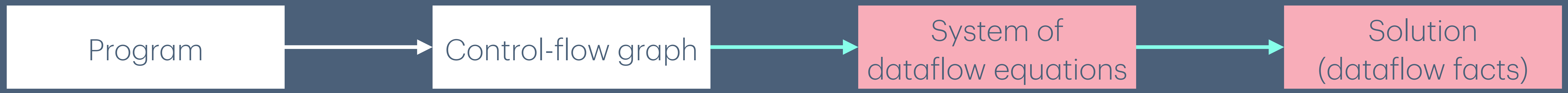
Newtonian Program Analysis of Probabilistic Programs

Di Wang and Thomas Reps

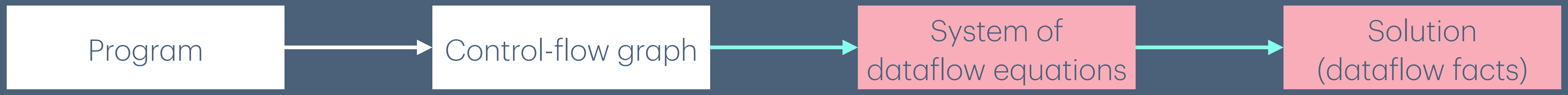
OOPSLA'24

2024.07.02

A Pipeline of Program Analysis

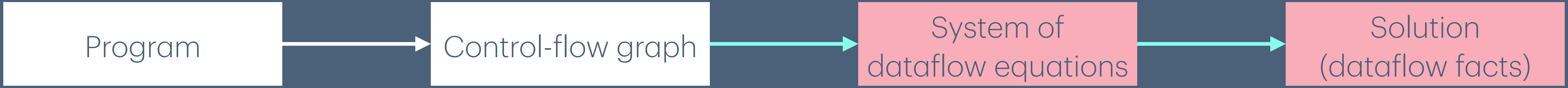


A Pipeline of Program Analysis

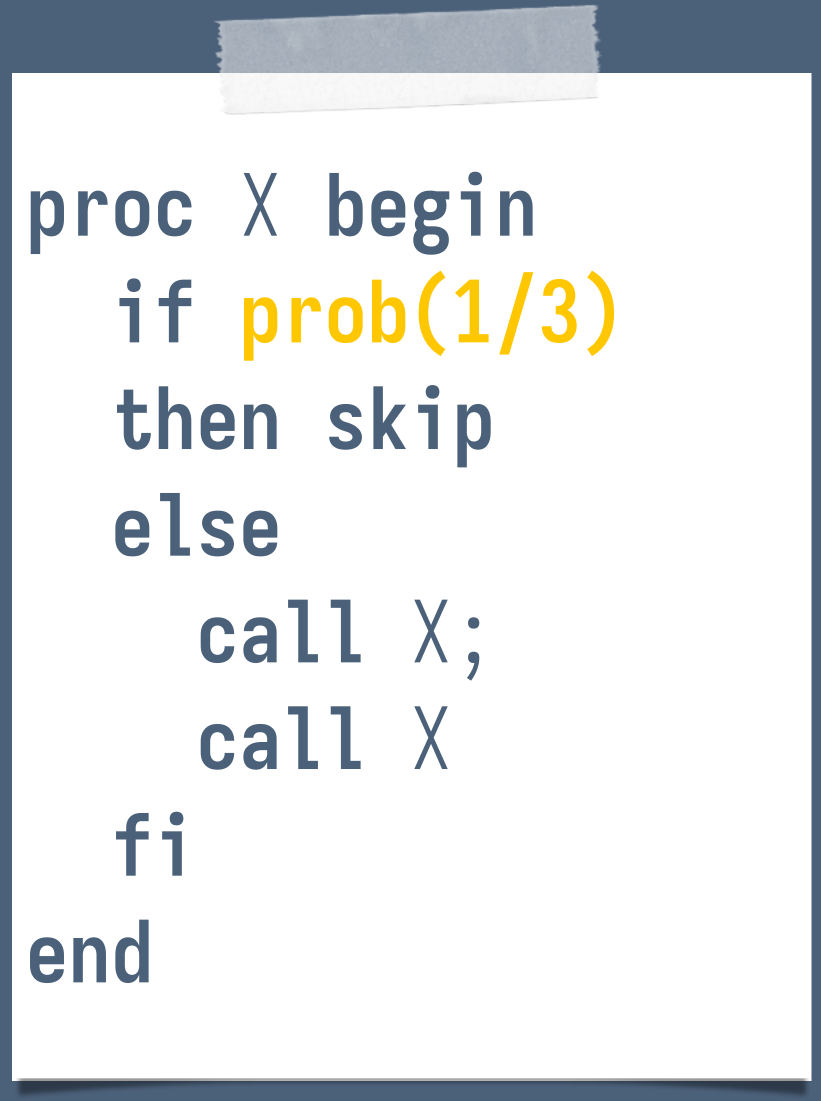


```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

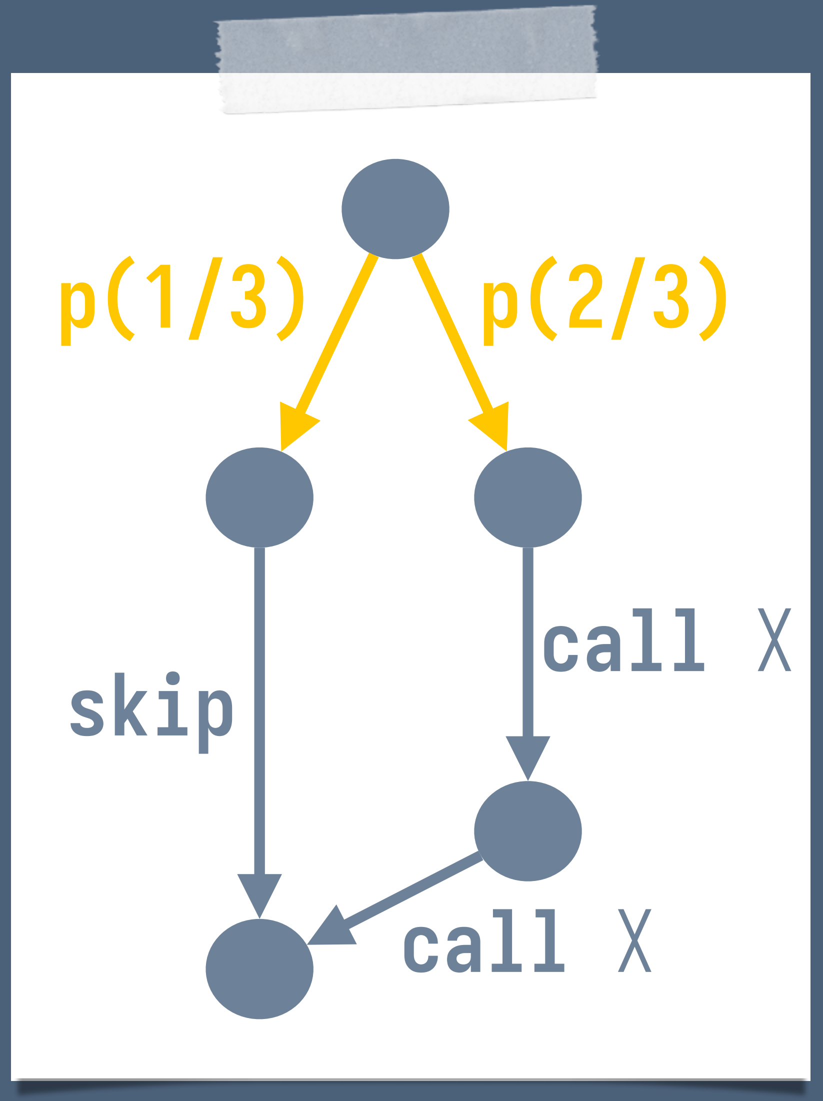
A Pipeline of Program Analysis



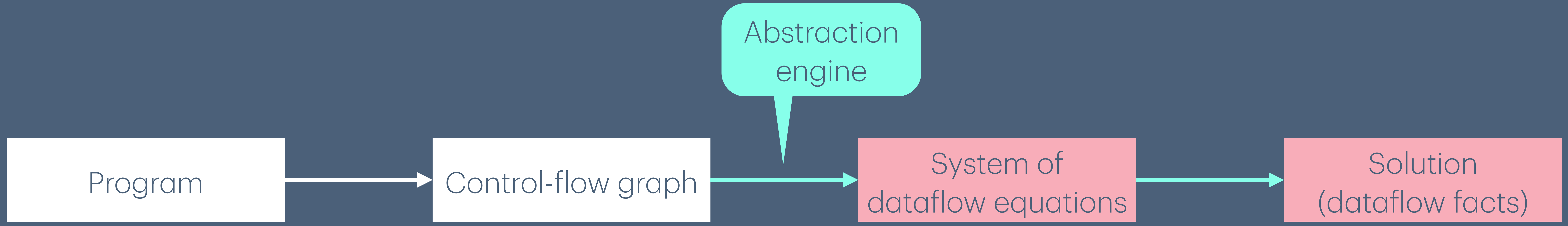
Program → Control-flow graph → System of dataflow equations → Solution (dataflow facts)



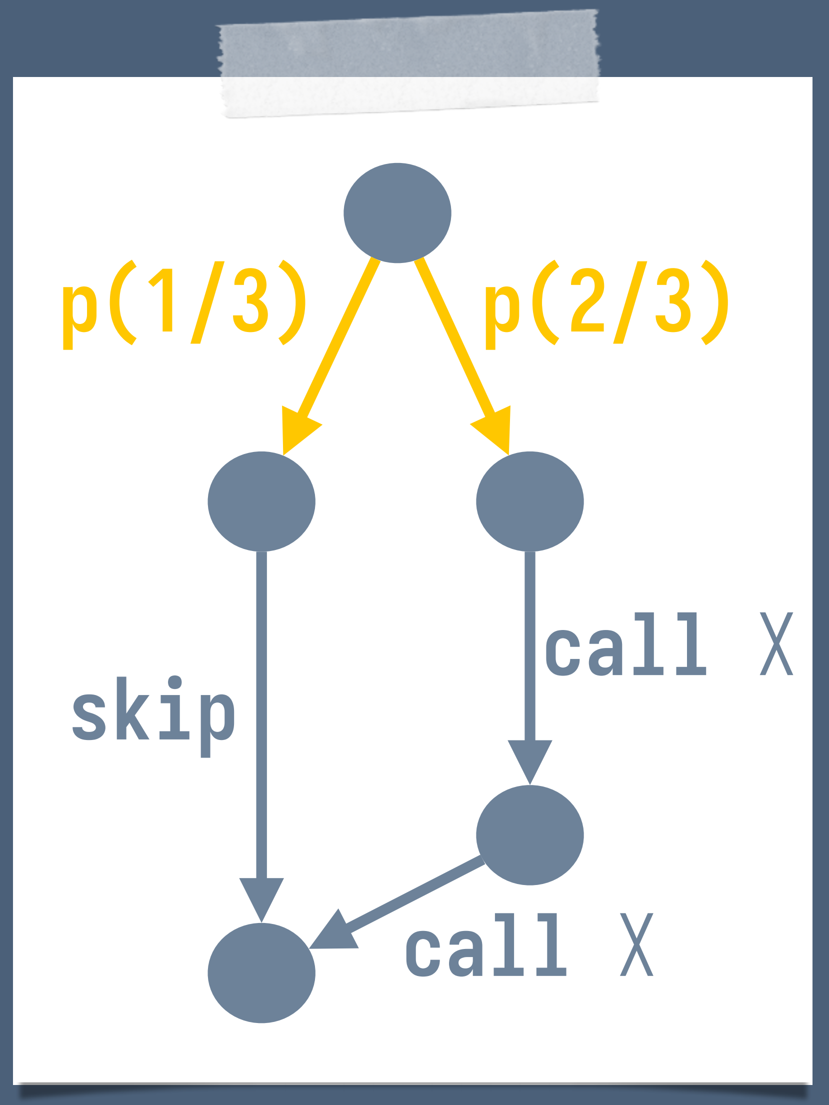
```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



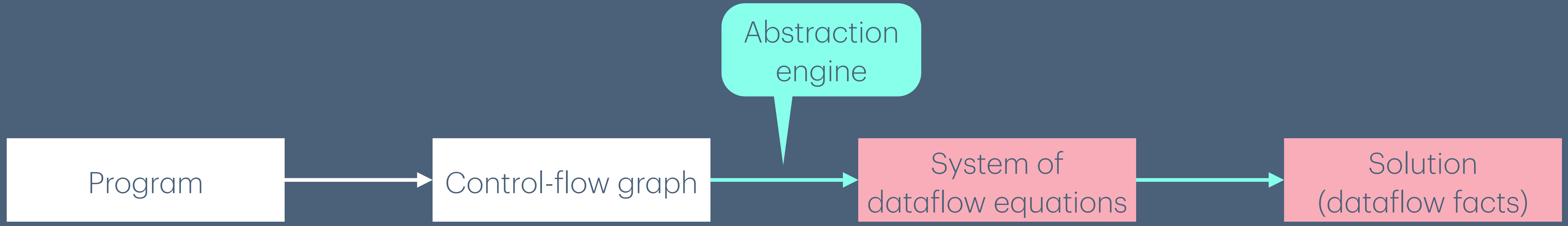
A Pipeline of Program Analysis



```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



A Pipeline of Program Analysis



Abstraction engine

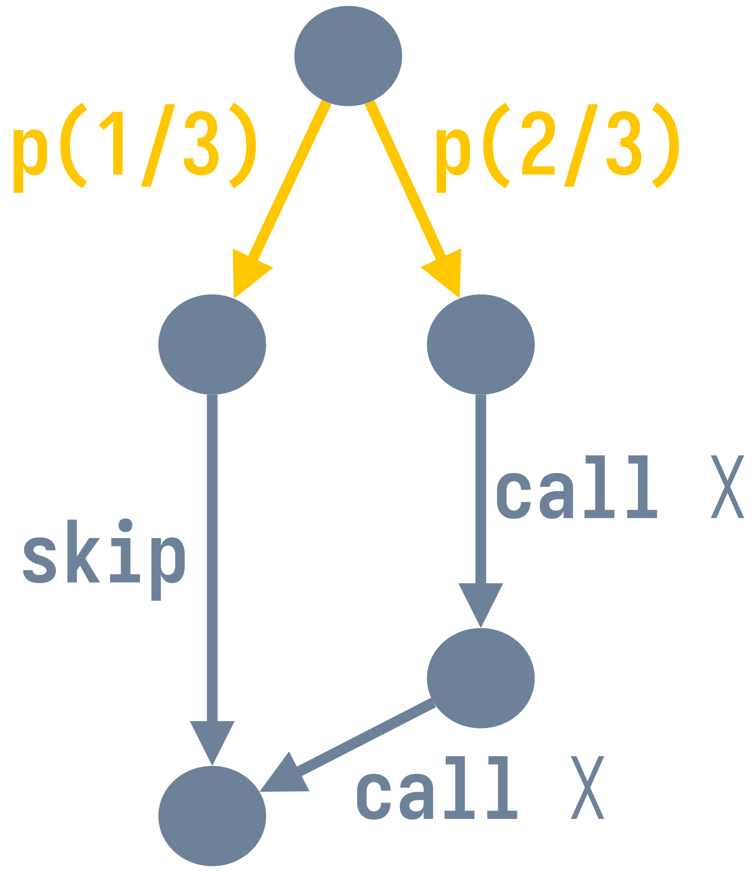
Program

Control-flow graph

System of dataflow equations

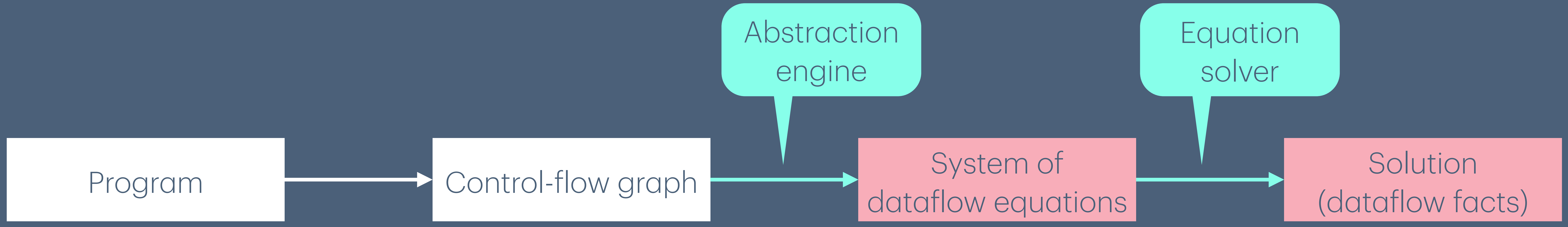
Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

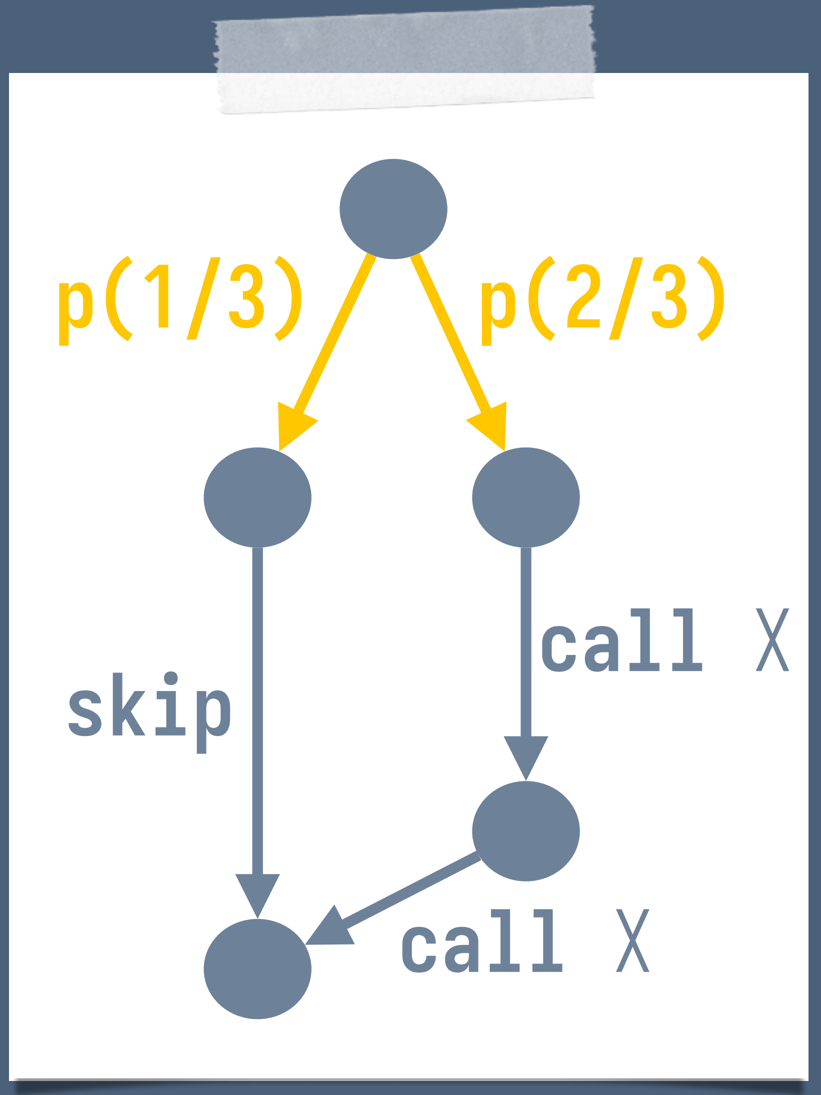


$$X = (\underline{\mathsf{p}(1/3)} \otimes \underline{\mathsf{skip}}) \oplus (\underline{\mathsf{p}(2/3)} \otimes X \otimes X)$$

A Pipeline of Program Analysis

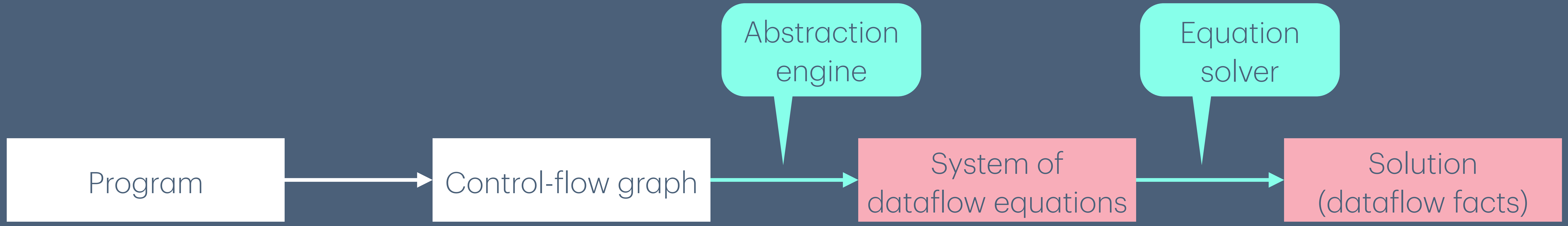


```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\underline{\mathtt{p}(1/3)} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p}(2/3)} \otimes X \otimes X)$$

A Pipeline of Program Analysis



Abstraction engine

Equation solver

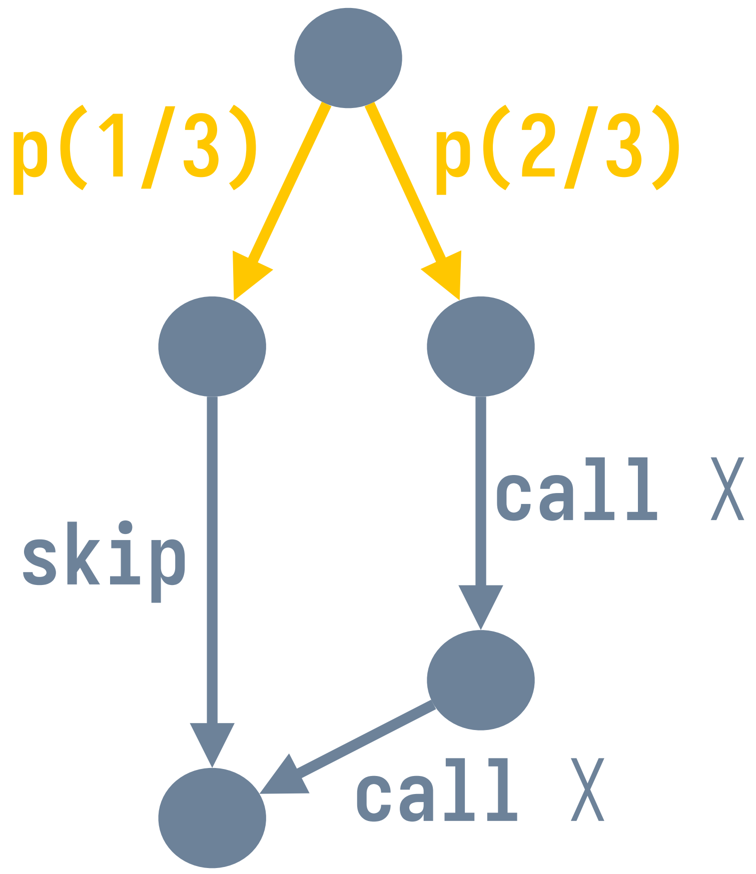
Program

Control-flow graph

System of dataflow equations

Solution (dataflow facts)

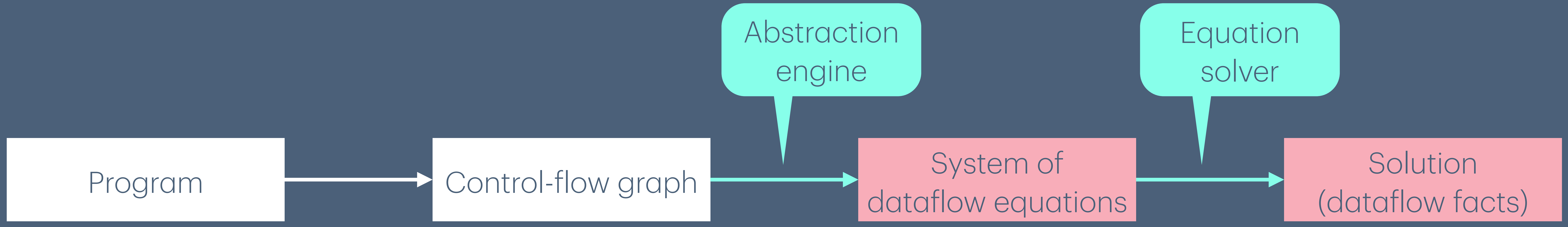
```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



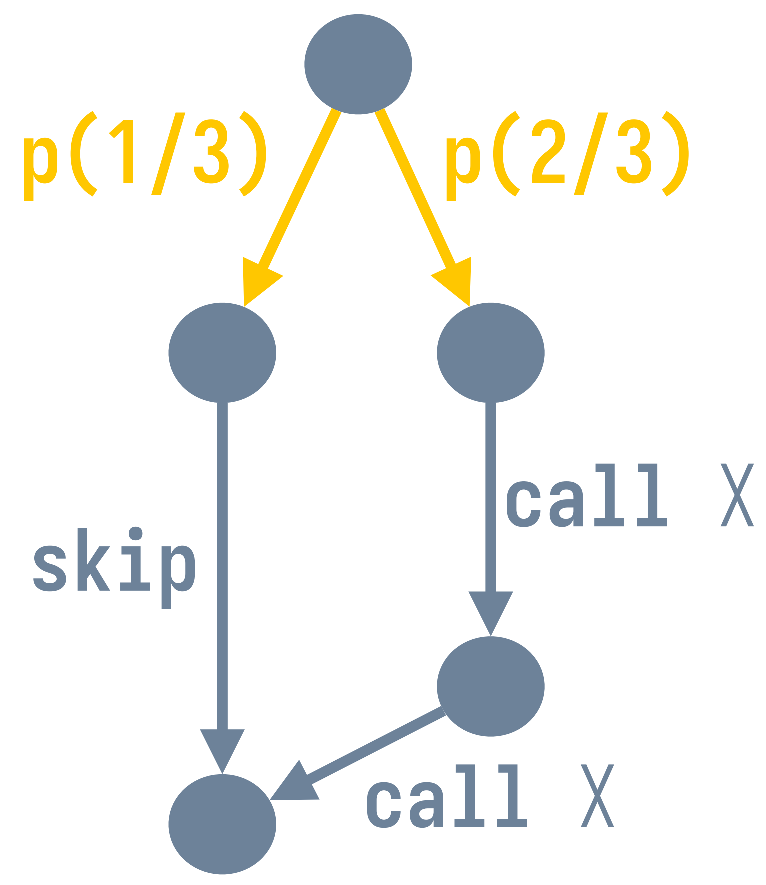
$$X = (\underline{\texttt{p}(1/3)} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p}(2/3)} \otimes X \otimes X)$$

X = "termination probability is 1/2"

A Pipeline of Program Analysis



```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

X = "termination probability is 1/2"

Also a **procedure summary** for X

The Functional Approach

[Sharir and Pnueli 1981]

The Functional Approach

[Sharir and Pnueli 1981]

- Given possibly recursive procedures $\{P_i\}$ and an abstract semantics, i.e.,

The Functional Approach

[Sharir and Pnueli 1981]

- Given possibly recursive procedures $\{P_i\}$ and an abstract semantics, i.e.,
    - a transformer on each control-flow edge, e.g., `skip`

The Functional Approach

[Sharir and Pnueli 1981]

- Given possibly recursive procedures $\{P_i\}$ and an abstract semantics, i.e.,
    - a transformer on each control-flow edge, e.g., `skip`
    - extend ($\otimes$) and combine ($\oplus$) operations

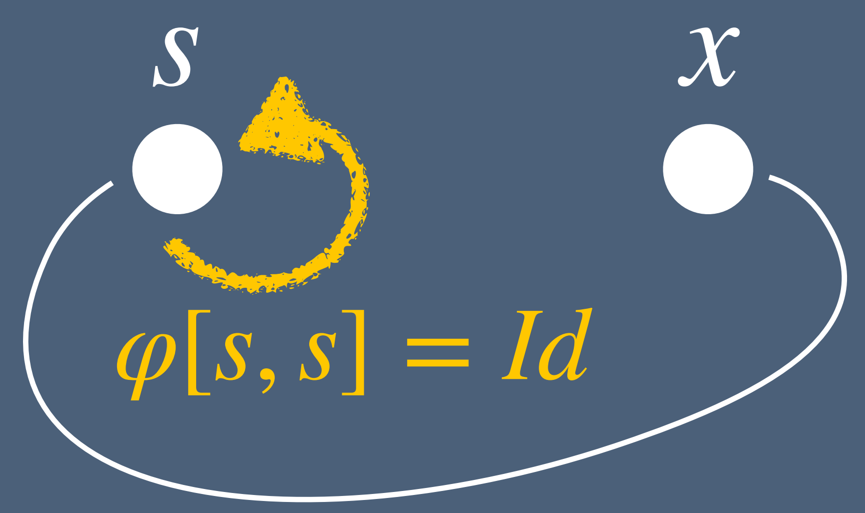
The Functional Approach

[Sharir and Pnueli 1981]

- Given possibly recursive procedures $\{P_i\}$ and an abstract semantics, i.e.,
  - a transformer on each control-flow edge, e.g., `skip`
  - extend ($\otimes$) and combine ($\oplus$) operations
- Find a procedure summary for each P_i

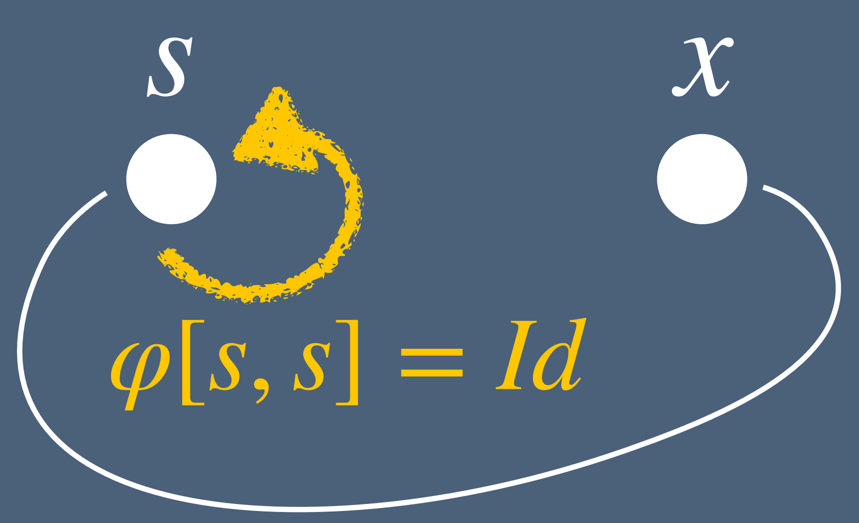
The Functional Approach

[Sharir and Pnueli 1981]



The Functional Approach

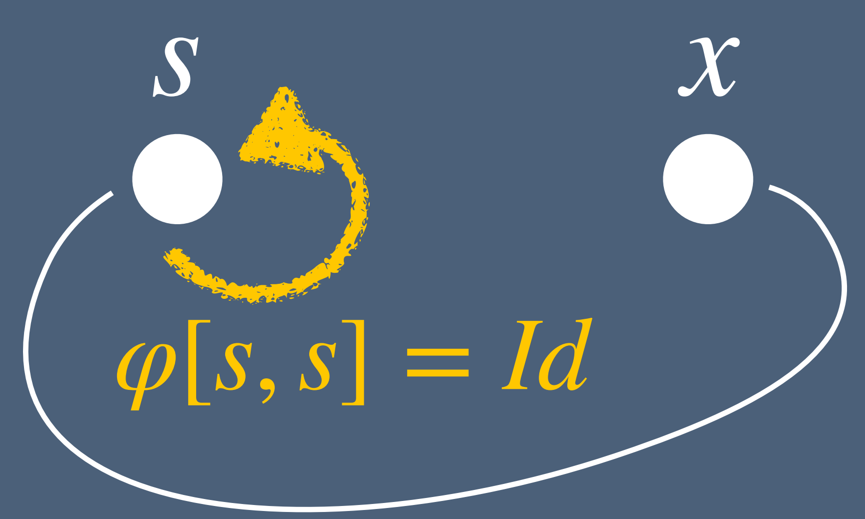
[Sharir and Pnueli 1981]

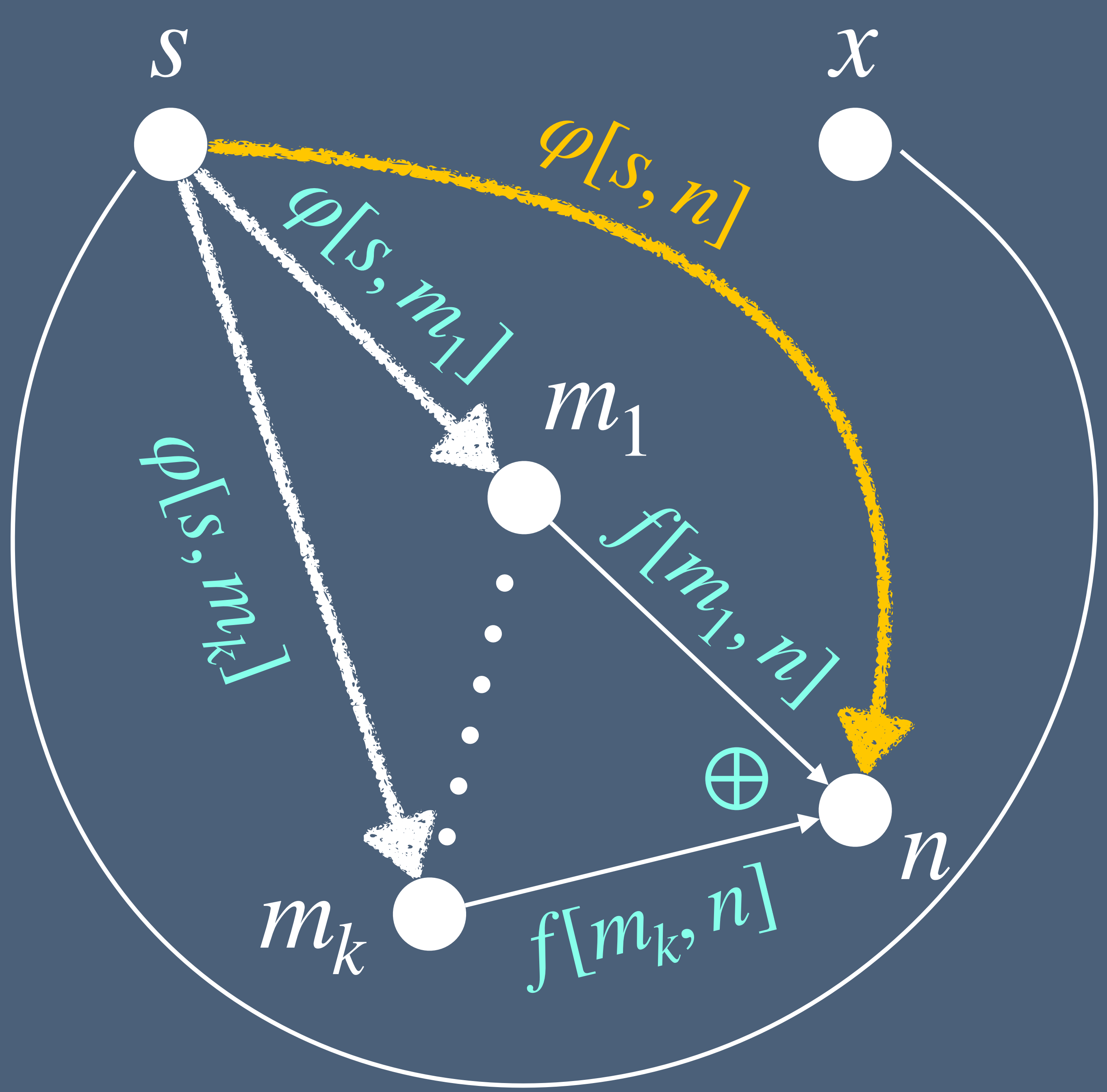


$$\varphi[s, s] = Id$$

The Functional Approach

[Sharir and Pnueli 1981]

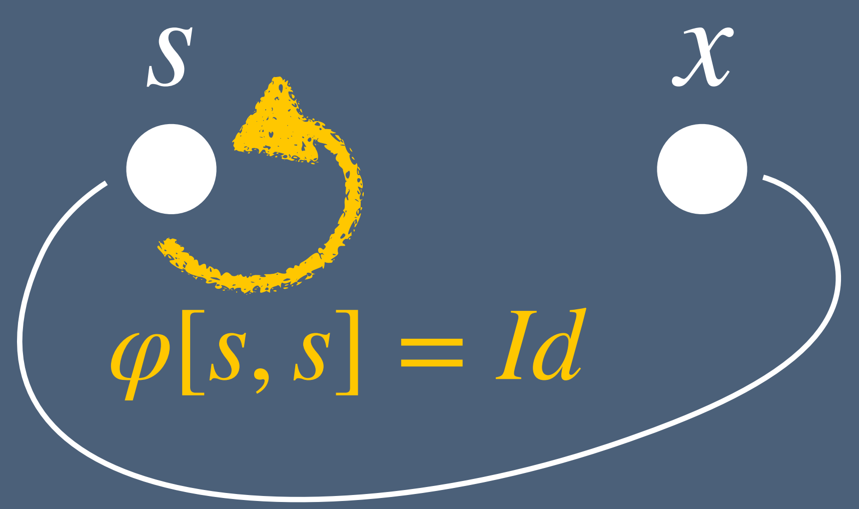


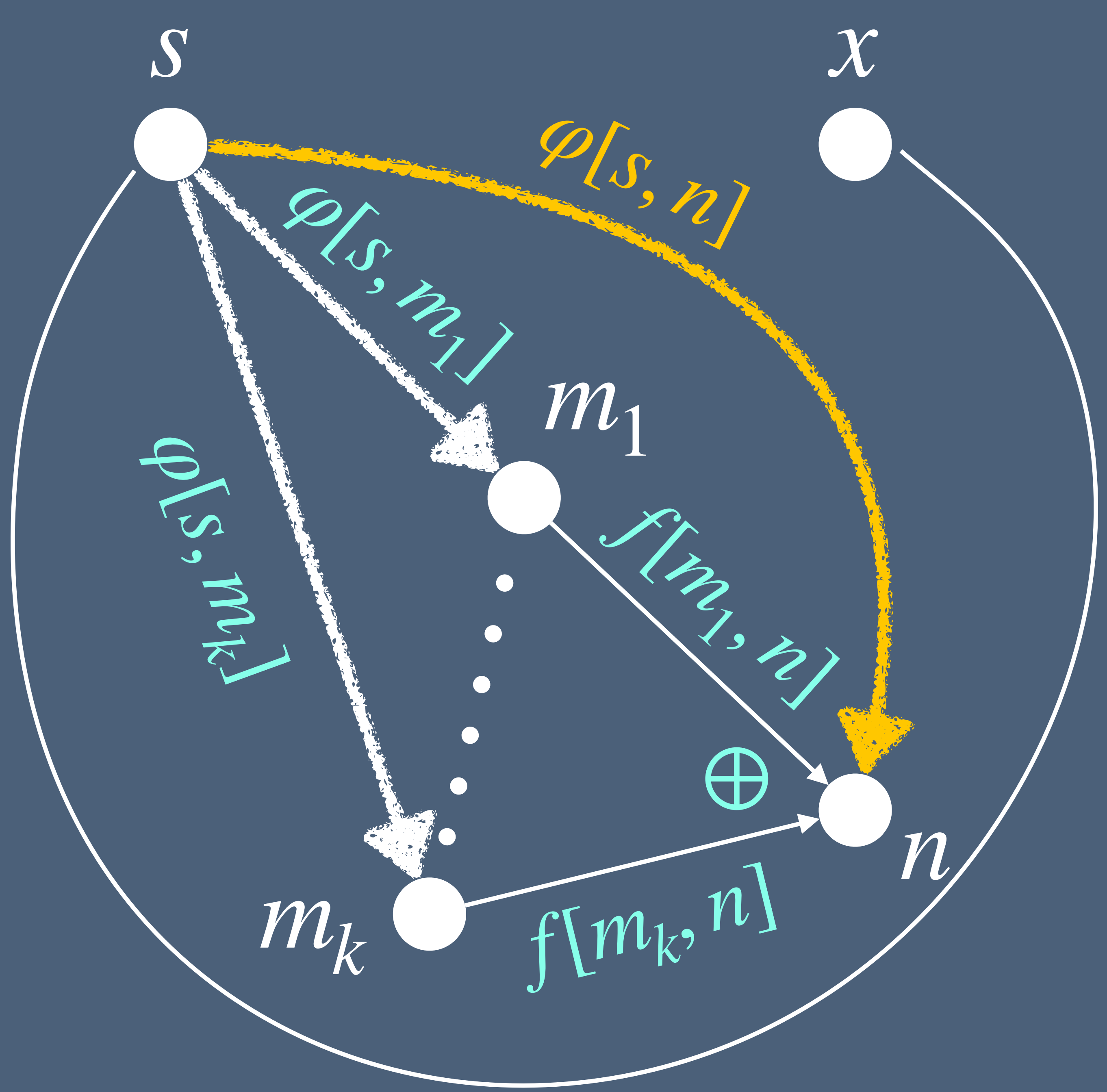


$$\varphi[s,s] = Id$$

The Functional Approach

[Sharir and Pnueli 1981]



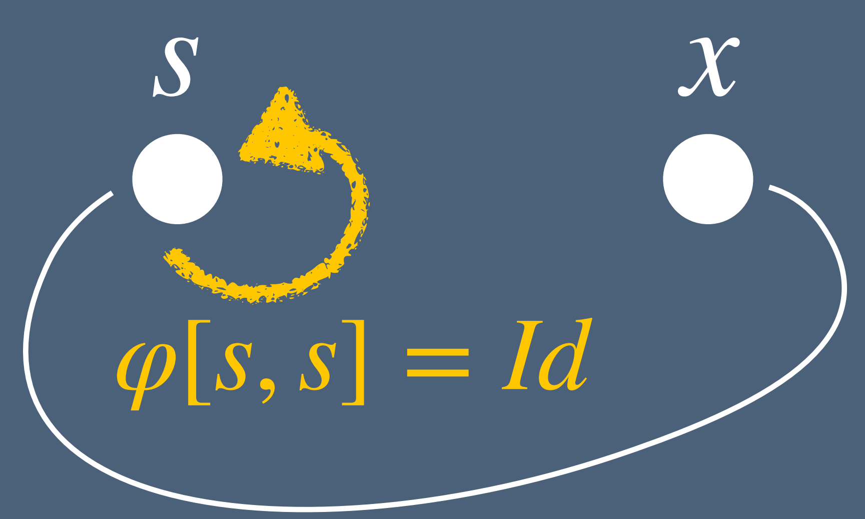


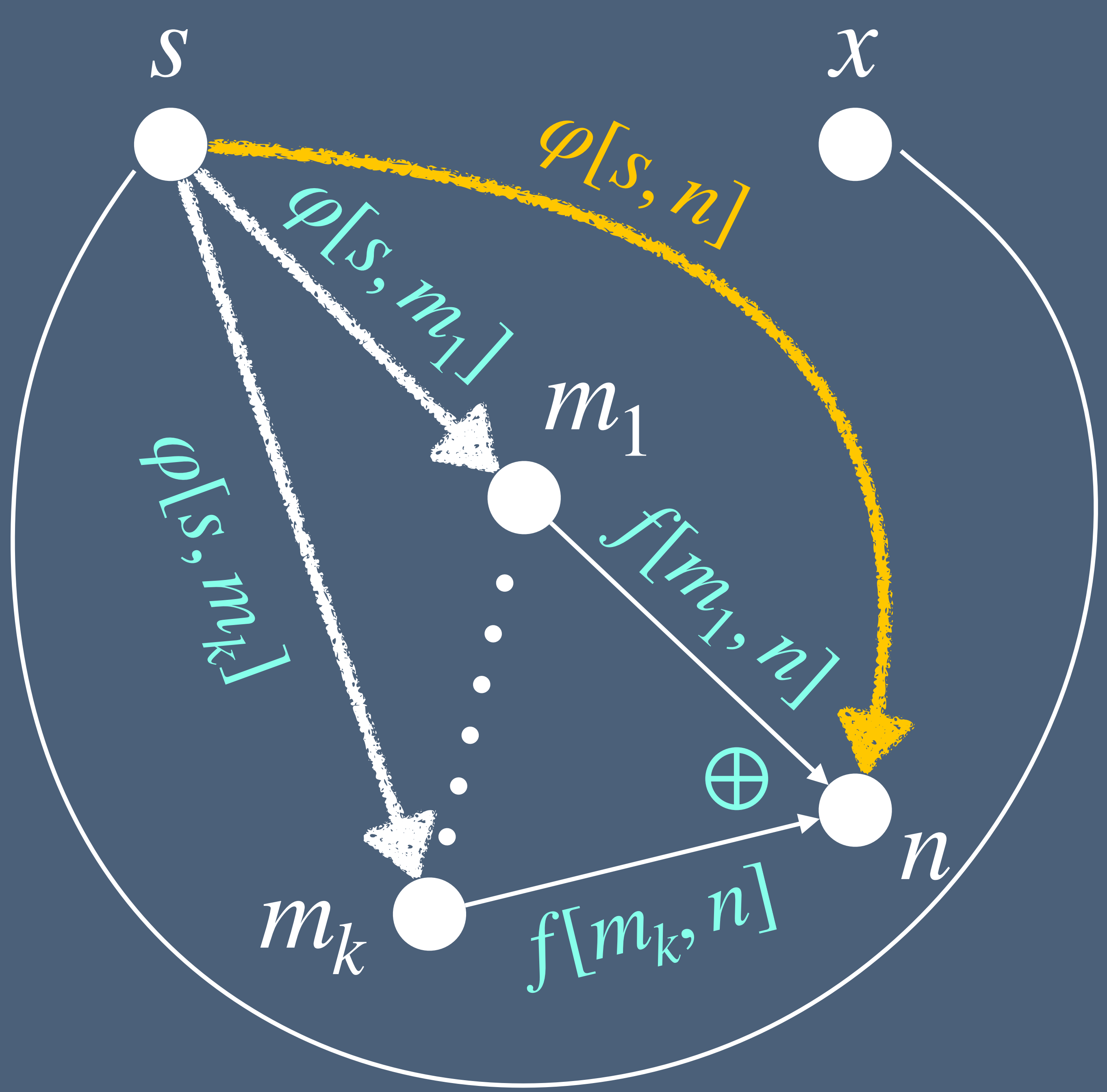
$$\varphi[s,s] = Id$$

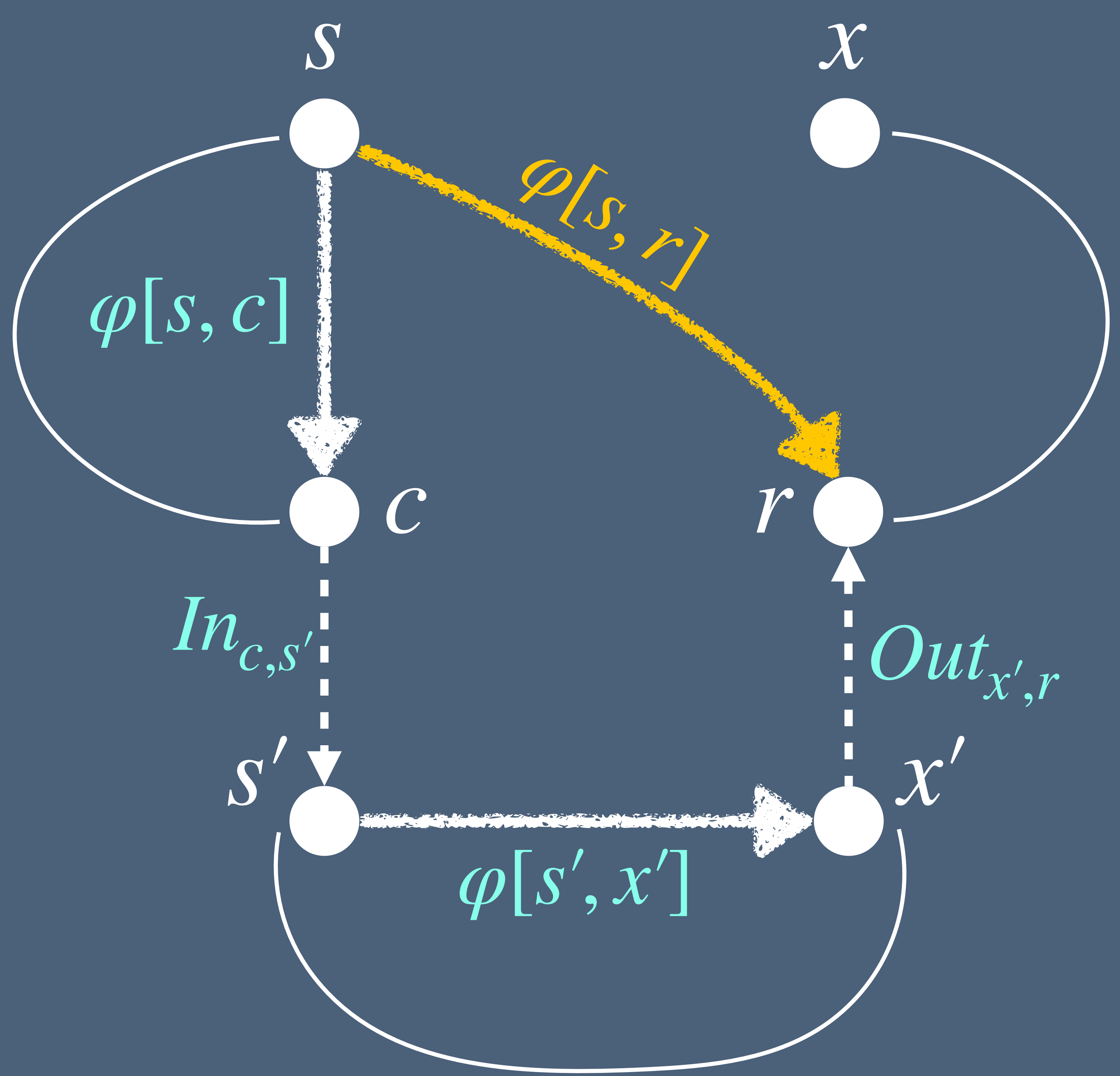
$$\begin{aligned}\varphi[s,n] = {} & (\varphi[s,m_1] \otimes f[m_1,n]) \\ & \oplus \ldots \\ & \oplus (\varphi[s,m_k] \otimes f[m_k,n])\end{aligned}$$

