A programming language characterizing quantum polynomial time

Emmanuel Hainry, Romain Péchoux, Mário Silva

Université de Lorraine, Nancy, France

Model of a quantum program

- For n qubits, input and output are **unit-norm complex vectors** in the 2^n bit state-space

$$|\Psi_{\text{in}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \qquad |\Psi_{\text{out}}\rangle = \sum_{i=0}^{2^n-1} \beta_i |i\rangle \qquad \alpha_i, \beta_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = \sum_{i=0}^{2^n-1} |\beta_i|^2 = 1$$

Model of a quantum program

- For n qubits, input and output are **unit-norm complex vectors** in the 2^n bit state-space

$$|\Psi_{\text{in}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i|i\rangle \qquad |\Psi_{\text{out}}\rangle = \sum_{i=0}^{2^n-1} \beta_i|i\rangle \qquad \alpha_i, \beta_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = \sum_{i=0}^{2^n-1} |\beta_i|^2 = 1$$

- Programs are **reversible** and **norm-preserving** $\implies$ Programs encode **unitary transformations**

$$|\Psi_{\text{out}}\rangle = U|\Psi_{\text{in}}\rangle, \quad U^\dagger U = \mathbf{1}$$

Model of a quantum program

- For n qubits, input and output are **unit-norm complex vectors** in the 2^n bit state-space

$$|\Psi_{\text{in}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \qquad |\Psi_{\text{out}}\rangle = \sum_{i=0}^{2^n-1} \beta_i |i\rangle \qquad \alpha_i, \beta_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = \sum_{i=0}^{2^n-1} |\beta_i|^2 = 1$$

- Programs are **reversible** and **norm-preserving** $\Longrightarrow$ Programs encode **unitary transformations**

$$|\Psi_{\text{out}}\rangle = U|\Psi_{\text{in}}\rangle, \quad U^\dagger U = \mathbf{1}$$

- The **outcome** of the computation is the result of **measuring the output**: $\text{Probability}(i) = |\beta_i|^2$

Model of a quantum program

- For n qubits, input and output are **unit-norm complex vectors** in the 2^n bit state-space

$$|\Psi_{\text{in}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \qquad |\Psi_{\text{out}}\rangle = \sum_{i=0}^{2^n-1} \beta_i |i\rangle \qquad \alpha_i, \beta_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = \sum_{i=0}^{2^n-1} |\beta_i|^2 = 1$$

- Programs are **reversible** and **norm-preserving** $\Rightarrow$ Programs encode **unitary transformations**

$$|\Psi_{\text{out}}\rangle = U|\Psi_{\text{in}}\rangle, \quad U^\dagger U = \mathbf{1}$$

- The **outcome** of the computation is the result of **measuring the output**: $\text{Probability}(i) = |\beta_i|^2$

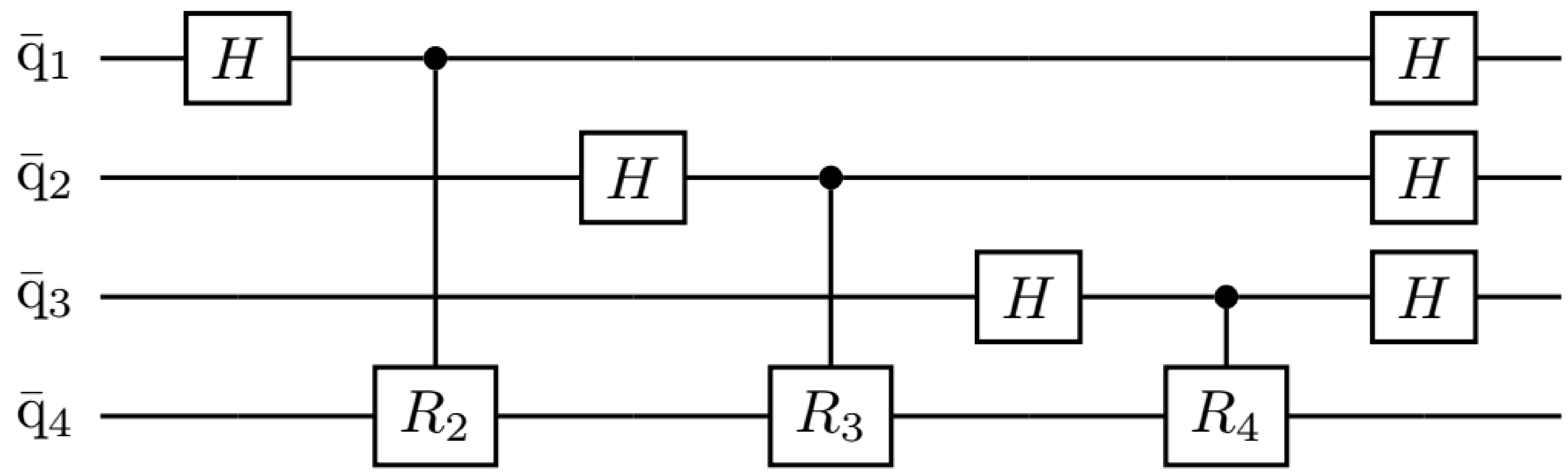
- A **function** $f : \{0,1\}^* \to \{0,1\}^*$ is **successfully approximated** by a program if

$$\forall x \in \{0,1\}^*, |\Psi_{\text{in}}\rangle = |\tilde{x}\rangle \implies \text{Probability}(\widetilde{f(x)}) \geq \frac{2}{3}$$

Motivation

set of instructions

$\textbf{decl}\ \mathrm{rec}[\mathrm{x}](\bar{\mathrm{q}})\{$
$\quad \textbf{if}\ |\bar{\mathrm{q}}| > 1\ \textbf{then}$
$\qquad \bar{\mathrm{q}}[1] \mathrel{*}= \mathrm{H};$
$\qquad \textbf{qcase}\ \bar{\mathrm{q}}[1]\ \textbf{of}\ \{$
$\qquad\quad 0 \rightarrow \textbf{skip};$
$\qquad\quad 1 \rightarrow \bar{\mathrm{q}}[|\bar{\mathrm{q}}|] \mathrel{*}= \mathrm{R}(\mathrm{x}); \}$
$\qquad \bar{\mathrm{q}}[1] \mathrel{*}= \mathrm{H};$
$\qquad \textbf{call}\ \mathrm{rec}[\mathrm{x}+1](\bar{\mathrm{q}} \ominus [1]);$
$\quad \textbf{else}\ \textbf{skip}; \},$
$::$
$\textbf{call}\ \mathrm{rec}[2](\bar{\mathrm{q}})$



