La pleine paresse, une certaine optimalité

Sharing of subterms
and Reduction strategies
for Higher-order rewriting

Thibaut Balabonski
November 16, 2012

▶

Reduction strategies

Three historical results

Thesis & Approach

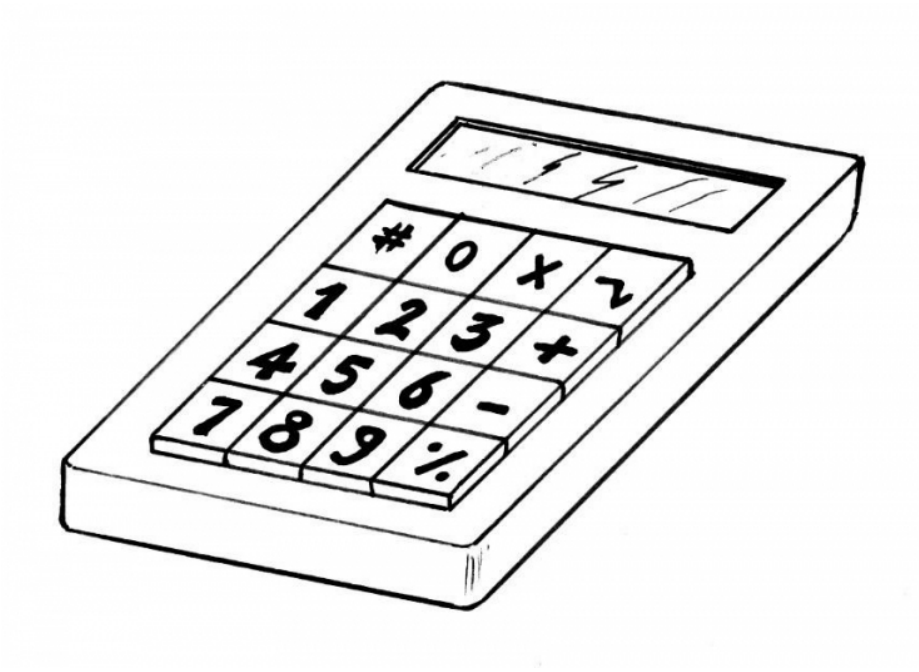
Lévy-optimality

Sharing-via-Labelling

Lambda-lifting

Let's compute

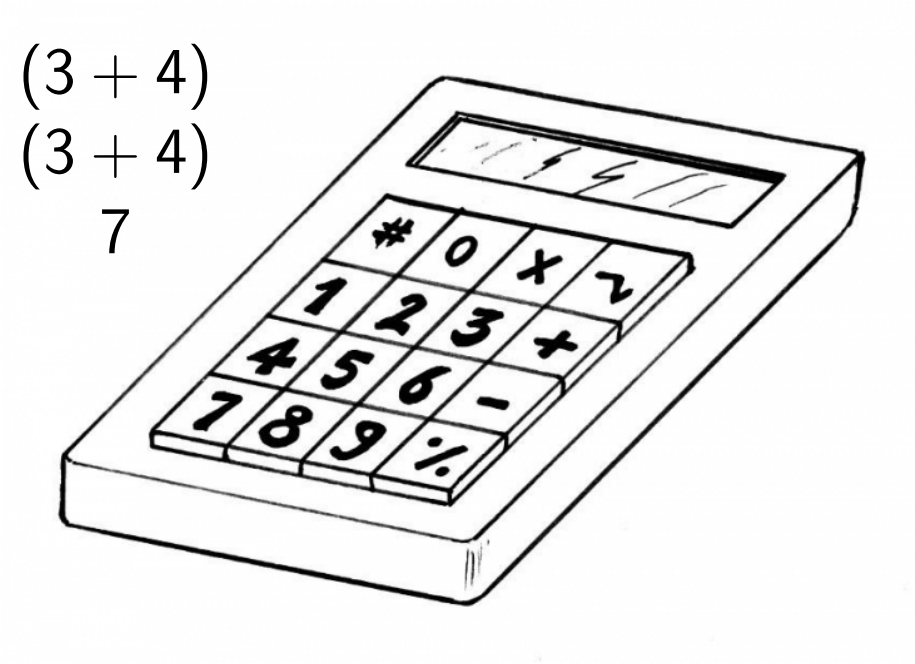
$$(1+2) \times (3+4)$$



Let's compute

$$(1+2) \times (3+4)$$

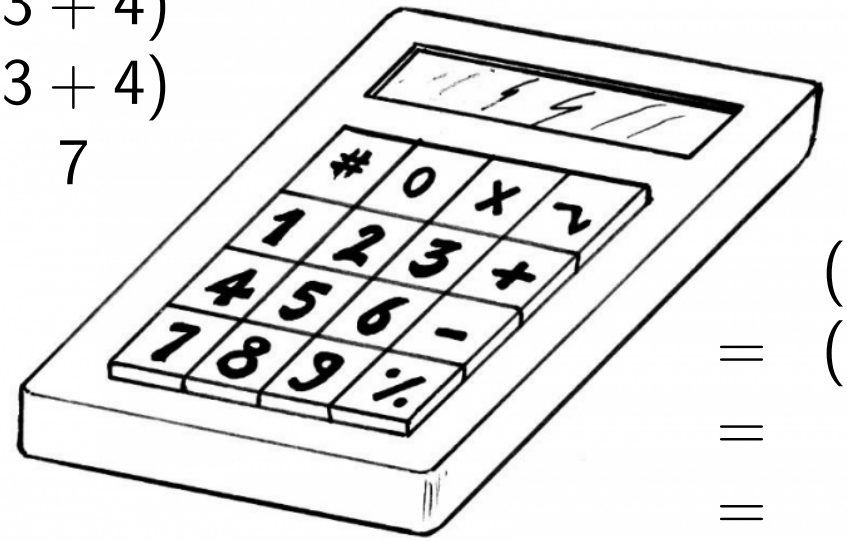
$$\begin{aligned} & (1+2) \times (3+4) \\ = \quad & 3 \times (3+4) \\ = \quad & 3 \times 7 \\ = \quad & 21 \end{aligned}$$



Let's compute

$$(1+2) \times (3+4)$$

$$\begin{aligned} & (1+2) \times (3+4) \\ = \; & 3 \times (3+4) \\ = \; & 3 \times 7 \\ = \; & 21 \end{aligned}$$

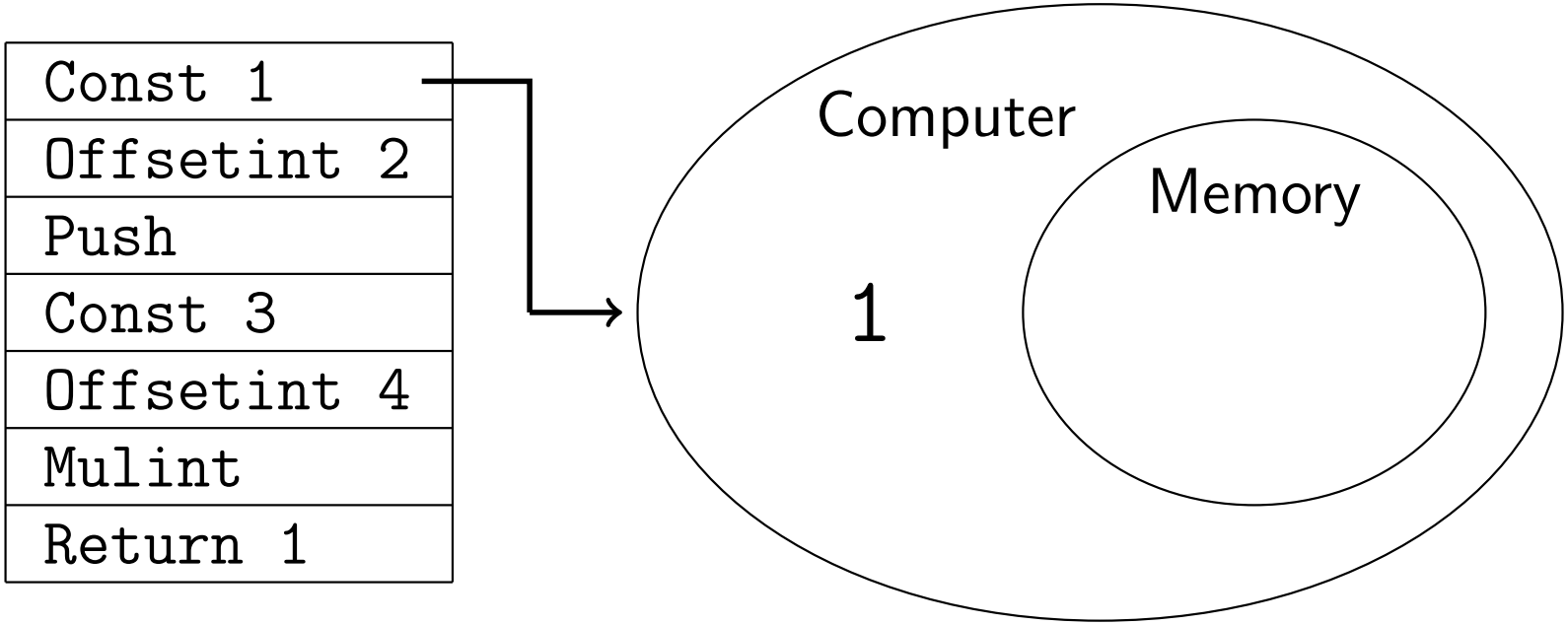


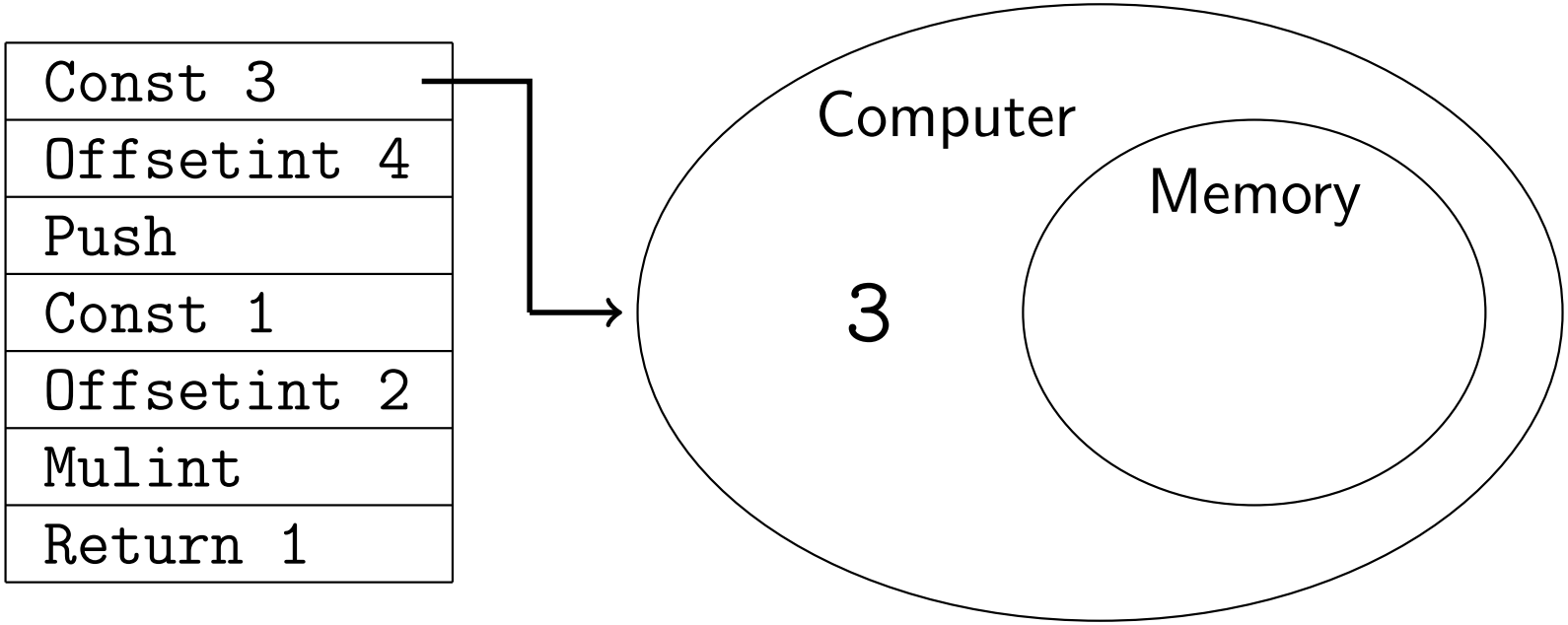
$$\begin{aligned} & (1+2) \times (3+4) \\ = \; & (1+2) \times 7 \\ = \; & 3 \times 7 \\ = \; & 21 \end{aligned}$$