The Functional Approach

[Sharir and Pnueli 1981]





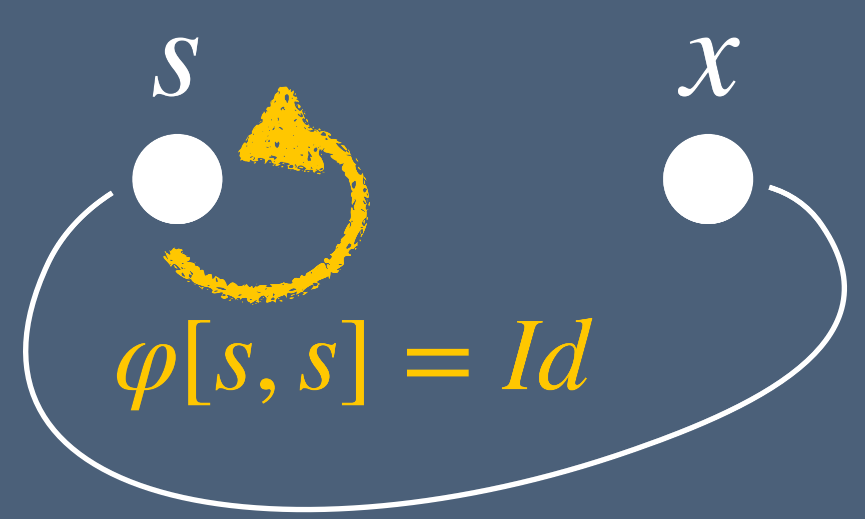


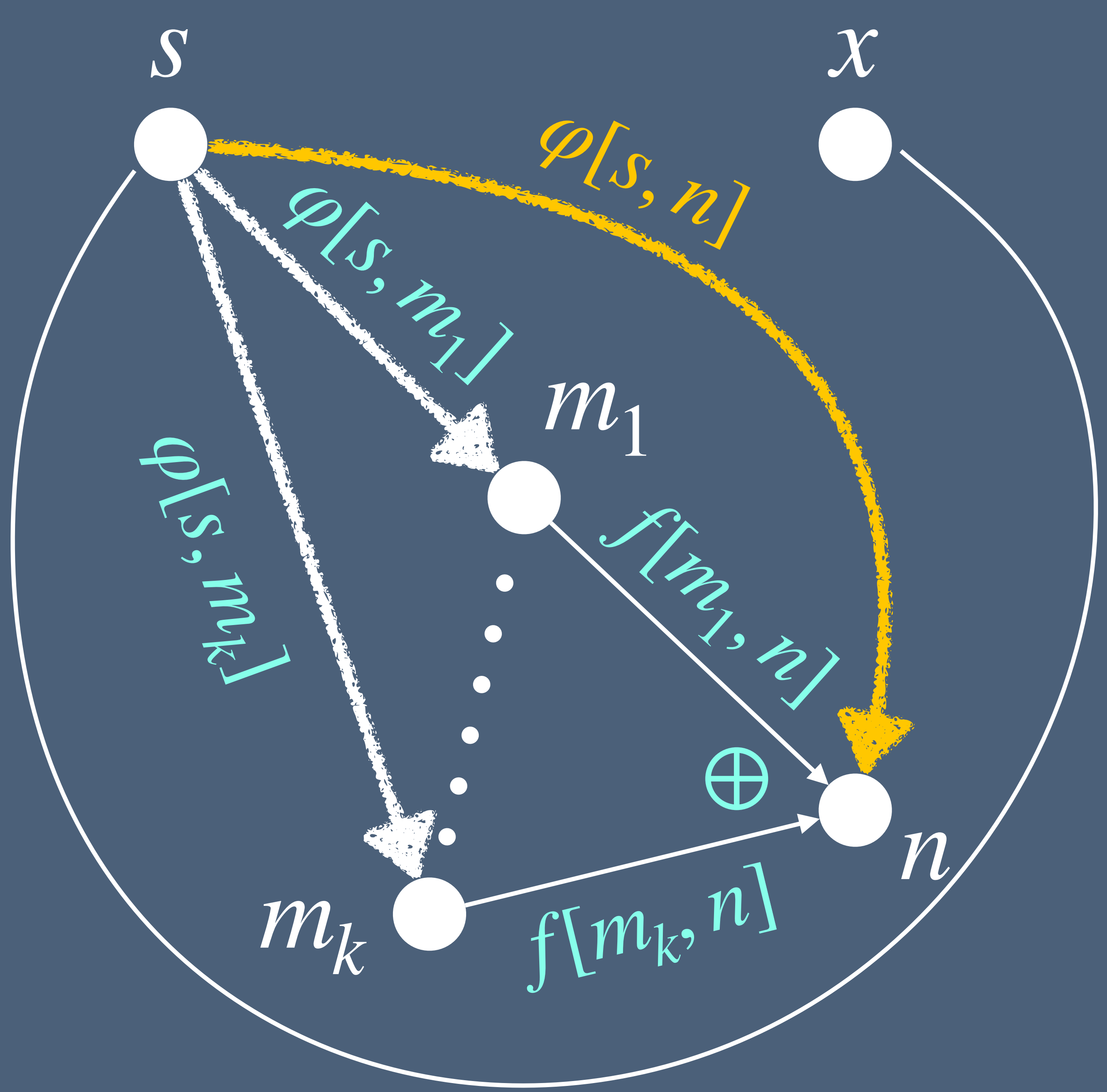
$$\varphi[s,s] = Id$$

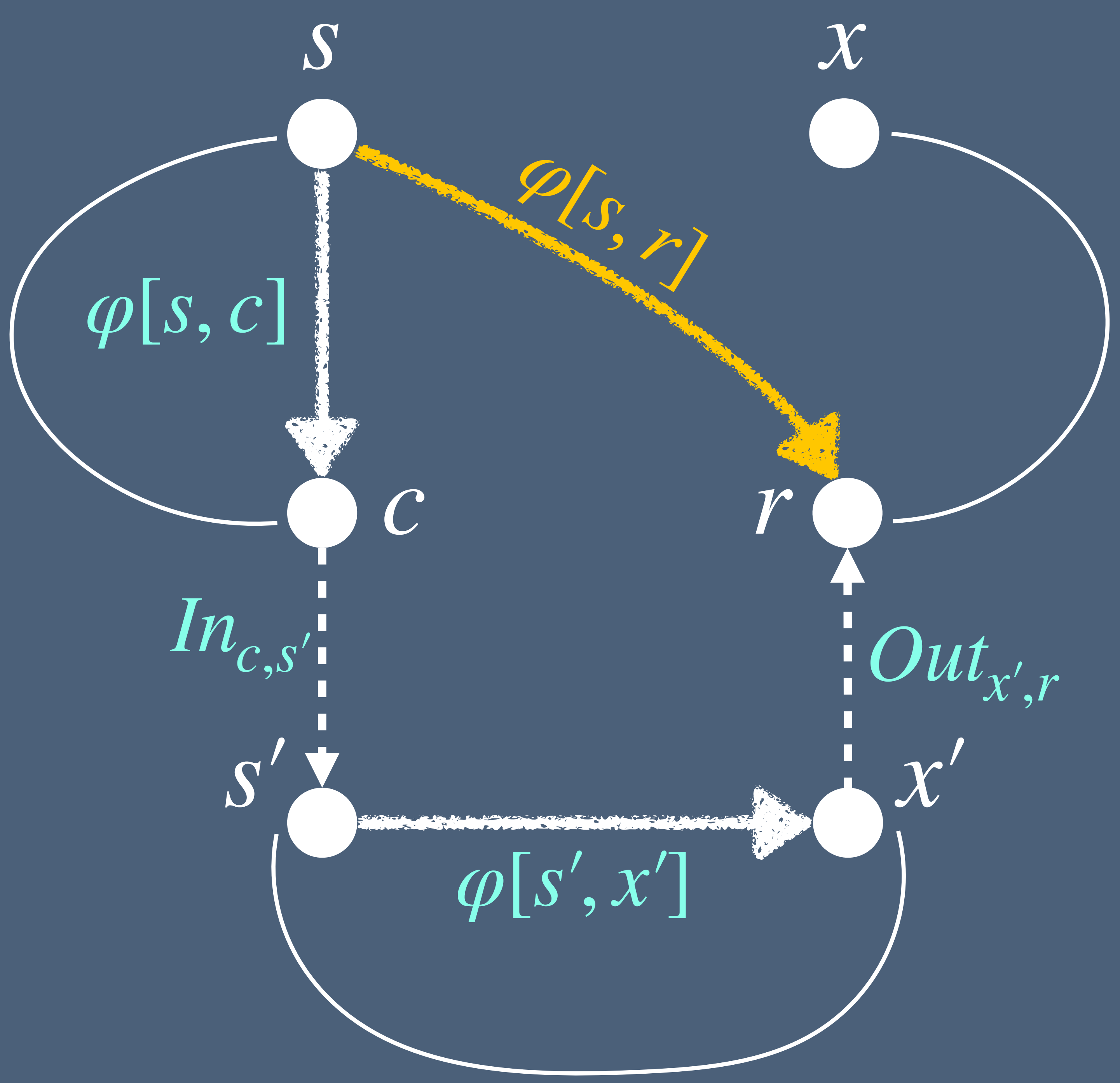
$$\begin{aligned}\varphi[s,n] = {} & (\varphi[s,m_1] \otimes f[m_1,n]) \\ & \oplus \ldots \\ & \oplus (\varphi[s,m_k] \otimes f[m_k,n])\end{aligned}$$

The Functional Approach

[Sharir and Pnueli 1981]







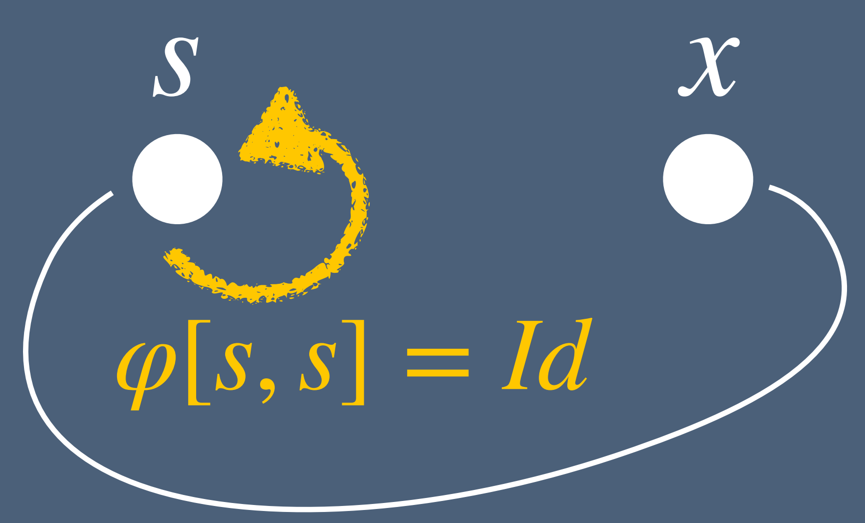
$$\varphi[s,s] = Id$$

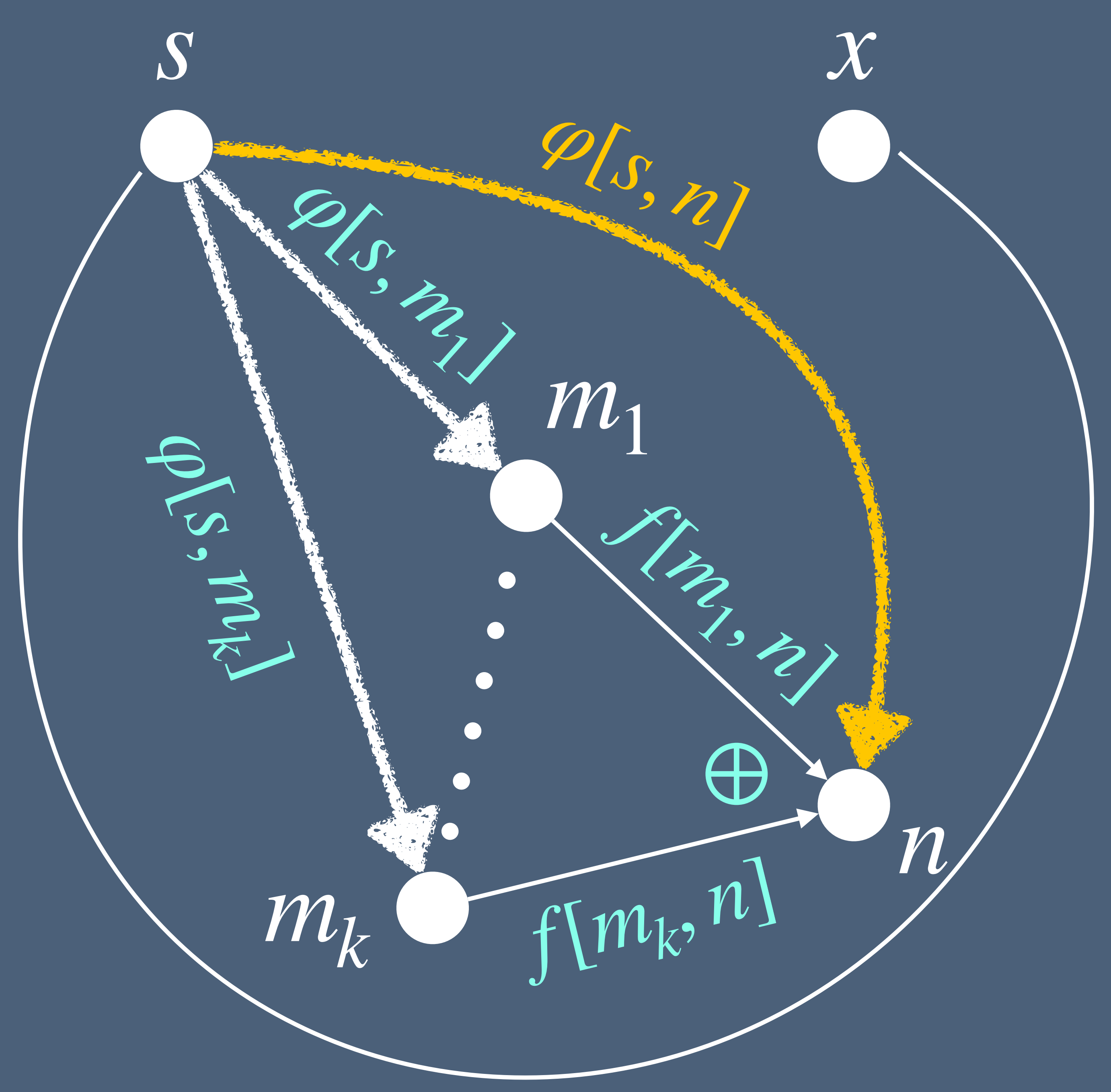
$$\begin{aligned}\varphi[s,n] = {} & (\varphi[s,m_1] \otimes f[m_1,n]) \\ & \oplus \ldots \\ & \oplus (\varphi[s,m_k] \otimes f[m_k,n])\end{aligned}$$

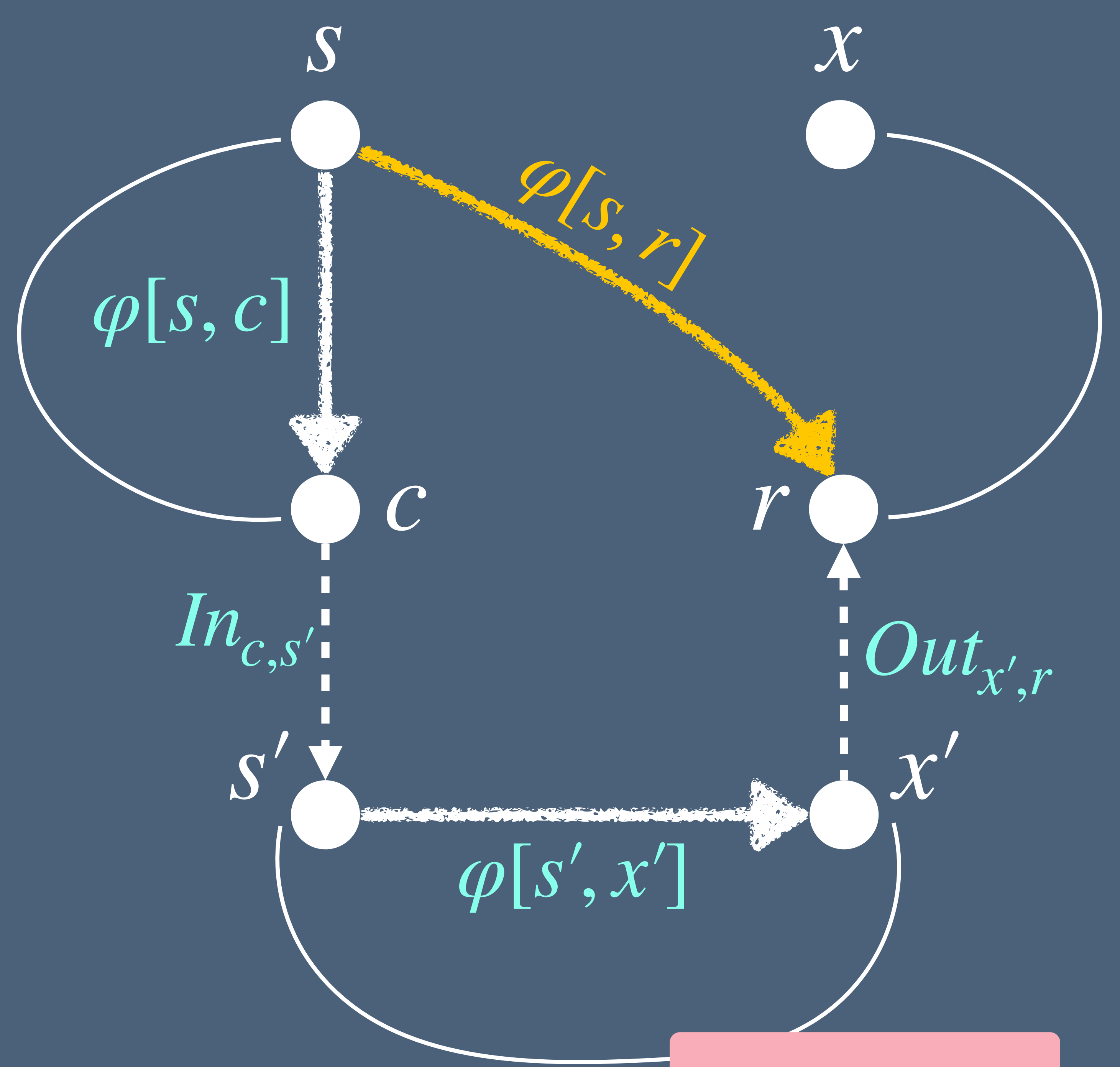
$$\begin{aligned}\varphi[s,r] = {} & \varphi[s,c] \otimes In_{c,s'} \\ & \otimes \varphi[s',x'] \otimes Out_{x',r}\end{aligned}$$

The Functional Approach

[Sharir and Pnueli 1981]







$$\varphi[s,s] = Id$$

$$\begin{aligned}\varphi[s,n] = {} & (\varphi[s,m_1] \otimes f[m_1,n]) \\ & \oplus \ldots \\ & \oplus (\varphi[s,m_k] \otimes f[m_k,n])\end{aligned}$$

procedure summary

$$\begin{aligned}\varphi[s,r] = {} & \varphi[s,c] \otimes In_{c,s'} \\ & \otimes \boxed{\varphi[s',x']} \otimes Out_{x',r}\end{aligned}$$

The Functional Approach

[Sharir and Pnueli 1981]

The Functional Approach

[Sharir and Pnueli 1981]

- Let X_i represent the **procedure summary** of P_i, i.e., $\varphi[s_i, x_i]$

The Functional Approach

[Sharir and Pnueli 1981]

- Let X_i represent the **procedure summary** of P_i, i.e., $\varphi[s_i, x_i]$

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N) \\
X_2 &= f_2(X_1, X_2, \ldots, X_N) \\
&\vdots \\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$

The Functional Approach

[Sharir and Pnueli 1981]

- Let X_i represent the **procedure summary** of P_i, i.e., $\varphi[s_i, x_i]$
- Solve the equation system using successive approximation (Kleene or chaotic)

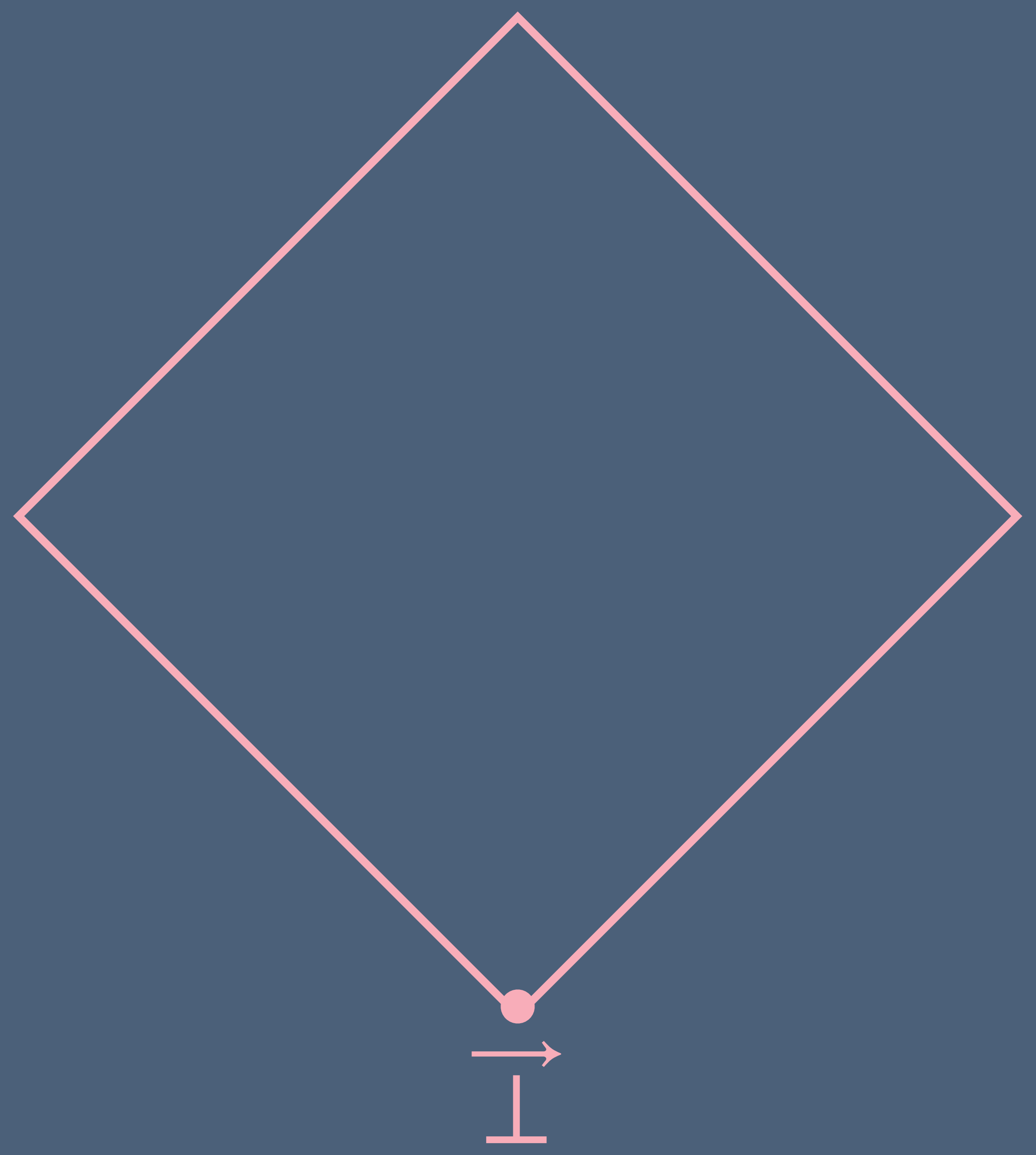
$$\begin{aligned} X_1 &= f_1(X_1, X_2, \ldots, X_N) \\ X_2 &= f_2(X_1, X_2, \ldots, X_N) \\ &\vdots \\ X_N &= f_N(X_1, X_2, \ldots, X_N) \end{aligned}$$

The Functional Approach

[Sharir and Pnueli 1981]

- Let X_i represent the **procedure summary** of P_i, i.e., $\varphi[s_i, x_i]$
- Solve the equation system using successive approximation (Kleene or chaotic)

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N)\\
X_2 &= f_2(X_1, X_2, \ldots, X_N)\\
&\vdots\\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$

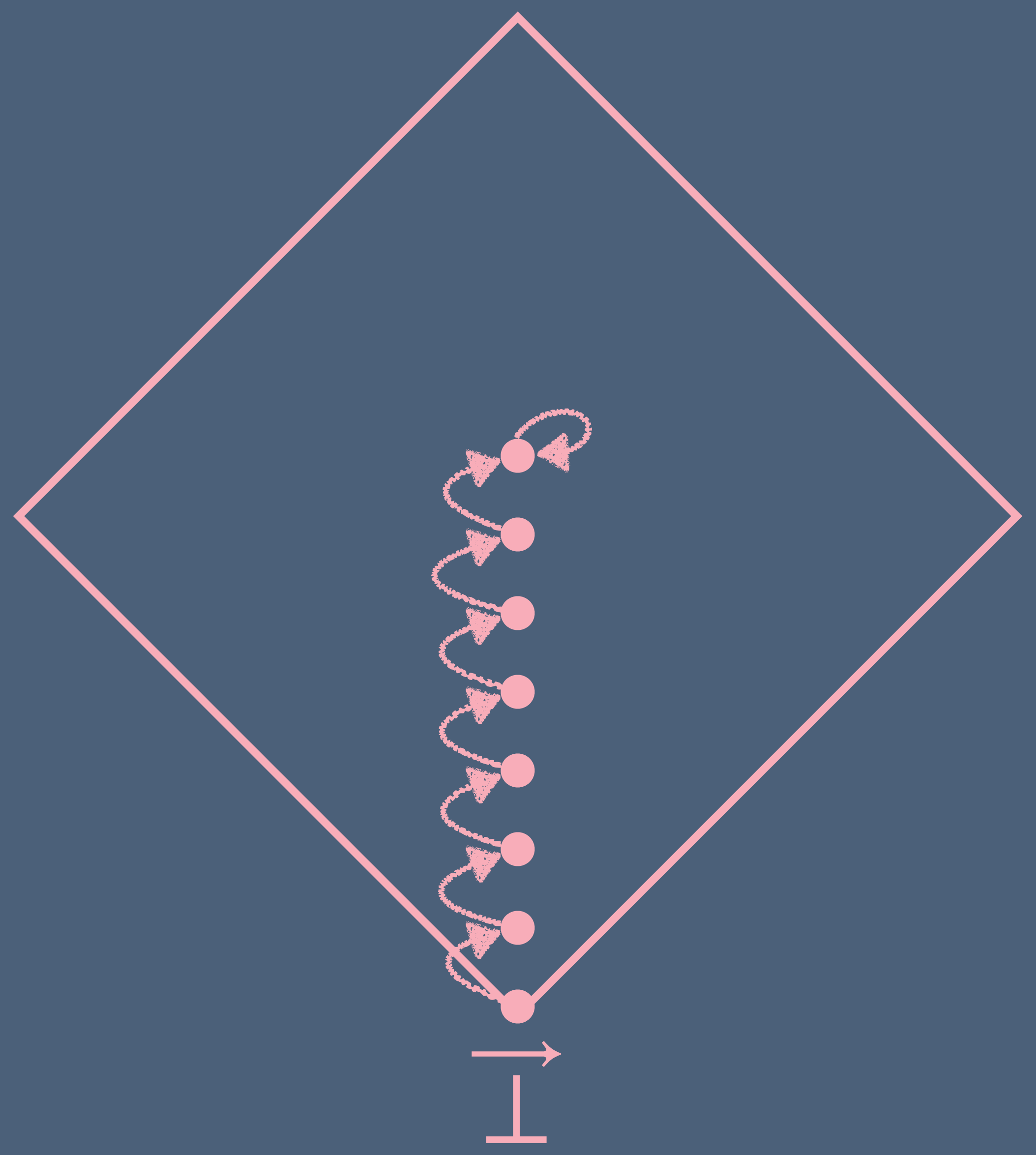


The Functional Approach

[Sharir and Pnueli 1981]

- Let X_i represent the **procedure summary** of P_i, i.e., $\varphi[s_i, x_i]$
- Solve the equation system using successive approximation (Kleene or chaotic)

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N)\\
X_2 &= f_2(X_1, X_2, \ldots, X_N)\\
&\vdots\\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$



Termination-Probability Analysis

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$
  - Transformers: $\underline{\texttt{skip}}$ = 1, $\underline{\texttt{prob(1/3)}}$ = 1/3

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$
  - Transformers: $\underline{\texttt{skip}}$ = 1, $\underline{\texttt{prob(1/3)}}$ = 1/3
  - Extend ($\otimes$): Multiplication

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$
  - Transformers: $\underline{\texttt{skip}}$ = 1, $\underline{\texttt{prob(1/3)}}$ = 1/3
  - Extend ($\otimes$): Multiplication
  - Combine ($\oplus$): Addition

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$
  - Transformers: `skip` = 1, `prob(1/3)` = 1/3
  - Extend ($\otimes$): Multiplication
  - Combine ($\oplus$): Addition

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

- Abstract semantics: Termination probability on $[0,1]$
  - Transformers: `skip` = 1, `prob(1/3)` = 1/3
  - Extend ($\otimes$): Multiplication
  - Combine ($\oplus$): Addition

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

$$\begin{aligned}
\kappa^{(0)} &= 0 \\
\kappa^{(1)} &= \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \kappa^{(0)} \cdot \kappa^{(0)} = \frac{1}{3} \\
\kappa^{(2)} &= \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \kappa^{(1)} \cdot \kappa^{(1)} = \frac{11}{27} \\
&\vdots \\
\kappa^{(\infty)} &= \frac{1}{2}
\end{aligned}$$

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

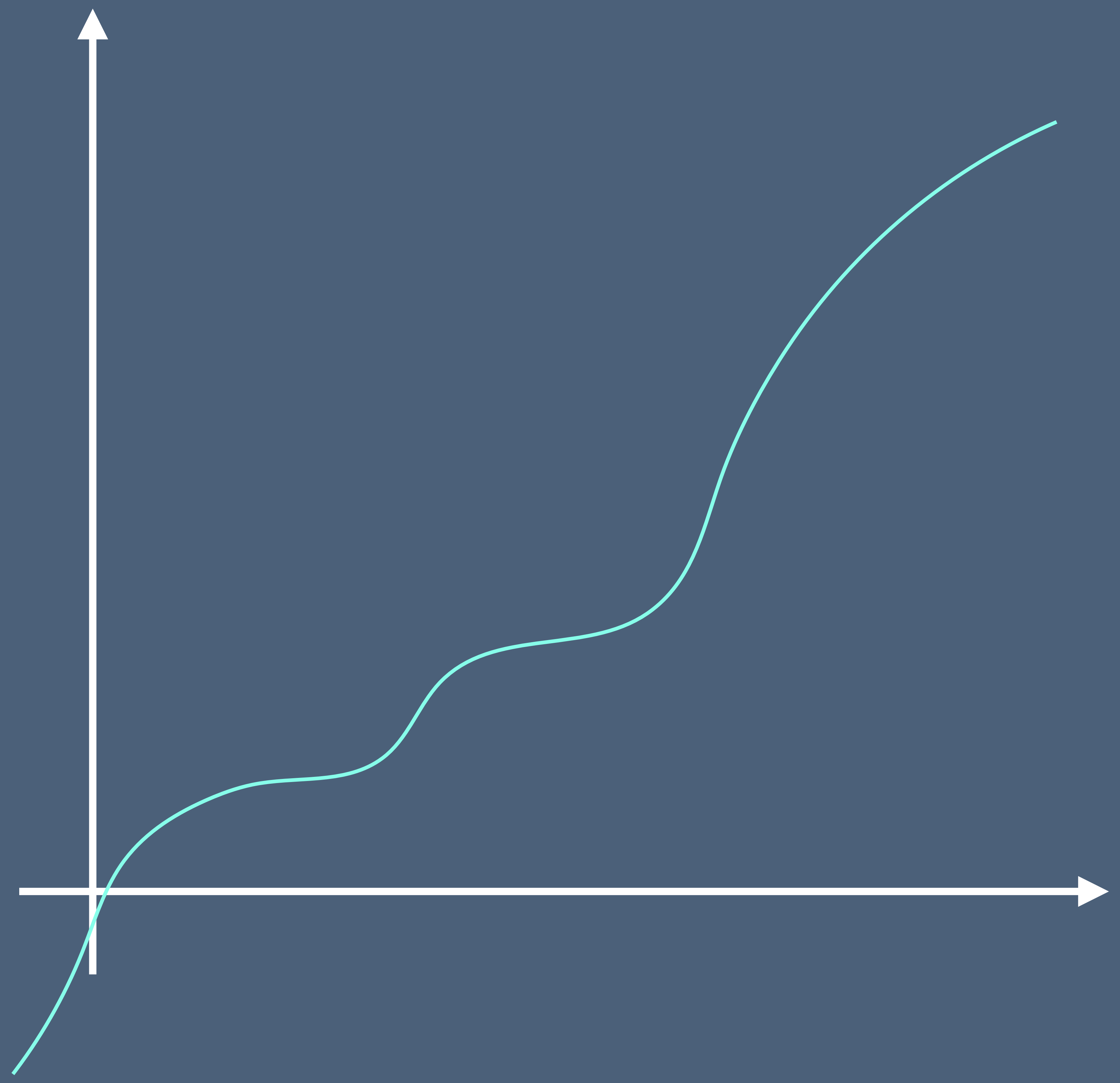
Newton's Method for Finding Roots

Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function

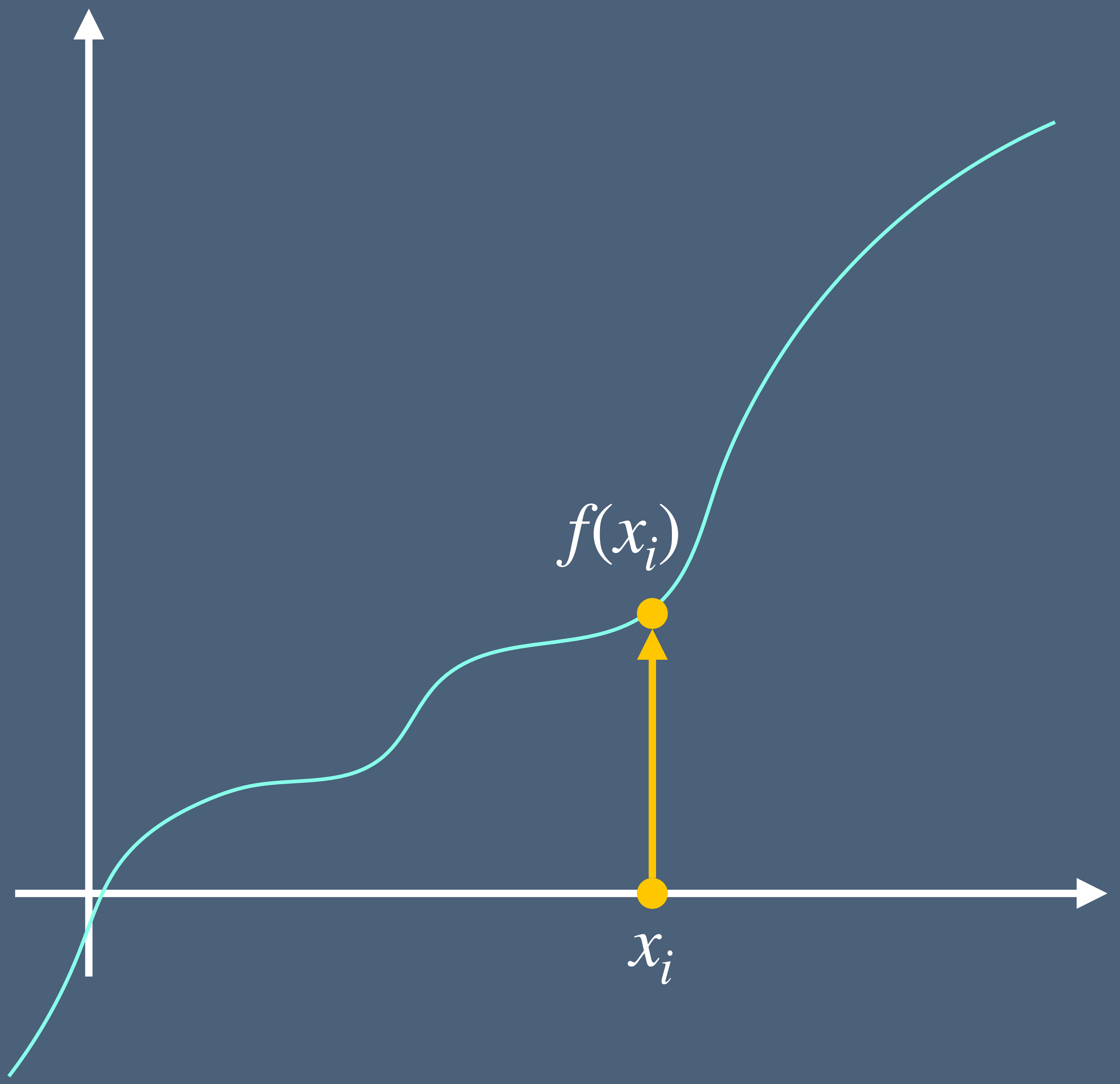
Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function



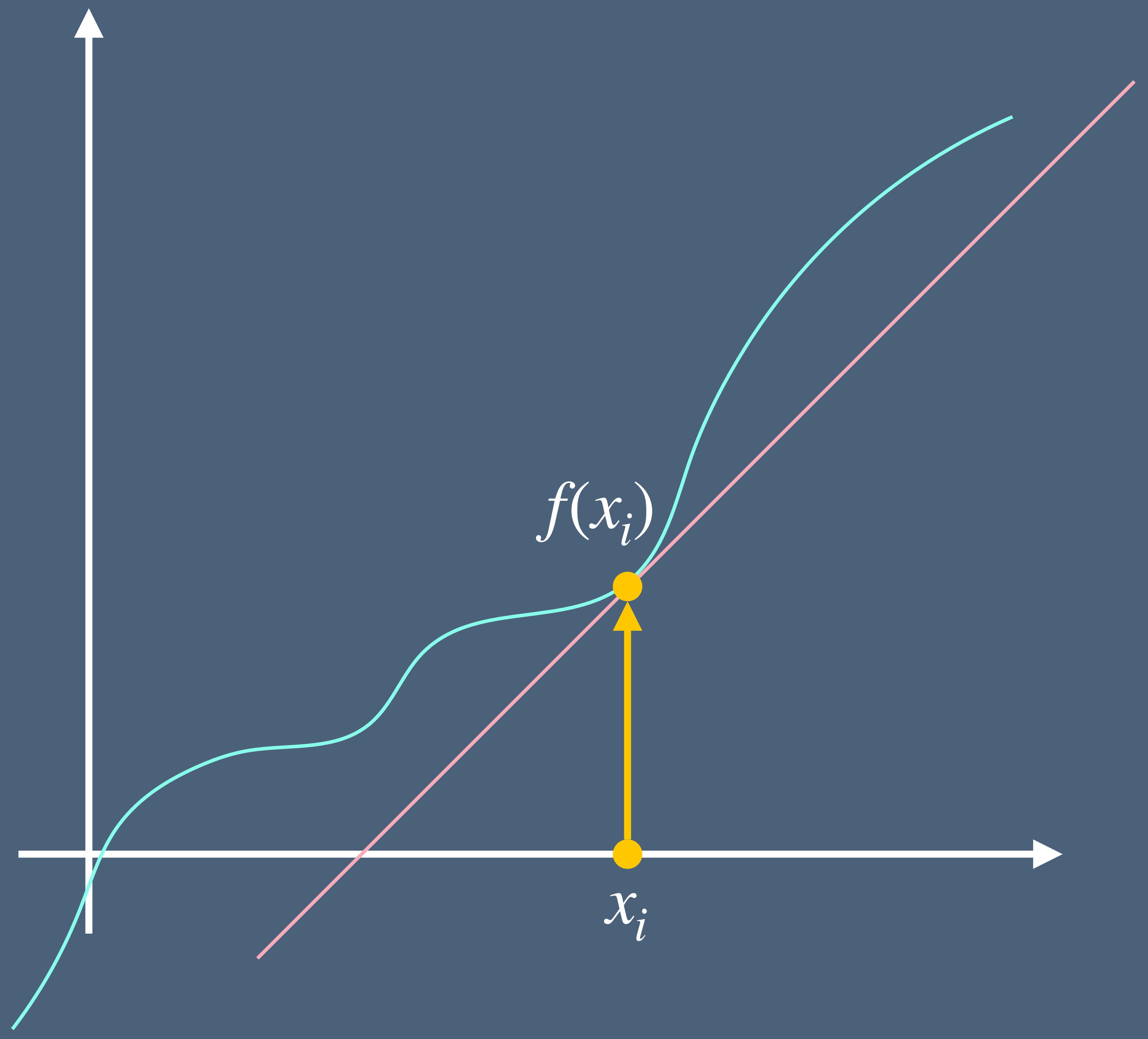
Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function



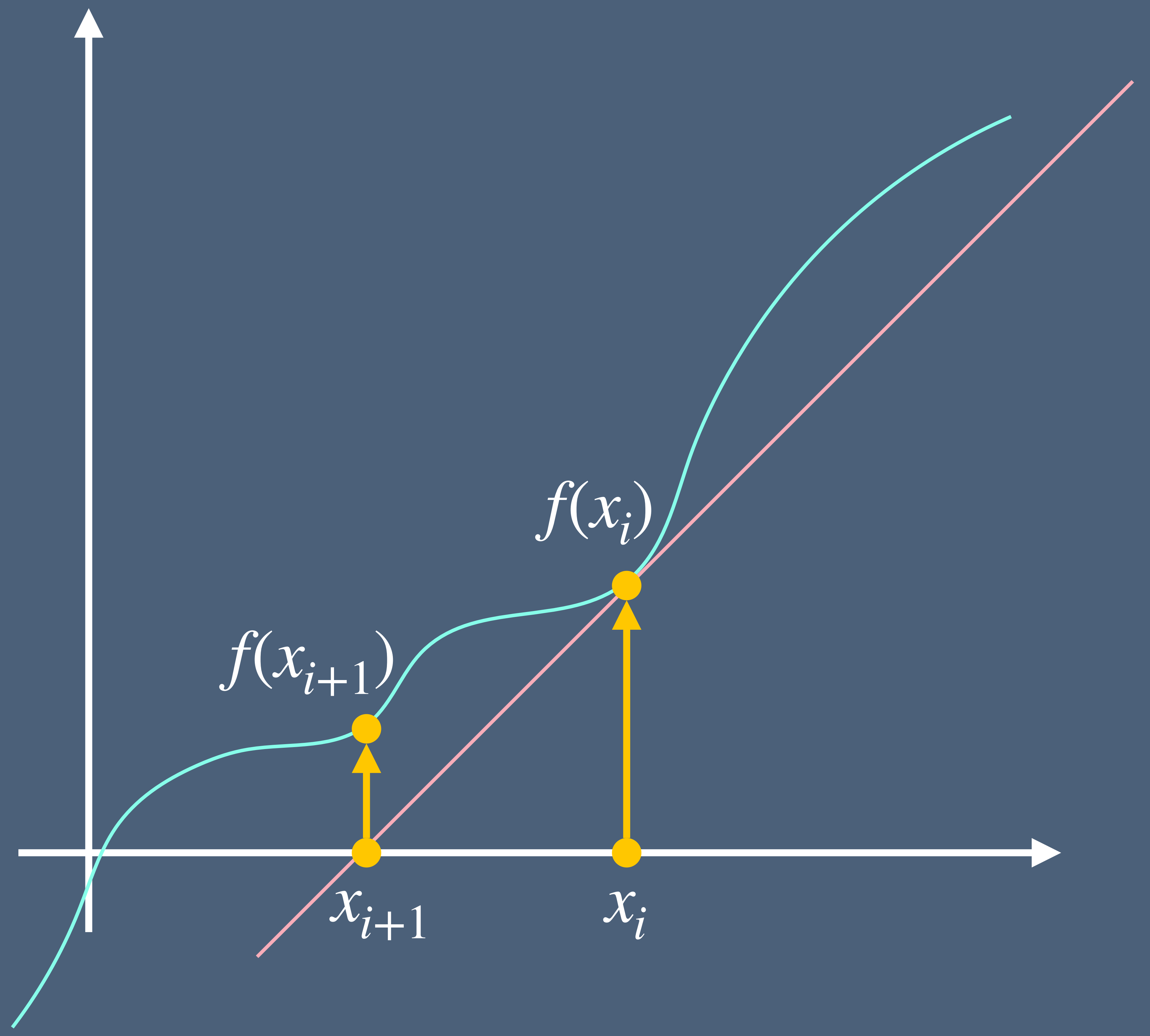
Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function



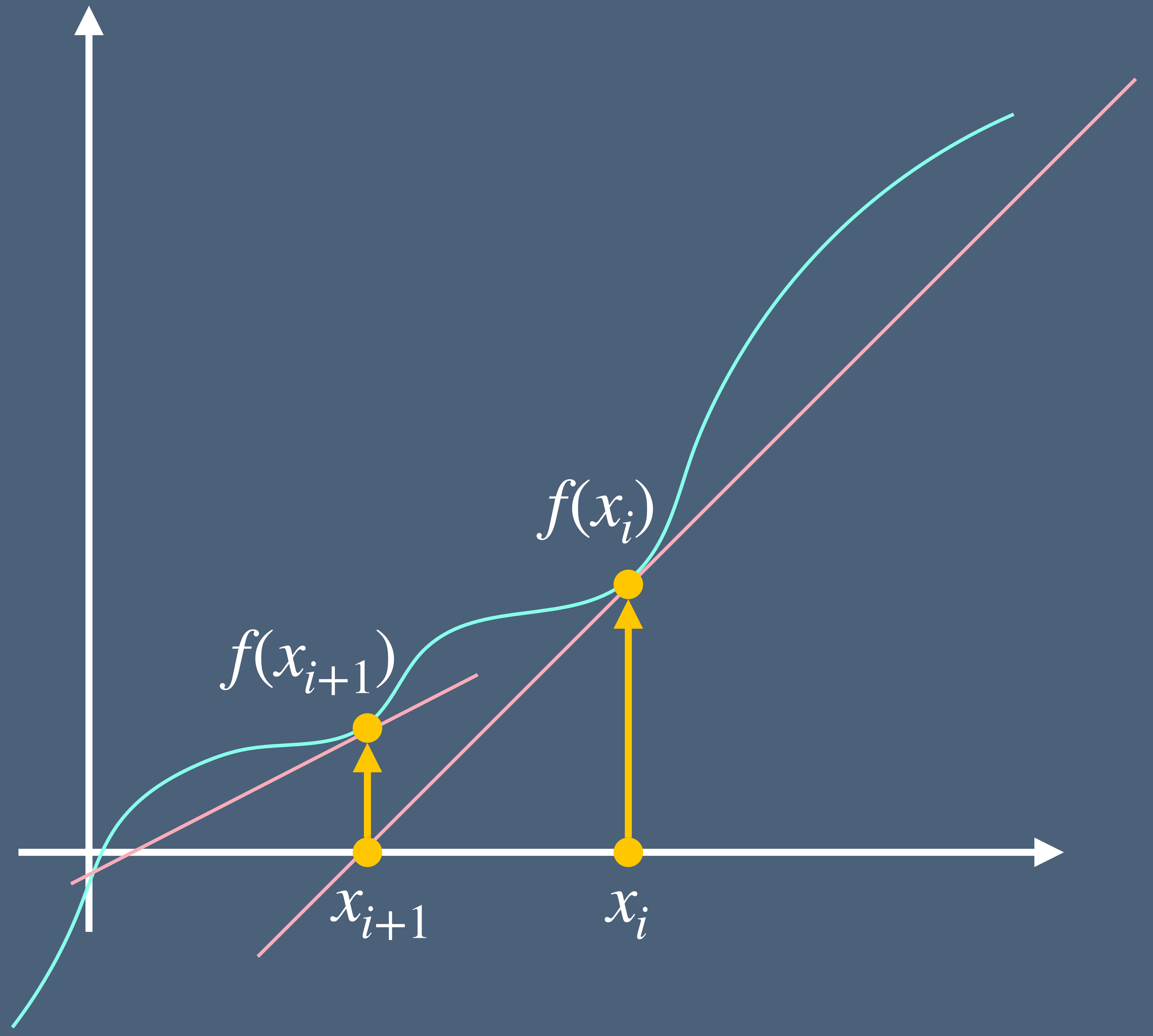
Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function