quantum circuit

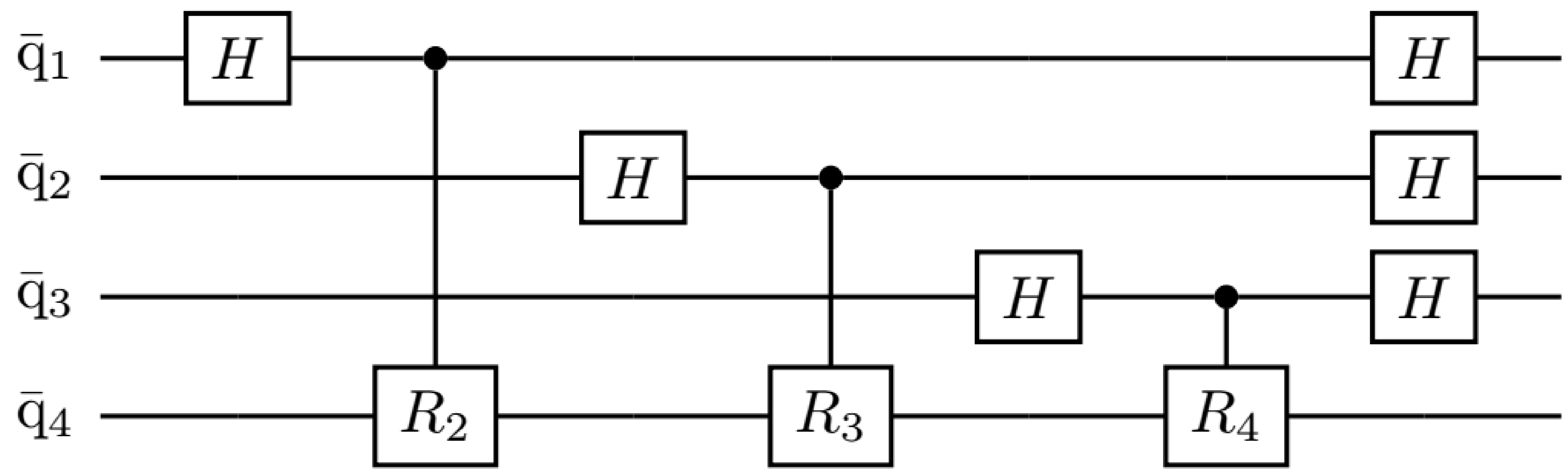
set of instructions

$$
\begin{array}{l}
\textbf{decl}\ \mathrm{rec}[\mathrm{x}](\bar{\mathrm{q}})\{\\
\quad \textbf{if}\ |\bar{\mathrm{q}}| > 1\ \textbf{then}\\
\qquad \bar{\mathrm{q}}[1]\ \texttt{*=}\ \mathrm{H};\\
\qquad \textbf{qcase}\ \bar{\mathrm{q}}[1]\ \textbf{of}\ \{\\
\qquad\quad 0 \rightarrow \textbf{skip};\\
\qquad\quad 1 \rightarrow \bar{\mathrm{q}}[|\bar{\mathrm{q}}|]\ \texttt{*=}\ \mathrm{R}(\mathrm{x});\}\\
\qquad \bar{\mathrm{q}}[1]\ \texttt{*=}\ \mathrm{H};\\
\qquad \textbf{call}\ \mathrm{rec}[\mathrm{x}+1](\bar{\mathrm{q}} \ominus [1]);\\
\quad \textbf{else}\ \textbf{skip};\},\\
::\\
\textbf{call}\ \mathrm{rec}[2](\bar{\mathrm{q}})
\end{array}
$$

Motivation

Soundness: Does the set of instructions encode a family of circuits that grows polynomially on the size of the input?

Completeness: For any such polynomial transformation, can we always find a corresponding program?



quantum circuit

Related work

Bellantoni & Cook (1992) *“A new recursion-theoretic characterization of the polytime functions”:*

- class of functions sound and complete for FP

Selinger (2004) *“Towards a quantum programming language”:*

- simple programming language with loops and recursion

Dal Lago et al. (2010) *“Quantum implicit computational complexity”:*

- quantum lambda calculus characterization of BQP

Yamakami (2020) **“**A schematic definition of quantum polynomial time computability”:

- class of functions sound and complete for FBQP

The syntax of FOQ

First-Order Quantum

$$
\begin{array}{l}
\mathbf{decl}\ \mathrm{rec}[\mathrm{x}](\bar{\mathrm{q}})\{ \\
\quad \mathbf{if}\ |\bar{\mathrm{q}}| > 1\ \mathbf{then} \\
\quad\quad \bar{\mathrm{q}}[1]\ \mathbin{{*}{=}}\ \mathrm{H}; \\
\quad\quad \mathbf{qcase}\ \bar{\mathrm{q}}[1]\ \mathbf{of}\ \{ \\
\quad\quad\quad 0 \rightarrow \mathbf{skip}; \\
\quad\quad\quad 1 \rightarrow \bar{\mathrm{q}}[|\bar{\mathrm{q}}|]\ \mathbin{{*}{=}}\ \mathrm{R}(\mathrm{x}); \} \\
\quad\quad \bar{\mathrm{q}}[1]\ \mathbin{{*}{=}}\ \mathrm{H}; \\
\quad\quad \mathbf{call}\ \mathrm{rec}[\mathrm{x}+1](\bar{\mathrm{q}} \ominus [1]); \\
\quad \mathbf{else}\ \mathbf{skip}; \}, \\
:: \\
\mathbf{call}\ \mathrm{rec}[2](\bar{\mathrm{q}})
\end{array}
$$

The syntax of FOQ

First-Order Quantum

$$
\begin{array}{l}
\mathbf{decl}\ \mathrm{rec}[\mathrm{x}](\bar{\mathrm{q}})\{\\
\quad \mathbf{if}\ |\bar{\mathrm{q}}| > 1\ \mathbf{then}\\
\qquad \bar{\mathrm{q}}[1]\ \mathrel{*}=\ \mathrm{H};\\
\qquad \mathbf{qcase}\ \bar{\mathrm{q}}[1]\ \mathbf{of}\ \{\\
\qquad\quad 0 \to \mathbf{skip};\\
\qquad\quad 1 \to \bar{\mathrm{q}}[|\bar{\mathrm{q}}|]\ \mathrel{*}=\ \mathrm{R}(\mathrm{x});\}\\
\qquad \bar{\mathrm{q}}[1]\ \mathrel{*}=\ \mathrm{H};\\
\qquad \mathbf{call}\ \mathrm{rec}[\mathrm{x}+1](\bar{\mathrm{q}} \ominus [1]);\\
\quad \mathbf{else}\ \mathbf{skip};\},\\
::\\
\mathbf{call}\ \mathrm{rec}[2](\bar{\mathrm{q}})
\end{array}
$$

procedure declarations

program body

The syntax of FOQ

First-Order Quantum

$$
\begin{array}{l}
\mathbf{decl}\ \mathrm{rec}[x](\bar{q})\{\\
\quad \mathbf{if}\ |\bar{q}| > 1\ \mathbf{then}\\
\qquad \bar{q}[1]\ \texttt{*=}\ \mathrm{H};\\
\qquad \mathbf{qcase}\ \bar{q}[1]\ \mathbf{of}\ \{\\
\qquad\quad 0 \to \mathbf{skip};\\
\qquad\quad 1 \to \bar{q}[|\bar{q}|]\ \texttt{*=}\ \mathrm{R}(x);\}\\
\qquad \bar{q}[1]\ \texttt{*=}\ \mathrm{H};\\
\qquad \mathbf{call}\ \mathrm{rec}[x+1](\bar{q} \ominus [1]);\\
\quad \mathbf{else}\ \mathbf{skip};\},\\
::\\
\mathbf{call}\ \mathrm{rec}[2](\bar{q})
\end{array}
$$

procedure declarations

« **decl** proc[integer input](quantum input){S} »

quantum control

« **qcase** cqubit **of** {0 → S0, 1 → S1} »

- branches S0 and S1 cannot affect cqubit

(recursive) procedure call

« **call** proc[integer](qubits) »

(some) Denotational Semantics

$$\frac{}{(\textbf{skip}, |\psi\rangle, A, l) \xrightarrow{0} (\top, |\psi\rangle, A, l)} \ (\text{Skip})$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \notin A}{(\mathrm{s[i]} \ \texttt{*=} \ \mathrm{U}^f(\mathrm{j});, |\psi\rangle, A, l) \xrightarrow{0} (\bot, |\psi\rangle, A, l)} \ (\text{Asg}_\bot)$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{U}^f(\mathrm{j}), l) \Downarrow_{\mathbb{C}^{2\times 2}} M}{(\mathrm{s[i]} \ \texttt{*=} \ \mathrm{U}^f(\mathrm{j});, |\psi\rangle, A, l) \xrightarrow{0} (\top, I_{2^{n-1}} \otimes M \otimes I_{2^{l(|\psi\rangle)-n}} |\psi\rangle, A, l)} \ (\text{Asg}_\top)$$

$$\frac{(\mathrm{S}_1, |\psi\rangle, A, l) \xrightarrow{m_1} (\top, |\psi'\rangle, A, l) \qquad (\mathrm{S}_2, |\psi'\rangle, A, l) \xrightarrow{m_2} (\diamond, |\psi''\rangle, A, l)}{(\mathrm{S}_1 \ \mathrm{S}_2, |\psi\rangle, A, l) \xrightarrow{m_1+m_2} (\diamond, |\psi''\rangle, A, l)} \ (\text{Seq}_\diamond)$$

(some) Denotational Semantics

$$\frac{(\mathrm{b}, l) \Downarrow_{\mathbb{B}} b \in \mathbb{B} \qquad (\mathrm{S}_b, |\psi\rangle, A, l) \xrightarrow{m_b} (\diamond, |\psi'\rangle, A, l)}{(\textbf{if } \mathrm{b} \textbf{ then } \mathrm{S}_{\textbf{true}} \textbf{ else } \mathrm{S}_{\textbf{false}}, |\psi\rangle, A, l) \xrightarrow{m_b} (\diamond, |\psi'\rangle, A, l)} \text{(If)}$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{S}_k, |\psi\rangle, A \backslash \{n\}, l) \xrightarrow{m_k} (\top, |\psi_k\rangle, A \backslash \{n\}, l)}{(\textbf{qcase } \mathrm{s[i]} \textbf{ of } \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}, |\psi\rangle, A, l) \xrightarrow{\max_k m_k} (\top, \sum_k |k\rangle_n \langle k|_n |\psi_k\rangle, A, l)} (\mathrm{Case}_\top)$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{S}_k, |\psi\rangle, A \backslash \{n\}, l) \xrightarrow{m_k} (\diamond_k, |\psi_k\rangle, A \backslash \{n\}, l) \qquad \bot \in \{\diamond_0, \diamond_1\}}{(\textbf{qcase } \mathrm{s[i]} \textbf{ of } \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}, |\psi\rangle, A, l) \xrightarrow{\max_k m_k} (\bot, |\psi\rangle, A, l)} (\mathrm{Case}_\bot)$$

