Quantum Functional Programming Language & Its Denotational Semantics

Ichiro Hasuo
Dept. Computer Science
University of Tokyo

Naohiko Hoshino
Research Inst. for Math. Sci.
Kyoto University

Talk based on:

I. Hasuo & N. Hoshino,
Semantics of Higher-Order Quantum Computation via Geometry of Interaction,
to appear in Proc. *Logic in Computer Science (LICS)*, June 2011.

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language & Its Denotational Semantics

What?

Why?

Overview

- Why programming language?
- Why functional programming language?
- Why semantics?
- Why denotational semantics?

Overview

- Why programming language?
- Why functional programming language?
- Why semantics?
- Why denotational semantics?

<u>Contribution</u>

First denotational semantics for full-featured QFPL

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

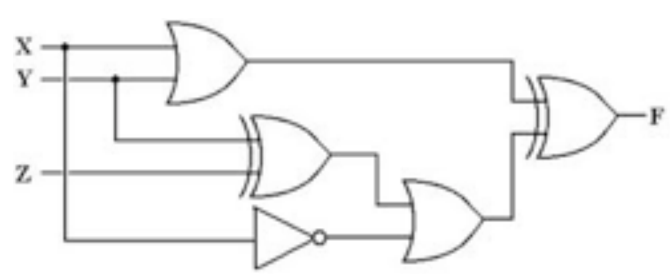
Q1. Why programming language?

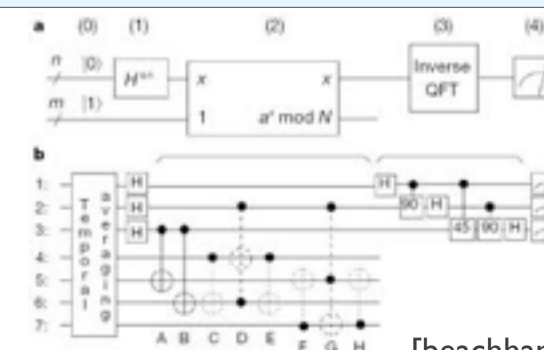
Formalisms

- We need one...   for describing/studying quantum algorithms

Formalisms

- We need one...   for describing/studying quantum algorithms

<table>
<tr><th></th><th>Classical</th><th></th><th>Quantum</th><th></th></tr>
<tr><td>(Boolean) circuit</td><td>

[Null-Lobur]</td><td>Quantum circuit</td><td>
[beachhandball.es]</td></tr>
<tr><td>Programming language</td><td><pre>
int i,j;
int factorial(int k)
{
    j=1;
    for (i=1; i<=k; i++)
        j=j*i;
    return j;
}
</pre></td><td></td><td></td></tr>
</table>

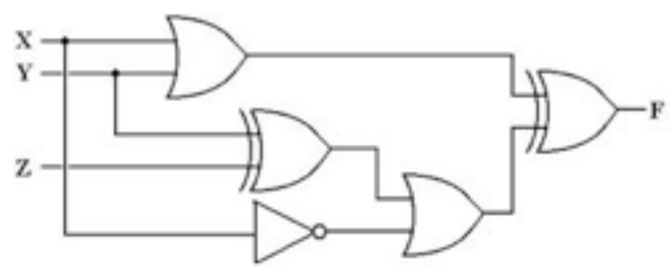
Hasuo (Tokyo), Hoshino (Kyoto)

Formalisms

- We need one... for describing/studying quantum algorithms

Classical

(Boolean) circuit



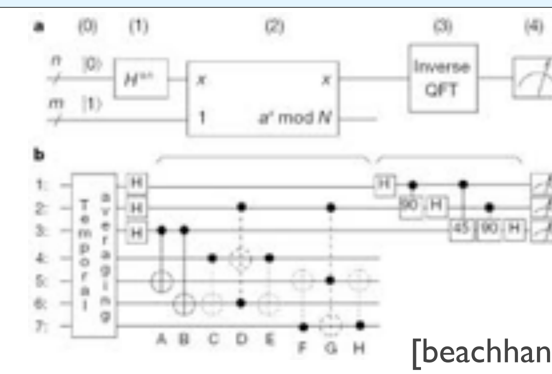
[Null-Lobur]

Programming language

```
int i,j;
int factorial(int k)
{
    j=1;
    for (i=1; i<=k; i++)
        j=j*i;
    return j;
}
```

Quantum

Quantum circuit



[beachhandball.es]

Quantum programming language

$$\begin{aligned}\mathbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \mathbf{EPR} * \ \mathit{in} \\ & \quad \mathit{let}\ f = \mathbf{BellMeasure}\ x\ \mathit{in} \\ & \quad \mathit{let}\ g = \mathbf{U}\ y \\ & \quad \mathit{in}\ \langle f, g\rangle.\end{aligned}$$

[Selinger-Valiron]

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψA, ψB;
        ψEPR aliasfor [ψA, ψB];
        channel[int] c withends [c0, c1];

        ψEPR = createEPR();
        c = new channel[int]();
        fork bert(c0, ψB);

        angela(c1, ψA);
}

void angela(channelEnd[int] c1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c1, r);
}
```

```
qbit bert(channelEnd[int] c0, qbit stto) {
        int i;

        i = recv (c0);
        if (i == 0) {
                opB0(stto);
        } else if (i == 1) {
                opB1(stto);
        } else if (i == 2) {
                opB2(stto);
        } else {
                opB3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

$$\begin{aligned}\textbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \textbf{EPR} * \mathit{in} \\ & \quad \mathit{let}\ f = \textbf{BellMeasure}\ x\ \mathit{in} \\ & \quad \mathit{let}\ g = \textbf{U}\ y \\ & \quad\ \ \mathit{in}\ \langle f, g\rangle.\end{aligned}$$

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}

qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

$$
\begin{aligned}
\mathbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \mathbf{EPR} * \mathit{in} \\
& \quad \mathit{let}\ f = \mathbf{BellMeasure}\ x\ \mathit{in} \\
& \quad \mathit{let}\ g = \mathbf{U}\ y \\
& \quad\ \ \mathit{in}\ \langle f, g\rangle.
\end{aligned}
$$

- "High-level" ➜ new algorithms?

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}

qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

$$\begin{aligned}\mathbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \mathbf{EPR} * \mathit{in} \\ & \quad \mathit{let}\ f = \mathbf{BellMeasure}\ x\ \mathit{in} \\ & \quad \mathit{let}\ g = \mathbf{U}\ y \\ & \quad\ \ \mathit{in}\ \langle f, g\rangle.\end{aligned}$$

- "High-level" ➜ new algorithms?
- Well-developed techniques for correctness guarantee (*verification*)
  - Type system
  - Program model checking
  - etc.

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

Q2. Why *functional* programming language?

(Classical) Functional Programming Languages

- Computation as *evaluation of mathematical functions*
- Avoids *(memory) state* or *mutable data*
- Scheme, Erlang, ML (SML, OCaml), Haskell, F#, ...

```
int i,j;
int factorial(int k)
{
    j=1;
    for (i=1; i<=k; i++)
        j=j*i;
    return j;
}
```

Factorial in C

```
fun factorial x =
    if x = 0 then 1 else x * factorial (x-1)
```

Factorial in ML

(Classical) Functional Programming Languages

- Higher-order computation

twice $f = \lambda x. f(fx)$ as

```
fun twice (f : int -> int) : int -> int =
    fn (x : int) => f (f x)
```

- Modularity, code reusability

(Classical) Functional Programming Languages

- Higher-order computation

twice $f = \lambda x.f(fx)$ as

```
fun twice (f : int -> int) : int -> int =
    fn (x : int) => f (f x)
```

  - Modularity, code reusability

- Mathematically clean
  - Programs as *functions!*

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to quantum

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to quantum

- Uniform treatment of *quantum data* and *classical data*

Quantum Functional Programming

- “Mathematical”
  - ➜ *Mathematical transfer* from classical t

“quantum data, classical control”

- Uniform treatment of *quantum data* and *classical data*

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to

"quantum data,
classical control"

- Uniform treatment of *quantum data* and *classical data*
  - Nicely enforced by *types*

```
0 : int        + : int * int -> int
```

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to

"quantum data, classical control"

- Uniform treatment of *quantum data* and *classical data*
  - Nicely enforced by *types*

```
0 : int        + : int * int -> int
```

$$\mathtt{new}\,|0\rangle \;:\; \mathtt{qbit} \qquad \mathtt{tt} \;:\; !\,\mathtt{bit}$$
$$\mathtt{meas} \;:\; \mathtt{qbit} \multimap !\,\mathtt{bit}$$

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to

"quantum data, classical control"

- Uniform treatment of *quantum data* and *classical data*
  - Nicely enforced by *types*

```
0 : int        + : int * int -> int
```

$$\mathtt{new}\,|0\rangle \;:\; \mathtt{qbit} \qquad\qquad \mathtt{tt} \;:\; !\,\mathtt{bit}$$
$$\mathtt{meas} \;:\; \mathtt{qbit} \multimap !\,\mathtt{bit}$$

! : "duplicable"

Quantum Functional Programming

- "Mathematical"
  - ➜ *Mathematical transfer* from classical to

"quantum data, classical control"

- Uniform treatment of *quantum data* and *classical data*
  - Nicely enforced by *types*

```
0 : int        + : int * int -> int
```

$$\texttt{new}\,|0\rangle \ : \ \texttt{qbit} \qquad\qquad \texttt{tt} \ : \ !\texttt{bit}$$
$$\texttt{meas} \ : \ \texttt{qbit} \multimap !\texttt{bit}$$

! : "duplicable"

--o : "linear function"
(input is used only once)

Our Language $q\lambda_\ell$

λ-*calculus* (prototype FPL)

- Based on [Selinger-Valiron, 2008]
- *Types:*

λ-calculus
(prototype FPL)

Our Language qλ_ℓ

- Based on [Selinger-Valiron, 2008]
- *Types:*

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid \; !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

λ-calculus
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Seling … 08]
- *Types:*

n-qubit state

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid \; !A \mid A \multimap B \mid A \boxtimes B \mid \dots$$

λ-*calculus*
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Seling
- *Types:*

n-qubit state

(classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid \; !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

λ-*calculus*
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Seling
- *Types:*

n-qubit state

(classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid \; !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

A, duplicable

Our Language $q\lambda_\ell$

λ-calculus
(prototype FPL)

- Based on [Seling
- *Types:*

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid\ !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

n-qubit state

(classical) bit

A, duplicable

linear function

λ-*calculus*
(prototype FPL)

Our Language qλ_ℓ

- Based on [Seling
- *Types:*

n-qubit state

(classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid$$
$$!\,A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

A, duplicable

linear function

pair of A & B

λ-calculus
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Seling

n-qubit state

(classical) bit

- *Types:*

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid$$
$$!\,A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

A, duplicable

linear function

pair of A & B

- *Programs or terms:*

$$M, N ::=$$
$$x \in \mathbf{Var} \mid \lambda x^{A}.M \mid M\,N \mid$$
$$\langle M, N\rangle \mid \mathtt{let}\,\langle x^{A}, y^{B}\rangle = M\,\mathtt{in}\,N$$
$$\mathtt{letrec}\,f^{A}x = M\,\mathtt{in}\,N \mid$$
$$\mathtt{new}\,|\mathbf{0}\rangle \mid \mathtt{meas}_i^{n+1} \mid U \mid \mathtt{cmp}_{m,n}$$

λ-calculus
(prototype FPL)

Our Language qλ_ℓ

- Based on [Seling… 0]
- *Types:*

n-qubit state

(classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid\ !A \mid A \multimap B \mid A \boxtimes B \mid \dots$$

A, duplicable

linear function

pair of A & B

- *Programs or terms:*

$$\begin{aligned} M, N ::=\ & \\ & x \in \mathbf{Var} \mid \lambda x^A.M \mid MN \mid \\ & \langle M, N\rangle \mid \mathtt{let}\ \langle x^A, y^B\rangle = M\ \mathtt{in}\ N \\ & \mathtt{letrec}\ f^A x = M\ \mathtt{in}\ N \mid \\ & \mathtt{new}\ |0\rangle \mid \mathtt{meas}_i^{n+1} \mid U \mid \mathtt{cmp}_{m,n} \end{aligned}$$

← function

λ-calculus
(prototype FPL)

Our Language qλ_ℓ

- Based on [Seling... 2006]
- *Types:*

n-qubit state | (classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid\\ !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

A, duplicable | linear function | pair of A & B

- *Programs or terms:*

$$M, N ::=$$
$$x \in \mathbf{Var} \mid \lambda x^A.M \mid MN \mid \qquad \leftarrow \text{function}$$
$$\langle M, N\rangle \mid \mathtt{let}\ \langle x^A, y^B\rangle = M\ \mathtt{in}\ N \qquad \leftarrow \text{pairing}$$
$$\mathtt{letrec}\ f^A x = M\ \mathtt{in}\ N \mid$$
$$\mathtt{new}\ |0\rangle \mid \mathtt{meas}_i^{n+1} \mid U \mid \mathtt{cmp}_{m,n}$$

λ-calculus
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Seling... ...2006]
- *Types:*

n-qubit state

(classical) bit

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid\ !\,A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

A, duplicable

linear function

pair of A & B

- *Programs or terms:*

$$\begin{aligned} M, N ::=\ & x \in \mathbf{Var} \mid \lambda x^A.M \mid M\,N \mid && \leftarrow \text{function} \\ & \langle M, N\rangle \mid \mathtt{let}\ \langle x^A, y^B\rangle = M\ \mathtt{in}\ N && \leftarrow \text{pairing} \\ & \mathtt{letrec}\ f^A x = M\ \mathtt{in}\ N \mid && \leftarrow \text{recursive def. of func.} \\ & \mathtt{new}\ |0\rangle \mid \mathtt{meas}_i^{n+1} \mid U \mid \mathtt{cmp}_{m,n} \end{aligned}$$

λ-calculus
(prototype FPL)

Our Language $q\lambda_\ell$

- Based on [Selinger-Valiron 2008]
- *Types:*

$$A, B ::= n\text{-}\mathtt{qbit} \mid \mathtt{bit} \mid \; !A \mid A \multimap B \mid A \boxtimes B \mid \ldots$$

n-qubit state

(classical) bit

A, duplicable

linear function

pair of A & B

- *Programs or terms:*

$$\begin{aligned} M, N ::= & \\ & x \in \mathbf{Var} \mid \lambda x^A.M \mid MN \mid & \leftarrow \text{function} \\ & \langle M, N \rangle \mid \mathtt{let}\, \langle x^A, y^B \rangle = M \,\mathtt{in}\, N & \leftarrow \text{pairing} \\ & \mathtt{letrec}\, f^A x = M \,\mathtt{in}\, N \mid & \leftarrow \text{recursive def. of func.} \\ & \mathtt{new}\, |0\rangle \mid \mathtt{meas}_i^{n+1} \mid U \mid \mathtt{cmp}_{m,n} & \leftarrow \text{quantum primitives} \end{aligned}$$

Our Language qλ_ℓ

- *Typing rules:* N.B. Only some are shown. Very much simplified

$$\frac{}{!\,\Delta, x : A \vdash x : A}\ (\mathrm{Ax}.1)$$

$$\frac{}{!\,\Delta \vdash \mathtt{new}\,|0\rangle : \mathtt{qbit}}\ (\mathrm{Ax}.2) \qquad \frac{}{!\,\Delta \vdash \mathtt{meas} : !(\mathtt{qbit} \multimap \mathtt{bit})}\ (\mathrm{Ax}.2)$$

$$\frac{x : A, \Delta \vdash M : B}{\Delta \vdash \lambda x^A.M : A \multimap B}\ (\multimap.\mathrm{I}_1)$$

$$\frac{!\,\Delta, \Gamma_1 \vdash M : A \multimap B \quad !\,\Delta, \Gamma_2 \vdash N : A}{!\,\Delta, \Gamma_1, \Gamma_2 \vdash MN : B}\ (\multimap.\mathrm{E}), (\dagger)$$

$$\frac{!\,\Delta, \Gamma_1 \vdash M_1 : A_1 \quad !\,\Delta, \Gamma_2 \vdash M_2 : A_2}{!\,\Delta, \Gamma_1, \Gamma_2 \vdash \langle M_1, M_2 \rangle : A_1 \boxtimes A_2}\ (\boxtimes.\mathrm{I}), (\dagger)$$

Our Language qλ_ℓ

- ***Typing rules:*** N.B. Only some are shown. Very much simplified

$$\frac{}{!\Delta, x : A \vdash x : A}\ (\mathrm{Ax.1})$$

$$\frac{}{!\Delta \vdash \mathtt{new}\ |0\rangle : \mathtt{qbit}}\ (\mathrm{Ax.2}) \qquad \frac{}{!\Delta \vdash \mathtt{meas} : !(\mathtt{qbit} \multimap \mathtt{bit})}\ (\mathrm{Ax.2})$$

$$\frac{x : A, \Delta \vdash M : B}{\Delta \vdash \lambda x^A.M : A \multimap B}\ (\multimap.\mathrm{I}_1)$$

$$\frac{!\Delta, \Gamma_1 \vdash \underline{M : A \multimap B} \quad !\Delta, \Gamma_2 \vdash \underline{N : A}}{!\Delta, \Gamma_1, \Gamma_2 \vdash \underline{MN : B}}\ (\multimap.\mathrm{E}), (\dagger)$$

$$\frac{!\Delta, \Gamma_1 \vdash M_1 : A_1 \quad !\Delta, \Gamma_2 \vdash M_2 : A_2}{!\Delta, \Gamma_1, \Gamma_2 \vdash \langle M_1, M_2 \rangle : A_1 \boxtimes A_2}\ (\boxtimes.\mathrm{I}), (\dagger)$$

Type Discipline

- Typable ➜ "safe"
  - Guarantees minimal "correctness"

$$\vdash f^{A \multimap B} x^{A} : B \qquad \nvdash f^{A \multimap B} y^{A \multimap A}$$

$$\vdash \mathtt{meas}(x^{\mathtt{qbit}}) : \mathtt{bit} \qquad \nvdash \langle \mathtt{meas}(x^{\mathtt{qbit}}), \mathtt{meas}(Hx^{\mathtt{qbit}}) \rangle$$

Type Discipline

- Typable ➜ "safe"
  - Guarantees minimal "correctness"

$$\vdash f^{A \multimap B} x^{A} : B \qquad \nvdash f^{A \multimap B} y^{A \multimap A}$$

$$\vdash \mathtt{meas}(x^{\mathtt{qbit}}) : \mathtt{bit} \qquad \nvdash \langle \mathtt{meas}(x^{\mathtt{qbit}}), \mathtt{meas}(Hx^{\mathtt{qbit}}) \rangle$$

Faulty program

```
fun isValue t =
  case t of
    Num _ => 1
  | _ => false
```

compile

Type error

```
ex.sml:22.3-24.15 Error: types of rules don't agree
[literal]
  earlier rule(s): term -> int
  this rule: term -> bool
  in rule:
    _ => false
```

Examples

In [Selinger-Valiron]; similar in ours

- Quantum teleportation

$$\mathbf{EPR} = \lambda x.CNOT\langle H(new\,0), new\,0\rangle$$
$$\mathbf{BellMeasure} =$$
$$\lambda q_2.\lambda q_1.(let\ \langle x,y\rangle = CNOT\langle q_1,q_2\rangle\ in\ \langle meas(Hx), meas\ y\rangle$$
$$\mathbf{U} = \lambda q.\lambda\langle x,y\rangle.if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)\ else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q).$$
$$\begin{aligned}\mathbf{telep} = \quad & let\ \langle x,y\rangle = \mathbf{EPR} * in \\ & let\ f = \mathbf{BellMeasure}\ x\ in \\ & let\ g = \mathbf{U}\ y \\ & in\ \langle f,g\rangle.\end{aligned}$$

- (Fair) cointoss, repeated Hadamard

Flip coin:
- head ➜ Hadamard and flip again
- tail ➜ done

$$\mathbf{c} = \lambda *.\, meas(H(new\,0))$$
$$M = let\ rec\ f\ x = (if\,(\mathbf{c}\,*)\ then\ H\ (f\ x)\ else\ x)\ in\ f\ p$$

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language & Its Denotational Semantics

Q3. Why semantics?

Semantics

Semantics

- “Meaning” of a program

Semantics

- "Meaning" of a program
- For *reasoning about* programs

Semantics

- "Meaning" of a program

- For *reasoning about* programs
  - $M \cong N$: "M and N have the same meaning, i.e. computational content"

Semantics

- "Meaning" of a program

- For *reasoning about* programs
  - $M \cong N$: "M and N have the same meaning, i.e. computational content"

$$\lambda x.\,(x - x) \overset{??}{\cong} \lambda x.\,0$$

Semantics

- "Meaning" of a program
- For *reasoning about* programs
  - $M \cong N$: "M and N have the same meaning, i.e. computational content"

$$\lambda x.\,(x - x) \overset{??}{\cong} \lambda x.\,0$$

(stupid) sort $\cong$ quick sort

Semantics

- For functional languages:
  - *Operational*: how the program is *transformed/evaluated/reduced*

    $$(\lambda x.\, 1 + x)3 \;\longrightarrow\; 1 + 3 \;\longrightarrow\; 4$$

  - *Denotational*: "meaning" as a mathematical *function*

    $$[\![\lambda x.\, 1 + x]\!] \;=\; (\text{function } \mathbb{N} \to \mathbb{N},\; n \mapsto 1 + n)$$

Operational vs. Denotational

Operational	Denotational
$(\boldsymbol{\lambda x.\,1+x})\mathbf{3} \;\longrightarrow\; \mathbf{1+3} \;\longrightarrow\; \mathbf{4}$	$[\![\boldsymbol{\lambda x.\,1+x}]\!] \;=\; (\text{function } \mathbb{N}\to\mathbb{N},\; \boldsymbol{n}\mapsto \mathbf{1}+\boldsymbol{n})$
reduction-based *dynamic*	"mathematical" *static*

Hasuo (Tokyo), Hoshino (Kyoto)

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\, 1 + x)3 \;\longrightarrow\; 1 + 3 \;\longrightarrow\; 4$	$[\![\lambda x.\, 1 + x]\!] \;=\; (\text{function } \mathbb{N} \to \mathbb{N},\; n \mapsto 1 + n)$
reduction-based *dynamic*	"mathematical" *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

Hasuo (Tokyo), Hoshino (Kyoto)

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\, 1 + x)3 \;\longrightarrow\; 1 + 3 \;\longrightarrow\; 4$	$[\![\lambda x.\, 1 + x]\!] \;=\; (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1 + n)$
reduction-based *dynamic*	“mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\, 1 + x)3 \longrightarrow 1 + 3 \longrightarrow 4$	$[\![\lambda x.\, 1 + x]\!] = (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1 + n)$
reduction-based *dynamic*	"mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\text{opr.}} N$$

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\,1+x)3 \longrightarrow 1+3 \longrightarrow 4$	$[\![\lambda x.\,1+x]\!] = (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1+n)$
reduction-based *dynamic*	"mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\text{opr.}} N$$

$$\downarrow$$
$$\downarrow$$
$$\downarrow$$
$$\vdots$$

Hasuo (Tokyo), Hoshino (Kyoto)

Operational vs. Denotational

<u>Operational</u>	<u>Denotational</u>
$(\lambda x.\, 1 + x)3 \;\longrightarrow\; 1 + 3 \;\longrightarrow\; 4$	$[\![\lambda x.\, 1 + x]\!] \;=\; (\text{function } \mathbb{N} \to \mathbb{N},\; n \mapsto 1 + n)$
reduction-based *dynamic*	“mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\text{opr.}} N$$

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\, 1 + x)3 \longrightarrow 1 + 3 \longrightarrow 4$	$[\![\lambda x.\, 1 + x]\!] = (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1 + n)$
reduction-based *dynamic*	"mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\text{opr.}} N$$

hard to show directly

Hasuo (Tokyo), Hoshino (Kyoto)

Operational vs. Denotational

<u>Operational</u>	<u>Denotational</u>
$(\lambda x.\, 1+x)3 \longrightarrow 1+3 \longrightarrow 4$	$[\![\lambda x.\, 1+x]\!] = (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1+n)$
reduction-based *dynamic*	"mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\mathrm{opr.}} N \quad \Longleftarrow \quad [\![M]\!] = [\![N]\!]$$

hard to show directly

Hasuo (Tokyo), Hoshino (Kyoto)

Operational vs. Denotational

Operational	Denotational
$(\lambda x.\, 1 + x)3 \longrightarrow 1 + 3 \longrightarrow 4$	$[\![\lambda x.\, 1 + x]\!] = (\text{function } \mathbb{N} \to \mathbb{N},\ n \mapsto 1 + n)$
reduction-based *dynamic*	“mathemati *static*
(akin to) machine implementation	comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)

powerful esp. for recursion

Goal:

$$M \cong_{\text{opr.}} N \qquad \Longleftarrow \qquad [\![M]\!] = [\![N]\!]$$

hard to show directly

Adequate denotational semantics:

$$[\![M]\!] = [\![N]\!] \iff M \cong_{\text{opr.}} N$$

Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

Q4. Why no denotational semantics before?