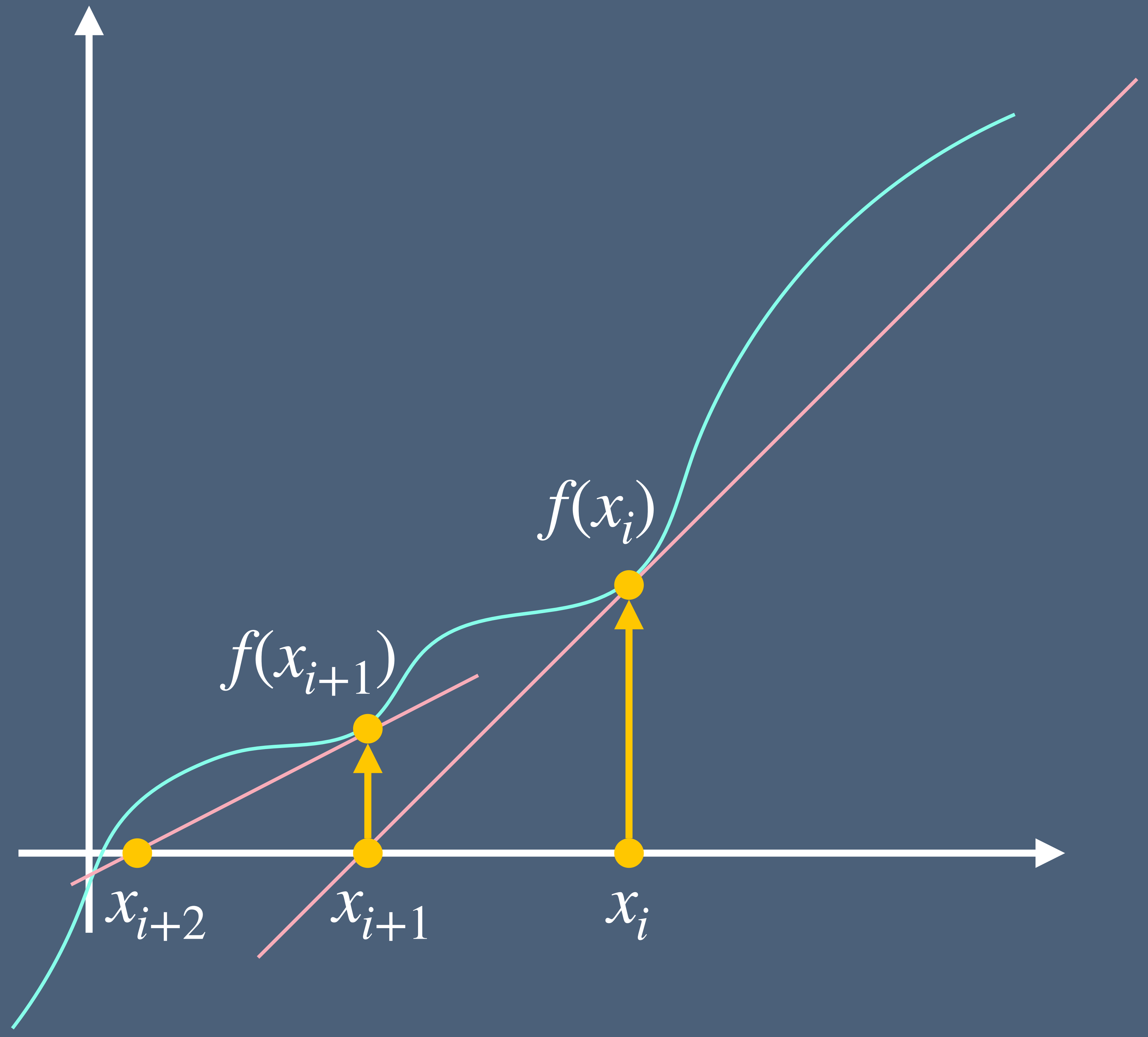
Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function



Newton's Method for Finding Roots

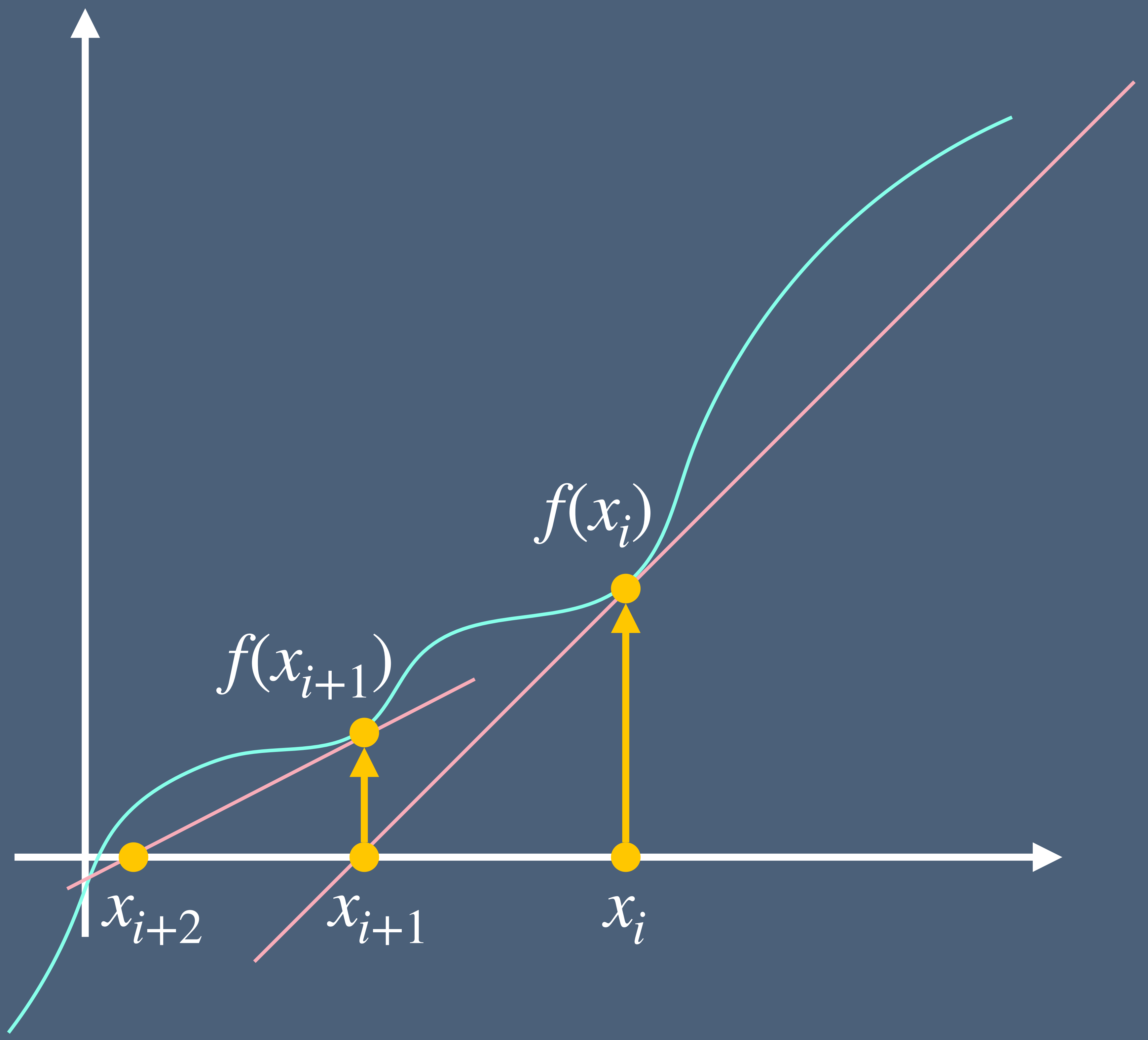
- A way to find **successively** better approximations of a root a function



Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function

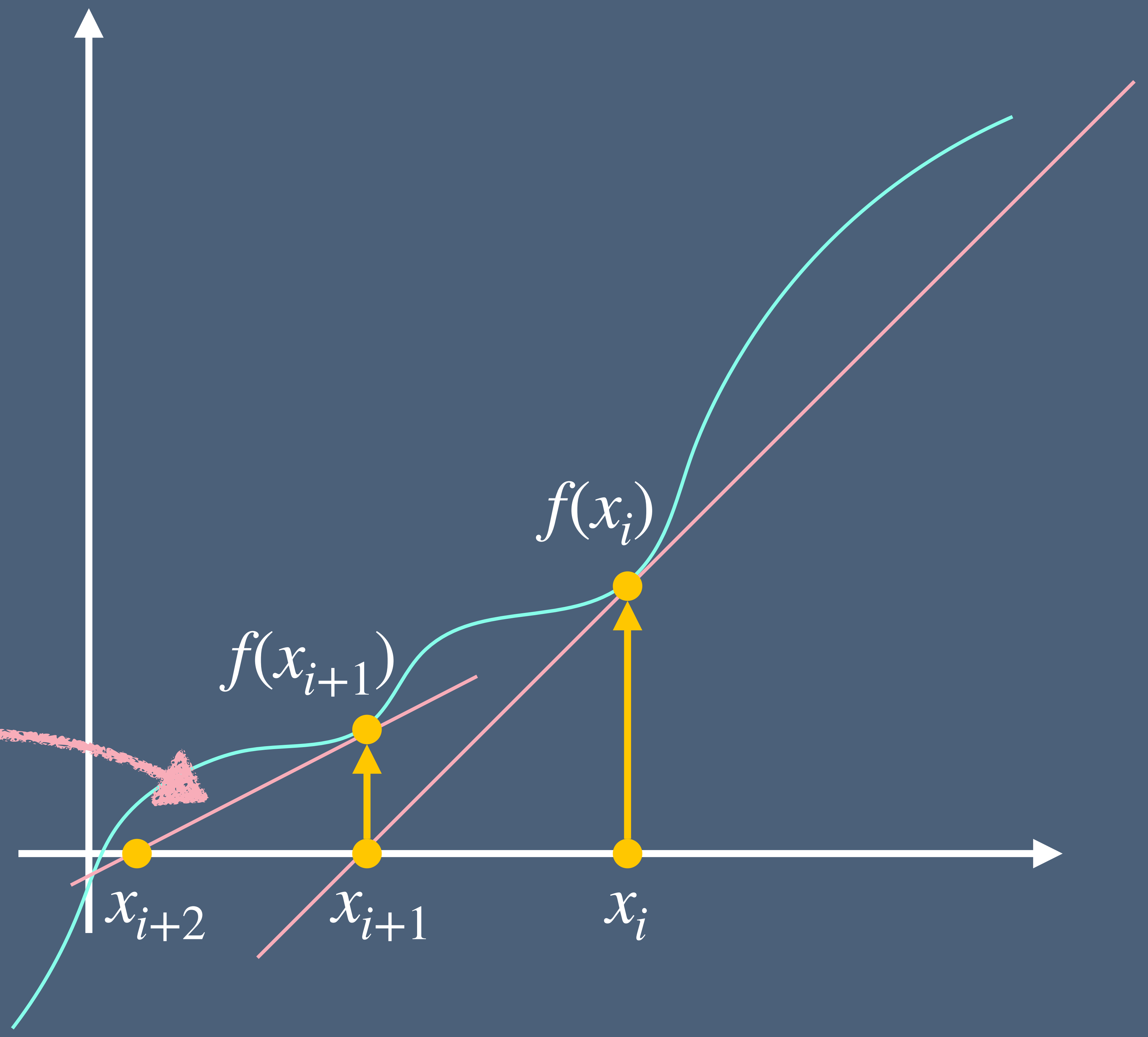
- Given a function f, its derivative f' and an initial x_0, repeatedly perform $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$



Newton's Method for Finding Roots

- A way to find **successively** better approximations of a root a function
- Given a function f, its derivative f' and an initial x_0, repeatedly perform $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$

Create a **linear model** to find a better approximation



Termination-Probability Analysis

via Newton's Method

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

Termination-Probability Analysis

via Newton's Method

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

- Reformulate the problem as root-finding:

$$f(X) = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X - X$$

$$f'(X) = \frac{4}{3} \cdot X - 1$$

Termination-Probability Analysis

via Newton's Method

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

- Reformulate the problem as root-finding:

$$f(X) = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X - X$$

$$f'(X) = \frac{4}{3} \cdot X - 1$$

- Newton's method:

$$\nu^{(i+1)} = \nu^{(i)} - \frac{f(\nu^{(i)})}{f'(\nu^{(i)})} = \frac{2{\nu^{(i)}}^{2} - 1}{4\nu^{(i)} - 3}$$

Termination-Probability Analysis

via Newton's Method

- Reformulate the problem as root-finding:

$$f(X) = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X - X$$

$$f'(X) = \frac{4}{3} \cdot X - 1$$

- Newton's method:

$$\nu^{(i+1)} = \nu^{(i)} - \frac{f(\nu^{(i)})}{f'(\nu^{(i)})} = \frac{2\nu^{(i)^2} - 1}{4\nu^{(i)} - 3}$$

$$X = \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot X \cdot X$$

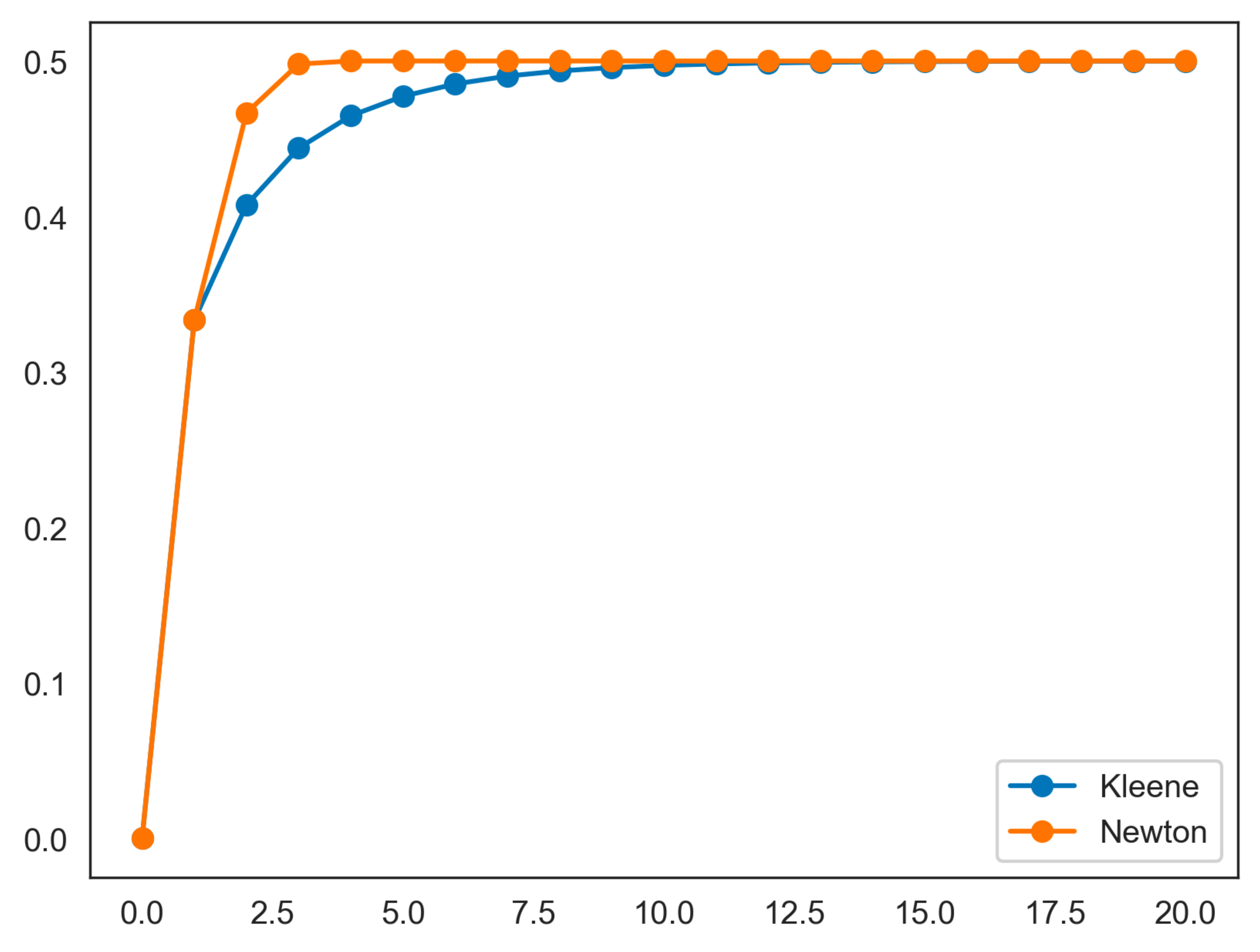
$$\begin{aligned}
\nu^{(0)} &= 0 \\
\nu^{(1)} &= \frac{2\nu^{(0)^2} - 1}{4\nu^{(0)} - 3} = \frac{1}{3} \\
\nu^{(2)} &= \frac{2\nu^{(1)^2} - 1}{4\nu^{(1)} - 3} = \frac{7}{15} \\
&\vdots \\
\nu^{(\infty)} &= \frac{1}{2}
\end{aligned}$$

Kleene vs Newton

which converges faster?

Kleene vs Newton

which converges faster?



Newton's Method for Program Analysis

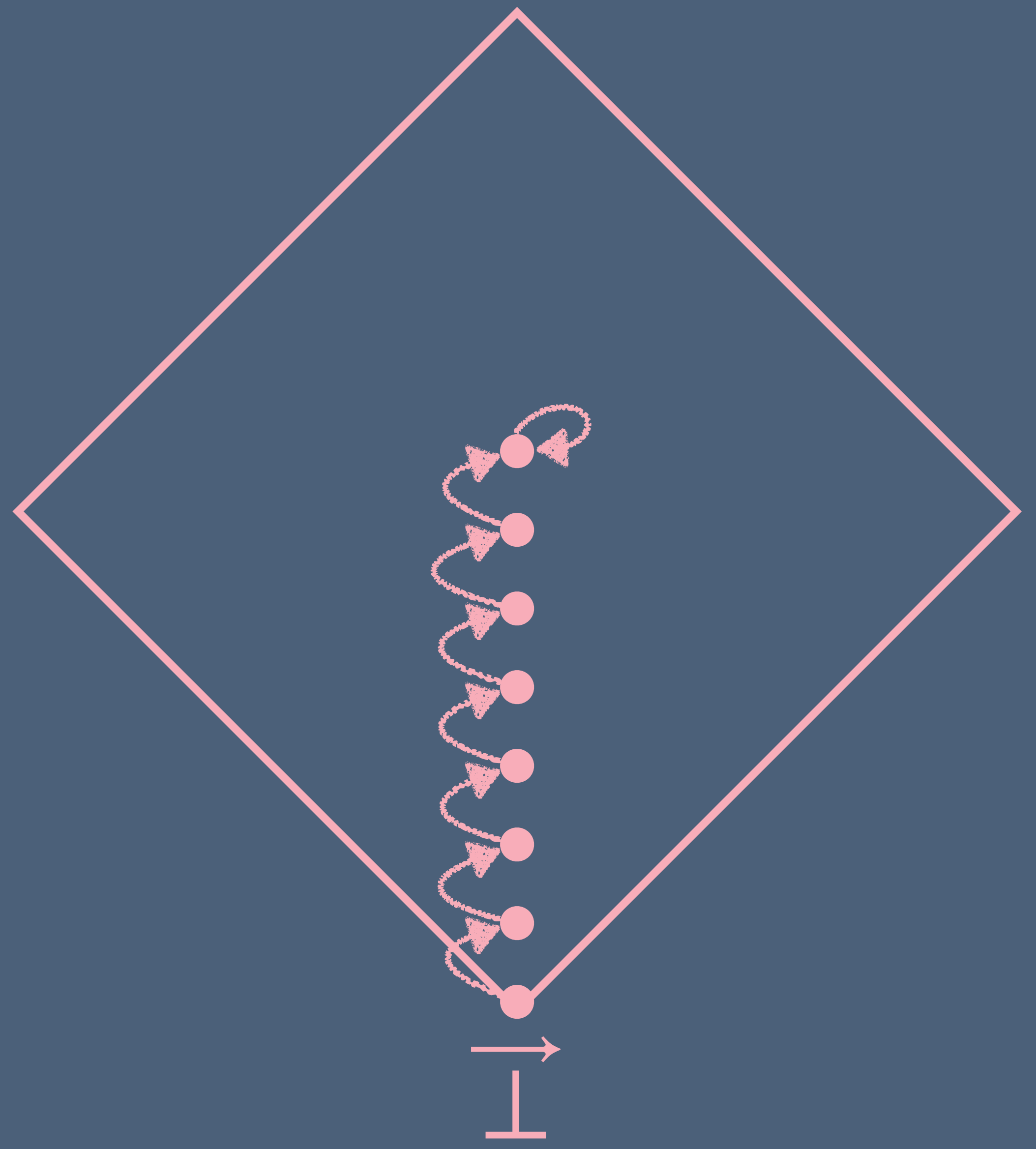
[Esparza, Kiefer, and Luttenberger 2008]

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- **Kleene iteration**

$$\begin{aligned}\vec{\kappa}^{(0)} &= \vec{\bot}\\ \vec{\kappa}^{(i+1)} &= \vec{f}(\vec{\kappa}^{(i)})\end{aligned}$$



Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- **Kleene iteration**

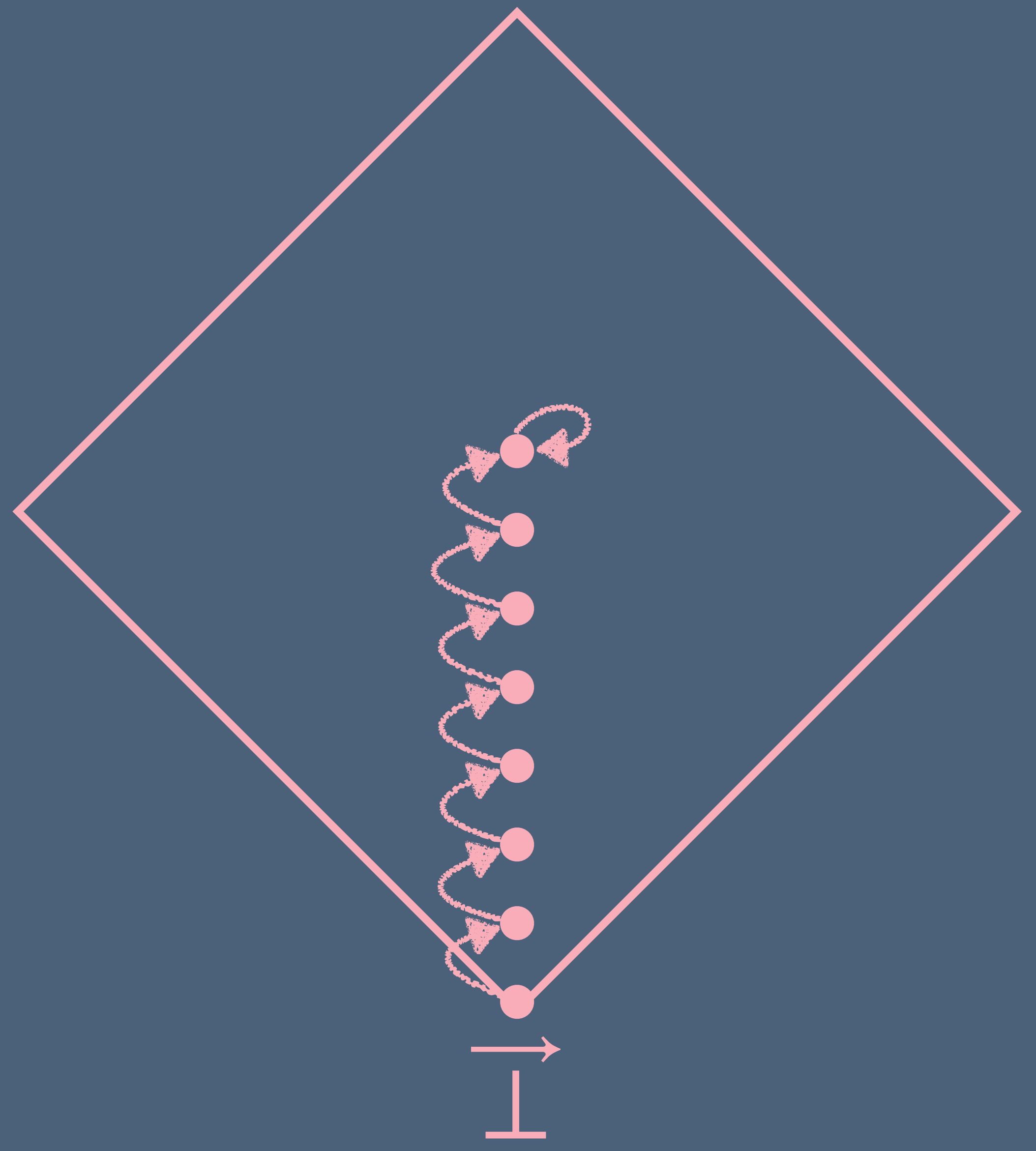
$$\vec{\kappa}^{(0)} = \vec{\bot}$$

$$\vec{\kappa}^{(i+1)} = \vec{f}(\vec{\kappa}^{(i)})$$

- **Newton iteration**

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathit{LinearCorrectionTerm}(\vec{f}, \vec{\nu}^{(i)})$$



Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- **Kleene iteration**

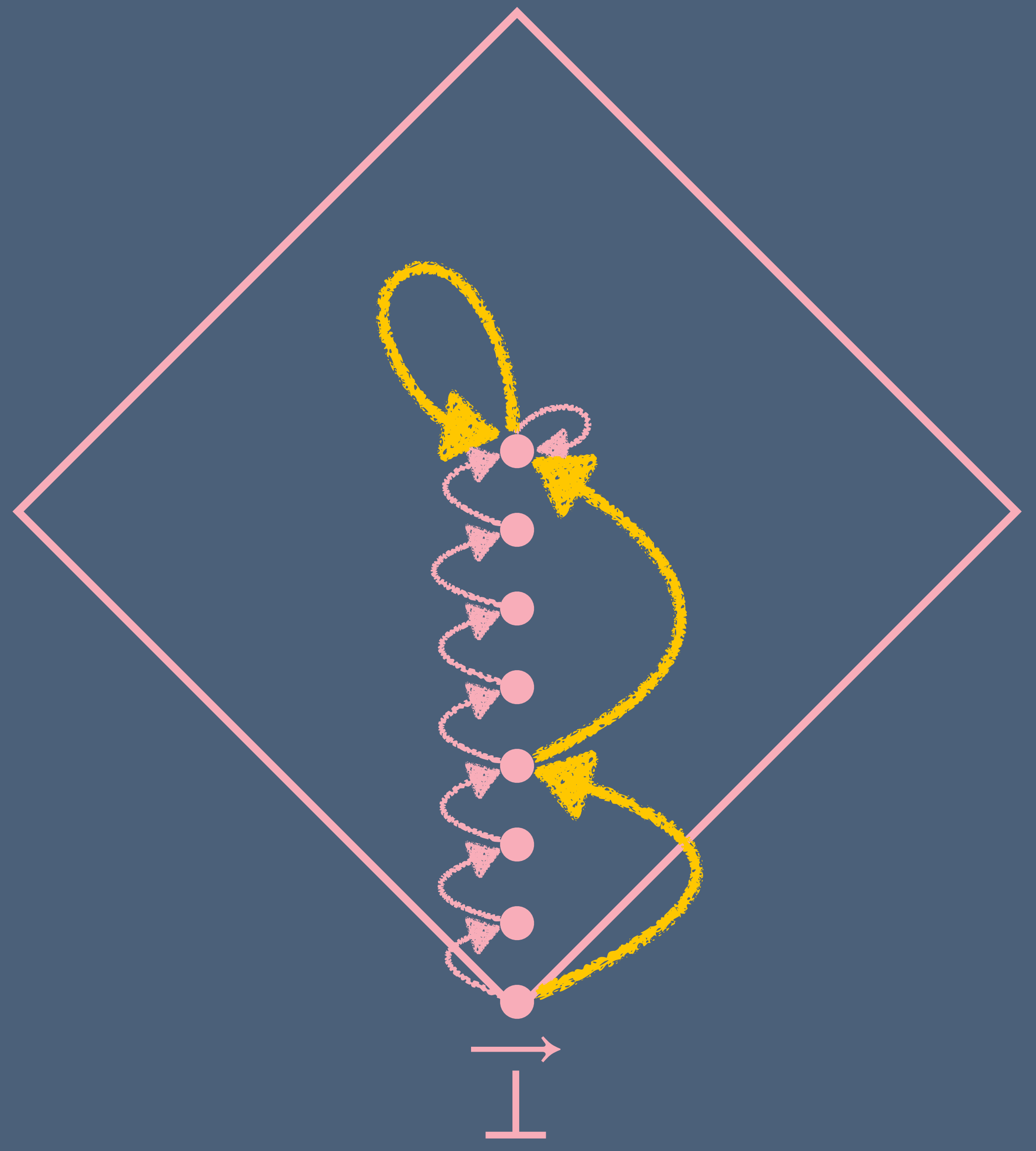
$$\vec{\kappa}^{(0)} = \vec{\bot}$$

$$\vec{\kappa}^{(i+1)} = \vec{f}(\vec{\kappa}^{(i)})$$

- **Newton iteration**

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{f}(\vec{\nu}^{(i)}) \oplus \mathit{LinearCorrectionTerm}(\vec{f}, \vec{\nu}^{(i)})$$



Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Esparza et al. had to tackle several issues:

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Esparza et al. had to tackle several issues:
  - Real-valued equations → **Algebraic semiring**
    - Numeric multiplication → Extend ($\otimes$)
    - Numeric addition → Combine ($\oplus$)

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Esparza et al. had to tackle several issues:
    - Real-valued equations → **Algebraic semiring**
        - Numeric multiplication → Extend ($\otimes$)
        - Numeric addition → Combine ($\oplus$)
    - **Root finding vs fixed-point finding?**

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Esparza et al. had to tackle several issues:
  - Real-valued equations → **Algebraic semiring**
    - Numeric multiplication → Extend ($\otimes$)
    - Numeric addition → Combine ($\oplus$)
  - **Root finding vs fixed-point finding?**
  - **Derivatives?**

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

$a \ominus b$ is some c such that $b \oplus c = a$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

Linear correction term

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

$a \ominus b$ is some c such that $b \oplus c = a$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Linear correction term

$a \ominus b$ is some c such that $b \oplus c = a$

Really a differential: $f(X) \xrightarrow{\nu} f'(\nu) \otimes Y$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Linear correction term

$a \ominus b$ is some c such that $b \oplus c = a$

Really a differential: $f(X) \xrightarrow{\nu} f'(\nu) \otimes Y$

$$D\,\text{const}\,|_\nu(Y) = \underline{0}$$

$$D\,X\,|_\nu(Y) = Y$$

$$D(g(X) \oplus h(X))\,|_\nu(Y) = Dg(X)\,|_\nu(Y) \oplus Dh(X)\,|_\nu(Y)$$

$$D(g(X) \otimes h(X))\,|_\nu(Y) = (Dg(X)\,|_\nu(Y) \otimes h(\nu)) \oplus (g(\nu) \otimes Dh(X)\,|_\nu(Y))$$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Linear correction term

$a \ominus b$ is some c such that $b \oplus c = a$

Really a differential: $f(X) \xrightarrow{\nu} f'(\nu) \otimes Y$

$$\begin{aligned}
D\,\text{const}\,|_\nu(Y) &= \underline{0} \\
D\,X\,|_\nu(Y) &= Y \\
D(g(X) \oplus h(X))\,|_\nu(Y) &= Dg(X)\,|_\nu(Y) \oplus Dh(X)\,|_\nu(Y) \\
D(g(X) \otimes h(X))\,|_\nu(Y) &= (Dg(X)\,|_\nu(Y) \otimes h(\nu)) \\
&\quad \oplus (g(\nu) \otimes Dh(X)\,|_\nu(Y))
\end{aligned}$$

Semiring constant $\underline{0}$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\bot}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Linear correction term

$a \ominus b$ is some c such that $b \oplus c = a$

Really a differential: $f(X) \xrightarrow{\nu} f'(\nu) \otimes Y$

$$D\,\mathrm{const}\,|_\nu(Y) = \underline{0}$$

Semiring constant $\underline{0}$

$$D\,X|_\nu(Y) = Y$$

$$D(g(X) \oplus h(X))|_\nu(Y) = Dg(X)|_\nu(Y) \oplus Dh(X)|_\nu(Y)$$

$$D(g(X) \otimes h(X))|_\nu(Y) = (Dg(X)|_\nu(Y) \otimes h(\nu)) \oplus (g(\nu) \otimes Dh(X)|_\nu(Y))$$

$$X \otimes X \xrightarrow{\nu} (Y \otimes \nu) \oplus (\nu \otimes Y)$$

Newton's Method for Program Analysis

[Esparza, Kiefer, and Luttenberger 2008]

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

Leibniz product rule

- Newton iteration for program analysis:

$$\vec{\nu}^{(0)} = \vec{\perp}$$

$$\vec{\nu}^{(i+1)} = \vec{\nu}^{(i)} \oplus \vec{Y}^{(i)}$$

where $\vec{Y}^{(i)}$ is the least solution to

$$\vec{Y} = (\vec{f}(\vec{\nu}^{(i)}) \ominus \vec{\nu}^{(i)}) \oplus D\vec{f}\,|_{\vec{\nu}^{(i)}}(\vec{Y})$$

Linear correction term

$a \ominus b$ is some c such that $b \oplus c = a$

Really a differential: $f(X) \xrightarrow{\nu} f'(\nu) \otimes Y$

$$D\,\text{const}\,|_{\nu}(Y) = \underline{0}$$

Semiring constant $\underline{0}$

$$D\,X\,|_{\nu}(Y) = Y$$

$$D(g(X) \oplus h(X))\,|_{\nu}(Y) = Dg(X)\,|_{\nu}(Y) \oplus Dh(X)\,|_{\nu}(Y)$$

$$D(g(X) \otimes h(X))\,|_{\nu}(Y) = (Dg(X)\,|_{\nu}(Y) \otimes h(\nu)) \oplus (g(\nu) \otimes Dh(X)\,|_{\nu}(Y))$$

$$X \otimes X \xrightarrow{\nu} (Y \otimes \nu) \oplus (\nu \otimes Y)$$

$$b \otimes X \otimes X \otimes c \xrightarrow{\nu} (b \otimes Y \otimes \nu \otimes c) \oplus (b \otimes \nu \otimes Y \otimes c)$$

Termination-Probability Analysis

via Newton's Method for Program Analysis

Termination-Probability Analysis

via Newton's Method for Program Analysis

$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Termination-Probability Analysis

via Newton's Method for Program Analysis

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

Linearization at ν

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$

where

$$\delta = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes \nu) \ominus \nu$$

Termination-Probability Analysis

via Newton's Method for Program Analysis

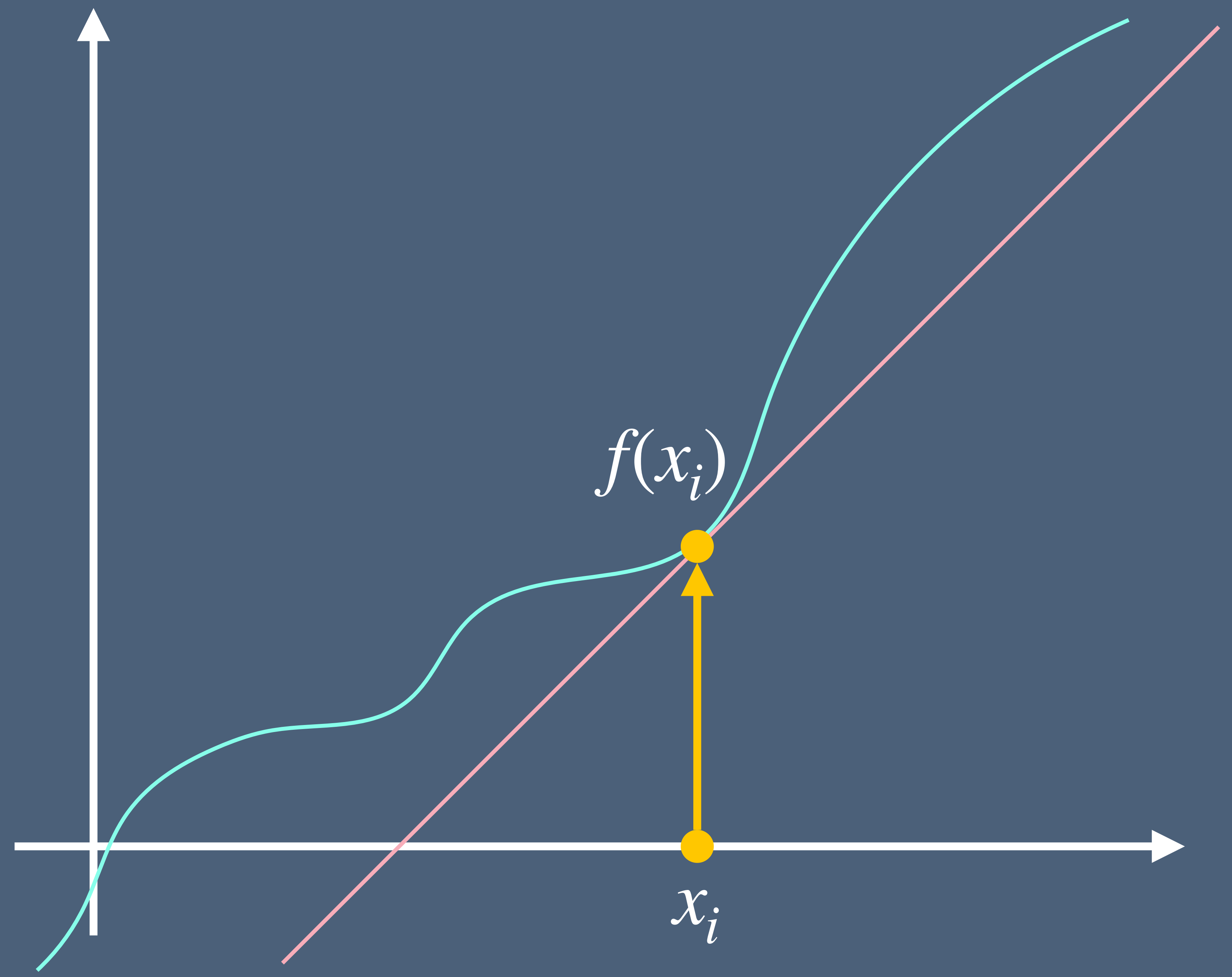
Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$

where

$$\delta = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes \nu) \ominus \nu$$



Termination-Probability Analysis

via Newton's Method for Program Analysis

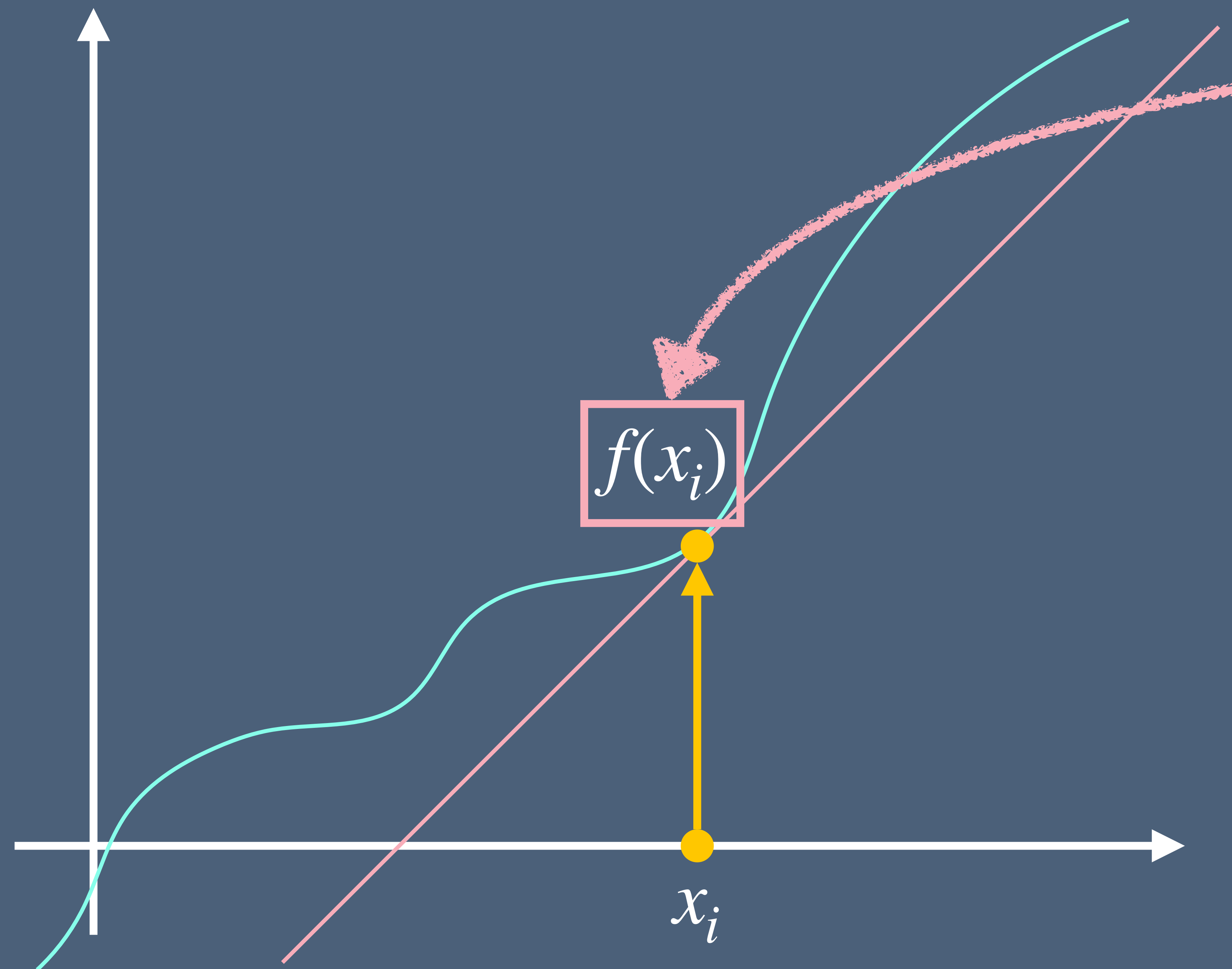
Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$

where

$$\delta = \boxed{(\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes \nu)} \ominus \nu$$



Termination-Probability Analysis

via Newton's Method for Program Analysis

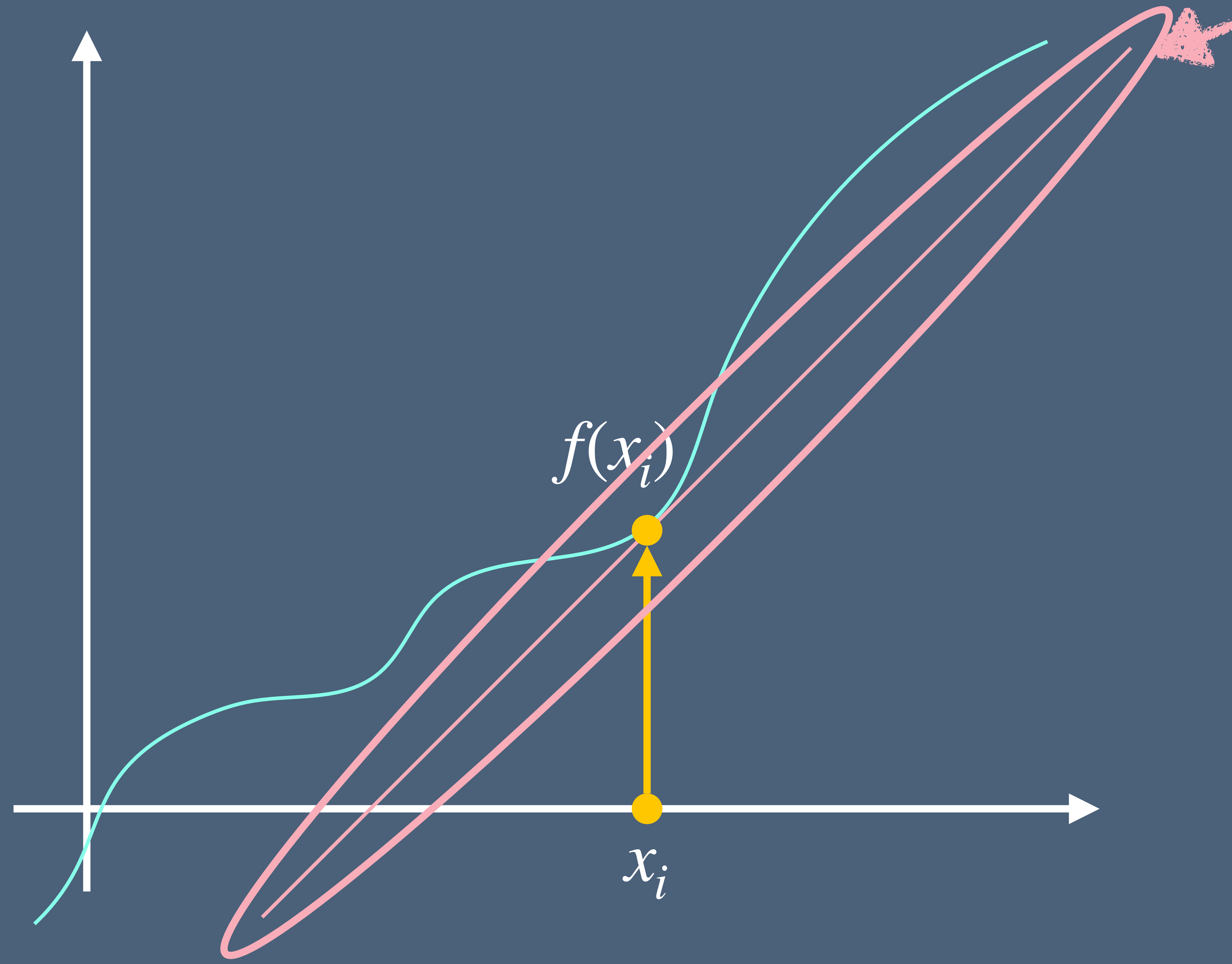
Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus \boxed{(\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)}$$

where

$$\delta = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes \nu) \ominus \nu$$



Termination-Probability Analysis

via Newton's Method for Program Analysis

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

Linearization at ν

$$Y = \delta \oplus \boxed{(\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)}$$

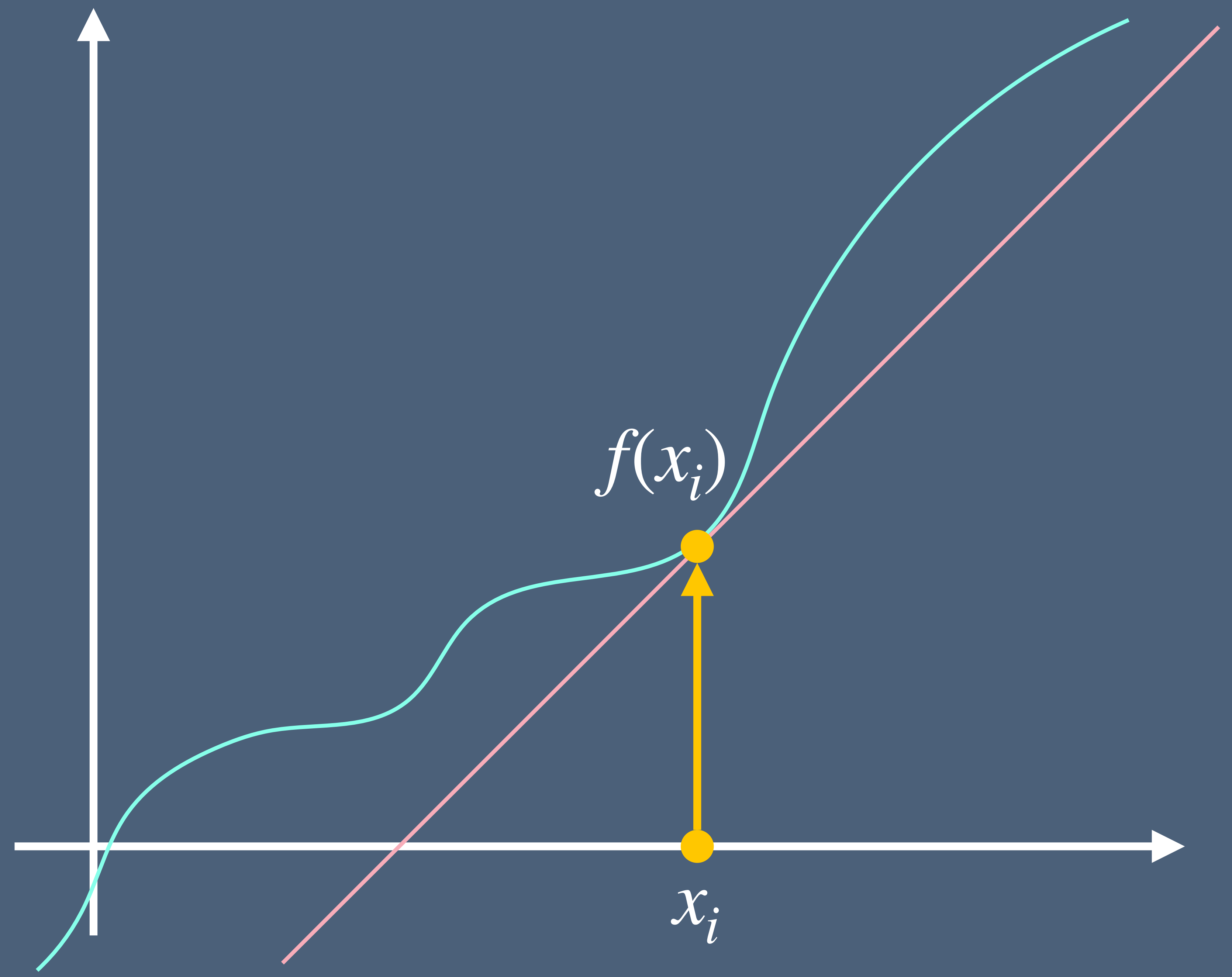
where

$$\delta = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes \nu) \ominus \nu$$

Each summand has only one variable

→

The equation becomes **linear**!