The width of a procedure

$$
\begin{aligned}
&\mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}^{\mathrm{proc}}),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathbf{skip};\,) \triangleq 0,\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{q}\ \boldsymbol{*}\!\!=\ \mathrm{U}^{f}(\mathrm{i});\,) \triangleq 0,\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1\ \ \mathrm{S}_2) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) + w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_2),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathbf{if}\ \ \mathrm{b}\ \ \mathbf{then}\ \ \mathrm{S}_{\mathbf{true}}\ \ \mathbf{else}\ \ \mathrm{S}_{\mathbf{false}}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\mathbf{true}}), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\mathbf{false}})),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathbf{qcase}\ \ \mathrm{q}\ \ \mathbf{of}\ \ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1)),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathbf{call}\ \ \mathrm{proc}'[\mathrm{i}](\mathrm{s});\,) \triangleq \begin{cases} 1 & \textit{if}\ \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}',\\ 0 & \textit{otherwise}. \end{cases}
\end{aligned}
$$

Restrictions on recursion

$\mathbf{WF}$ programs (well-founded)

- all mutually recursive calls decrease the number of qubits

  → ensured termination

$\mathbf{PFOQ}$ programs (polynomial time)

- all mutually recursive calls decrease the number of qubits
- at most one mutually recursive call per (quantum) branch

  → ensured termination in polynomial time

Restrictions on recursion

WF programs (well-founded) → ensured termination

$$\begin{aligned}&\forall \mathrm{proc} \in \mathrm{P},\\ &\qquad \forall \mathbf{call}\ \mathrm{proc}'[\mathrm{i}](s); \in \mathrm{S}^{\mathrm{proc}},\\ &\qquad\qquad \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}' \Rightarrow s = \bar{\mathrm{p}} \ominus [\mathrm{i}_1, \dots, \mathrm{i}_k]\end{aligned}$$

PFOQ programs (polynomial time) → ensured poly-time termination

$$\mathrm{P} \in \textsc{wf} \text{ and } \forall \mathrm{proc} \in \mathrm{P},\ \ \mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \leq 1$$

Restrictions on recursion

$$
\begin{array}{l}
\textbf{decl } \mathrm{rec}(\bar{\mathrm{q}})\{\\
\quad \bar{\mathrm{q}}[1] \mathrel{*=} \mathrm{H};\\
\quad \textbf{call } \mathrm{rot}[2](\bar{\mathrm{q}});\\
\quad \textbf{call } \mathrm{rec}(\bar{\mathrm{q}} \ominus [1]);\},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl } \mathrm{rot}[\mathrm{x}](\bar{\mathrm{q}})\{\\
\ \textbf{if } |\bar{\mathrm{q}}| > 1 \textbf{ then}\\
\quad \textbf{qcase } \ \bar{\mathrm{q}}[2] \textbf{ of } \{\\
\qquad 0 \to \textbf{skip};\\
\qquad 1 \to \bar{\mathrm{q}}[1] \mathrel{*=} \mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(\mathrm{x});\\
\quad \}\\
\quad \textbf{call } \mathrm{rot}[\mathrm{x}+1](\bar{\mathrm{q}} \ominus [2]);\\
\ \textbf{else skip};\},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl } \mathrm{inv}(\bar{\mathrm{q}})\{\\
\quad \textbf{if } |\bar{\mathrm{q}}| > 1 \textbf{ then}\\
\qquad \mathrm{SWAP}(\bar{\mathrm{q}}[1], \bar{\mathrm{q}}[|\bar{\mathrm{q}}|]);\\
\qquad \textbf{call } \mathrm{inv}(\bar{\mathrm{q}} \ominus [1, |\bar{\mathrm{q}}|]);\\
\quad \textbf{else skip};\} ::
\end{array}
$$

$$
\textbf{call } \mathrm{rec}(\bar{\mathrm{q}});\ \textbf{call } \mathrm{inv}(\bar{\mathrm{q}});
$$

Restrictions on recursion

$$
\begin{array}{l}
\textbf{decl } \mathrm{rec}(\bar{q})\{\\
\quad \bar{q}[1] \mathrel{*}= H;\\
\quad \textbf{call } \mathrm{rot}[2](\bar{q});\\
\quad \textbf{call } \mathrm{rec}(\bar{q} \ominus [1]);\},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl } \mathrm{rot}[x](\bar{q})\{\\
\ \textbf{if } |\bar{q}| > 1 \textbf{ then}\\
\quad \textbf{qcase } \ \bar{q}[2] \textbf{ of } \{\\
\quad\quad 0 \to \textbf{skip};\\
\quad\quad 1 \to \bar{q}[1] \mathrel{*}= \mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(x);\\
\quad \}\\
\quad \textbf{call } \mathrm{rot}[x+1](\bar{q} \ominus [2]);\\
\ \textbf{else skip};\},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl } \mathrm{inv}(\bar{q})\{\\
\ \textbf{if } |\bar{q}| > 1 \textbf{ then}\\
\quad \mathrm{SWAP}(\bar{q}[1], \bar{q}[|\bar{q}|]);\\
\quad \textbf{call } \mathrm{inv}(\bar{q} \ominus [1, |\bar{q}|]);\\
\ \textbf{else skip};\} ::
\end{array}
$$

$$\textbf{call } \mathrm{rec}(\bar{q}); \textbf{ call } \mathrm{inv}(\bar{q});$$

$$\mathrm{QFT} \in \mathrm{WF}$$

Restrictions on recursion

$$
\begin{array}{l}
\textbf{decl}\ \mathrm{rec}(\bar{q})\{\\
\quad \boxed{\begin{array}{l}\bar{q}[1] \mathrel{*}= H;\\ \textbf{call}\ \mathrm{rot}[2](\bar{q});\\ \textbf{call}\ \mathrm{rec}(\bar{q}\ominus[1]);\end{array}}\ \},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl}\ \mathrm{rot}[x](\bar{q})\{\\
\boxed{\begin{array}{l}\textbf{if}\ |\bar{q}|>1\ \textbf{then}\\
\quad \textbf{qcase}\ \ \bar{q}[2]\ \textbf{of}\ \{\\
\qquad 0 \to \textbf{skip};\\
\qquad 1 \to \bar{q}[1] \mathrel{*}= \mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(x);\\
\quad \}\\
\quad \textbf{call}\ \mathrm{rot}[x+1](\bar{q}\ominus[2]);\end{array}}\\
\textbf{else skip};\},
\end{array}
$$

$$
\begin{array}{l}
\textbf{decl}\ \mathrm{inv}(\bar{q})\{\\
\boxed{\begin{array}{l}\textbf{if}\ |\bar{q}|>1\ \textbf{then}\\
\quad \mathrm{SWAP}(\bar{q}[1],\bar{q}[|\bar{q}|]);\\
\quad \textbf{call}\ \mathrm{inv}(\bar{q}\ominus[1,|\bar{q}|]);\end{array}}\\
\textbf{else skip};\}::
\end{array}
$$

$$\textbf{call}\ \mathrm{rec}(\bar{q});\ \textbf{call}\ \mathrm{inv}(\bar{q});$$

$$\mathrm{QFT} \in \textsc{wf}$$

$$\mathrm{QFT} \in \textsc{pfoq}$$

Results

- All terminating programs (in particular **WF** programs) have an inverse program in **FOQ**.