Challenges

$$[\![\boldsymbol{H}]\!] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad : \quad \mathbb{C}^2 \longrightarrow \mathbb{C}^2 \ , \quad \text{isn't it?}$$

Challenges

$$[\![\boldsymbol{H}]\!] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad : \quad \mathbb{C}^2 \longrightarrow \mathbb{C}^2 \quad , \quad \text{isn't it?}$$

- "Quantum data, classical control"
  - ➜ not clear how to accommodate duplicable data in Hilbert spaces

Technical Contributions

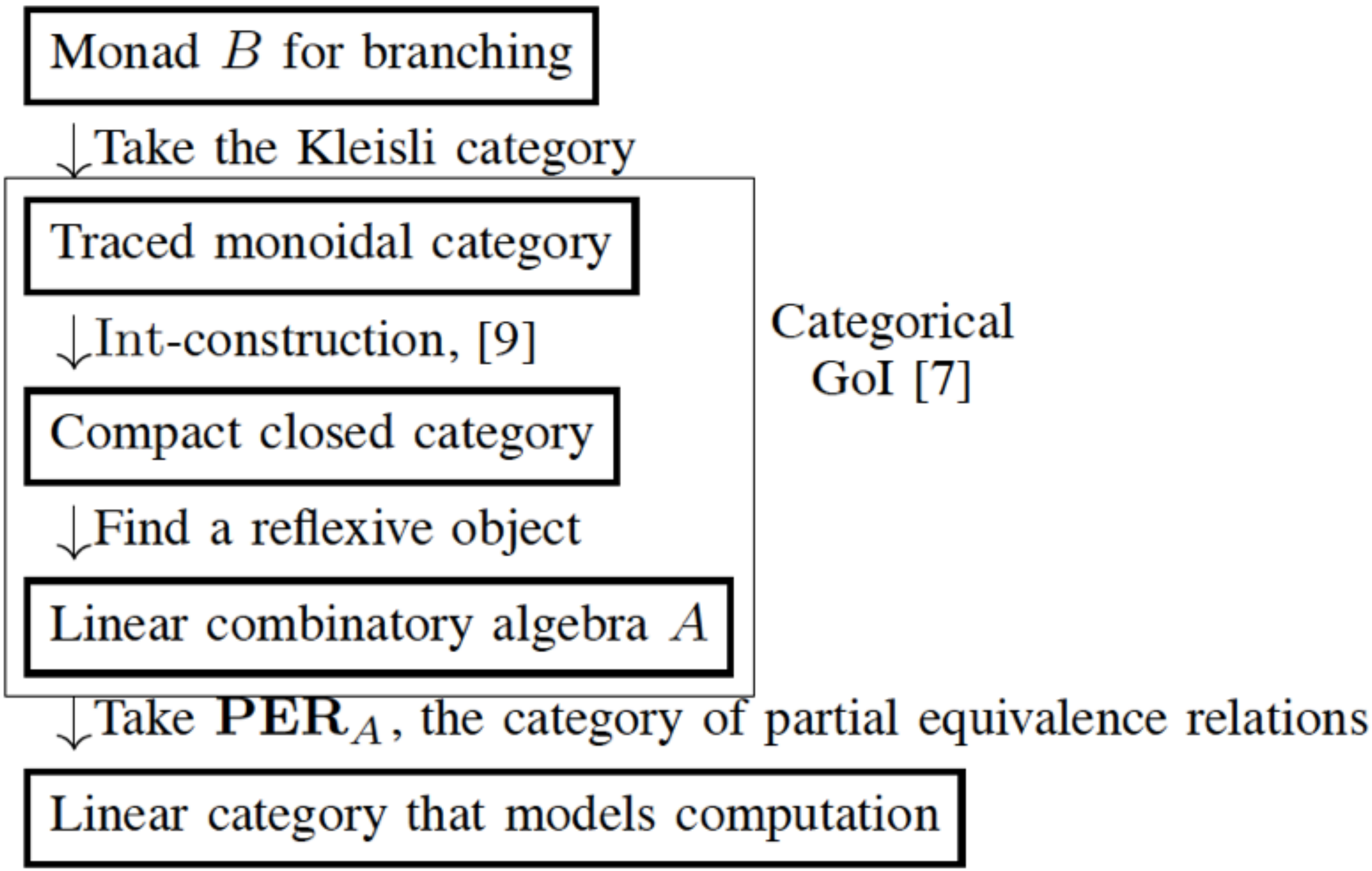
Technical Contributions

- Quantum functional programming language
  - Based on [Selinger-Valiron]
  - w/ recursion, classical data (by !)

Technical Contributions

- Quantum functional programming language
  - Based on [Selinger-Valiron]
  - w/ recursion, classical data (by !)
- Its denotational semantics
  - First one for fully-featured QFPL

Full-fledged Semantical Technologies



Hasuo (Tokyo), Hoshino (Kyoto)

Full-fledged Semantical Technologies

Monad B for branching

$\downarrow$ Take the Kleisli category

Traced monoidal category

$\downarrow$ Int-construction, [9]

Compact closed category

$\downarrow$ Find a reflexive object

Linear combinatory algebra A

$\downarrow$ Take $\mathbf{PER}_A$, the category of partial equivalence relations

Linear category that models computation

Categorical GoI [7]

Hasuo (Tokyo), Hoshino (Kyoto)

Geometry of Interaction

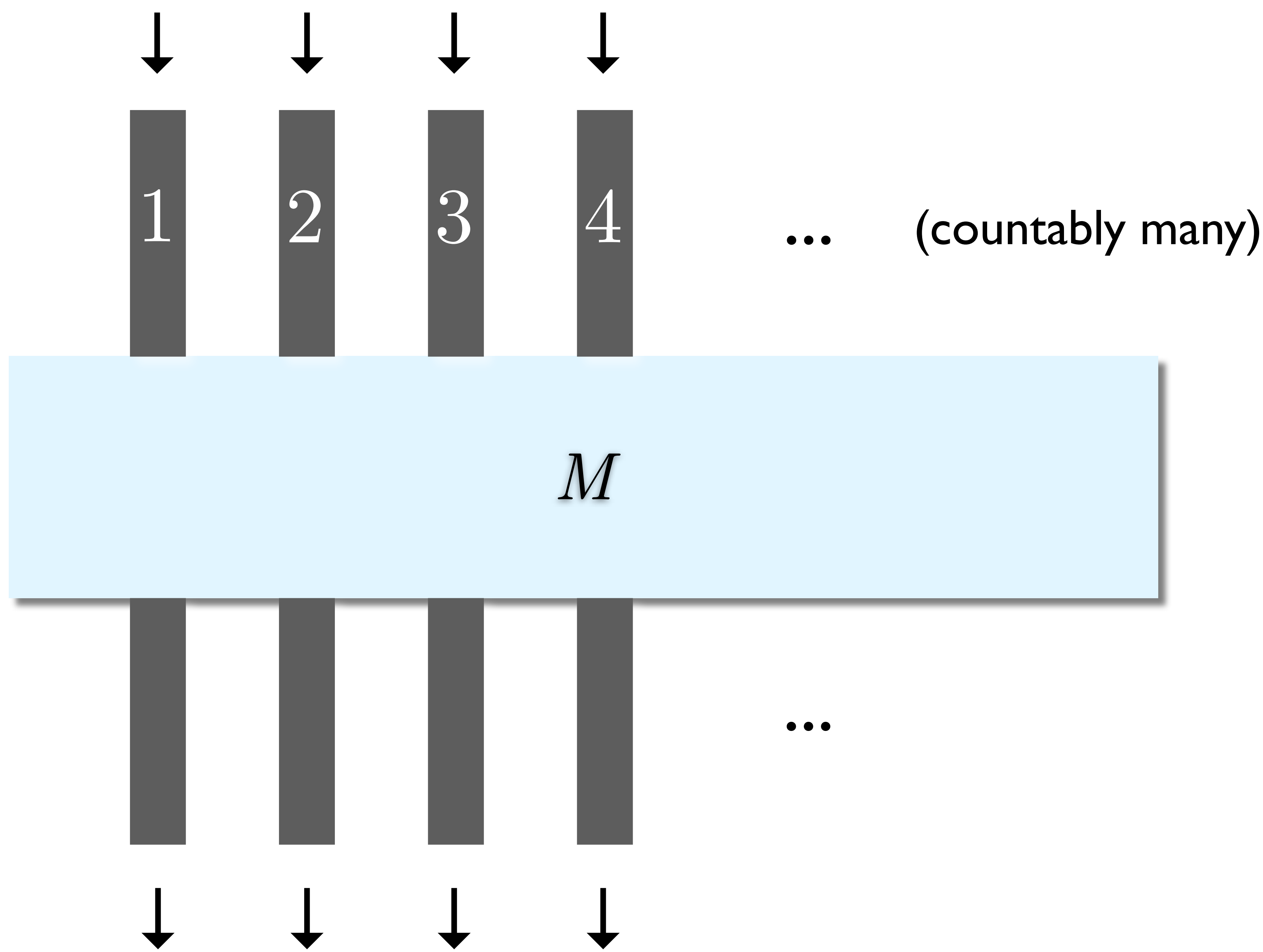
- Originally by J.-Y. Girard, 1989:
  - Computation as *player of a game*
  - cf. Game semantics (Abramsky et al., Hyland-Ong)
  - We use *categorical* formulation: Abramsky, Haghverdi and Scott, 2002

Geometry of Interaction

- Originally by J.-Y. Girard, 1989:
  - Computation as *player of a game*
  - cf. Game semantics (Abramsky et al., Hyland-Ong)
  - We use *categorical* formulation: Abramsky, Haghverdi and Scott, 2002
- *Axiomatization of what is "classical control"*

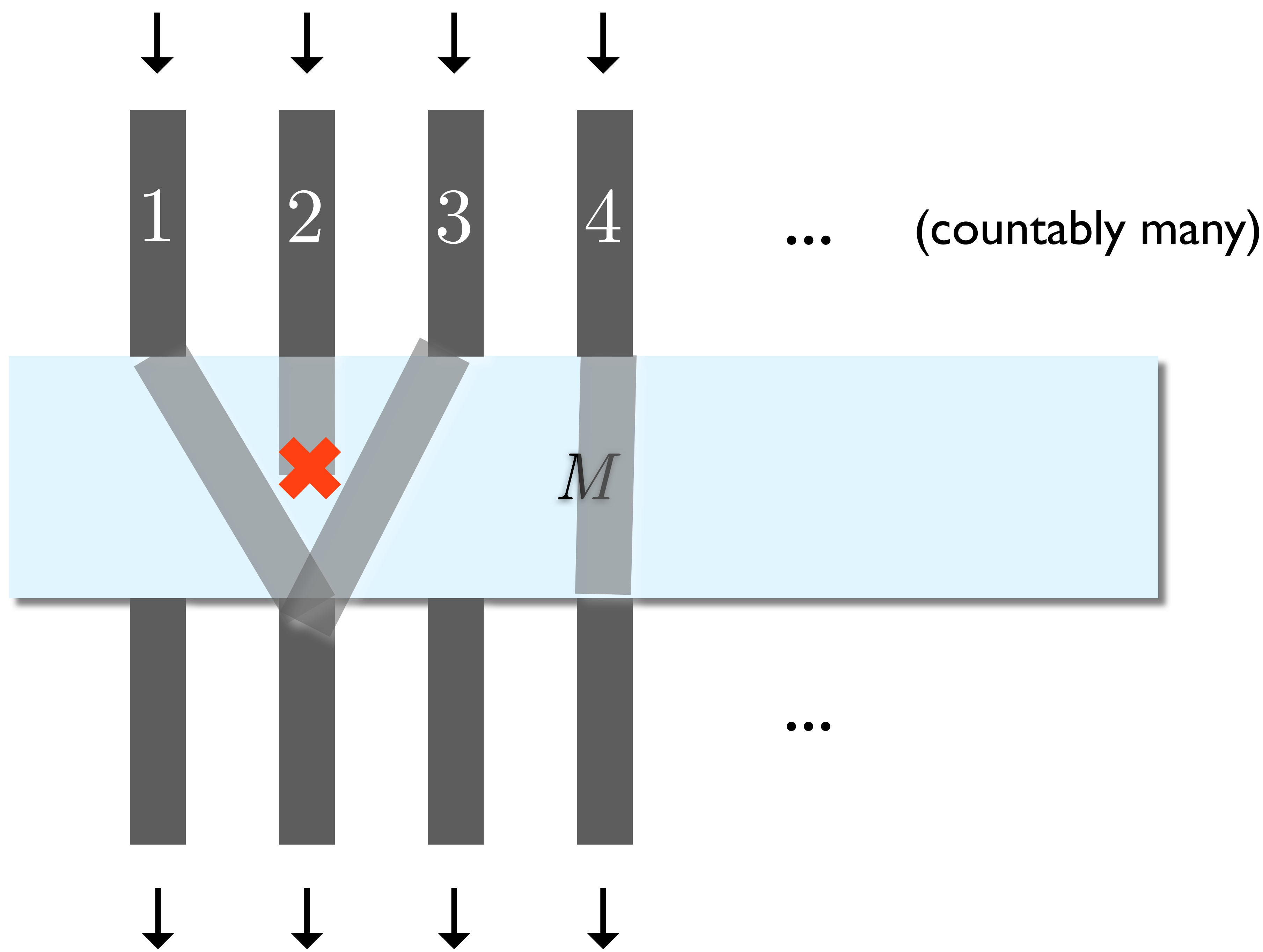
(Particle-Style) Geometry of Interaction

$$[\![M]\!] =$$



(Particle-Style) Geometry of Interaction

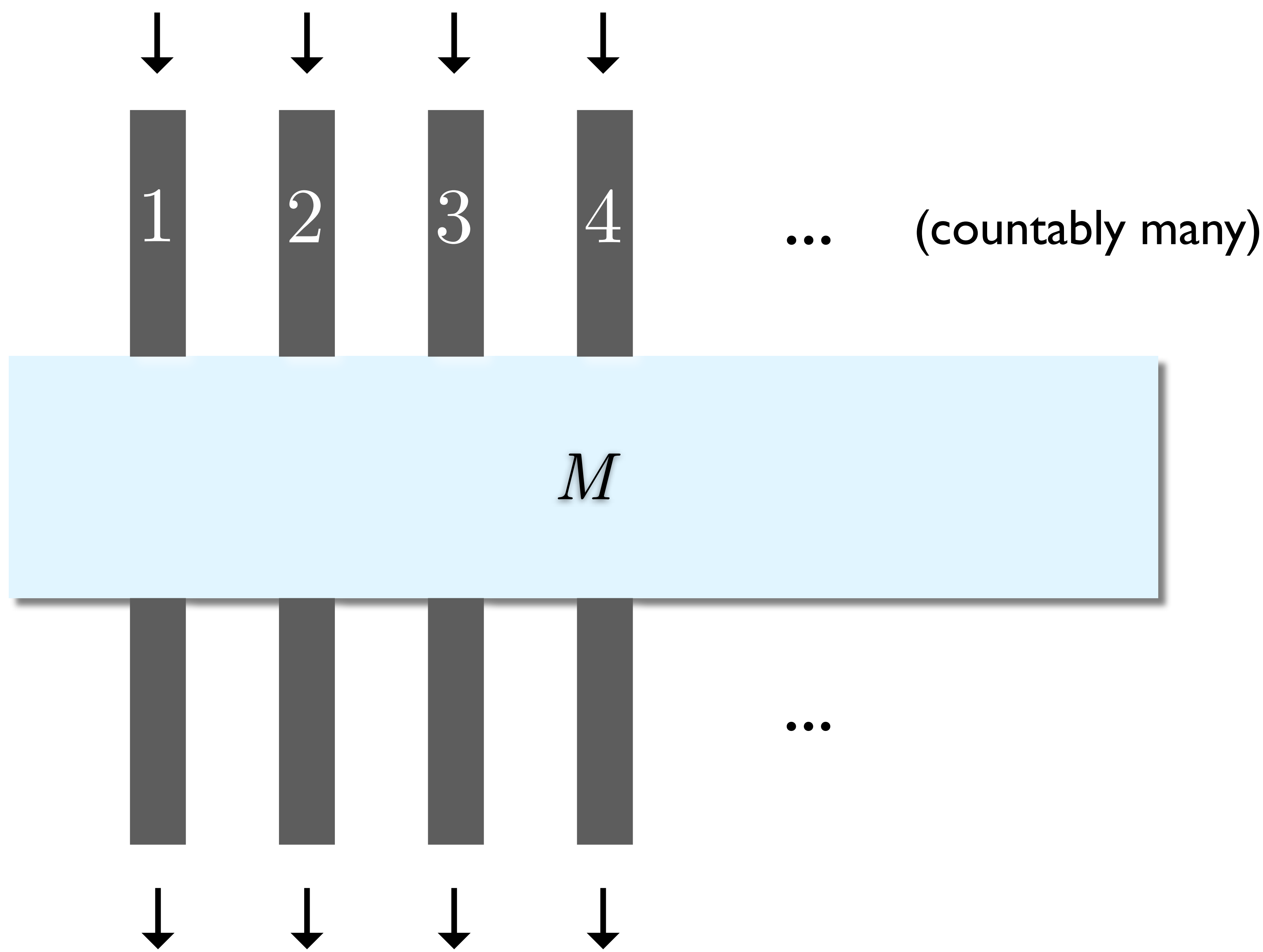
$$[\![M]\!] =$$



Hasuo (Tokyo), Hoshino (Kyoto)

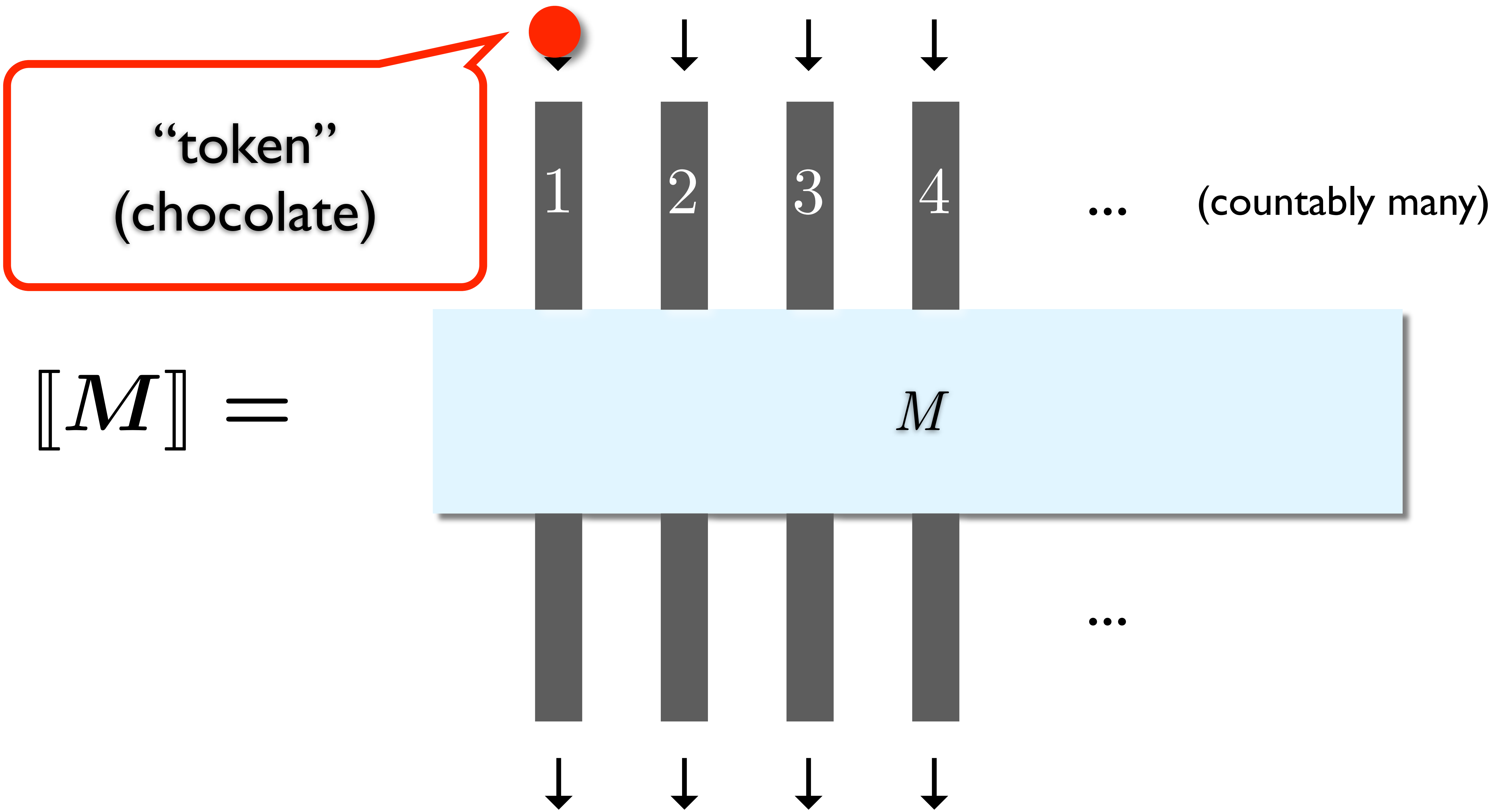
(Particle-Style) Geometry of Interaction

$[\![M]\!] =$



Hasuo (Tokyo), Hoshino (Kyoto)

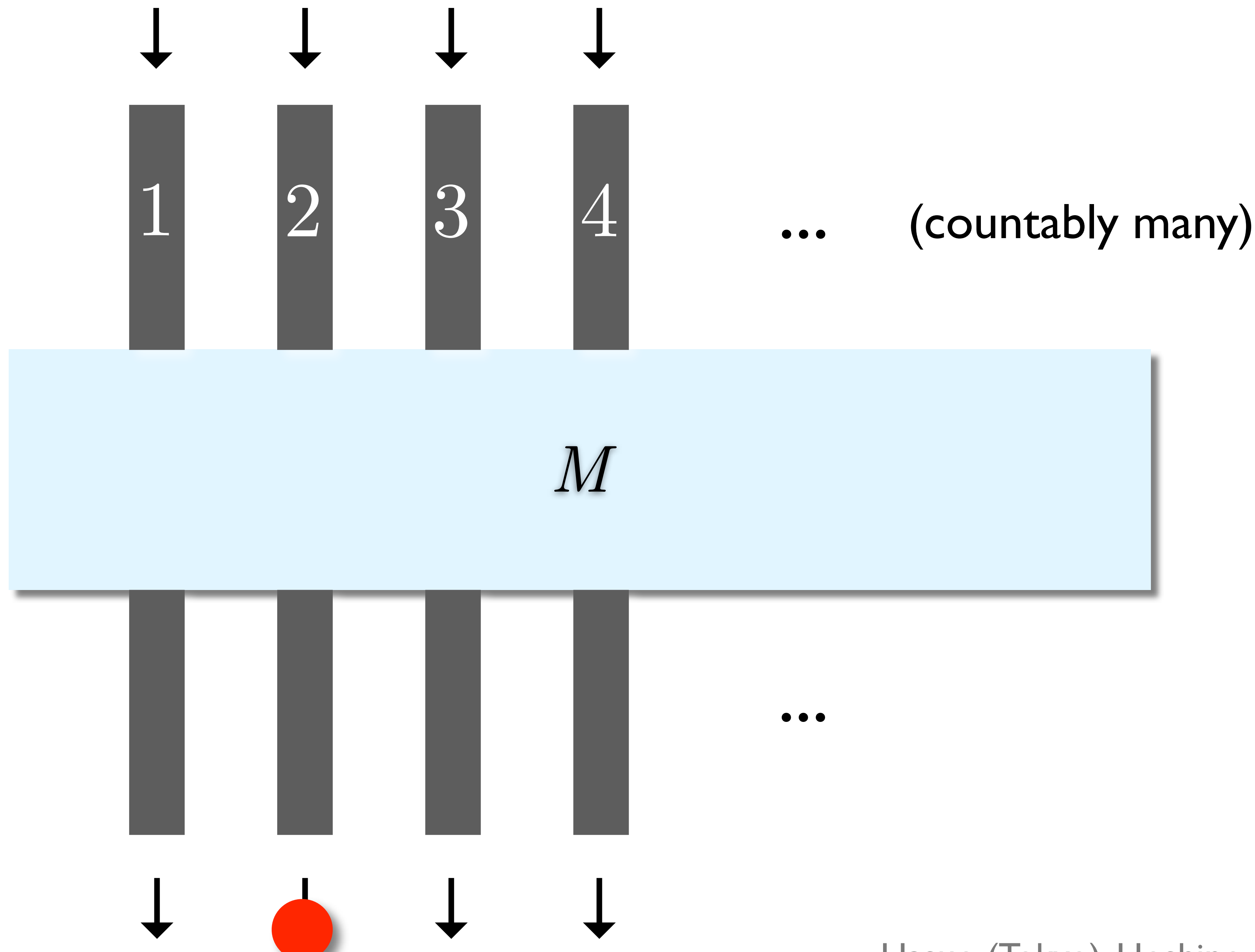
(Particle-Style) Geometry of Interaction