Termination-Probability Analysis

via Newton's Method for Program Analysis

Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$

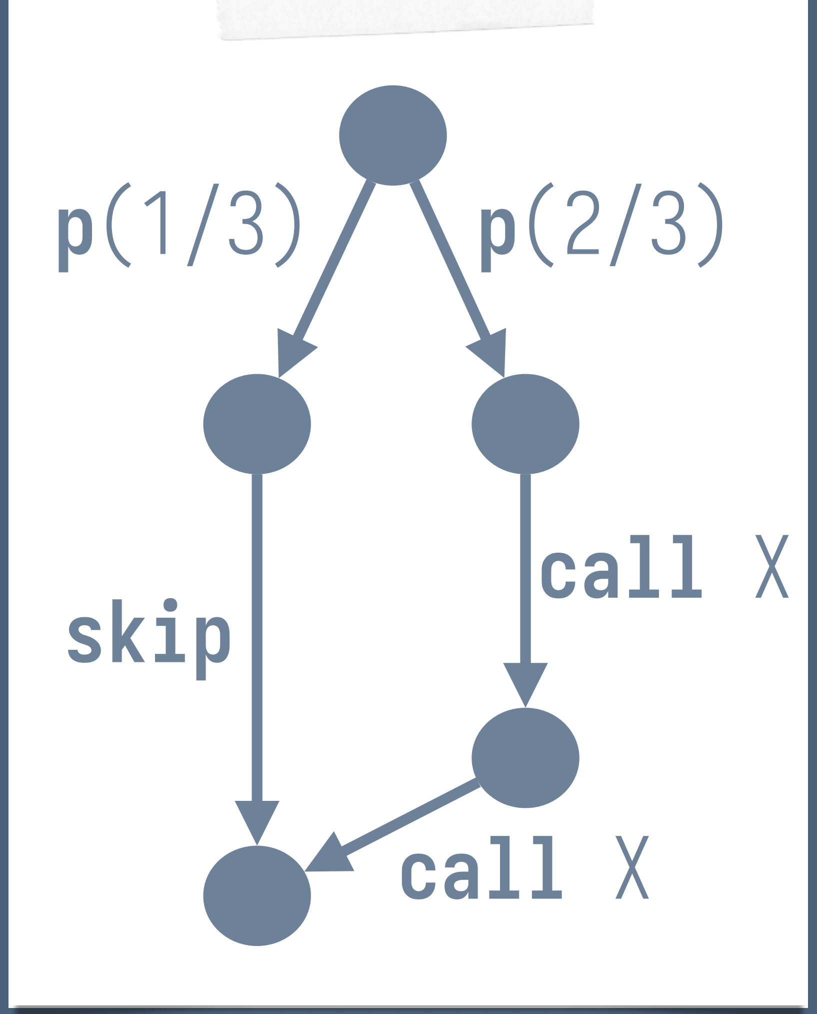
Termination-Probability Analysis

via Newton's Method for Program Analysis

Linearization at ν

$$X = (\underline{\mathsf{p}(1/3)} \otimes \underline{\mathsf{skip}}) \oplus (\underline{\mathsf{p}(2/3)} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathsf{p}(2/3)} \otimes Y \otimes \nu) \oplus (\underline{\mathsf{p}(2/3)} \otimes \nu \otimes Y)$$



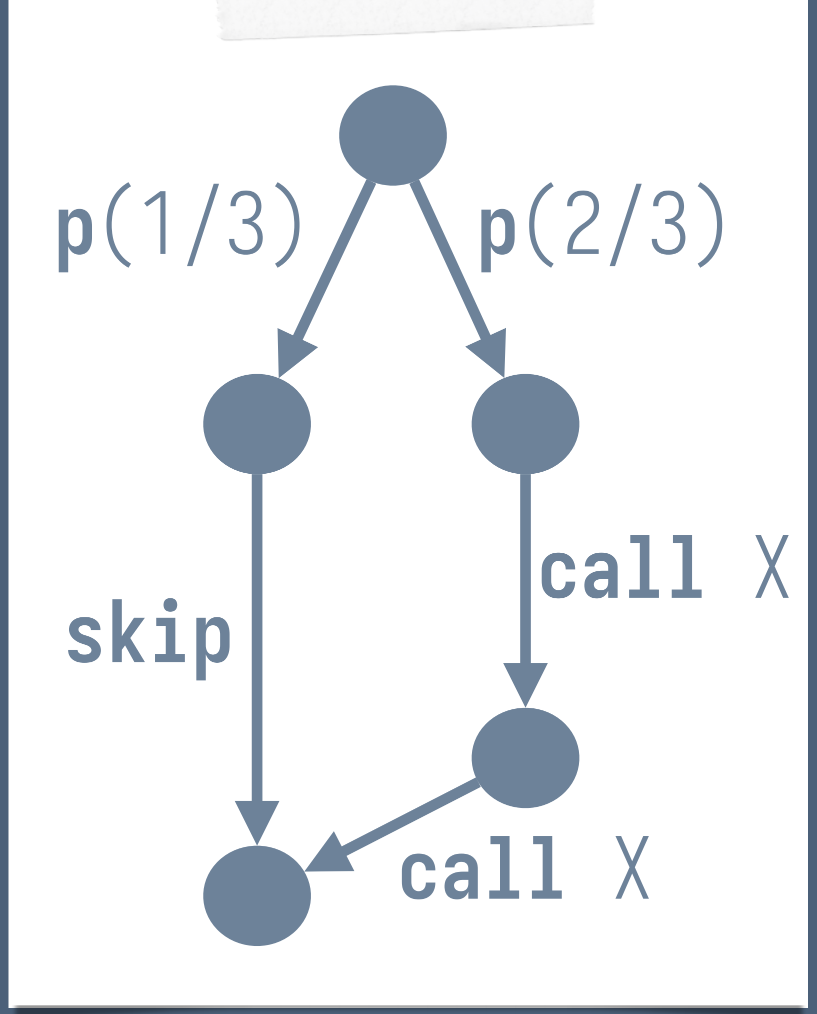
Termination-Probability Analysis

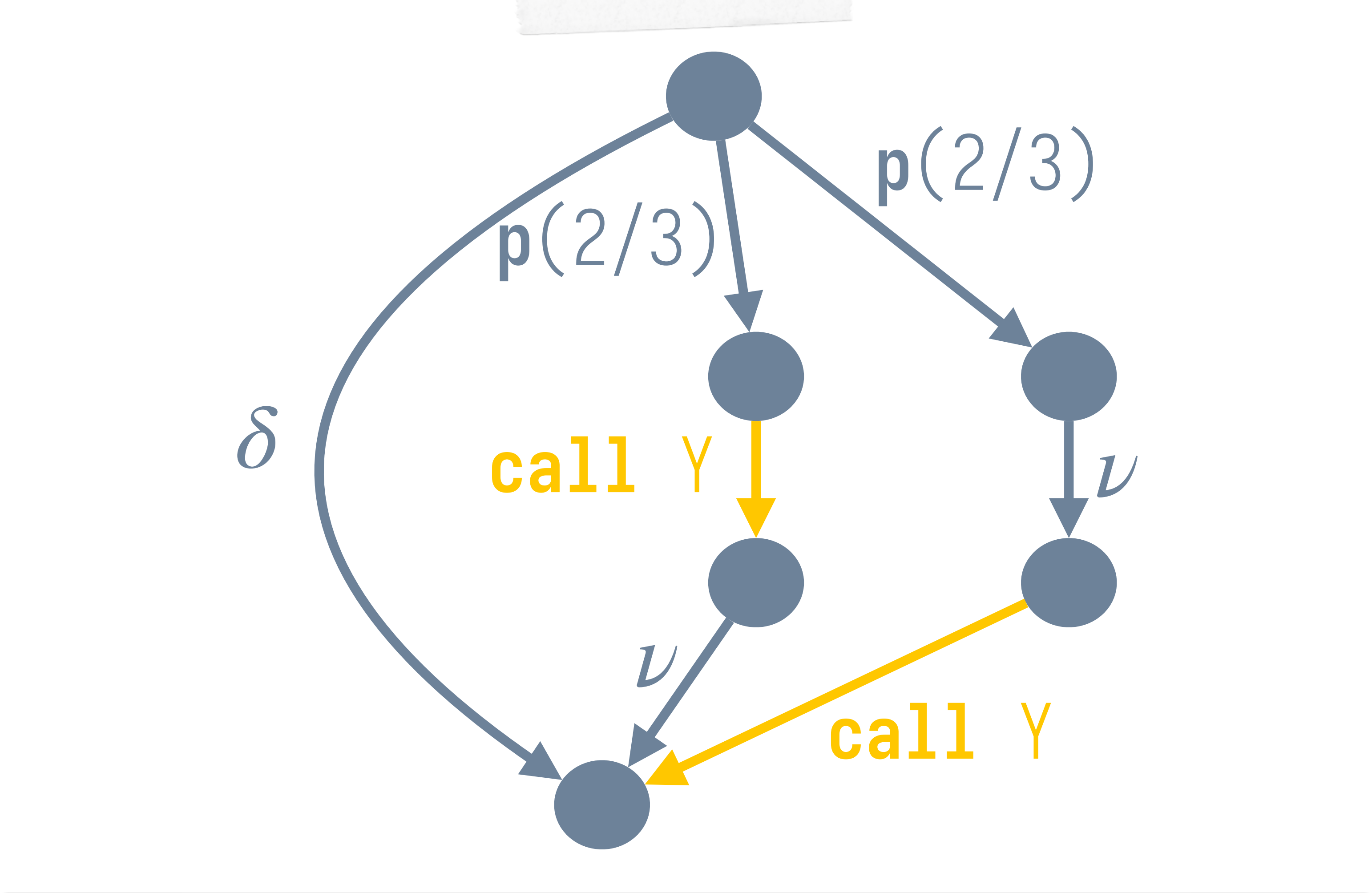
via Newton's Method for Program Analysis

Linearization at ν

$$X = (\underline{\mathrm{p}(1/3)} \otimes \underline{\mathrm{skip}}) \oplus (\underline{\mathrm{p}(2/3)} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathrm{p}(2/3)} \otimes Y \otimes \nu) \oplus (\underline{\mathrm{p}(2/3)} \otimes \nu \otimes Y)$$





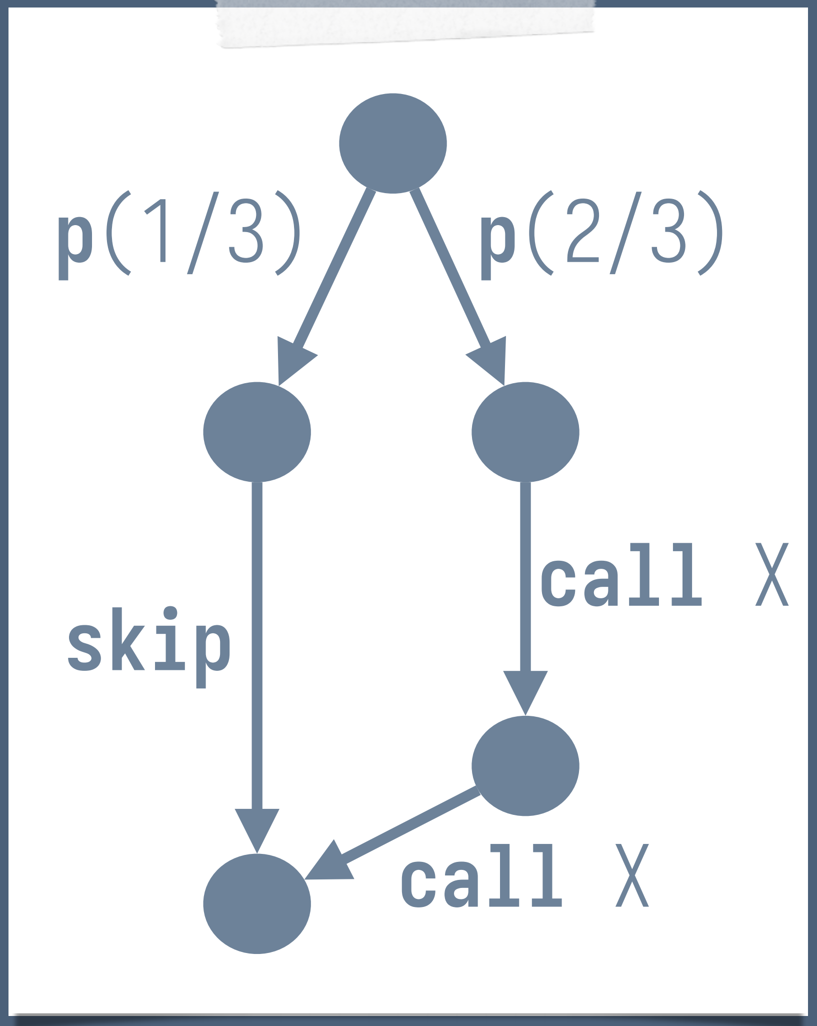
Termination-Probability Analysis

via Newton's Method for Program Analysis

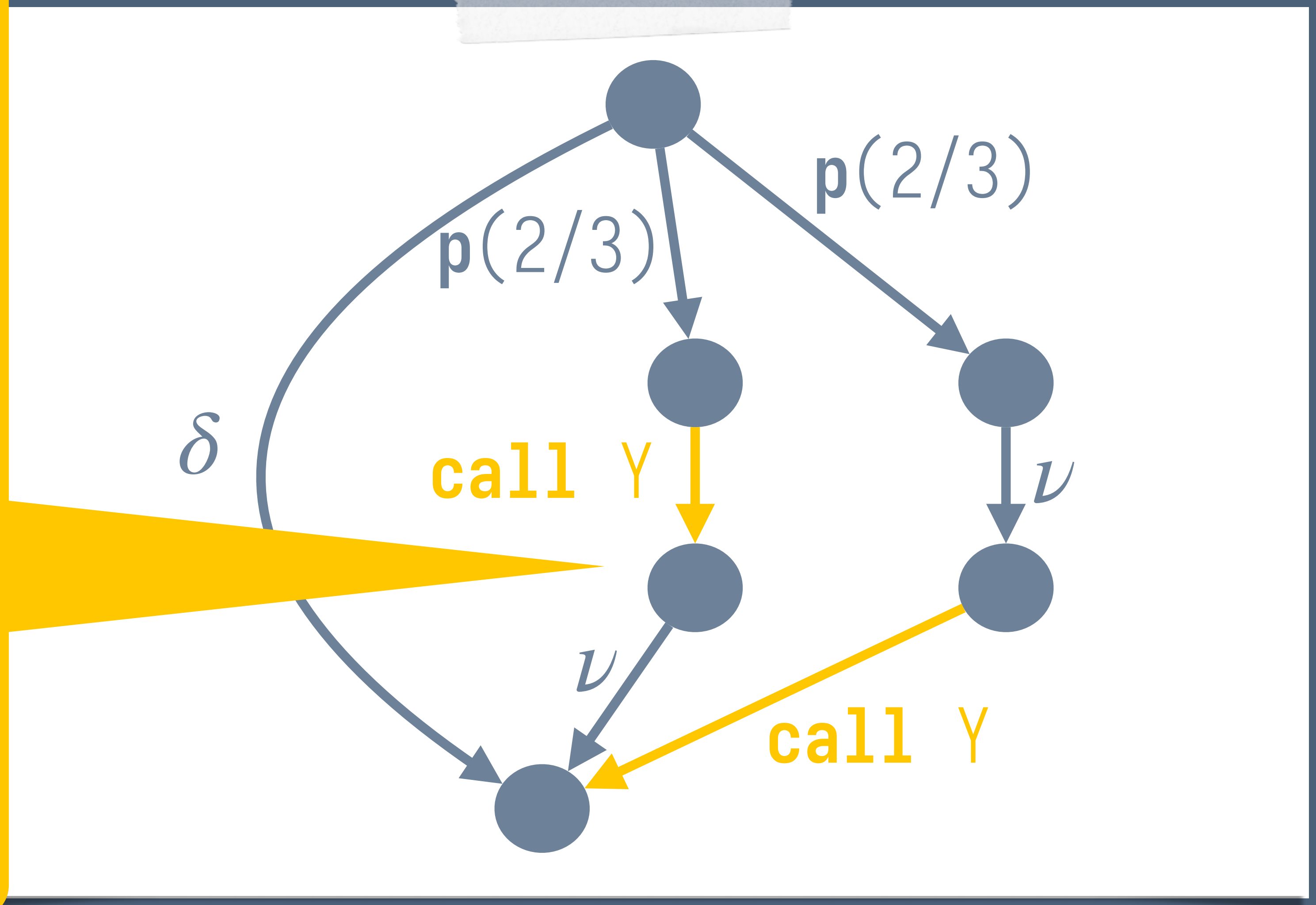
Linearization at ν

$$X = (\underline{\mathtt{p}(1/3)} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p}(2/3)} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p}(2/3)} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p}(2/3)} \otimes \nu \otimes Y)$$



- At 1st call, perform **exploration**; at 2nd call, use the summary (ν)



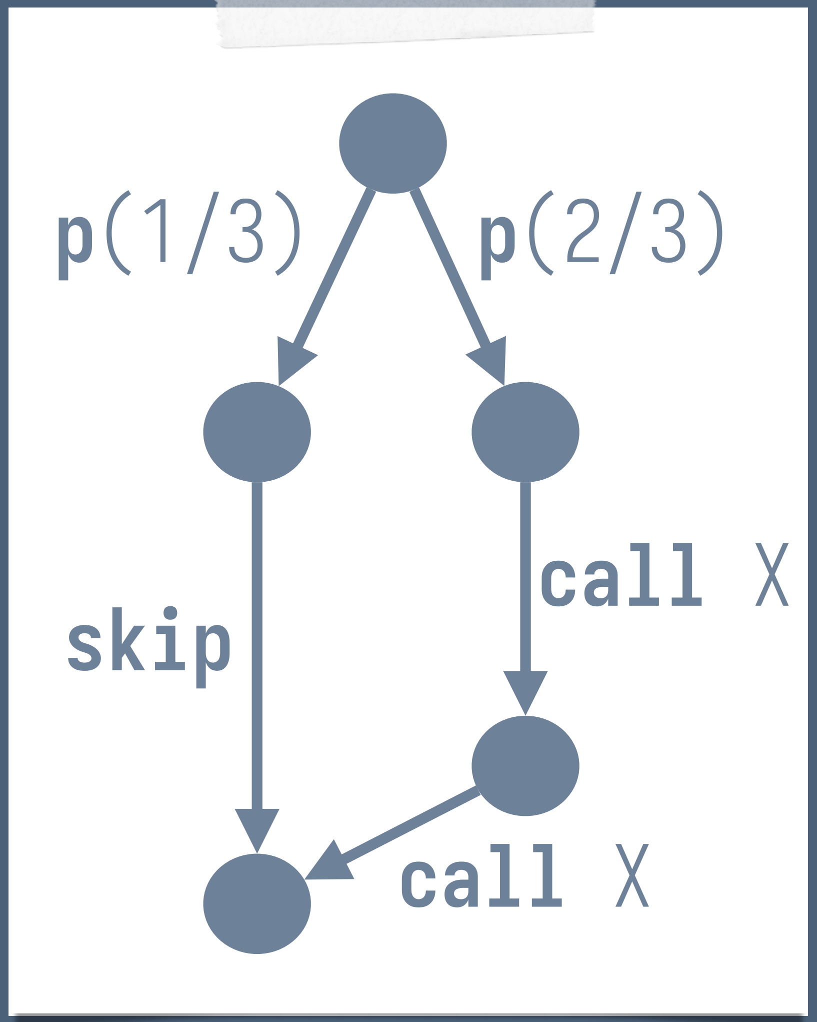
Termination-Probability Analysis

via Newton's Method for Program Analysis

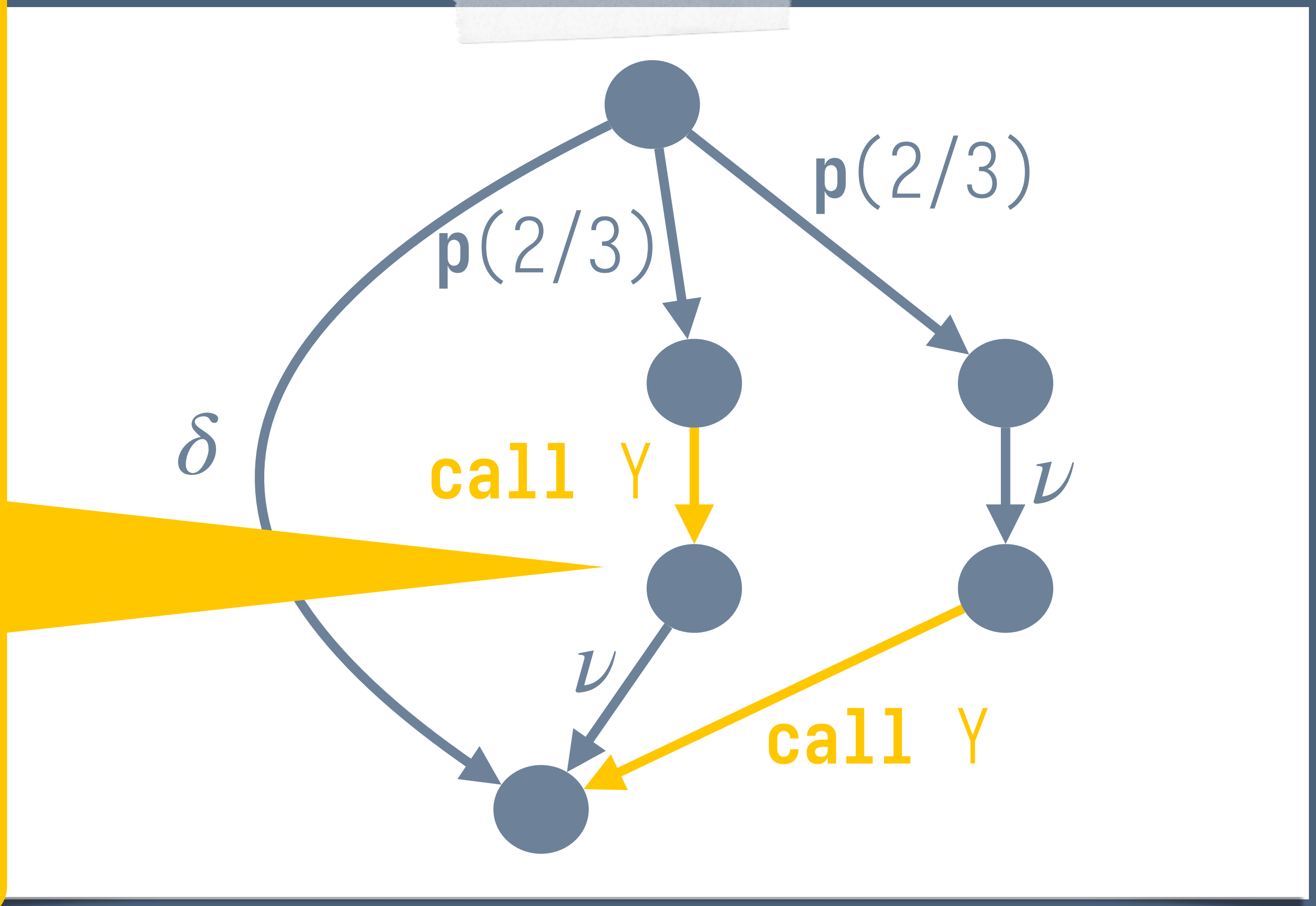
Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$



- At 1st call, perform **exploration**; at 2nd call, use the summary (ν)
- At 1st call, use ν; at 2nd call, perform **exploration**



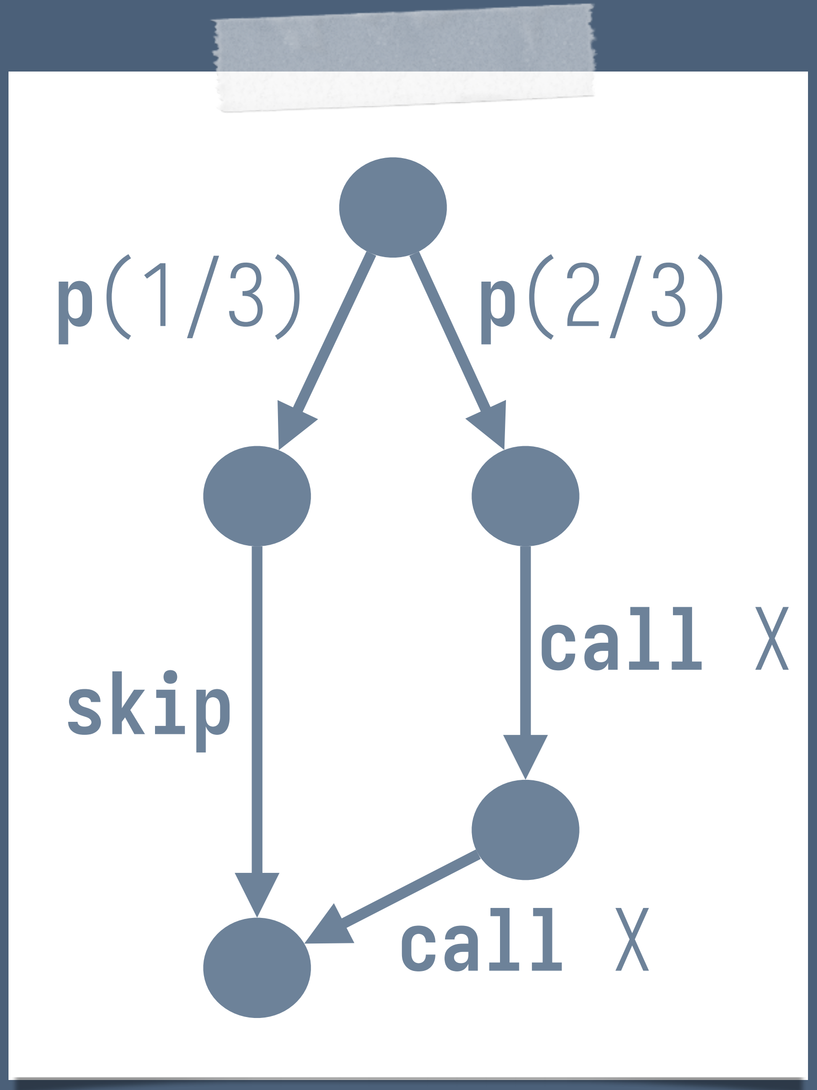
Termination-Probability Analysis

via Newton's Method for Program Analysis

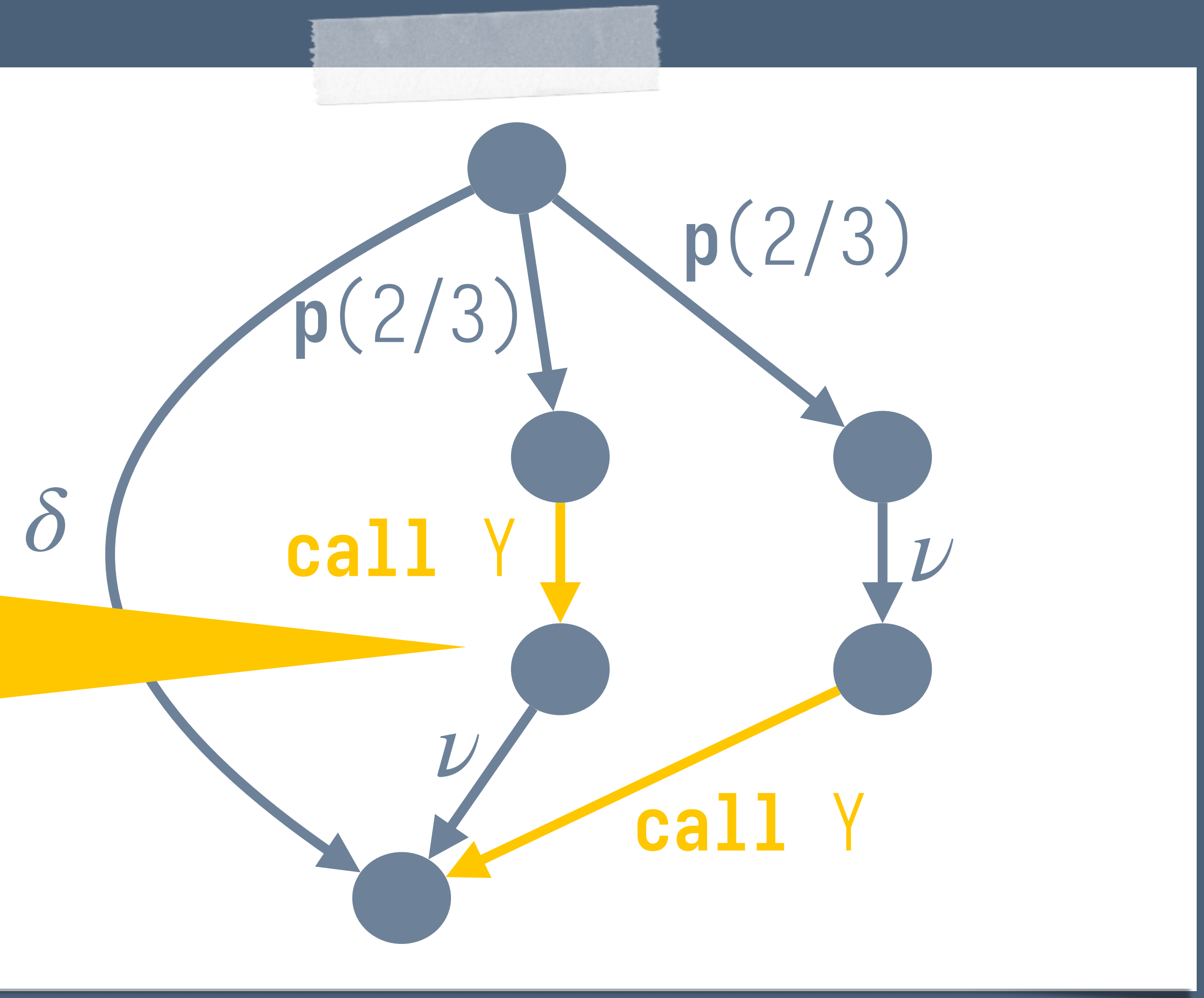
Linearization at ν

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$



- At 1st call, perform **exploration**; at 2nd call, use the summary (ν)
- At 1st call, use ν; at 2nd call, perform **exploration**
- Combine via $\oplus$



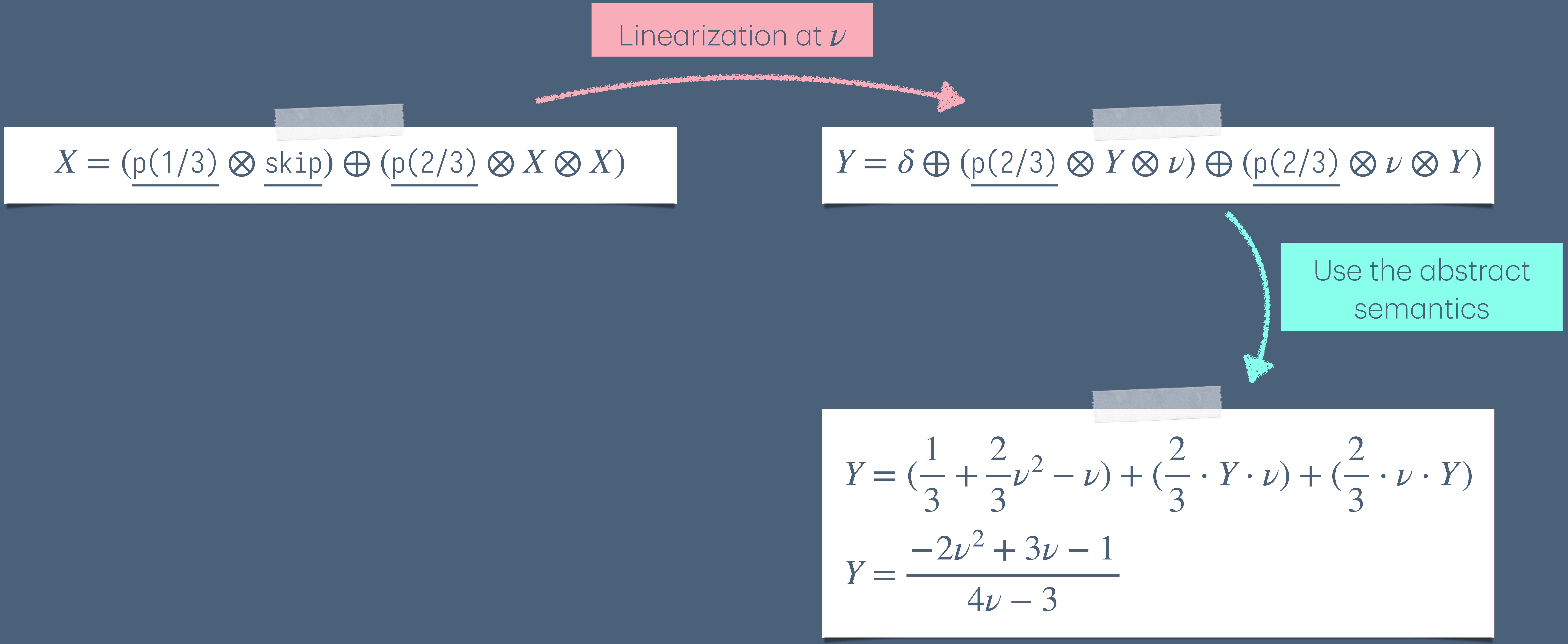
Termination-Probability Analysis

via Newton's Method for Program Analysis

$$X = (\underline{\mathtt{p(1/3)}} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p(2/3)}} \otimes X \otimes X) \xrightarrow{\text{Linearization at } \nu} Y = \delta \oplus (\underline{\mathtt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p(2/3)}} \otimes \nu \otimes Y)$$

Termination-Probability Analysis

via Newton's Method for Program Analysis



$$X = (\underline{\texttt{p(1/3)}} \otimes \underline{\texttt{skip}}) \oplus (\underline{\texttt{p(2/3)}} \otimes X \otimes X)$$

Linearization at ν

$$Y = \delta \oplus (\underline{\texttt{p(2/3)}} \otimes Y \otimes \nu) \oplus (\underline{\texttt{p(2/3)}} \otimes \nu \otimes Y)$$

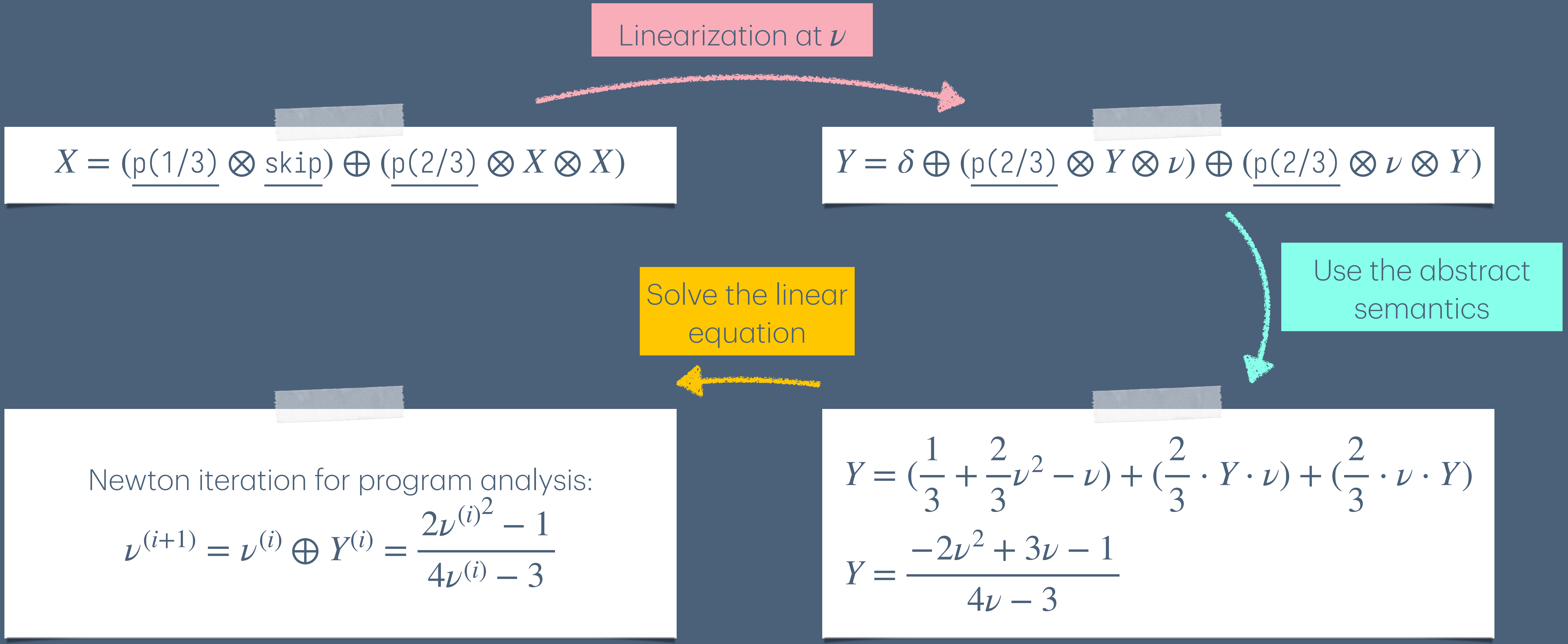
Use the abstract semantics

$$Y = (\frac{1}{3} + \frac{2}{3}\nu^2 - \nu) + (\frac{2}{3} \cdot Y \cdot \nu) + (\frac{2}{3} \cdot \nu \cdot Y)$$

$$Y = \frac{-2\nu^2 + 3\nu - 1}{4\nu - 3}$$

Termination-Probability Analysis

via Newton's Method for Program Analysis



Linearization at ν

$$X = (\underline{\mathtt{p}(1/3)} \otimes \underline{\mathtt{skip}}) \oplus (\underline{\mathtt{p}(2/3)} \otimes X \otimes X)$$

$$Y = \delta \oplus (\underline{\mathtt{p}(2/3)} \otimes Y \otimes \nu) \oplus (\underline{\mathtt{p}(2/3)} \otimes \nu \otimes Y)$$

Use the abstract semantics

$$Y = (\frac{1}{3} + \frac{2}{3}\nu^2 - \nu) + (\frac{2}{3} \cdot Y \cdot \nu) + (\frac{2}{3} \cdot \nu \cdot Y)$$

$$Y = \frac{-2\nu^2 + 3\nu - 1}{4\nu - 3}$$

Solve the linear equation

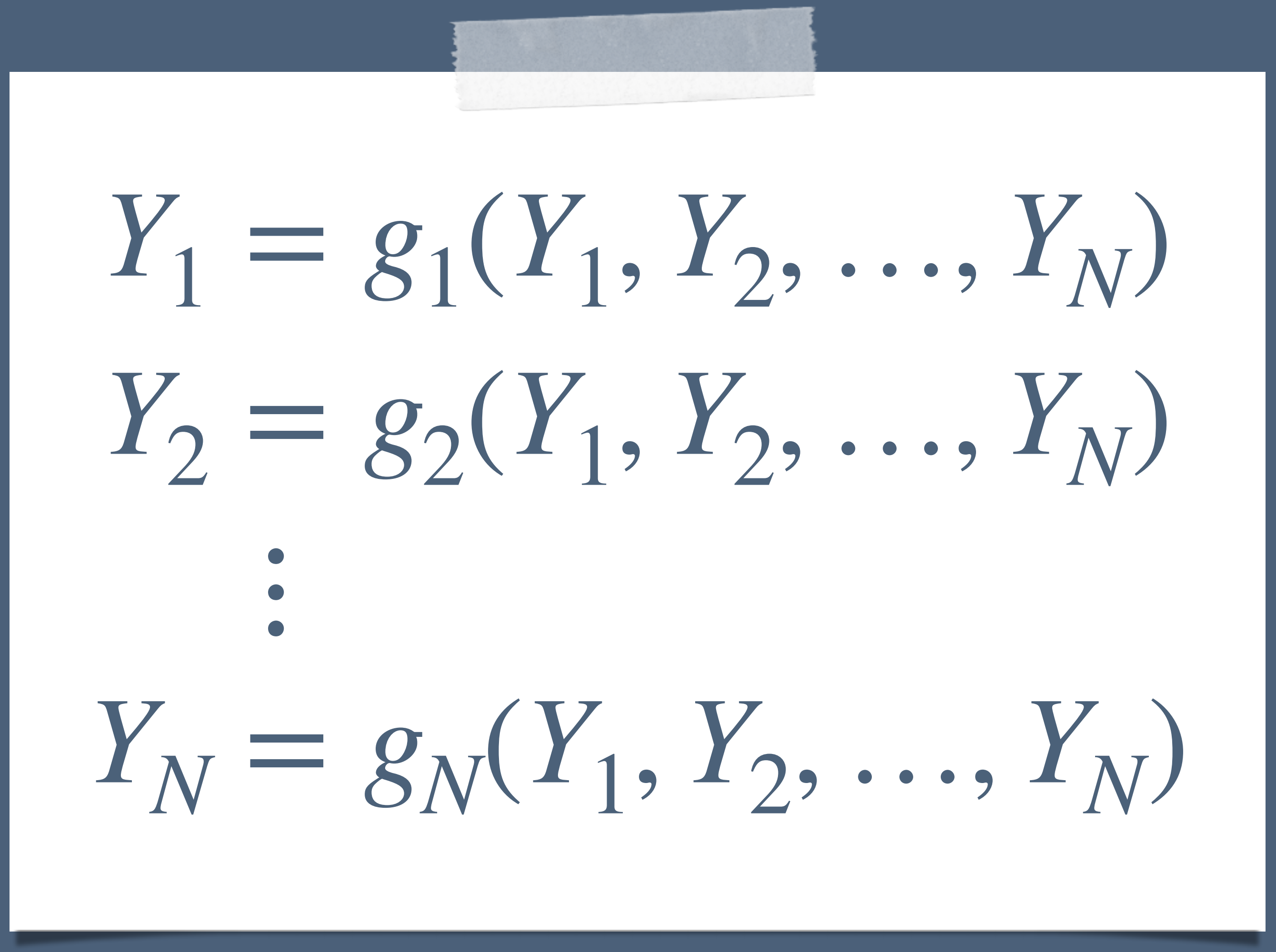
Newton iteration for program analysis:

$$\nu^{(i+1)} = \nu^{(i)} \oplus Y^{(i)} = \frac{2\nu^{(i)^2} - 1}{4\nu^{(i)} - 3}$$

So far so good?

So far so good?

- Each Newton iteration generates a system of **linear** equations:



$$\begin{aligned} Y_1 &= g_1(Y_1, Y_2, \ldots, Y_N) \\ Y_2 &= g_2(Y_1, Y_2, \ldots, Y_N) \\ &\vdots \\ Y_N &= g_N(Y_1, Y_2, \ldots, Y_N) \end{aligned}$$

So far so good?

- Each Newton iteration generates a system of **linear** equations:

$$
\begin{aligned}
Y_1 &= g_1(Y_1, Y_2, \ldots, Y_N) \\
Y_2 &= g_2(Y_1, Y_2, \ldots, Y_N) \\
&\vdots \\
Y_N &= g_N(Y_1, Y_2, \ldots, Y_N)
\end{aligned}
$$

Each g has the form:

$$a \oplus (b_1 \otimes Y_{i_1} \otimes c_1) \oplus (b_2 \otimes Y_{i_2} \otimes c_2) \oplus \ldots \oplus (b_k \otimes Y_{i_k} \otimes c_k)$$

So far so good?

- Each Newton iteration generates a system of **linear** equations:

$$
\begin{aligned}
Y_1 &= g_1(Y_1, Y_2, \ldots, Y_N) \\
Y_2 &= g_2(Y_1, Y_2, \ldots, Y_N) \\
&\vdots \\
Y_N &= g_N(Y_1, Y_2, \ldots, Y_N)
\end{aligned}
$$

Each g has the form:

$$a \oplus (b_1 \otimes Y_{i_1} \otimes c_1) \oplus (b_2 \otimes Y_{i_2} \otimes c_2) \oplus \ldots \oplus (b_k \otimes Y_{i_k} \otimes c_k)$$

- However, Newton's method is efficient **only if one can solve linear equations efficiently**

So far so good?