PFOQ ~ FBQP

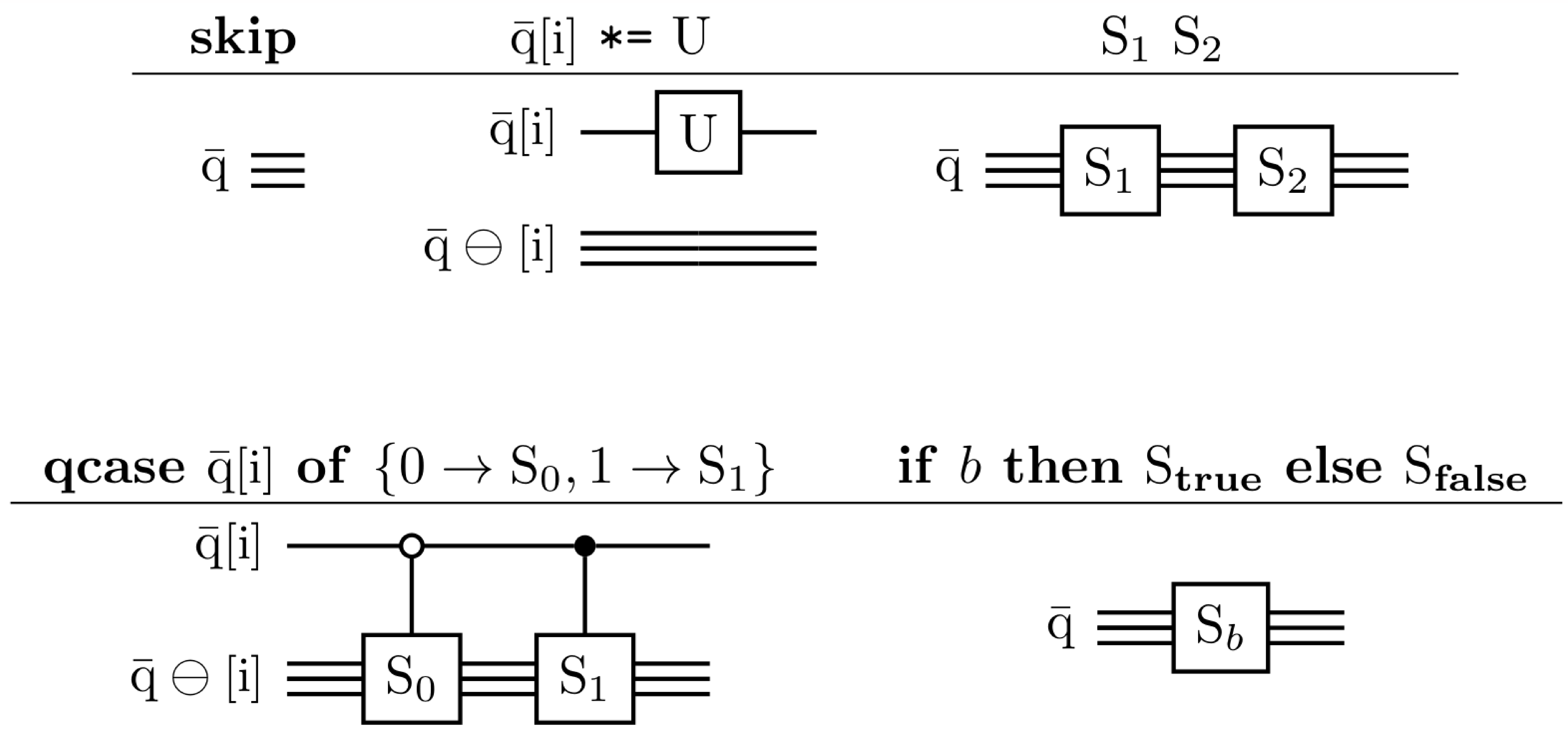
Soundness. If a PFOQ program successfully approximates some function f , then f is in FBQP. (*proof: simulation by a poly-time quantum Turing machine.)*

Completeness. For any function f in FBQP, there exists a PFOQ program that successfully approximates f. *(proof: simulation of Yamakami's function algebra.)*

PFOQ programs correspond to uniform families of poly-sized circuits

$$\{\text{Programs}, \mathbb{N}\} \longmapsto^{\text{compile}} \text{Circuits} \quad \text{where} \quad |\text{compile}(\text{P}, n)| \in O(\text{poly}(n))$$

Circuit compilation



Building a poly-sized circuit

PFOQ program

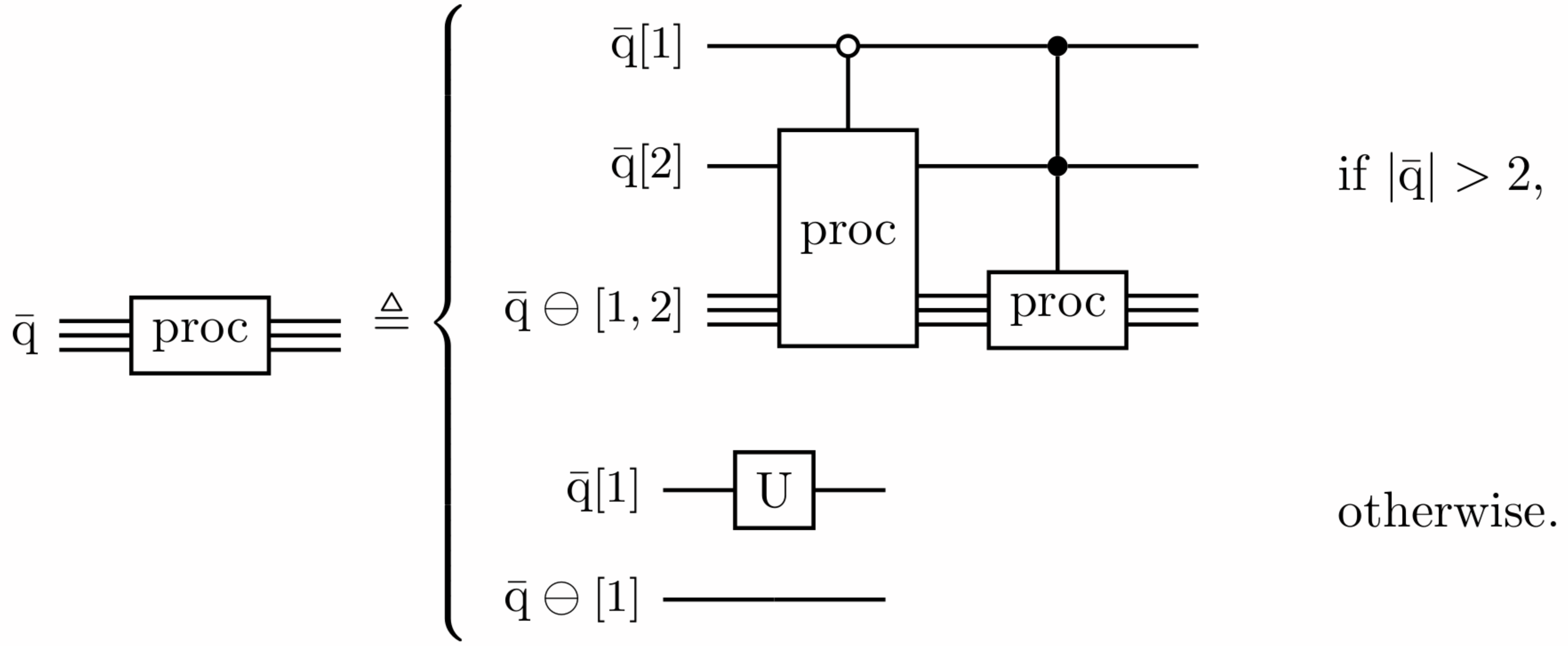
$$
\begin{array}{l}
\textbf{decl}\ \mathrm{proc}(\bar{\mathrm{q}})\{\\
\quad \textbf{if}\ |\bar{\mathrm{q}}| > 2 :\\
\quad \textbf{qcase}\ \bar{\mathrm{q}}[1]\ \textbf{of}\\
\quad \{0 \to \textbf{call}\ \mathrm{proc}(\bar{\mathrm{q}} \ominus [1]);\\
\quad\ \ 1 \to \textbf{qcase}\ \bar{\mathrm{q}}[2]\ \textbf{of}\\
\qquad\qquad \{0 \to \textbf{skip};,\\
\qquad\qquad\ 1 \to \textbf{call}\ \mathrm{proc}(\bar{\mathrm{q}} \ominus [1,2]);\}\}\\
\quad \textbf{else}\ \bar{\mathrm{q}}[1]\ \mathbf{*=}\ \mathrm{U};\}\\
::\ \textbf{call}\ \mathrm{proc}(\bar{\mathrm{q}});
\end{array}
$$