Reduction space

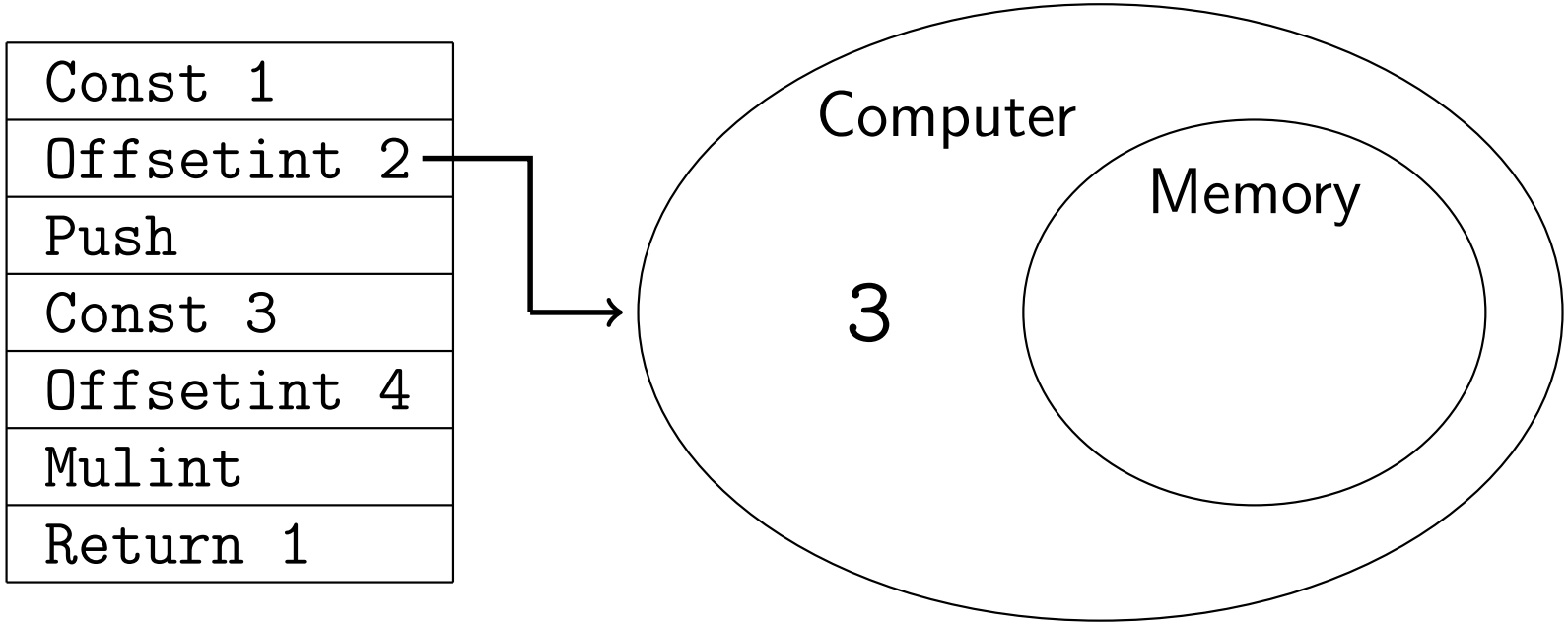
$$
\begin{tikzcd}
 & (1+2)\times(3+4) \arrow[dl] \arrow[dr] & \\
3\times(3+4) \arrow[dr] & & (1+2)\times 7 \arrow[dl] \\
 & 3\times 7 \arrow[d] & \\
 & 21 &
\end{tikzcd}
$$