Hasuo (Tokyo), Hoshino (Kyoto)

(Particle-Style) Geometry of Interaction

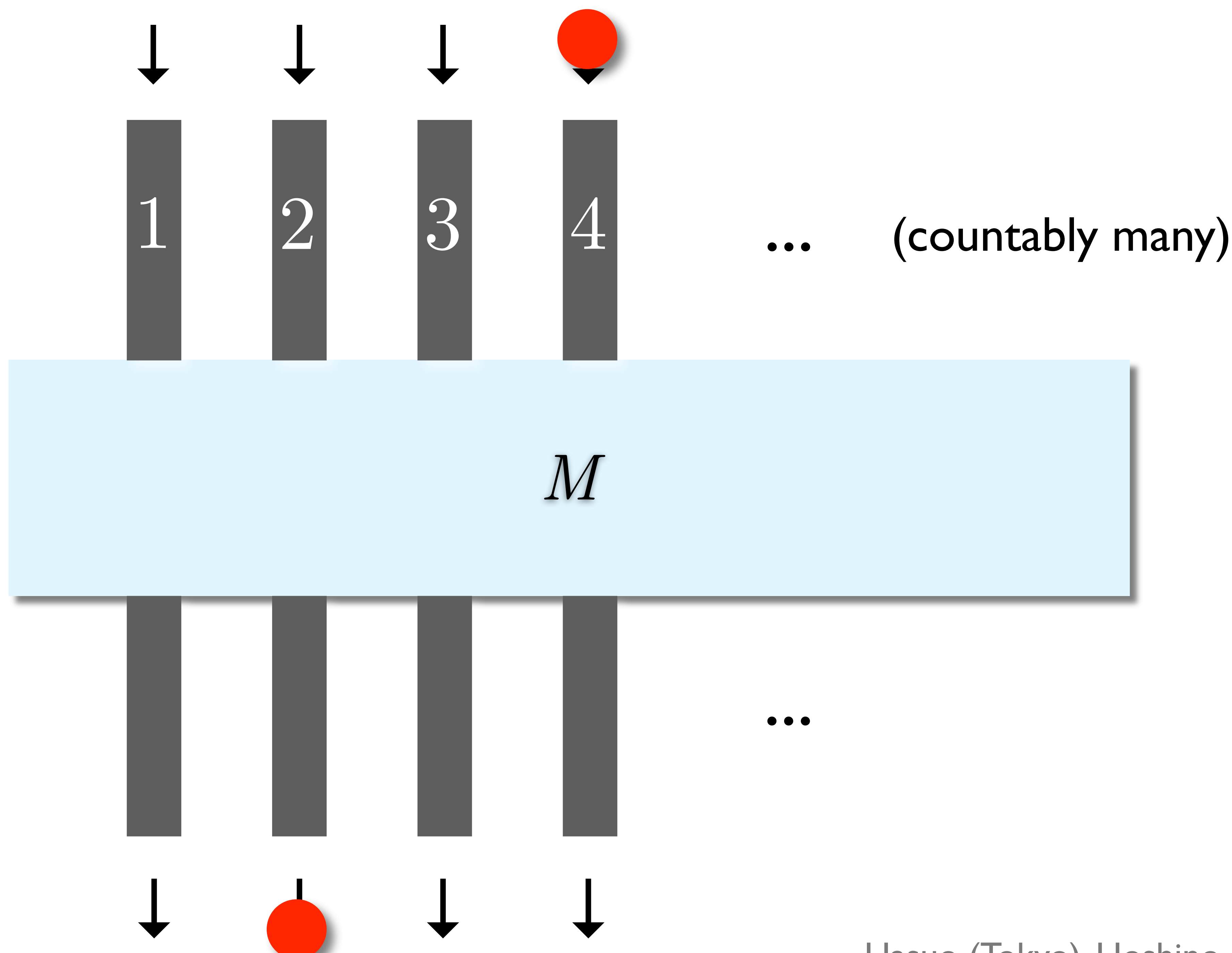
$[\![M]\!] =$



Hasuo (Tokyo), Hoshino (Kyoto)

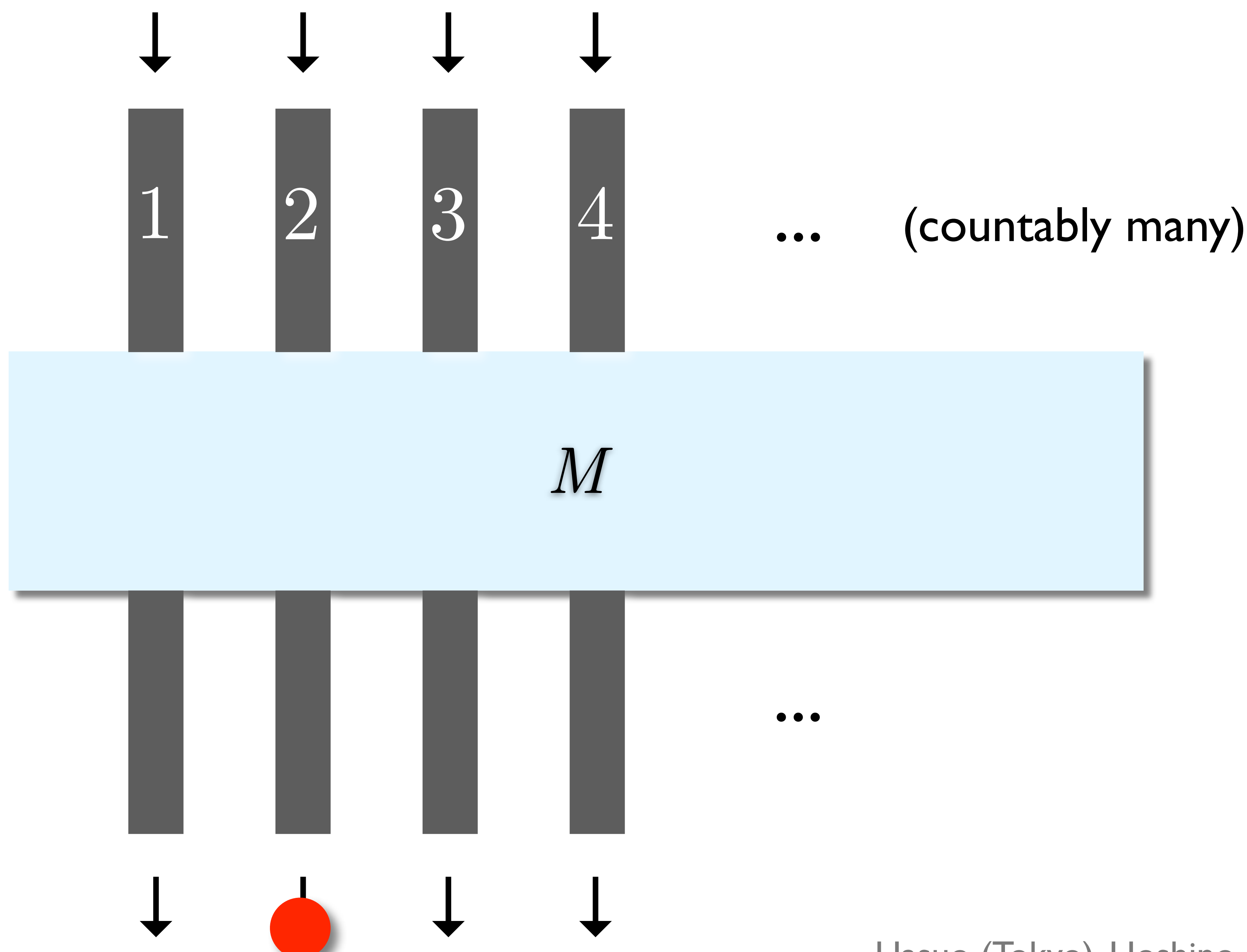
(Particle-Style) Geometry of Interaction

$$[\![M]\!] =$$



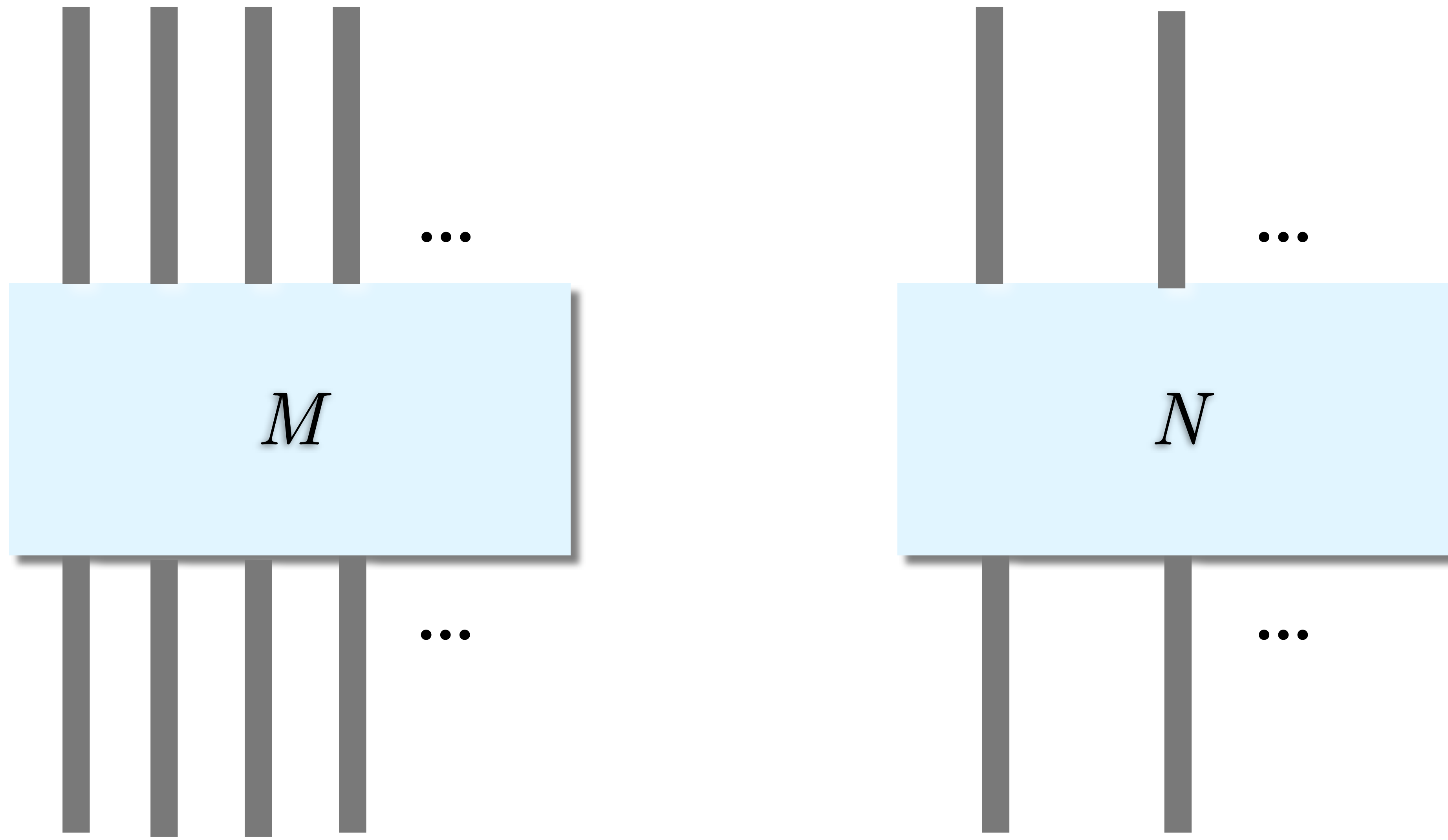
(Particle-Style) Geometry of Interaction

$$[\![M]\!] =$$

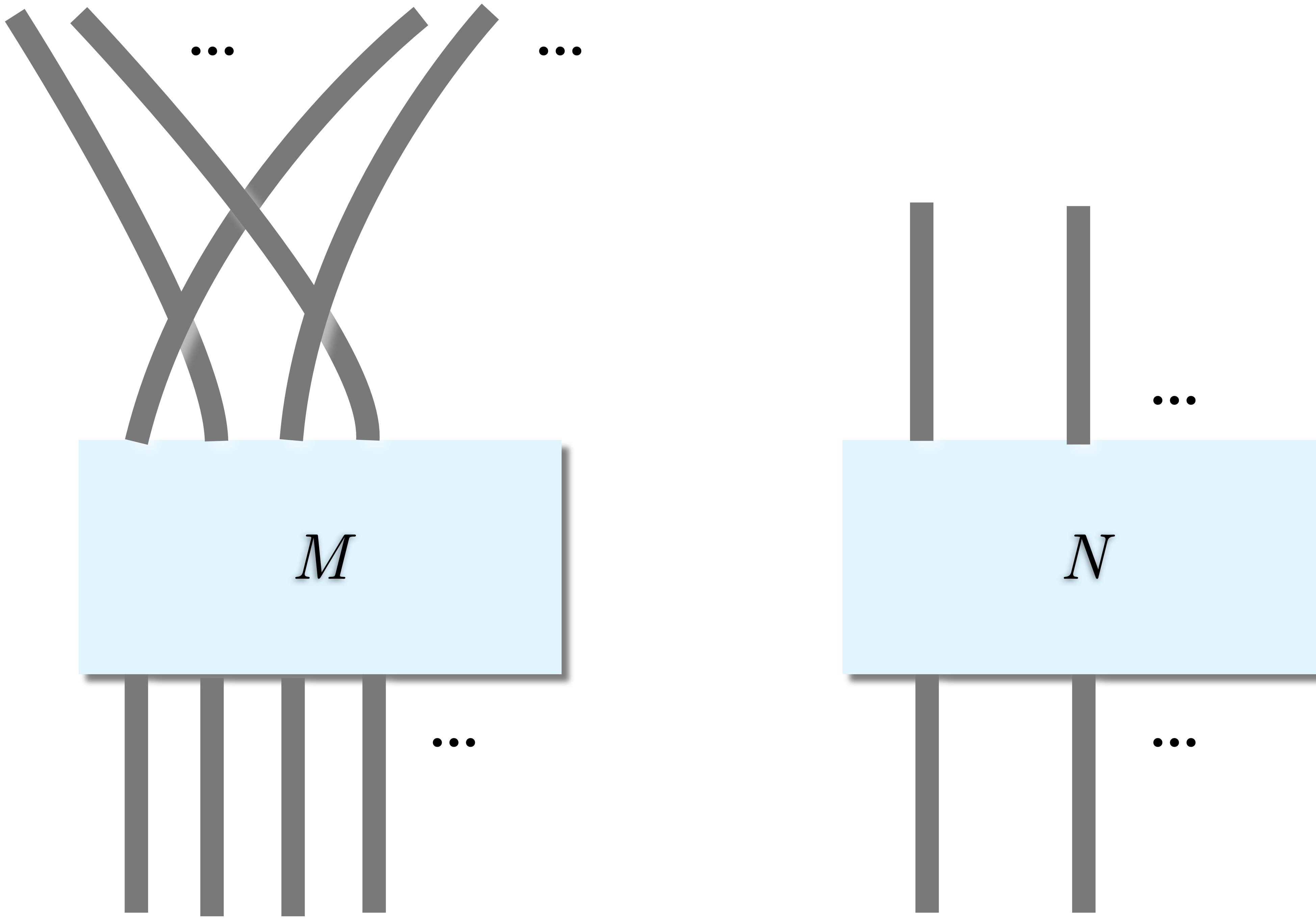


Hasuo (Tokyo), Hoshino (Kyoto)

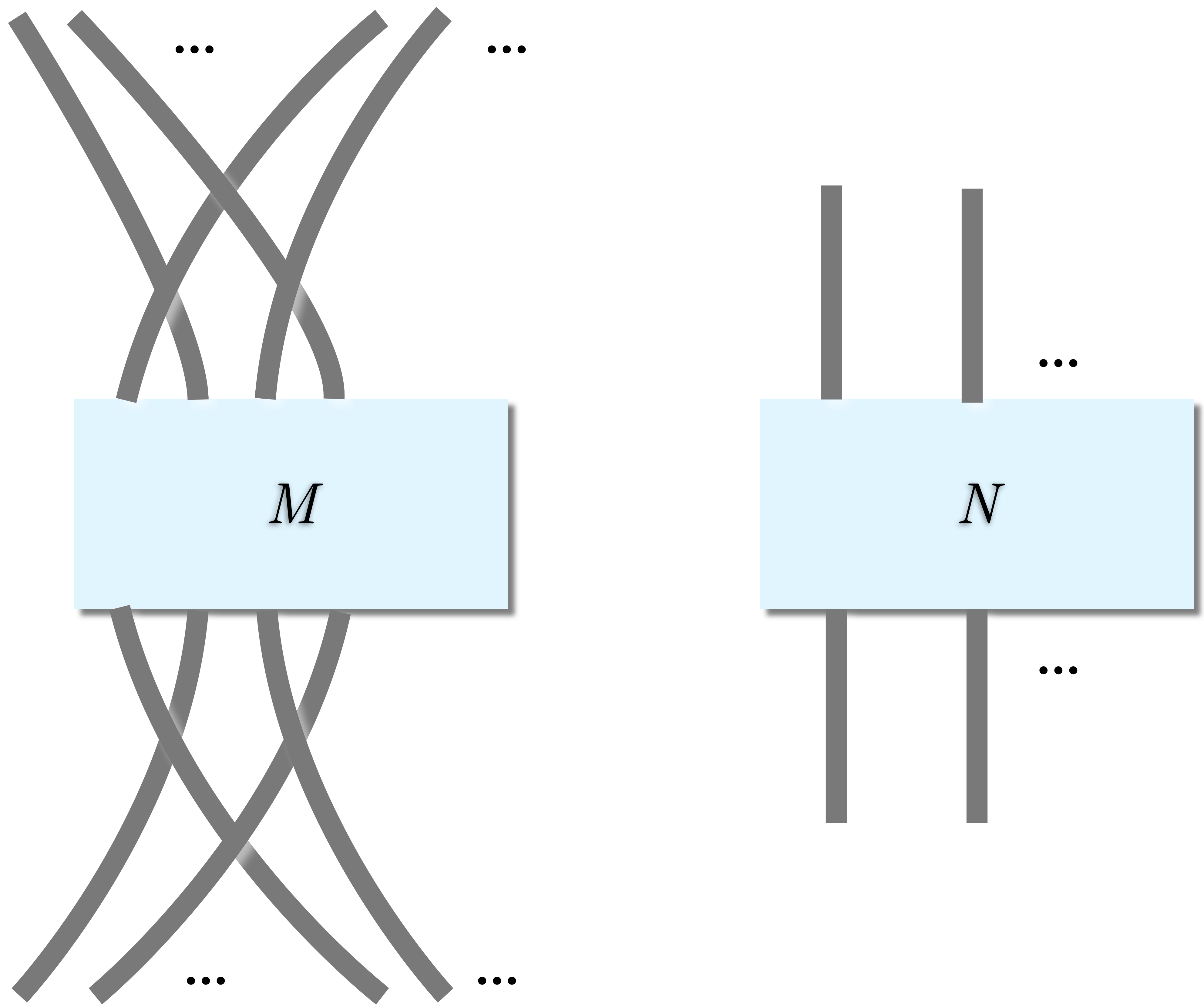
$$[\![\boldsymbol{M}\boldsymbol{N}]\!] =$$



$$[\![\boldsymbol{M}\boldsymbol{N}]\!] =$$

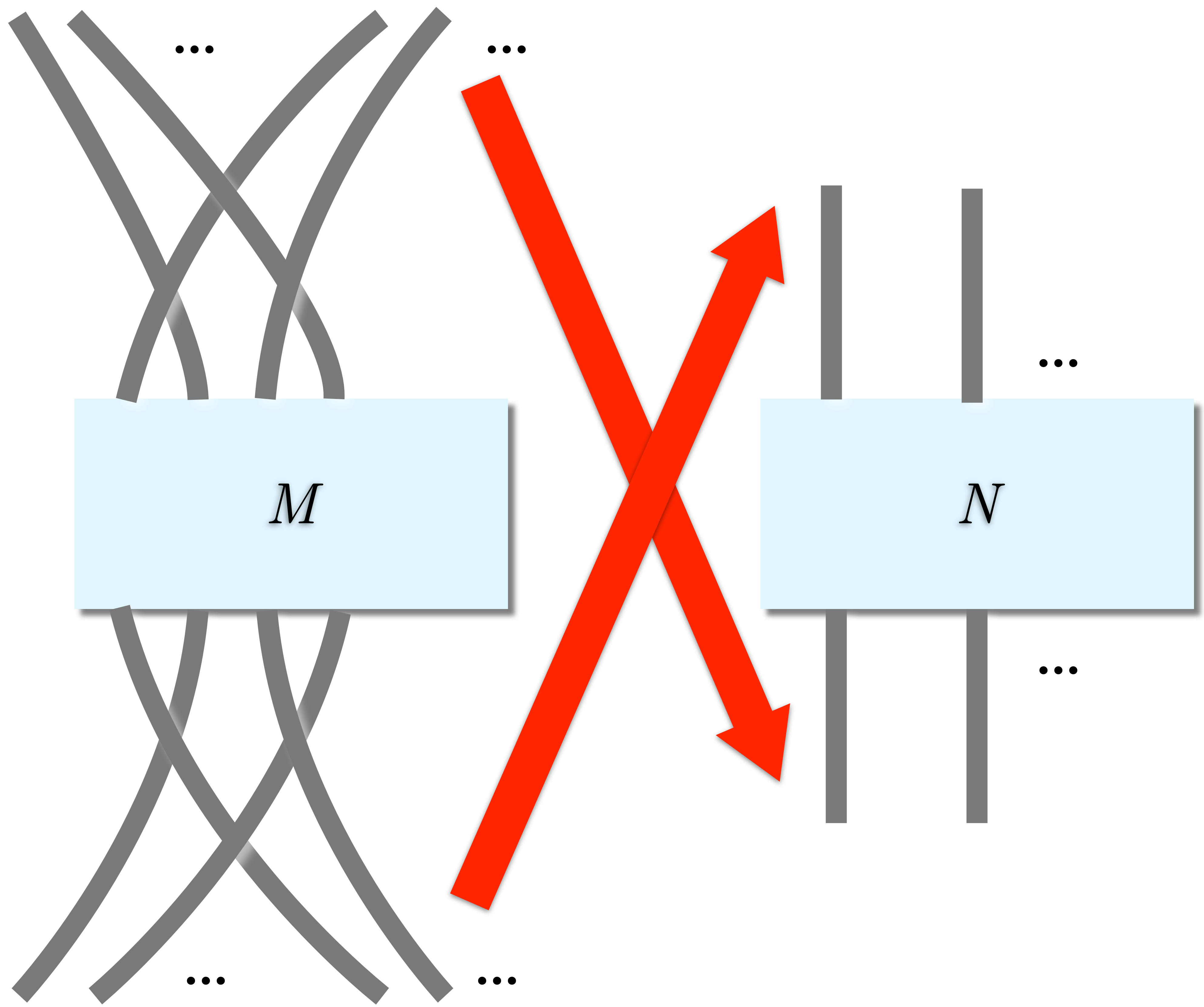


$$[\![\boldsymbol{M} \boldsymbol{N}]\!] =$$



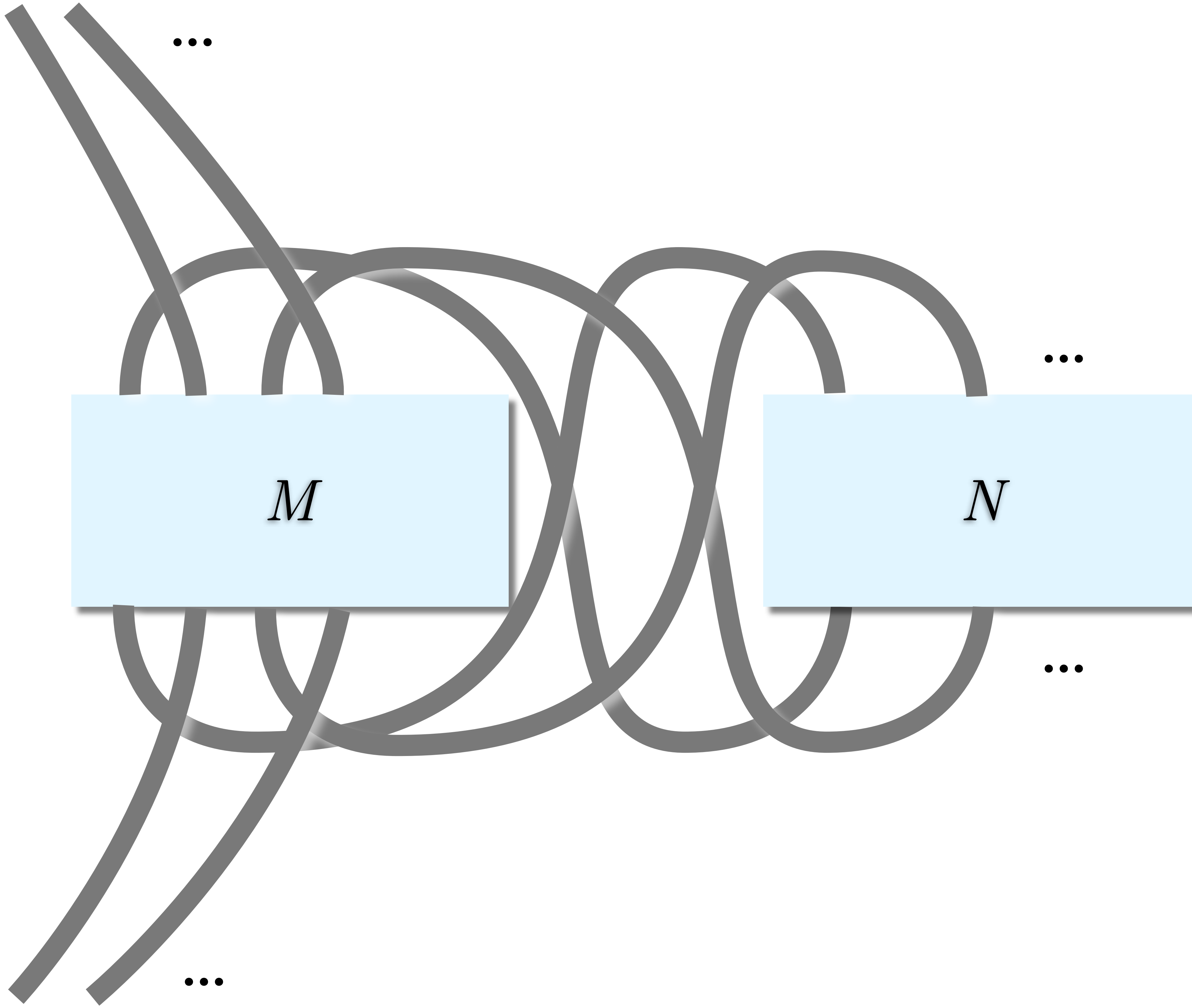
$[\![\boldsymbol{M}\,\boldsymbol{N}]\!]$

