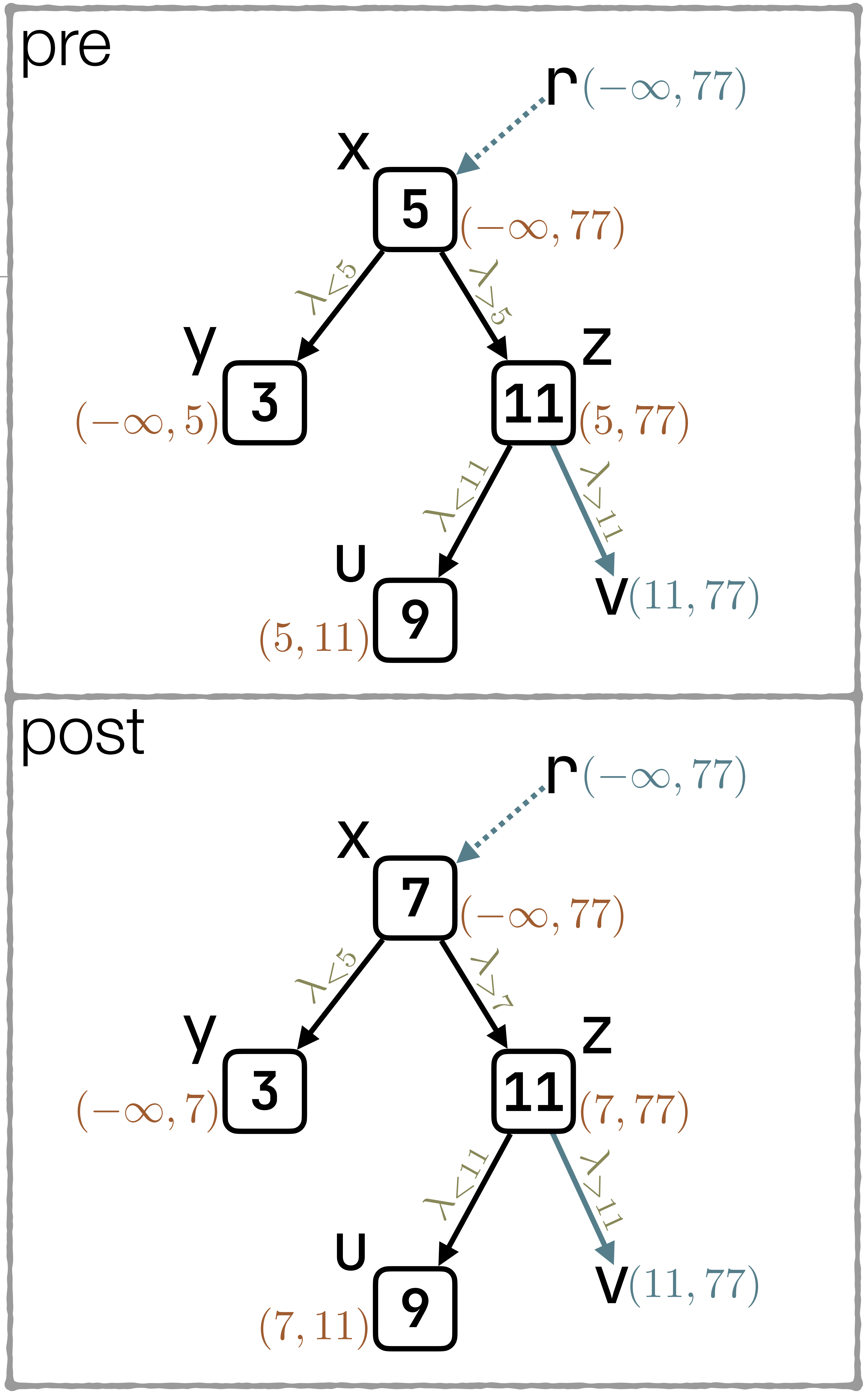
Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point



Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point



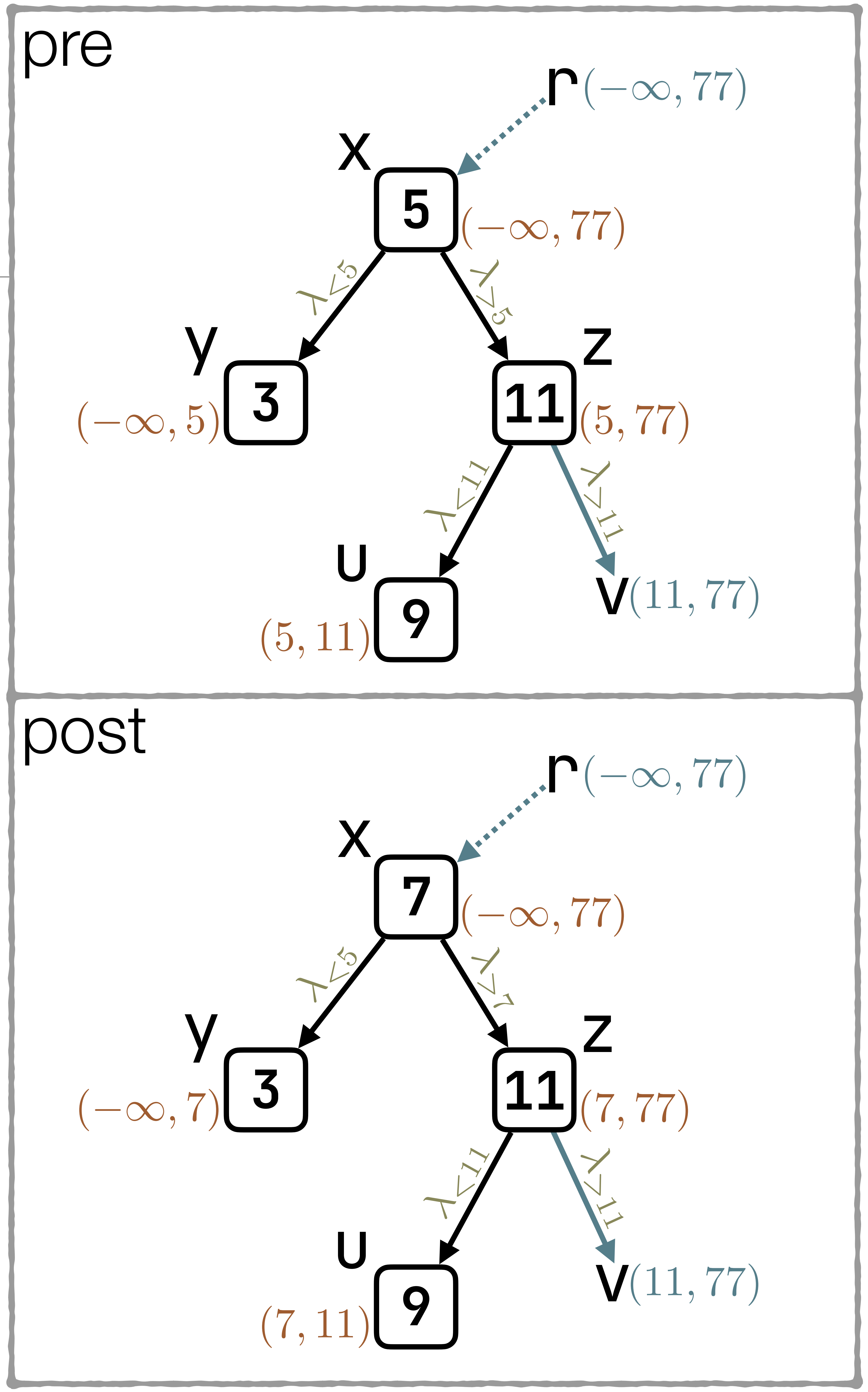
Finding Footprints

- Algorithm:
  1. add physical footprint
  2. compute outflow (for all inflows)
  3. add nodes if pre/post outflow differs
  4. repeat until fixed point

Not minimal.

Incomplete.

Works well in practice.



Flow Framework

- Ghost state for heap graphs
- Inspired by data-flow analysis
- Formalizes inductive heap invariants

Frame Inference

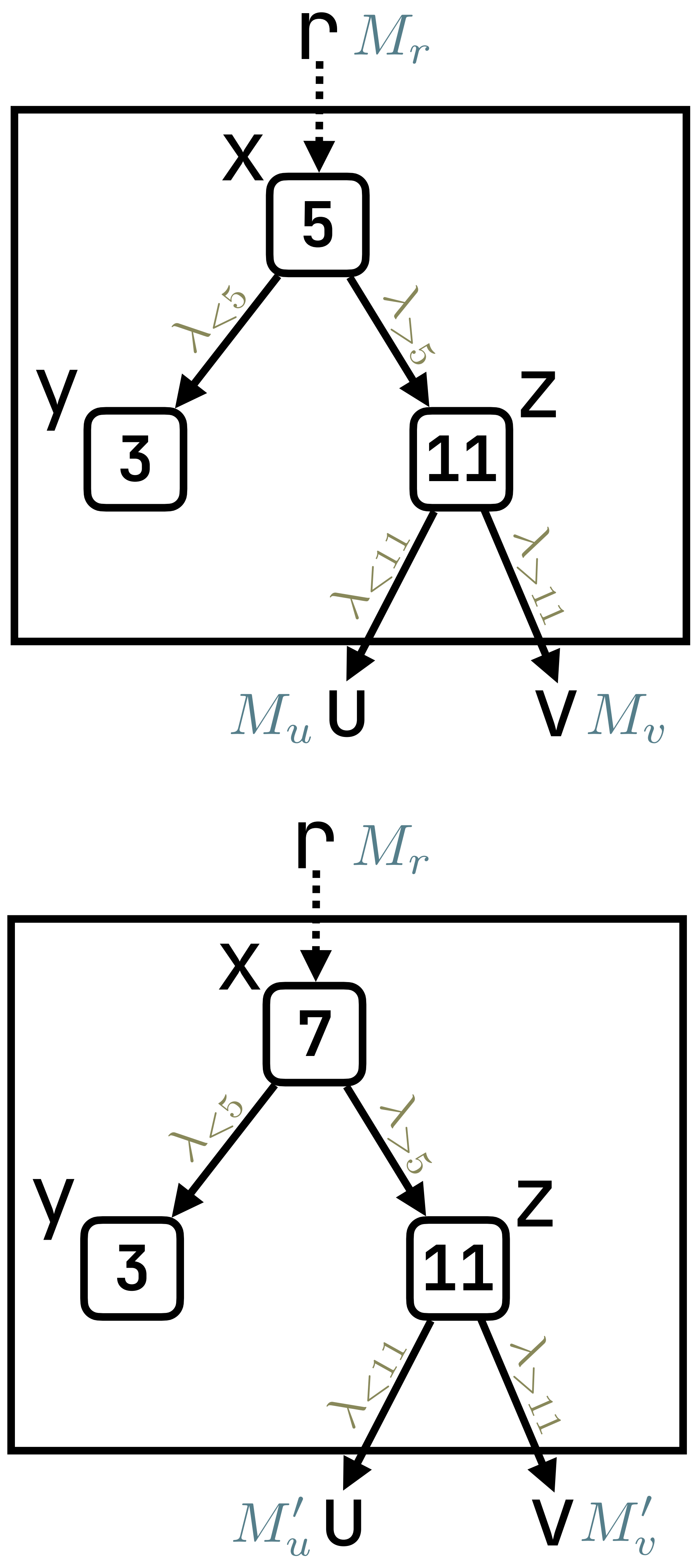
- Separation & flows
- Frame-preserving updates
- Finding footprints algorithmically