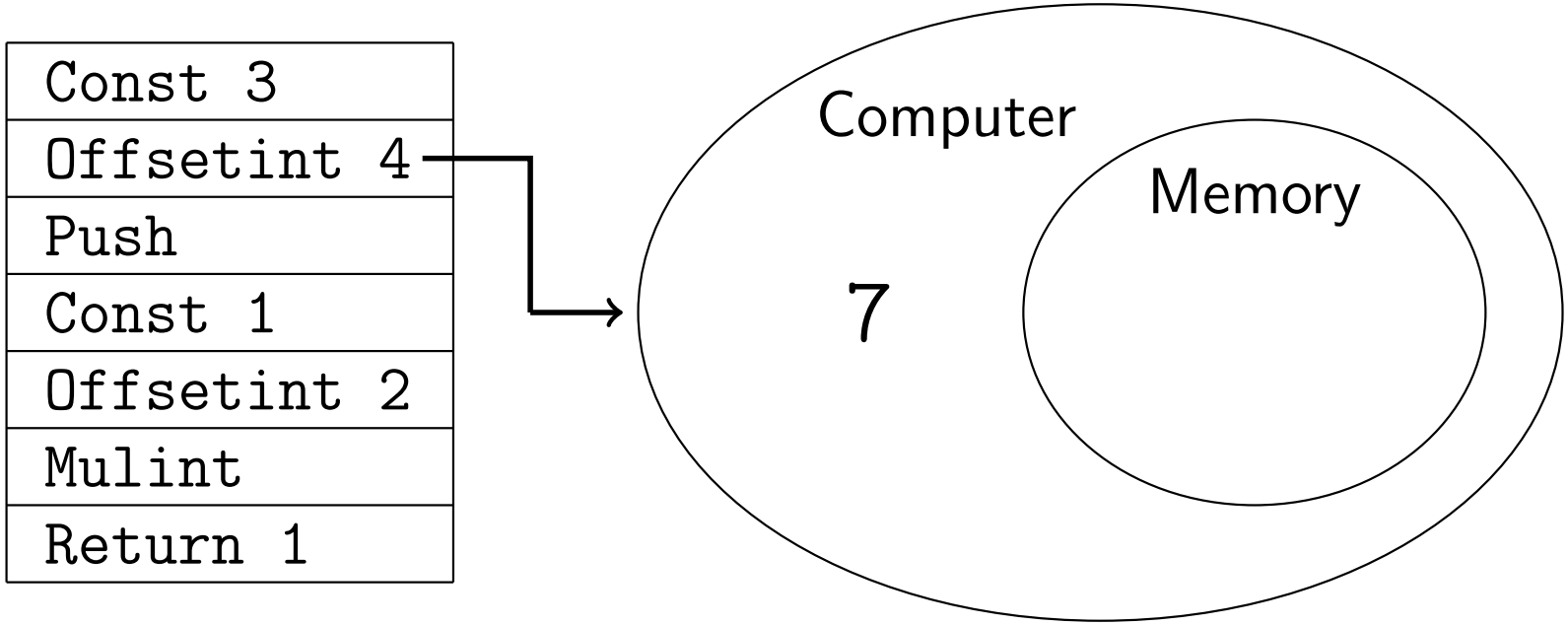
The computer has to choose a path



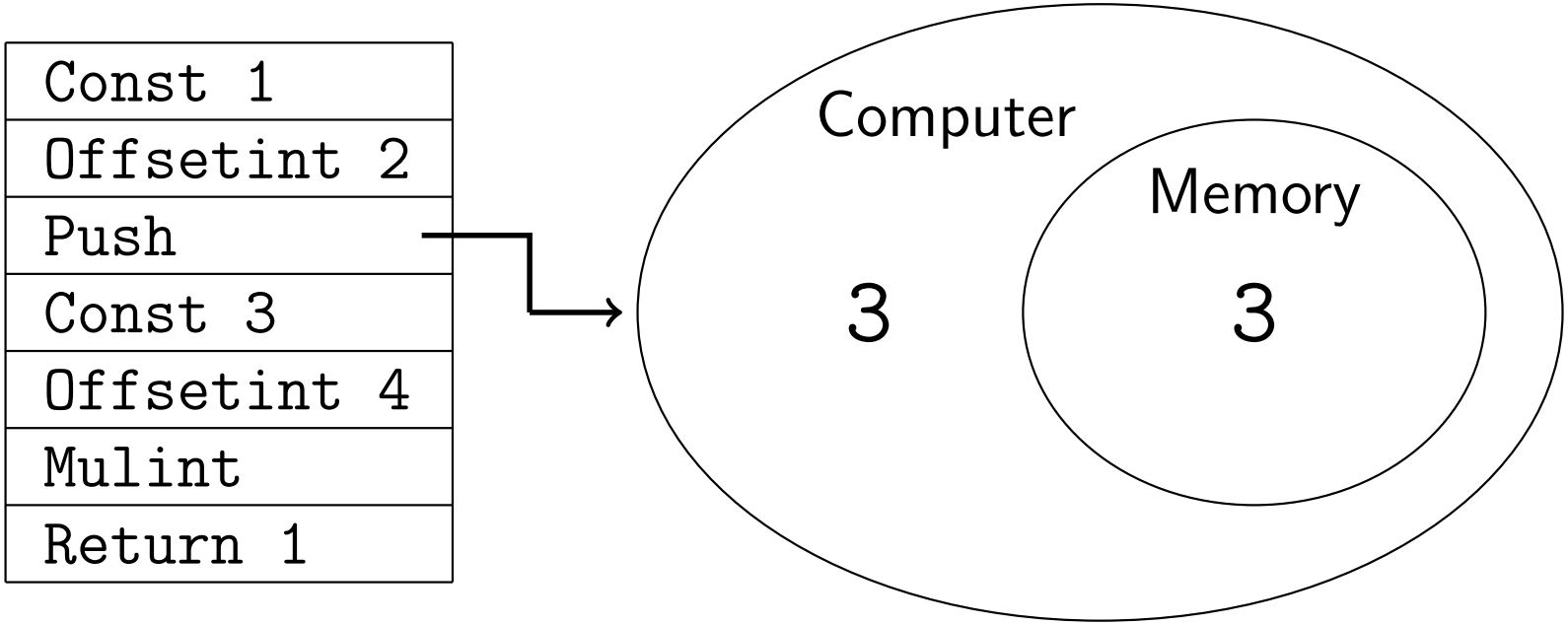


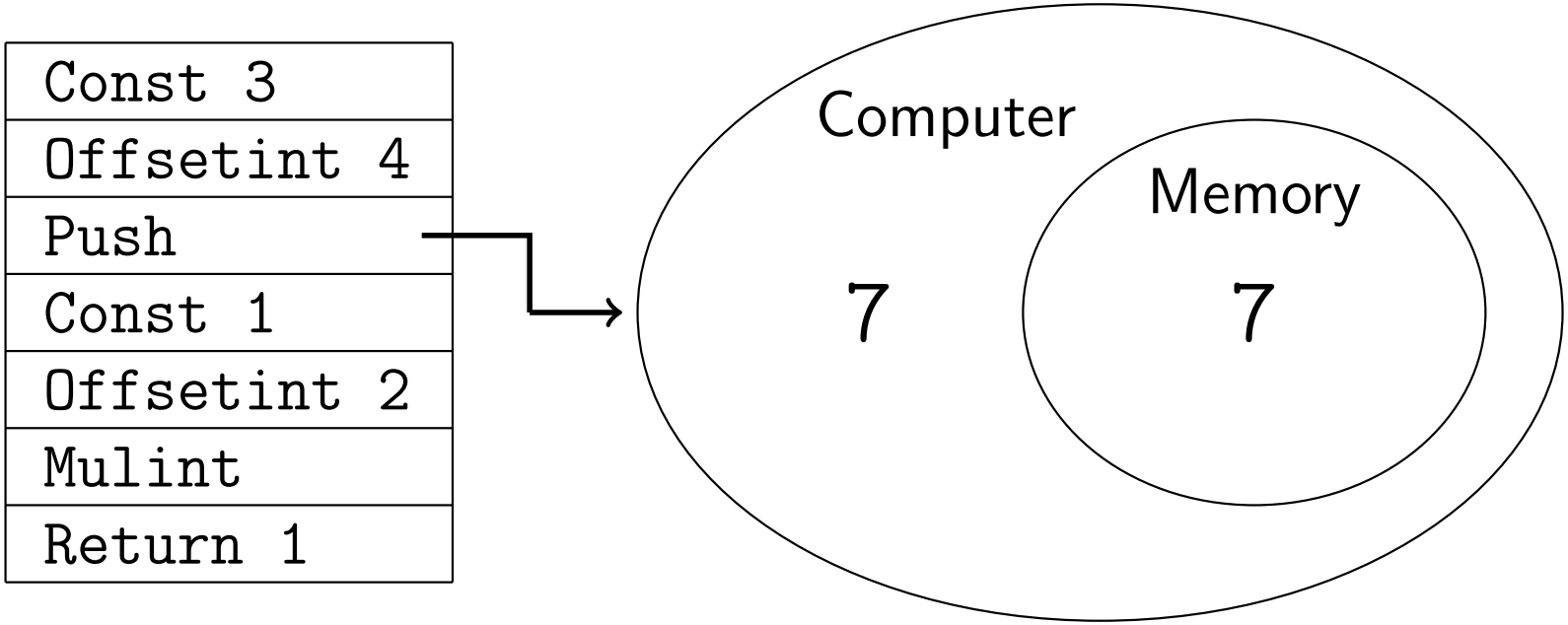
The computer has to choose a path



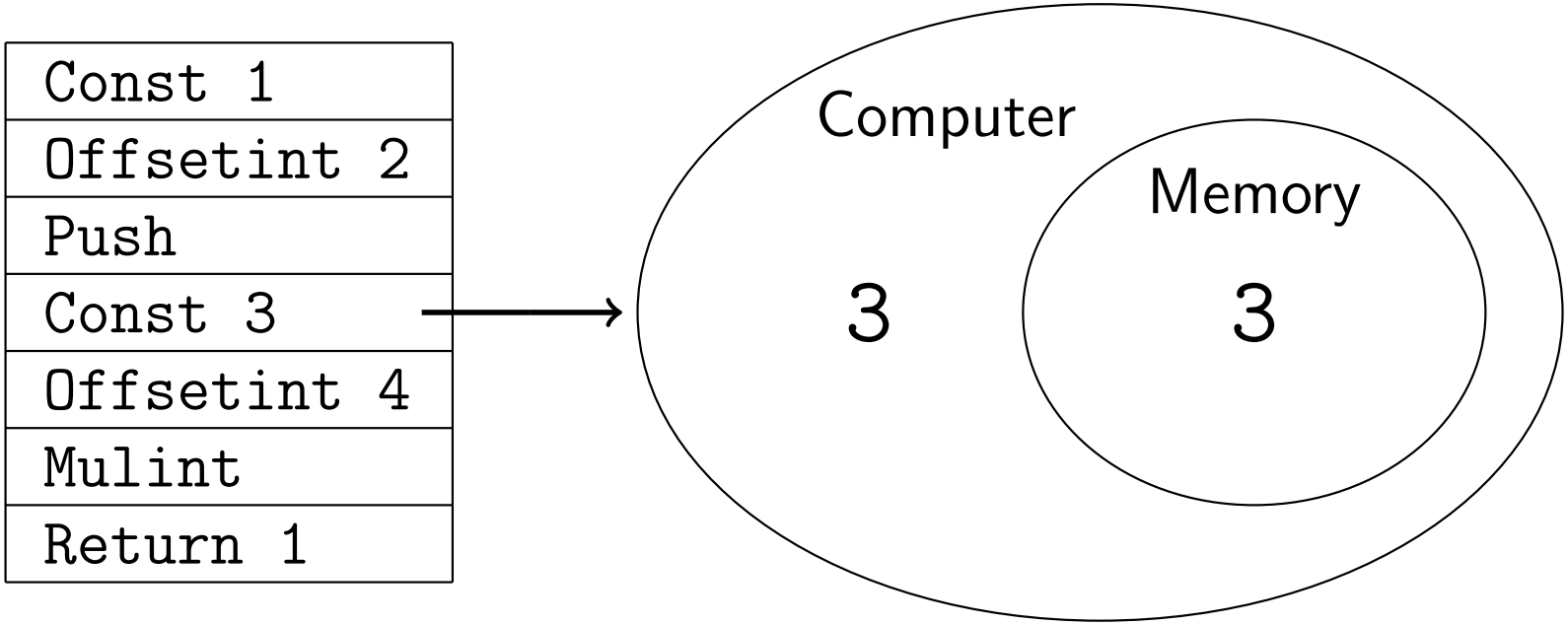


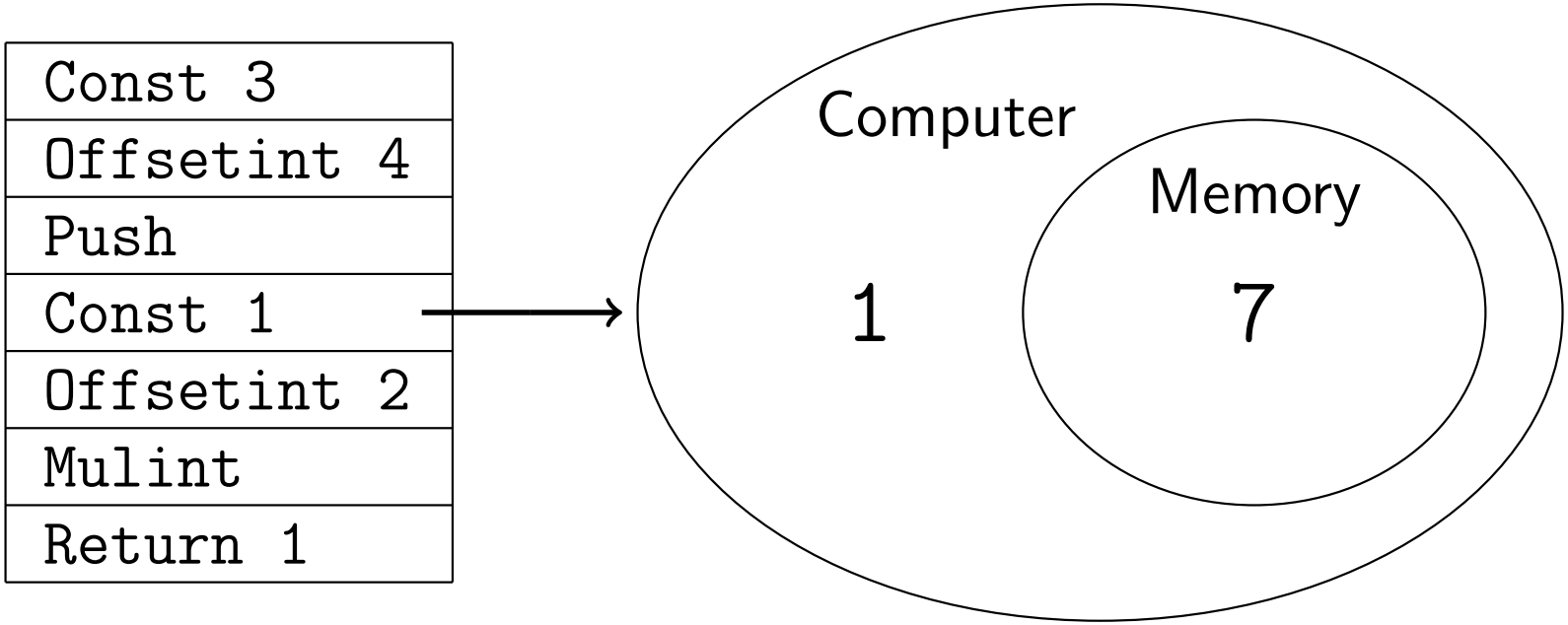
The computer has to choose a path



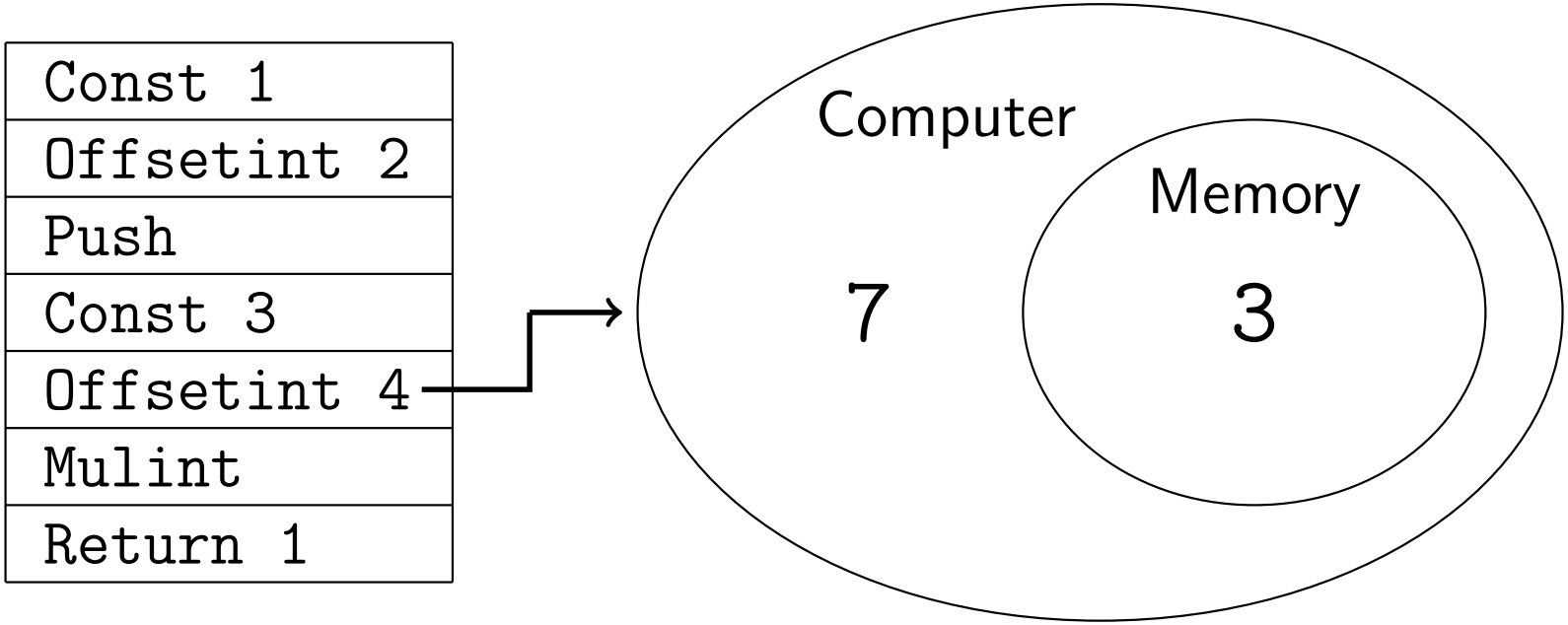


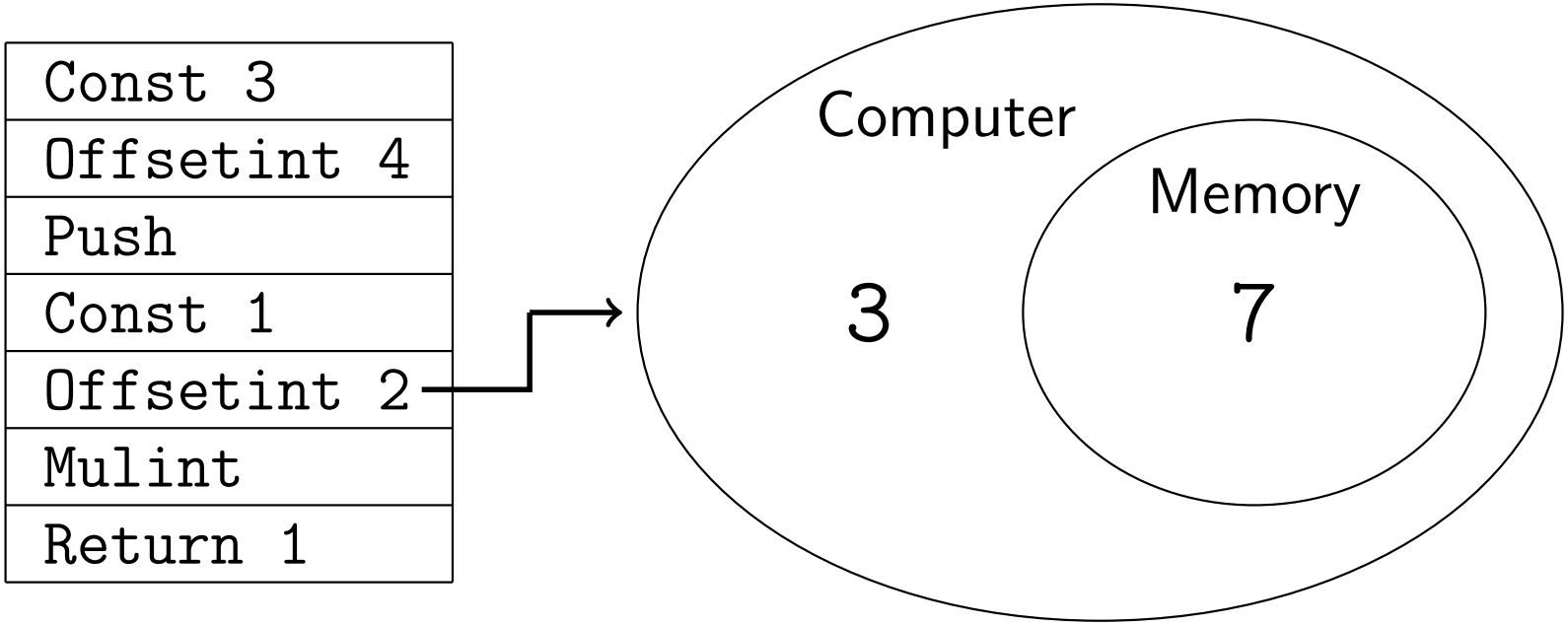
The computer has to choose a path



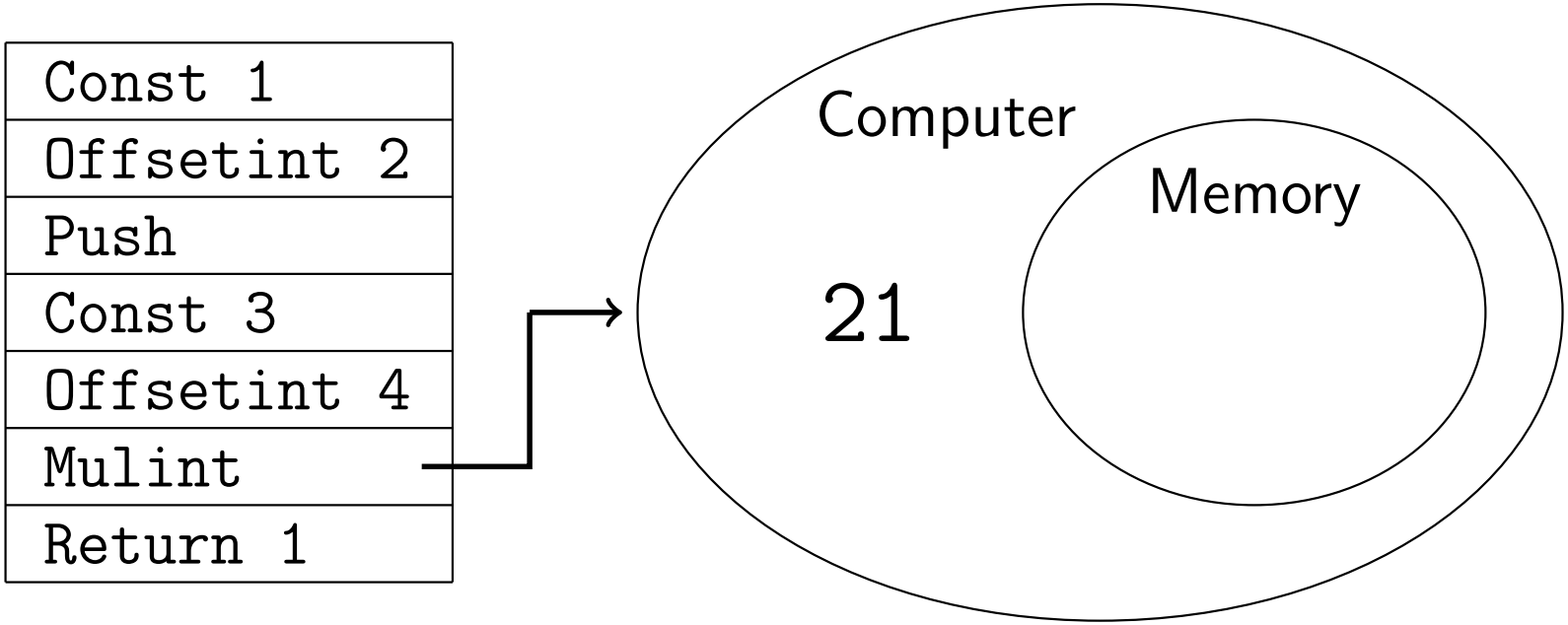