- Each Newton iteration generates a system of **linear** equations:

$$\begin{aligned} Y_1 &= g_1(Y_1, Y_2, \ldots, Y_N) \\ Y_2 &= g_2(Y_1, Y_2, \ldots, Y_N) \\ &\vdots \\ Y_N &= g_N(Y_1, Y_2, \ldots, Y_N) \end{aligned}$$

Each g has the form:

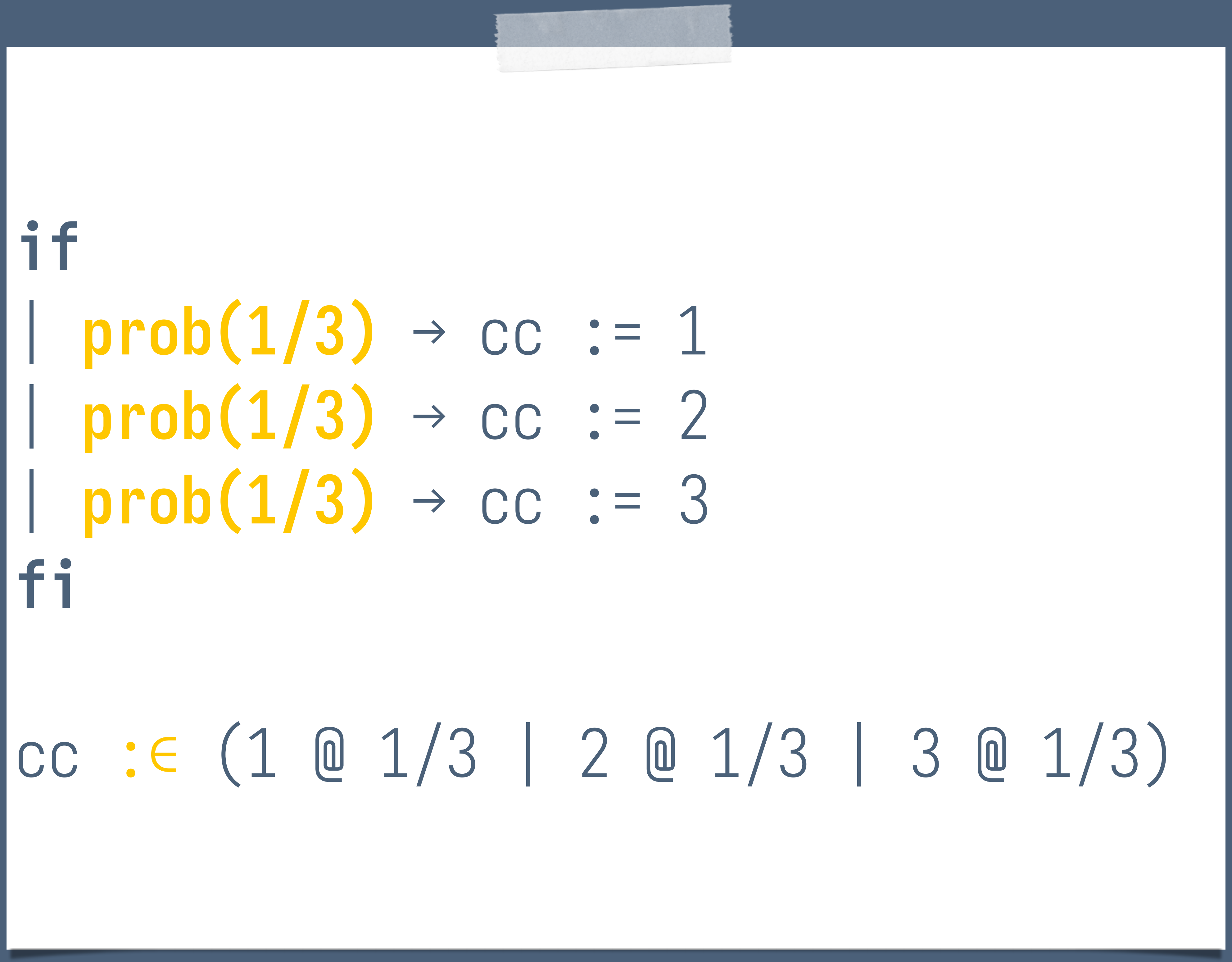
$$a \oplus (b_1 \otimes Y_{i_1} \otimes c_1) \oplus (b_2 \otimes Y_{i_2} \otimes c_2) \oplus \ldots \oplus (b_k \otimes Y_{i_k} \otimes c_k)$$

- However, Newton's method is efficient **only if one can solve linear equations efficiently**
  - [Reps, Turetsky, and Prabhu 2016] proposed a general solution that uses tensor products

Probabilistic Programs

Probabilistic Programs

- We have already seen probabilistic branching



```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi

cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

Probabilistic Programs

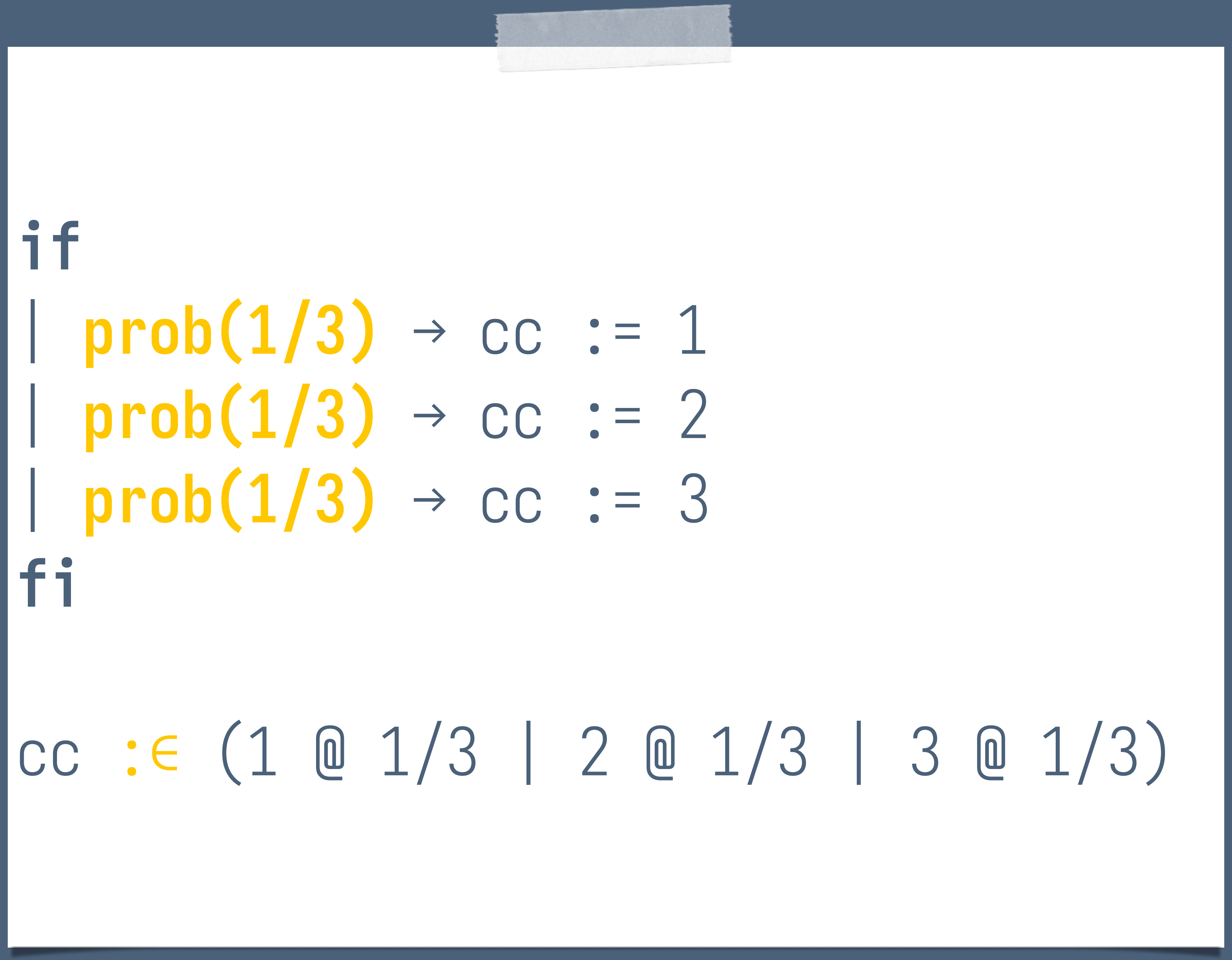
- We have already seen probabilistic branching
- True randomness

```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi

cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

Probabilistic Programs

- We have already seen probabilistic branching
- True randomness
- A distribution of execution paths



```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi

cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

Probabilistic Programs

- We have already seen probabilistic branching
- True randomness
- A distribution of execution paths
- **Probabilistic nondeterminism**

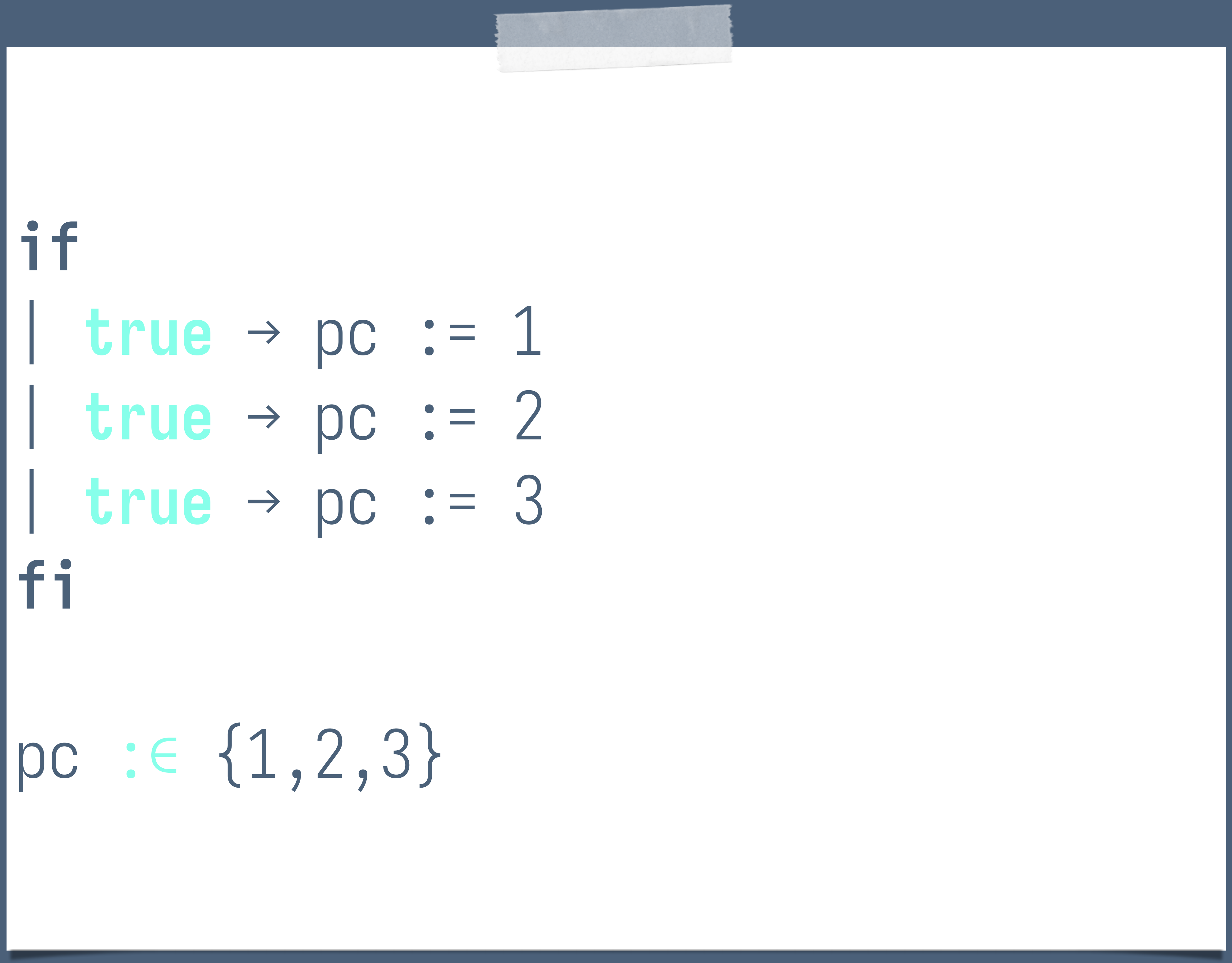
```
if
| prob(1/3) → cc := 1
| prob(1/3) → cc := 2
| prob(1/3) → cc := 3
fi

cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

Probabilistic Programs

Probabilistic Programs

- There are also other kinds of branching

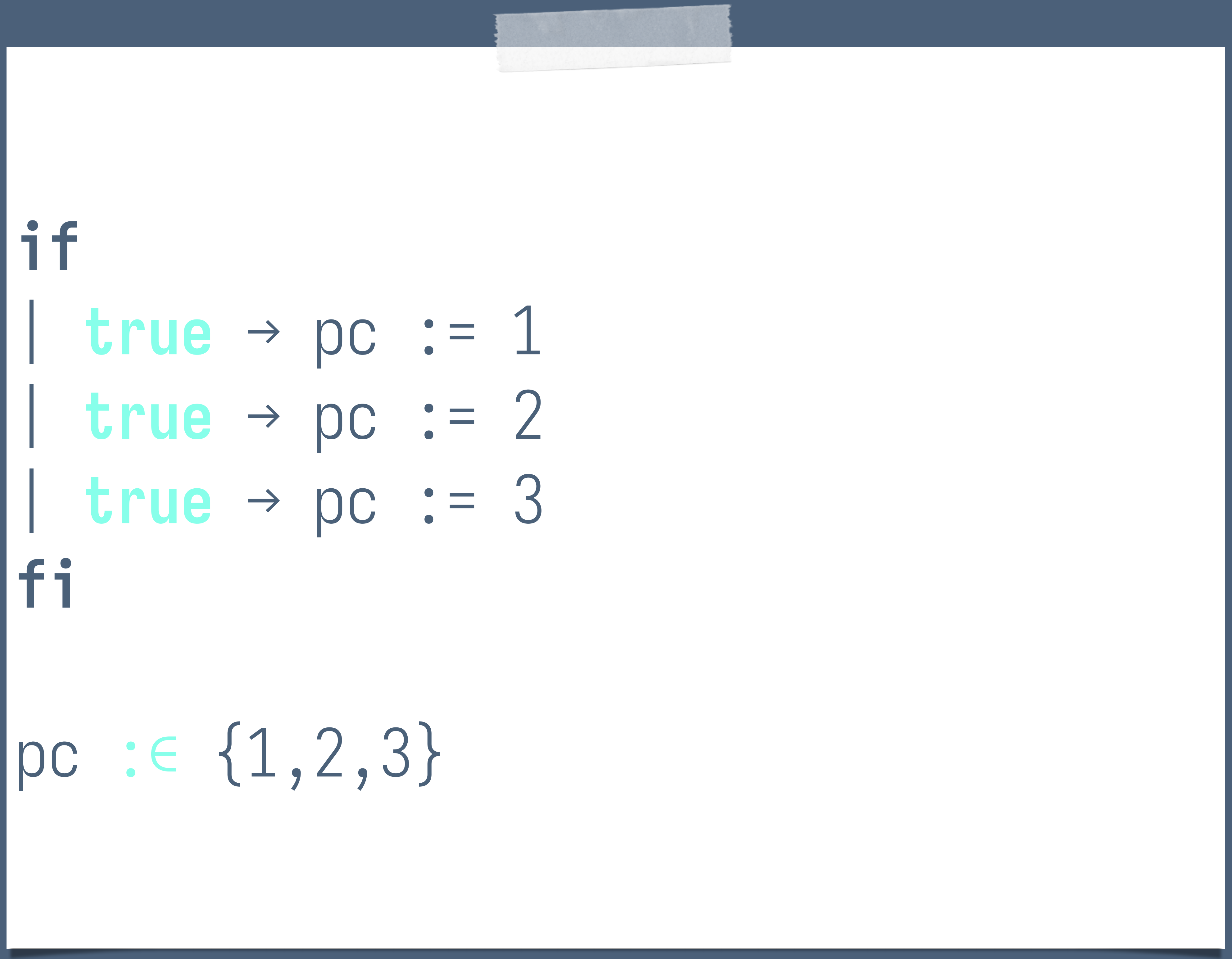


```
if
| true → pc := 1
| true → pc := 2
| true → pc := 3
fi

pc :∈ {1,2,3}
```

Probabilistic Programs

- There are also other kinds of branching
- Dijkstra's **Guarded Command Language** (GCL)

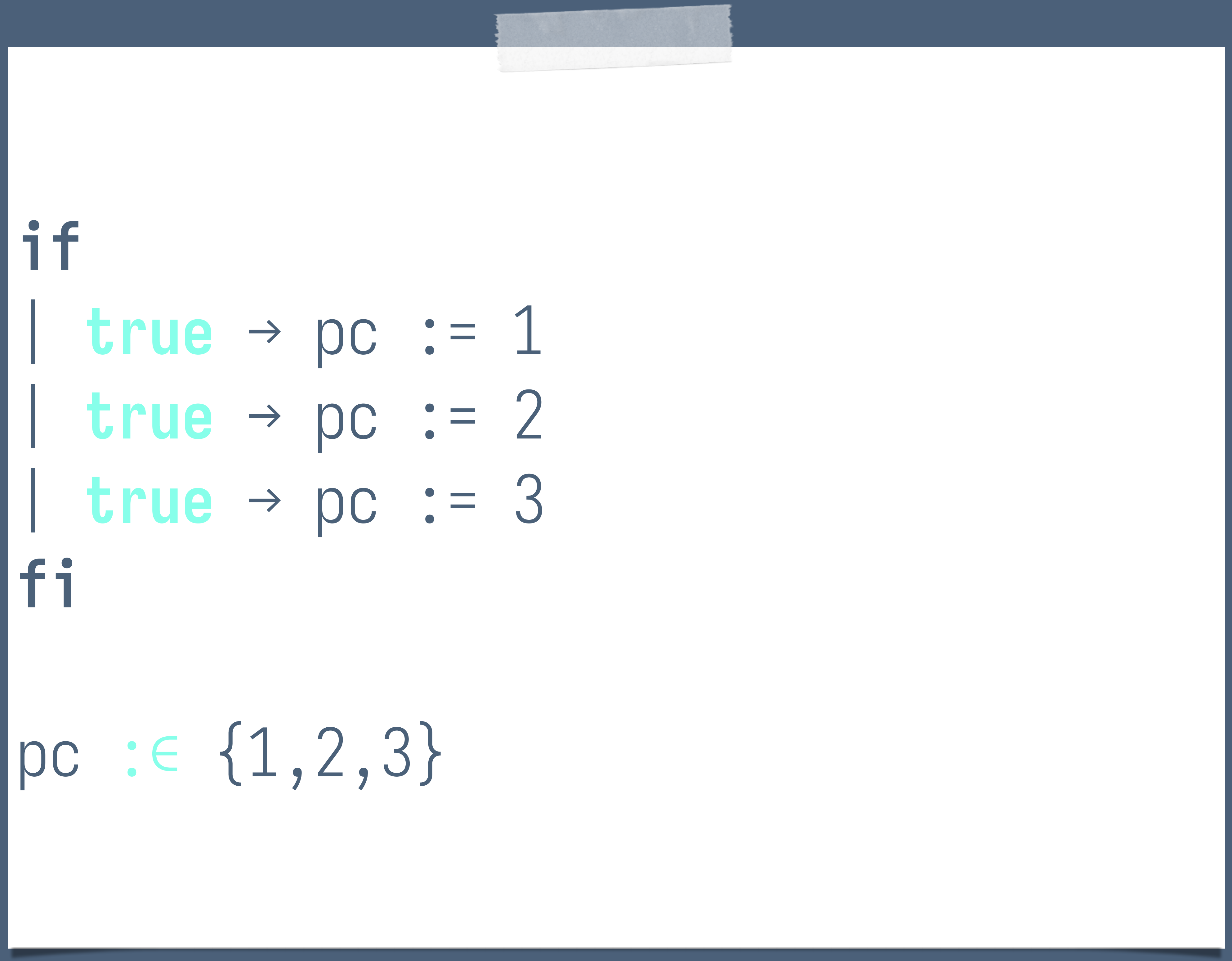


```
if
| true → pc := 1
| true → pc := 2
| true → pc := 3
fi

pc :∈ {1,2,3}
```

Probabilistic Programs

- There are also other kinds of branching
- Dijkstra's **Guarded Command Language** (GCL)
- A set of execution paths



```
if
| true → pc := 1
| true → pc := 2
| true → pc := 3
fi

pc :∈ {1,2,3}
```

Probabilistic Programs

- There are also other kinds of branching
- Dijkstra's **Guarded Command Language** (GCL)
- A set of execution paths
- **Demonic nondeterminism**

```
if
| true → pc := 1
| true → pc := 2
| true → pc := 3
fi

pc :∈ {1,2,3}
```

The Monty-Hall Puzzle

as a probabilistic program

The Monty-Hall Puzzle

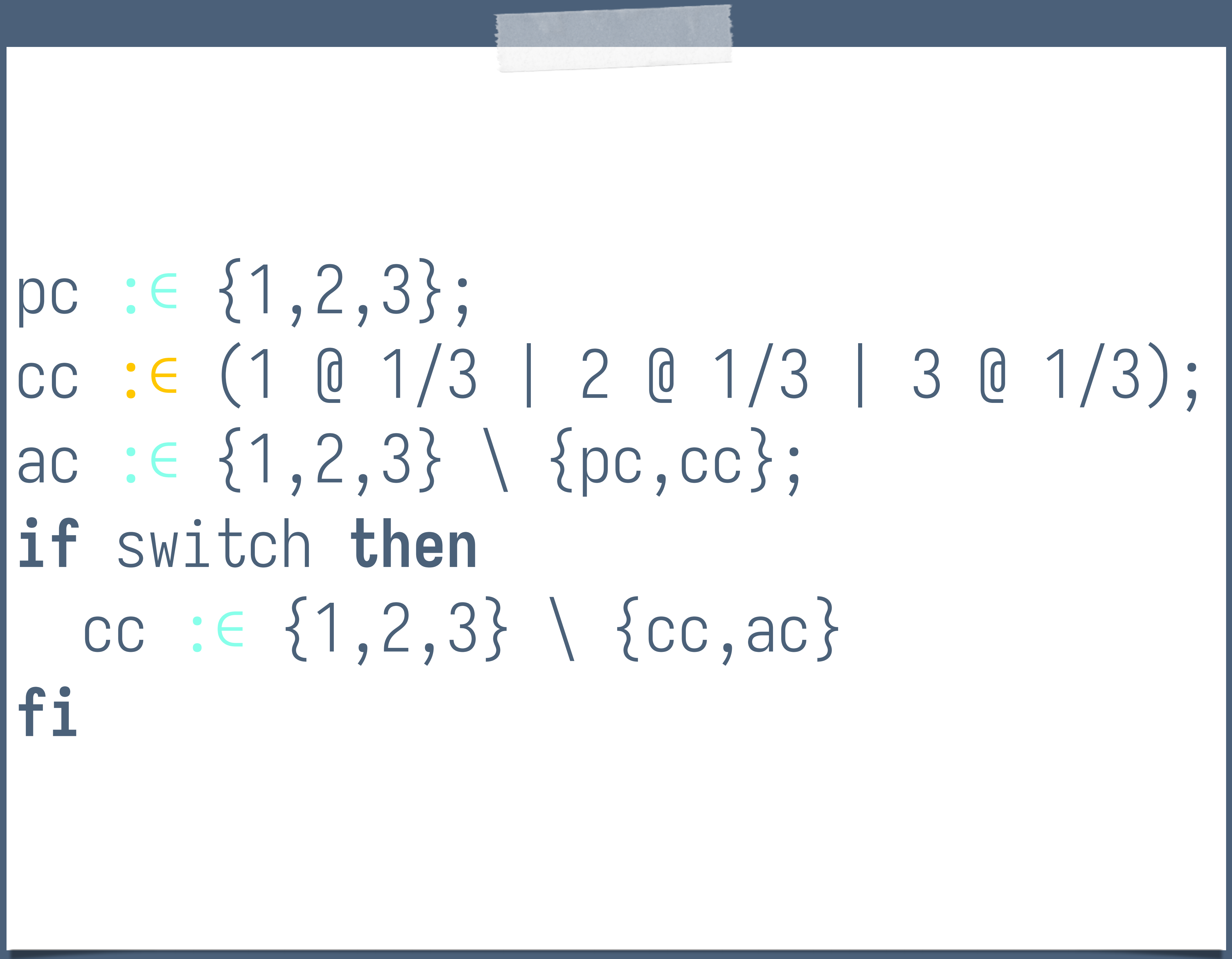
as a probabilistic program

- Programs can use multiple kinds of branching

The Monty-Hall Puzzle

as a probabilistic program

- Programs can use multiple kinds of branching
- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)

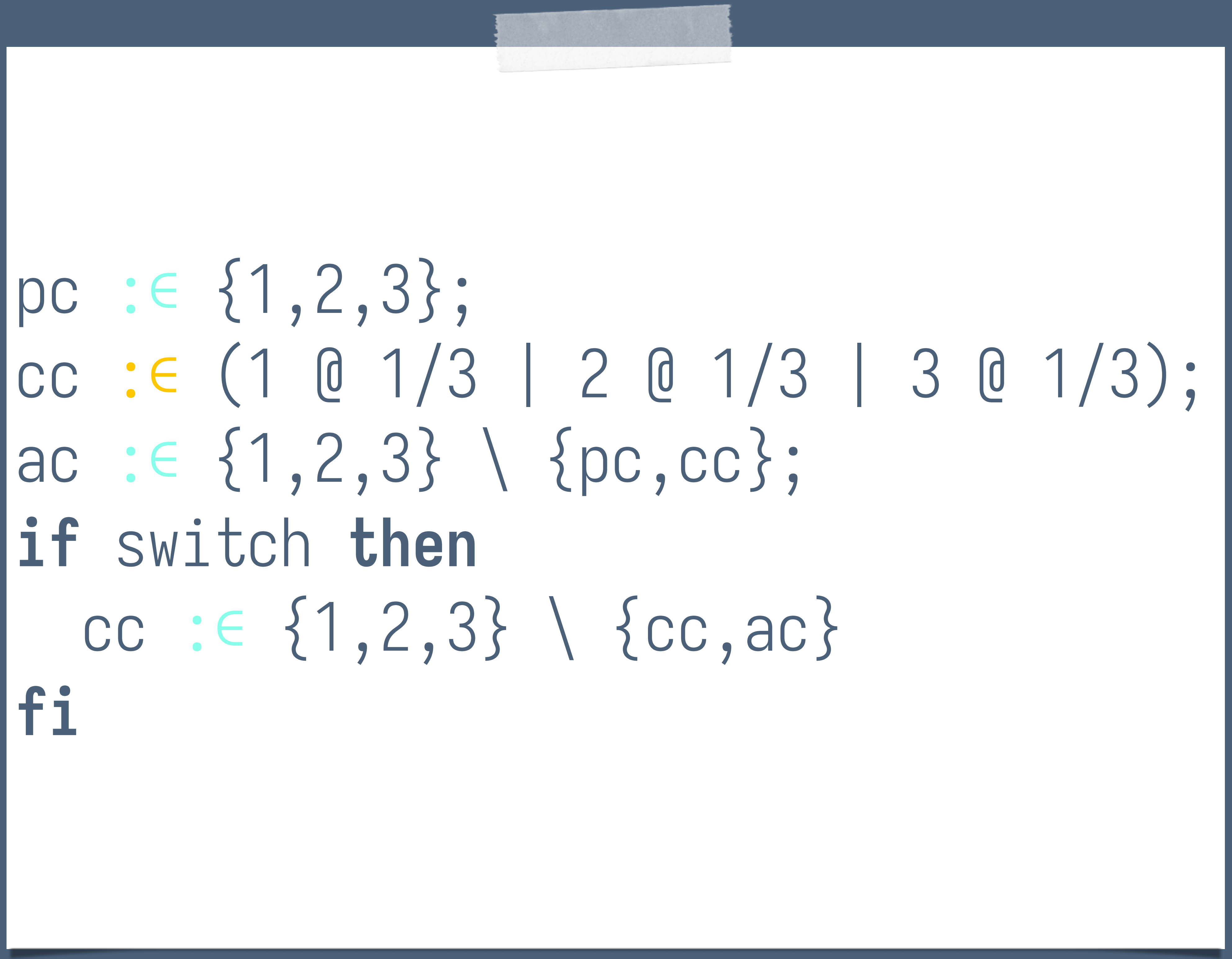


```
pc :∈ {1,2,3};
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
ac :∈ {1,2,3} \ {pc,cc};
if switch then
  cc :∈ {1,2,3} \ {cc,ac}
fi
```

The Monty-Hall Puzzle

as a probabilistic program

- Programs can use multiple kinds of branching
- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)
- Combine three kinds of branching:
  - Probabilistic
  - Demonic
  - Conditional



```
pc :∈ {1,2,3};
cc :∈ (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
ac :∈ {1,2,3} \ {pc,cc};
if switch then
  cc :∈ {1,2,3} \ {cc,ac}
fi
```

Termination-Probability Analysis

of Boolean programs

Termination-Probability Analysis

of Boolean programs

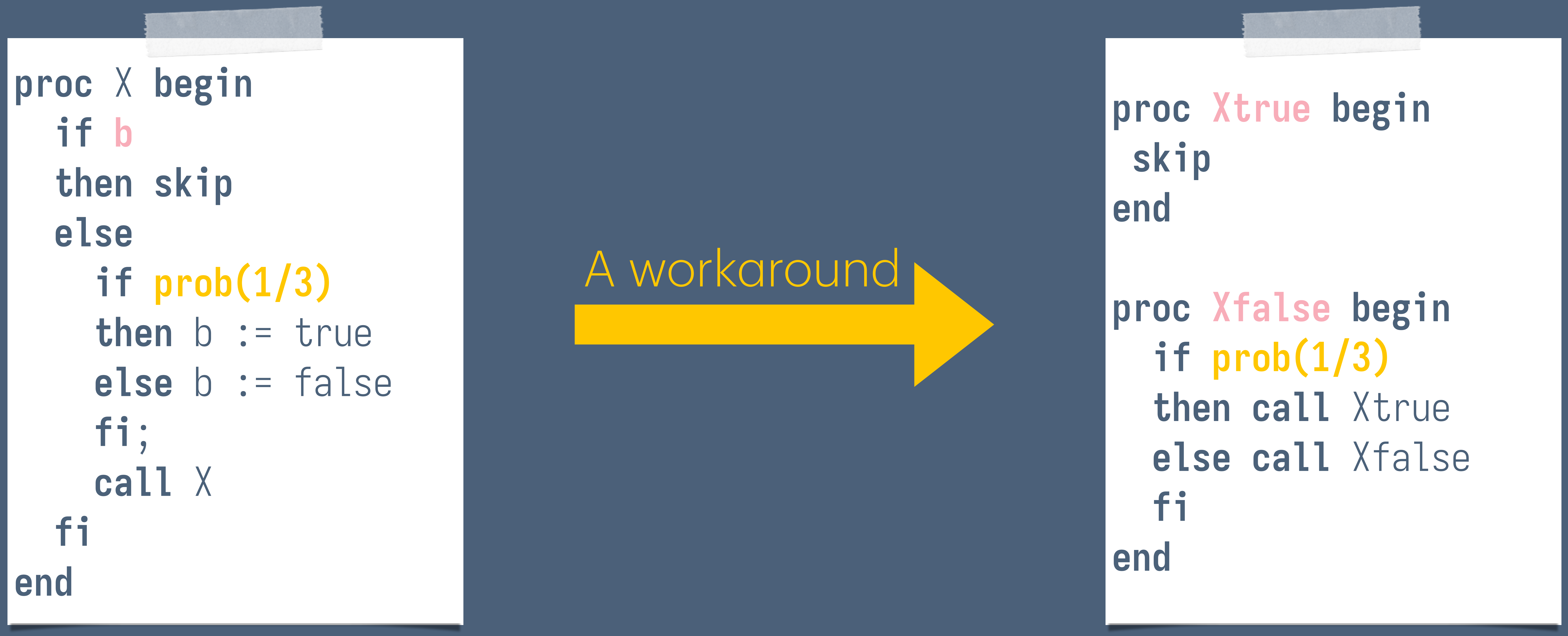
- Problem: A semiring has **only one** combine (⊕) operation

```
proc X begin
  if b
  then skip
  else
    if prob(1/3)
    then b := true
    else b := false
    fi;
    call X
  fi
end
```

Termination-Probability Analysis

of Boolean programs

- Problem: A semiring has **only one** combine (⊕) operation



```
proc X begin
  if b
  then skip
  else
    if prob(1/3)
    then b := true
    else b := false
    fi;
    call X
  fi
end
```

A workaround

```
proc Xtrue begin
 skip
end

proc Xfalse begin
  if prob(1/3)
  then call Xtrue
  else call Xfalse
  fi
end
```

Termination-Probability Analysis

of Boolean programs

- Problem: A semiring has **only one** combine (⊕) operation

```
proc X begin
  if b
  then skip
  else
    if prob(1/3)
    then b := true
    else b := false
    fi;
    call X
  fi
end
```

A workaround

```
proc Xtrue begin
 skip
end

proc Xfalse begin
  if prob(1/3)
  then call Xtrue
  else call Xfalse
  fi
end
```

- Introduce extra procedures to encode different states

Termination-Probability Analysis

of Boolean programs

- Problem: A semiring has **only one** combine (⊕) operation

```
proc X begin
  if b
  then skip
  else
    if prob(1/3)
    then b := true
    else b := false
    fi;
    call X
  fi
end
```

A workaround

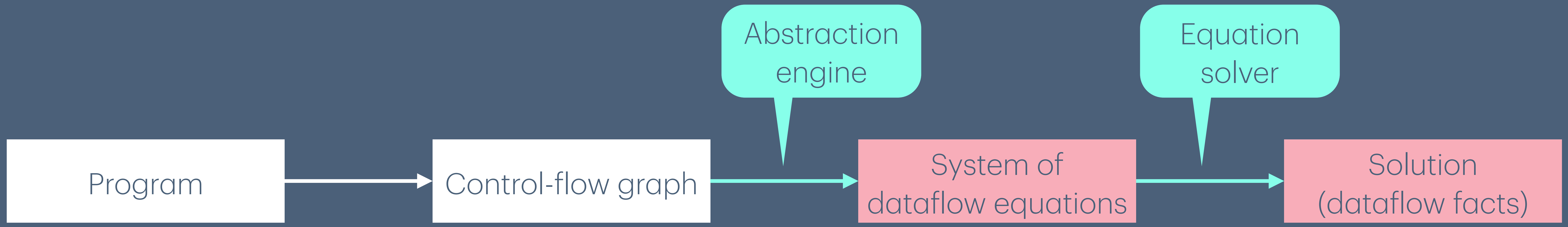
```
proc Xtrue begin
 skip
end

proc Xfalse begin
  if prob(1/3)
  then call Xtrue
  else call Xfalse
  fi
end
```

- Introduce extra procedures to encode different states
- Cannot handle **infinite state spaces**

Towards Multiple Combine Operations

Towards Multiple Combine Operations


Program

Control-flow graph

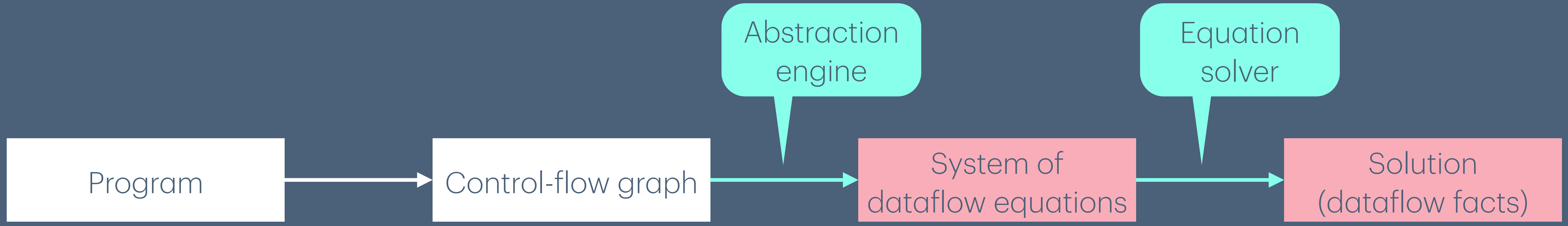
Abstraction engine

System of dataflow equations

Equation solver

Solution (dataflow facts)

Towards Multiple Combine Operations



Abstraction engine

Equation solver

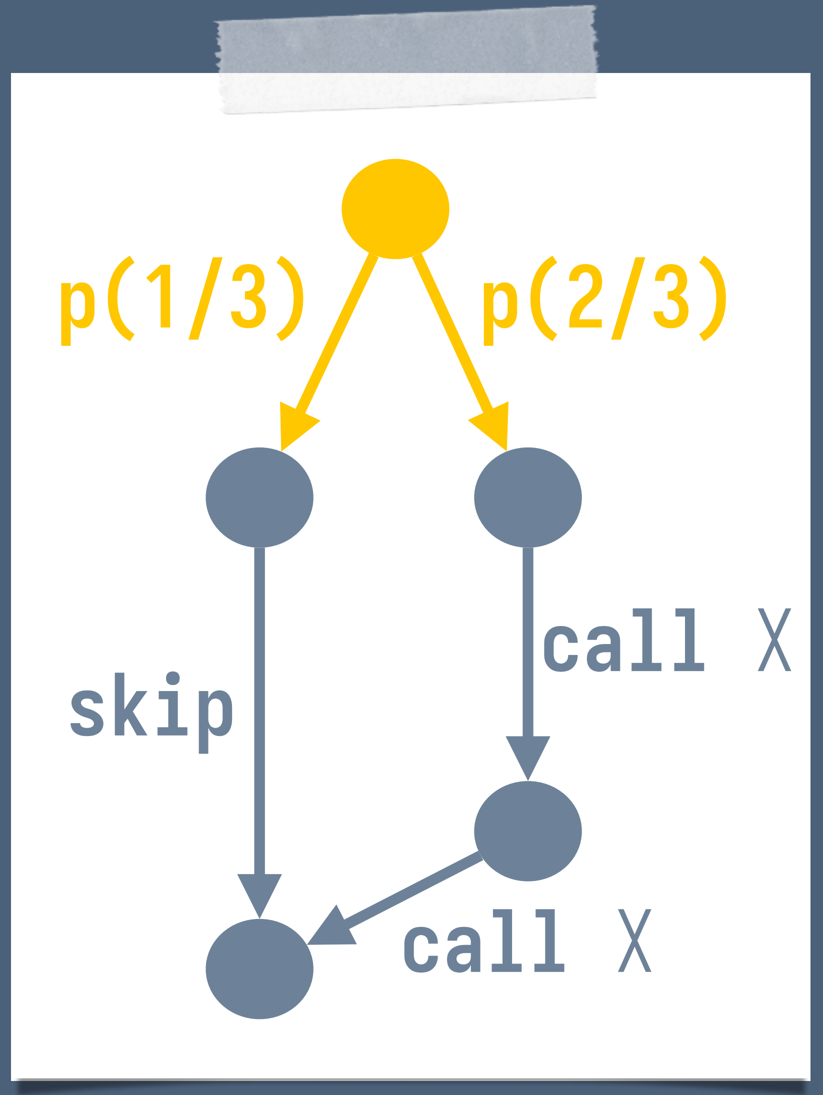
Program

Control-flow graph

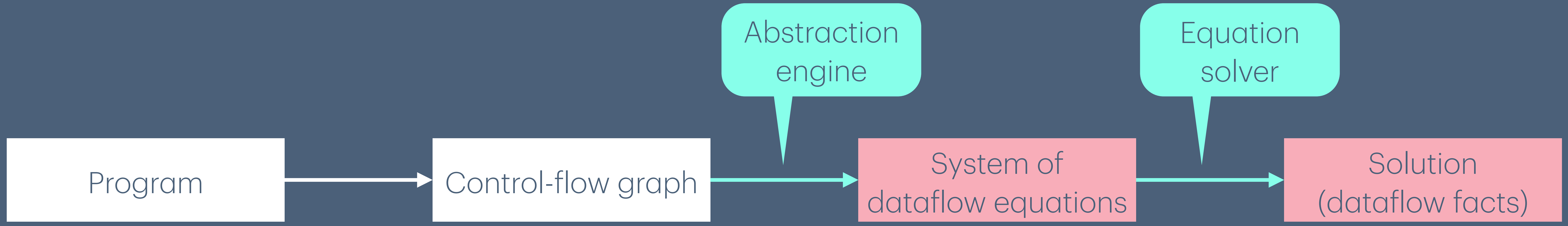
System of dataflow equations

Solution (dataflow facts)

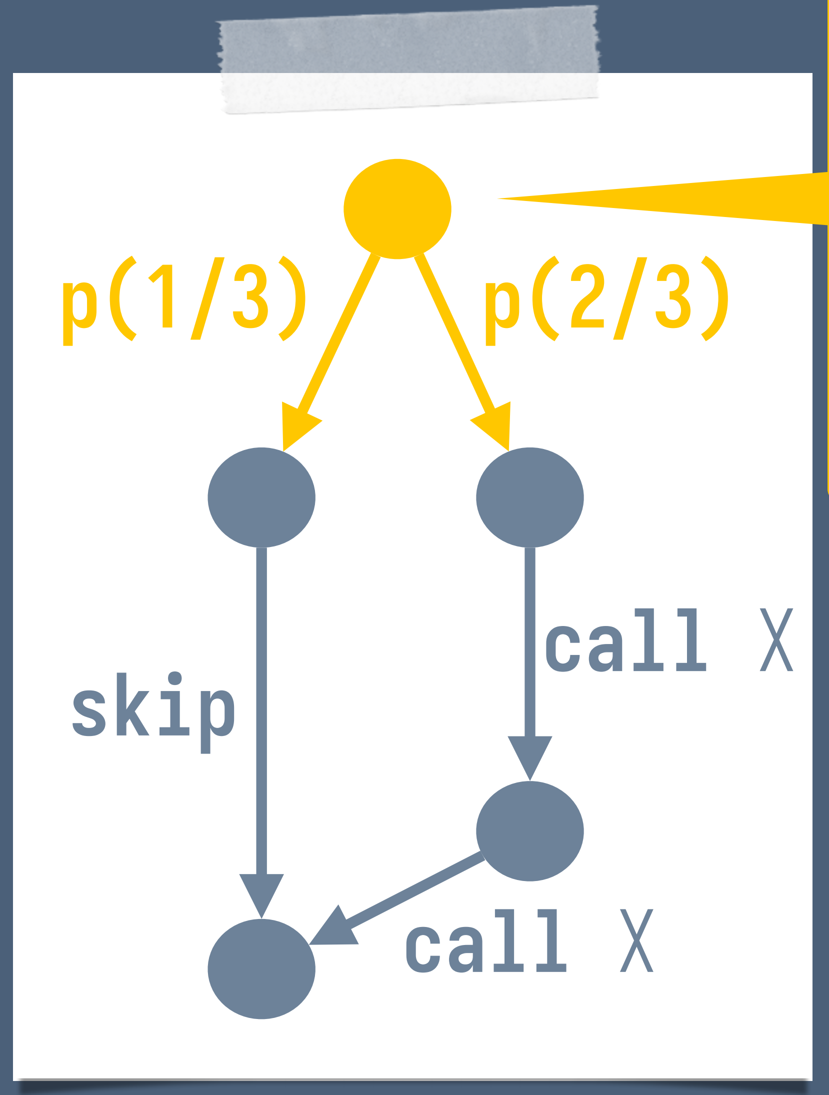
```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



Towards Multiple Combine Operations

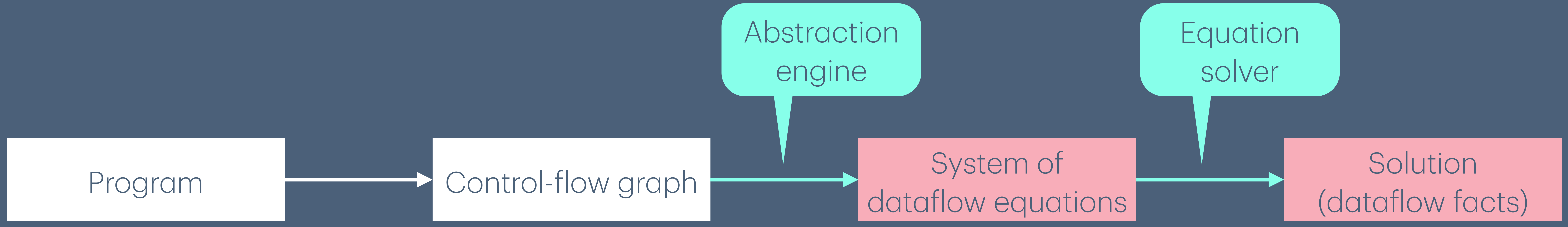


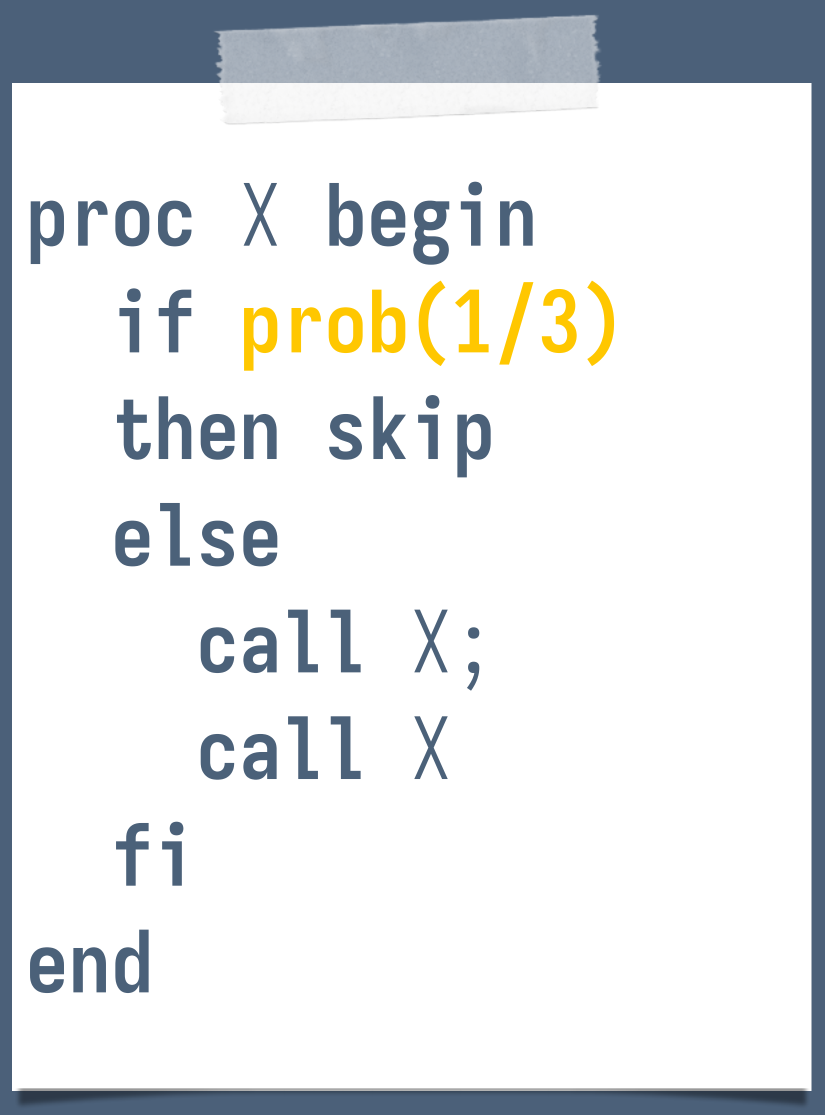
```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



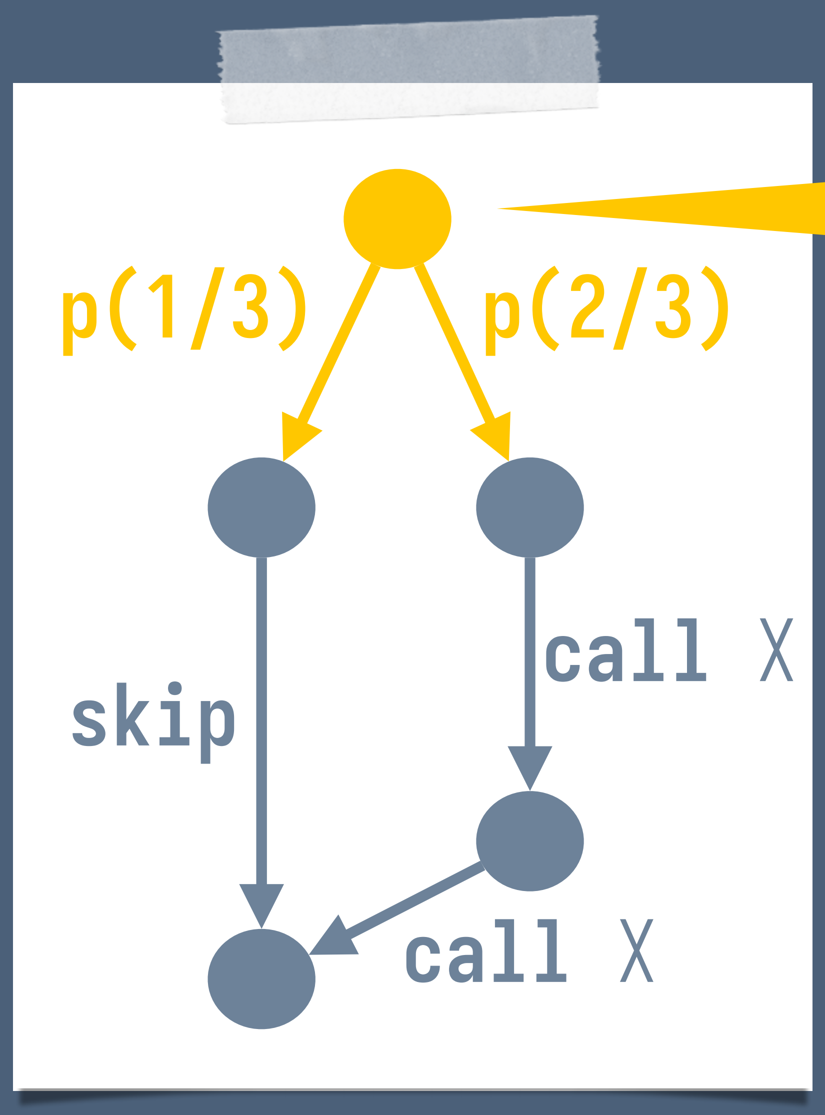
- Confluence is interpreted by $\oplus$, implicitly

Towards Multiple Combine Operations





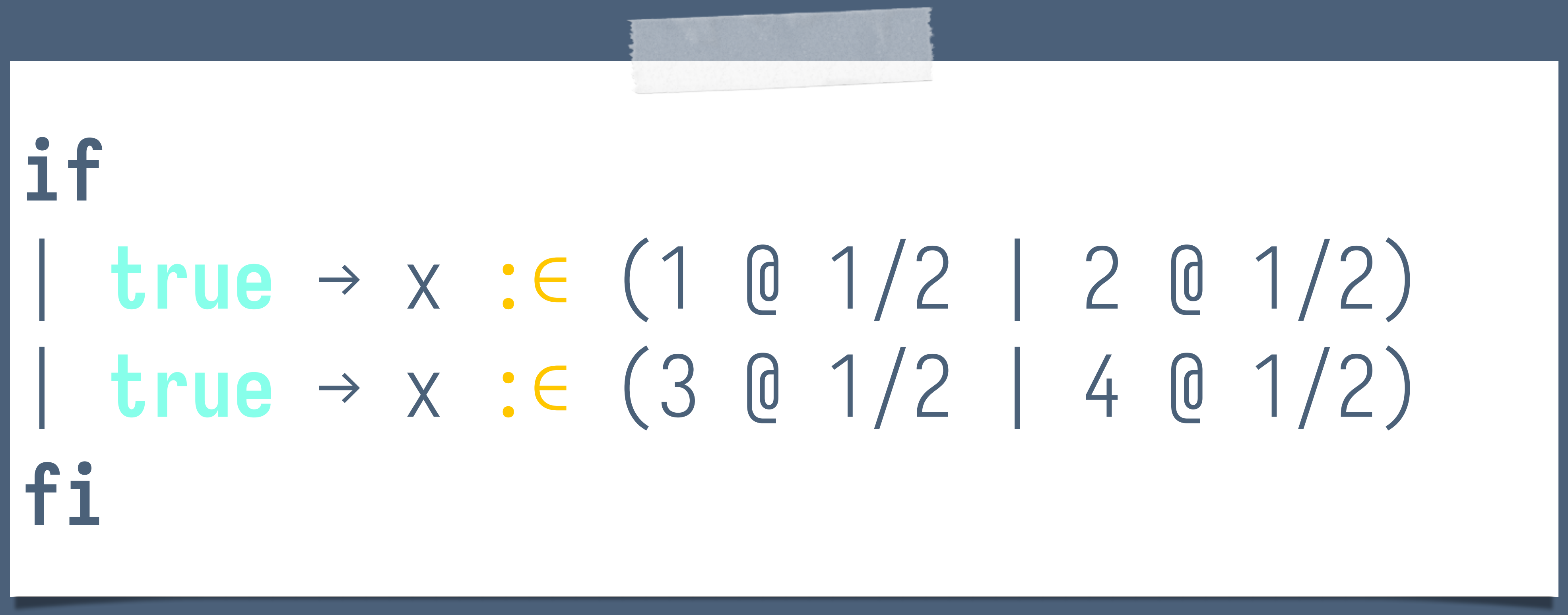
```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



- Confluence is interpreted by ⊕, implicitly
- To support **multiple combine operations**, we need to first distinguish different confluences in the graph

Control-flow Hyper-graph

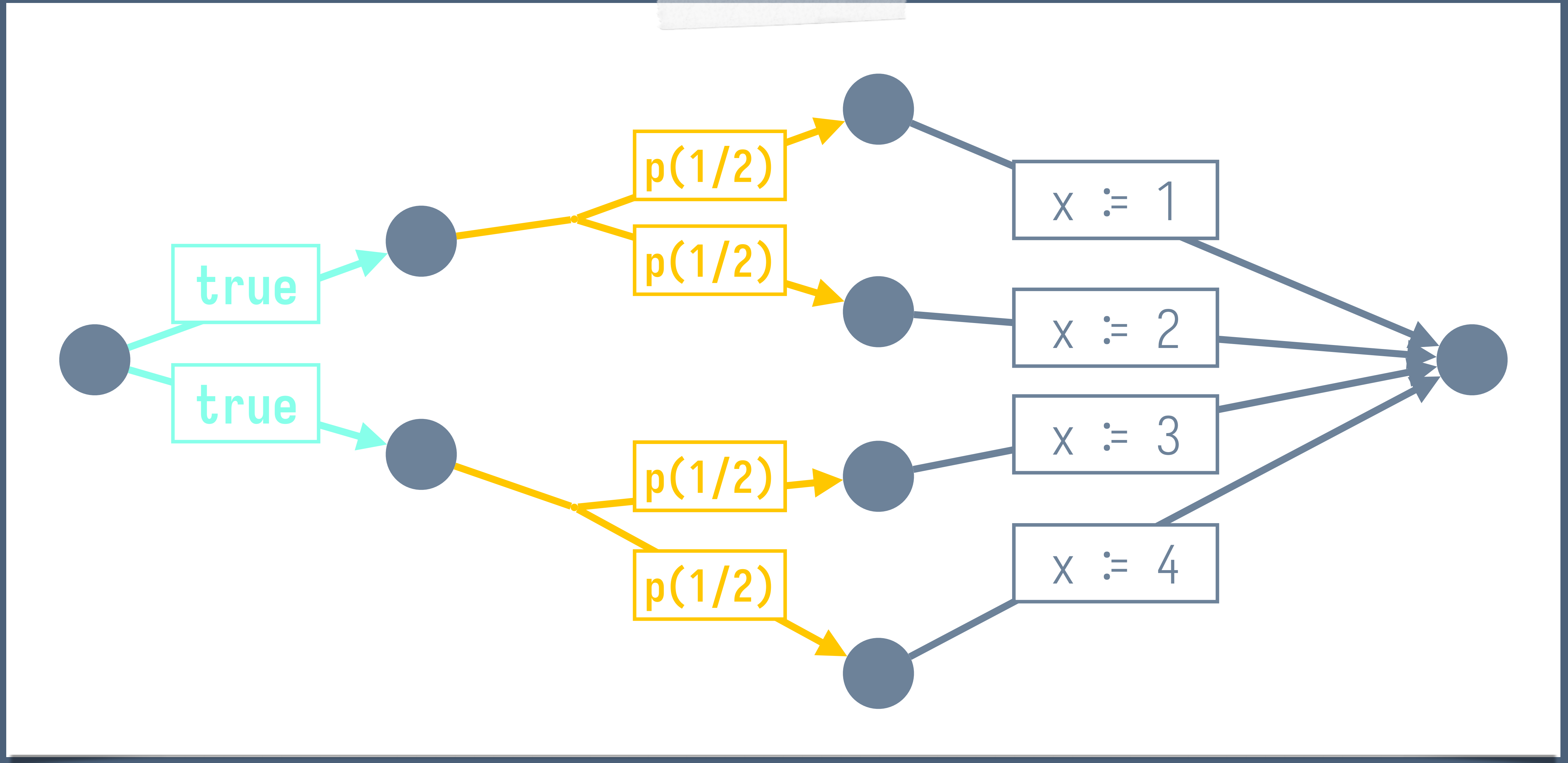
Control-flow Hyper-graph