$=$

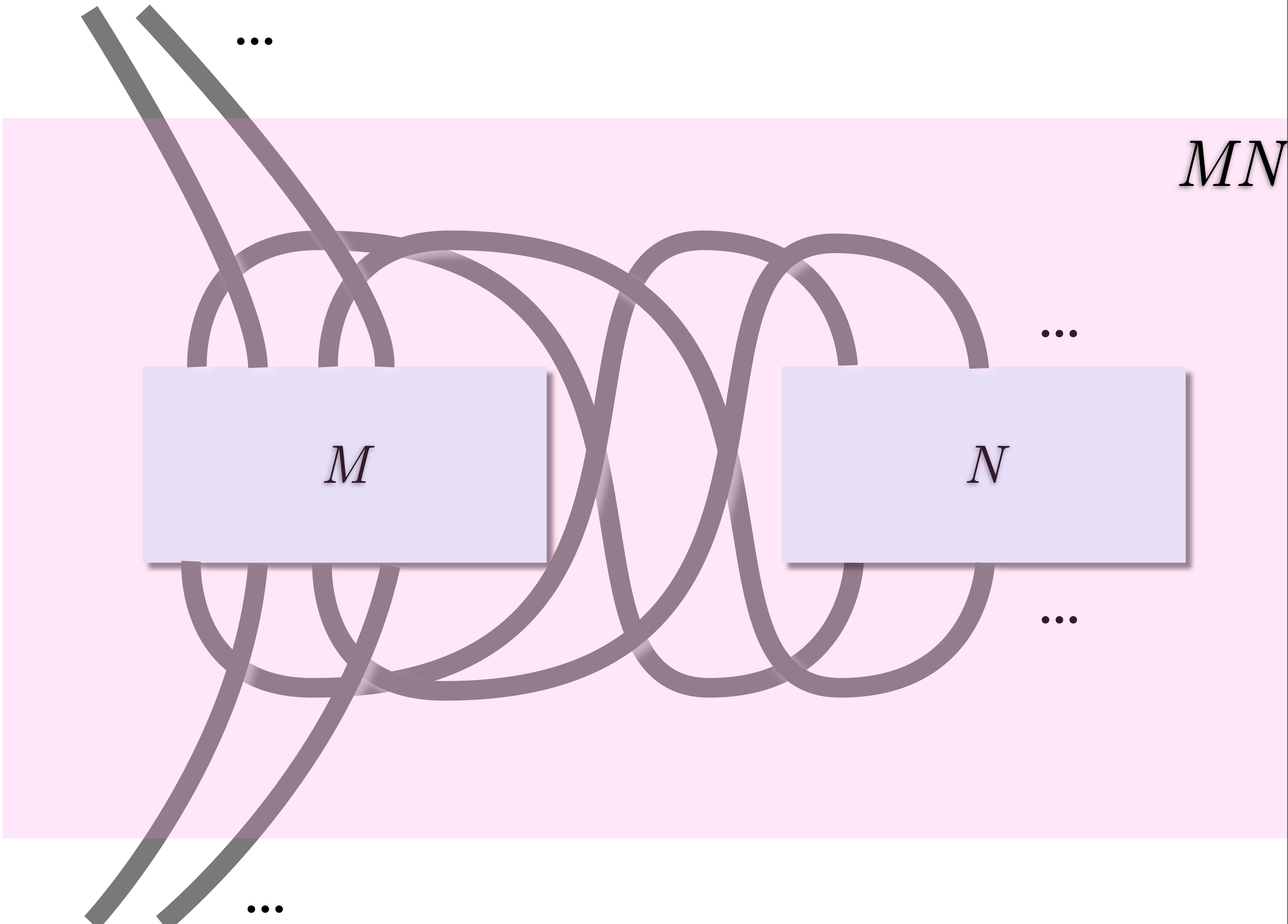


$[\![M N]\!]$

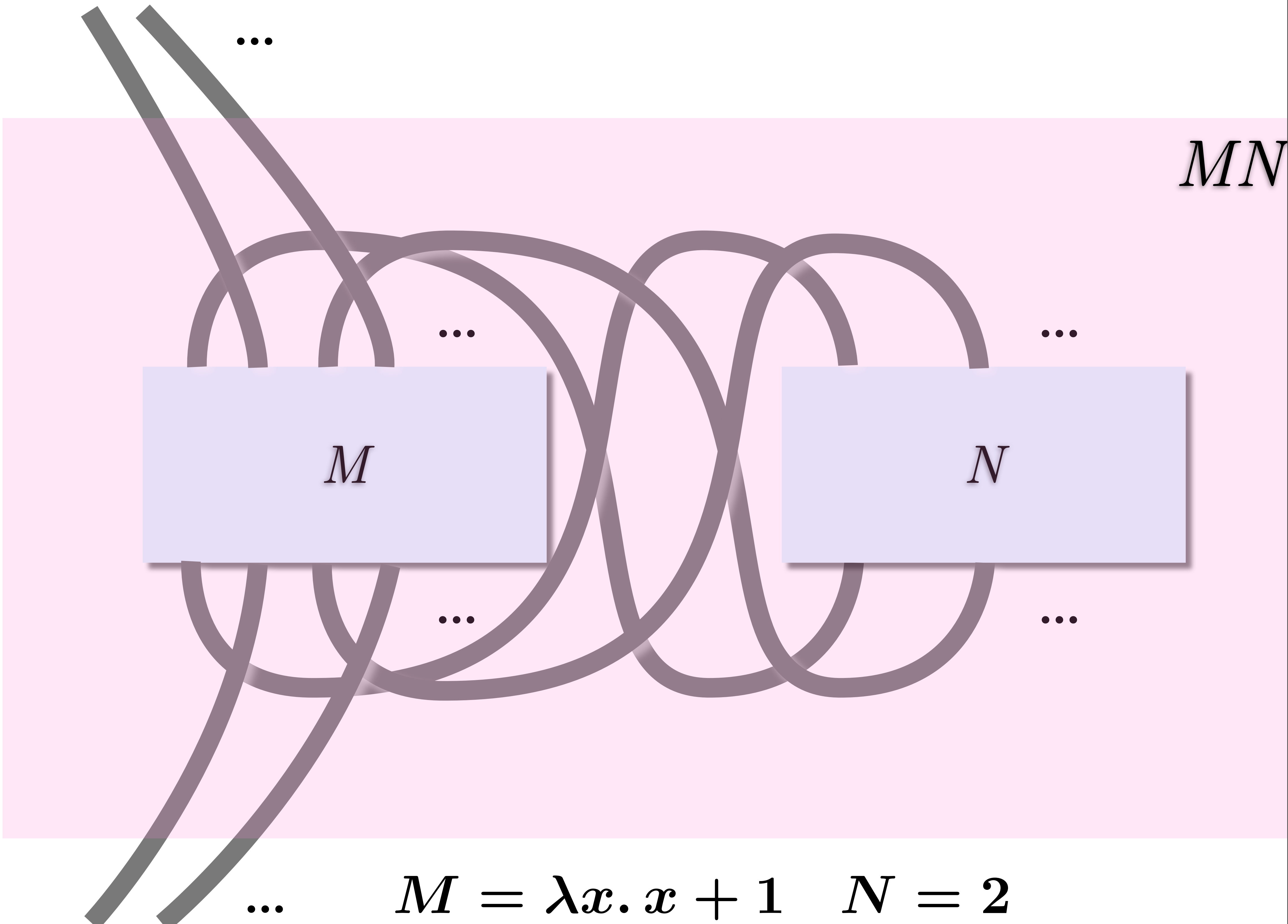
$=$



$$[\![\boldsymbol{MN}]\!] =$$

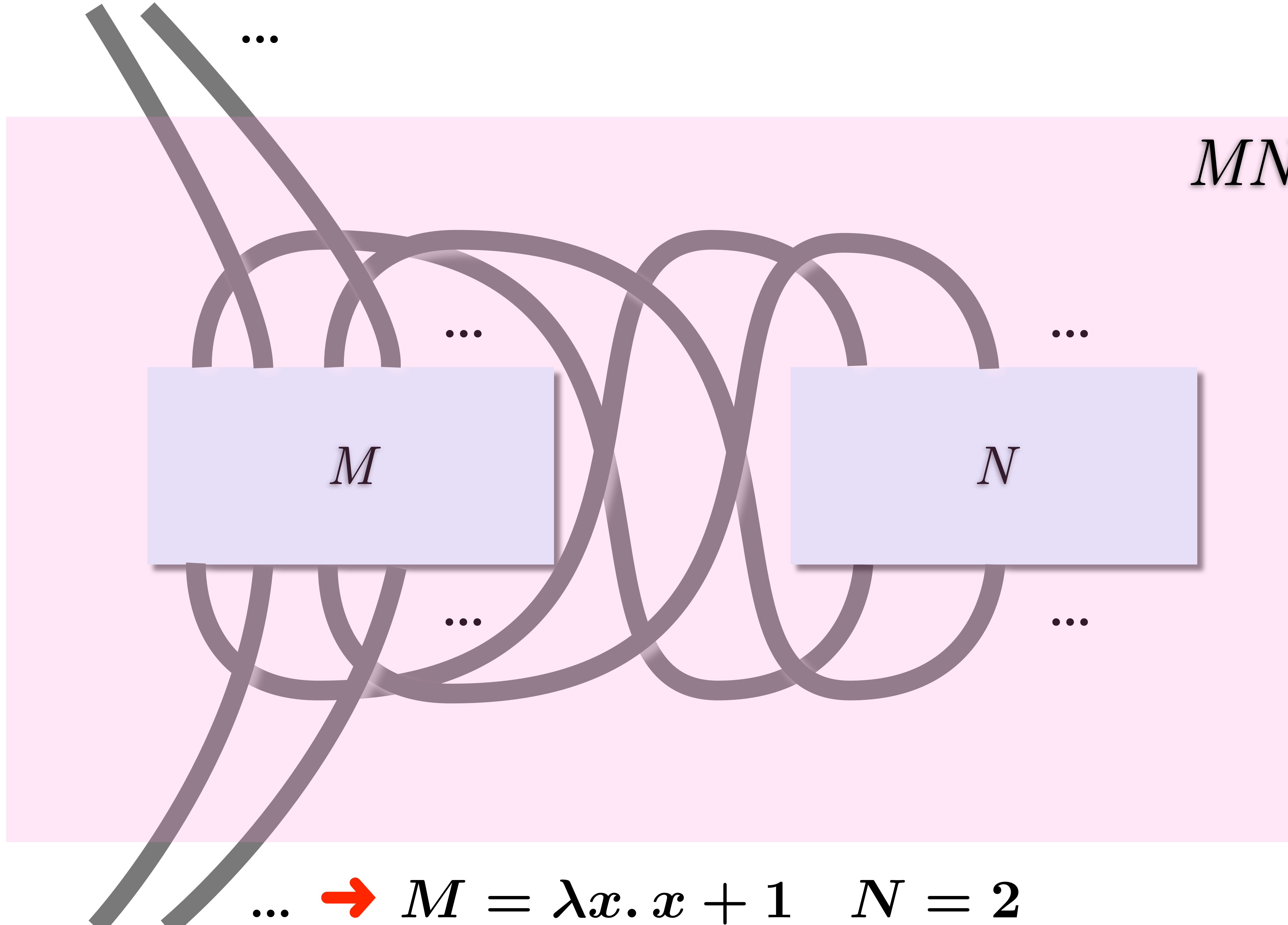


$[\![\boldsymbol{M\,N}]\!]$
$=$



$$\boldsymbol{M = \lambda x.\, x + 1 \quad N = 2}$$
$$\boldsymbol{M = \lambda x.\, 1 \quad N = 2}$$
$$\boldsymbol{M = \lambda f.\, f1 \quad N = \lambda x.\, (x + 1)}$$