Comparing Footprints

- Check if update is frame-preserving
- Efficient checks for general graphs

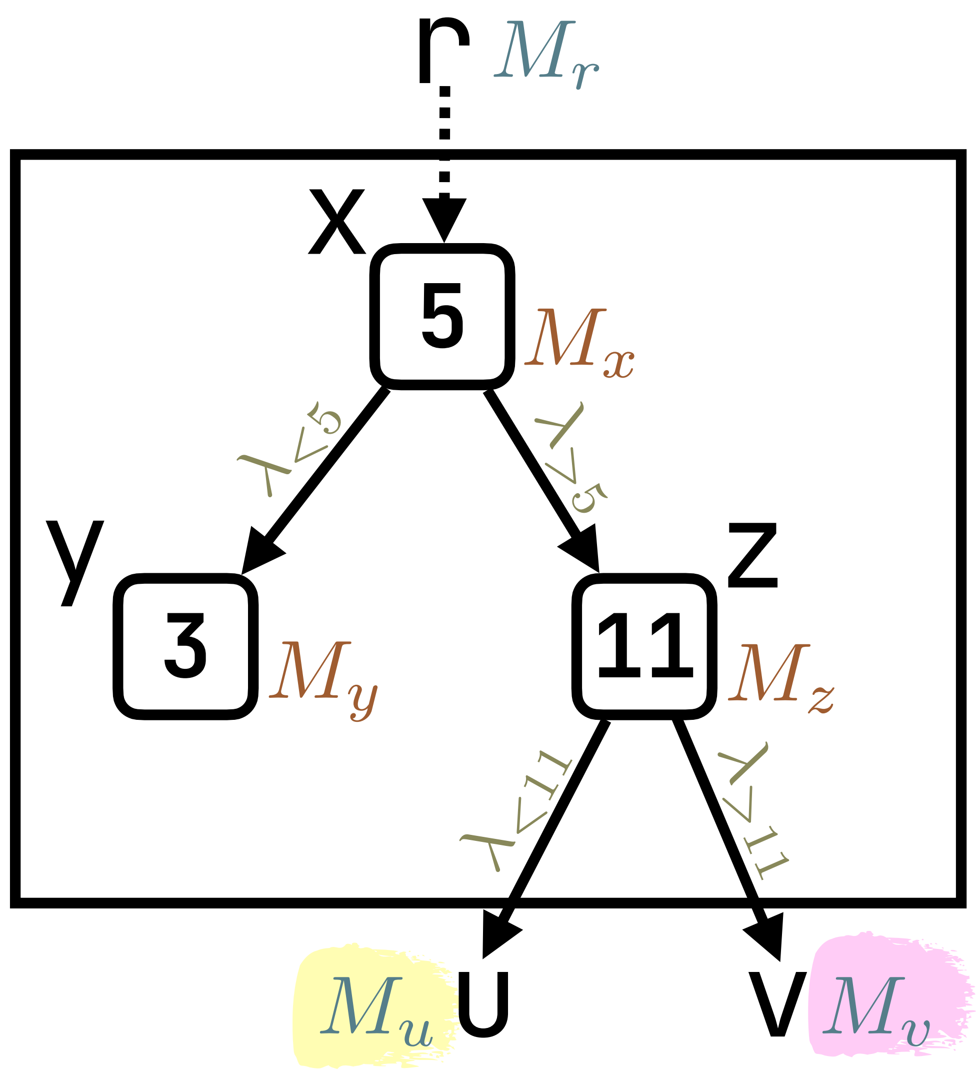
Overview

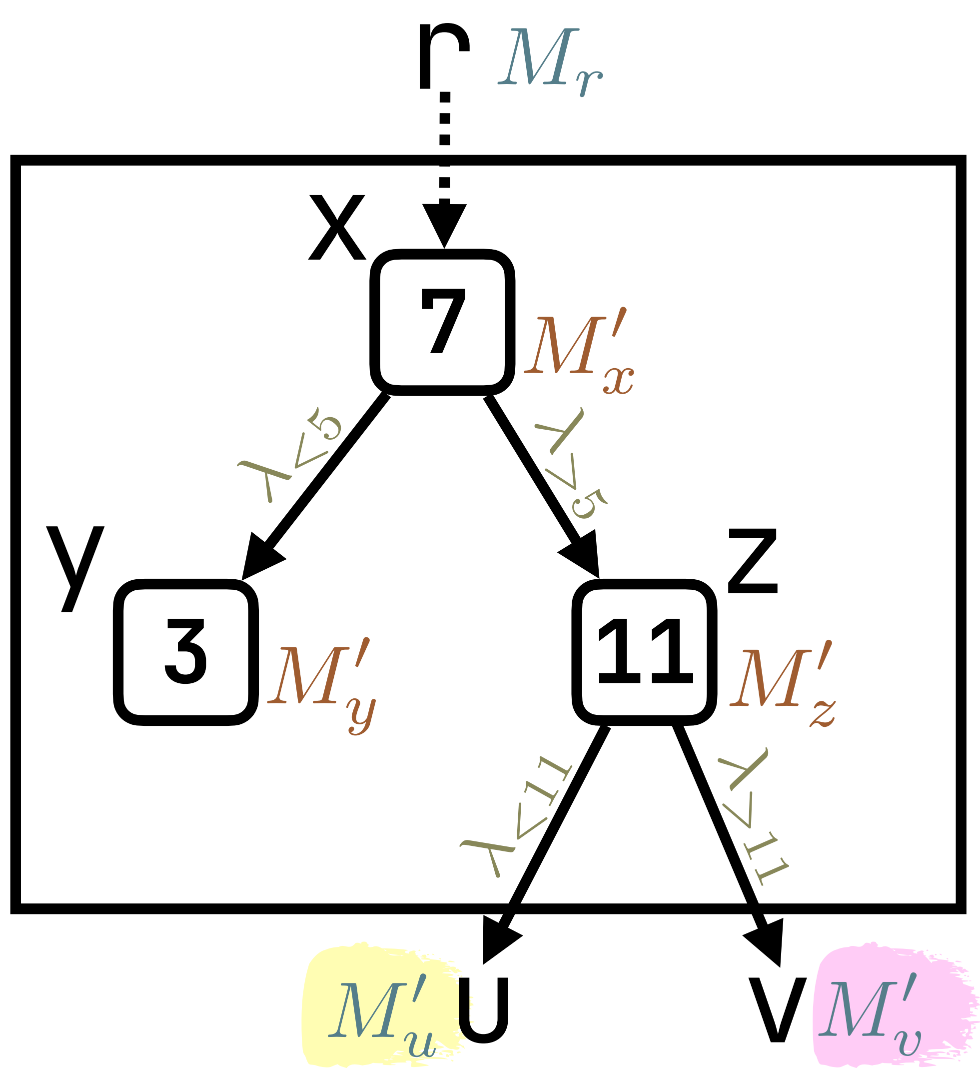
- Question: does $M_u = M'_u$ and $M_v = M'_v$ hold?



Overview

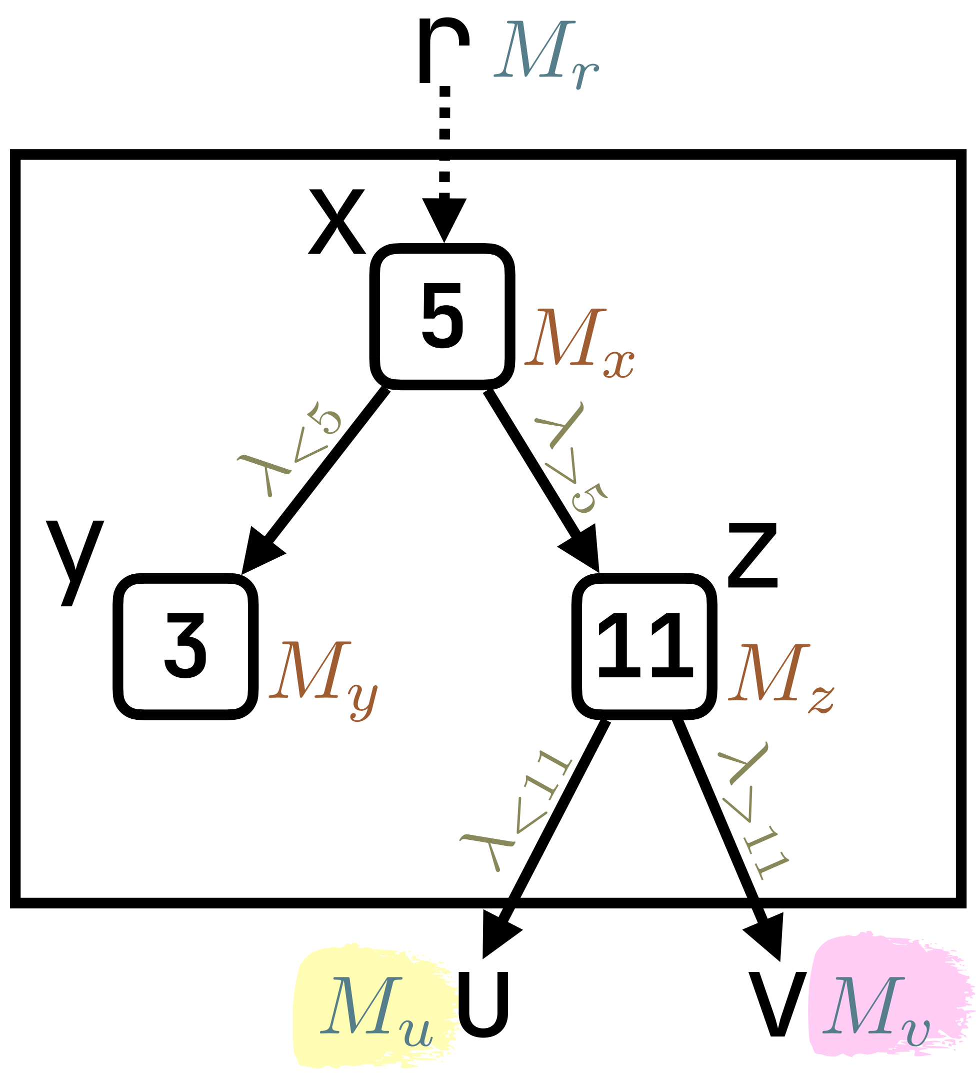
- Question: does $M_u = M'_u$ and $M_v = M'_v$ hold?
- Naive: compute fixed point (over functions), then check