```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```

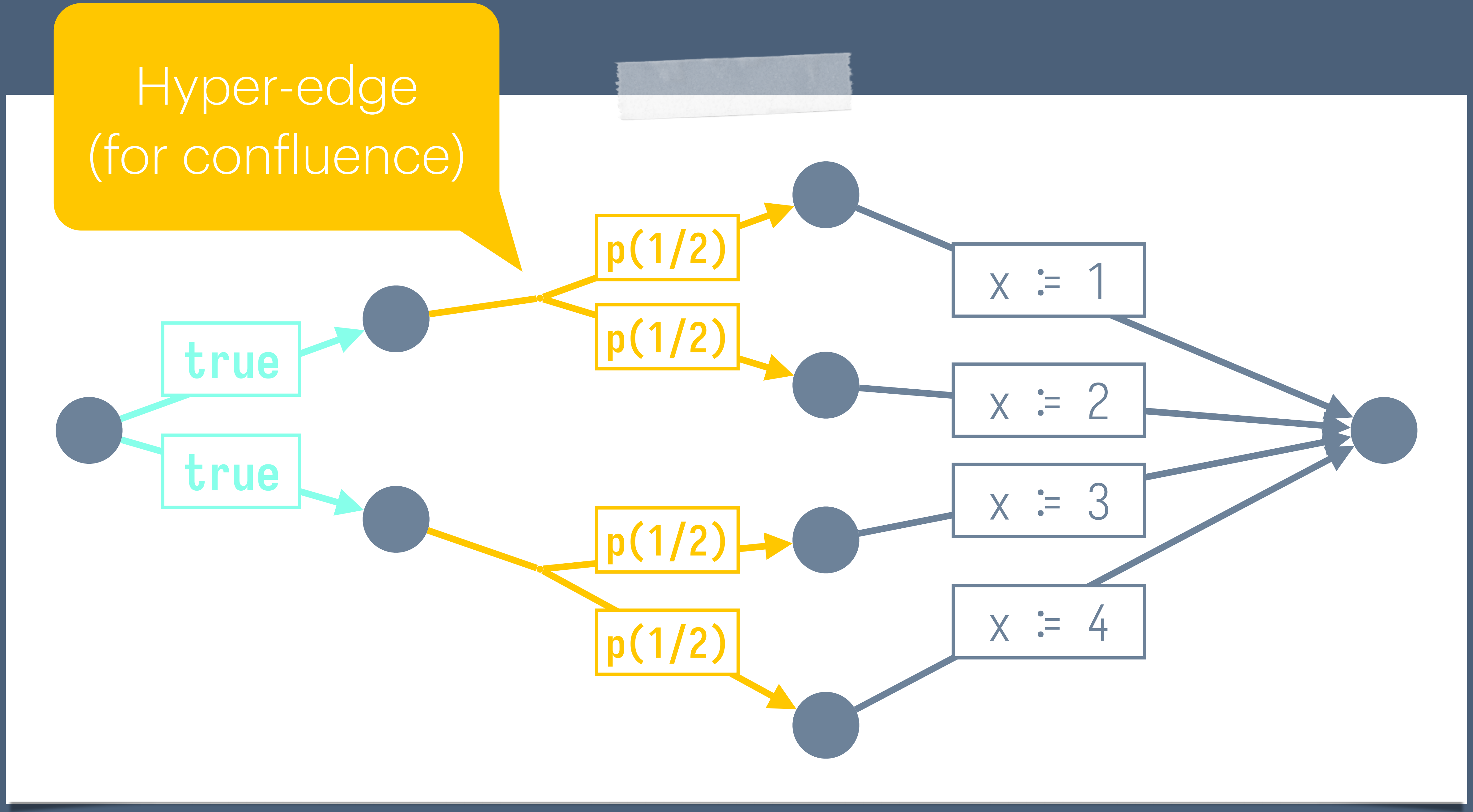
Control-flow Hyper-graph

```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```



Control-flow Hyper-graph

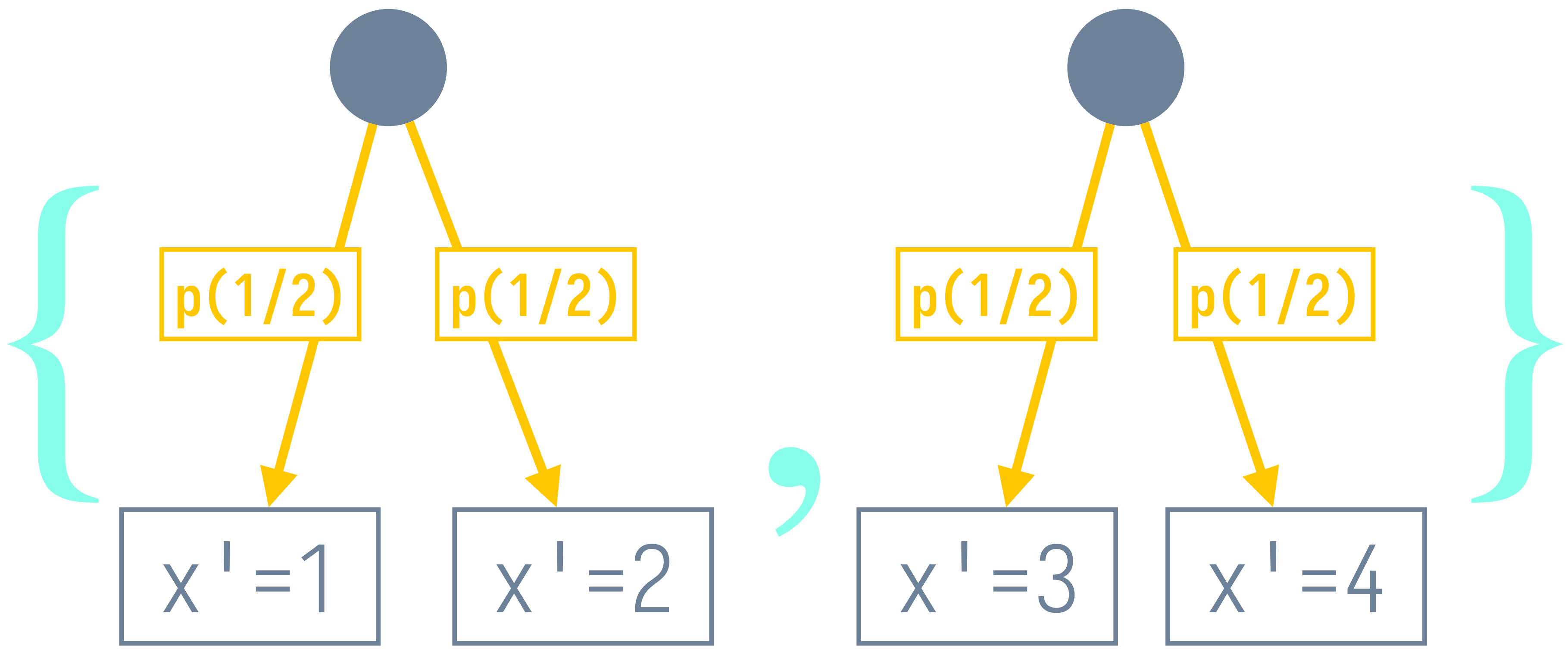
```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```

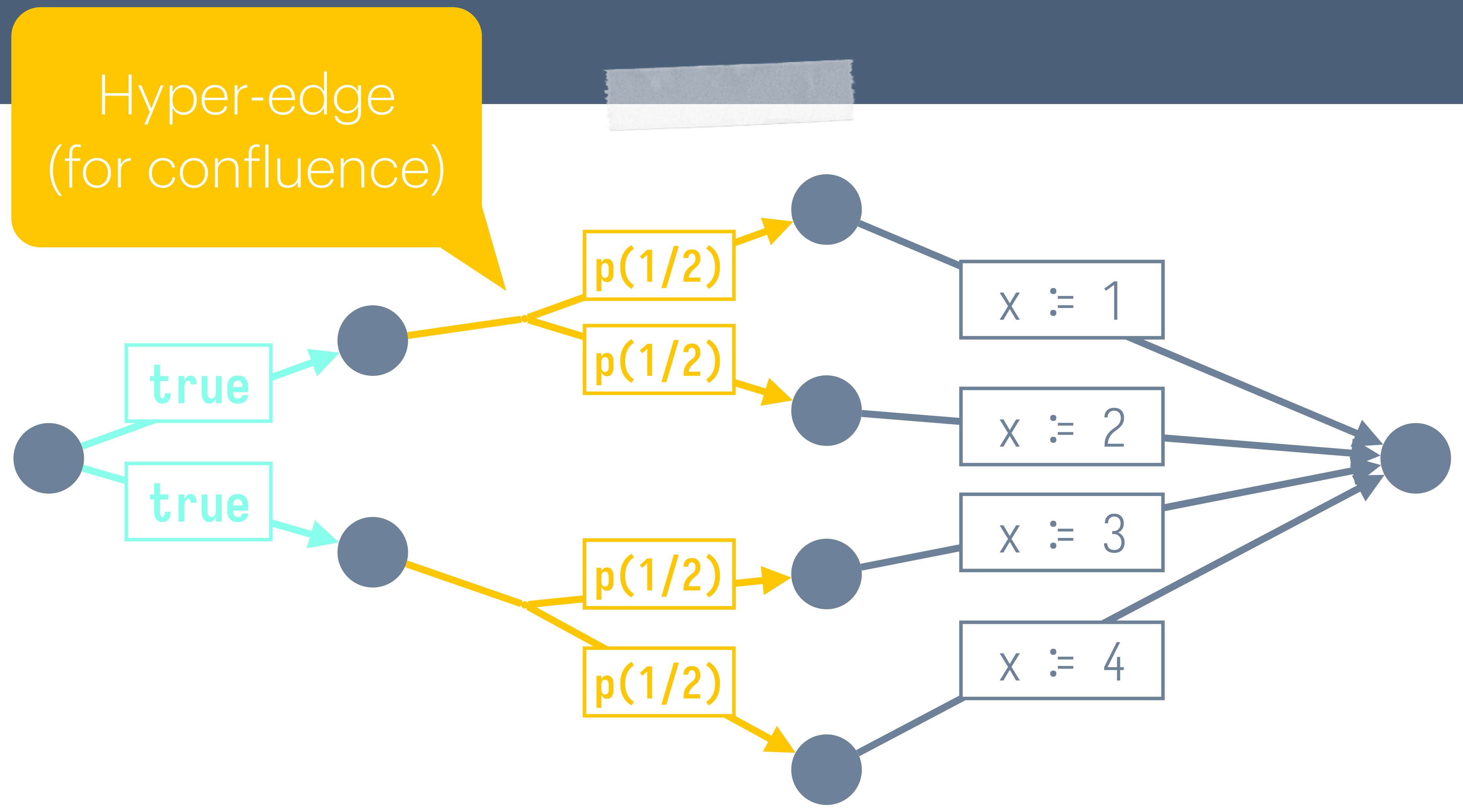


Hyper-edge
(for confluence)

Control-flow Hyper-graph

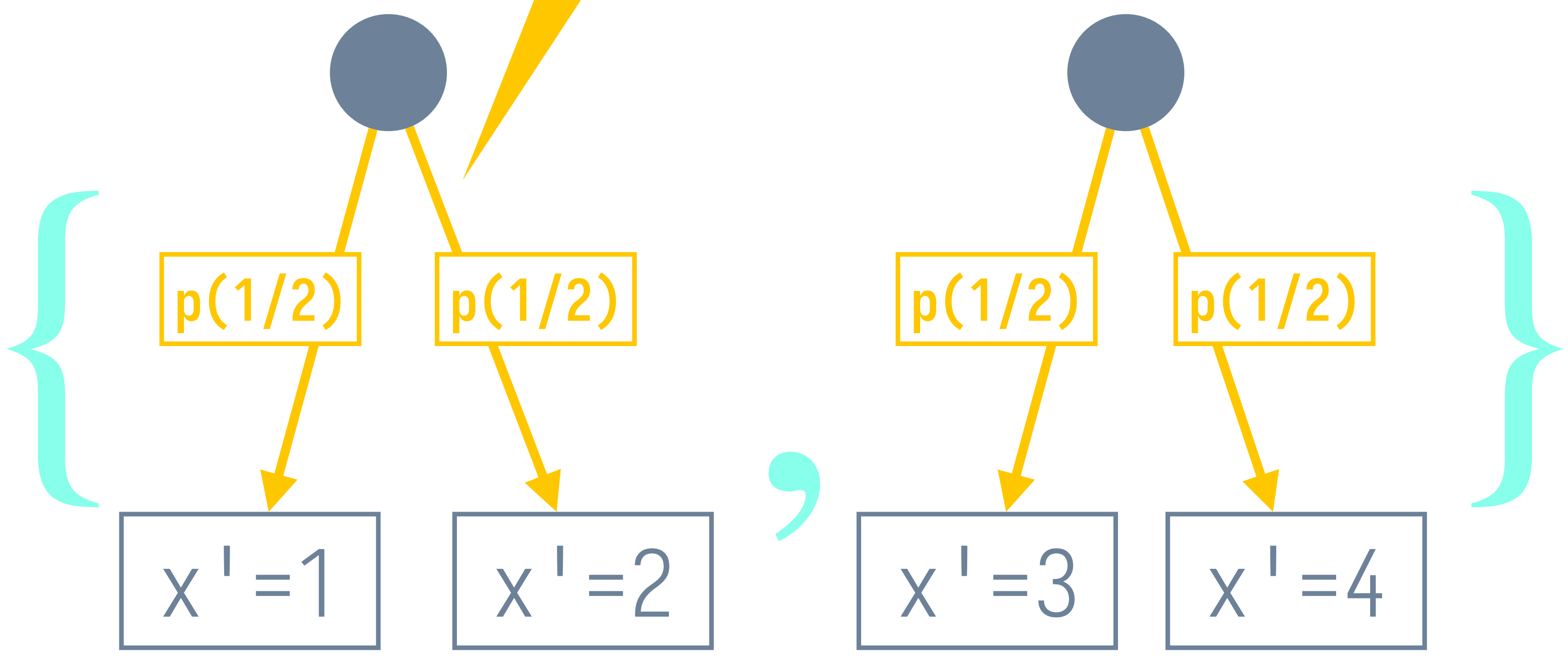
```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```




Hyper-edge
(for confluence)

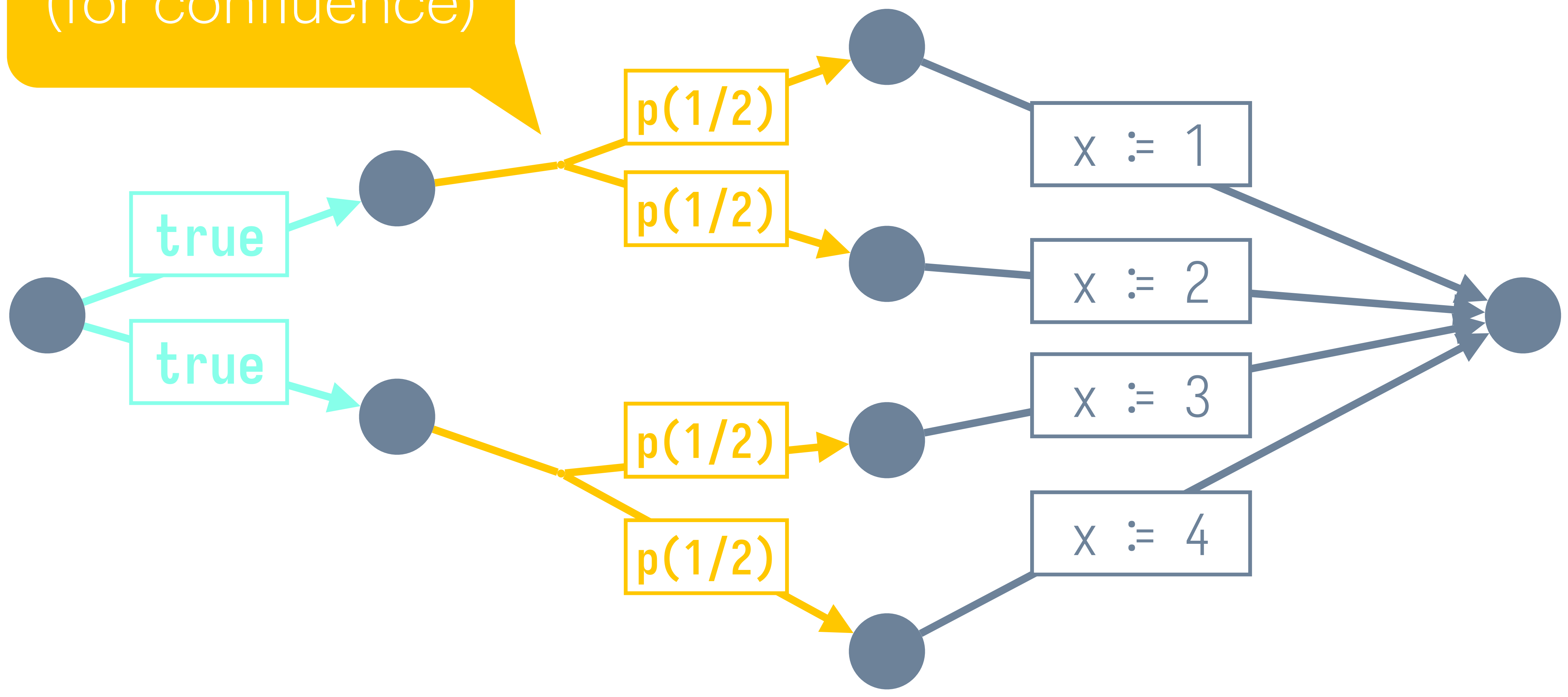
Control-flow Hyper-graph

```
if
| true → x :∈ (1 @ 1/2 | 2 @ 1/2)
| true → x :∈ (3 @ 1/2 | 4 @ 1/2)
fi
```



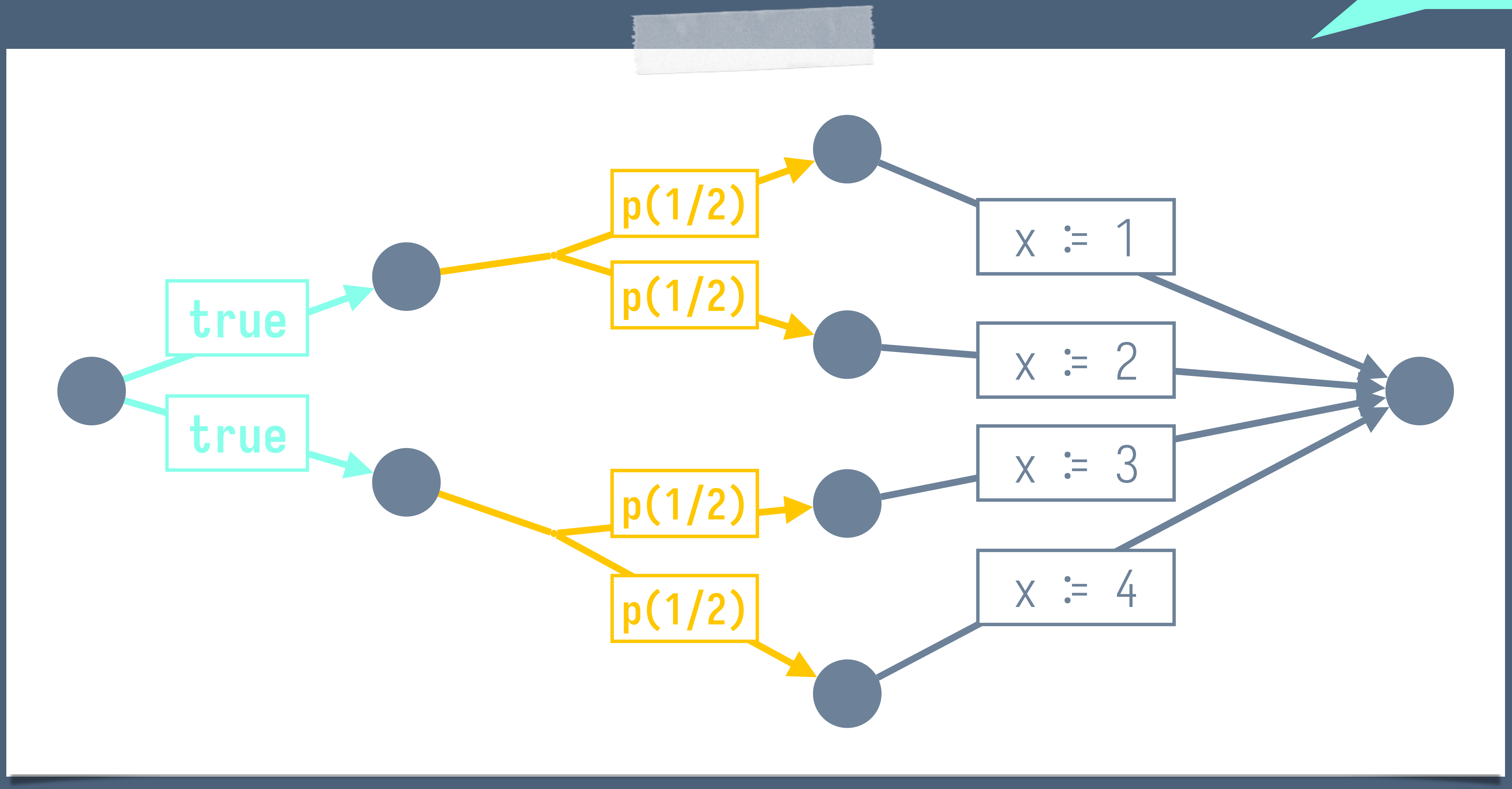
Hyper-path
(like a tree)

Hyper-edge
(for confluence)



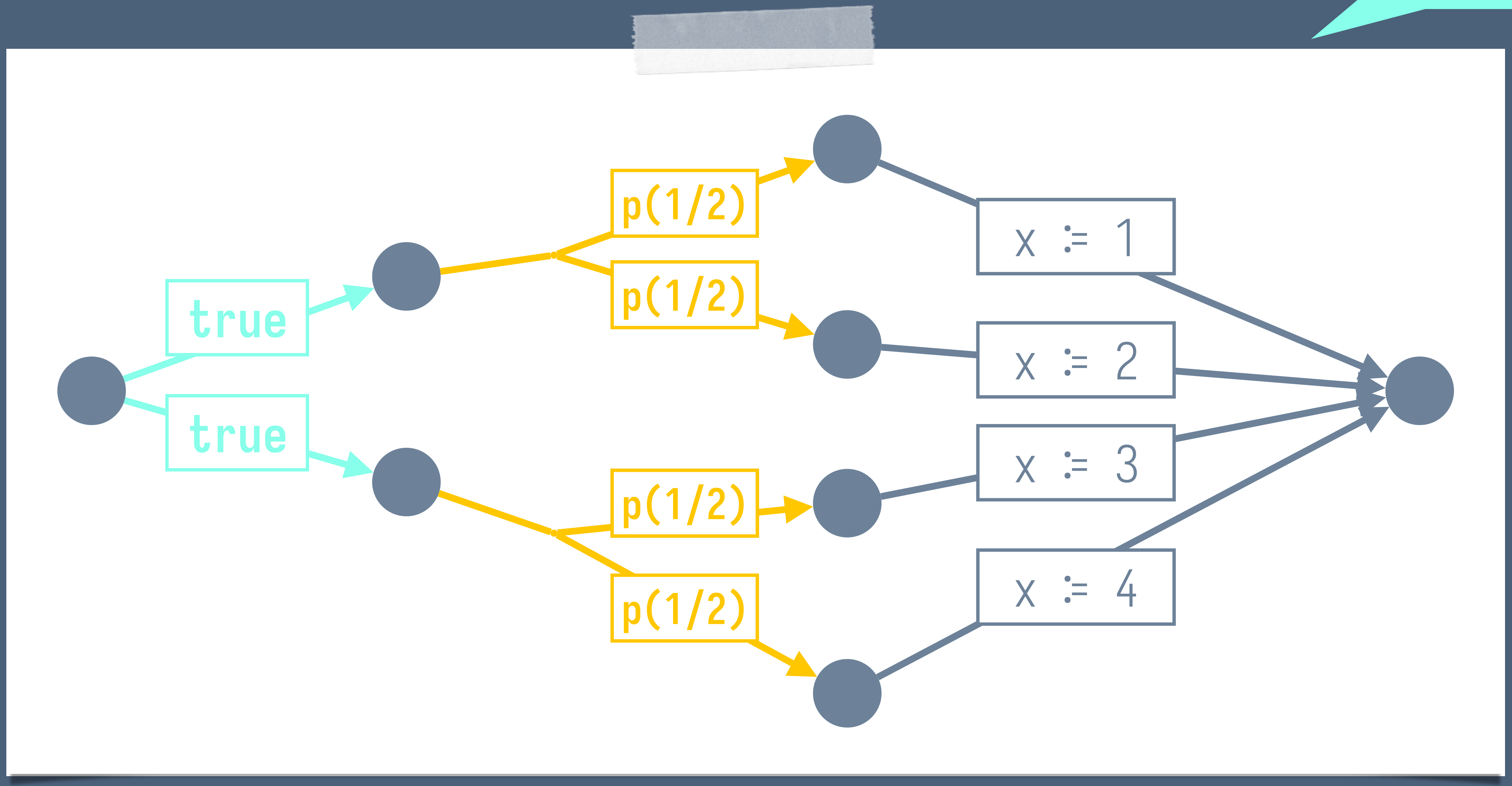
Tree Expression

Control-flow hyper-graph

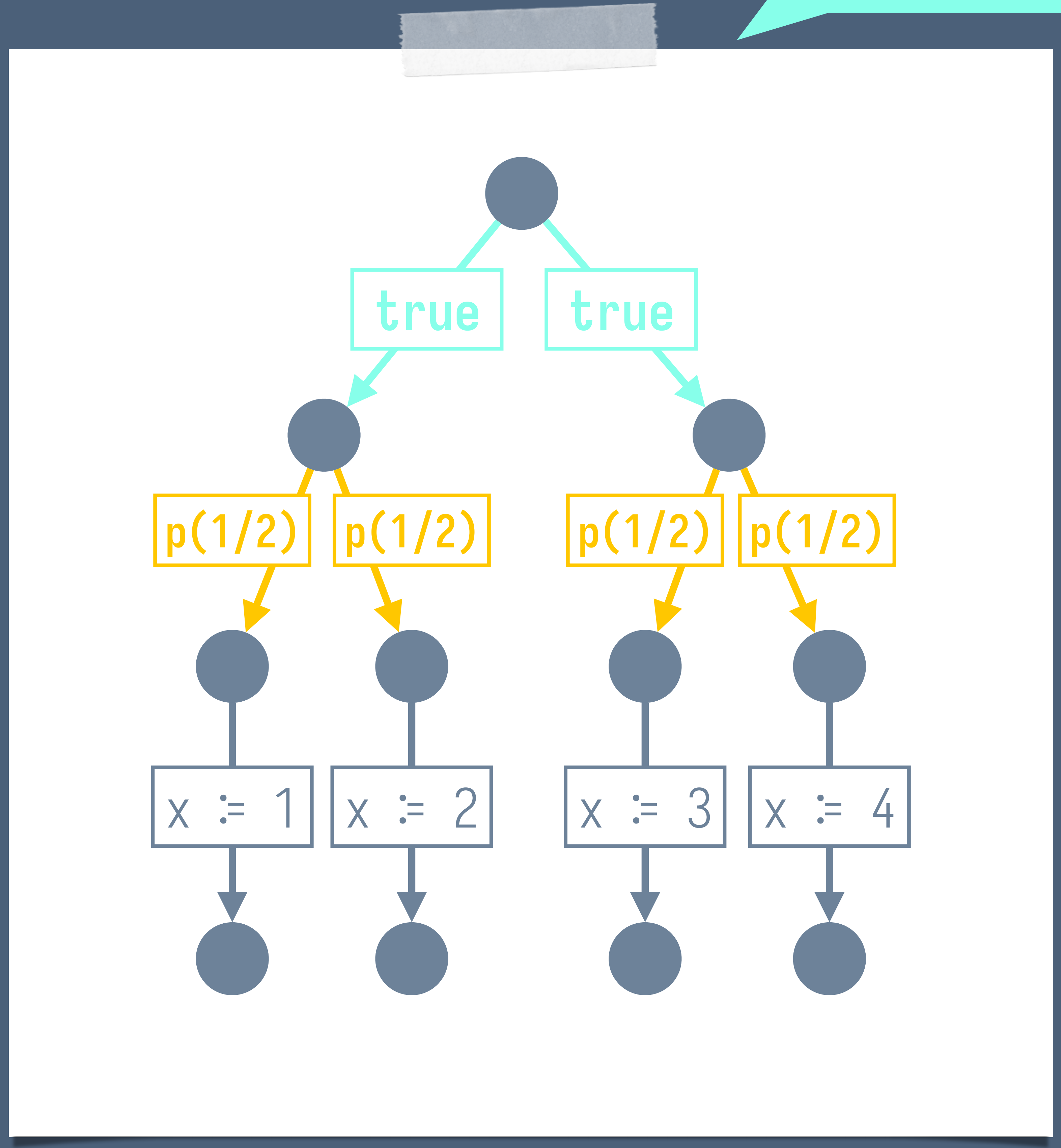


Tree Expression

Control-flow hyper-graph

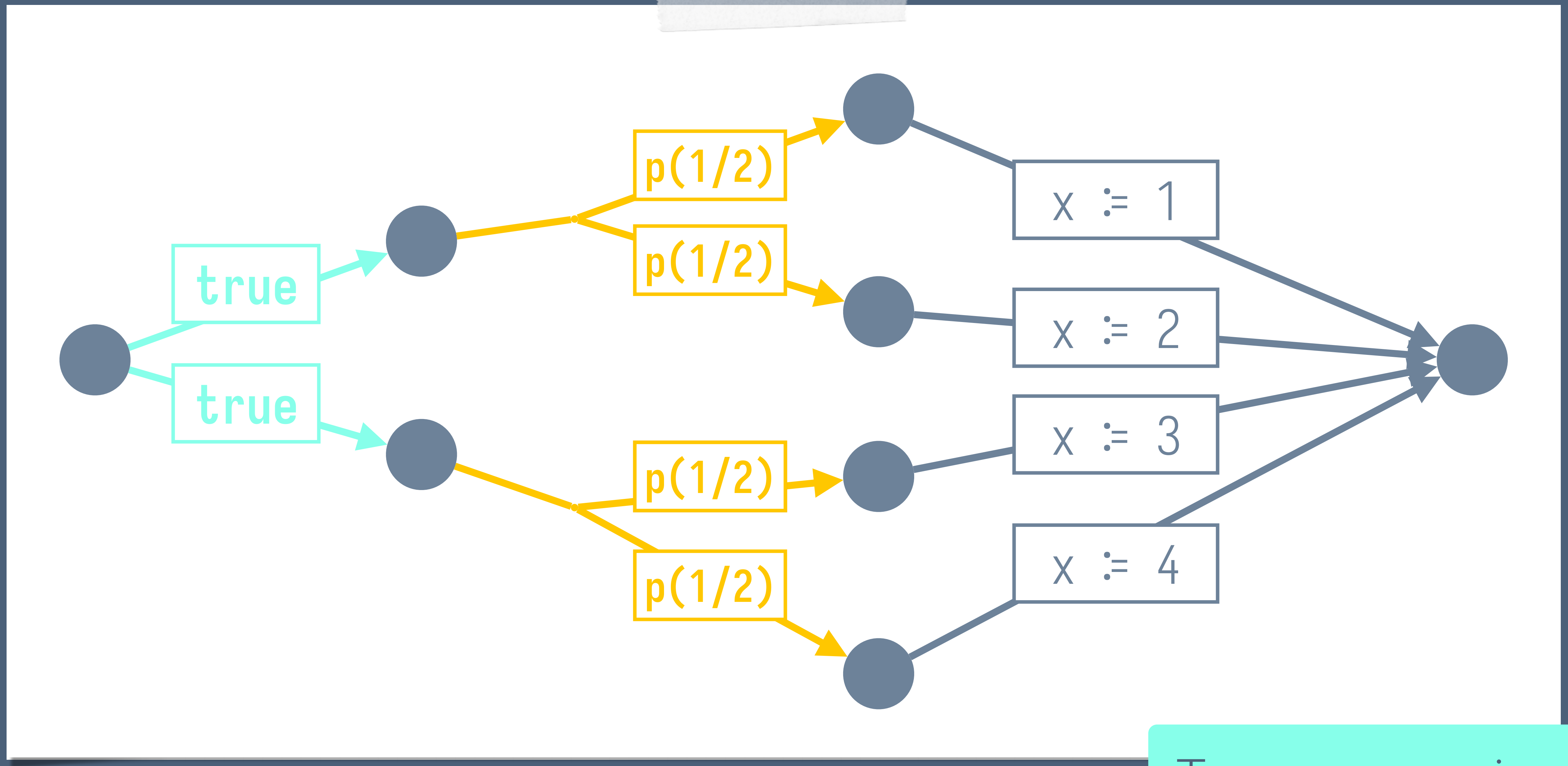


Tree expression (graphic)



Tree Expression

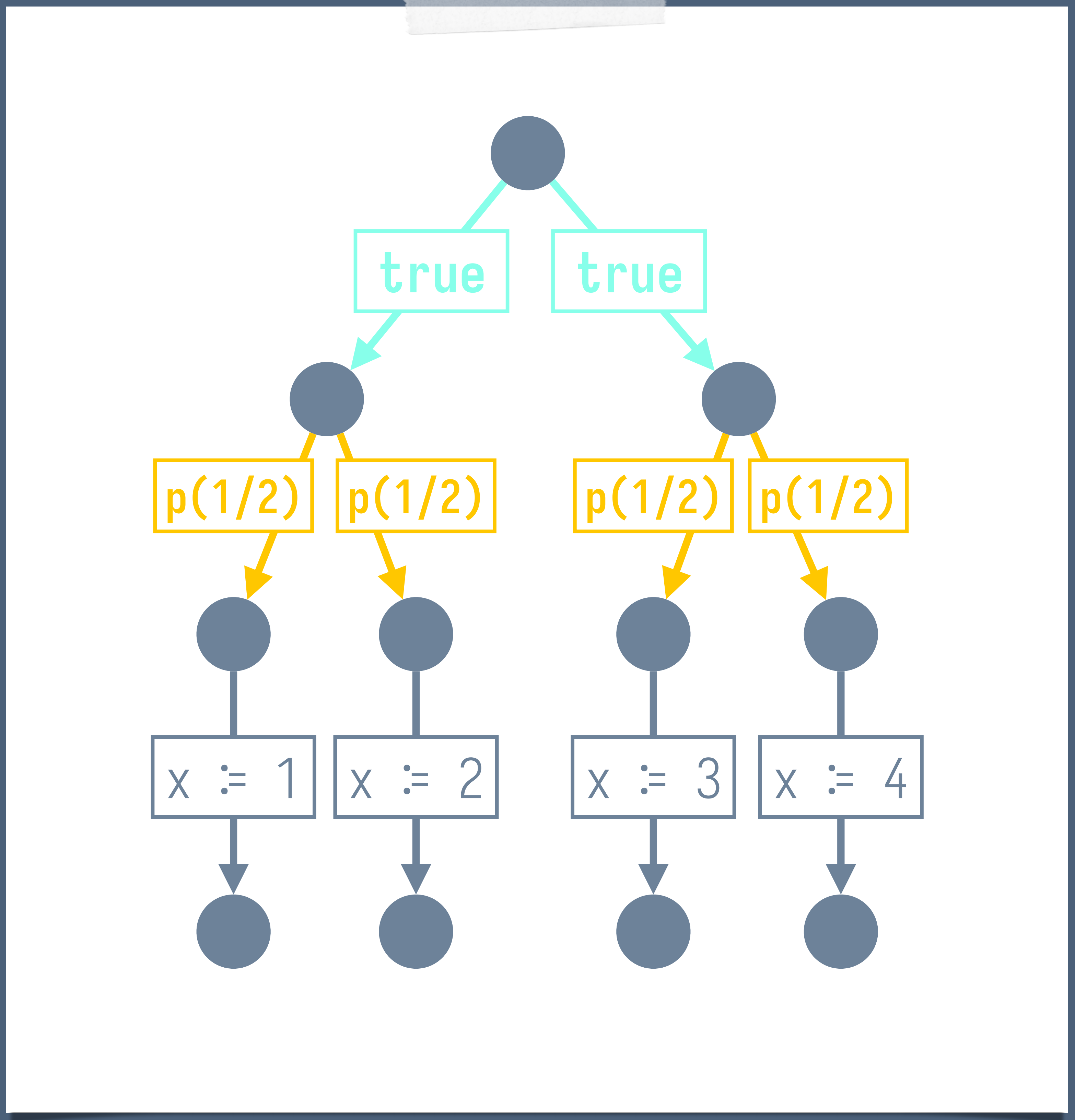
Control-flow hyper-graph



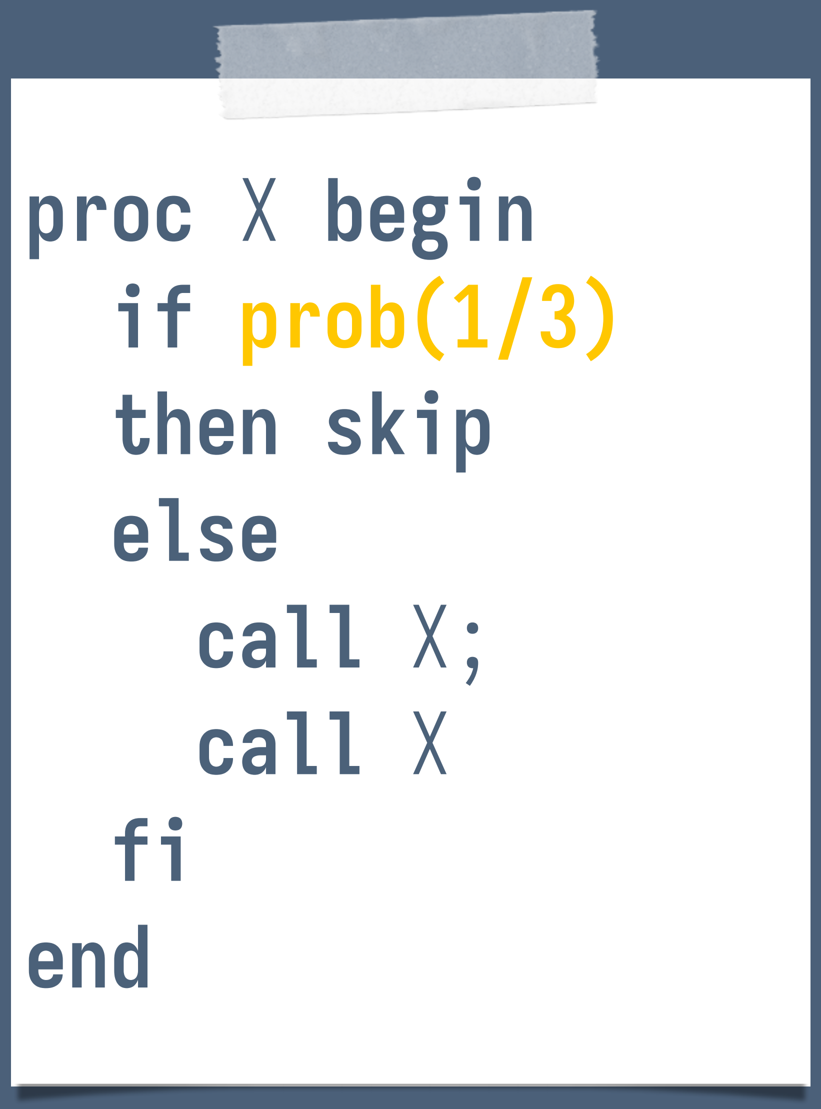
Tree expression (literal)

$$
\begin{aligned}
&ndet(prob[1/2](seq[\mathtt{x:=1}](\varepsilon), seq[\mathtt{x:=2}](\varepsilon)),\\
&\qquad prob[1/2](seq[\mathtt{x:=3}](\varepsilon), seq[\mathtt{x:=4}](\varepsilon)))
\end{aligned}
$$

Tree expression (graphic)



Tree Expression

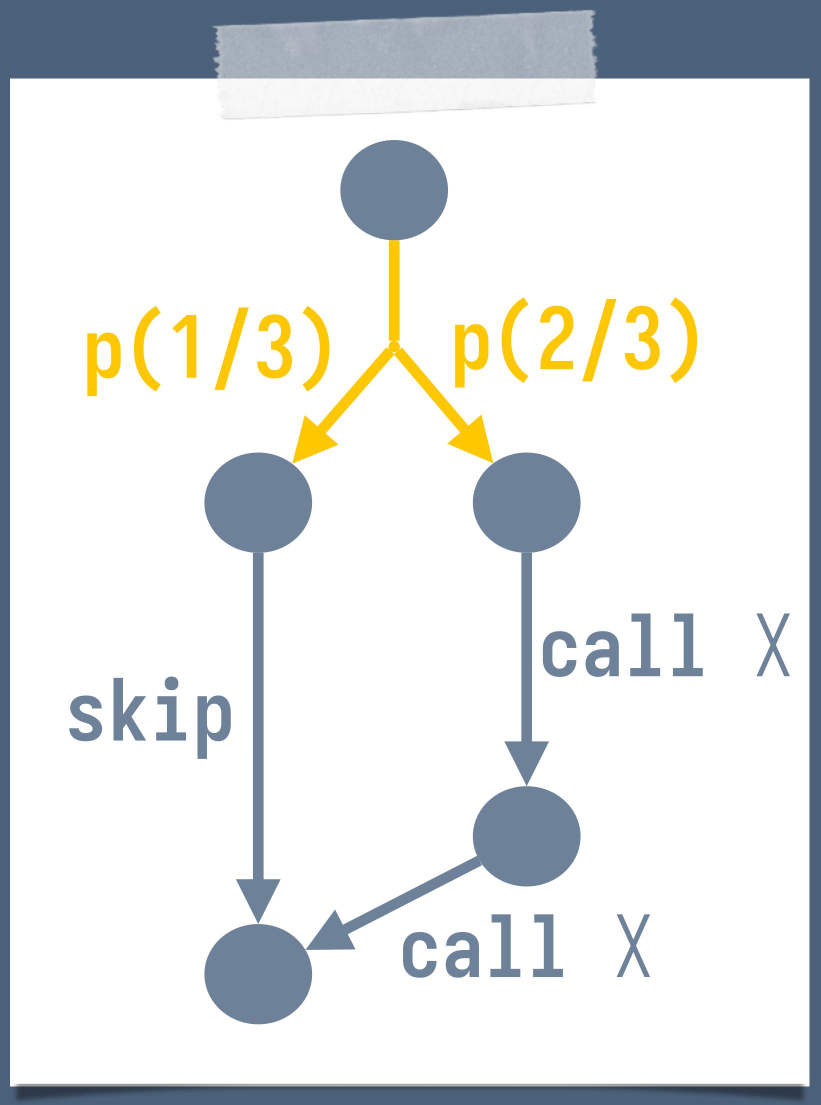


```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

Tree Expression

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

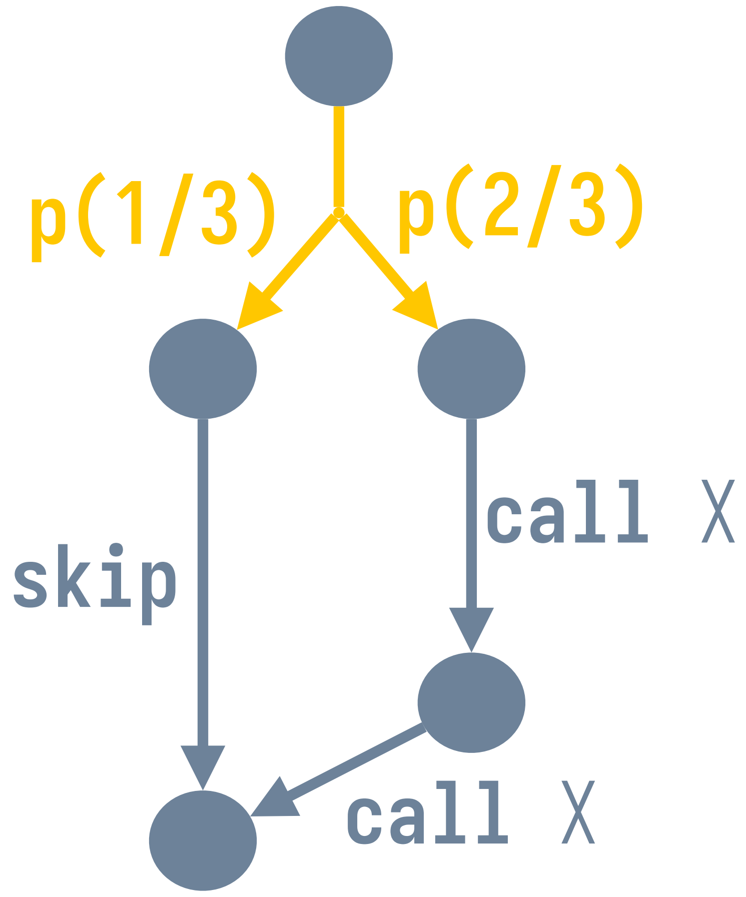
Control-flow hyper-graph



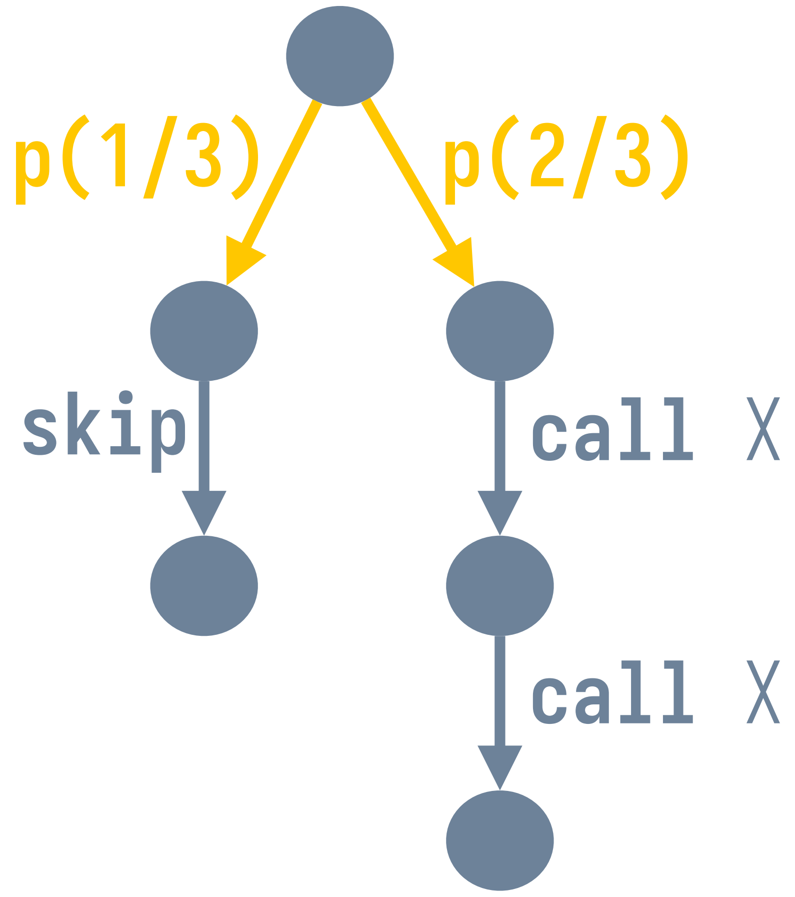
Tree Expression

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

Control-flow hyper-graph



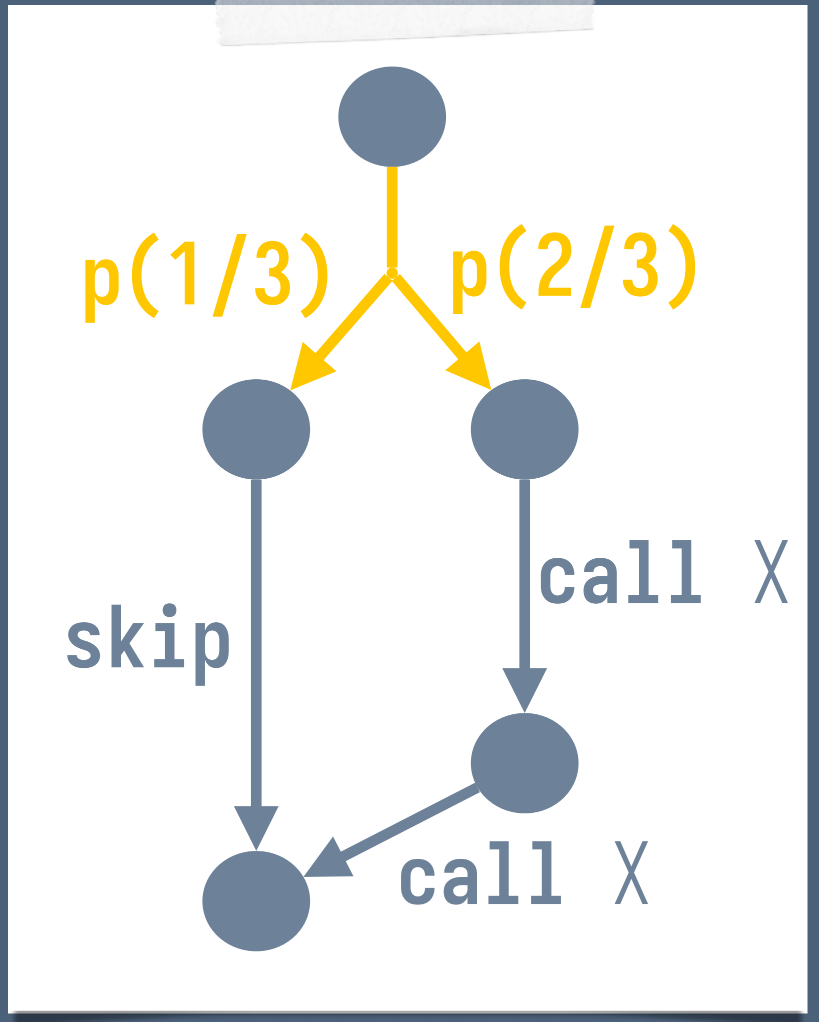
Tree expression (graphic)