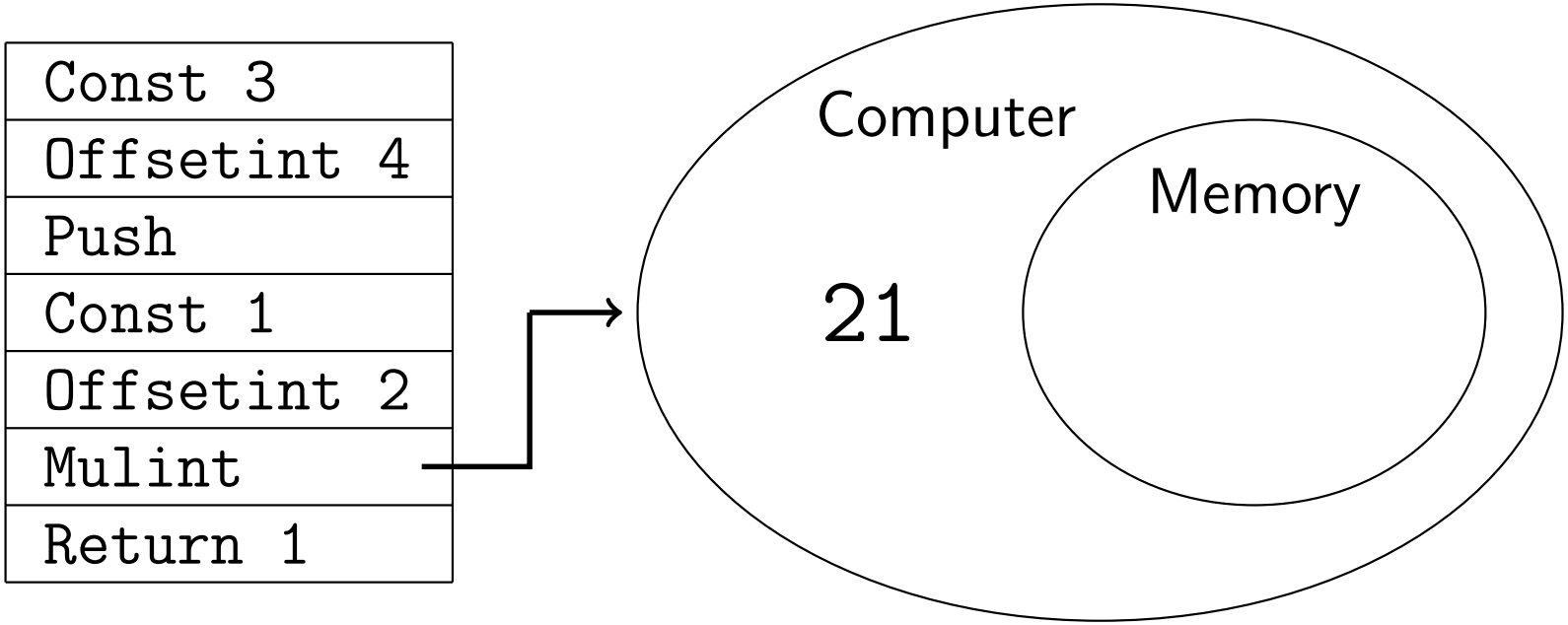
The computer has to choose a path



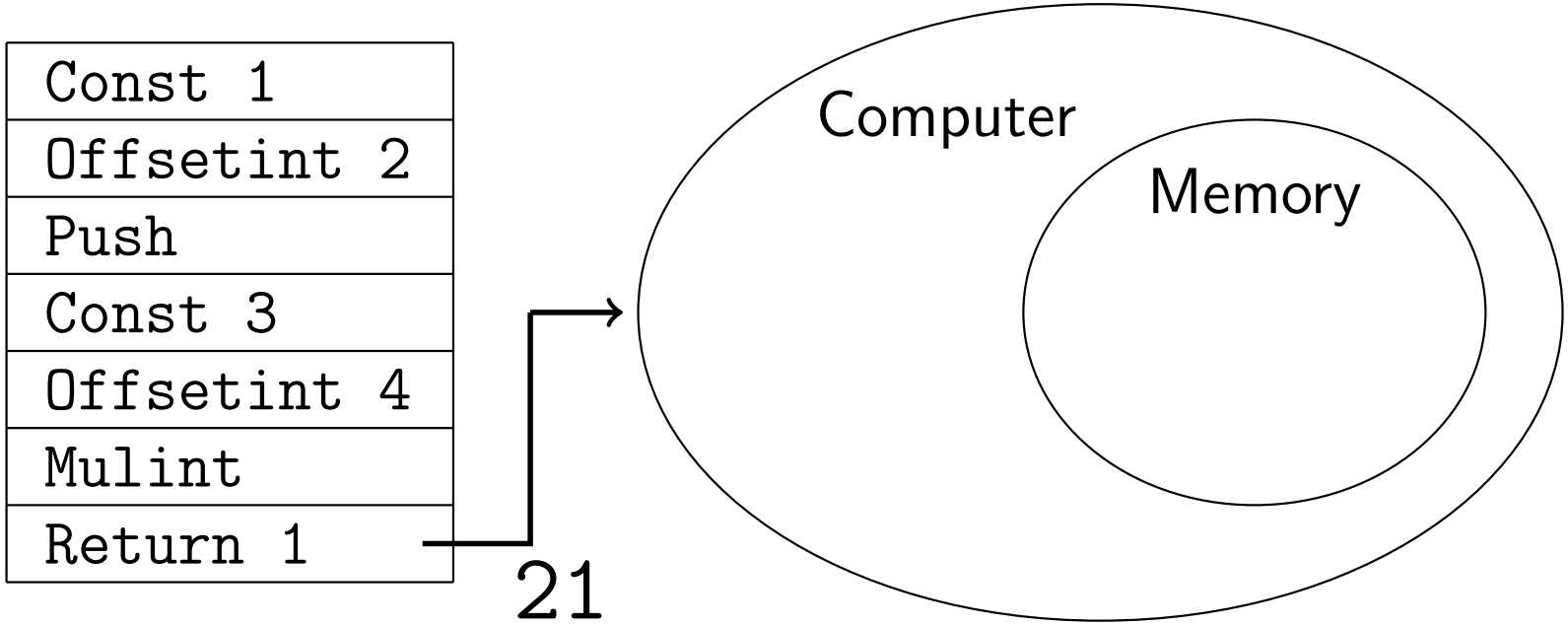


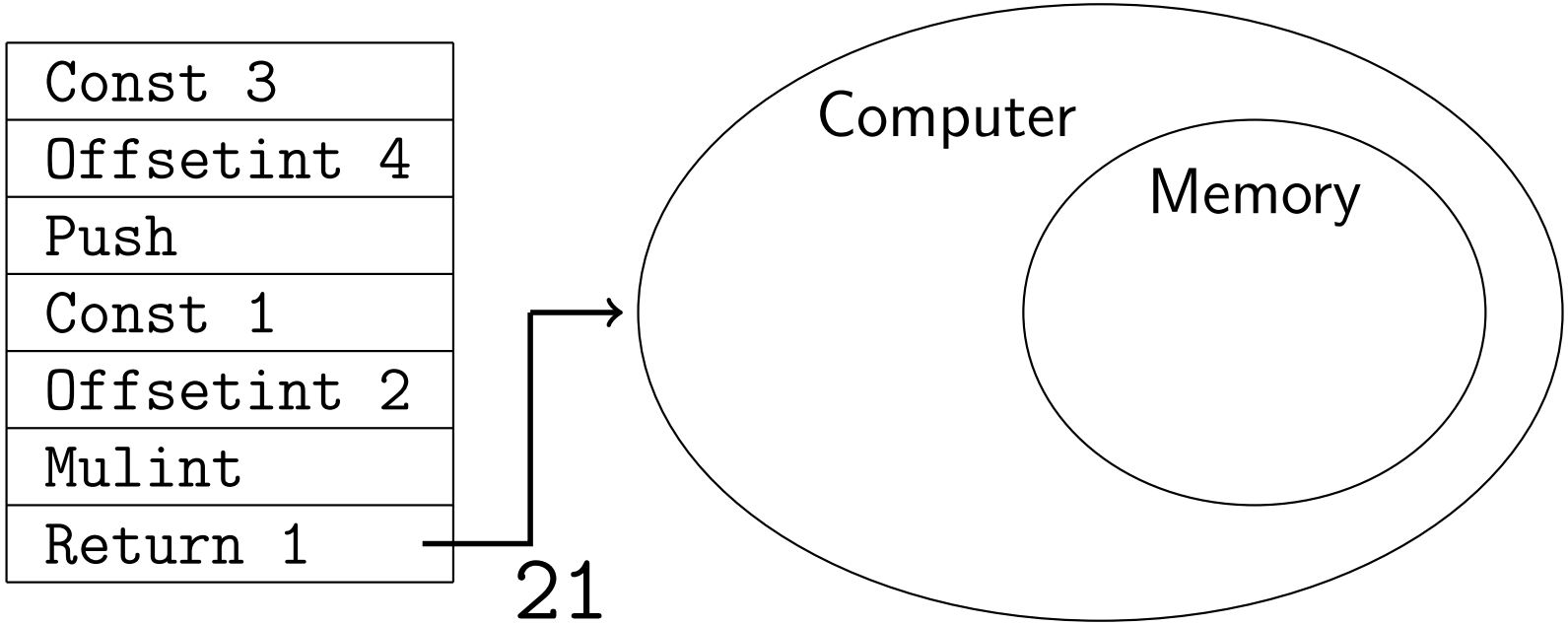
The computer has to choose a path





The computer has to choose a path





Different paths, different distances

- Outermost prevents unneeded computations.
- Innermost prevents duplication of subprograms.

$$\begin{tikzcd}
0 \times (1 + 2) \arrow[rr] \arrow[dr] & & 0 \\
& 0 \times 3 \arrow[ur] &
\end{tikzcd}$$

Different paths, different distances