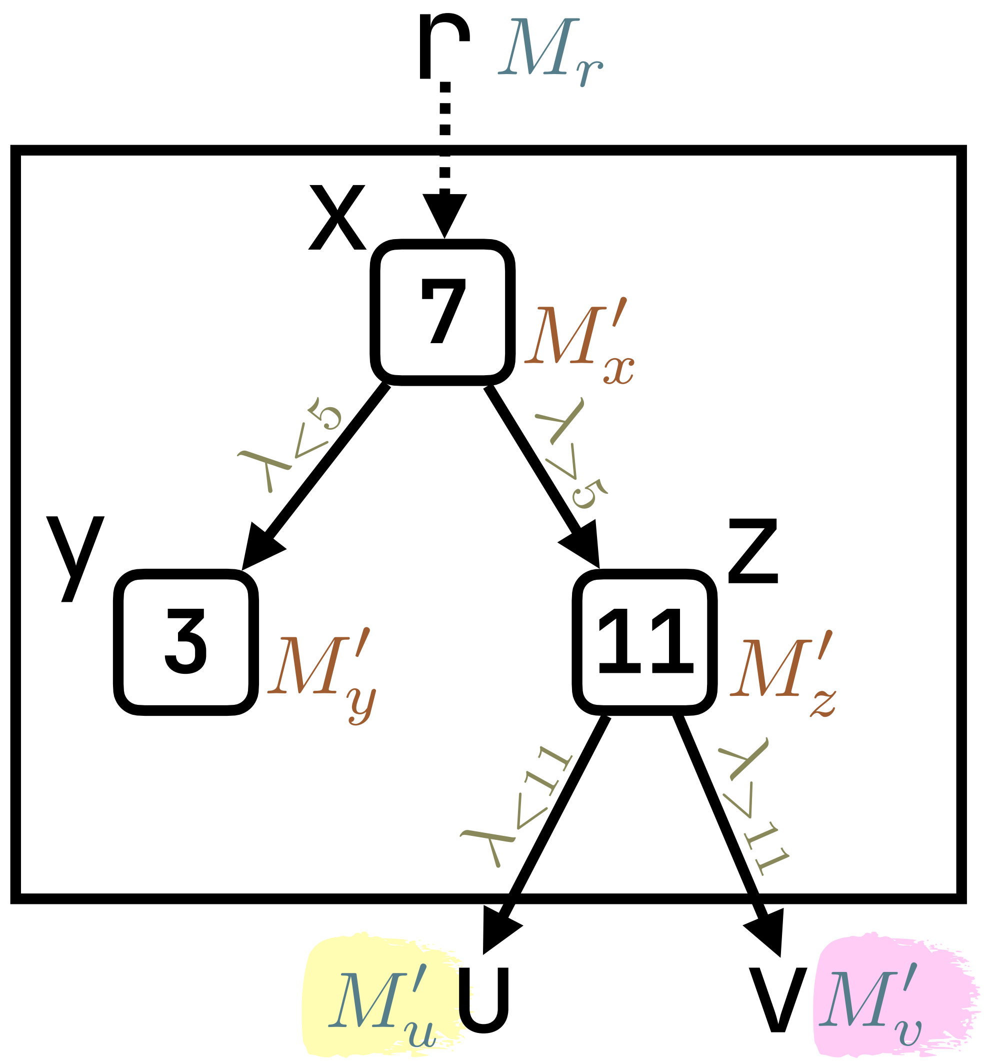




Overview

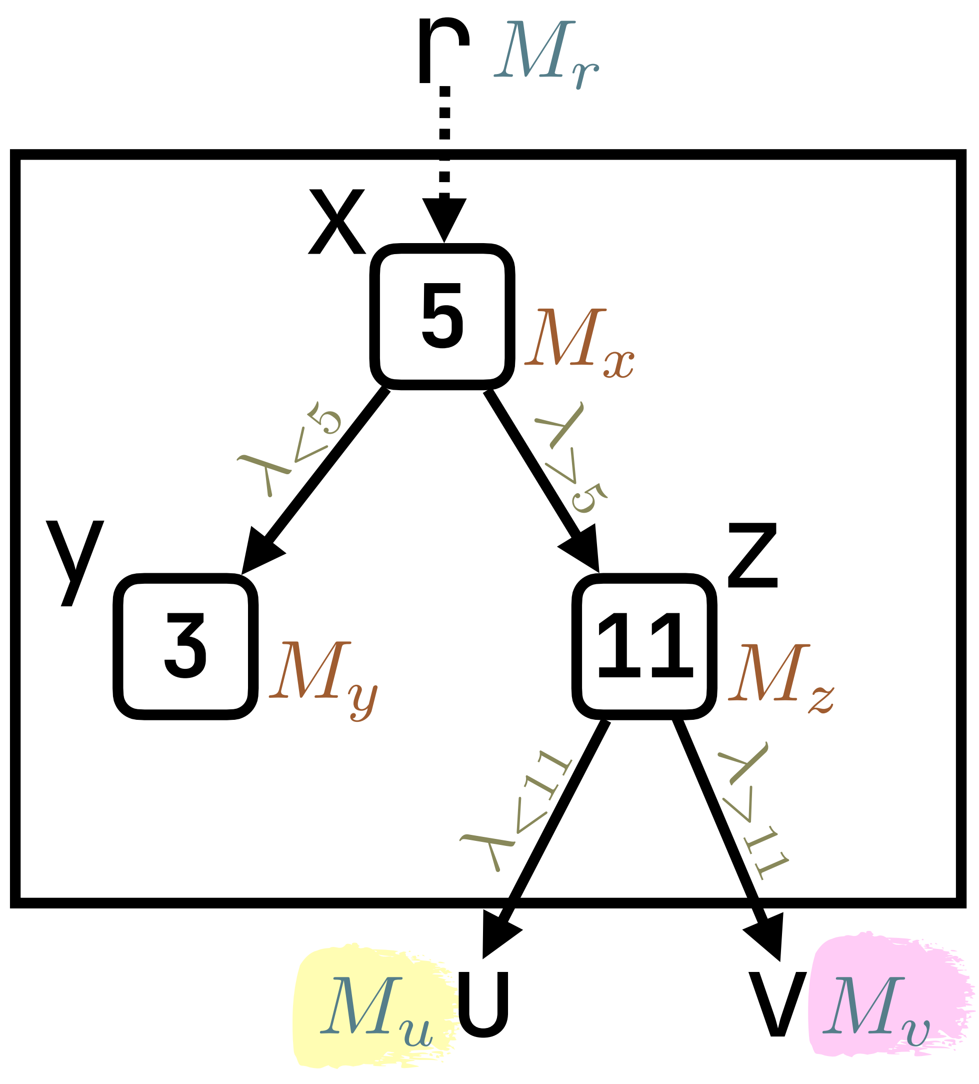
- Question: does $M_u = M'_u$ and $M_v = M'_v$ hold?
- Naive: compute fixed point (over functions), then check
- Challenges:
    1. fixed point might require "infinite" Kleene iteration
    2. no restriction on graph structure
    3. assertions denote (infinitely) many heap graphs

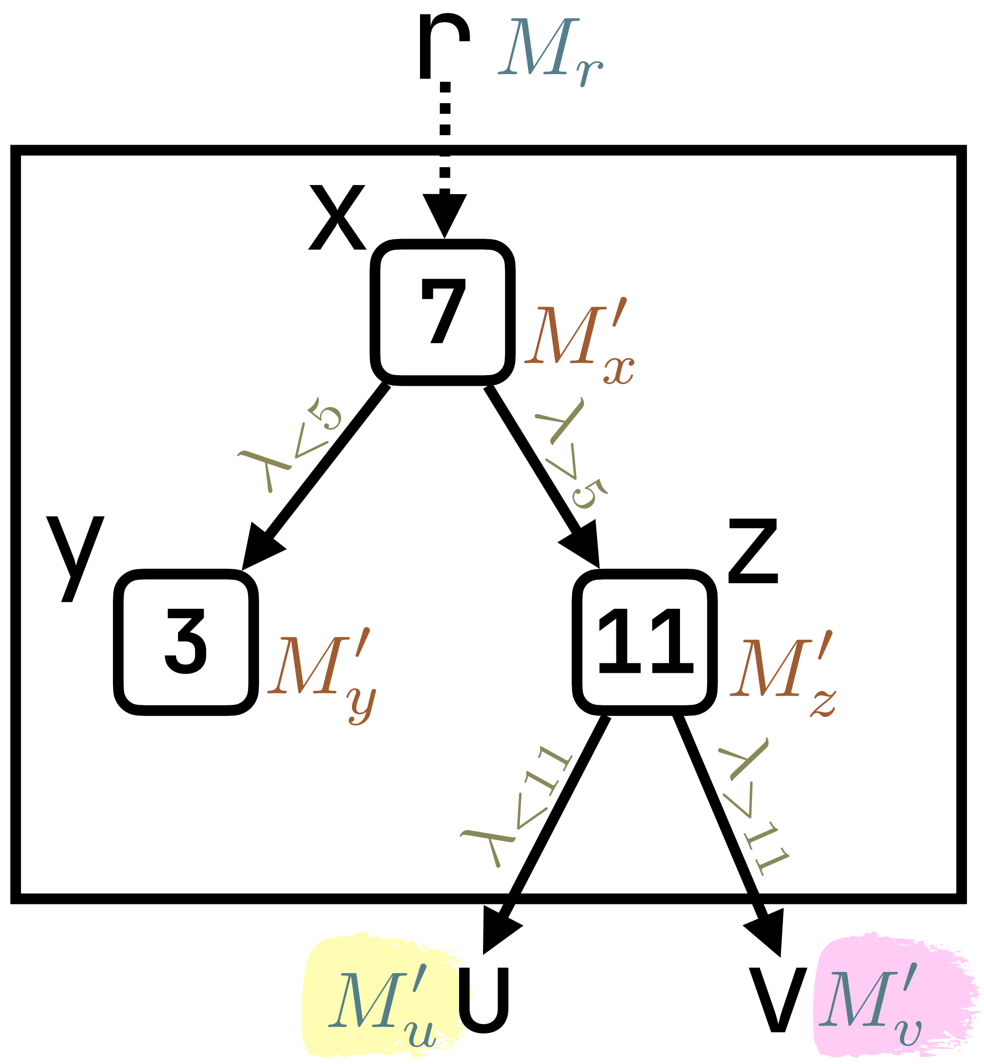




Overview

- Question: does $M_u = M'_u$ and $M_v = M'_v$ hold?
- Naive: compute fixed point (over functions), then check
- Challenges:
    1. fixed point might require "infinite" Kleene iteration
    2. no restriction on graph structure
    3. assertions denote (infinitely) many heap graphs
- Goal: automated & efficient approach

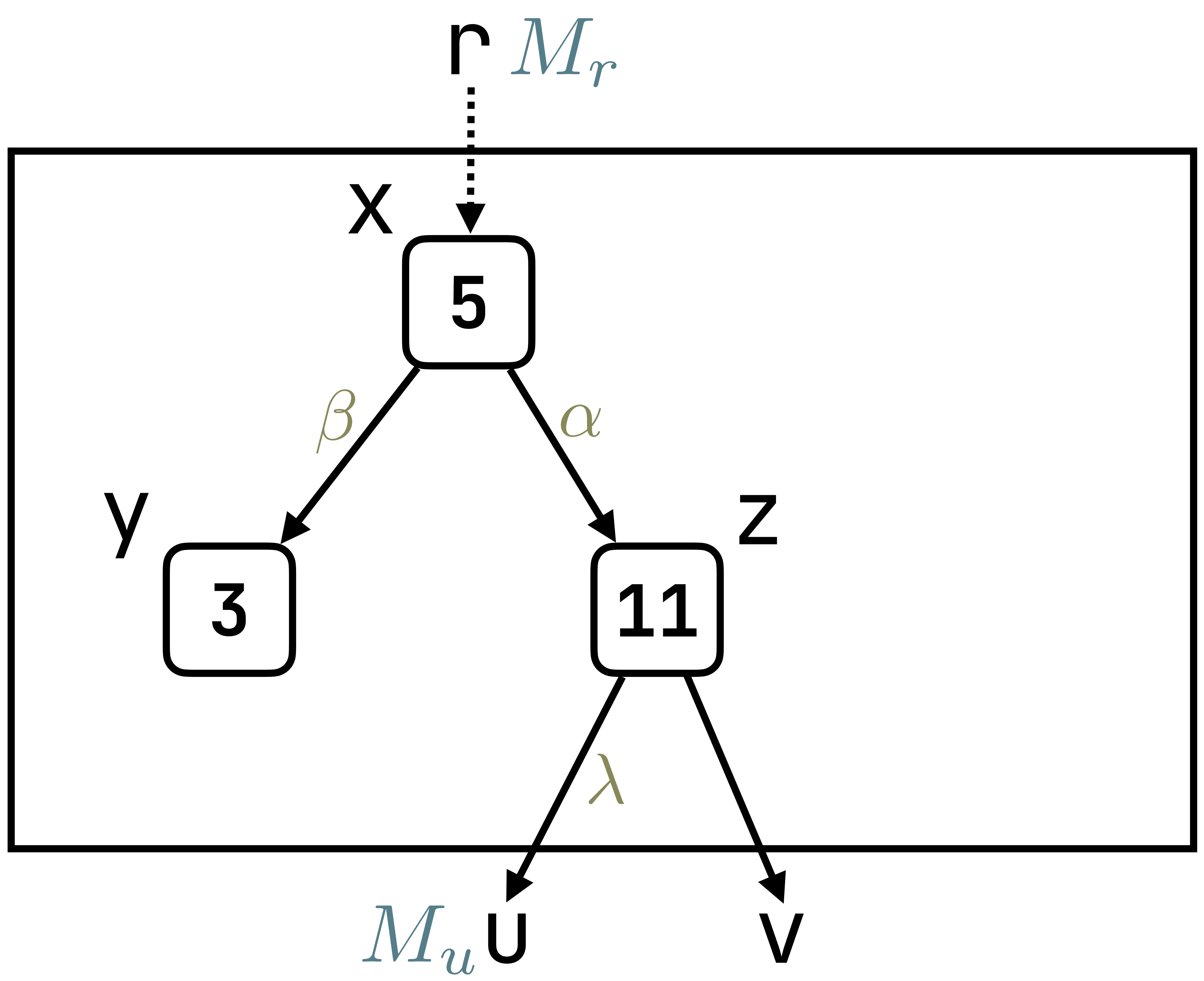




Avoiding Fixed Points

- Observation for trees:

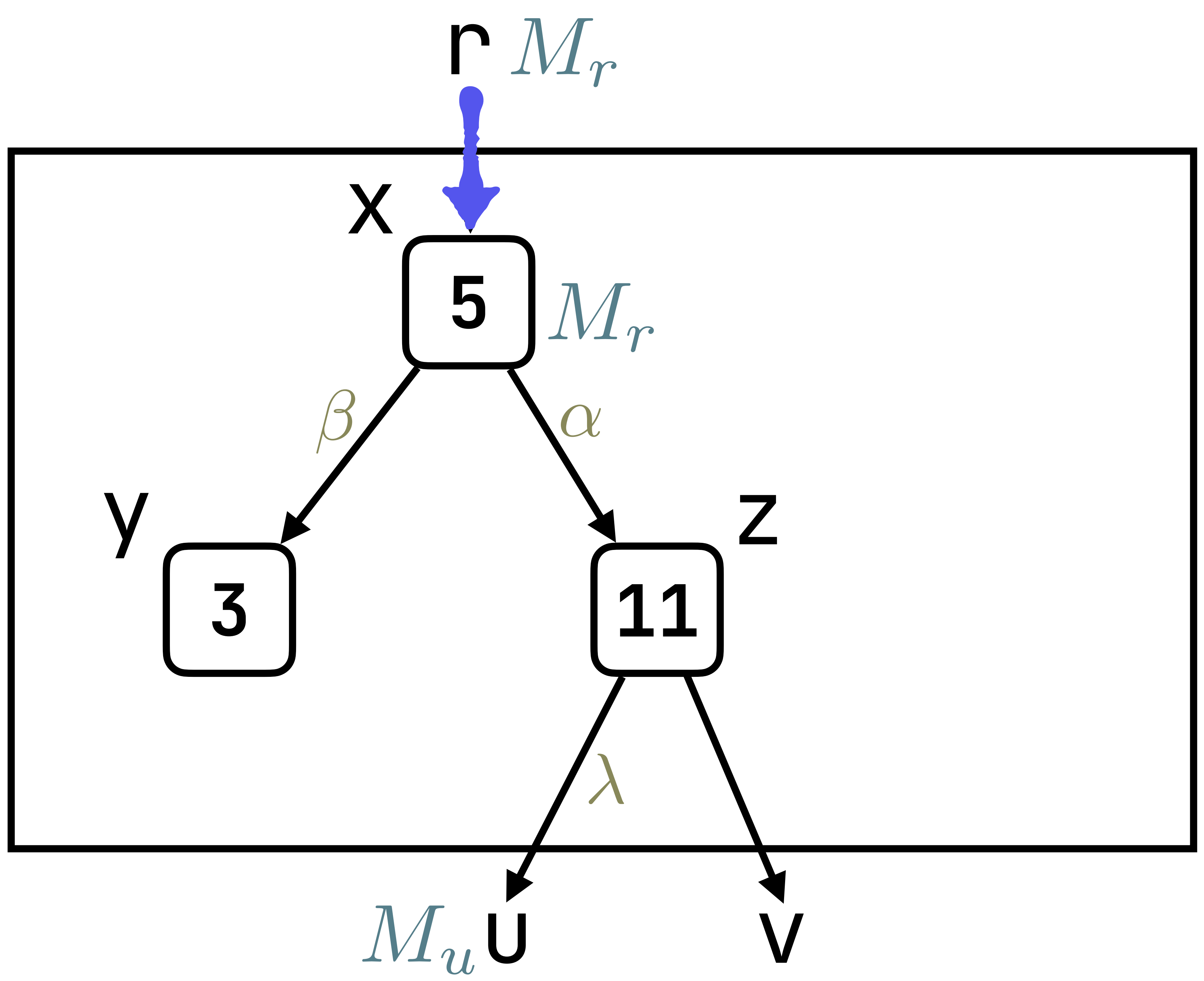
  fixed point = concatenation of edge functions along path



Avoiding Fixed Points

- Observation for trees:

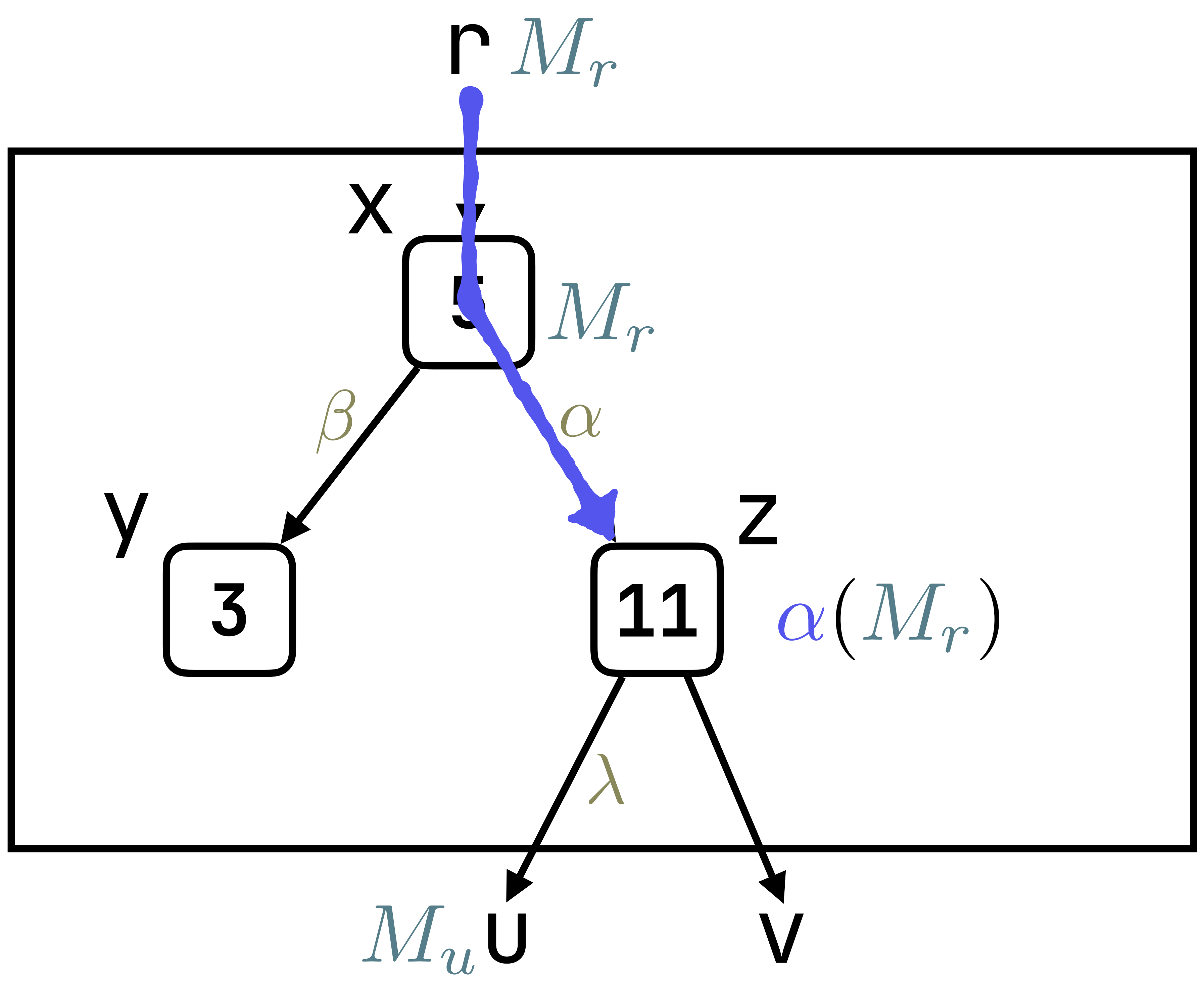
fixed point = *concatenation of edge functions along path*



Avoiding Fixed Points

- Observation for trees:

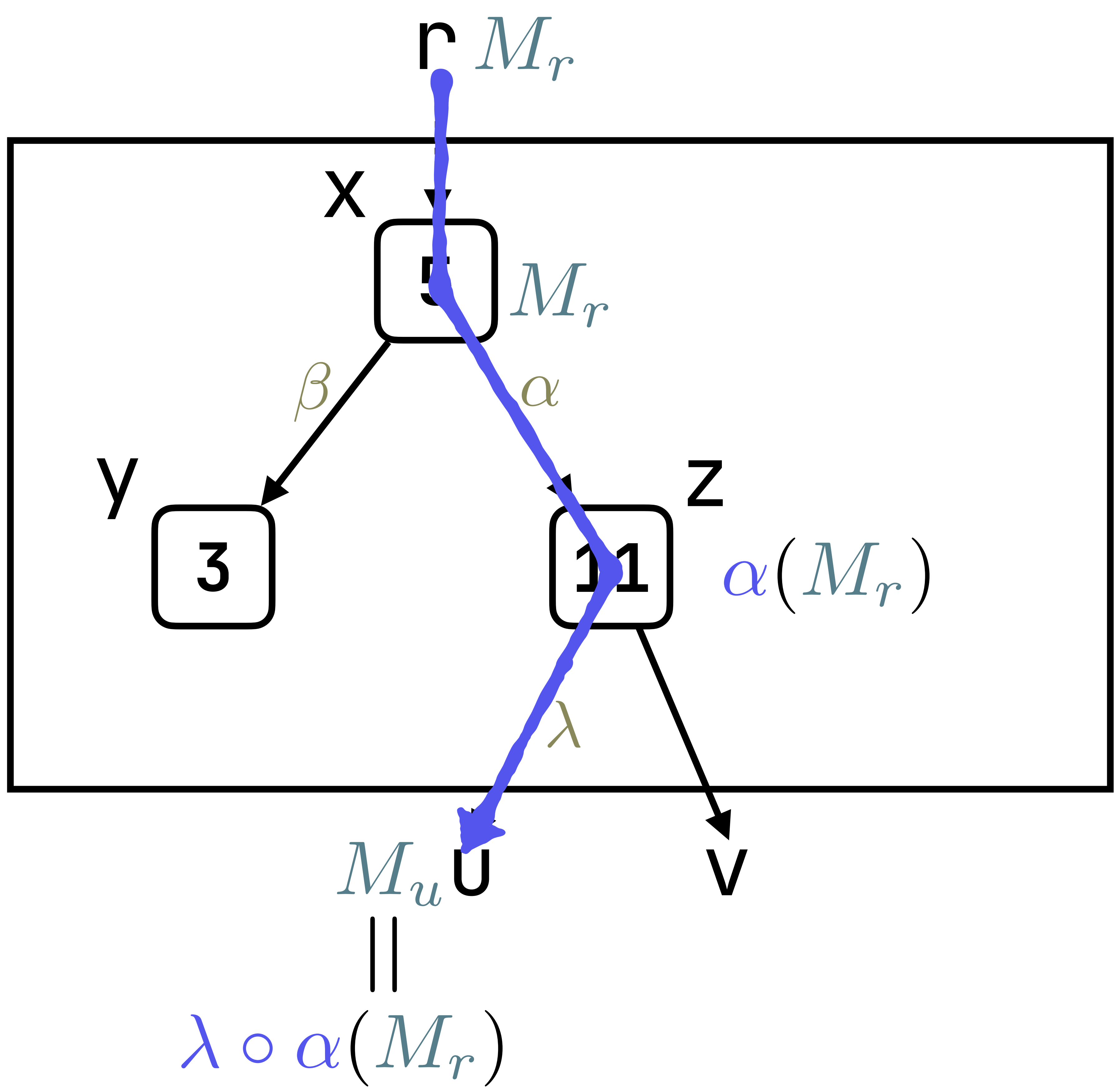
  fixed point = *concatenation of edge functions along path*