Building a poly-sized circuit

PFOQ program

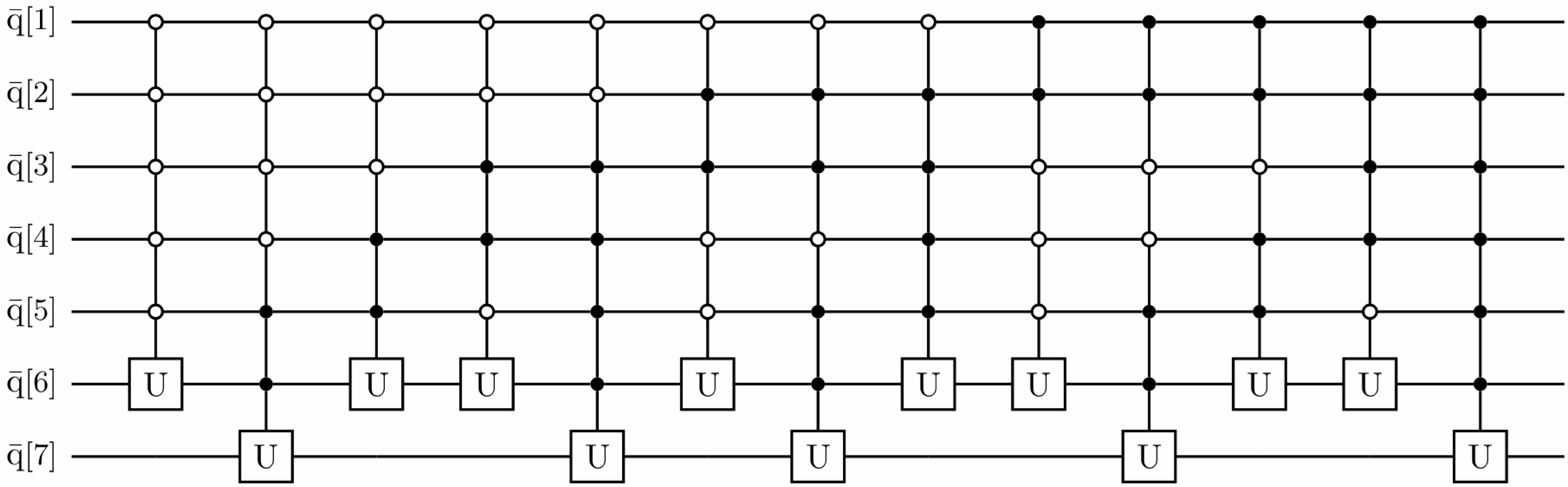
$$
\begin{array}{l}
\mathbf{decl}\ \mathrm{proc}(\bar{q})\{ \\
\quad \mathbf{if}\ |\bar{q}| > 2 : \\
\quad \mathbf{qcase}\ \bar{q}[1]\ \mathbf{of} \\
\quad \{0 \to \mathbf{call}\ \mathrm{proc}(\bar{q} \ominus [1]); \\
\quad\ \ 1 \to \mathbf{qcase}\ \bar{q}[2]\ \mathbf{of} \\
\qquad\qquad \{0 \to \mathbf{skip};, \\
\qquad\qquad\ \ 1 \to \mathbf{call}\ \mathrm{proc}(\bar{q} \ominus [1,2]);\}\} \\
\quad \mathbf{else}\ \bar{q}[1]\ \mathbf{*=}\ \mathrm{U};\} \\
::\ \mathbf{call}\ \mathrm{proc}(\bar{q});
\end{array}
$$