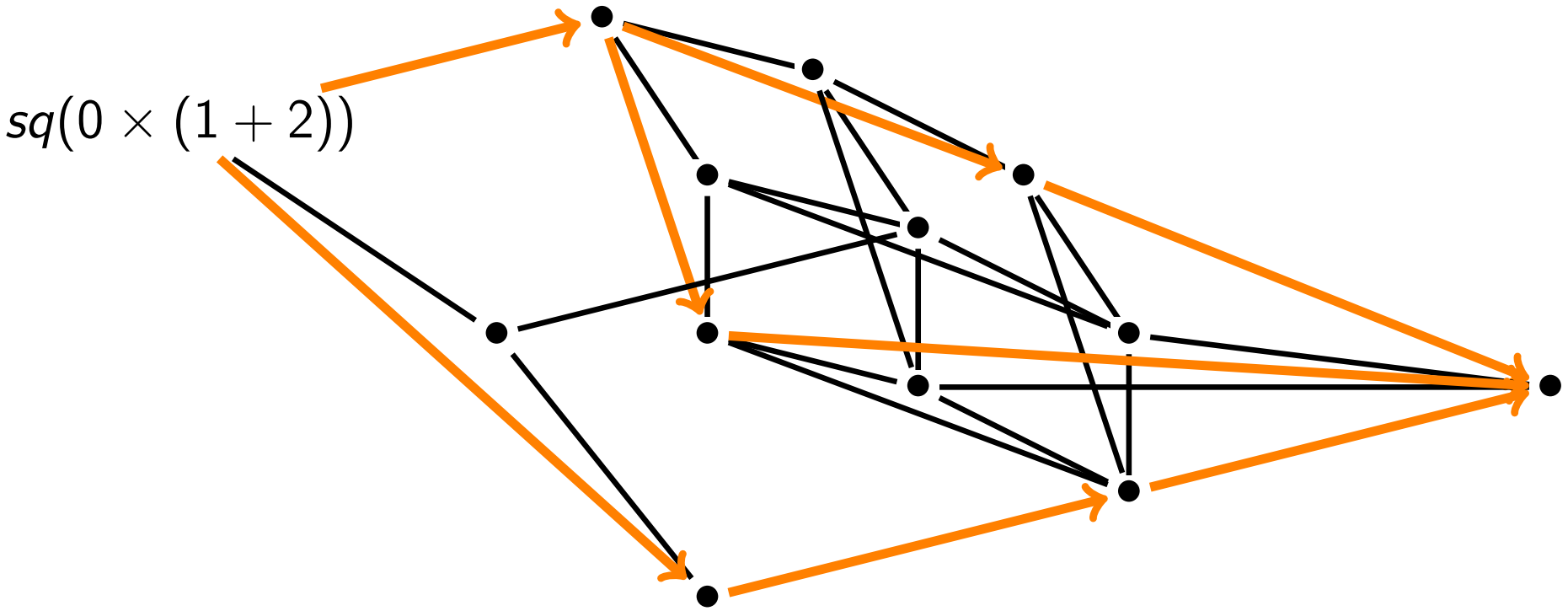
- Outermost prevents unneeded computations.
- Innermost prevents duplication of subprograms.

$$sq(e) \;=\; e \times e$$

$$\begin{tikzcd}
 & (1+2)\times(1+2) \arrow[r] \arrow[d] & 3\times(1+2) \arrow[d] & \\
sq(1+2) \arrow[ur] \arrow[dr] & (1+2)\times 3 \arrow[r] & 3\times 3 \arrow[r] & 9 \\
 & sq(3) \arrow[ur] & &
\end{tikzcd}$$

Different paths, different distances

- Outermost prevents unneeded computations.
- Innermost prevents duplication of subprograms.