Avoiding Fixed Points

- Observation for trees:

  fixed point = concatenation of edge functions along path

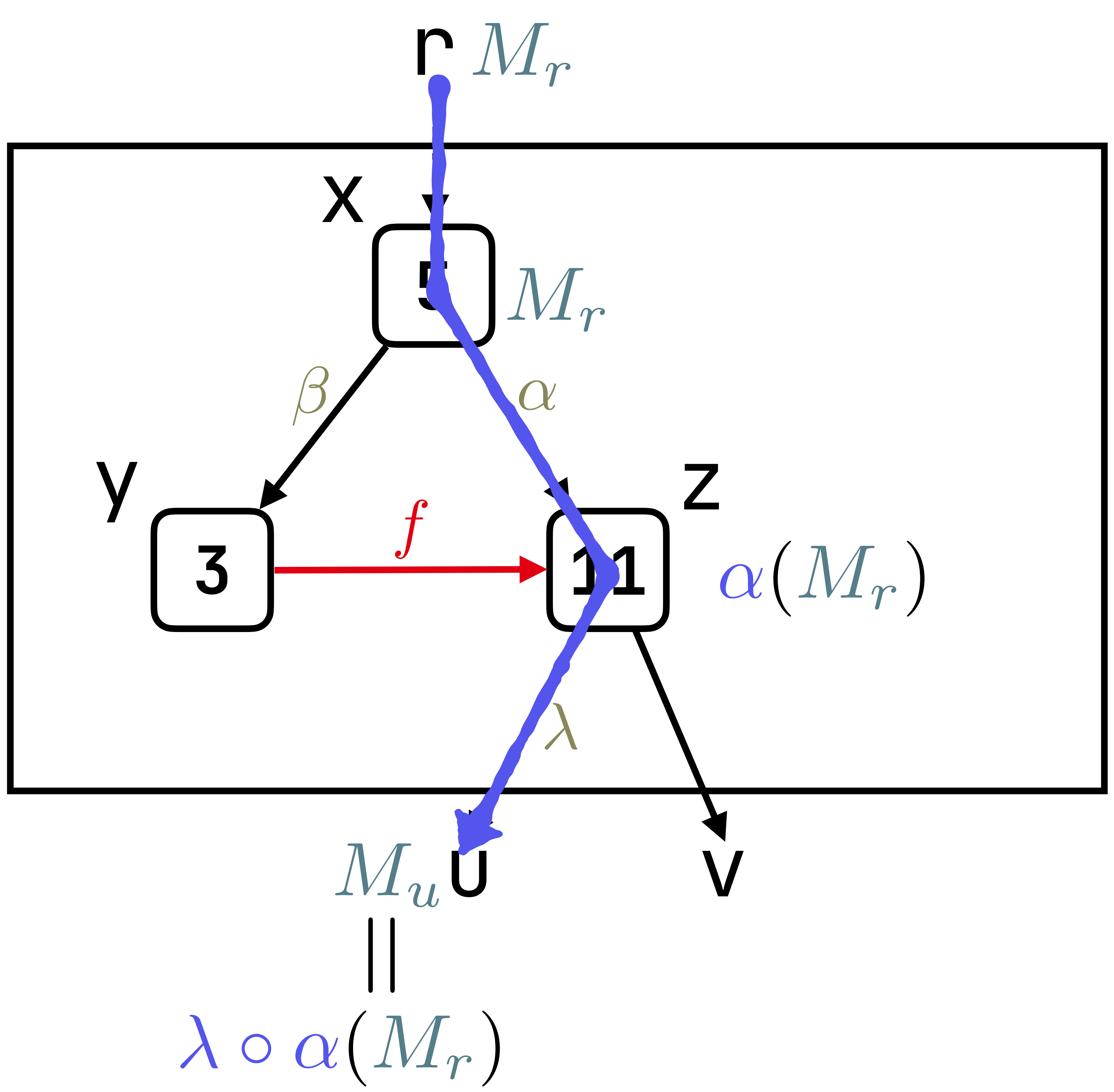


Avoiding Fixed Points

- Observation for trees:

  fixed point = concatenation of edge functions along path

  ➡ not true in general

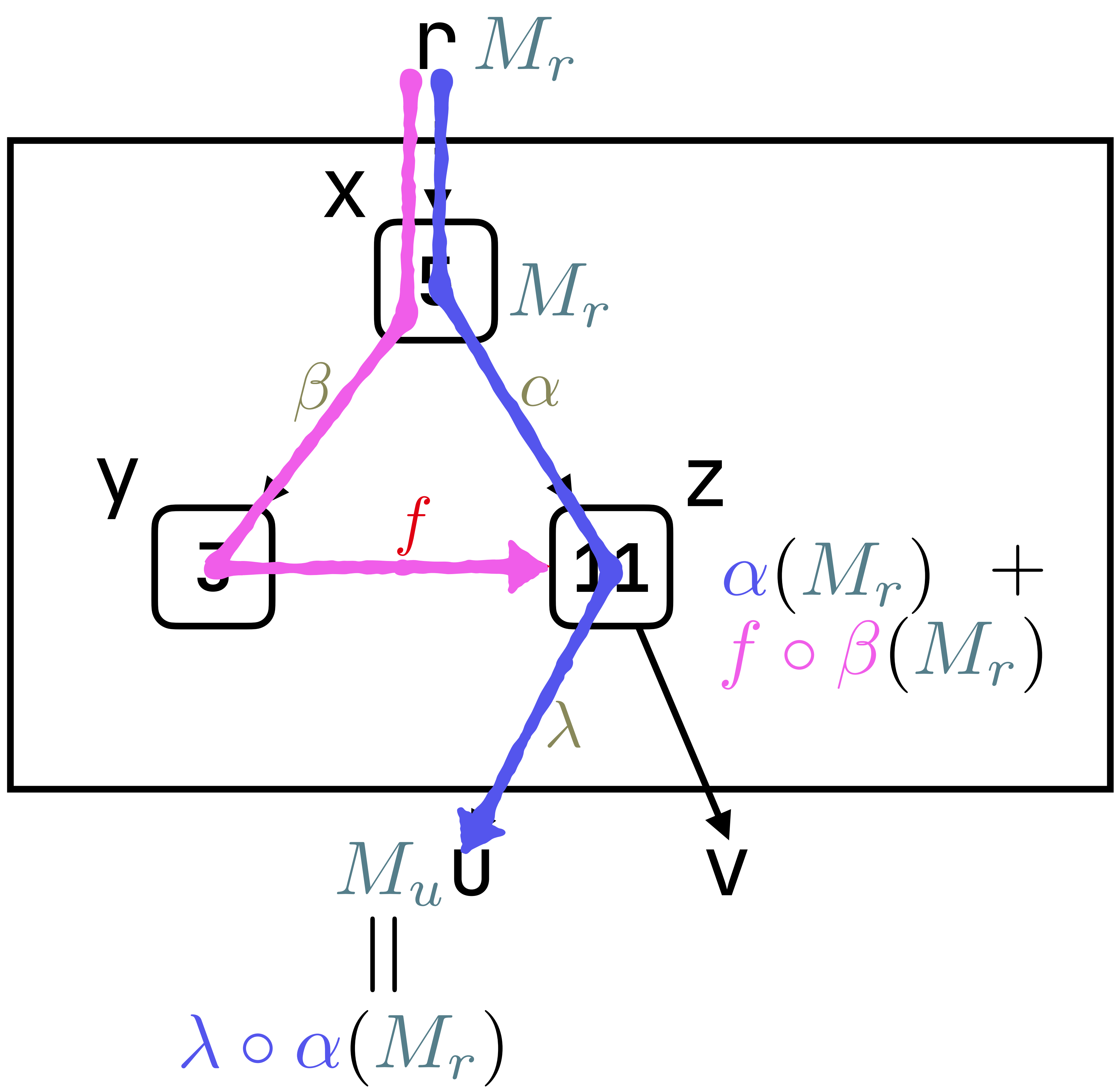


Avoiding Fixed Points

- Observation for trees:

  fixed point = *concatenation of edge functions along path*

  ➡ not true in general

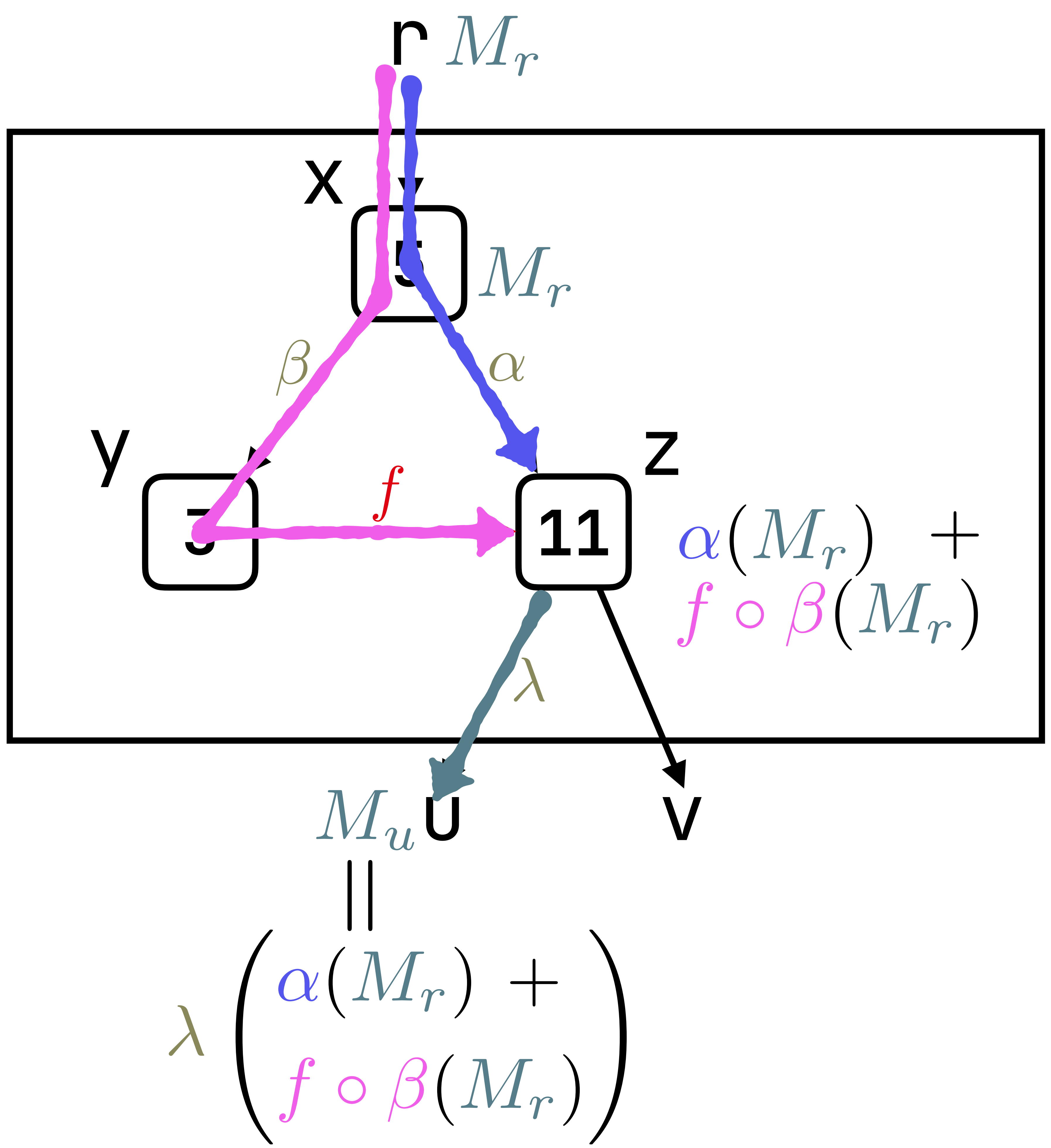


Avoiding Fixed Points

- Observation for trees:

  fixed point = concatenation of edge functions along path

  ➡ not true in general



Avoiding Fixed Points

- Observation for trees:

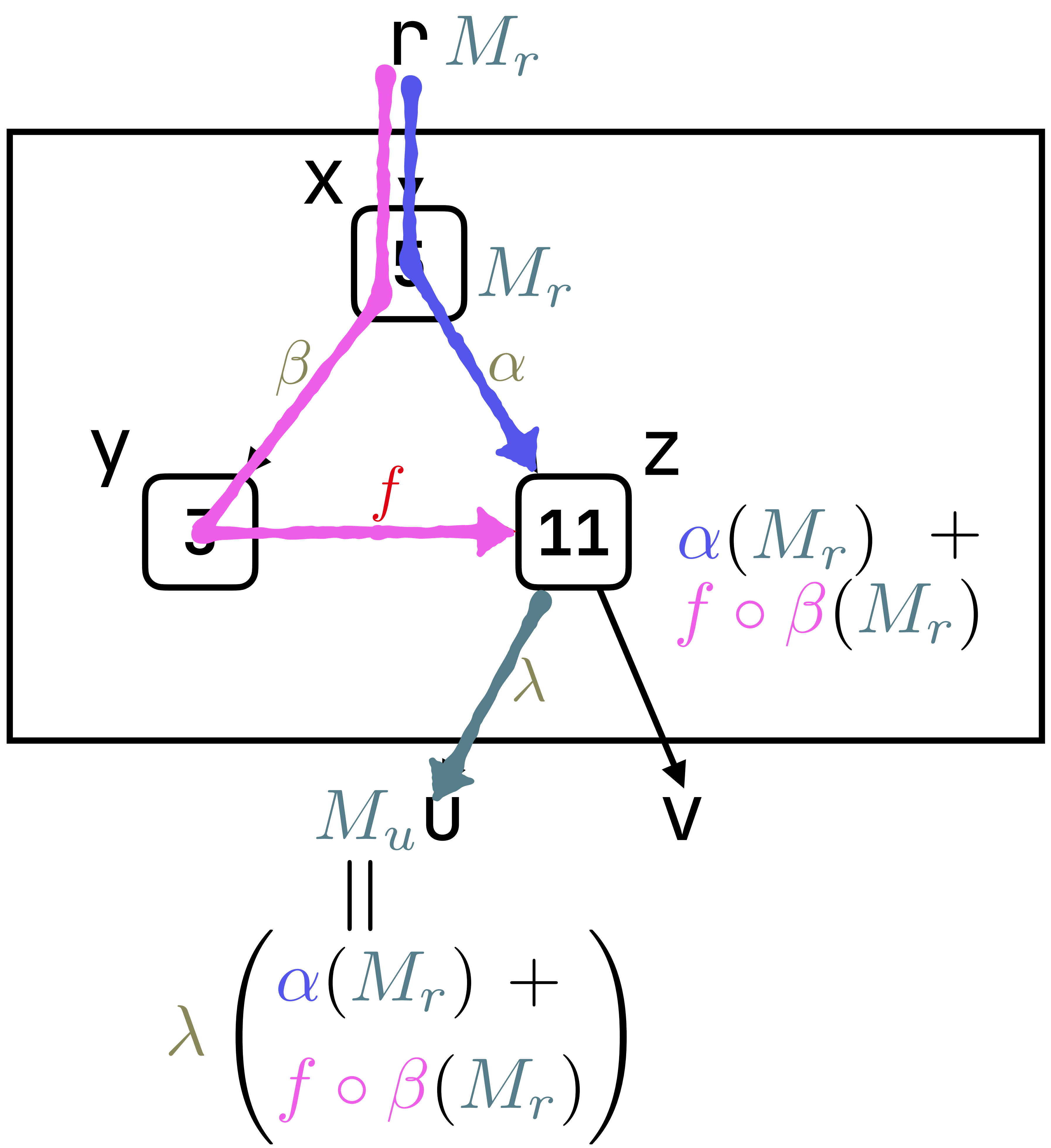
  fixed point = concatenation of edge functions along path

  ➡ not true in general

- Require **distributive** edge functions:

  $$f(m+n) = f(m) + f(n)$$

  ➡ edges do not react on "additional flow"



Avoiding Fixed Points

- Observation for trees:

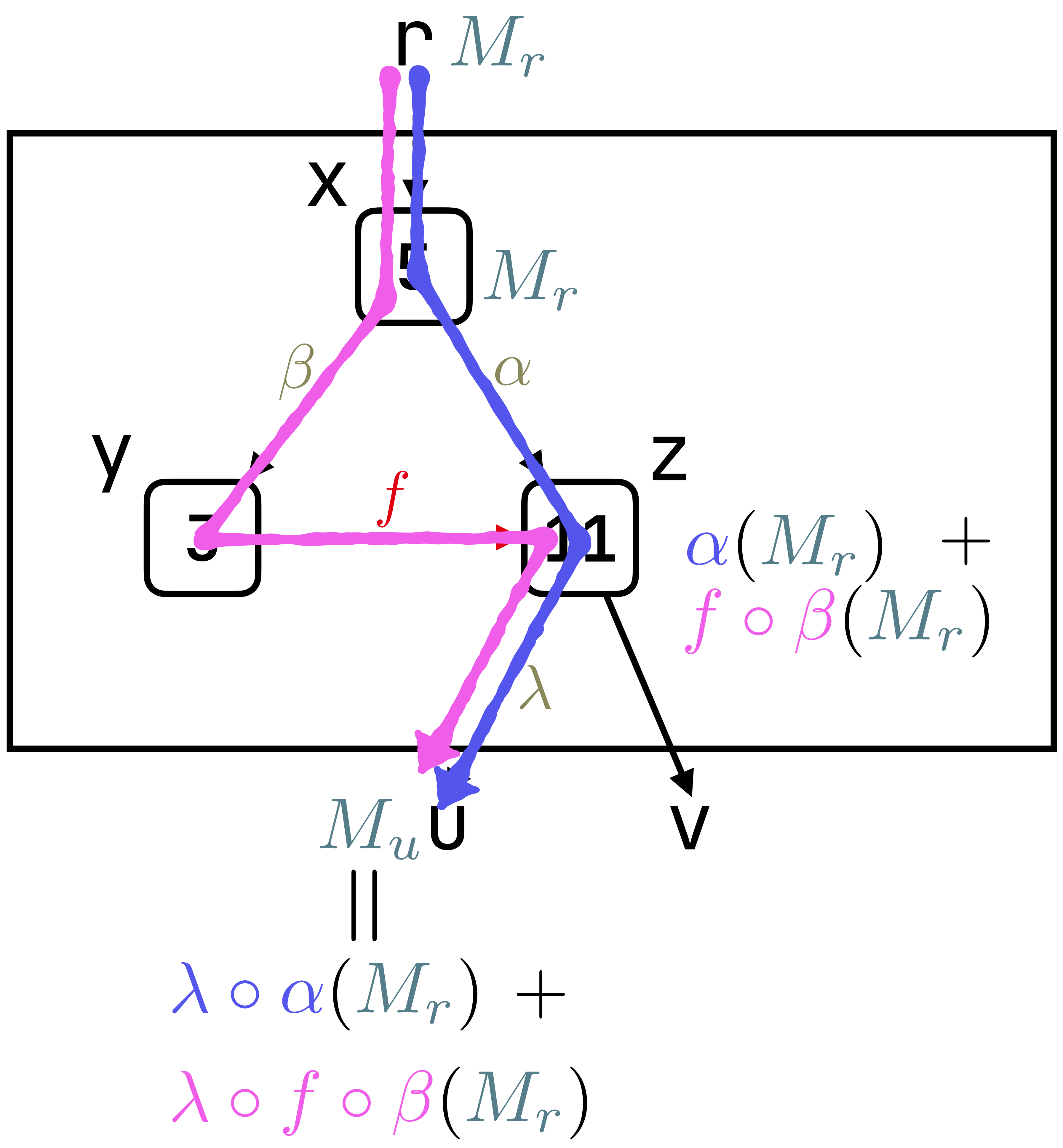
  fixed point = concatenation of edge functions along path

  ➡ not true in general

- Require **distributive** edge functions:

  $$f(m+n) = f(m) + f(n)$$

  ➡ edges do not react on "additional flow"



Avoiding Fixed Points

- Observation for trees:

  fixed point = *concatenation of edge functions along path*

  ➡ not true in general

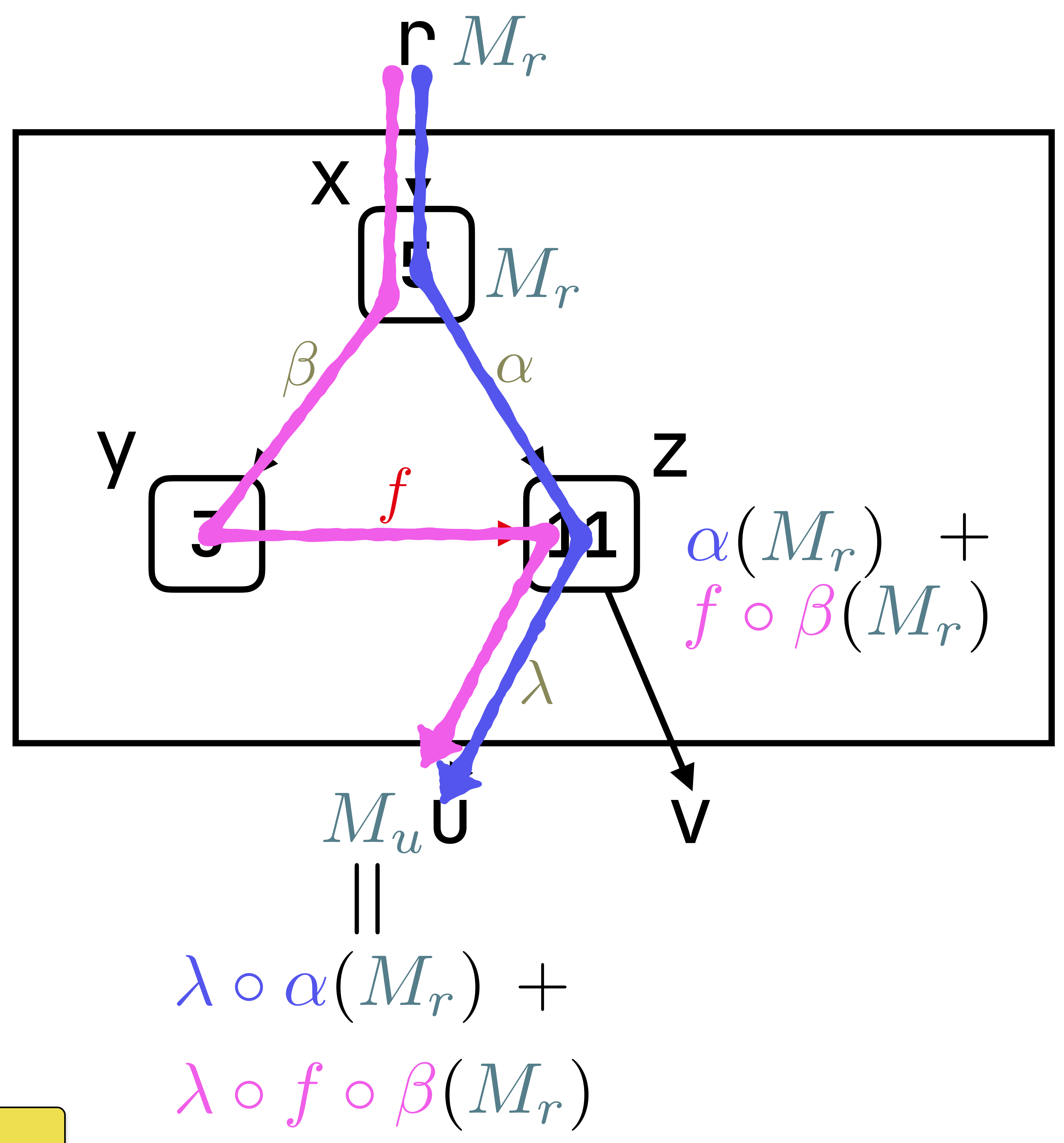
- Require **distributive** edge functions:

  $$f(m+n) = f(m) + f(n)$$

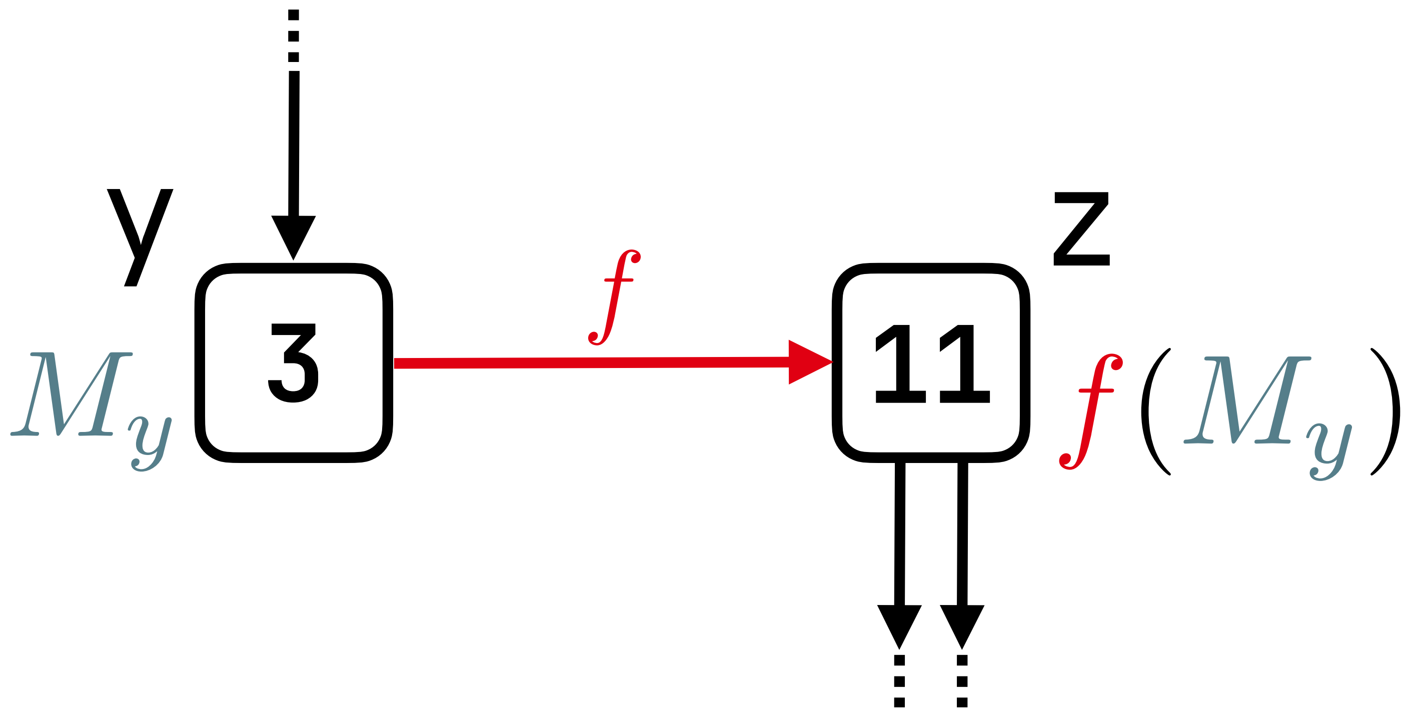
  ➡ edges do not react on "additional flow"

  ➡ fixed point = *sum over all paths*

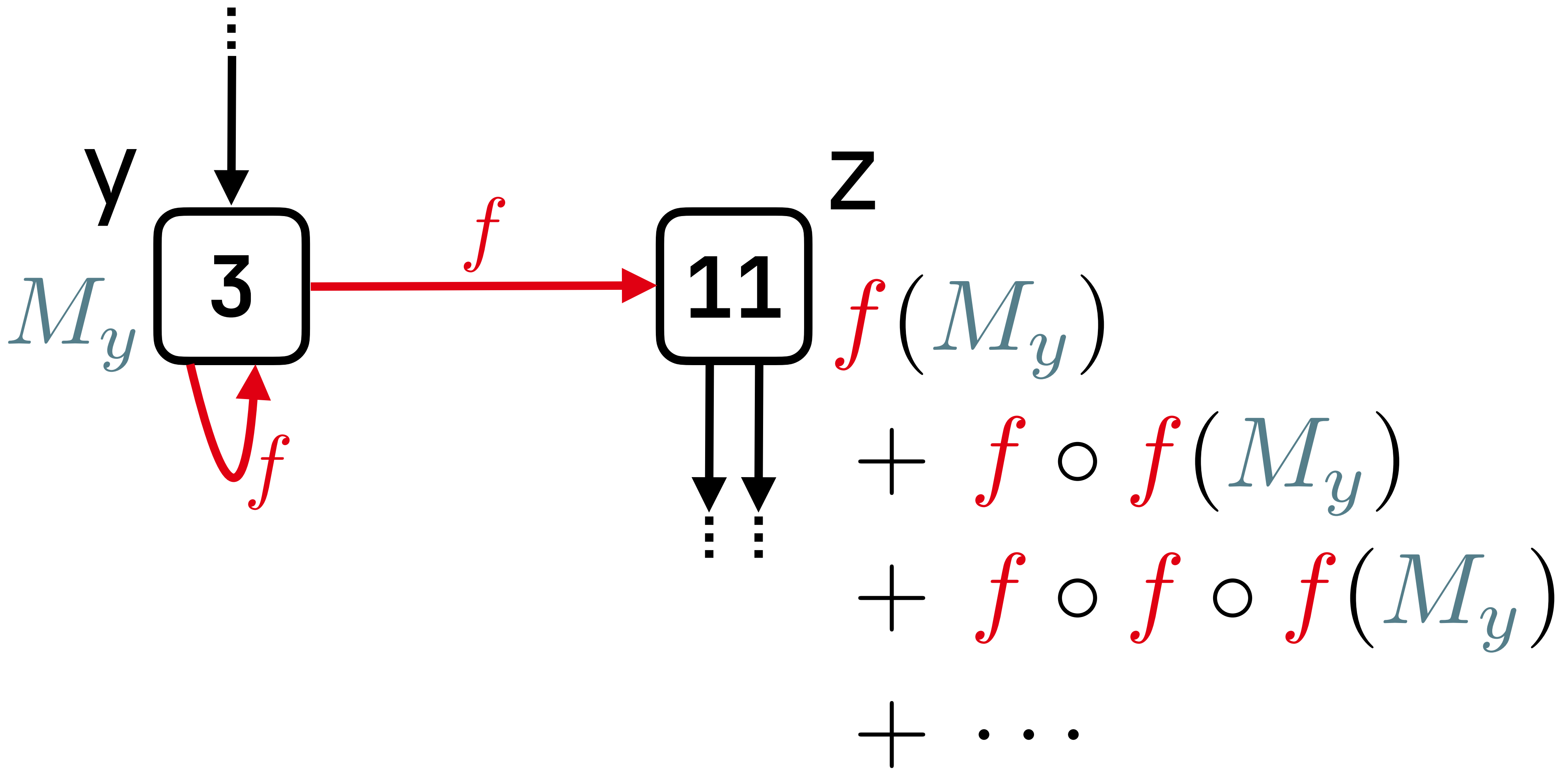
Infinite sum for cyclic graphs.