Possible compilation strategy



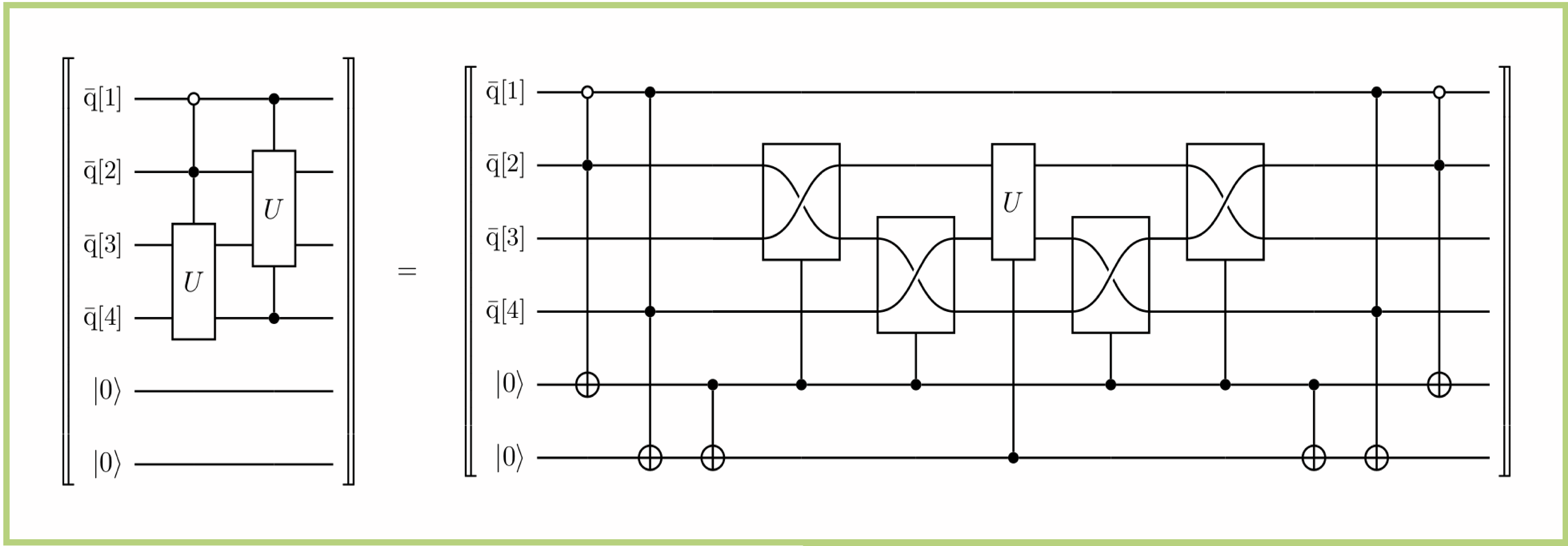
Building a poly-sized circuit

$n = 7$ grows in $O(n2^n)$

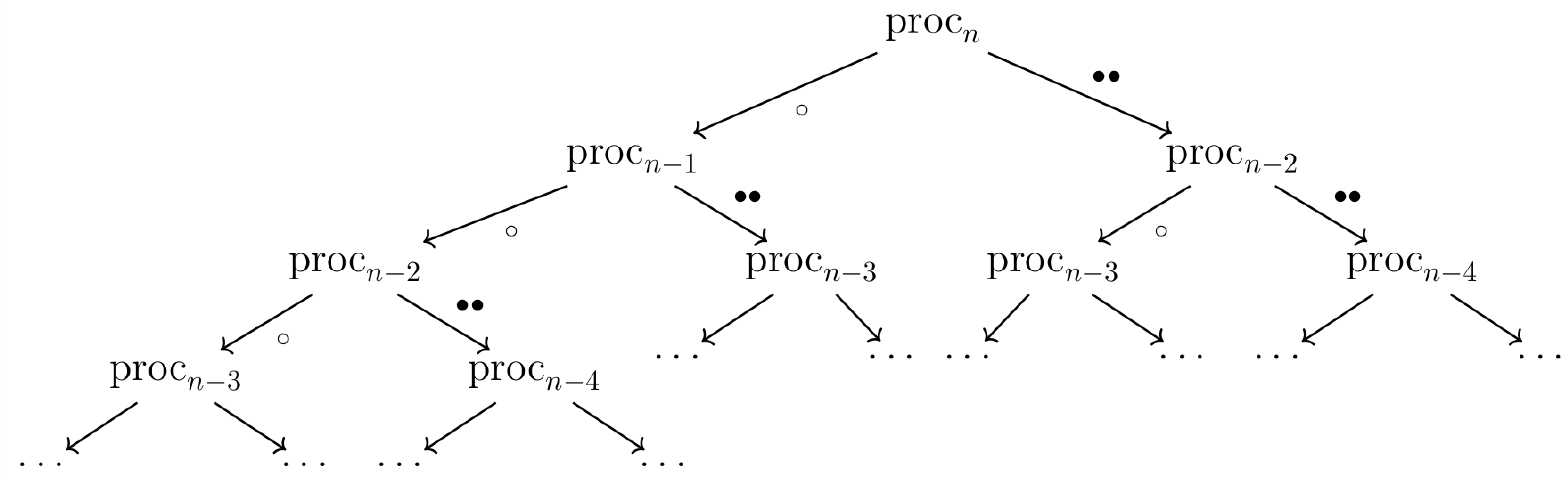


Building a poly-sized circuit

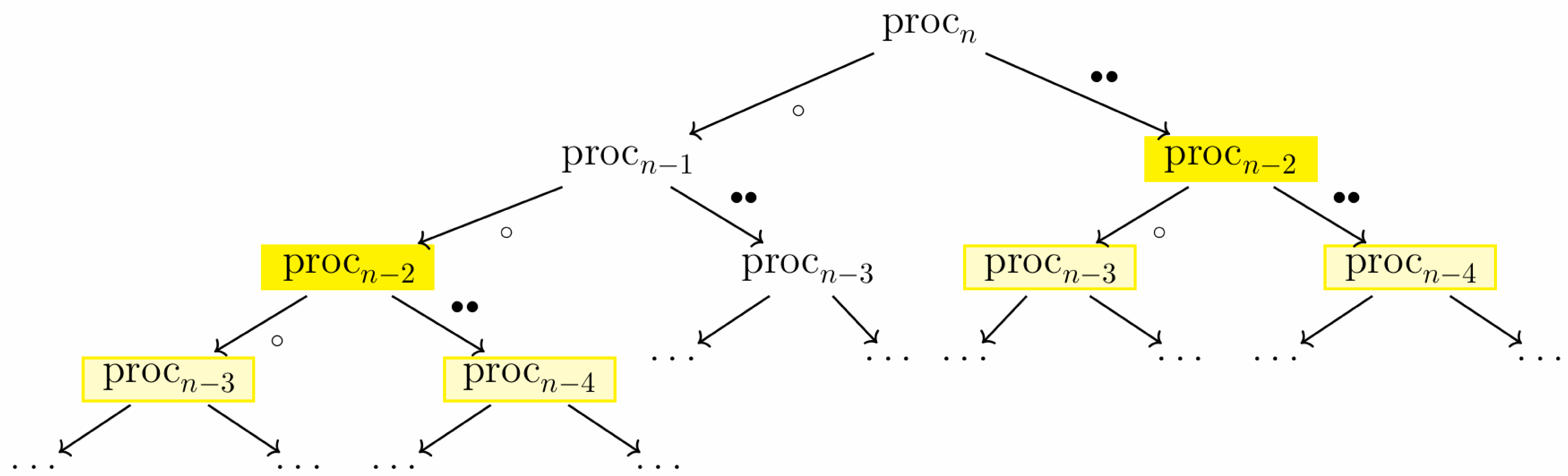
With complexity $O(kn)$ we can merge k adjacent copies of the same unitary from different branches.