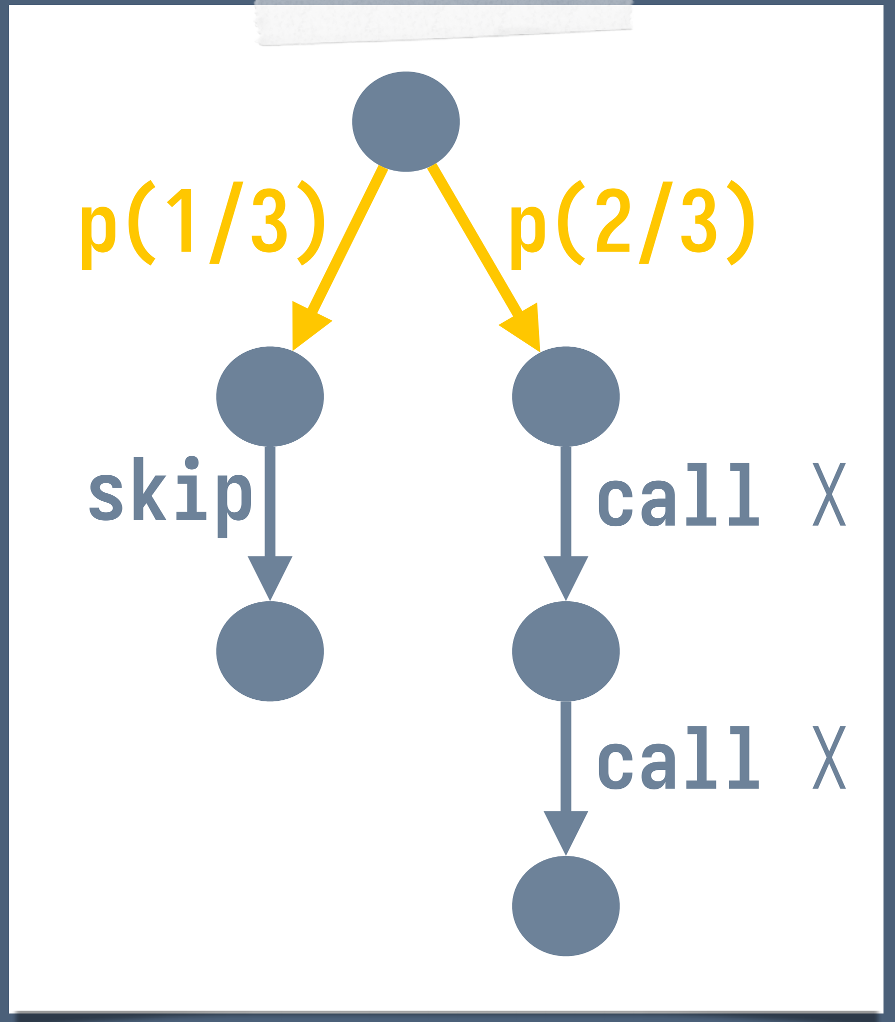
Tree Expression

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

Control-flow hyper-graph



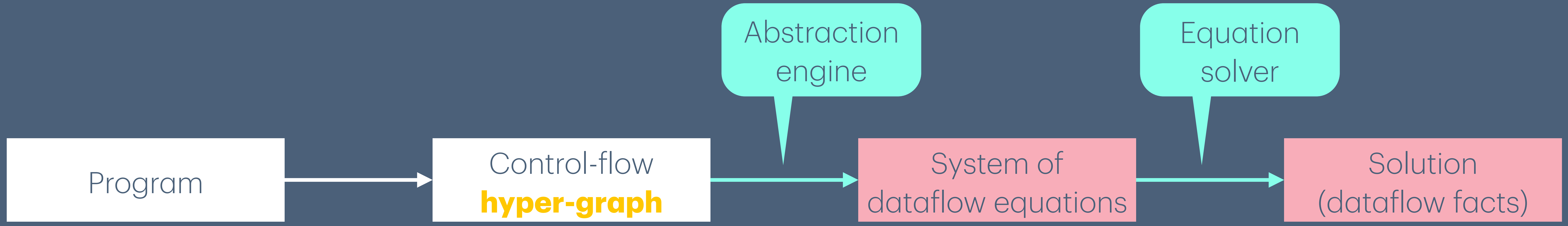
Tree expression (graphic)



Tree expression (literal)

$$X = prob[1/3](seq[\mathtt{skip}](\varepsilon), call[X](call[X](\varepsilon)))$$

Towards Multiple Combine Operations



Abstraction engine

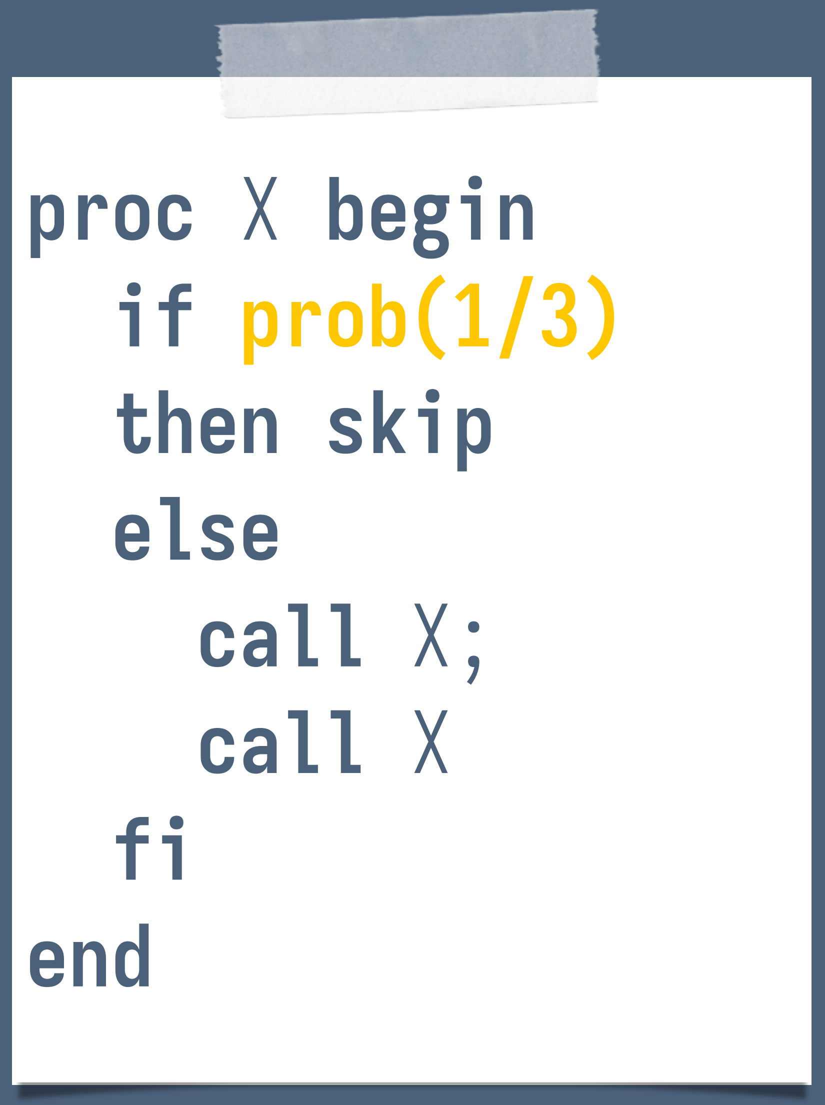
Equation solver

Program

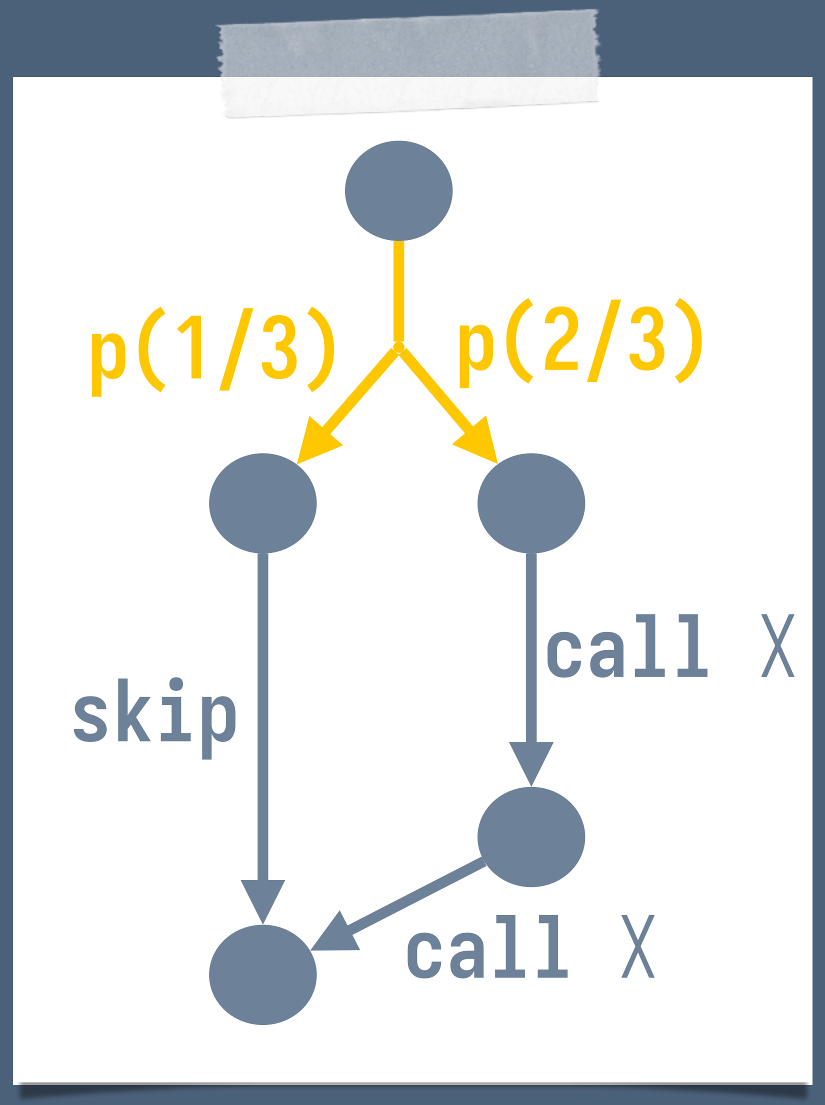
Control-flow **hyper-graph**

System of dataflow equations

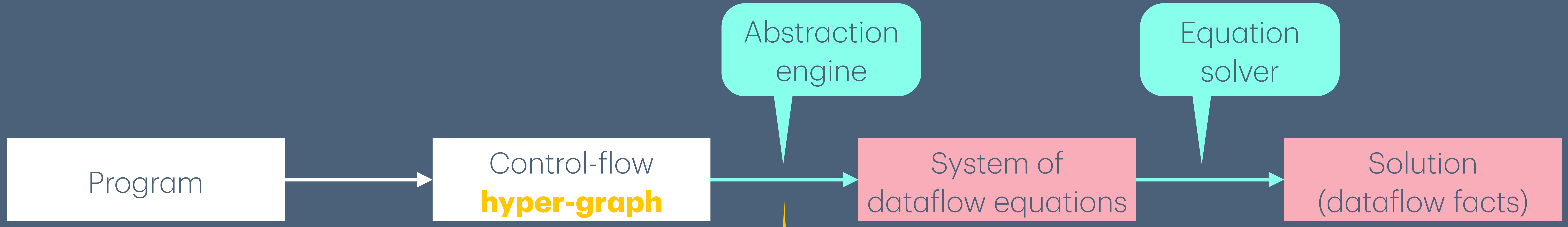
Solution (dataflow facts)



```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



Towards Multiple Combine Operations

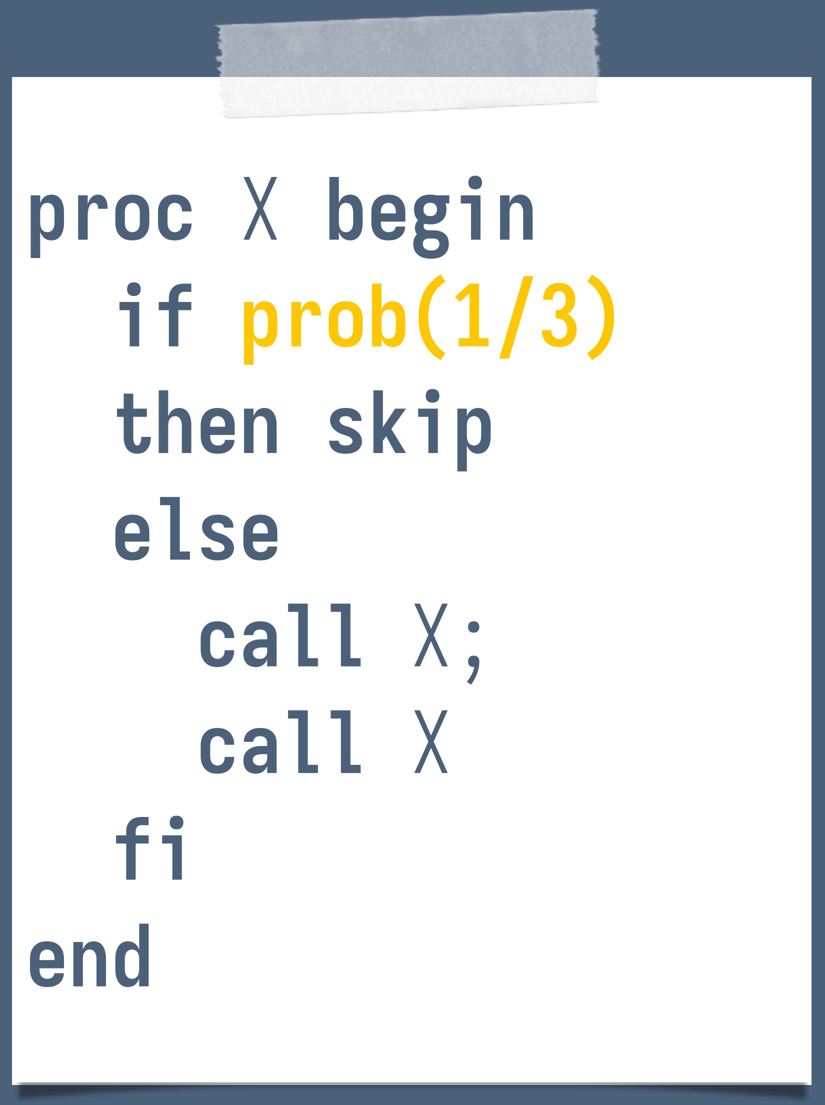


Abstraction engine

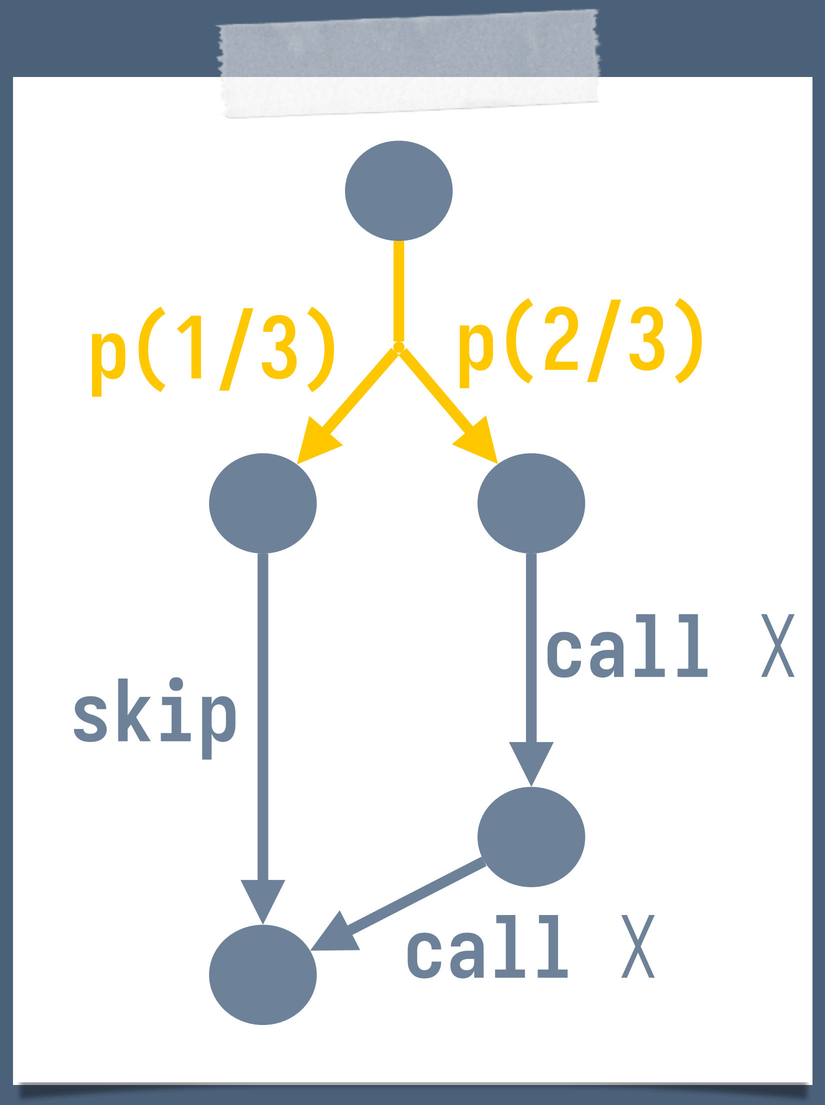
Equation solver

Program → Control-flow **hyper-graph** → System of dataflow equations → Solution (dataflow facts)

How to interpret tree expressions, algebraically?



```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



Markov Algebras

semirings + more combine operations

Markov Algebras

semirings + more combine operations

$$\left\langle M, \oplus , \otimes , {}_{\phi}\oplus , \sqcap , \underline{0}, \underline{1} \right\rangle$$

Markov Algebras

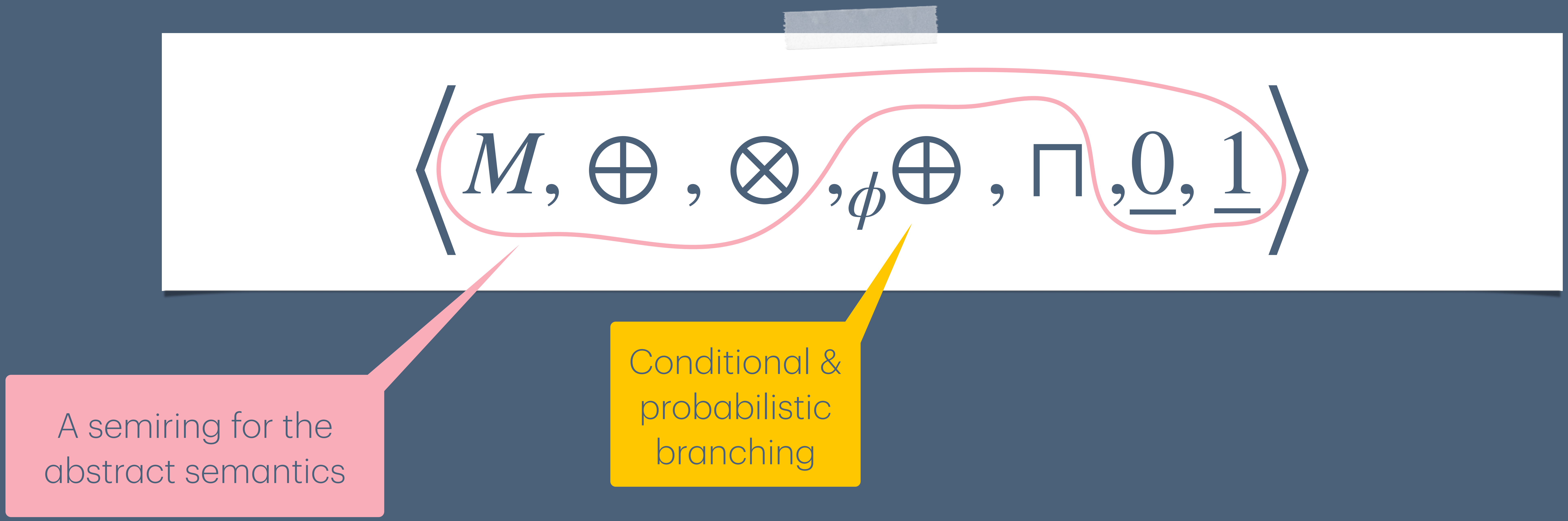
semirings + more combine operations

$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

A semiring for the abstract semantics

Markov Algebras

semirings + more combine operations



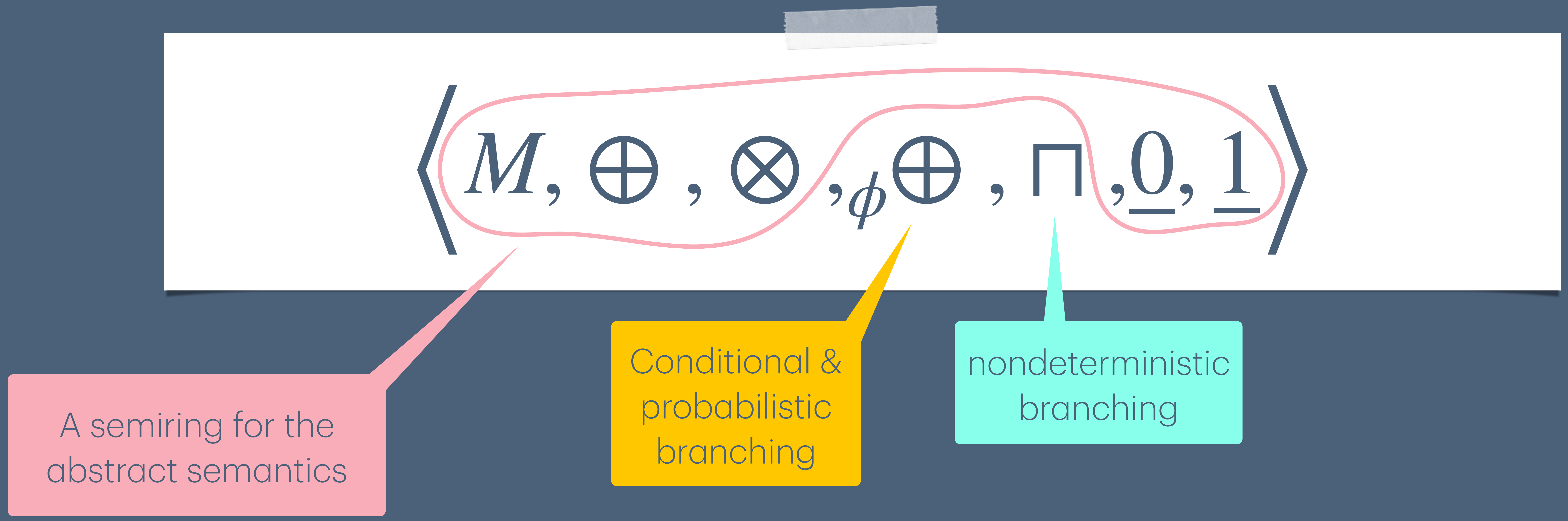
$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

A semiring for the abstract semantics

Conditional & probabilistic branching

Markov Algebras

semirings + more combine operations



$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

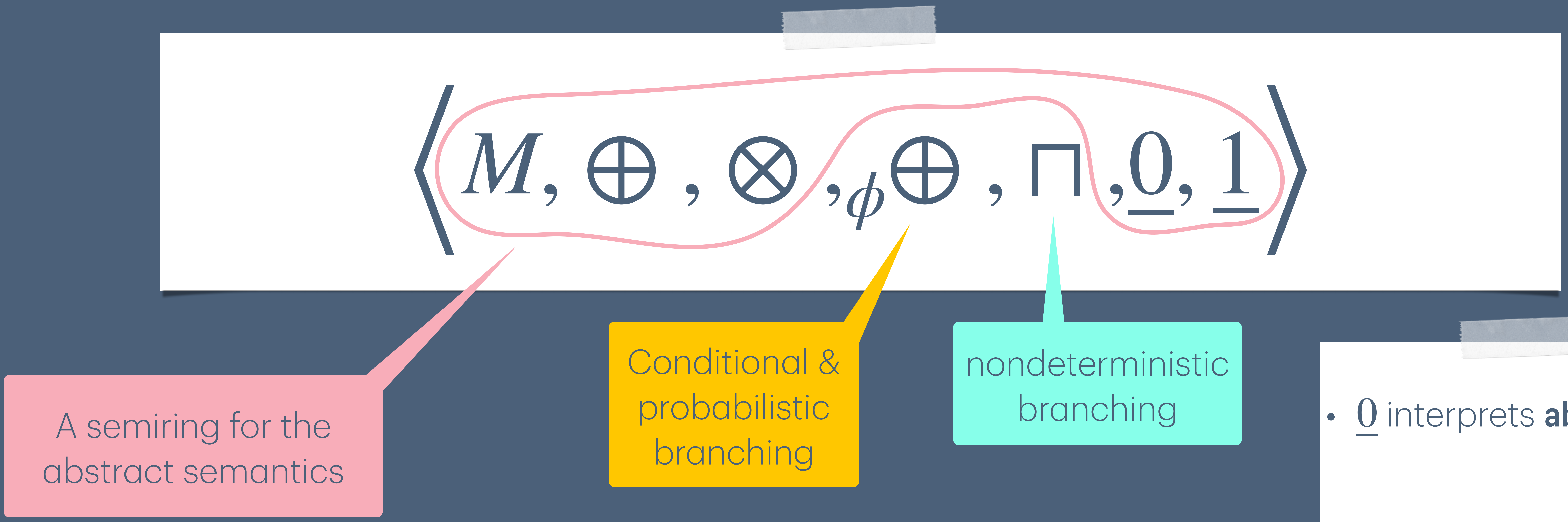
A semiring for the abstract semantics

Conditional & probabilistic branching

nondeterministic branching

Markov Algebras

semirings + more combine operations



$$\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \rangle$$

A semiring for the abstract semantics

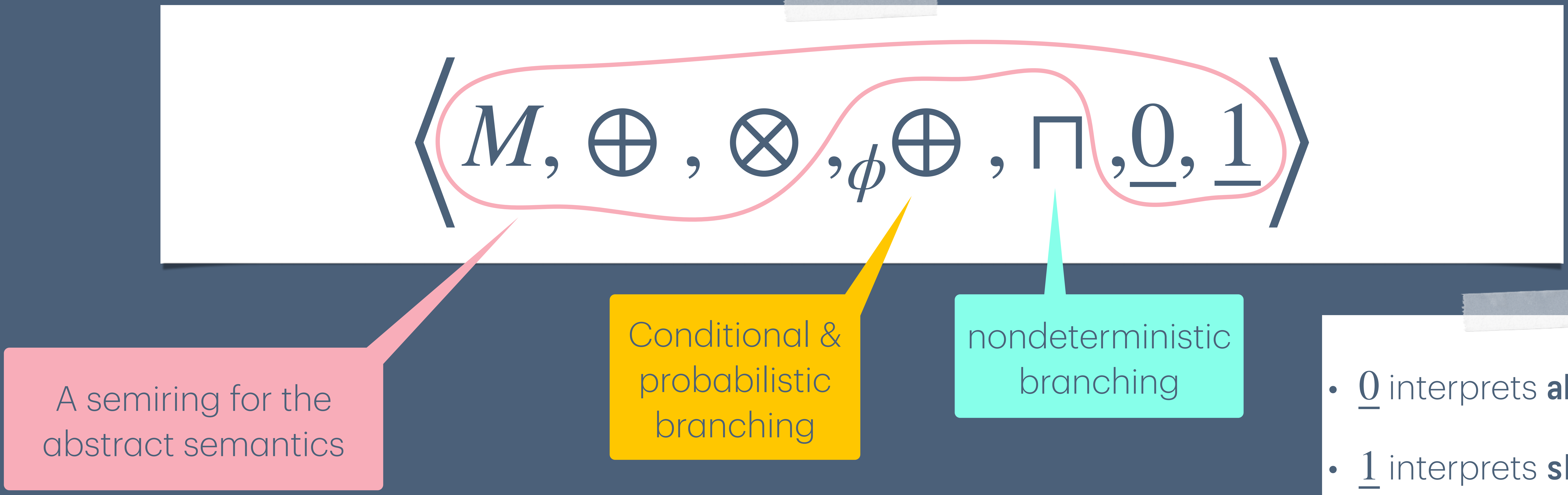
Conditional & probabilistic branching

nondeterministic branching

- $\underline{0}$ interprets **`abort`**

Markov Algebras

semirings + more combine operations



$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

A semiring for the abstract semantics

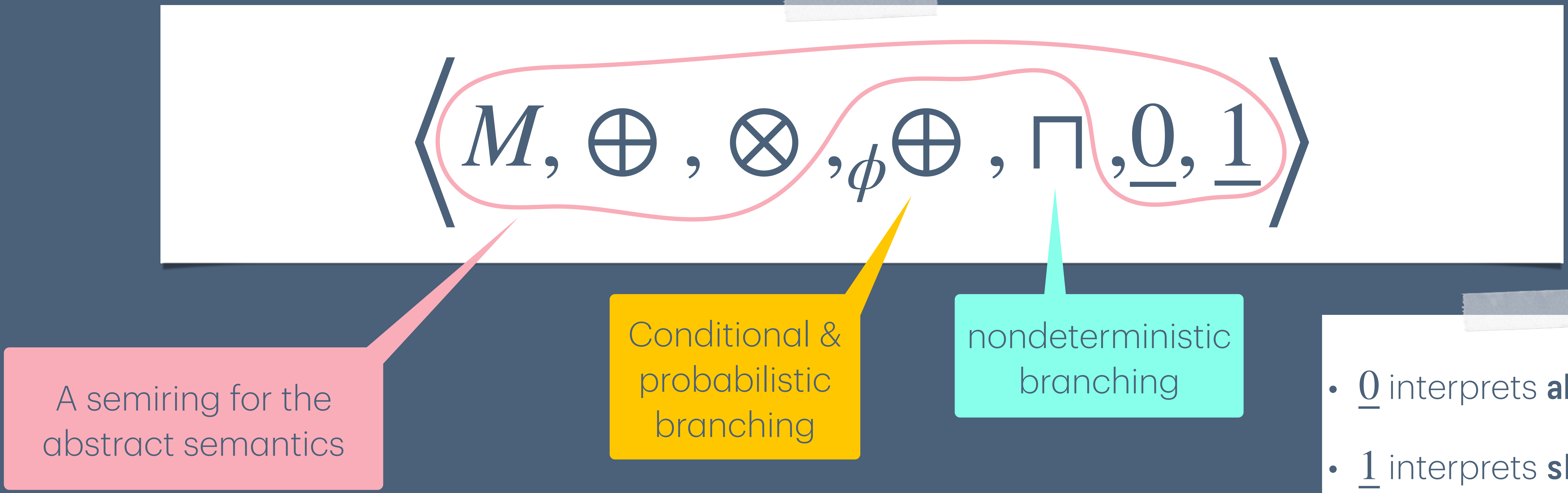
Conditional & probabilistic branching

nondeterministic branching

- $\underline{0}$ interprets **`abort`**
- $\underline{1}$ interprets **`skip`**

Markov Algebras

semirings + more combine operations



$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

A semiring for the abstract semantics

Conditional & probabilistic branching

nondeterministic branching

- $\underline{0}$ interprets **`abort`**
- $\underline{1}$ interprets **`skip`**
- Algebraic laws:
  - $a\,{}_{p}\!\oplus b = b\,{}_{1-p}\!\oplus a$
  - $a\,{}_{\varphi}\!\oplus b = b\,{}_{\neg\varphi}\!\oplus a$
  - ……

Interpretation of Tree Expressions

using a Markov Algebra

Interpretation of Tree Expressions

using a Markov Algebra

$$
\begin{aligned}
\mathscr{I}(prob[p](E_1, E_2)) &= \mathscr{I}(E_1)_{p}\!\oplus \mathscr{I}(E_2) \\
\mathscr{I}(cond[\varphi](E_1, E_2)) &= \mathscr{I}(E_1)_{\varphi}\!\oplus \mathscr{I}(E_2) \\
\mathscr{I}(ndet(E_1, E_2)) &= \mathscr{I}(E_1) \sqcap \mathscr{I}(E_2) \\
\mathscr{I}(seq[\mathtt{act}](E)) &= \underline{\mathtt{act}} \otimes \mathscr{I}(E) \\
\mathscr{I}(call[X_i](E)) &= X_i \otimes \mathscr{I}(E) \\
\mathscr{I}(\varepsilon) &= \underline{1}
\end{aligned}
$$

Interpretation of Tree Expressions

using a Markov Algebra

$$
\begin{aligned}
\mathscr{I}(\mathit{prob}[p](E_1, E_2)) &= \mathscr{I}(E_1) {}_p\!\oplus \mathscr{I}(E_2) \\
\mathscr{I}(\mathit{cond}[\varphi](E_1, E_2)) &= \mathscr{I}(E_1) {}_\varphi\!\oplus \mathscr{I}(E_2) \\
\mathscr{I}(\mathit{ndet}(E_1, E_2)) &= \mathscr{I}(E_1) \sqcap \mathscr{I}(E_2) \\
\mathscr{I}(\mathit{seq}[\texttt{act}](E)) &= \underline{\texttt{act}} \otimes \mathscr{I}(E) \\
\mathscr{I}(\mathit{call}[X_i](E)) &= X_i \otimes \mathscr{I}(E) \\
\mathscr{I}(\varepsilon) &= \underline{1}
\end{aligned}
$$

$$X = \mathit{prob}[1/3](\mathit{seq}[\texttt{skip}](\varepsilon), \mathit{call}[X](\mathit{call}[X](\varepsilon)))$$

Interpretation of Tree Expressions

using a Markov Algebra

$$\mathcal{I}(prob[p](E_1, E_2)) = \mathcal{I}(E_1)\,{}_p\!\oplus \mathcal{I}(E_2)$$

$$\mathcal{I}(cond[\varphi](E_1, E_2)) = \mathcal{I}(E_1)\,{}_\varphi\!\oplus \mathcal{I}(E_2)$$

$$\mathcal{I}(ndet(E_1, E_2)) = \mathcal{I}(E_1) \sqcap \mathcal{I}(E_2)$$

$$\mathcal{I}(seq[\mathtt{act}](E)) = \underline{\mathtt{act}} \otimes \mathcal{I}(E)$$

$$\mathcal{I}(call[X_i](E)) = X_i \otimes \mathcal{I}(E)$$

$$\mathcal{I}(\varepsilon) = \underline{1}$$

$$X = prob[1/3](seq[\mathtt{skip}](\varepsilon), call[X](call[X](\varepsilon)))$$

$$X = (\underline{\mathtt{skip}} \otimes \underline{1})\,{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

Interpretation of Tree Expressions

using a Markov Algebra

$$
\begin{aligned}
\mathcal{I}(prob[p](E_1,E_2)) &= \mathcal{I}(E_1)\,{}_p\!\oplus \mathcal{I}(E_2)\\
\mathcal{I}(cond[\varphi](E_1,E_2)) &= \mathcal{I}(E_1)\,{}_\varphi\!\oplus \mathcal{I}(E_2)\\
\mathcal{I}(ndet(E_1,E_2)) &= \mathcal{I}(E_1) \sqcap \mathcal{I}(E_2)\\
\mathcal{I}(seq[\mathtt{act}](E)) &= \underline{\mathtt{act}} \otimes \mathcal{I}(E)\\
\mathcal{I}(call[X_i](E)) &= X_i \otimes \mathcal{I}(E)\\
\mathcal{I}(\varepsilon) &= \underline{1}
\end{aligned}
$$

For the termination-probability analysis:

$$
\begin{aligned}
a \oplus b &= a + b\\
a \otimes b &= a \cdot b\\
a\,{}_p\!\oplus b &= p \cdot a + (1-p) \cdot b\\
&\ldots
\end{aligned}
$$

$$X = prob[1/3](seq[\mathtt{skip}](\varepsilon), call[X](call[X](\varepsilon)))$$

$$X = (\underline{\mathtt{skip}} \otimes \underline{1})\,{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

Interpretation of Tree Expressions

using a Markov Algebra

$$
\begin{aligned}
\mathscr{I}(prob[p](E_1, E_2)) &= \mathscr{I}(E_1)\,{}_p\!\oplus \mathscr{I}(E_2)\\
\mathscr{I}(cond[\varphi](E_1, E_2)) &= \mathscr{I}(E_1)\,{}_\varphi\!\oplus \mathscr{I}(E_2)\\
\mathscr{I}(ndet(E_1, E_2)) &= \mathscr{I}(E_1) \sqcap \mathscr{I}(E_2)\\
\mathscr{I}(seq[\mathtt{act}](E)) &= \underline{\mathtt{act}} \otimes \mathscr{I}(E)\\
\mathscr{I}(call[X_i](E)) &= X_i \otimes \mathscr{I}(E)\\
\mathscr{I}(\varepsilon) &= \underline{1}
\end{aligned}
$$

For the termination-probability analysis:

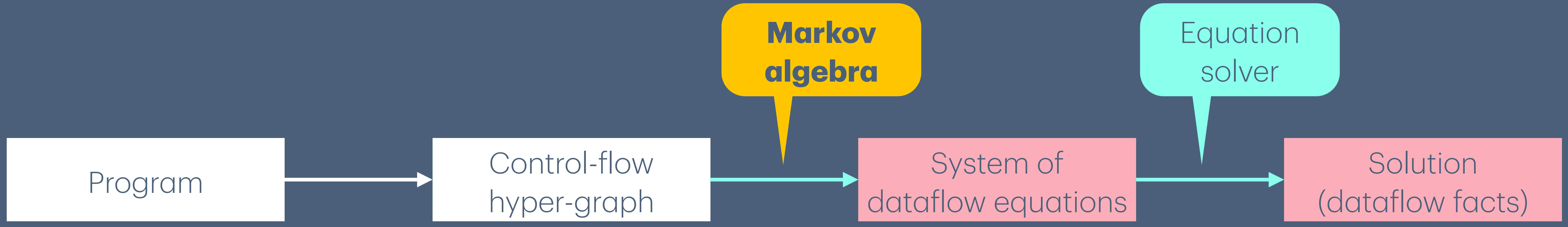
$$
\begin{aligned}
a \oplus b &= a + b\\
a \otimes b &= a \cdot b\\
a\,{}_p\!\oplus b &= p \cdot a + (1 - p) \cdot b\\
&\dots
\end{aligned}
$$

$$X = prob[1/3](seq[\mathtt{skip}](\varepsilon), call[X](call[X](\varepsilon)))$$

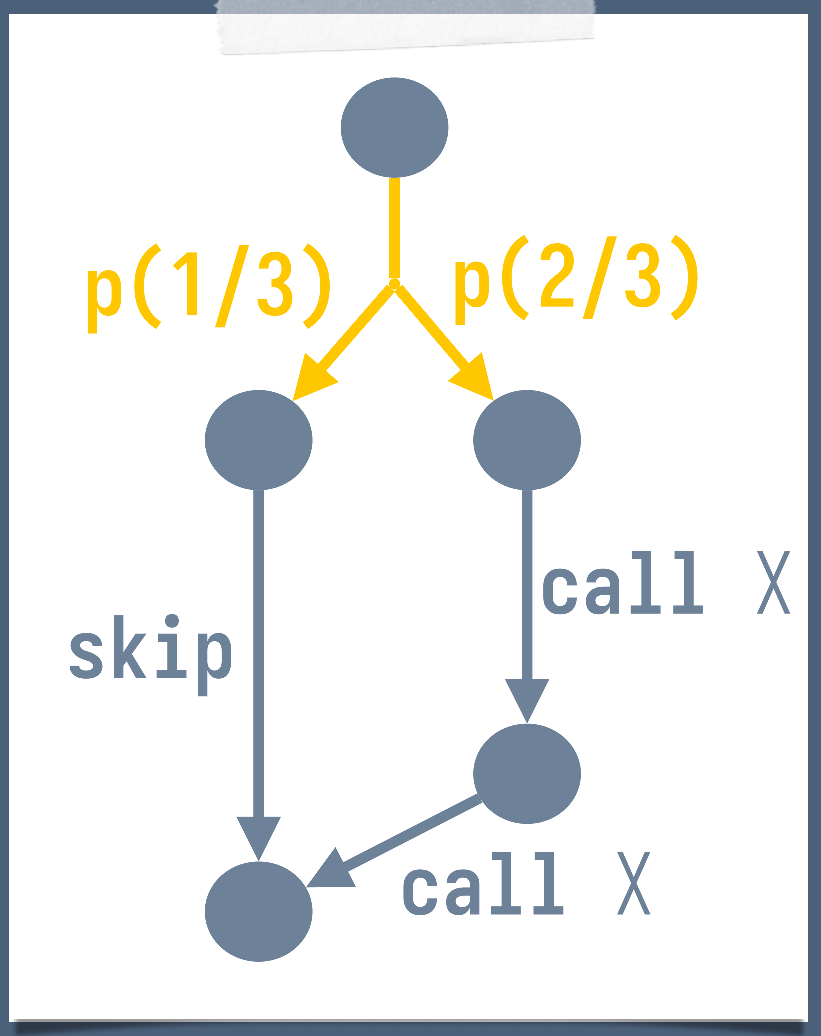
$$X = (\underline{\mathtt{skip}} \otimes \underline{1})\,{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