$$[\![MN]\!] =$$

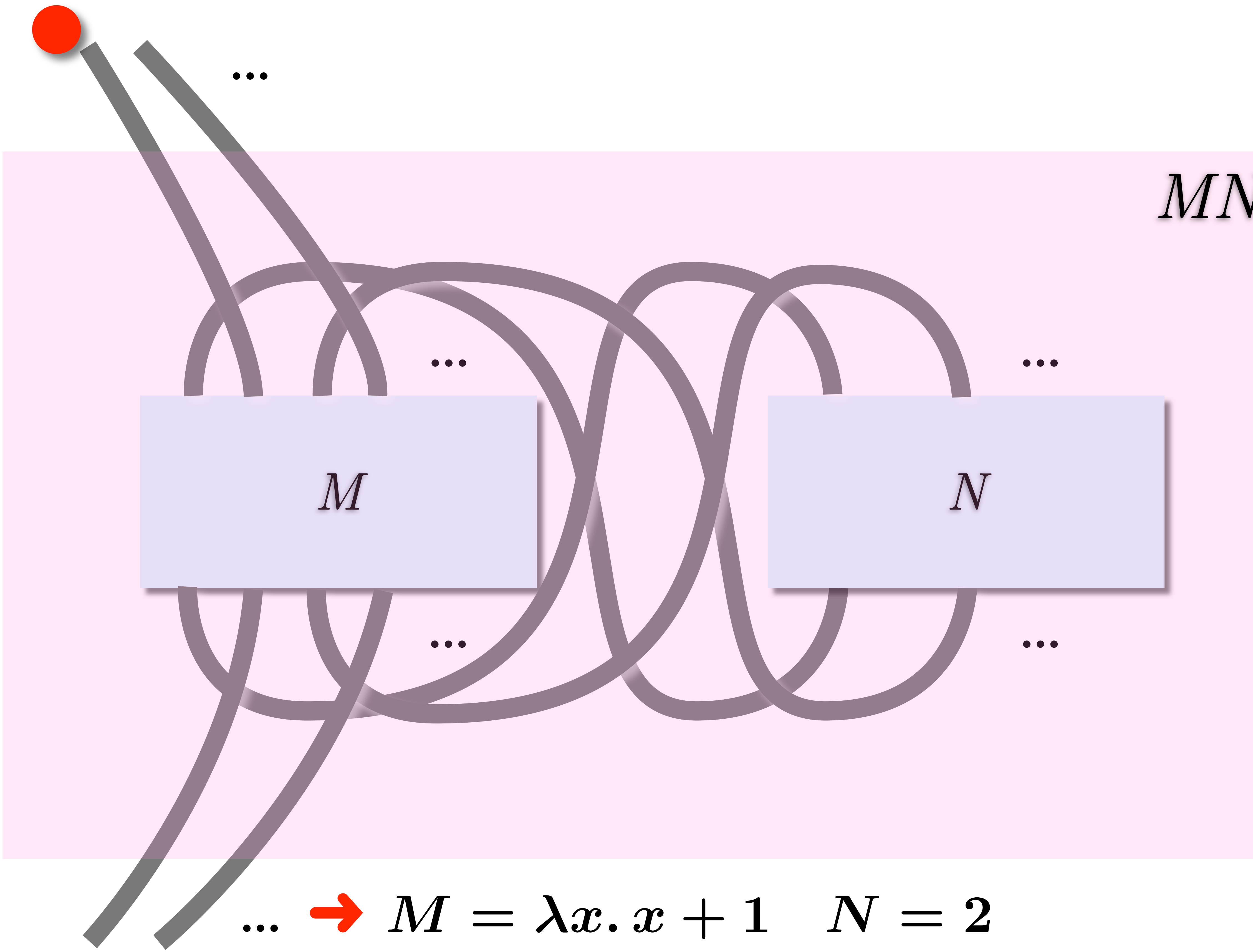


... ➜ $M = \lambda x.\, x + 1 \quad N = 2$

$M = \lambda x.\, 1 \quad N = 2$

$M = \lambda f.\, f 1 \quad N = \lambda x.\, (x + 1)$

$$[\![MN]\!] =$$

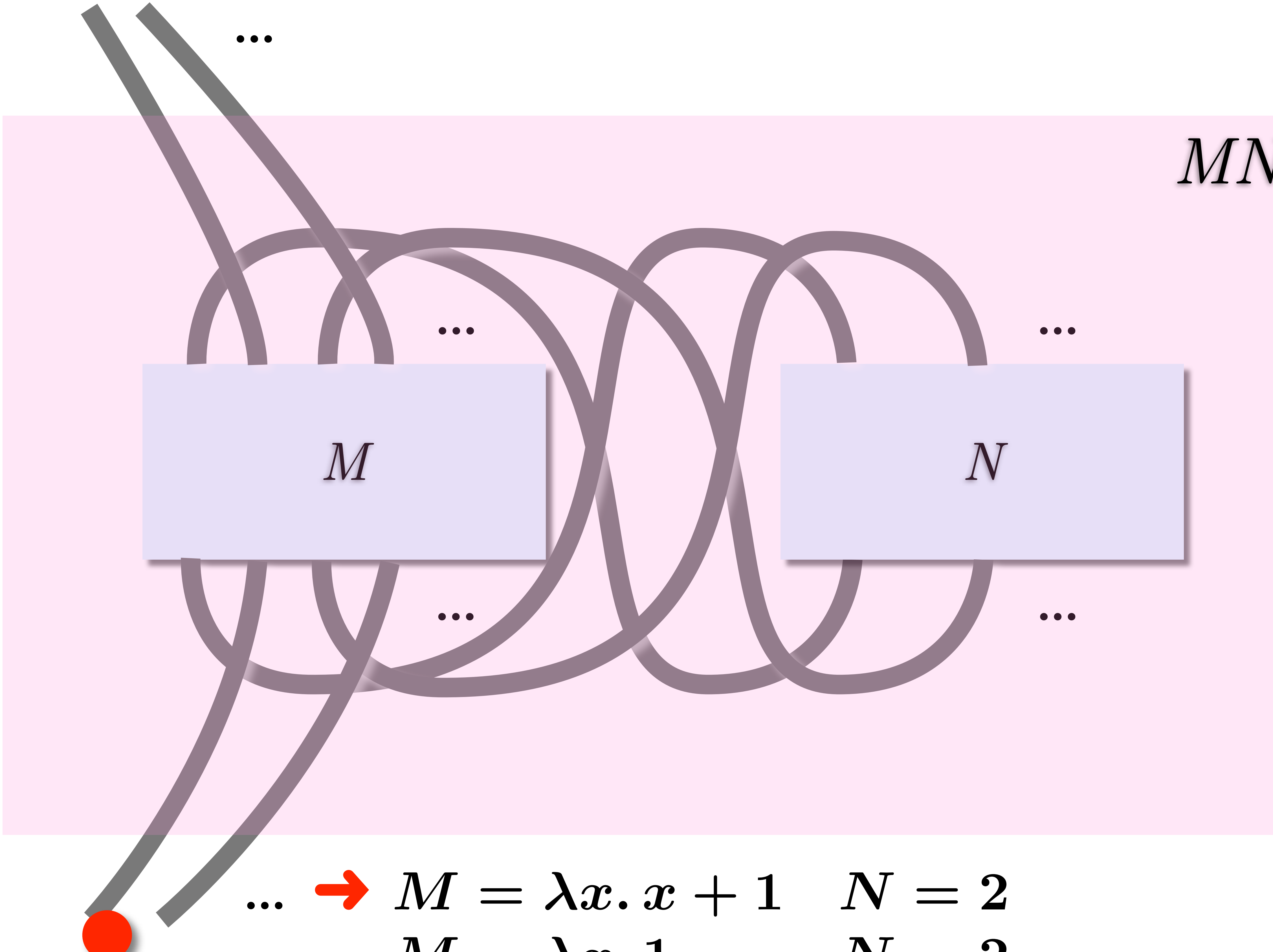


... ➜ $M = \lambda x.\, x + 1 \quad N = 2$

$M = \lambda x.\, 1 \quad N = 2$

$M = \lambda f.\, f 1 \quad N = \lambda x.\, (x + 1)$

$$[\![MN]\!] =$$

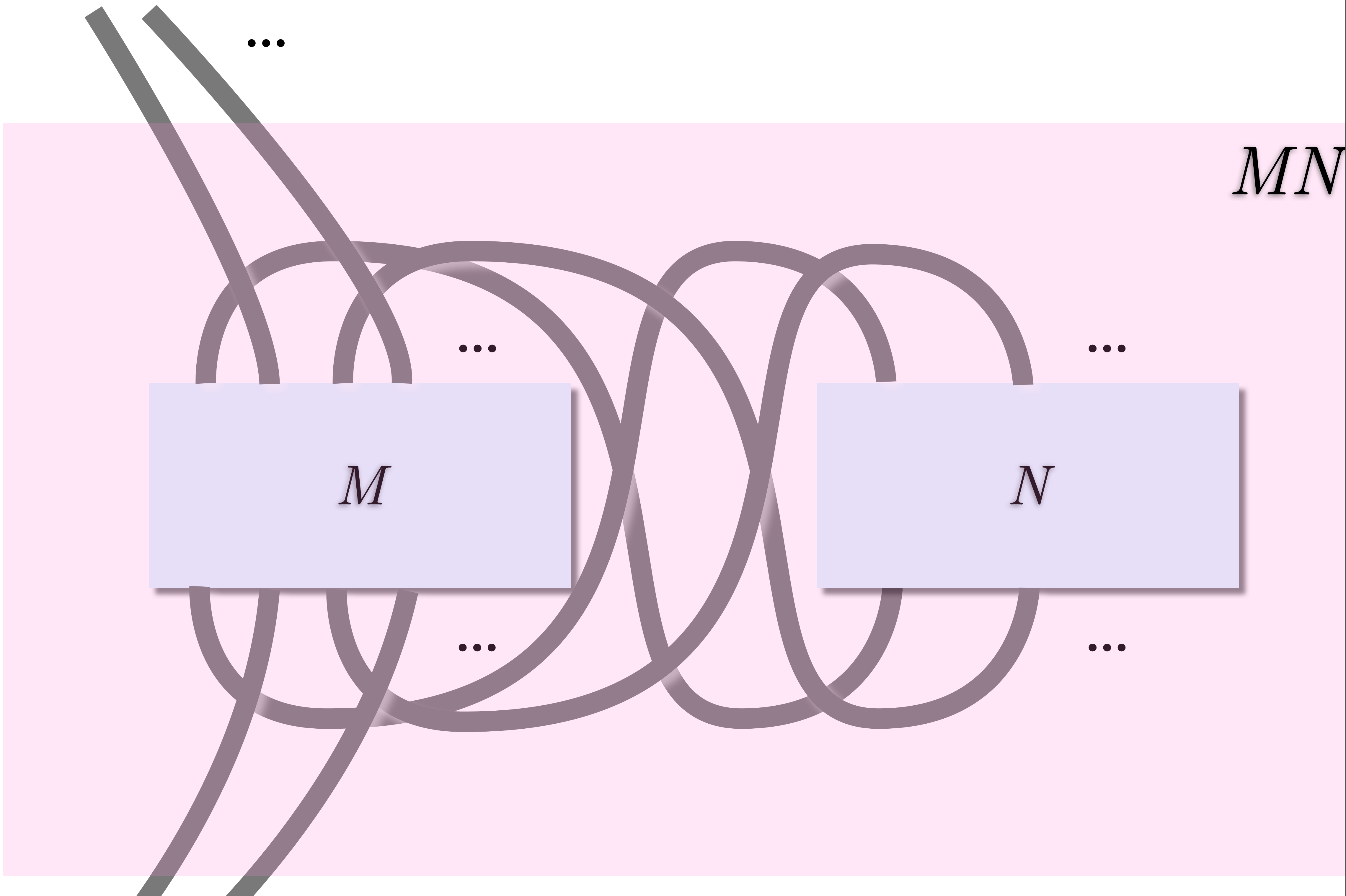


$M = \lambda x.\, x + 1 \quad N = 2$

$M = \lambda x.\, 1 \quad N = 2$

$M = \lambda f.\, f 1 \quad N = \lambda x.\, (x + 1)$

$$[\![\boldsymbol{M}\boldsymbol{N}]\!] =$$

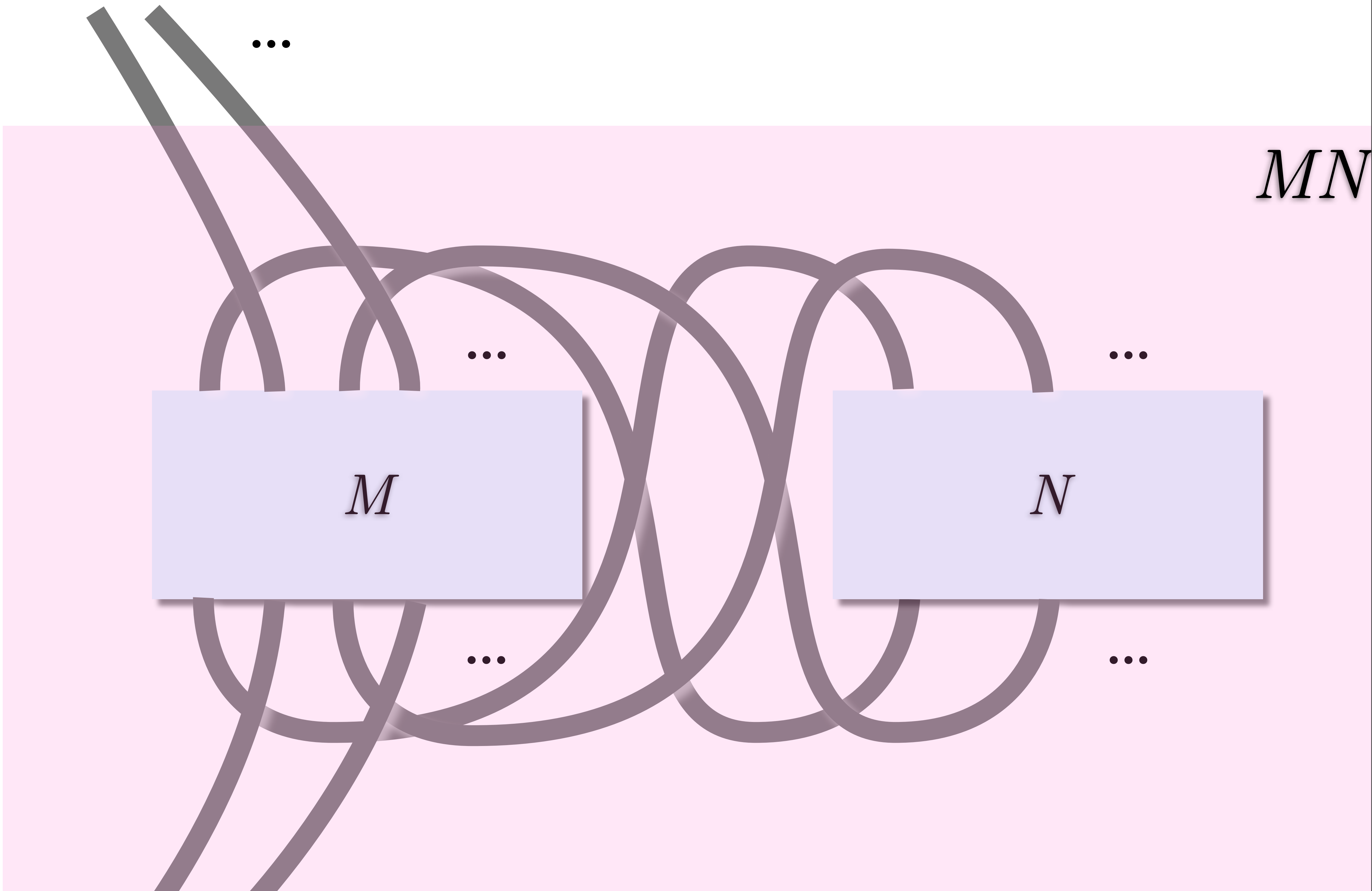


$$\boldsymbol{M = \lambda x.\, x + 1} \qquad \boldsymbol{N = 2}$$

$$\boldsymbol{M = \lambda x.\, 1} \qquad \boldsymbol{N = 2}$$

$$\boldsymbol{M = \lambda f.\, f 1} \qquad \boldsymbol{N = \lambda x.\, (x + 1)}$$

$$[\![MN]\!] =$$

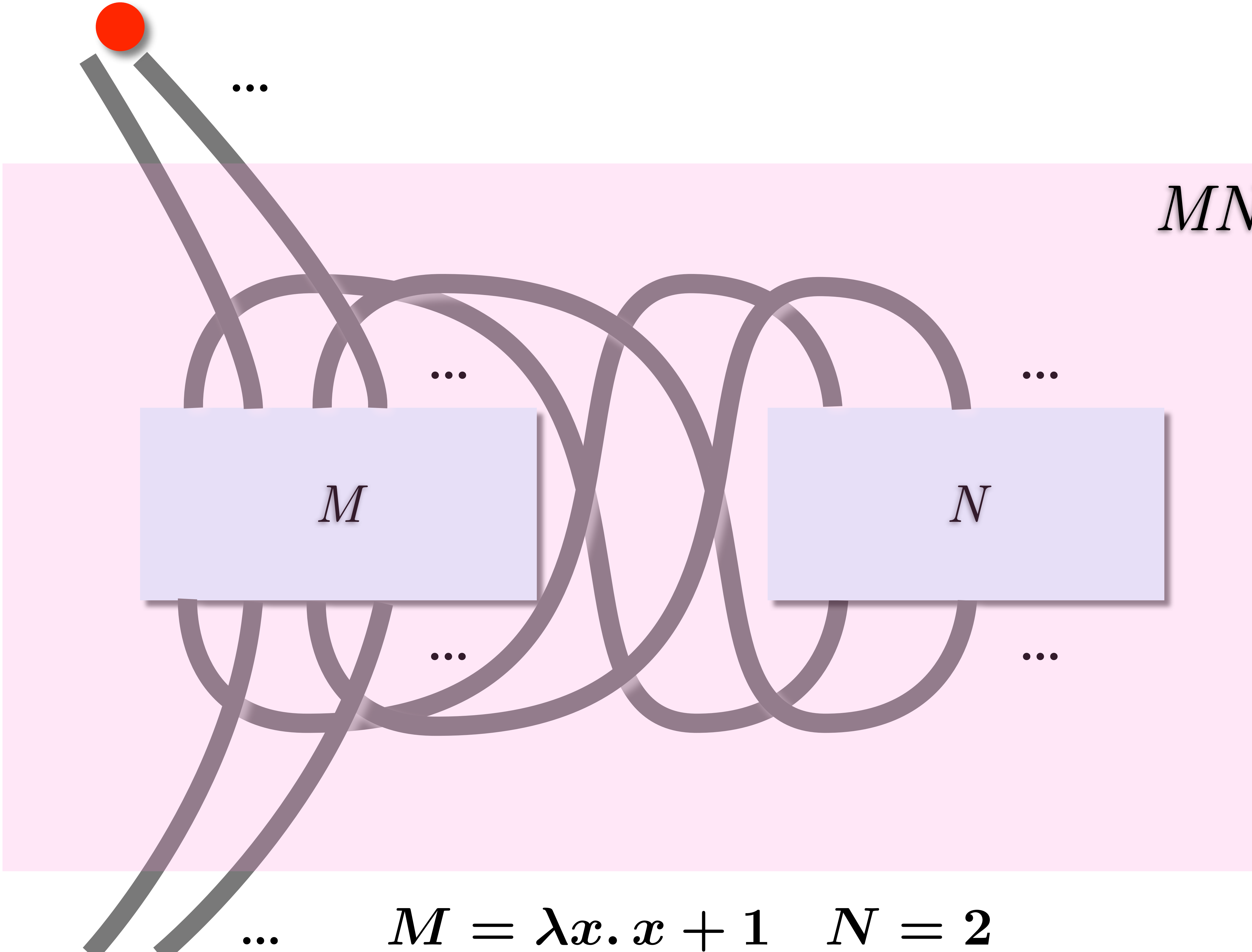


$M = \lambda x.\, x + 1 \quad N = 2$

➜ $M = \lambda x.\, 1 \quad N = 2$

$M = \lambda f.\, f1 \quad N = \lambda x.\, (x + 1)$

$[\![MN]\!] =$



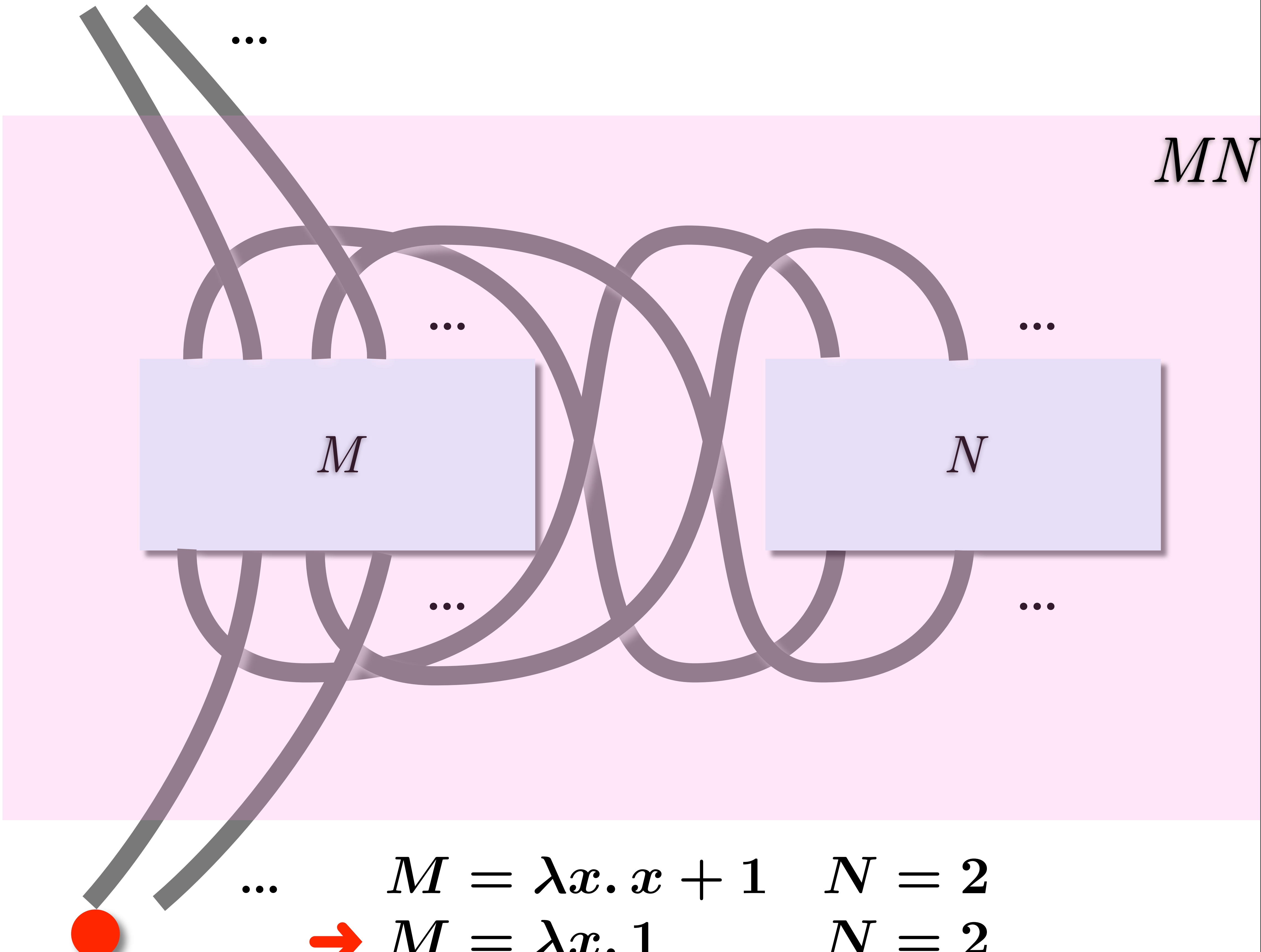
$$M = \lambda x.\, x + 1 \qquad N = 2$$

$$M = \lambda x.\, 1 \qquad N = 2$$

$$M = \lambda f.\, f 1 \qquad N = \lambda x.\, (x + 1)$$

$[\![MN]\!]$

$=$

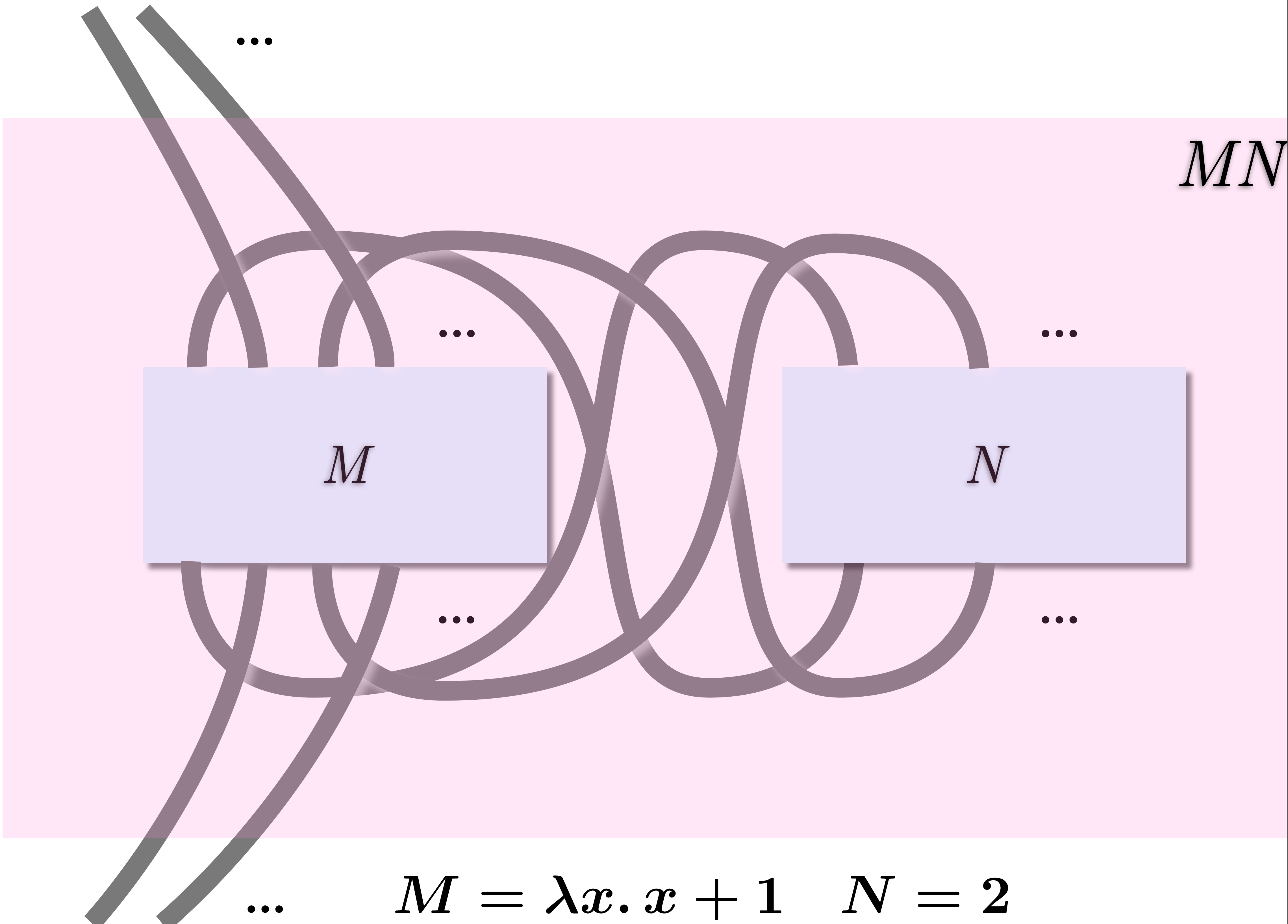


$M = \lambda x.\, x+1 \quad N = 2$

$\rightarrow M = \lambda x.\, 1 \quad N = 2$

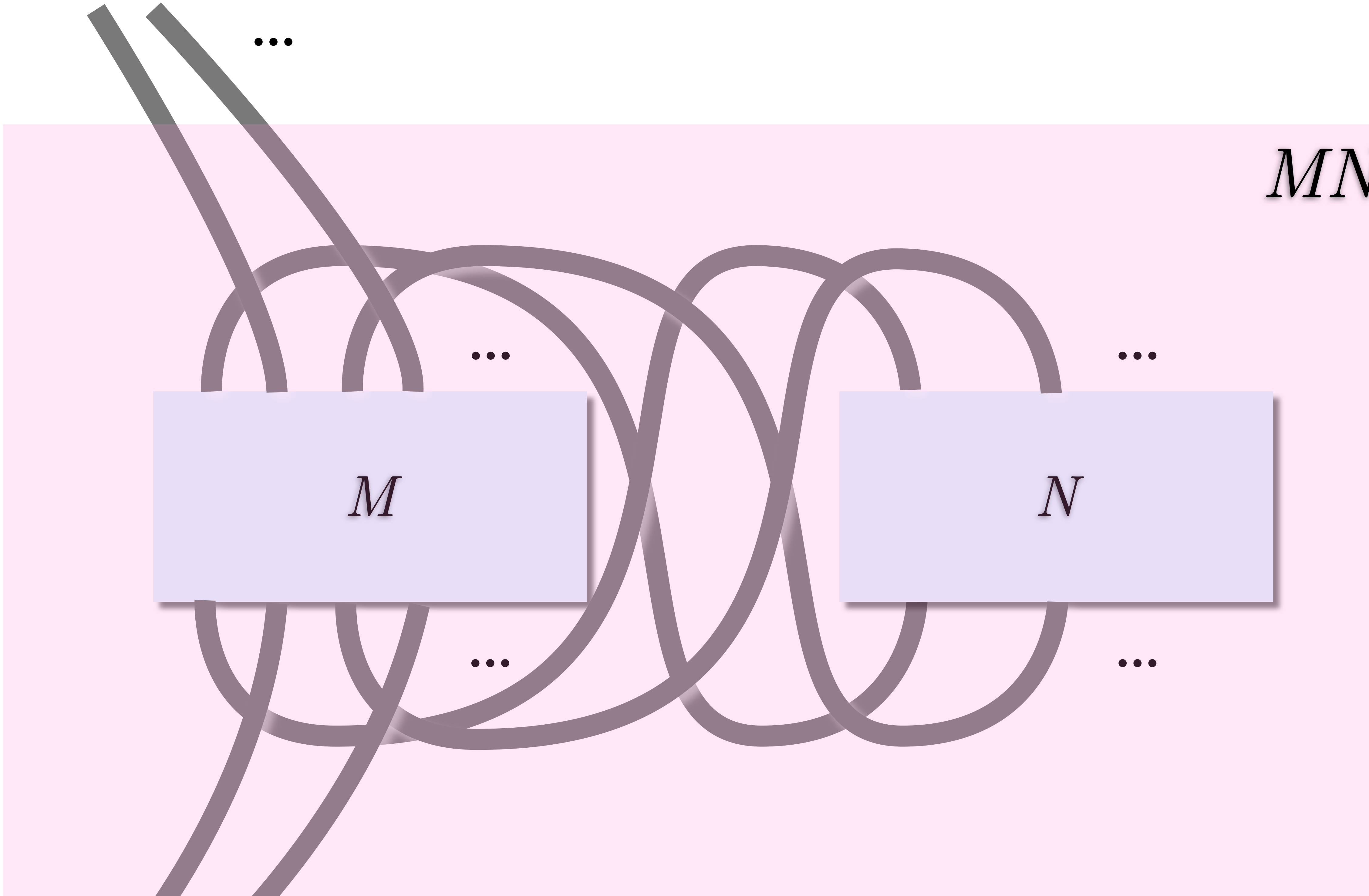
$M = \lambda f.\, f1 \quad N = \lambda x.\, (x+1)$

$$[\![\boldsymbol{MN}]\!] =$$



$$\boldsymbol{M = \lambda x.\, x + 1 \qquad N = 2}$$
$$\boldsymbol{M = \lambda x.\, 1 \qquad N = 2}$$
$$\boldsymbol{M = \lambda f.\, f 1 \qquad N = \lambda x.\, (x + 1)}$$