Handling Cycles



Handling Cycles

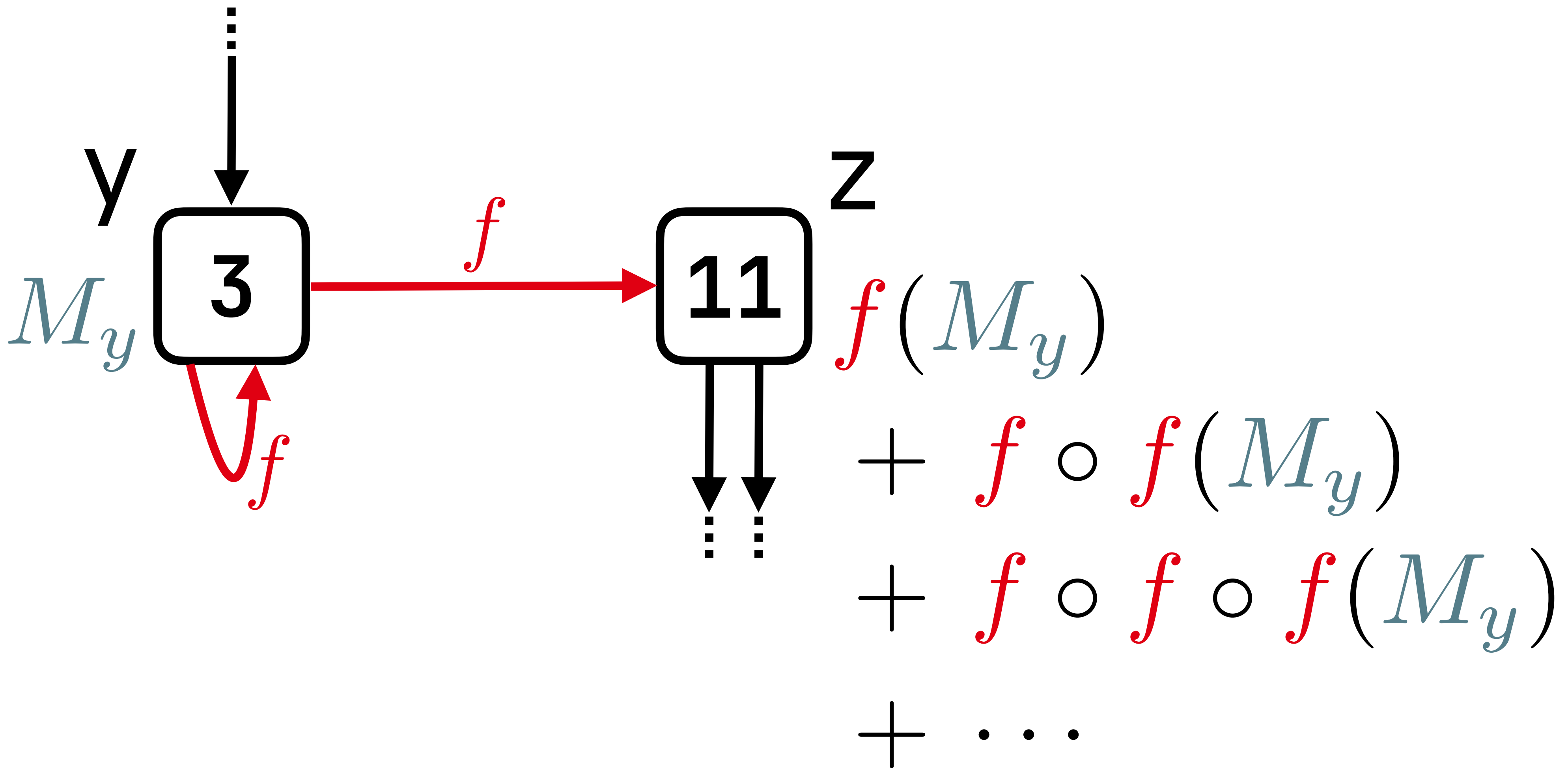


Handling Cycles

- Require decreasing edge functions

$$f(m) \leq m$$

➡ traversing "gains" information



Handling Cycles

- Require decreasing edge functions

$$f(m) \leq m$$

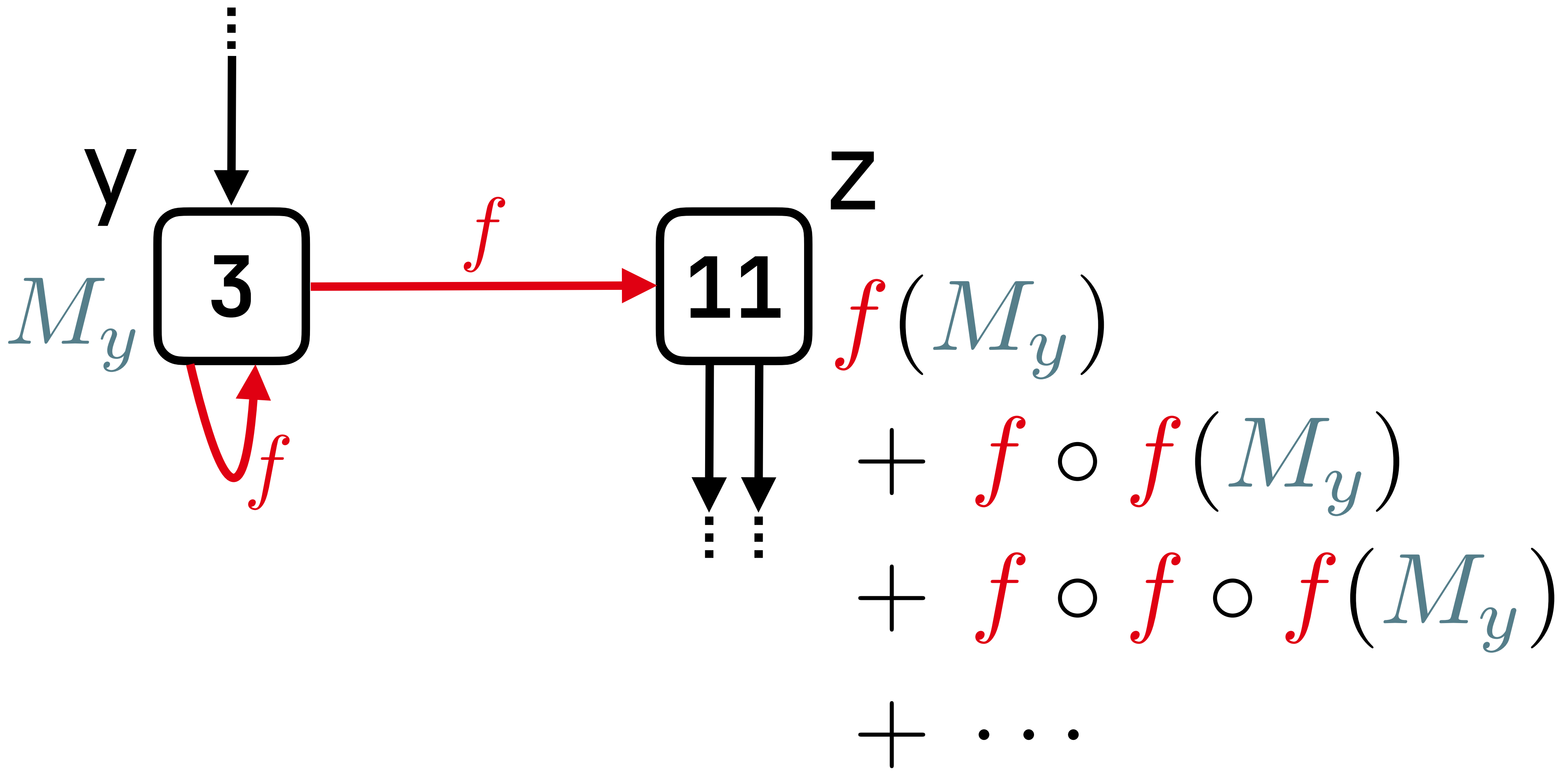
➡ traversing "gains" information

- Require idempotent addition

$$m + m = m$$

➡ $n + m = m$ if $n \leq m$

➡ flow information is "disjunctive"



Handling Cycles

- Require decreasing edge functions

$$f(m) \leq m$$

➡ traversing "gains" information

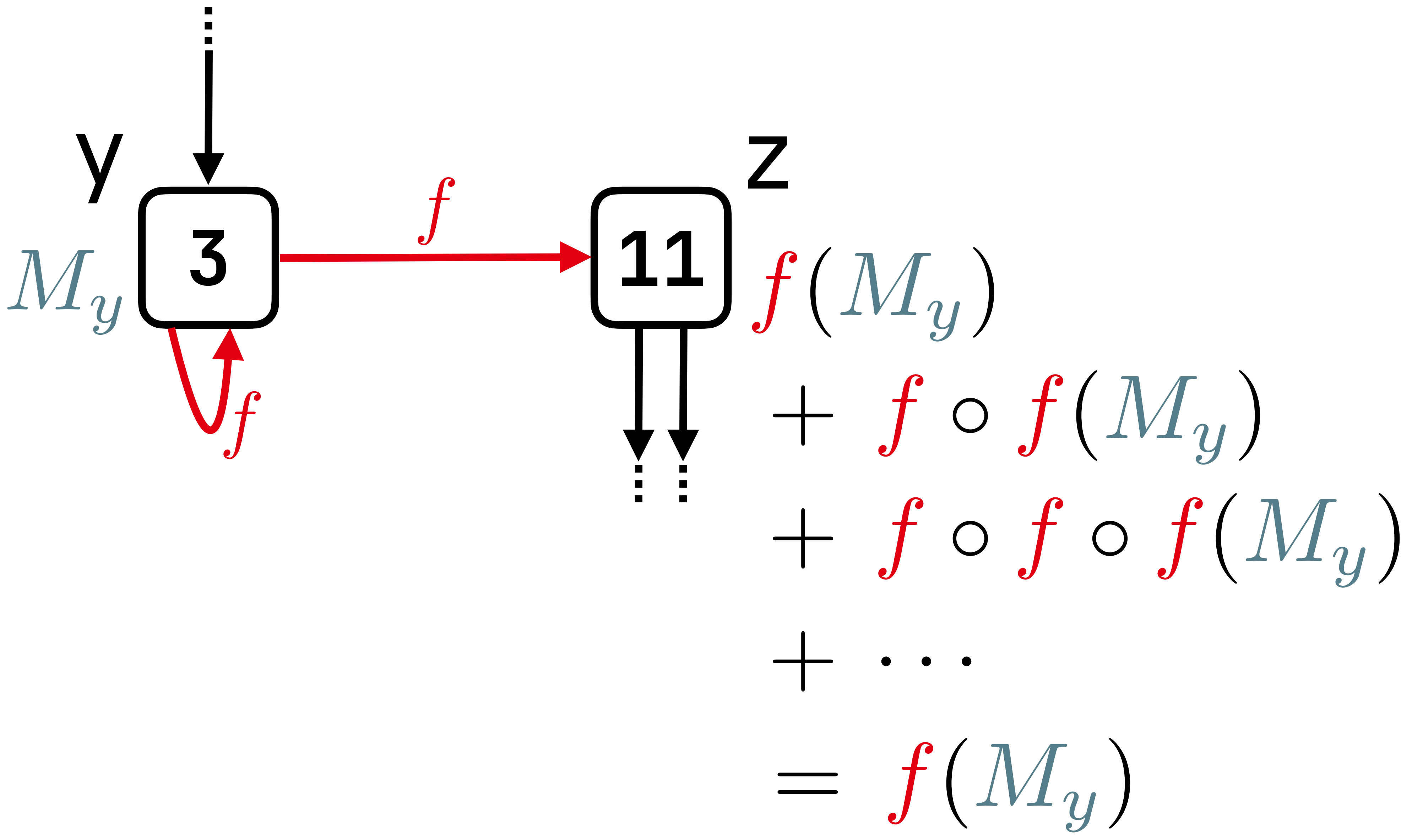
- Require idempotent addition

$$m + m = m$$

➡ $n + m = m$ if $n \leq m$

➡ flow information is "disjunctive"

➡ Combined: $\sum_{i>0} f^i(M_y) = f(M_y)$



Handling Cycles

- Require decreasing edge functions

  $$f(m) \leq m$$

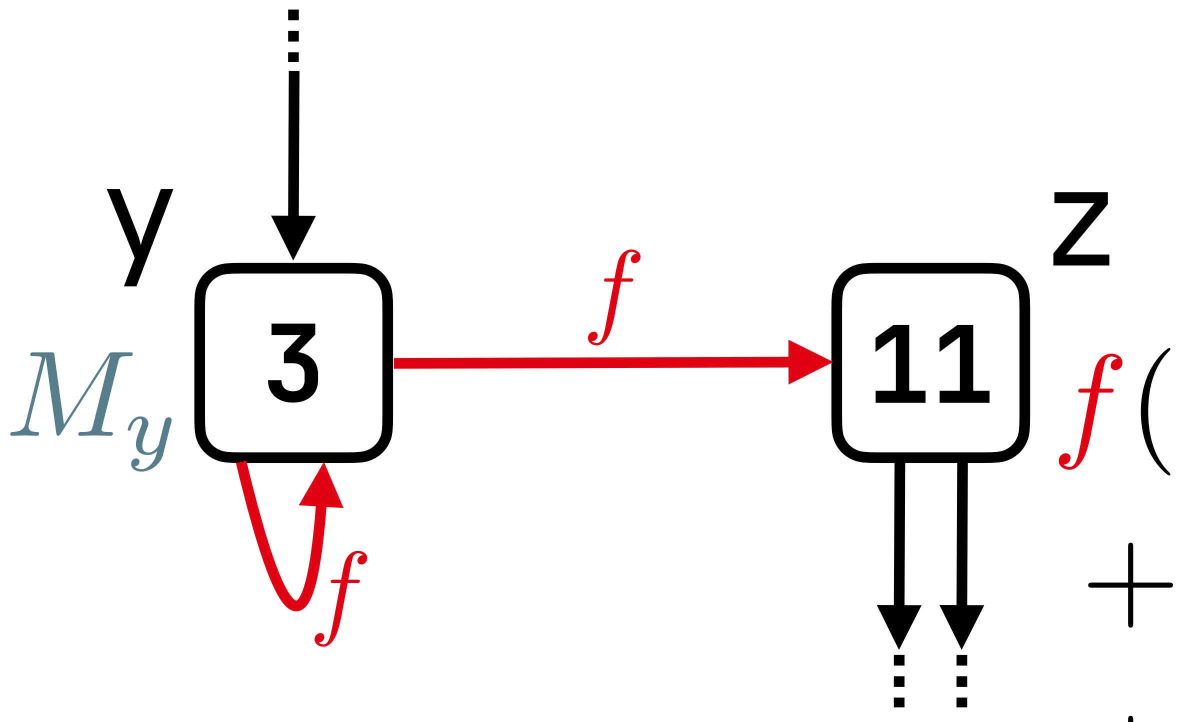
  ➡ traversing "gains" information

- Require idempotent addition

  $$m + m = m$$

  ➡ $n + m = m$ if $n \leq m$

  ➡ flow information is "disjunctive"



$$f(M_y) + f \circ f(M_y) + f \circ f \circ f(M_y) + \cdots = f(M_y)$$

➡ Combined: $\sum_{i>0} f^i(M_y) = f(M_y)$

Finite sum over all simple paths.

Review Challenges

Review Challenges

Challenge 1: fixed point might require "infinite" Kleene iteration

- require distributive & decreasing & idempotent

➡ fixed point = sum over all simple paths ✓

Review Challenges

Challenge 1: fixed point might require "infinite" Kleene iteration

- require distributive & decreasing & idempotent

➡ fixed point = sum over all simple paths ✓

Challenge 2: no restriction on graph structure

- our requirements target monoid and edge functions

➡ enabled by choice of flow ✓

Review Challenges

Challenge 1: fixed point might require "infinite" Kleene iteration

- require distributive & decreasing & idempotent

➡ fixed point = sum over all simple paths ✓

Challenge 2: no restriction on graph structure

- our requirements target monoid and edge functions

➡ enabled by choice of flow ✓

Challenge 3: assertions denote (infinitely) many heap graphs

➡ our requirements can be checked once upfront ✓

Review Challenges

Challenge 1: fixed point might require "infinite" Kleene iteration

- require distributive & decreasing & idempotent

➡ fixed point = sum over all simple paths ✓

Challenge 2: no restriction on graph structure

- our requirements target monoid and edge functions

➡ enabled by choice of flow ✓

Challenge 3: assertions denote (infinitely) many heap graphs

➡ our requirements can be checked once upfront ✓

efficient algorithm for computing footprints/frames

Thanks

Review Challenges

Challenge 1: fixed point might require "infinite" Kleene iteration

- require distributive & decreasing & idempotent

➡ fixed point = sum over all simple paths ✓

Challenge 2: no restriction on graph structure

- our requirements target monoid and edge functions

➡ enabled by choice of flow ✓

Challenge 3: assertions denote (infinitely) many heap graphs

➡ our requirements can be checked once upfront ✓

efficient algorithm for computing footprints/frames