Richer programming languages

- Functions
- Data structures

```
map(f, []      )  =  []
map(f, x : xs)  =  f(x) : map(f, xs)

> map(sq, 1 : 2 : 3 : [])
1 : 4 : 9 : []
```

Richer programming languages

- Functions
- Data structures

```
map(f, []      )  =  []
map(f, x : xs)  =  f(x) : map(f, xs)

> map(sq, 1 : 2 : 3 : [])
1 : 4 : 9 : []
```

Second-order rewriting
(Combinatory Reduction Systems, aka CRS, **Klop 1980**)

Three historical results on functions

A simple formalism with a complex behaviour

Thesis & Approach

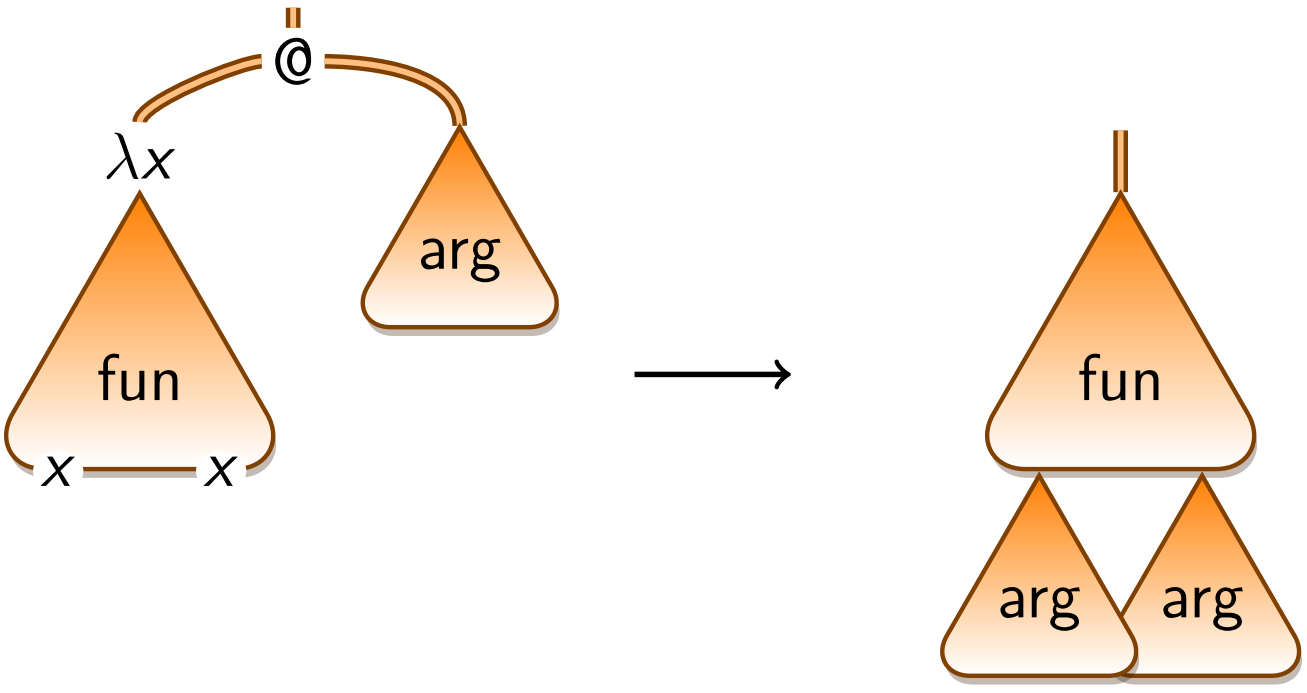
Lévy-optimality and causality

Sharing-via-Labelling

Lambda-lifting

Lambda-calculus: computing with functions

Church, 1936



$(\lambda x.f)a$ $\qquad$ $f^{\{x := a\}}$

$(\lambda x.xx)((\lambda y.y)a)$ $\qquad$ $((\lambda y.y)a)((\lambda y.y)a)$

Big picture: the λ-calculus

Lévy's optimality

Wadsworth's full laziness

Shortest simple path

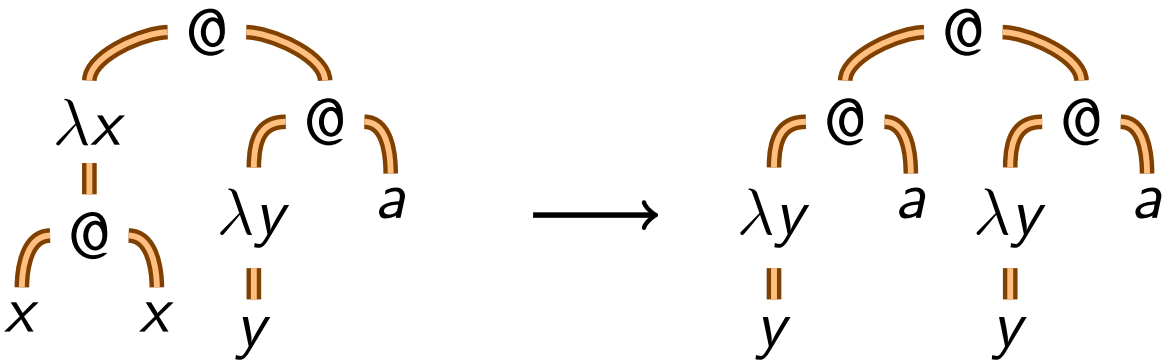
Shortest simple path

$$
\begin{tikzcd}
 & ((\lambda y.y)a)((\lambda y.y)a) \arrow[r] \arrow[d] & a((\lambda y.y)a) \arrow[d] \\
(\lambda x.xx)((\lambda y.y)a) \arrow[ur] \arrow[dr] & ((\lambda y.y)a)a \arrow[r] & aa \\
 & (\lambda x.xx)a \arrow[ur] &
\end{tikzcd}
$$

Theorem: uncomputability (Barendregt et al. 1976)

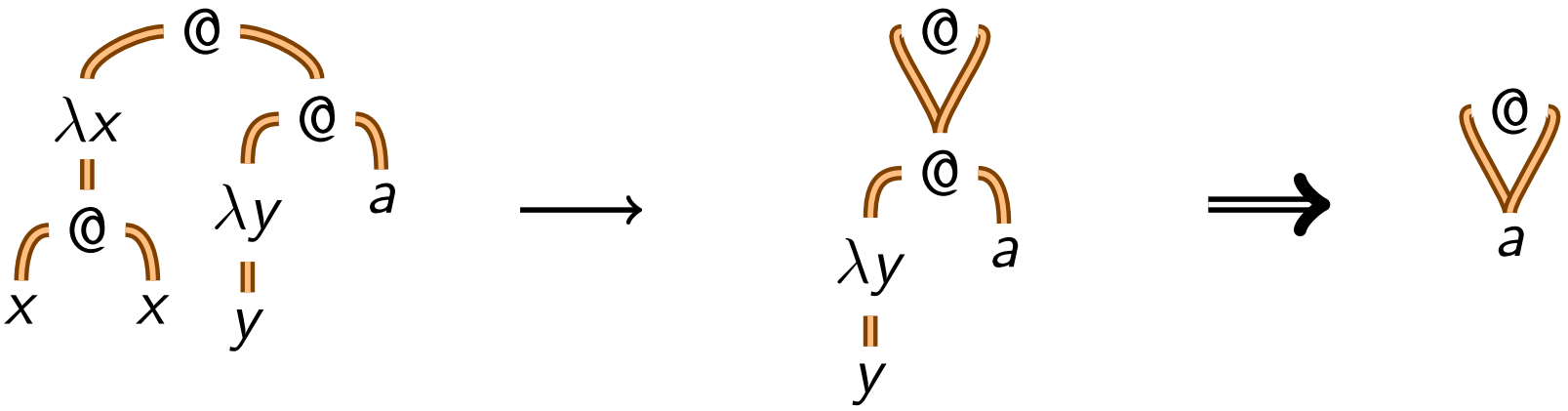
Optimal strategies for the λ-calculus
cannot be computable.

Wadsworth's full laziness (1971)



$$
\begin{tikzcd}
 & ((\lambda y.y)a)((\lambda y.y)a) \arrow[r] \arrow[d] & a((\lambda y.y)a) \arrow[d] \\
(\lambda x.xx)((\lambda y.y)a) \arrow[ur] \arrow[dr] & ((\lambda y.y)a)a \arrow[r] & aa \\
 & (\lambda x.xx)a \arrow[ur] &
\end{tikzcd}
$$

Wadsworth's full laziness (1971)



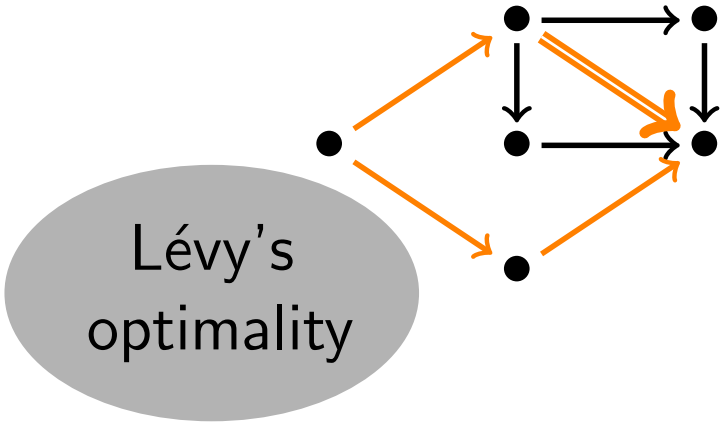
$$
\begin{tikzcd}
 & ((\lambda y.y)a)((\lambda y.y)a) \arrow[dr, Rightarrow] & \\
(\lambda x.xx)((\lambda y.y)a) \arrow[ur] \arrow[dr] & & aa \\
 & (\lambda x.xx)a \arrow[ur] &
\end{tikzcd}
$$

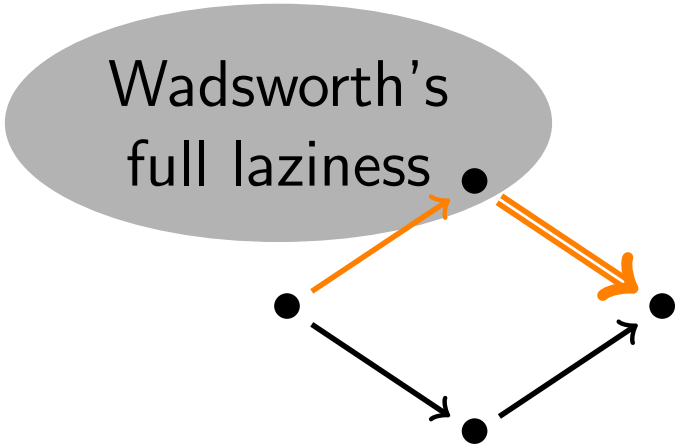
Lévy's optimality (1978)

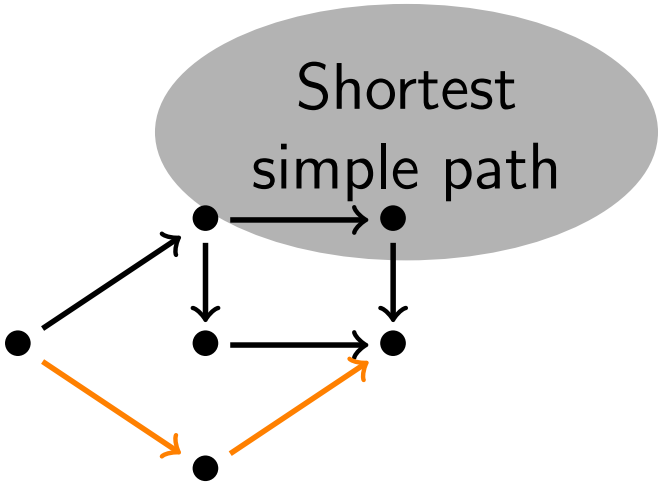
Allow the sharing of any two redexes
that could have been reduced in one step.

$$
\begin{tikzcd}
 & ((\lambda y.y)a)((\lambda y.y)a) \arrow[r] \arrow[d] \arrow[dr, Rightarrow] & a((\lambda y.y)a) \arrow[d] \\
(\lambda x.xx)((\lambda y.y)a) \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & ((\lambda y.y)a)a \arrow[r] & aa \\
 & (\lambda x.xx)a \arrow[ur, Rightarrow] &
\end{tikzcd}
$$

Big picture: the λ-calculus







Reduction strategies

Three historical results on functions

Thesis & Approach

Restrict. Simplify. Unify.