$$[\![M\,N]\!] =$$



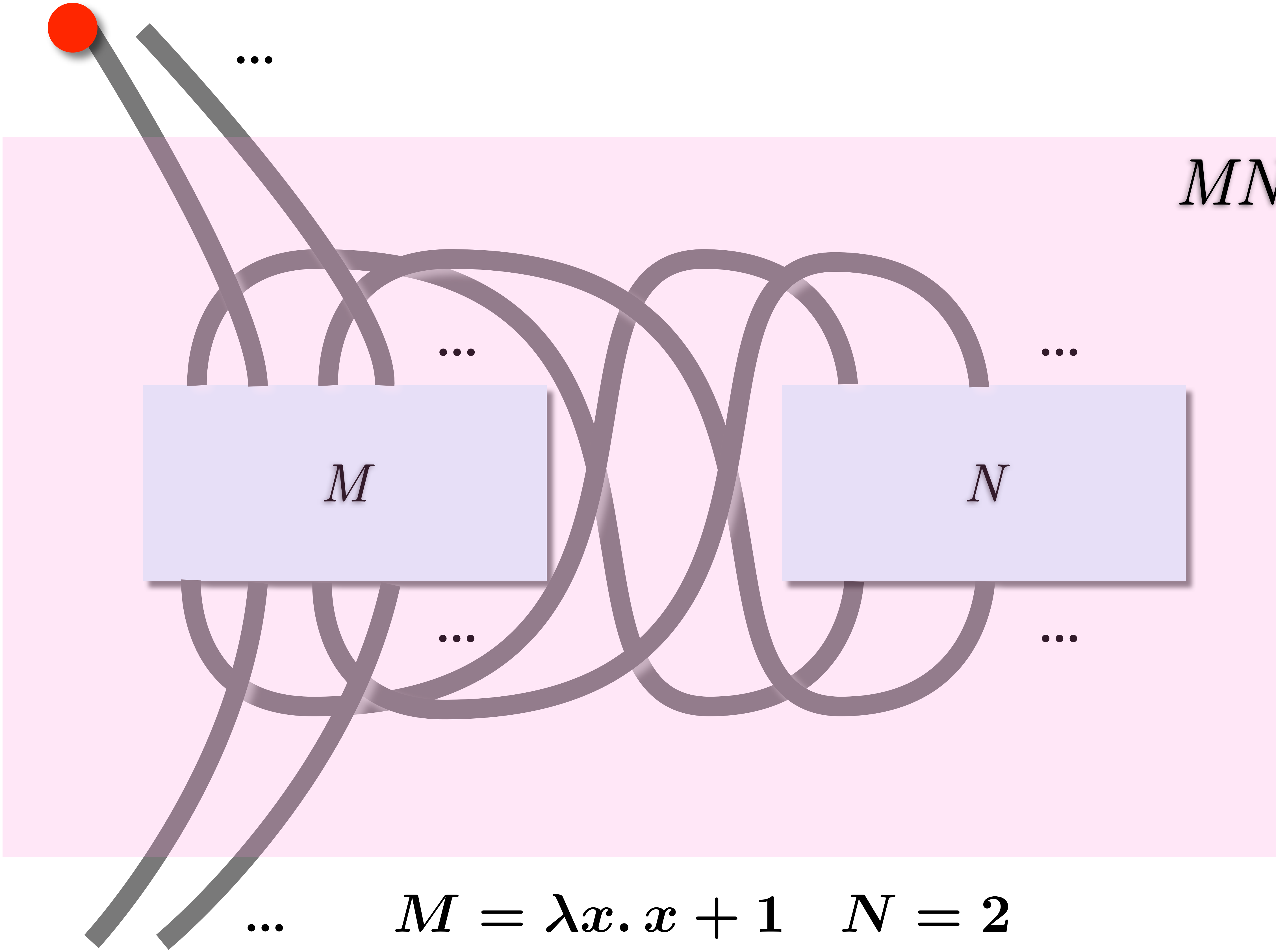
$$M = \lambda x.\, x + 1 \qquad N = 2$$

$$M = \lambda x.\, 1 \qquad N = 2$$

$$\rightarrow\ M = \lambda f.\, f 1 \qquad N = \lambda x.\, (x + 1)$$

$[\![MN]\!]$

$=$

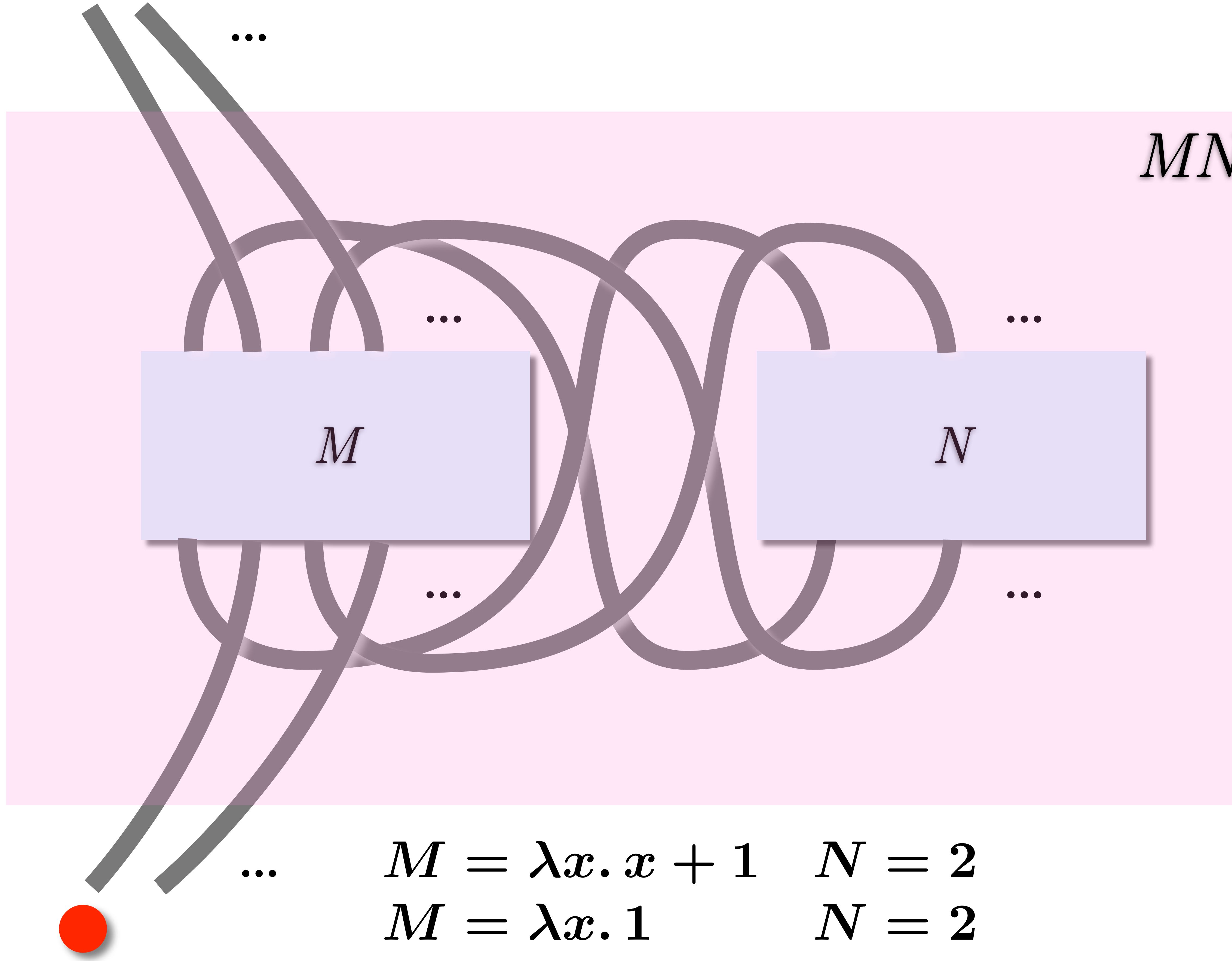


$M = \lambda x.\, x + 1 \quad N = 2$

$M = \lambda x.\, 1 \quad N = 2$

$\rightarrow M = \lambda f.\, f 1 \quad N = \lambda x.\, (x + 1)$

$$[\![MN]\!] =$$

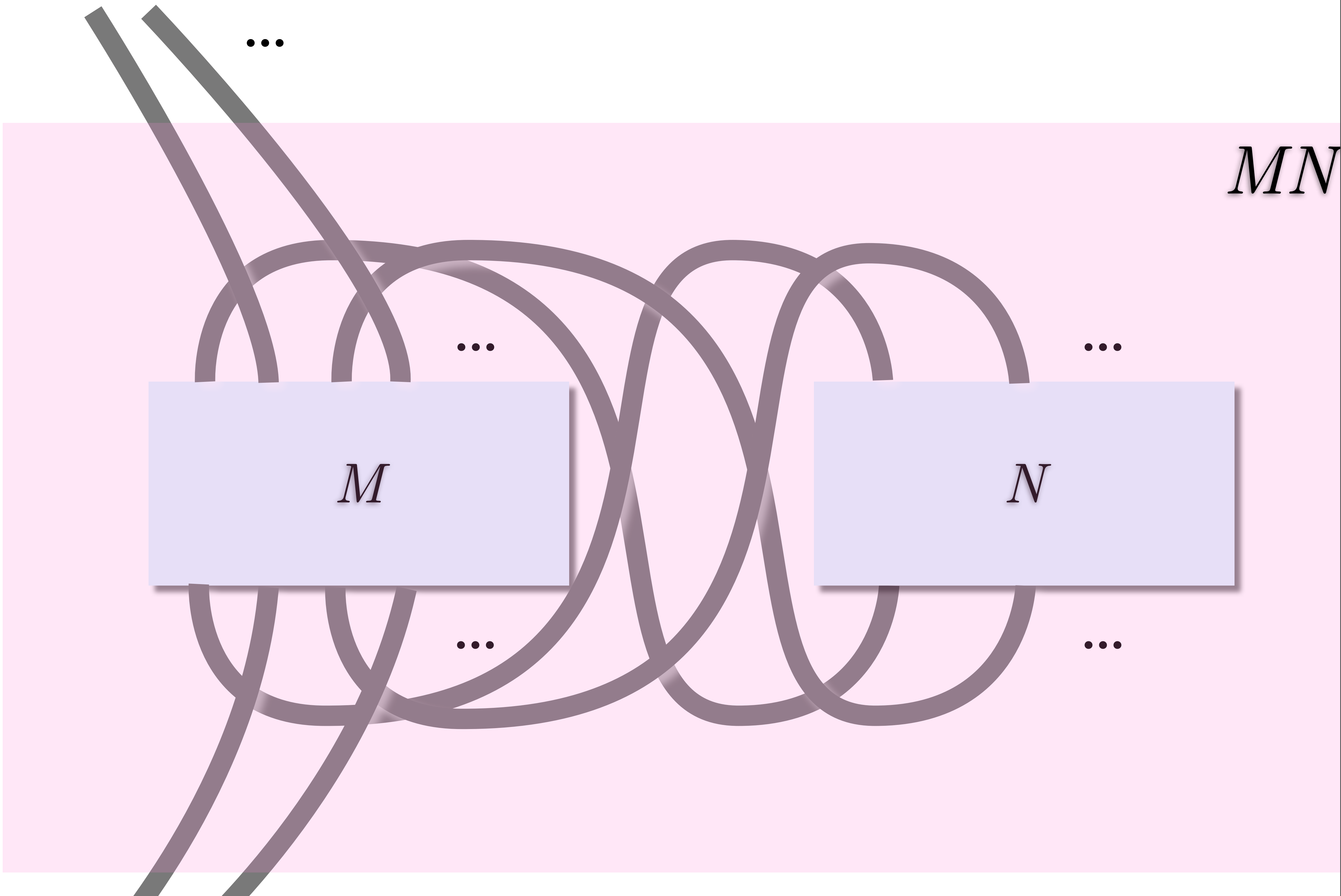


$$M = \lambda x.\, x + 1 \qquad N = 2$$

$$M = \lambda x.\, 1 \qquad N = 2$$

$$\rightarrow\ M = \lambda f.\, f1 \qquad N = \lambda x.\, (x + 1)$$

$$[\![M N]\!] =$$

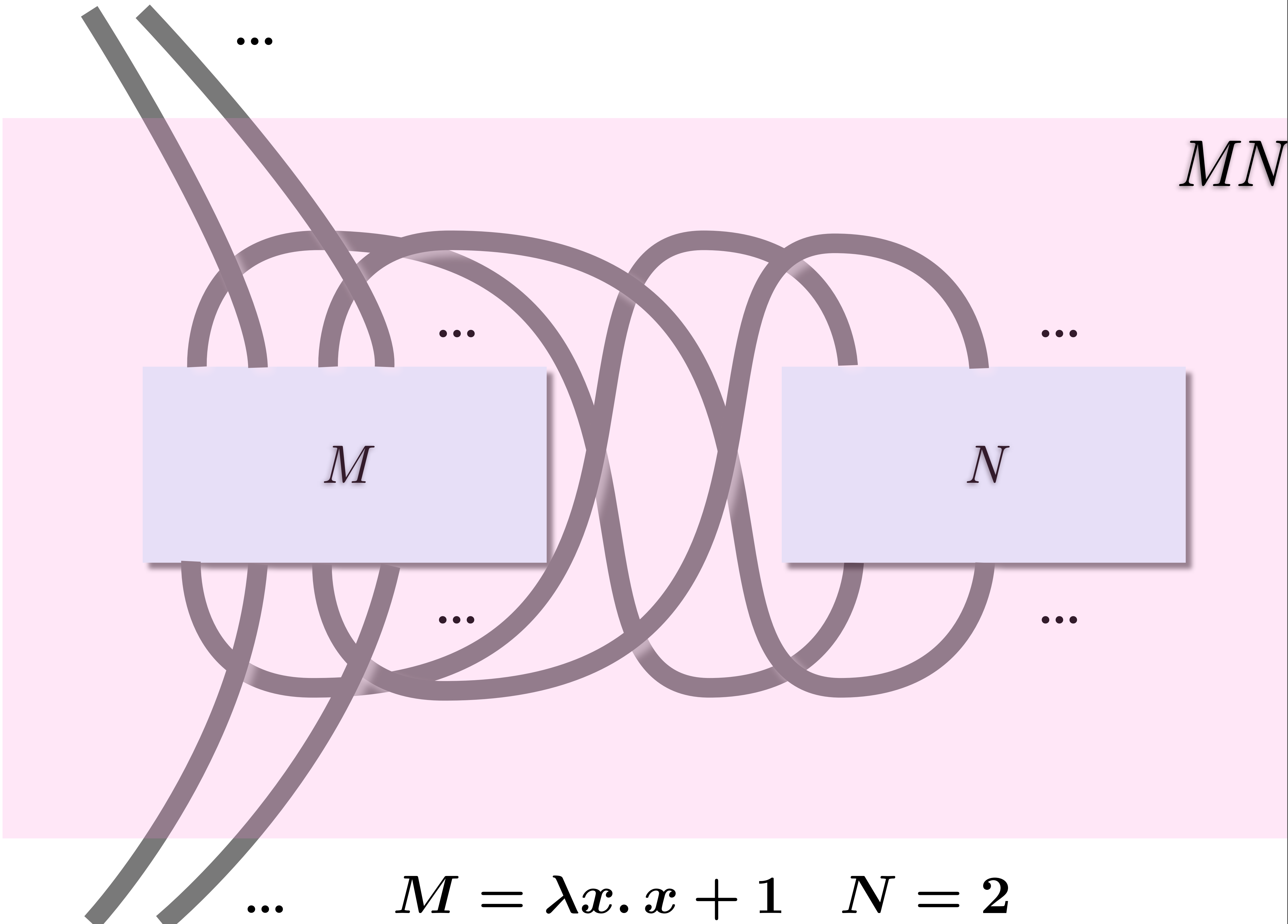


$$M = \lambda x.\, x + 1 \qquad N = 2$$

$$M = \lambda x.\, 1 \qquad N = 2$$

$$\rightarrow \; M = \lambda f.\, f 1 \qquad N = \lambda x.\, (x + 1)$$

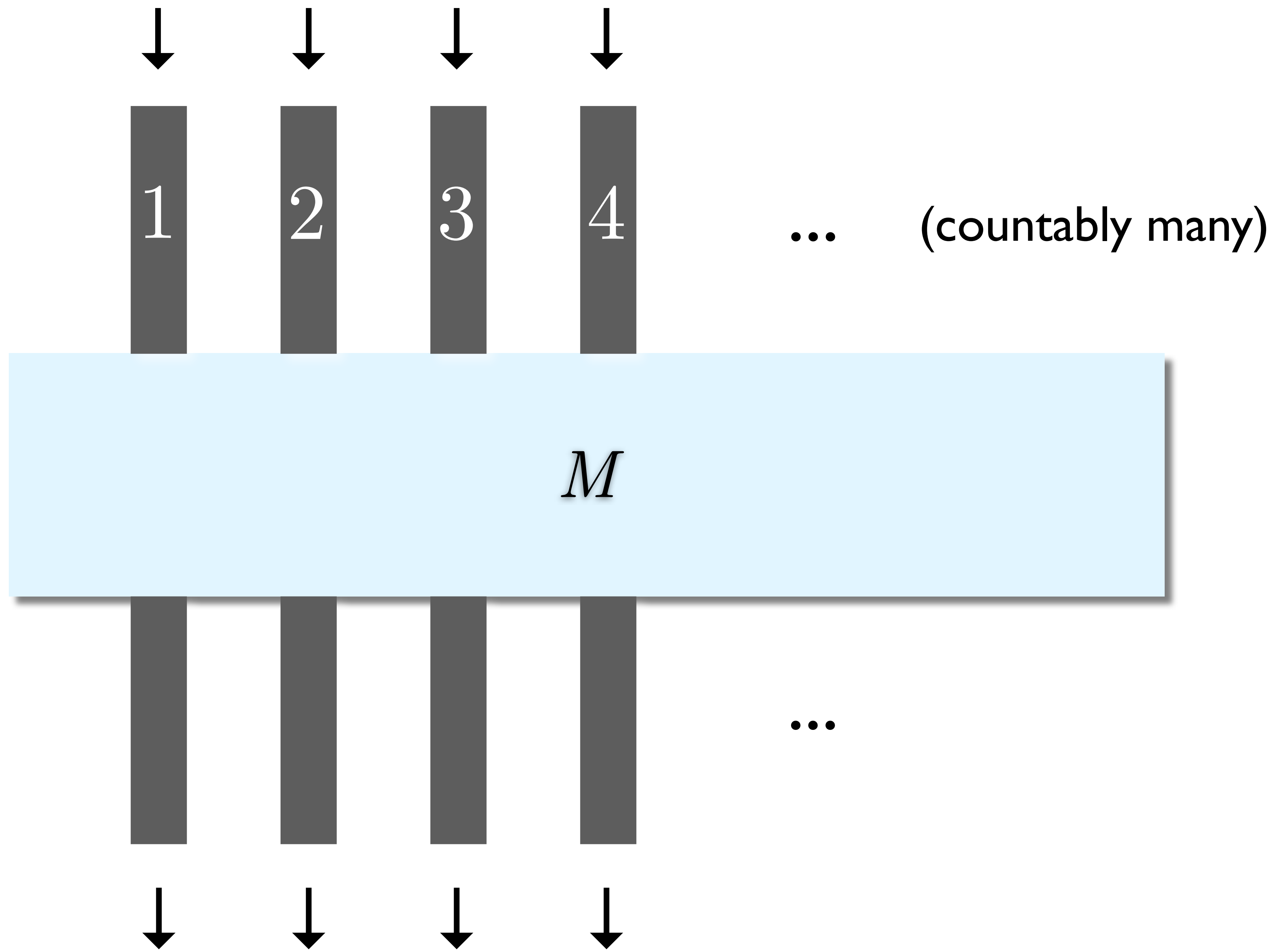
$$[\![\boldsymbol{MN}]\!] =$$



$$\boldsymbol{M = \lambda x.\, x+1 \quad N = 2}$$
$$\boldsymbol{M = \lambda x.\, 1 \quad N = 2}$$
$$\boldsymbol{M = \lambda f.\, f1 \quad N = \lambda x.\,(x+1)}$$

Quantum
Geometry of Interaction

$$[\![M]\!] =$$

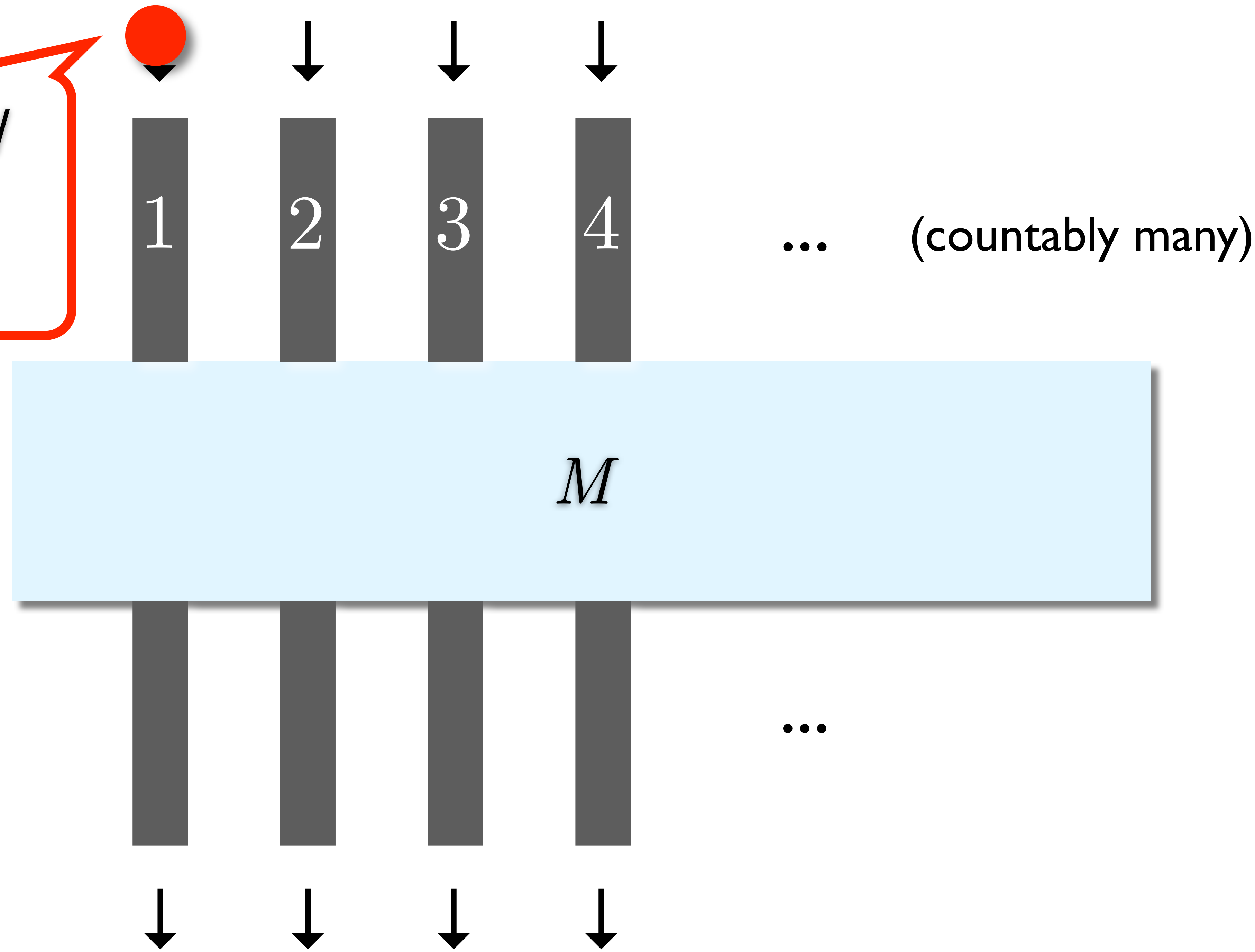


Hasuo (Tokyo), Hoshino (Kyoto)

Quantum
Geometry of Interaction

Not just a token/ particle, but *quantum state!*

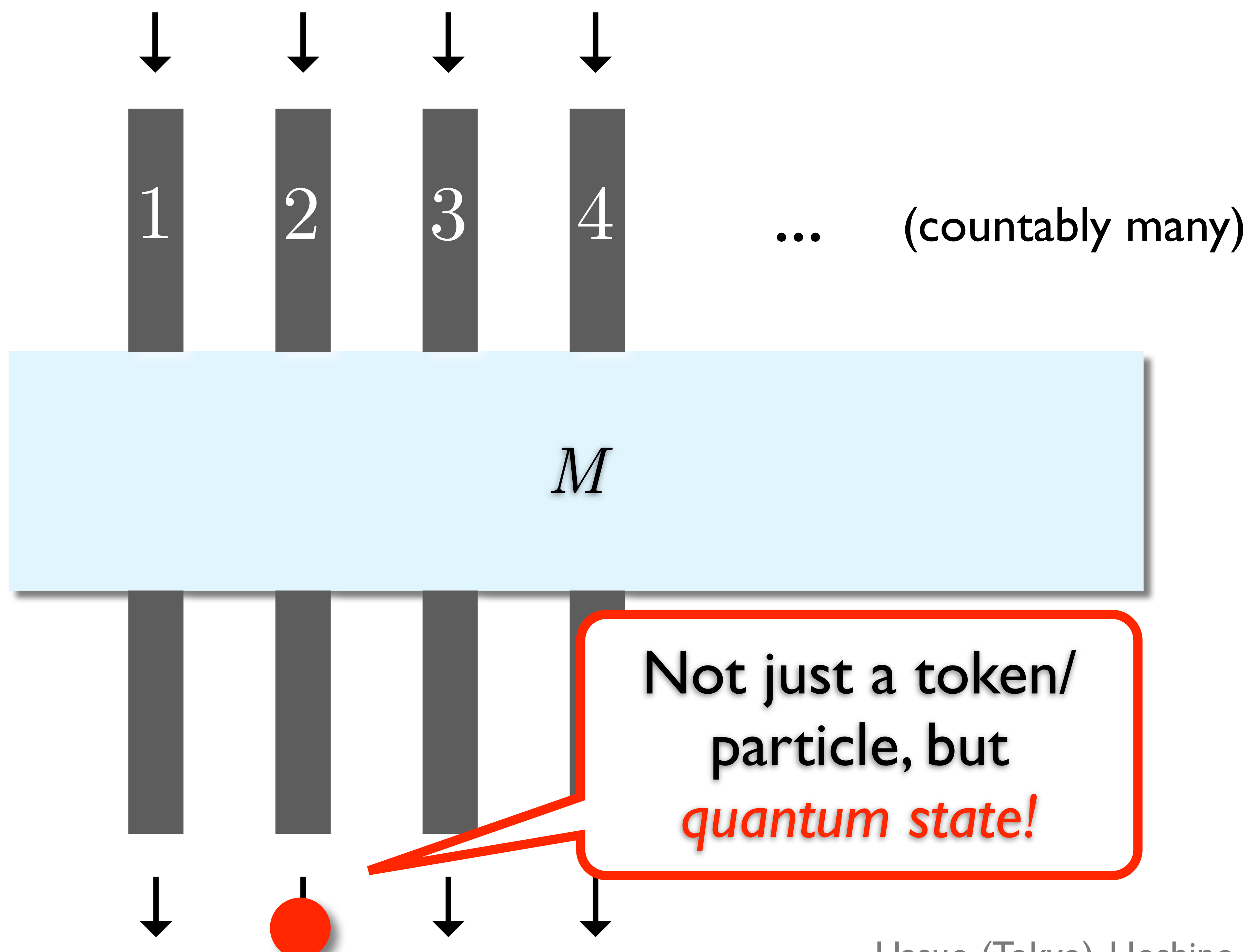
$$[\![M]\!] =$$



Hasuo (Tokyo), Hoshino (Kyoto)