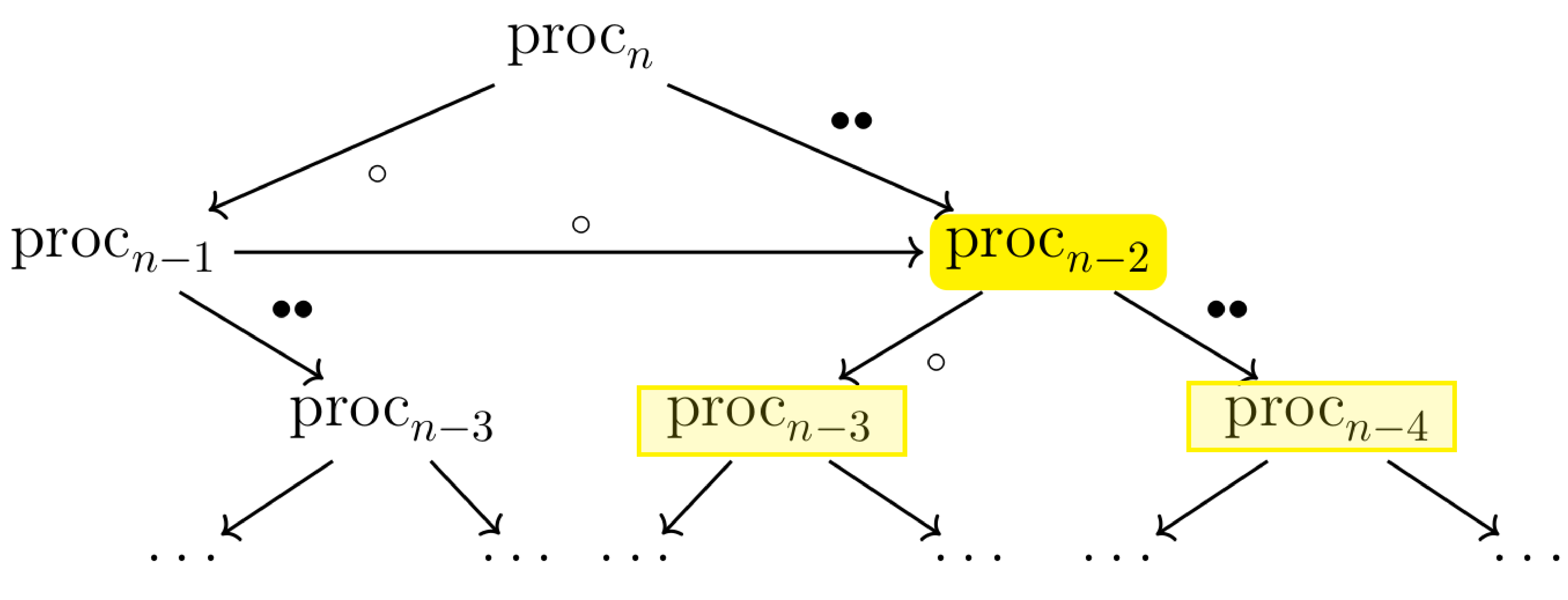
Circuit compilation



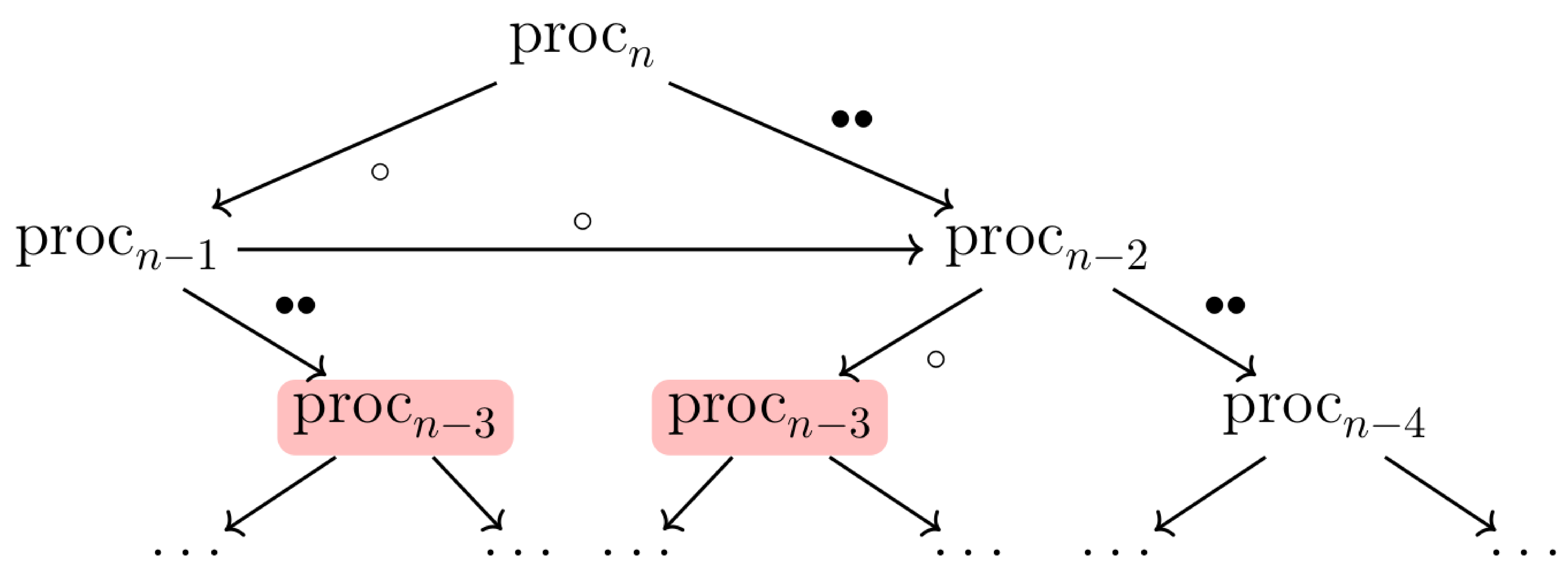
Circuit compilation



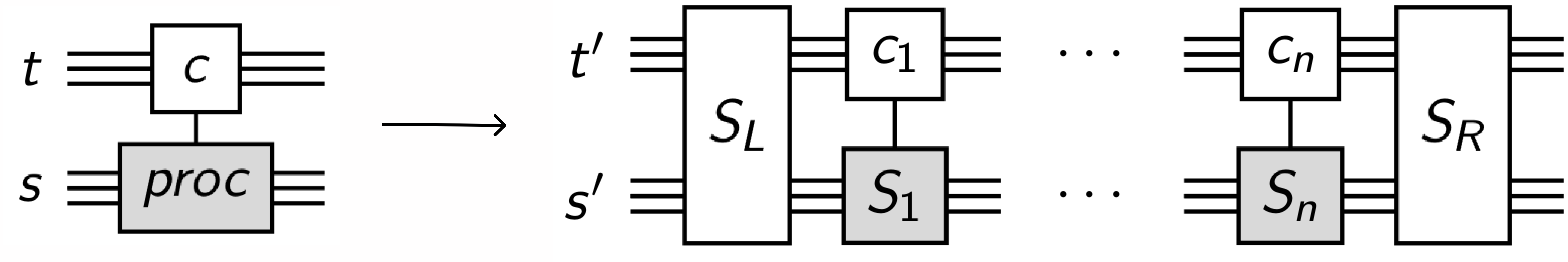
Circuit compilation



Circuit compilation

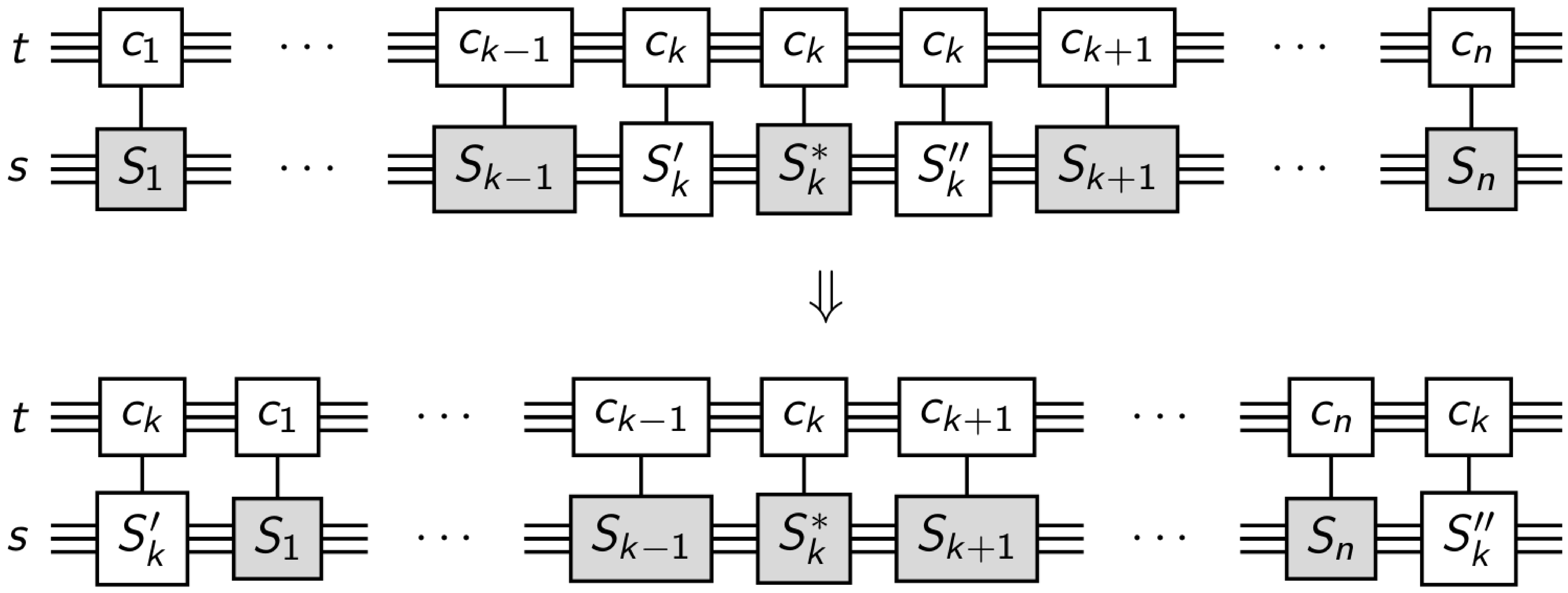


Guaranteeing adjacency



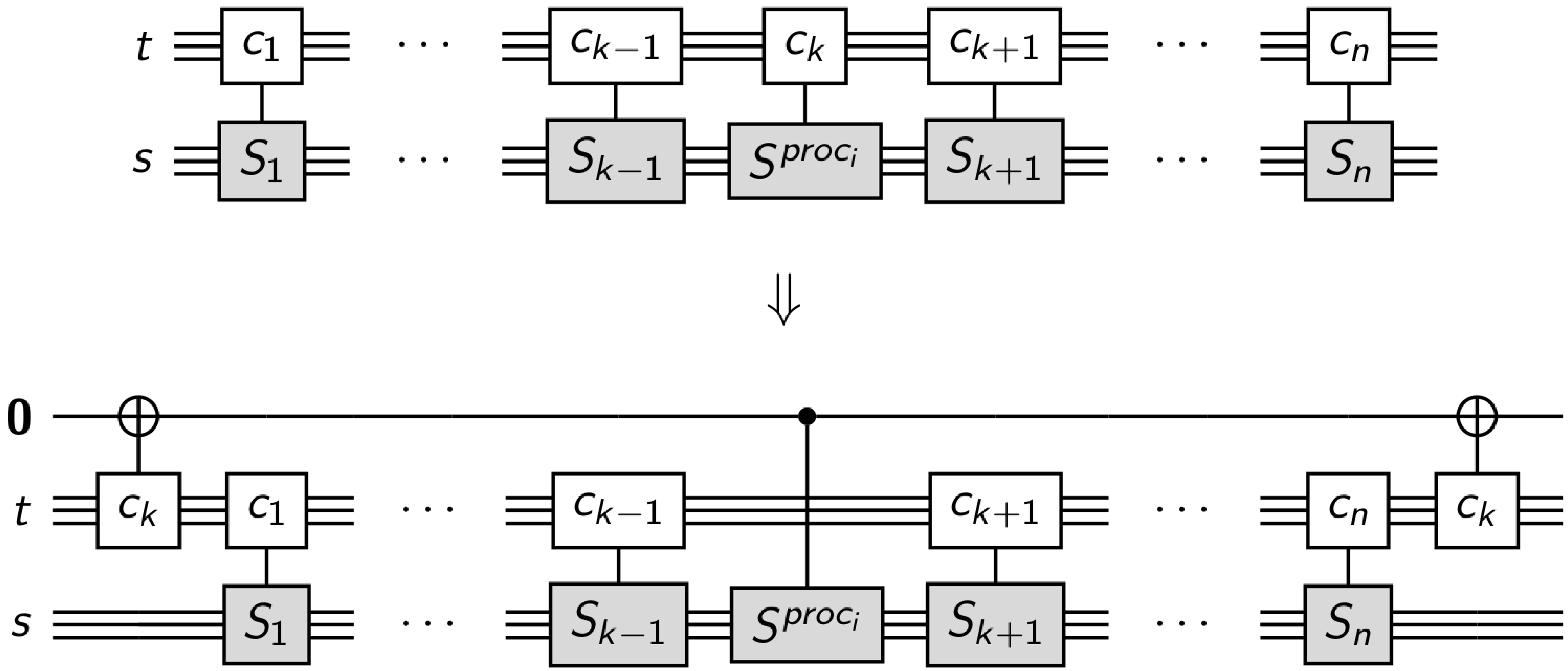
Guaranteeing adjacency

example: **Composition** $S_k = S'_k\ S^*_k\ S''_k$



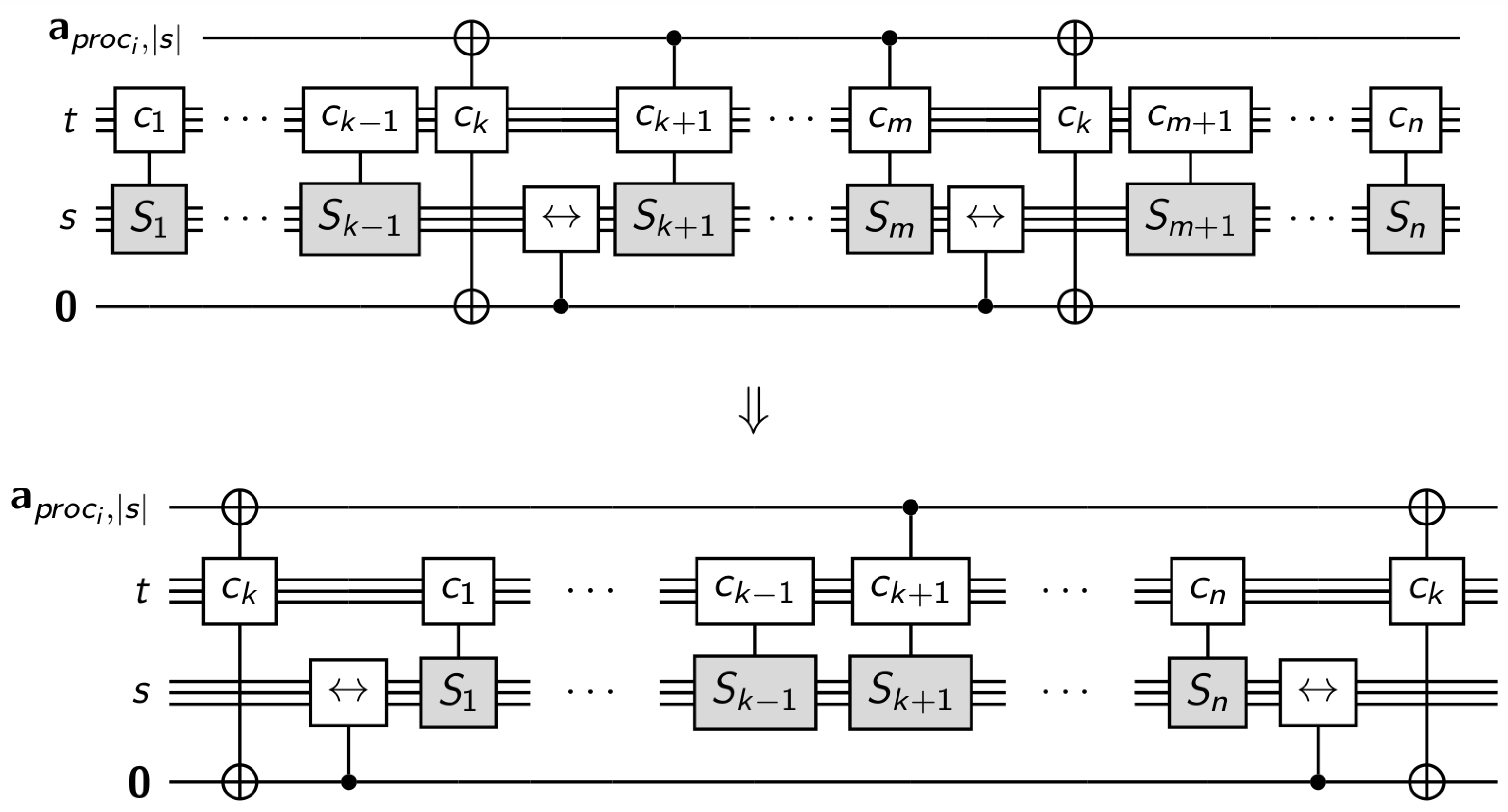
Guaranteeing adjacency

example: **Procedure call** $S_k = \textbf{call}\ proc_i(\cdot)$ (first occurrence of procedure and size)



Guaranteeing adjacency

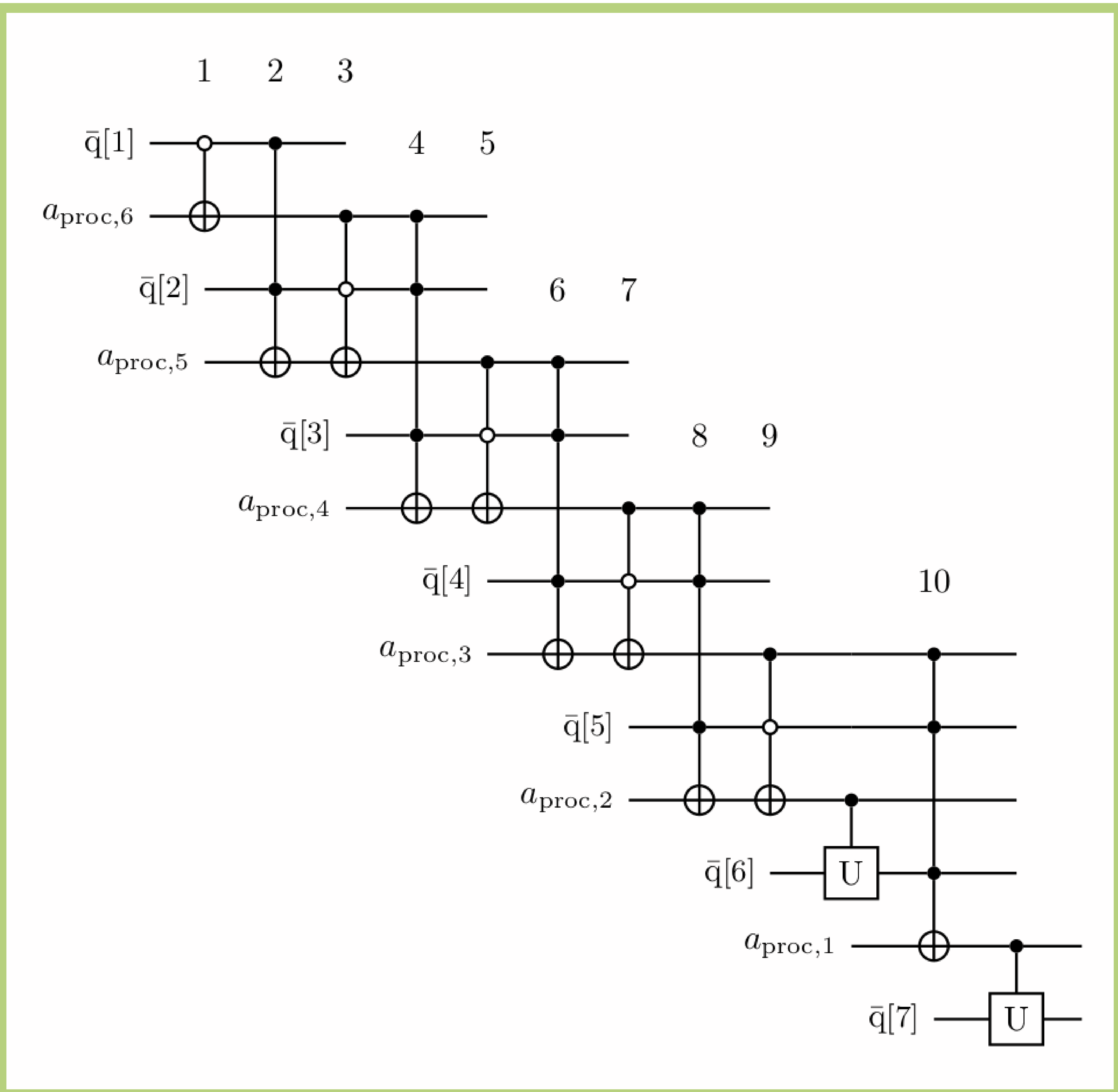
example: **Procedure call** (**not** the first occurrence)



Building a poly-sized circuit

$$
\begin{tikzcd}
{} \arrow[r, "2 \bullet\bullet"] \arrow[d, "1 \circ"'] & a_{\mathrm{proc},5} \arrow[r, "6 \bullet\bullet"] \arrow[d, "5 \circ"] & a_{\mathrm{proc},3} \arrow[r, "10 \bullet\bullet"] \arrow[d, "9 \circ"] & a_{\mathrm{proc},1} \\
a_{\mathrm{proc},6} \arrow[r, "4 \bullet\bullet"] \arrow[ur, "3 \circ"] & a_{\mathrm{proc},4} \arrow[r, "8 \bullet\bullet"] \arrow[ur, "7 \circ"] & a_{\mathrm{proc},2}
\end{tikzcd}
$$

- Same-sized instances of a procedure can always be merged
- In this case, all procedure calls can be computed using only $O(n)$ procedure instances



Conclusion

- **FOQ** is a first order quantum programming language with quantum control and recursive procedures.
- Syntactical restrictions allow for classes **WF** and **PFOQ** with properties of (poly-time) termination.
- **PFOQ** programs can be directly compiled into circuits that grow polynomially on the size of the input

Future work

- Expand the syntax (while loops, measurements);
- Applying restrictions to established languages (ProtoQuipper).

Thank you!