Lévy-optimality and causality

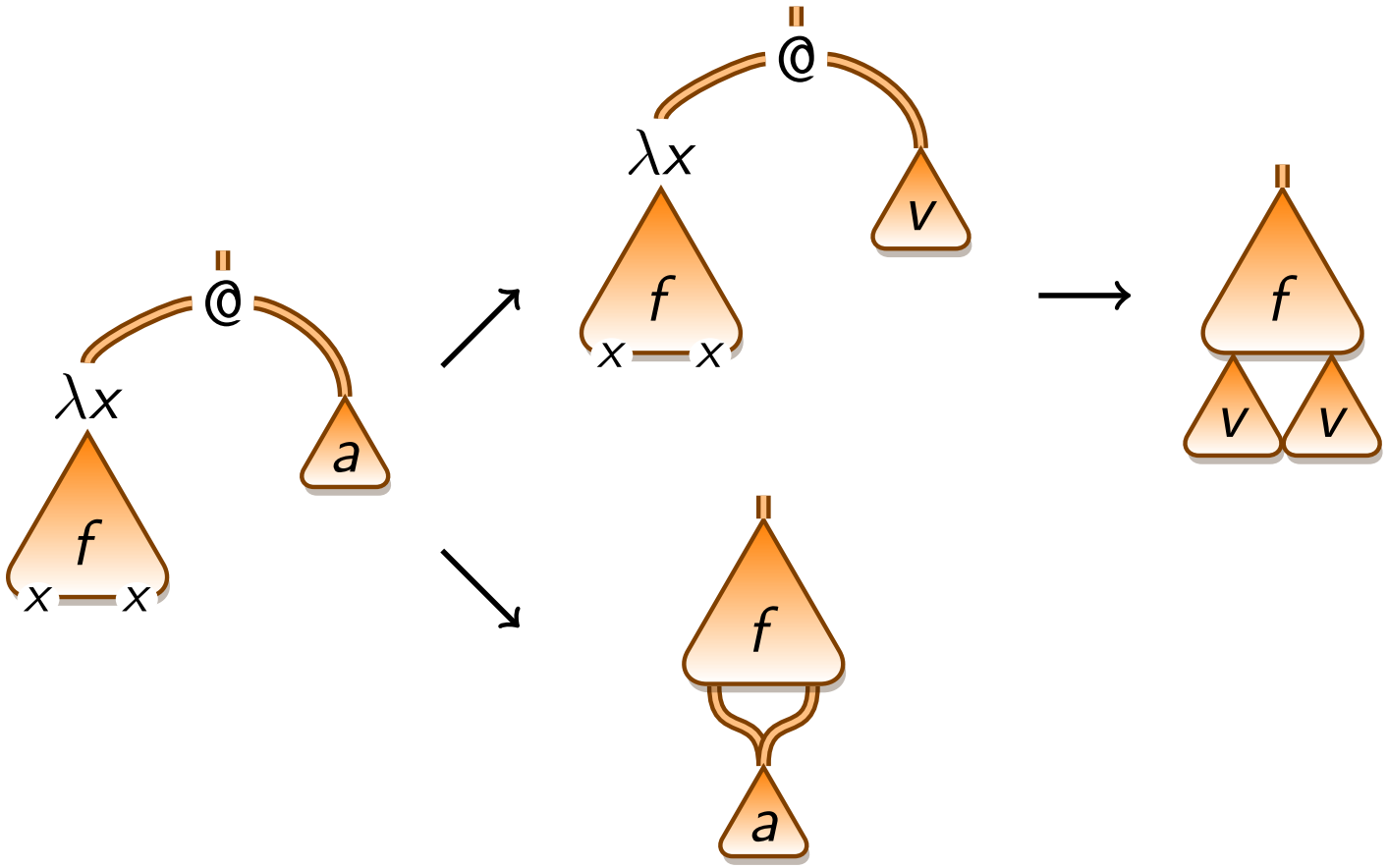
Sharing-via-Labelling

Lambda-lifting

Real compilers use weak reduction

Naive notion: not inside functions.

OCaml (Call-by-Value)

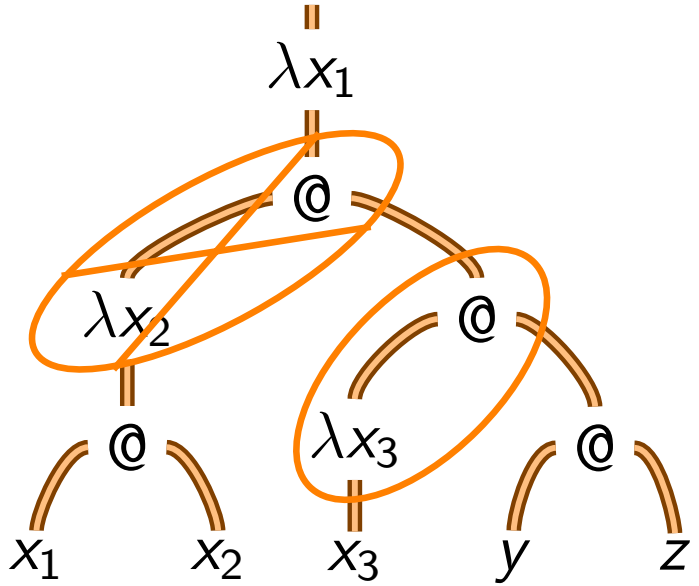


Haskell (Call-by-Need)

Hypothesis: weak reduction

Howard (1968); Çagman & Hindley (1998)

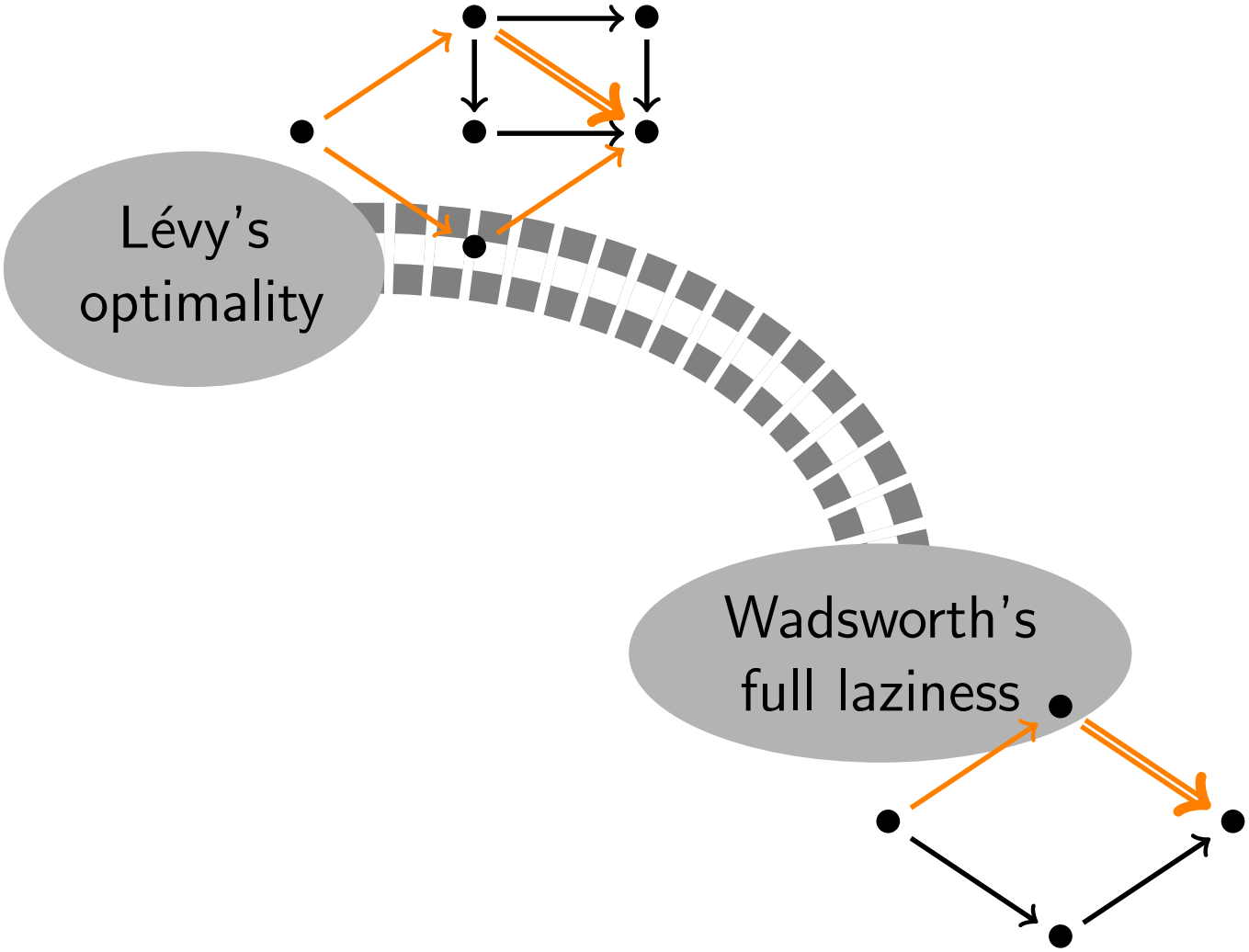
Reduction is forbidden
within the *skeleton* of a function.