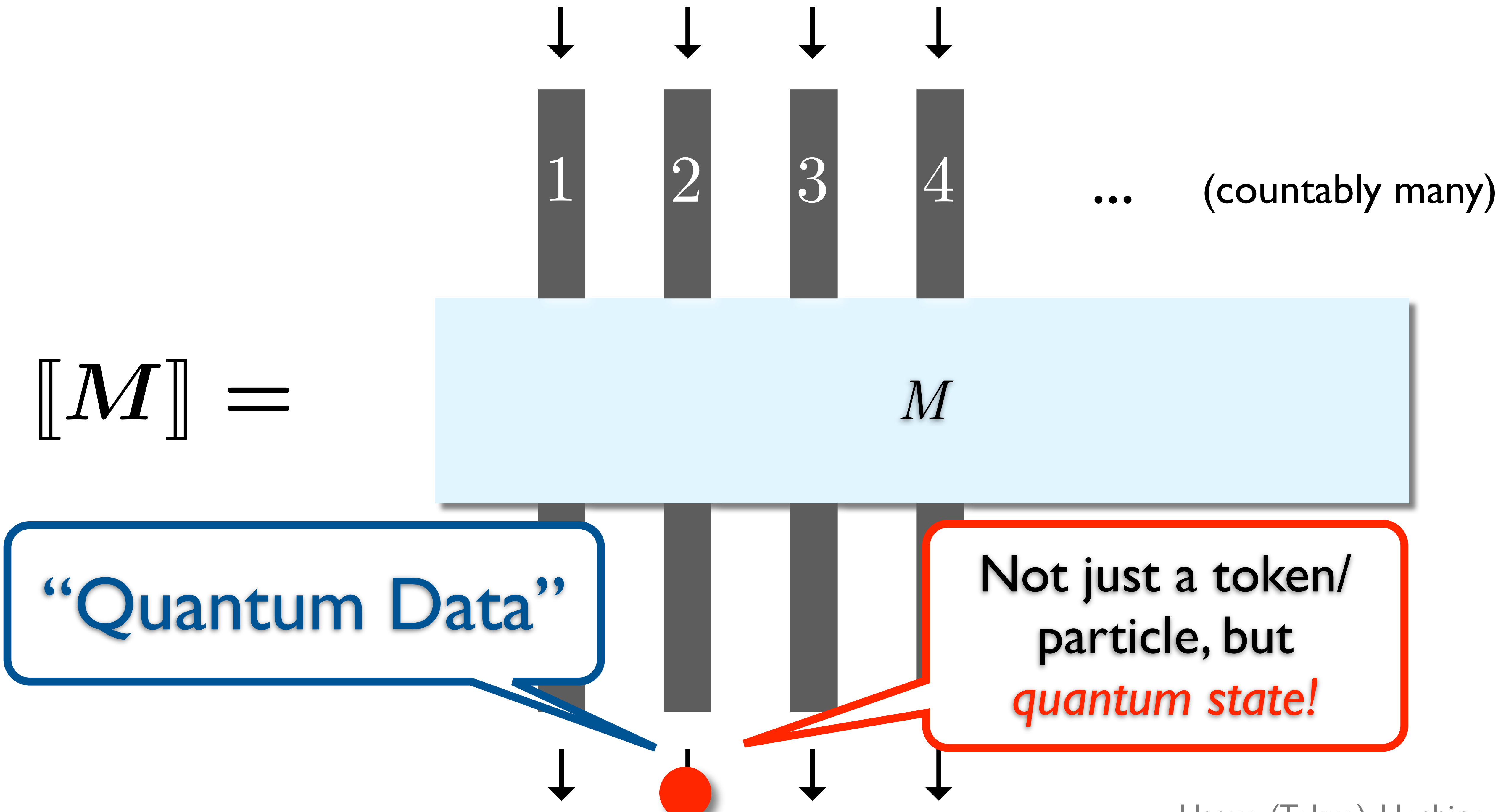
Quantum
Geometry of Interaction

$$[\![M]\!] =$$



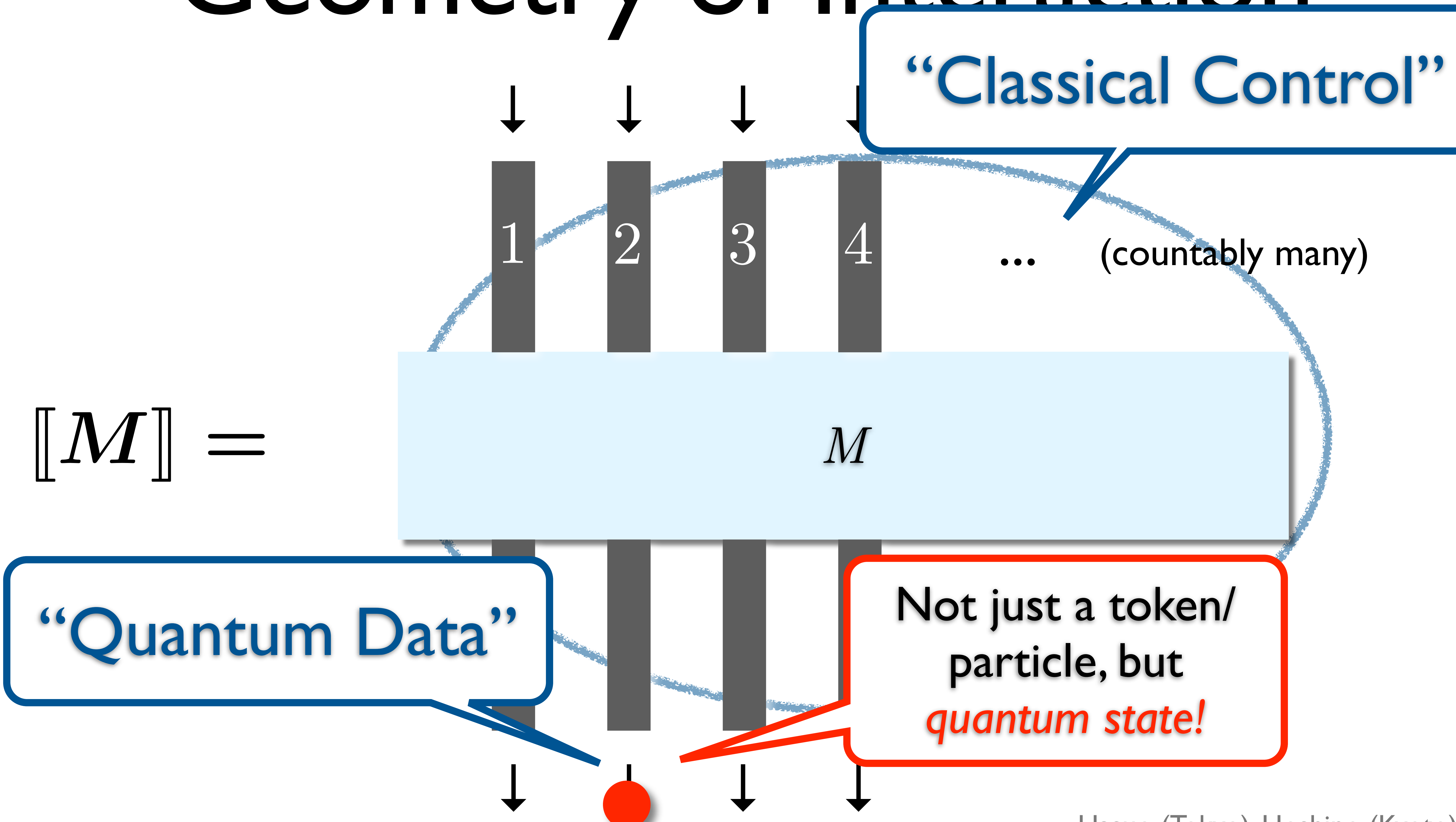
Hasuo (Tokyo), Hoshino (Kyoto)

Quantum
Geometry of Interaction



Hasuo (Tokyo), Hoshino (Kyoto)

Quantum Geometry of Interaction



Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

Q5. Why quantum computation?

Ans. You know why!

Conclusions

- Structured programming & mathematical semantics
- Quantum data, classical control
  - *Geometry of interaction* as the essence of classical control

Conclusions

- Structured programming & mathematical semantics
- Quantum data, classical control
  - *Geometry of interaction* as the essence of classical control

Thank you for your attention!

Ichiro Hasuo (Dept. CS, U Tokyo)
http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro/

Naohiko Hoshino (RIMS, Kyoto U)
http://www.kurims.kyoto-u.ac.jp/~naophiko/

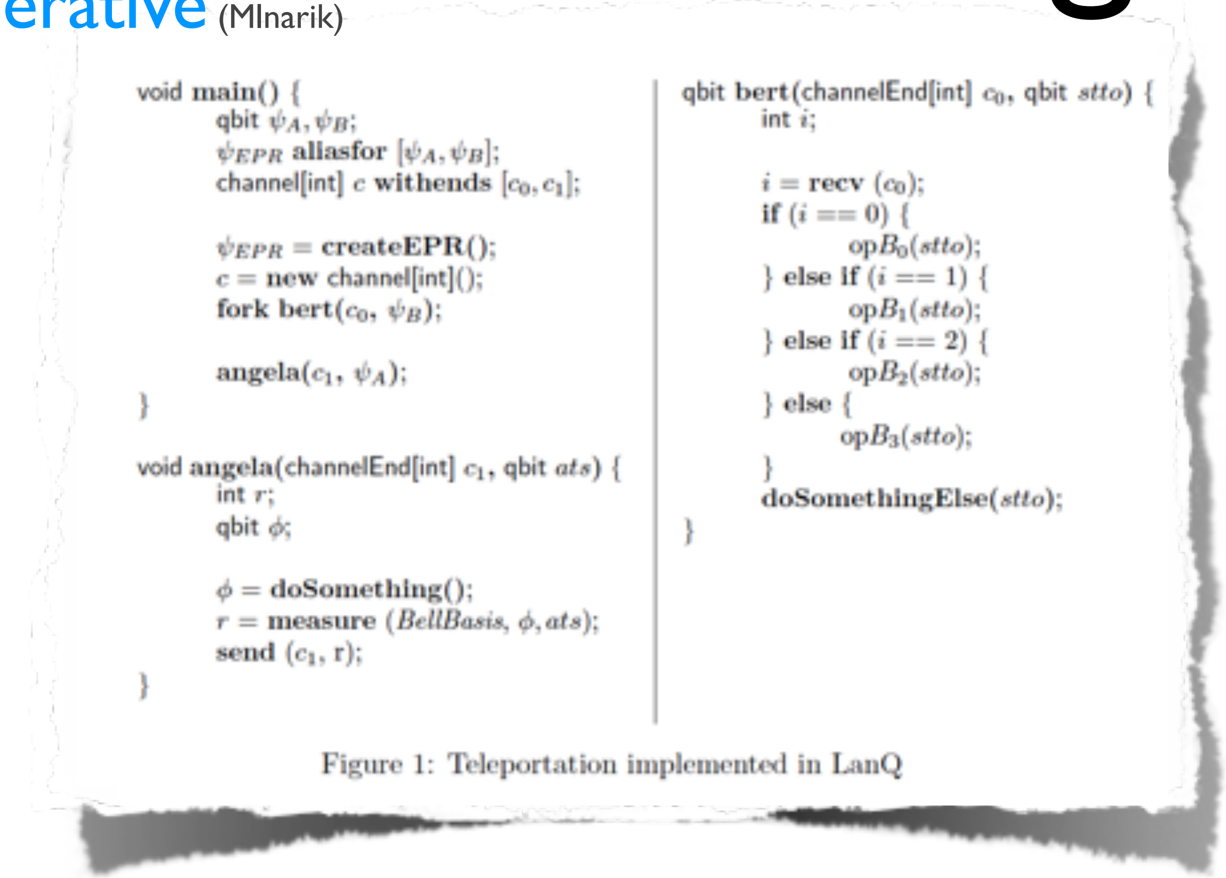
Quantum Programming Languages

Functional (Selinger, Valiron)

$$
\begin{aligned}
\textbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \textbf{EPR} * \mathit{in} \\
& \quad \mathit{let}\ f = \textbf{BellMeasure}\ x\ \mathit{in} \\
& \quad \mathit{let}\ g = \textbf{U}\ y \\
& \quad\ \ \mathit{in}\ \langle f, g\rangle.
\end{aligned}
$$

Quantum Programming Languages

Imperative (Mlnarik)

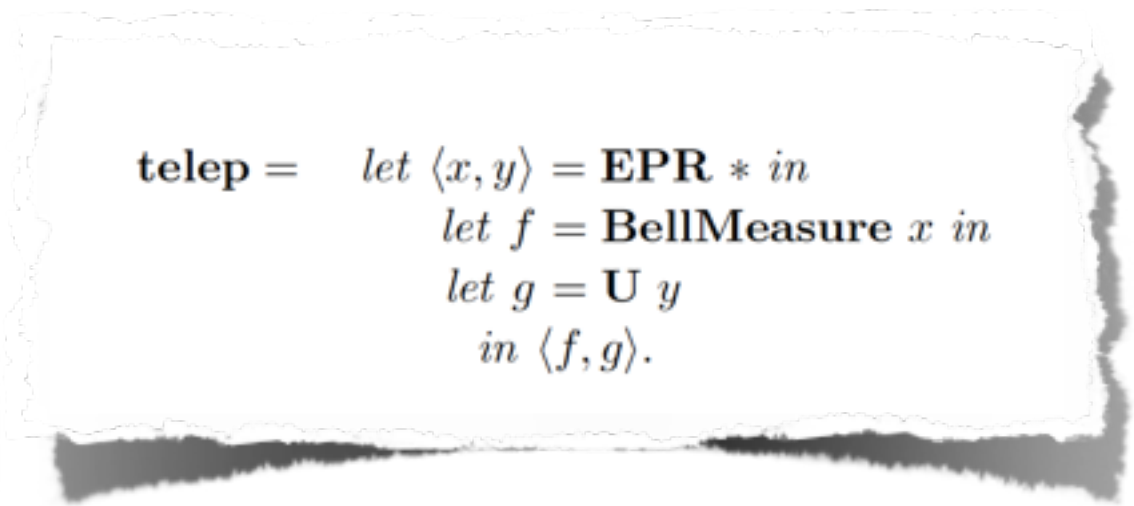


```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}
```

```
qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)



$$\begin{aligned}\textbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \textbf{EPR} * \mathit{in} \\ & \quad \mathit{let}\ f = \textbf{BellMeasure}\ x\ \mathit{in} \\ & \quad \mathit{let}\ g = \textbf{U}\ y \\ & \quad\ \mathit{in}\ \langle f, g\rangle.\end{aligned}$$

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}

qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

$$\begin{aligned}\mathbf{telep} = \quad & let\ \langle x, y\rangle = \mathbf{EPR} * in \\ & let\ f = \mathbf{BellMeasure}\ x\ in \\ & let\ g = \mathbf{U}\ y \\ & in\ \langle f, g\rangle.\end{aligned}$$

- "High-level" ➜ new algorithms?

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}
```

```
qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

$$\begin{aligned} \mathbf{telep} = \quad & let\ \langle x, y\rangle = \mathbf{EPR} * in \\ & \quad let\ f = \mathbf{BellMeasure}\ x\ in \\ & \quad let\ g = \mathbf{U}\ y \\ & \quad\ in\ \langle f, g\rangle. \end{aligned}$$

- "High-level" ➜ new algorithms?
- (Sometimes) good handling of quantum vs. classical data
  - *No-Cloning* vs. *Duplicable*

Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
        qbit ψ_A, ψ_B;
        ψ_EPR aliasfor [ψ_A, ψ_B];
        channel[int] c withends [c_0, c_1];

        ψ_EPR = createEPR();
        c = new channel[int]();
        fork bert(c_0, ψ_B);

        angela(c_1, ψ_A);
}

void angela(channelEnd[int] c_1, qbit ats) {
        int r;
        qbit φ;

        φ = doSomething();
        r = measure (BellBasis, φ, ats);
        send (c_1, r);
}

qbit bert(channelEnd[int] c_0, qbit stto) {
        int i;

        i = recv (c_0);
        if (i == 0) {
                opB_0(stto);
        } else if (i == 1) {
                opB_1(stto);
        } else if (i == 2) {
                opB_2(stto);
        } else {
                opB_3(stto);
        }
        doSomethingElse(stto);
}
```

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

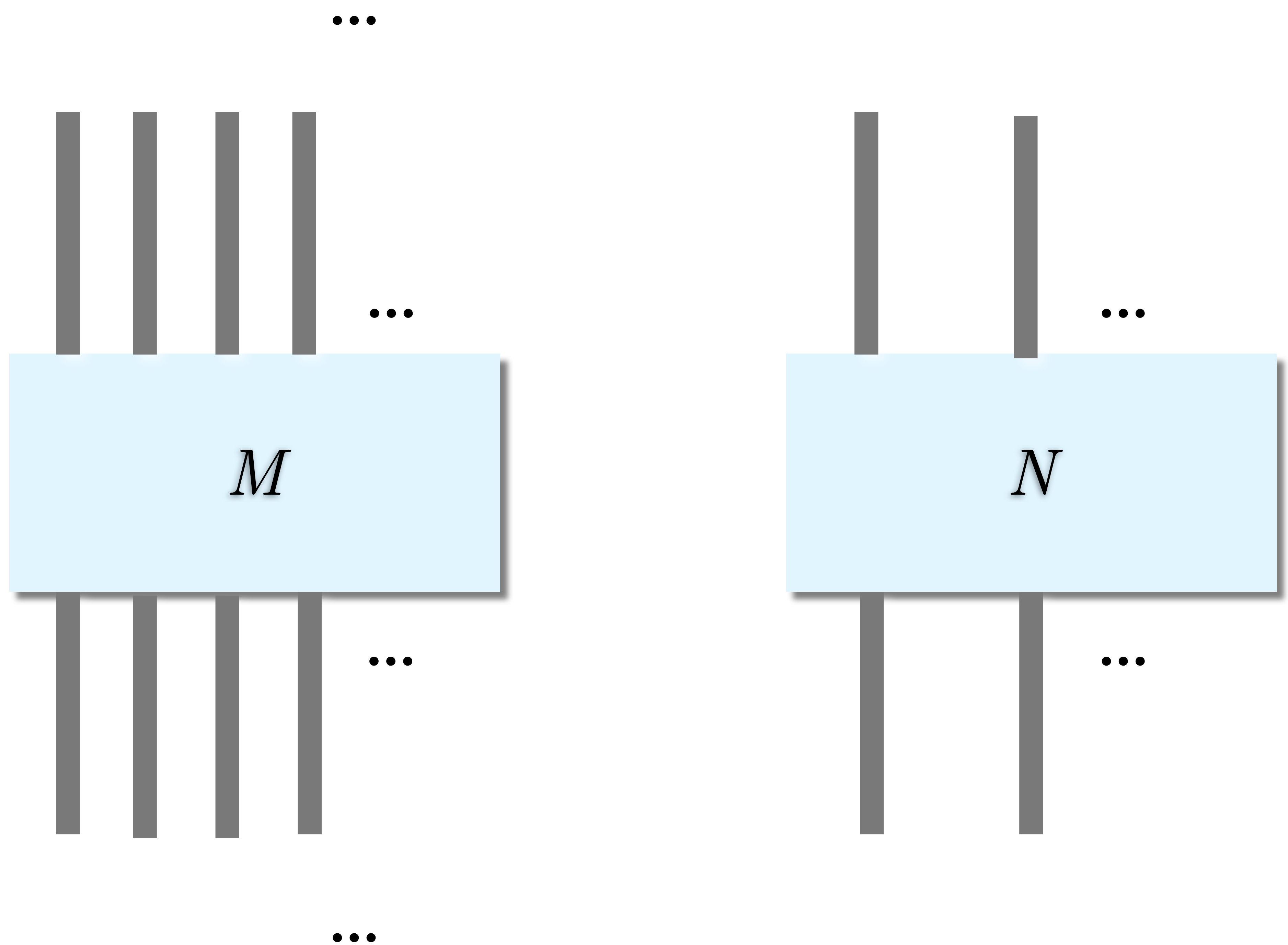
$$
\begin{aligned}
\mathbf{telep} = \quad & \mathit{let}\ \langle x, y\rangle = \mathbf{EPR} * \mathit{in} \\
& \quad \mathit{let}\ f = \mathbf{BellMeasure}\ x\ \mathit{in} \\
& \quad \mathit{let}\ g = \mathbf{U}\ y \\
& \quad\ \mathit{in}\ \langle f, g\rangle.
\end{aligned}
$$

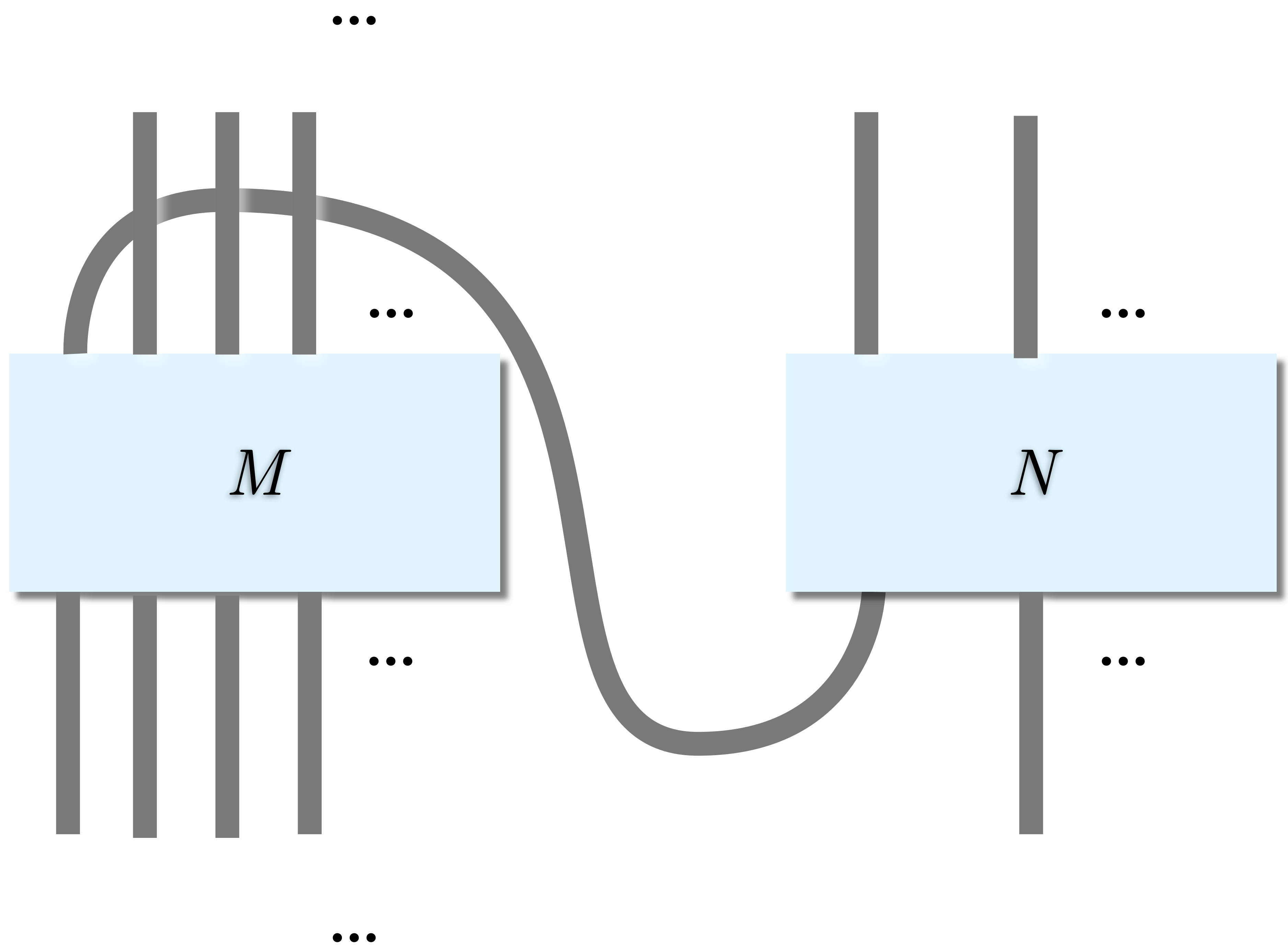
- "High-level" ➜ new algorithms?
- (Sometimes) good handling of quantum vs. classical data
  - *No-Cloning* vs. *Duplicable*
- Model *quantum communication protocols*