$$X = \frac{1}{3} \cdot (1 \cdot 1) + \frac{2}{3} \cdot (X \cdot X \cdot 1)$$

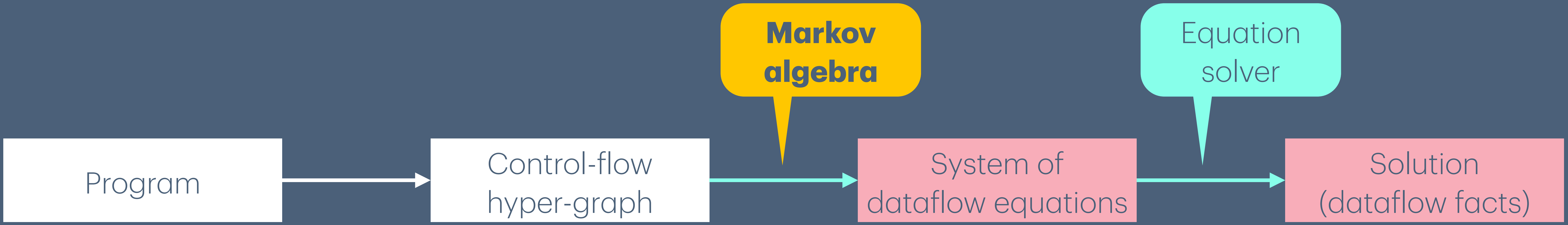
Towards Multiple Combine Operations



```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



Towards Multiple Combine Operations



Markov algebra

Equation solver

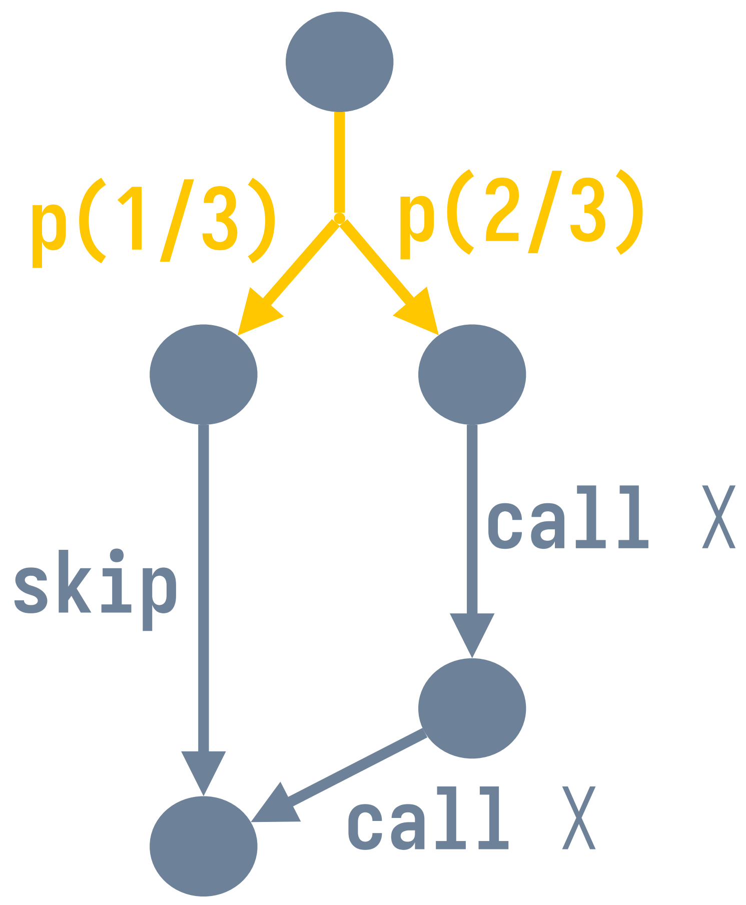
Program

Control-flow hyper-graph

System of dataflow equations

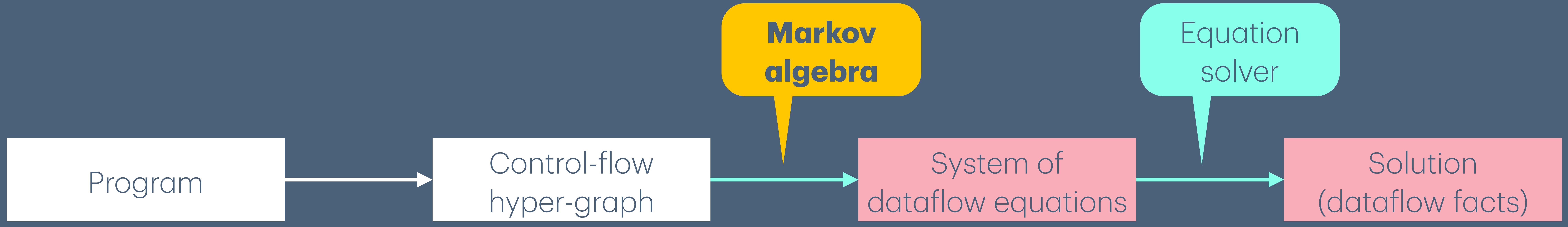
Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\underline{\texttt{skip}} \otimes \underline{1})_{1/3}\oplus (X \otimes X \otimes \underline{1})$$

Towards Multiple Combine Operations



Markov algebra

Equation solver

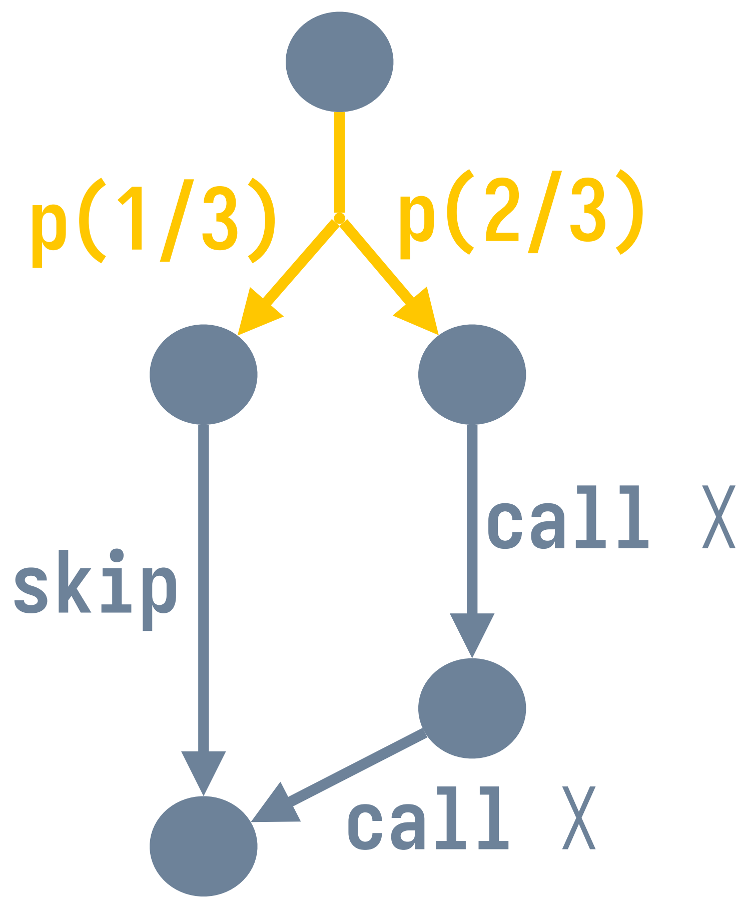
Program

Control-flow hyper-graph

System of dataflow equations

Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\underline{\mathtt{skip}} \otimes \underline{1})\,{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

How to solve such equations, algebraically?

Linearization of Tree Expressions

for Newton's Method

Linearization of Tree Expressions

for Newton's Method

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

$$D(g {}_{\phi}\!\oplus h) = Dg {}_{\phi}\!\oplus Dh$$

$$D(g \sqcap h) = ((g \oplus Dg) \sqcap (h \oplus Dh)) \ominus (g \sqcap h)$$

Linearization of Tree Expressions

for Newton's Method

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

$$D(g\,{}_{\phi}\!\oplus h) = Dg\,{}_{\phi}\!\oplus Dh$$

$$D(g \sqcap h) = ((g \oplus Dg) \sqcap (h \oplus Dh)) \ominus (g \sqcap h)$$

Carefully developed to render Newton's method sound

Linearization of Tree Expressions

for Newton's Method

- Syntactic linearization:

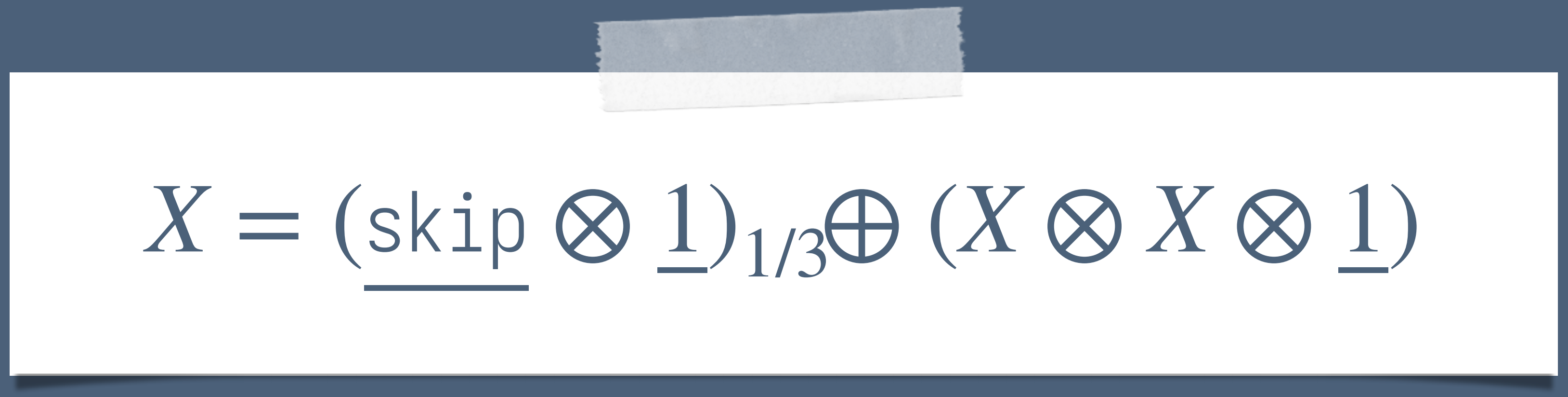
$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

$$D(g\, {}_{\phi}\!\oplus h) = Dg\, {}_{\phi}\!\oplus Dh$$

$$D(g \sqcap h) = ((g \oplus Dg) \sqcap (h \oplus Dh)) \ominus (g \sqcap h)$$

Carefully developed to render Newton's method sound



$$X = (\underline{\mathtt{skip}} \otimes \underline{1})\, {}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

Linearization of Tree Expressions

for Newton's Method

- Syntactic linearization:

$$D(g \oplus h) = Dg \oplus Dh$$

$$D(g \otimes h) = (Dg \otimes h) \oplus (g \otimes Dh)$$

$$D(g {}_{\phi}\!\oplus h) = Dg {}_{\phi}\!\oplus Dh$$

$$D(g \sqcap h) = ((g \oplus Dg) \sqcap (h \oplus Dh)) \ominus (g \sqcap h)$$

Carefully developed to render Newton's method sound

$$X = (\underline{\mathtt{skip}} \otimes \underline{1})_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

Linearization at ν

$$Y = \delta \oplus (\underline{0}_{1/3}\!\oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$

where

$$\delta = ((\underline{\mathtt{skip}} \otimes \underline{1})_{1/3}\!\oplus (\nu \otimes \nu \otimes \underline{1})) \ominus \nu$$

Linearization of Tree Expressions

for Newton's Method

$$X = (\underline{\texttt{skip}} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1}) \xrightarrow{\text{Linearization at } \nu} Y = \delta \oplus (\underline{0}_{1/3} \oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$

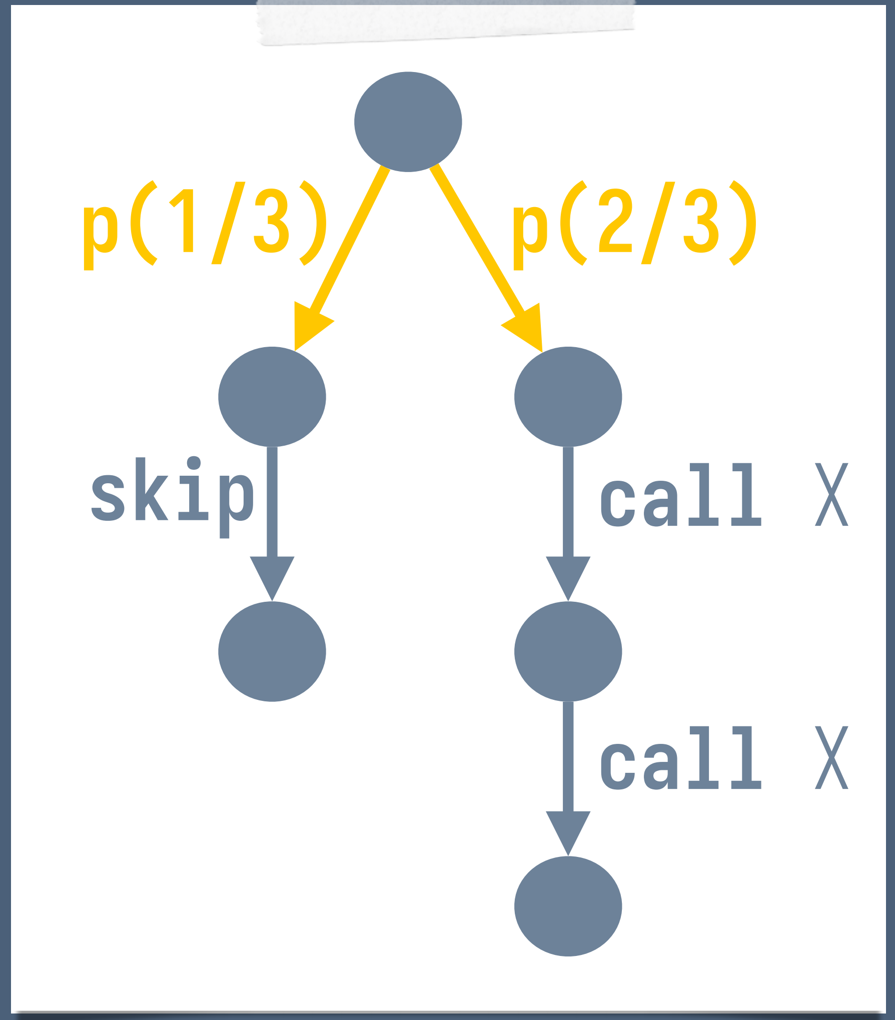
Linearization of Tree Expressions

for Newton's Method

$$X = (\underline{\mathtt{skip}} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1})$$

Linearization at ν

$$Y = \delta \oplus (\underline{0}_{1/3} \oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$



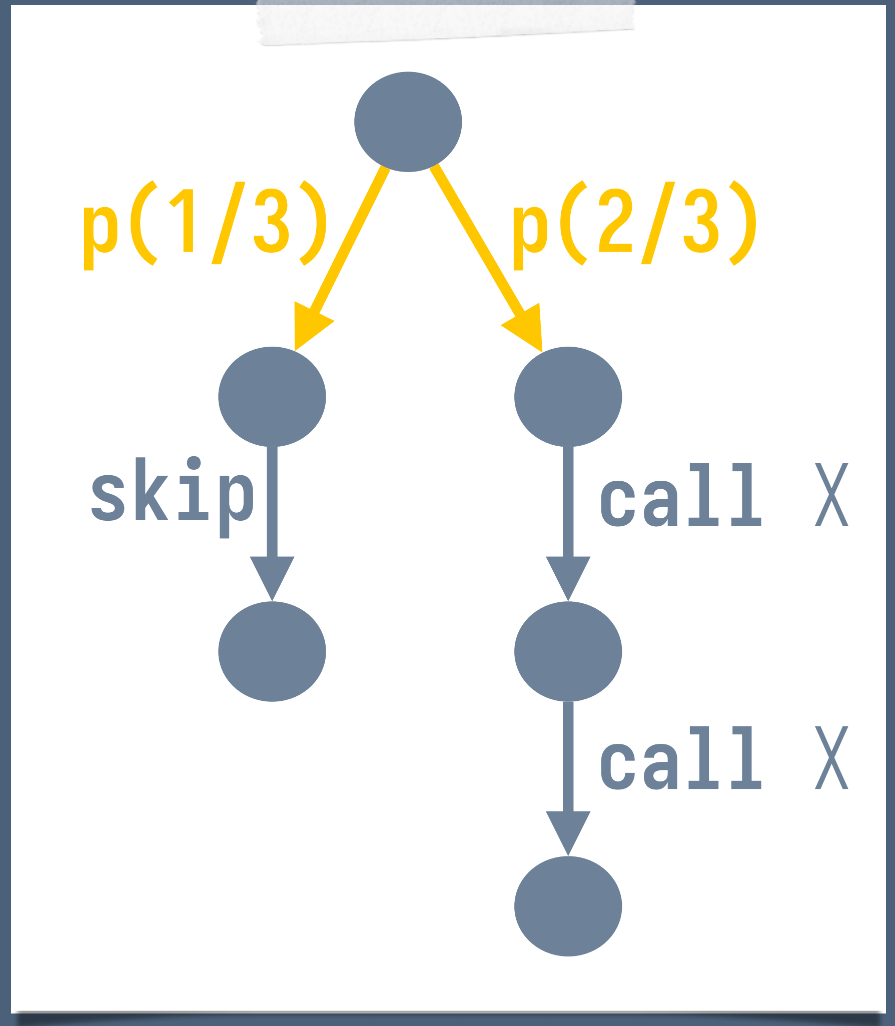
Linearization of Tree Expressions

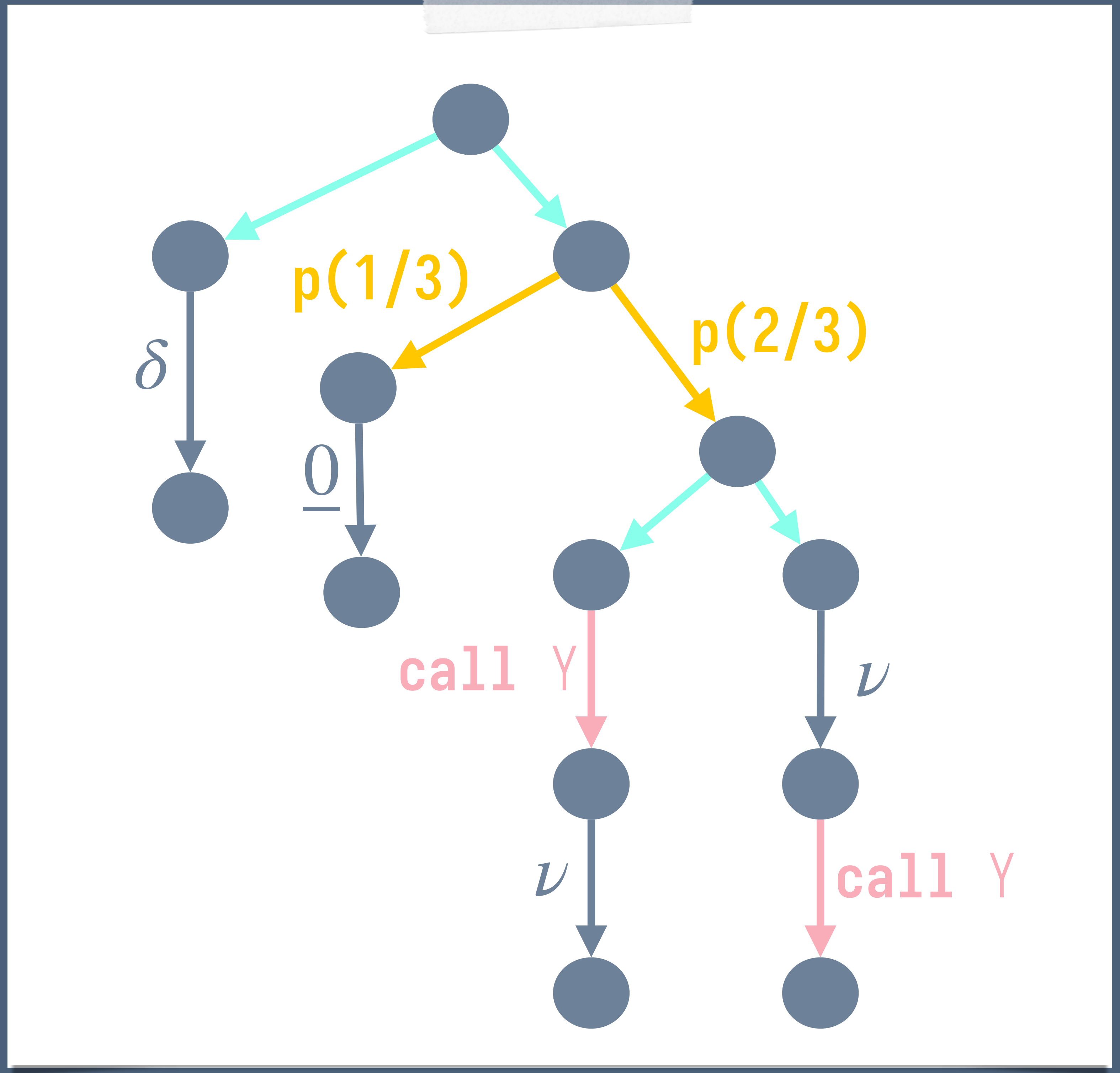
for Newton's Method

$$X = (\underline{\texttt{skip}} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1})$$

Linearization at ν

$$Y = \delta \oplus (\underline{0}_{1/3} \oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$





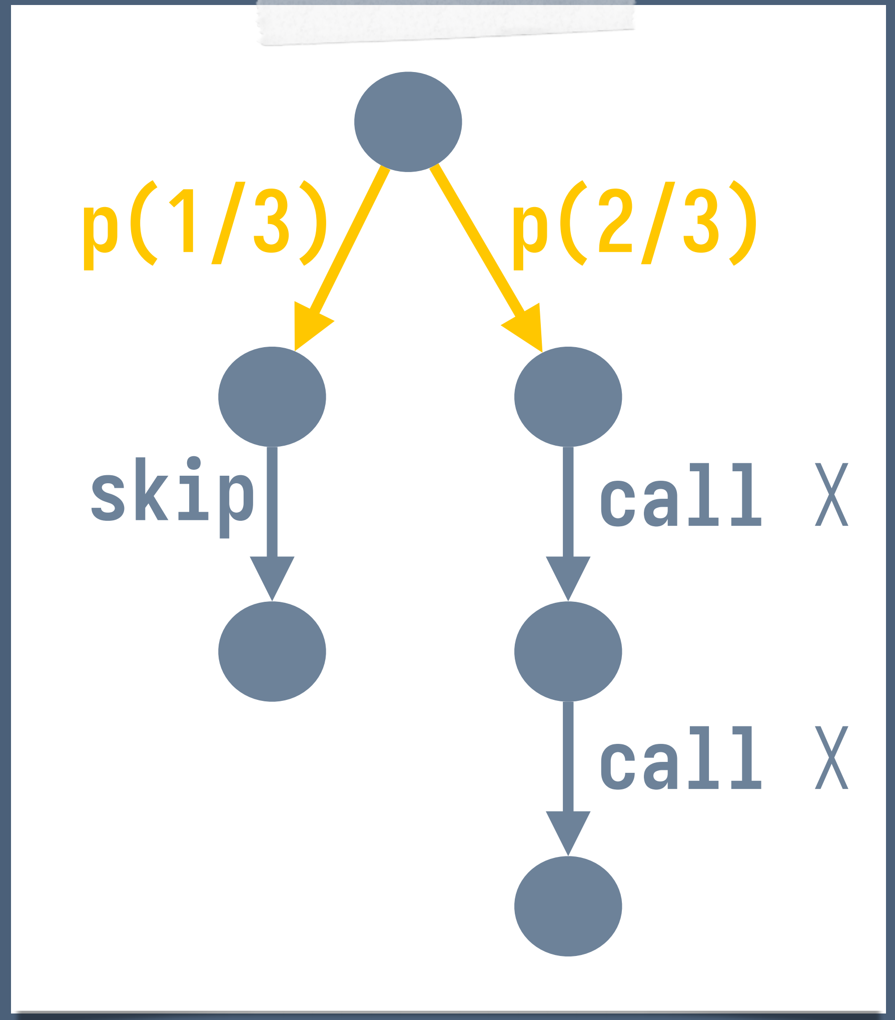
Linearization of Tree Expressions

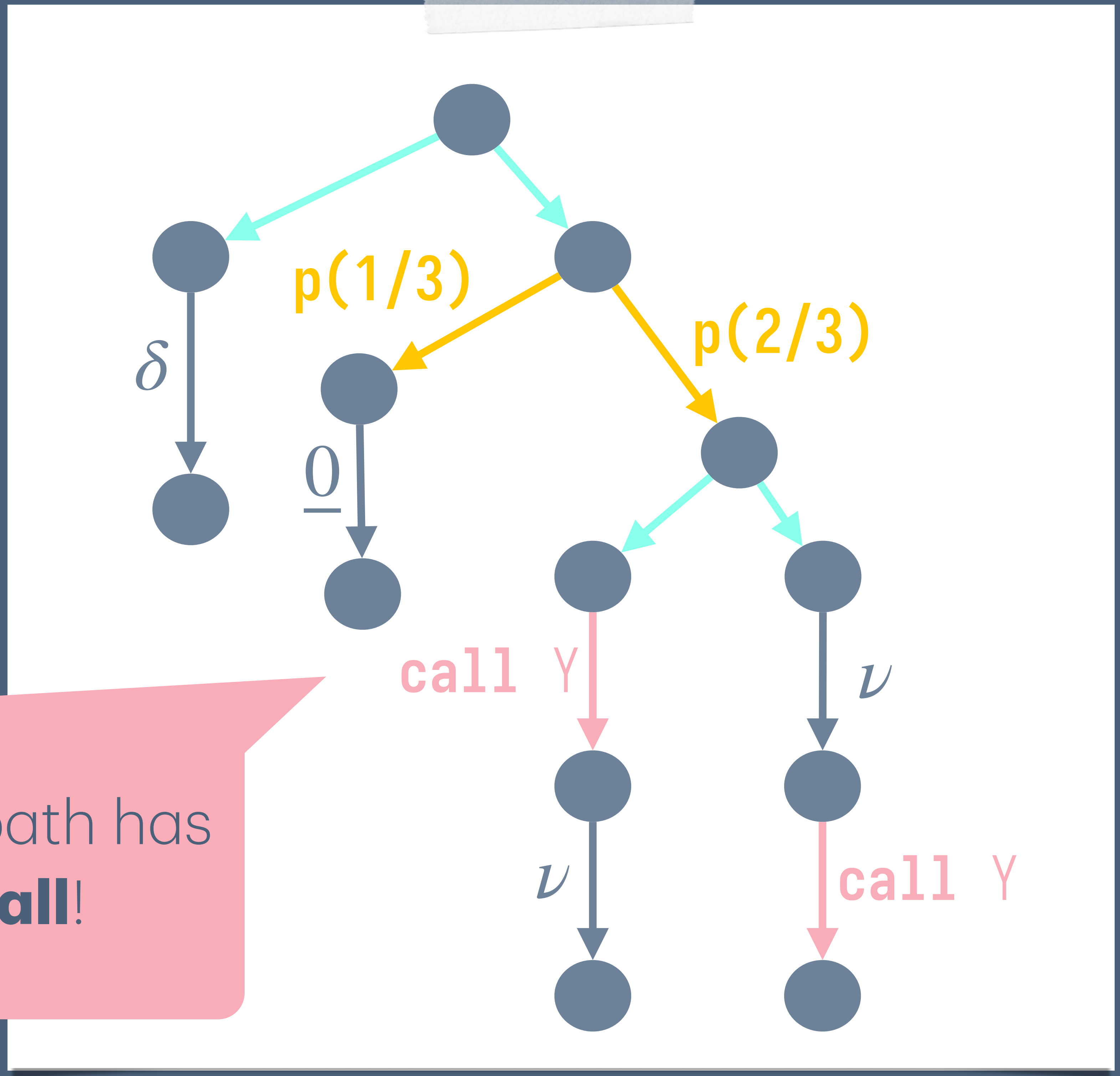
for Newton's Method

$$X = (\underline{\mathsf{skip}} \otimes \underline{1})\,{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

Linearization at ν

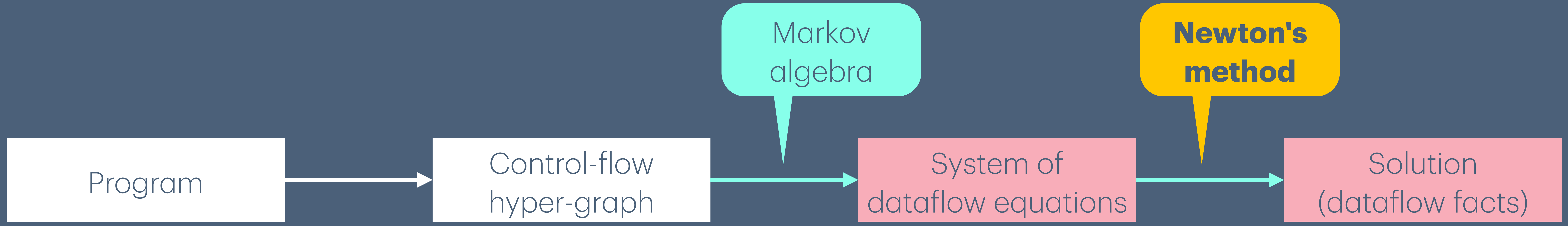
$$Y = \delta \oplus (\underline{0}\,{}_{1/3}\!\oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$





Every root-to-leaf path has **at most one call**!

Towards Multiple Combine Operations



Markov algebra

Newton's method

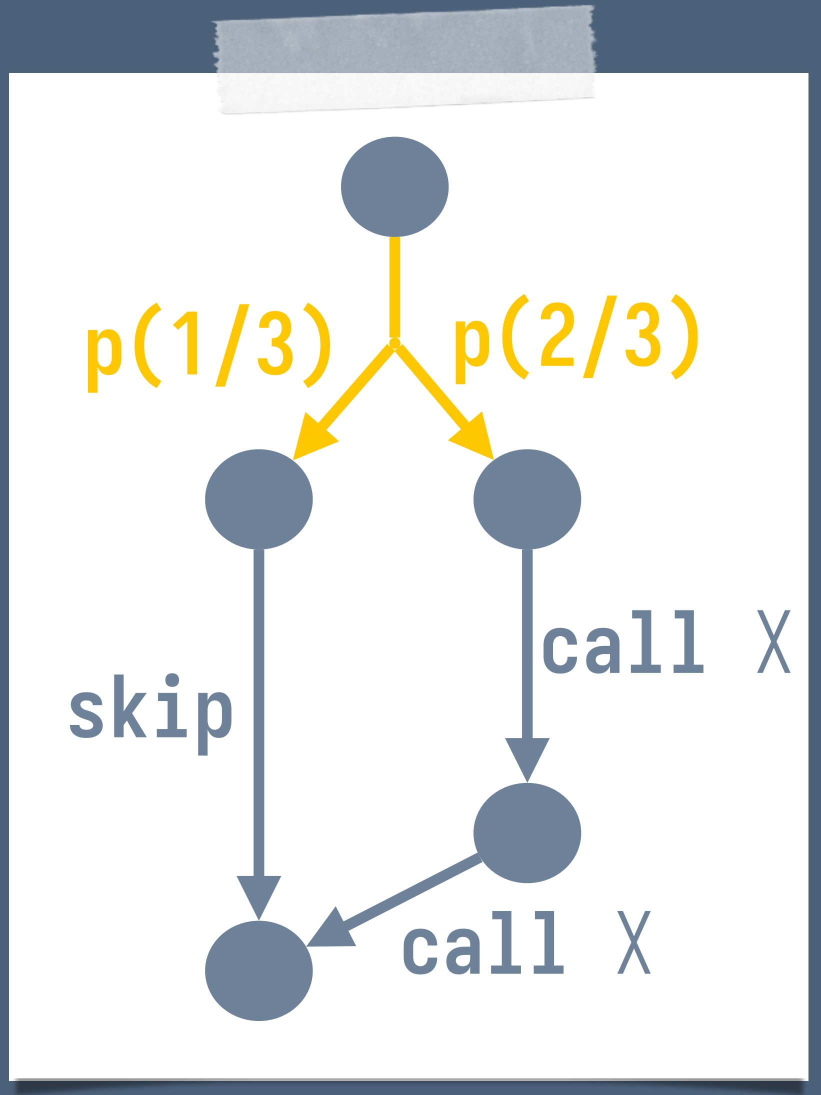
Program

Control-flow hyper-graph

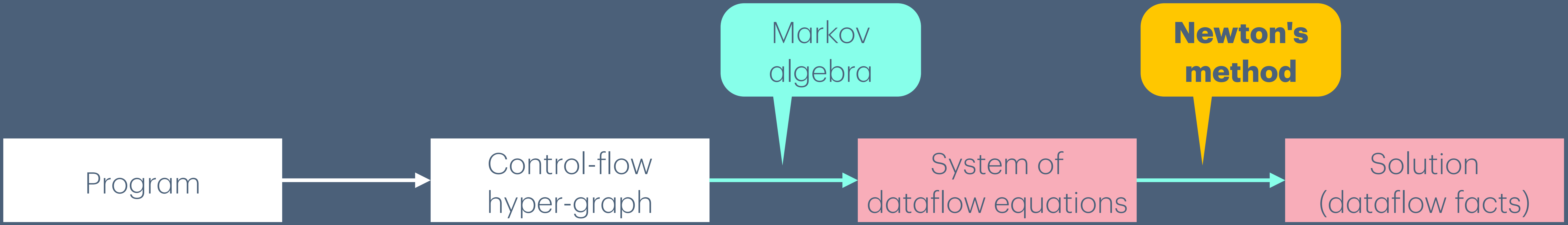
System of dataflow equations

Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



Towards Multiple Combine Operations



Markov algebra

Newton's method

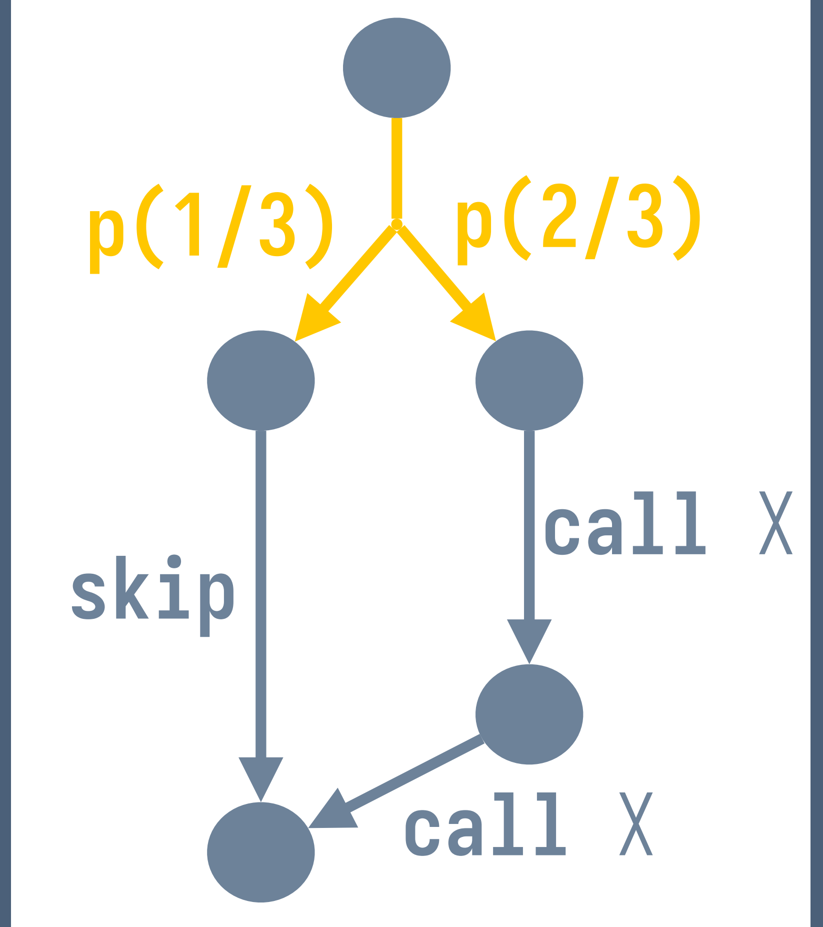
Program

Control-flow hyper-graph

System of dataflow equations

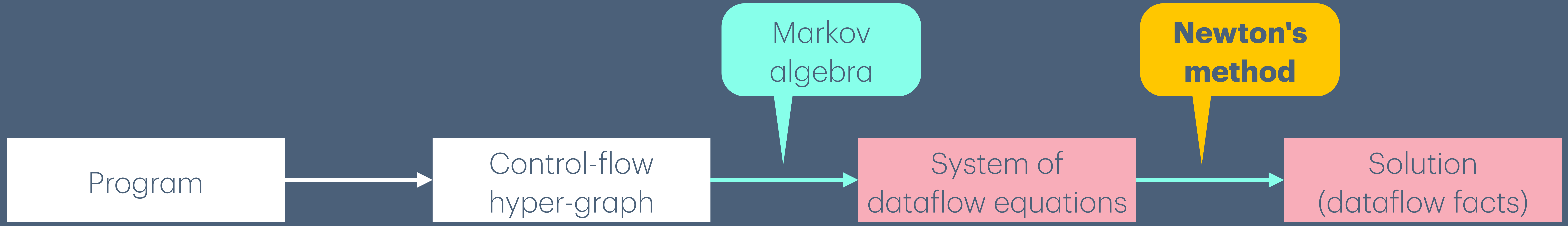
Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



For Newton's method to be efficient, we require an **analysis-supplied strategy** for solving linearized equations

Towards Multiple Combine Operations



Program

Control-flow hyper-graph

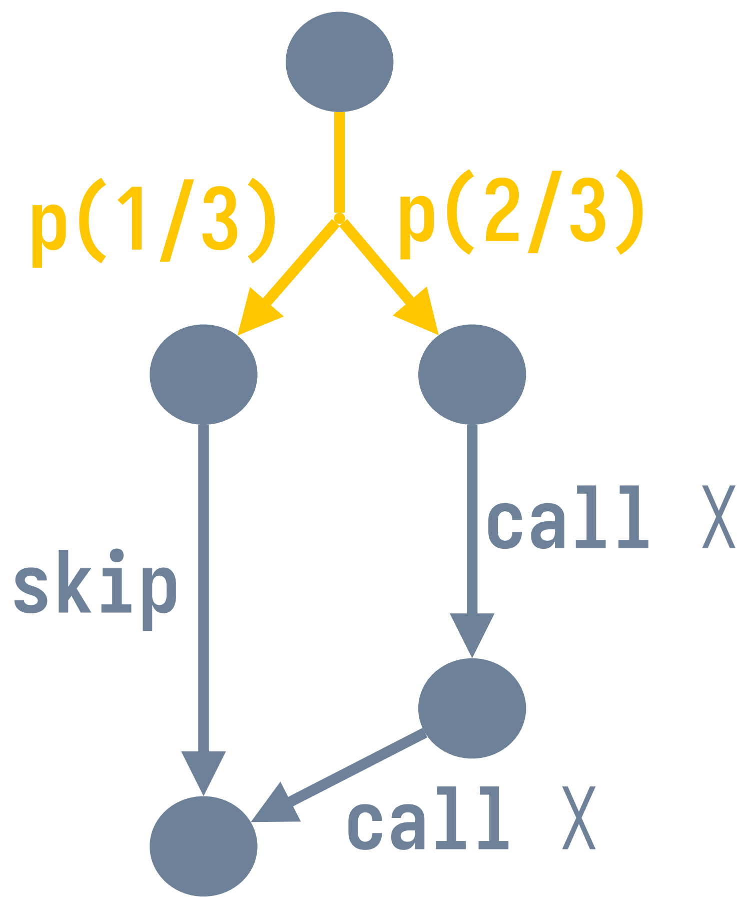
Markov algebra

System of dataflow equations

Newton's method

Solution (dataflow facts)

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



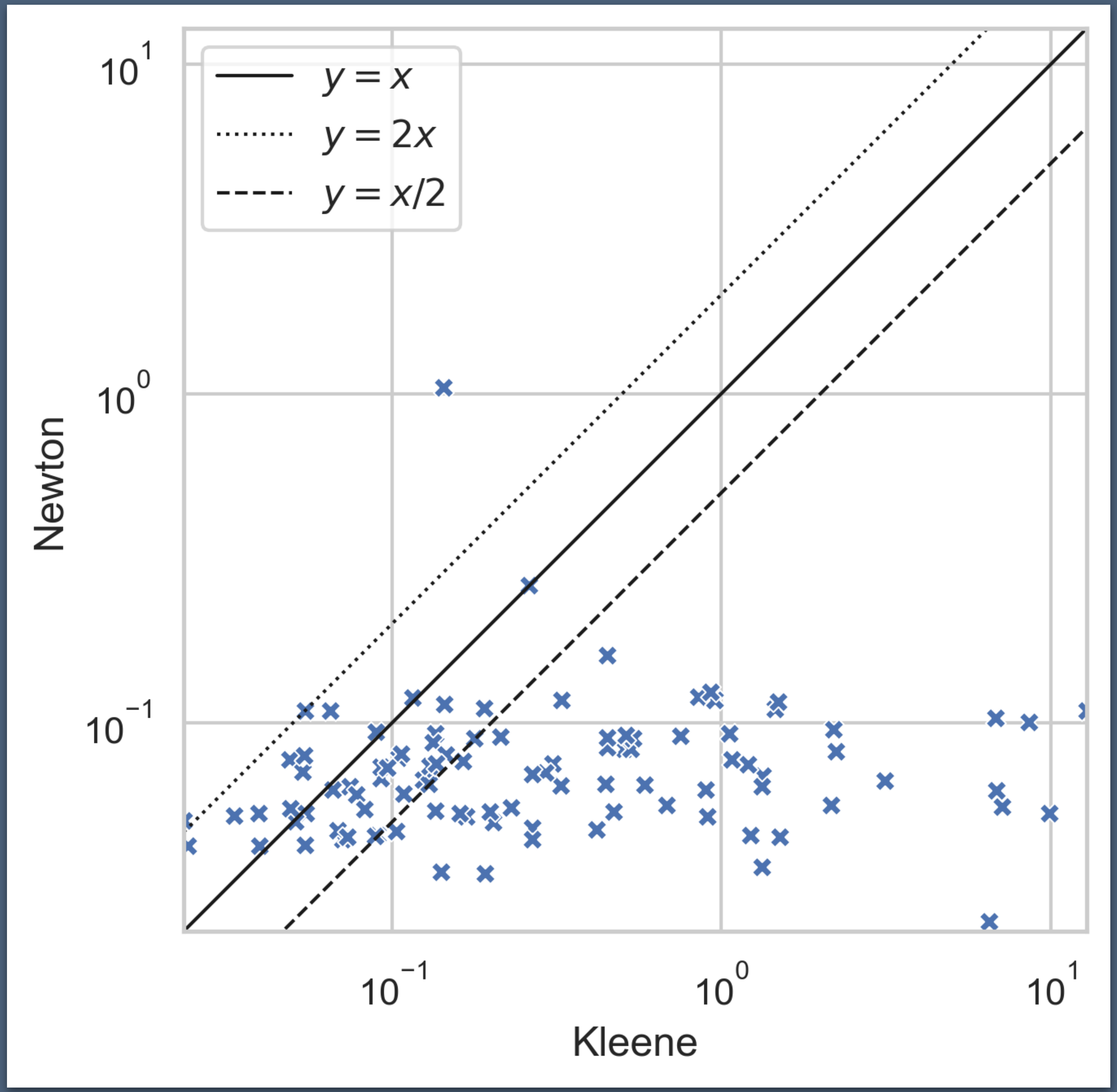
For Newton's method to be efficient, we require an **analysis-supplied strategy** for solving linearized equations

For example, termination-probability analysis:

- LP solvers
- BDD/ADD-based solvers

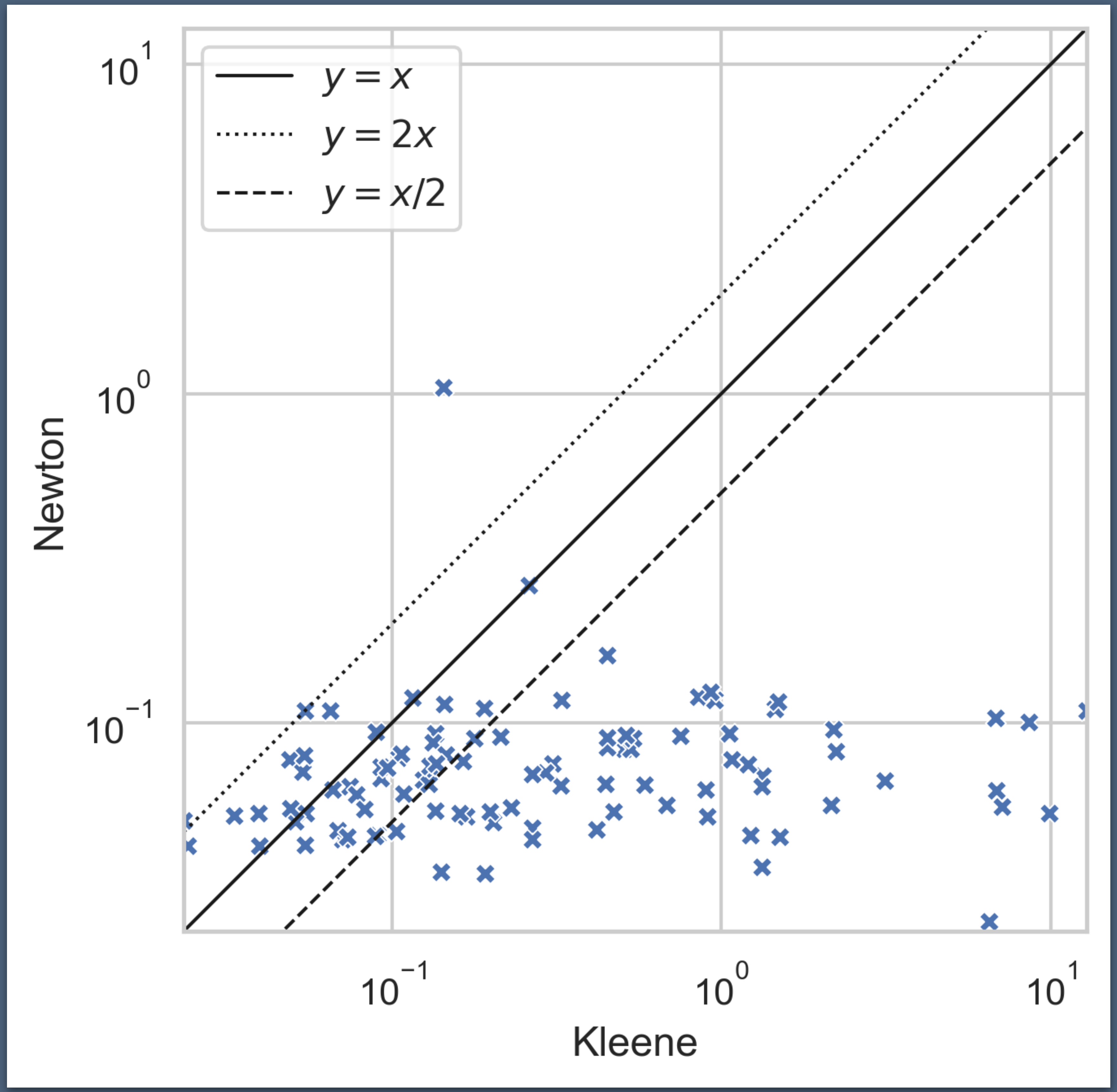
Case Studies (Selected)

Case Studies (Selected)

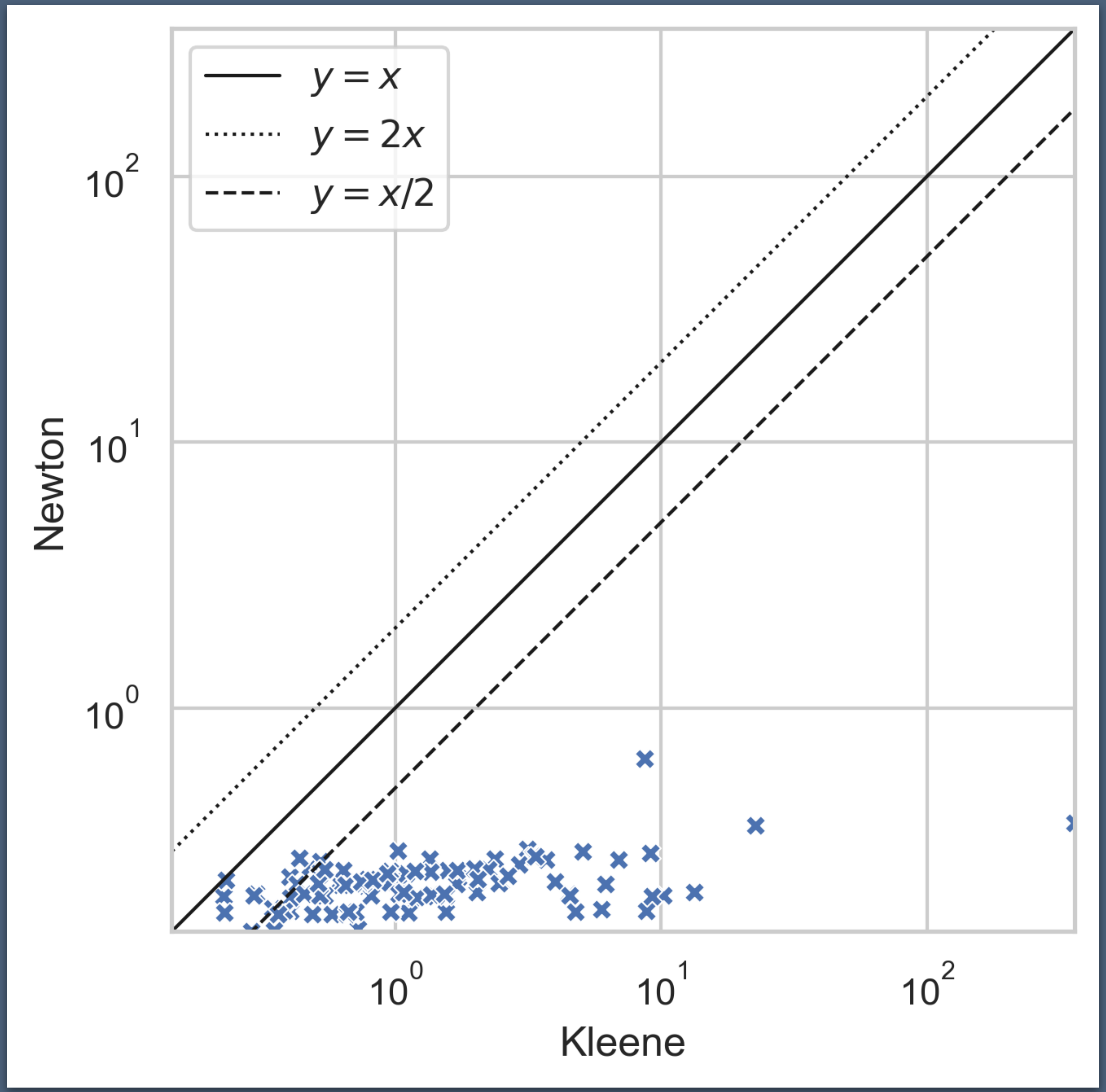


Termination-probability analysis

Case Studies (Selected)

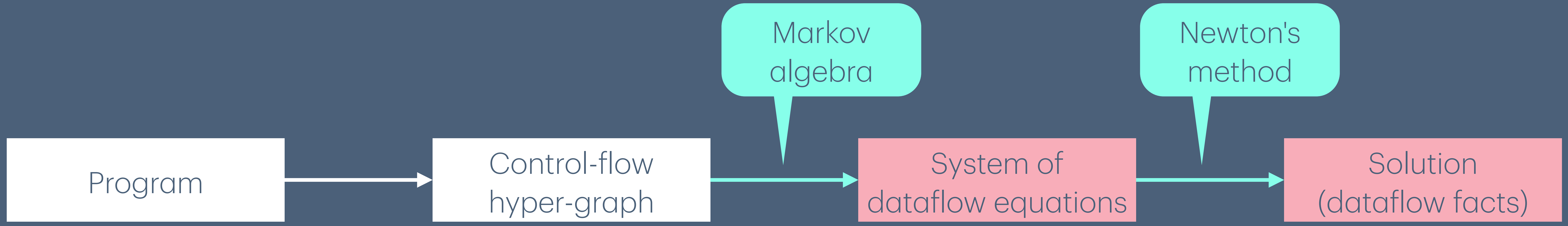


Termination-probability analysis

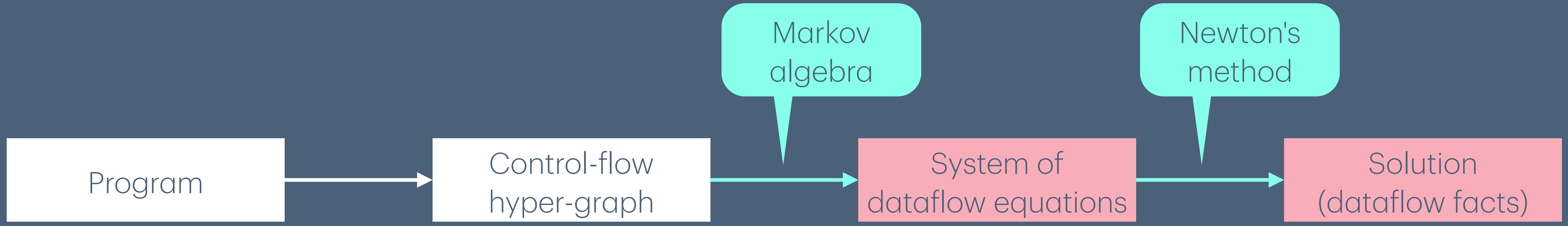


Moment-of-reward analysis

Summary

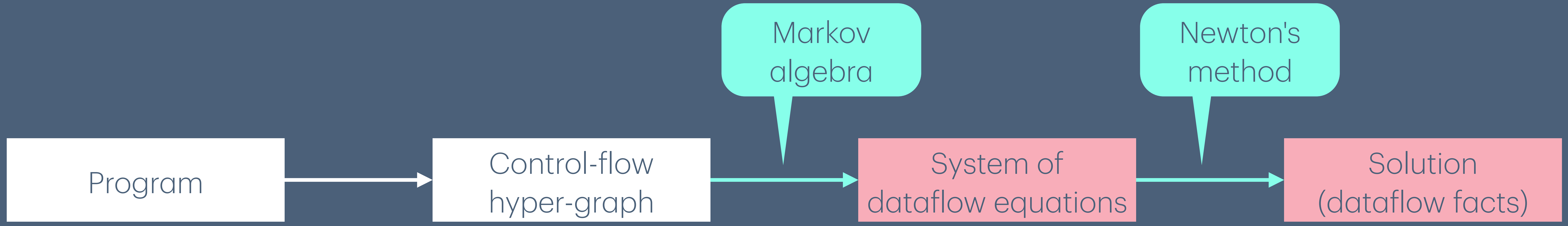

Markov algebra
Newton's method
Program
Control-flow hyper-graph
System of dataflow equations
Solution (dataflow facts)

Summary



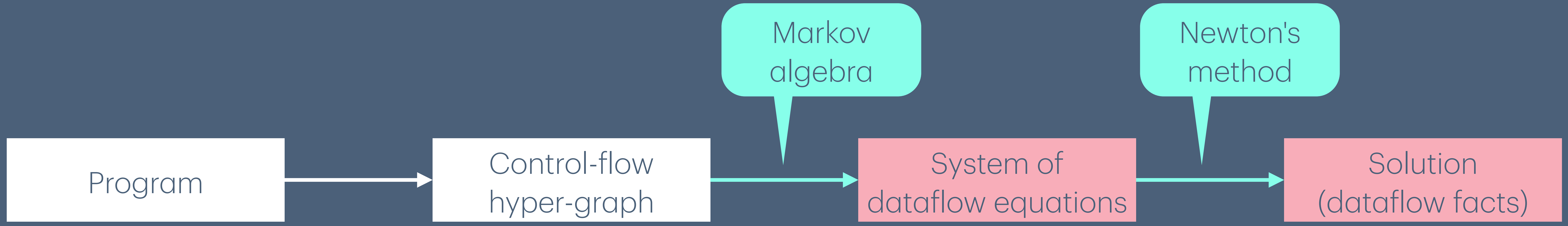
- Key takeaway: Extend Newtonian Program Analysis to support **more combine operations**

Summary



- Key takeaway: Extend Newtonian Program Analysis to support **more combine operations**
  - enabling analysis of programs with probabilistic, demonic, and conditional branching

Summary



- Key takeaway: Extend Newtonian Program Analysis to support **more combine operations**
  - enabling analysis of programs with probabilistic, demonic, and conditional branching
- More in the paper:
  - Support of loops and unstructured control-flow
  - More case studies (e.g., expectation-invariant analysis)