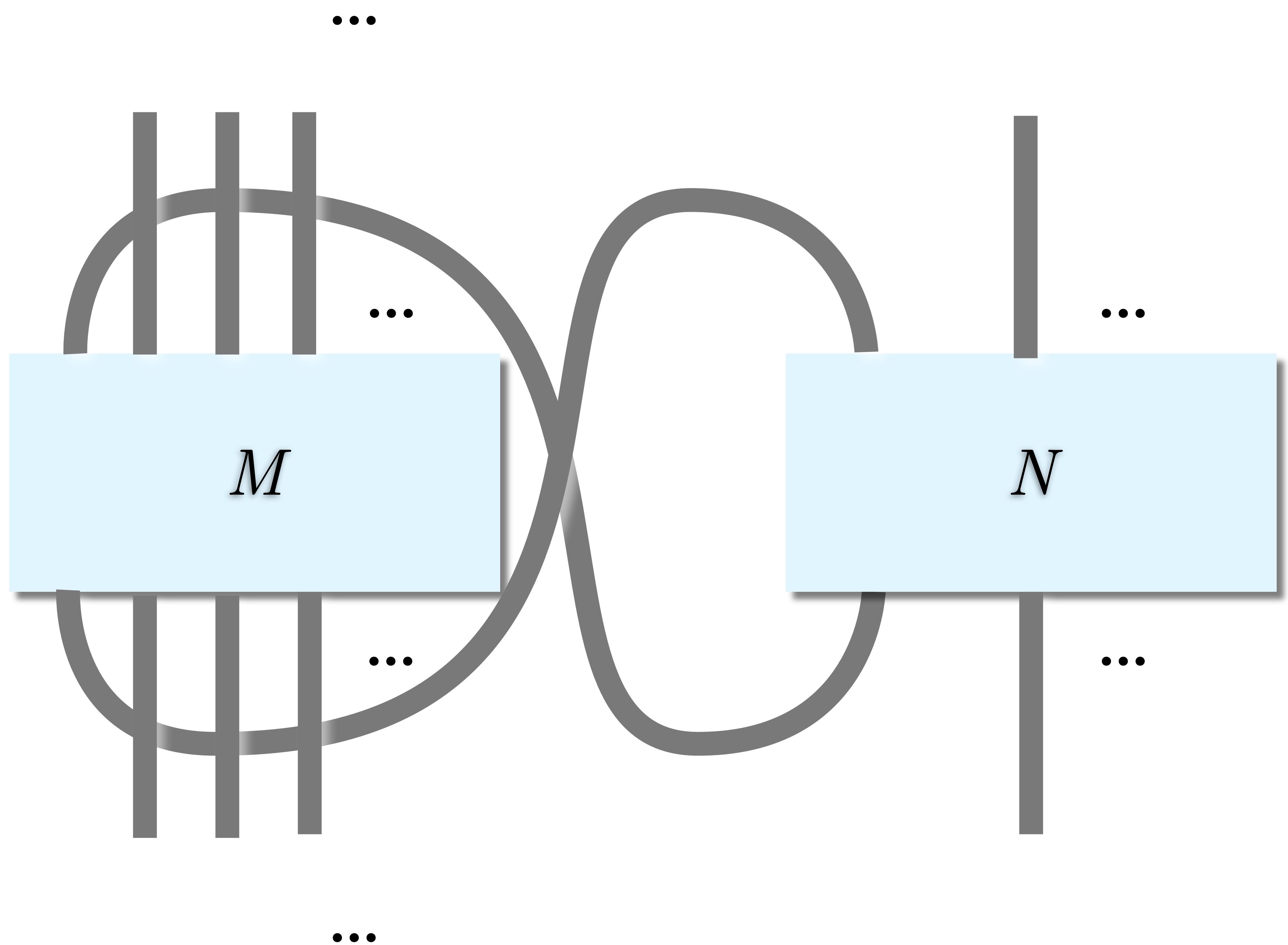
Quantum Functional Programming Language
&
Its Denotational Semantics

Quantum Functional Programming Language
&
Its Denotational Semantics

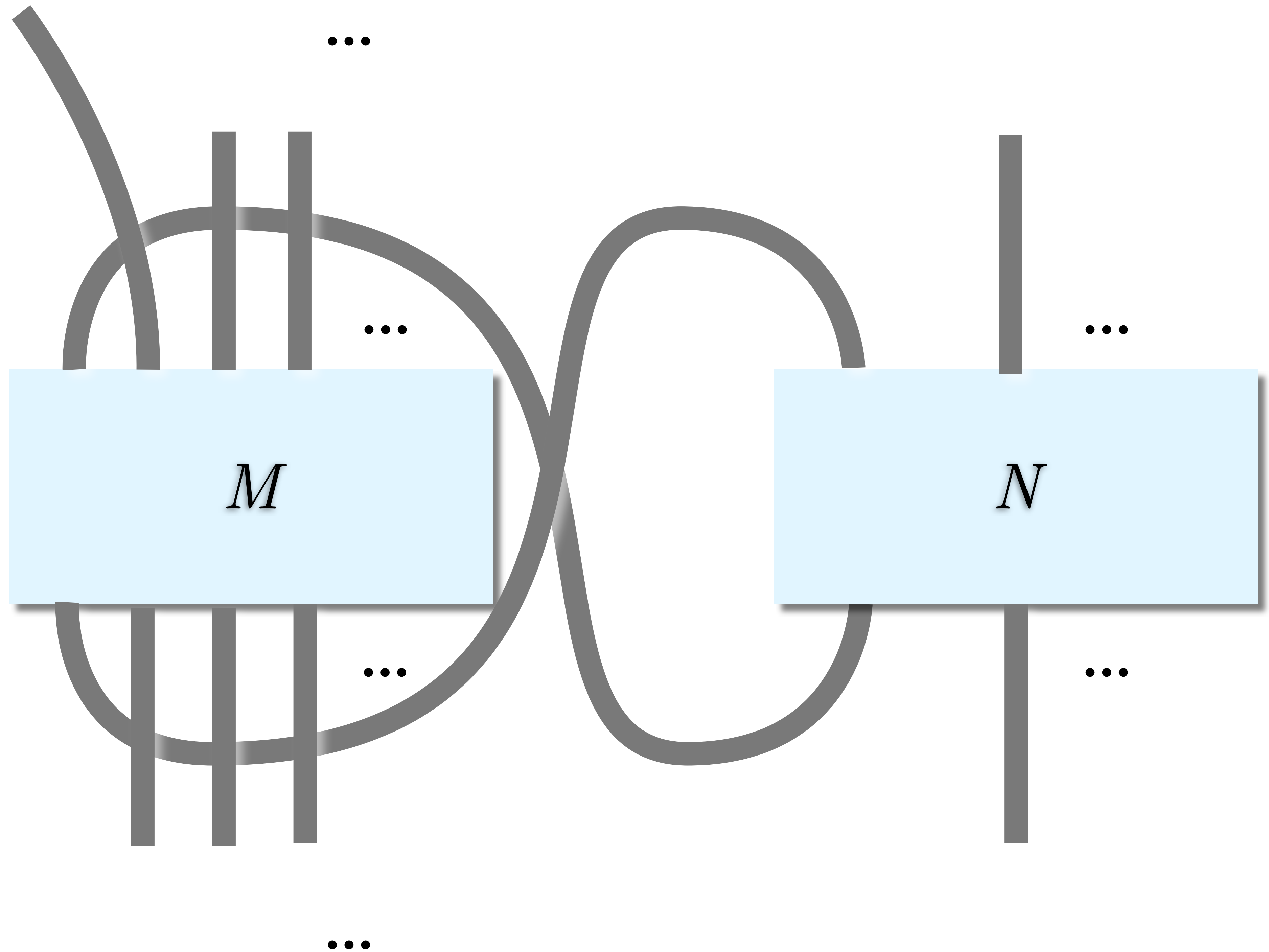
Q4. Why denotational semantics?


...
...
M
...
...
N
...
...

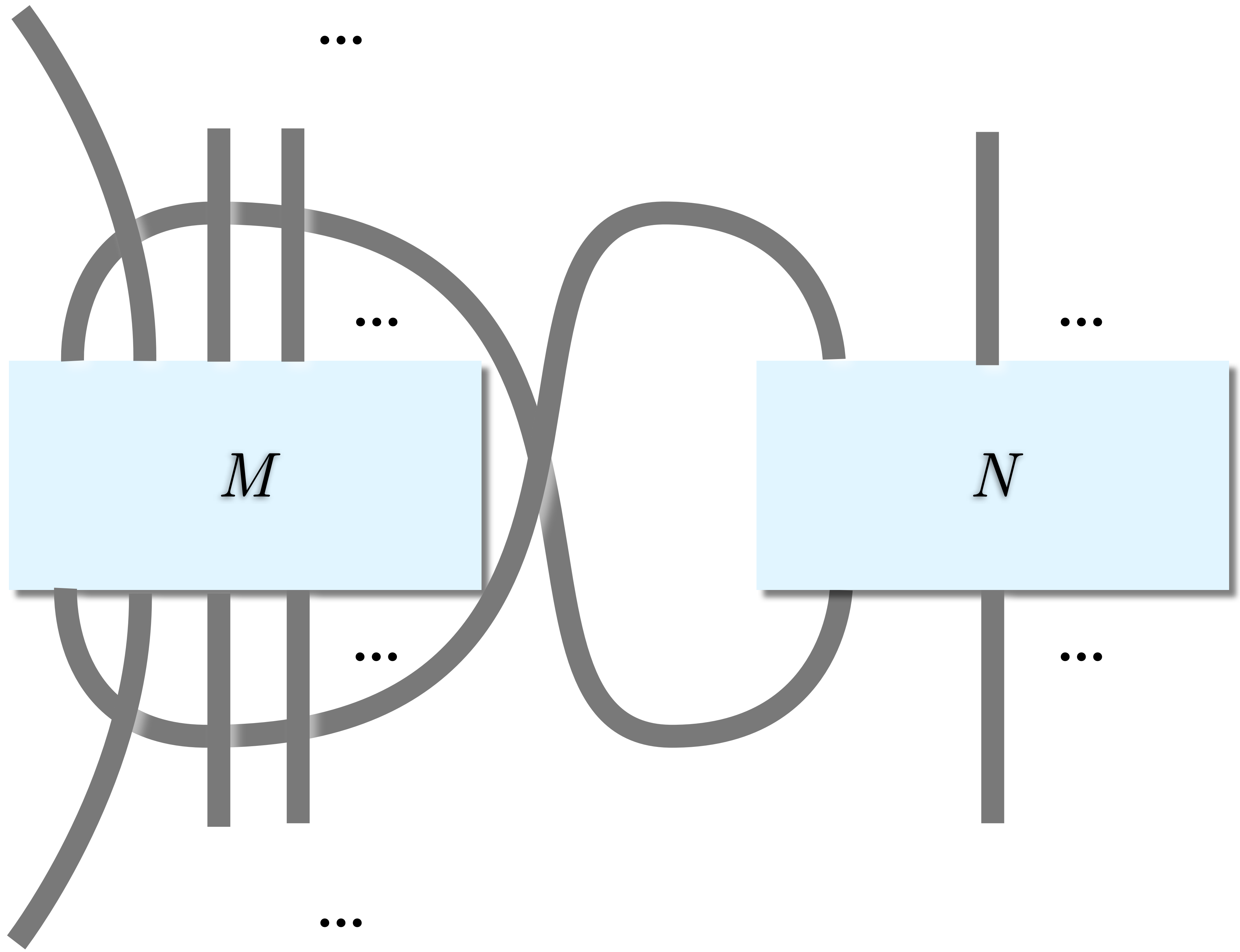

...
...
...
M
N
...
...
...

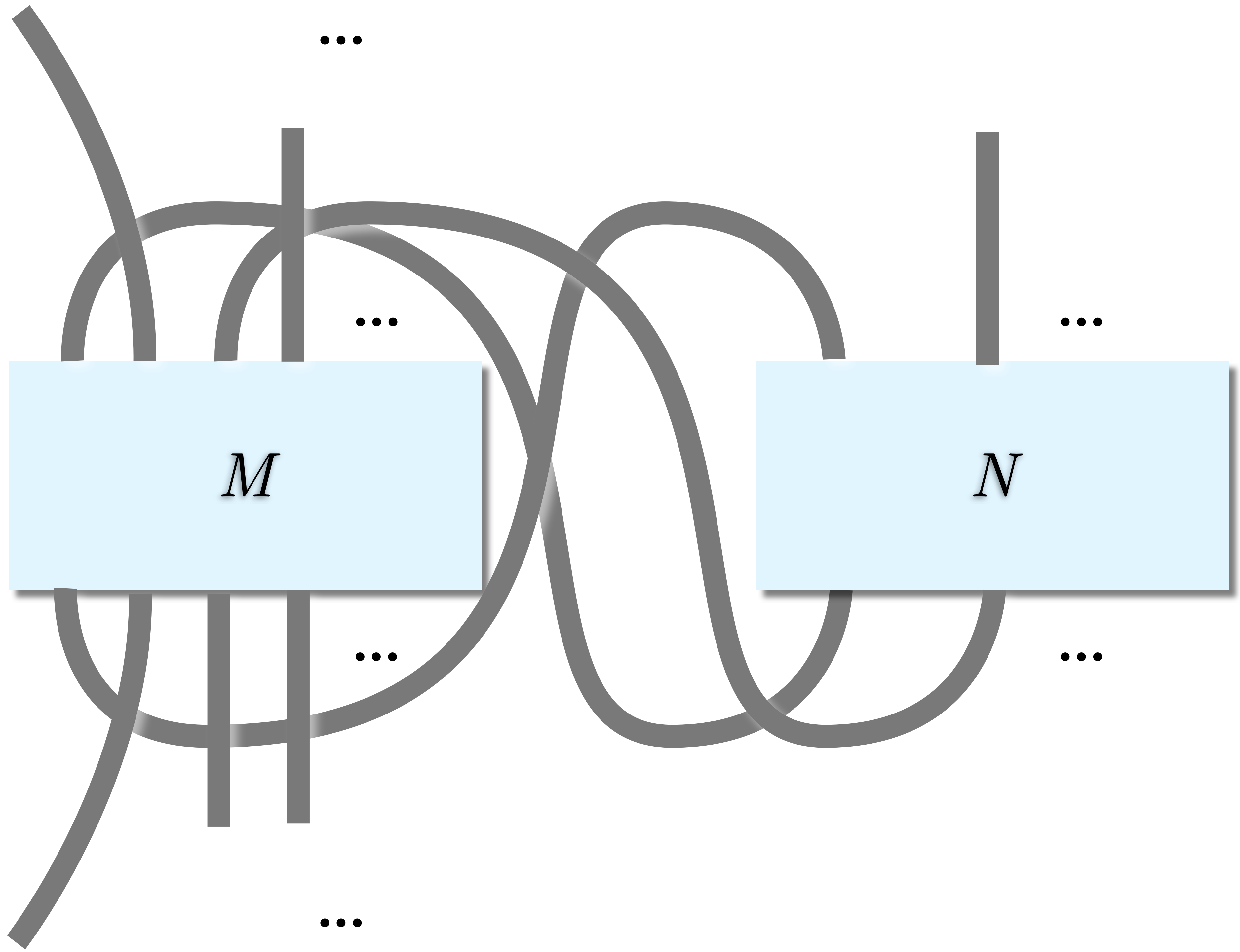


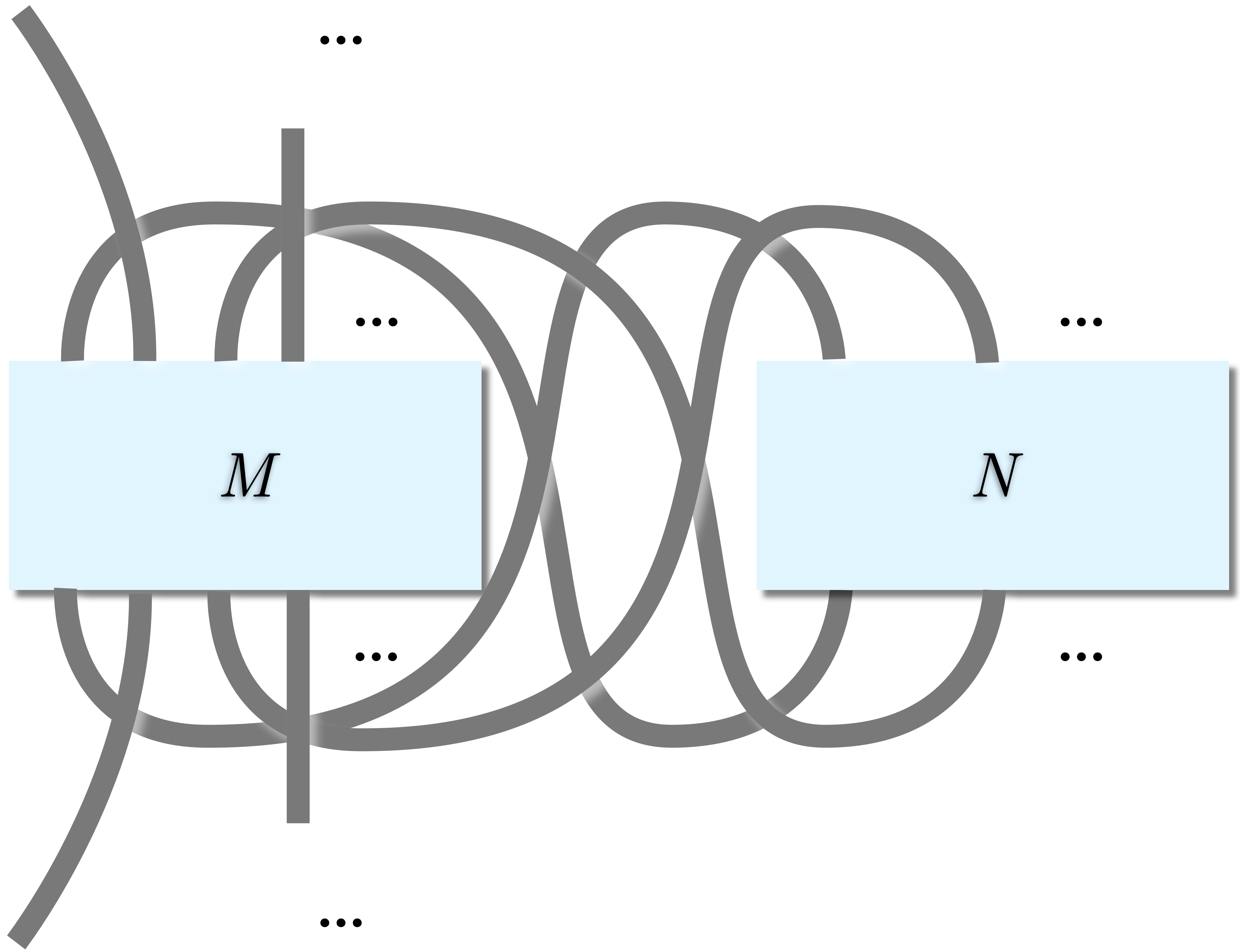
...
...
...
M
N
...
...
...



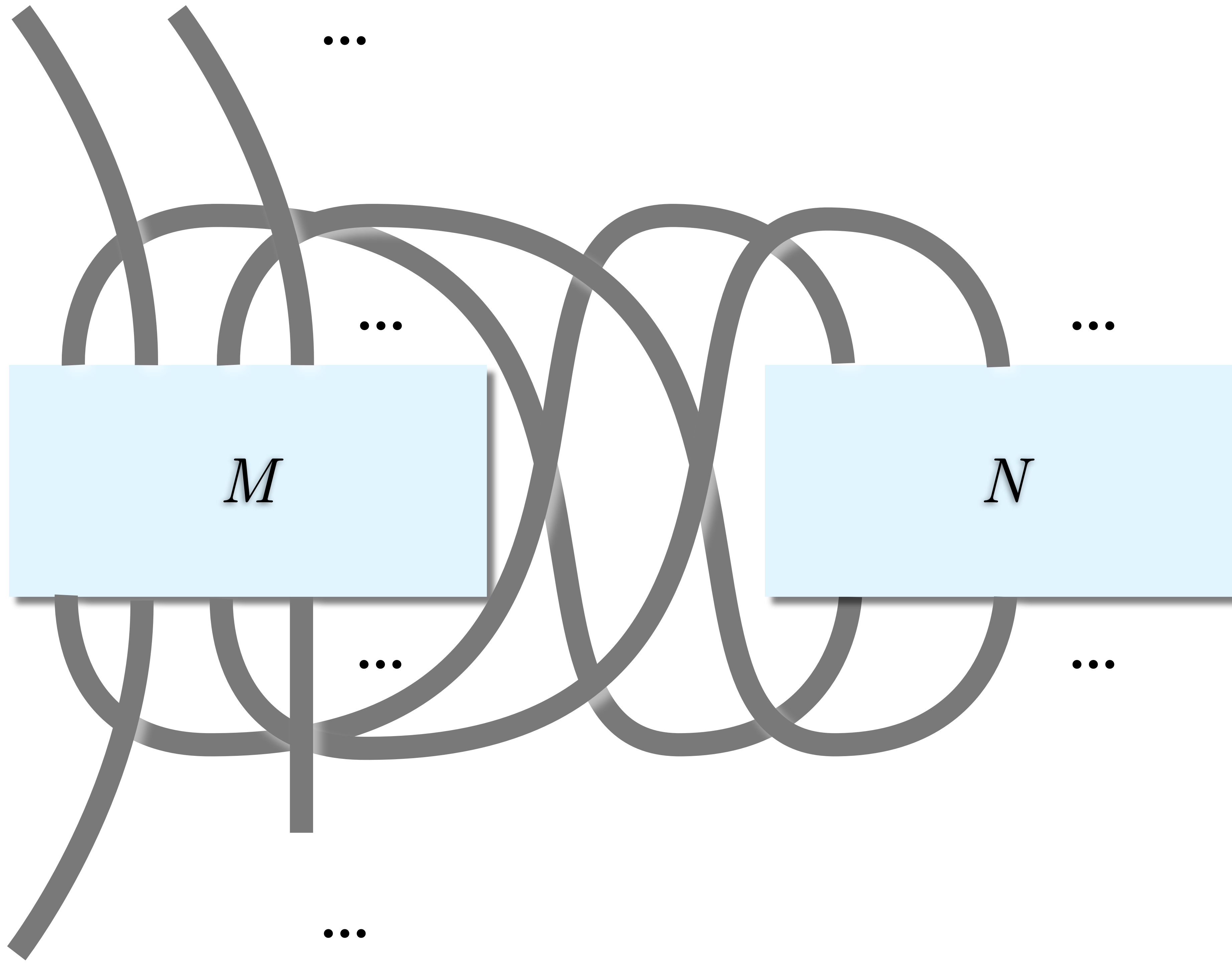
M
N


...
...
M
...
N
...
...
...

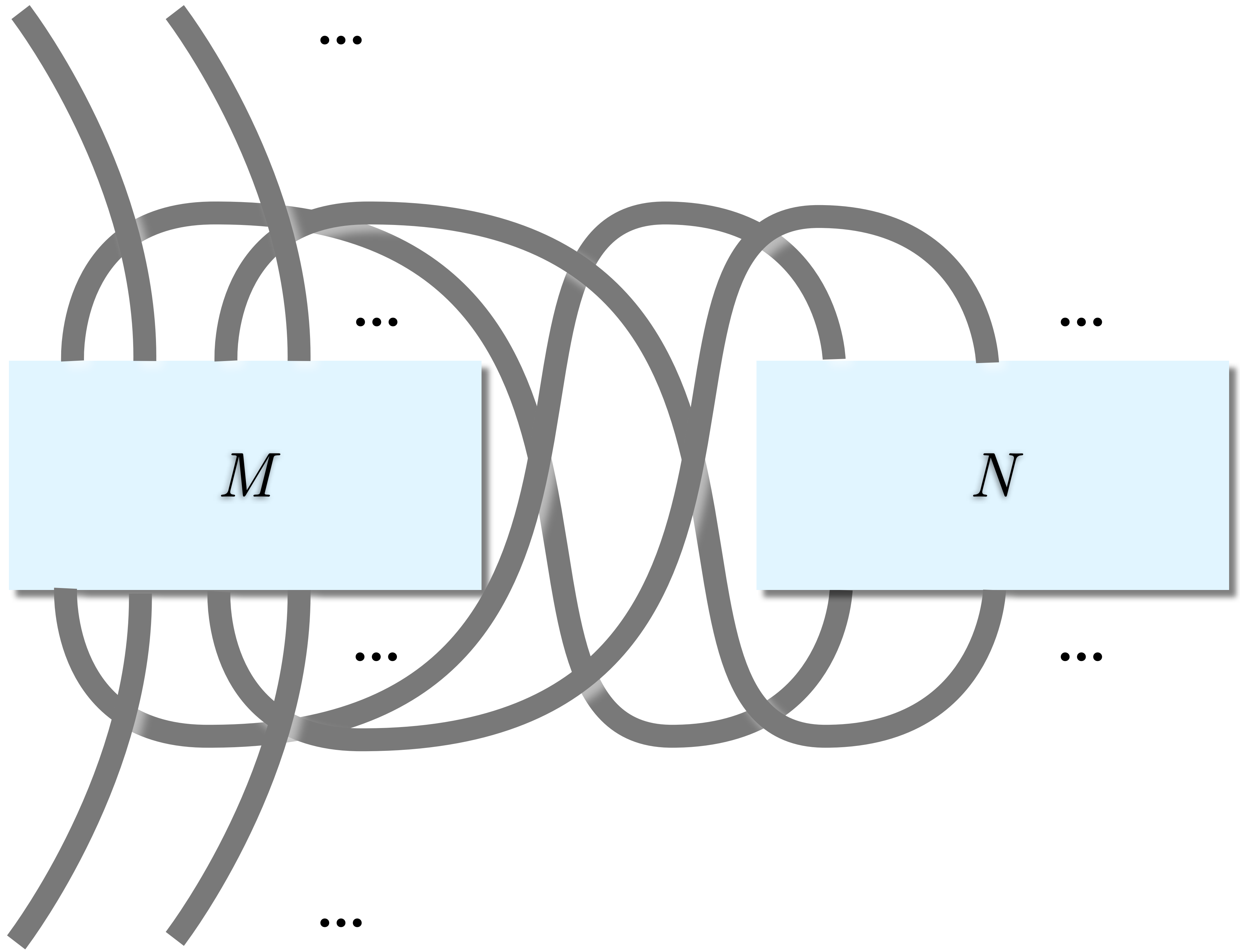

M
N



...
...
...
...
...
...
M
N



...
...
...
M
N
...
...
...


...
...
...
...
...
...
M
N