Two sides of a coin

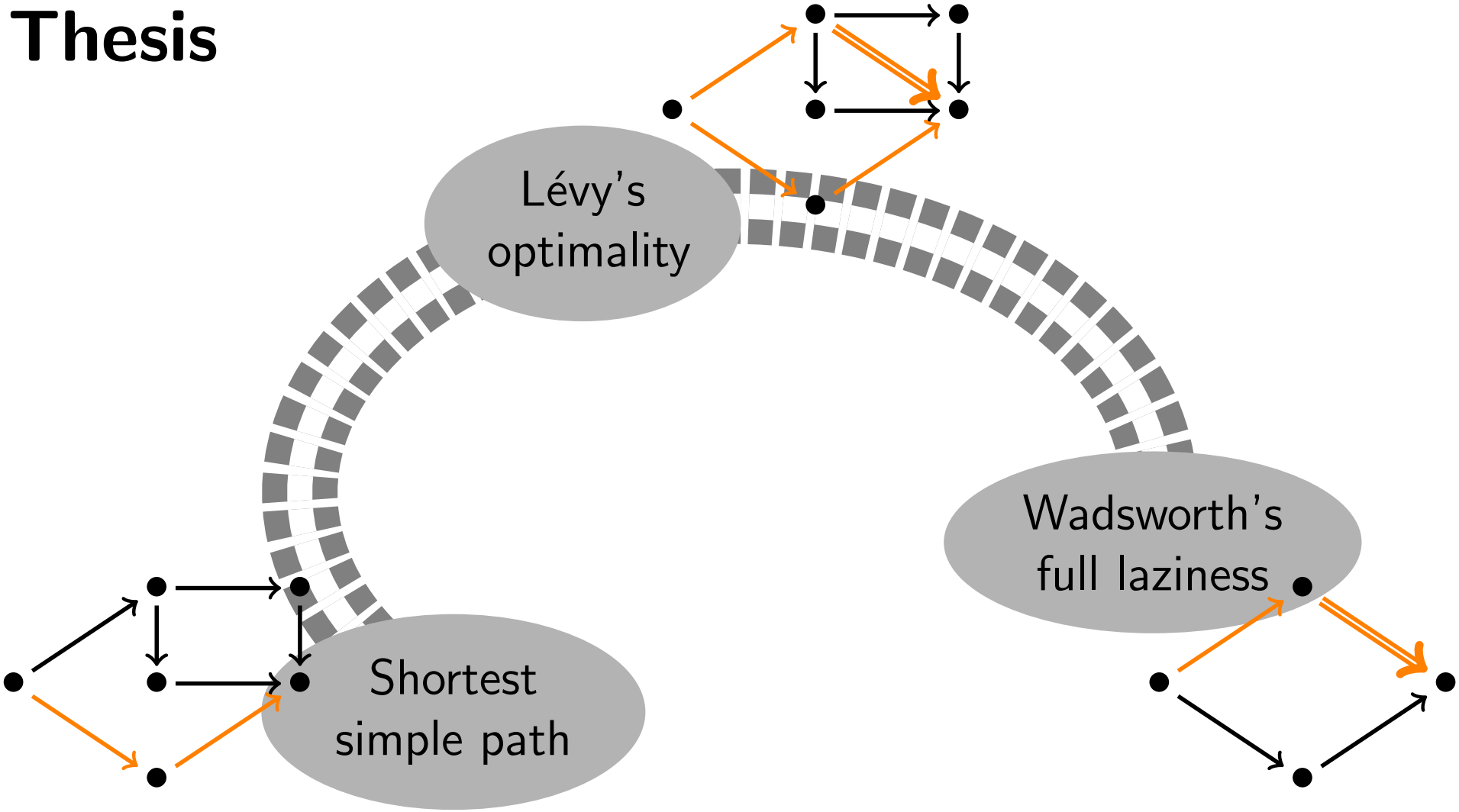
Skeletons Free expressions

Thesis



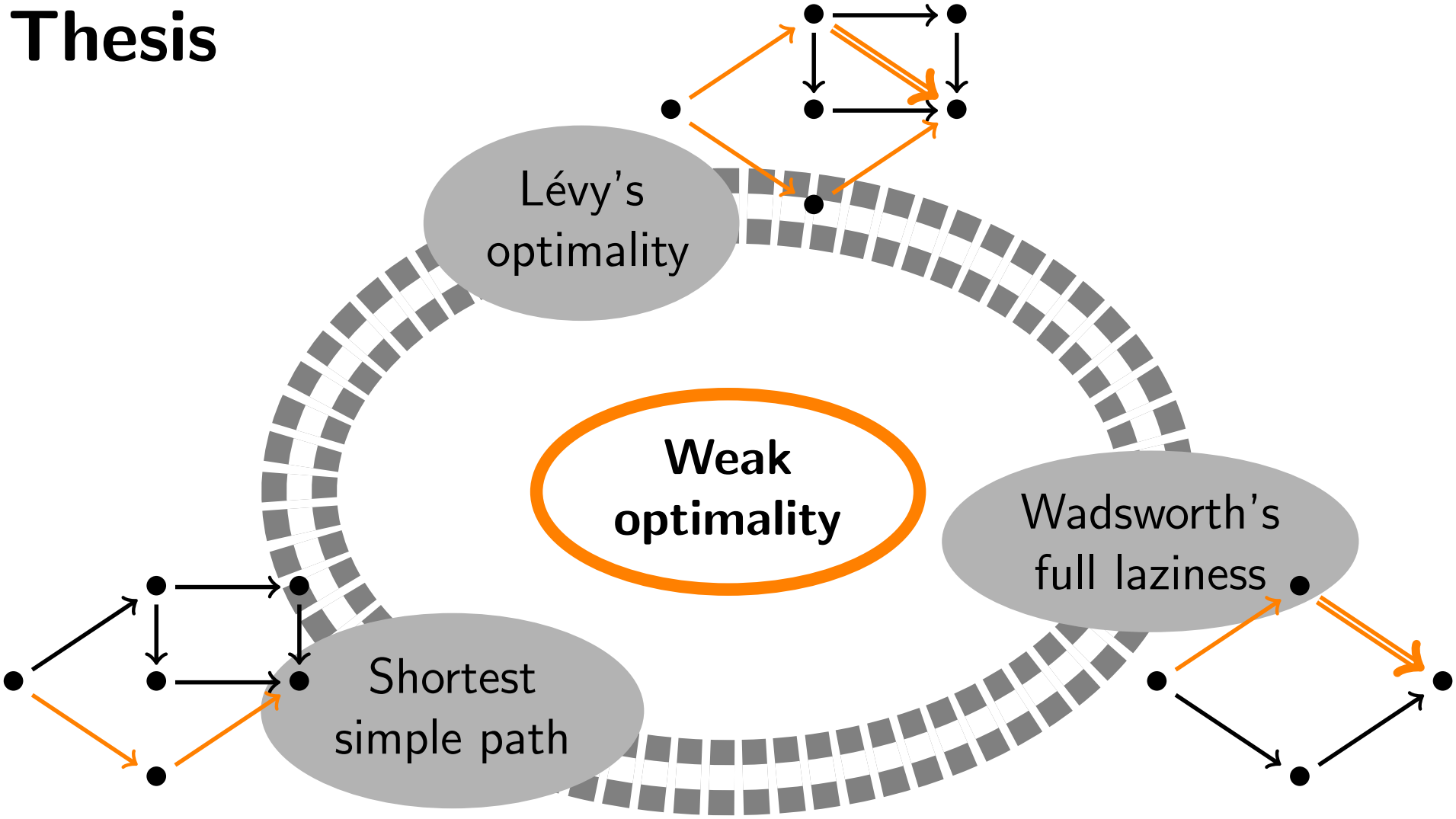
Wadsworth's full laziness
implements Lévy's optimality
in **weak** orthogonal **CRSs**.

Thesis



Wadsworth's full laziness
implements Lévy's optimality
in **weak** orthogonal **CRSs**.

Thesis



Wadsworth's full laziness
implements Lévy's optimality
in **weak** orthogonal **CRSs**.

Lévy-optimality and causality

An event is characterized by its causes.

Residuals up to permutation

Causal criterion

Two redexes are the same event iff:

- they have a **common ancestor**.
- they have been **created** by the same **cause**.

Rule κ_d:	$d(Z)$	$\longrightarrow$	$f(Z,Z)$
Rule κ_a:	a	$\longrightarrow$	b
Rule κ_b:	b	$\longrightarrow$	c

$$
\begin{tikzcd}
d(a) \arrow[r, "\kappa_d"] \arrow[dr, "\kappa_a"] & f(a,a) \arrow[r, two heads, "\kappa_a"] & f(b,b) \arrow[r, two heads, "\kappa_b"] & f(c,c) \\
& d(b) \arrow[ur, "\kappa_d"] \arrow[r, "\kappa_b"] & d(c) \arrow[ur, "\kappa_d"] &
\end{tikzcd}
$$

Understanding a process

Which computation steps are relevant?

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow$

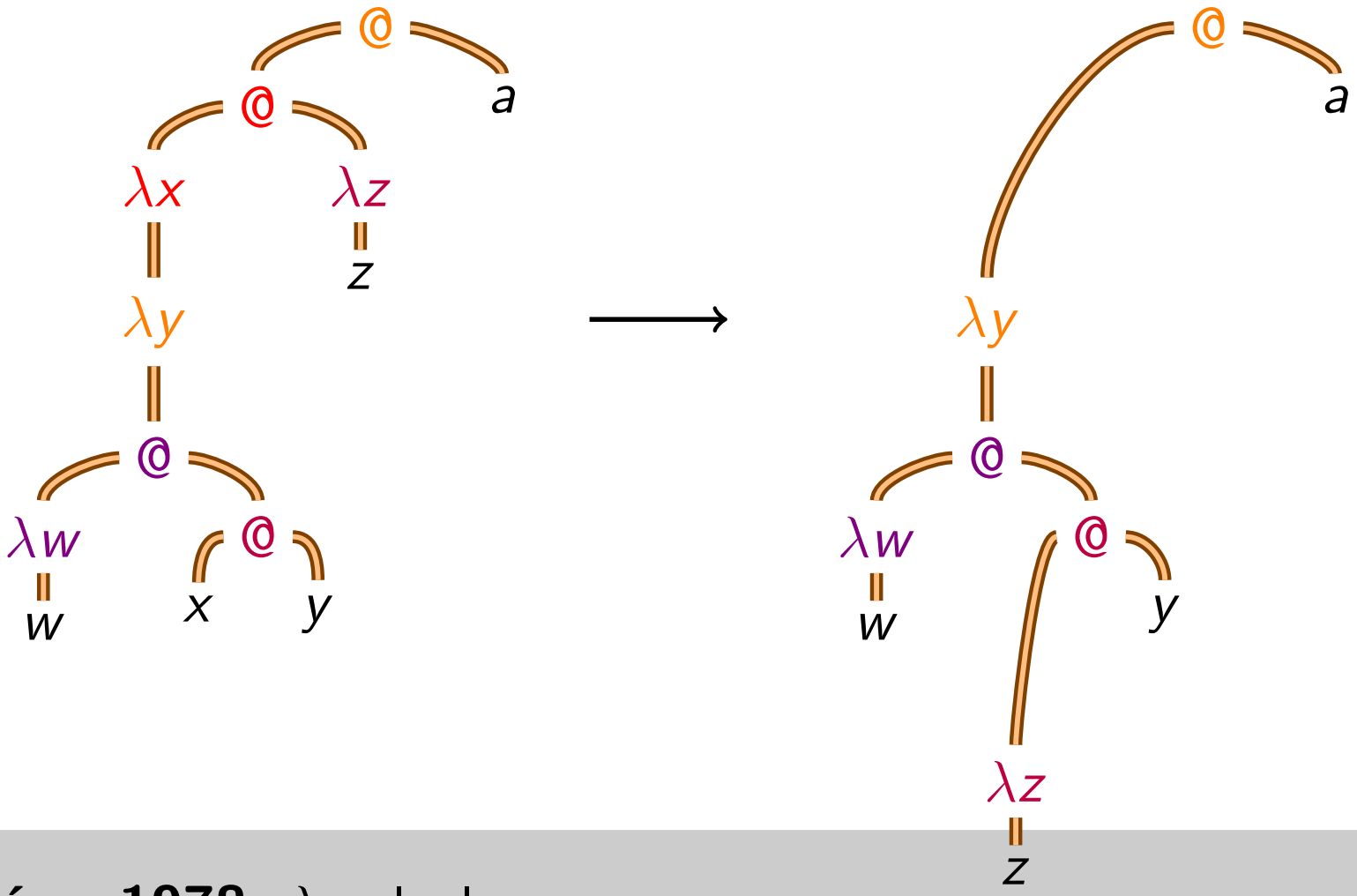
Back to basics

- analysis of direct causes and effects.
- how can redexes be created?

And later

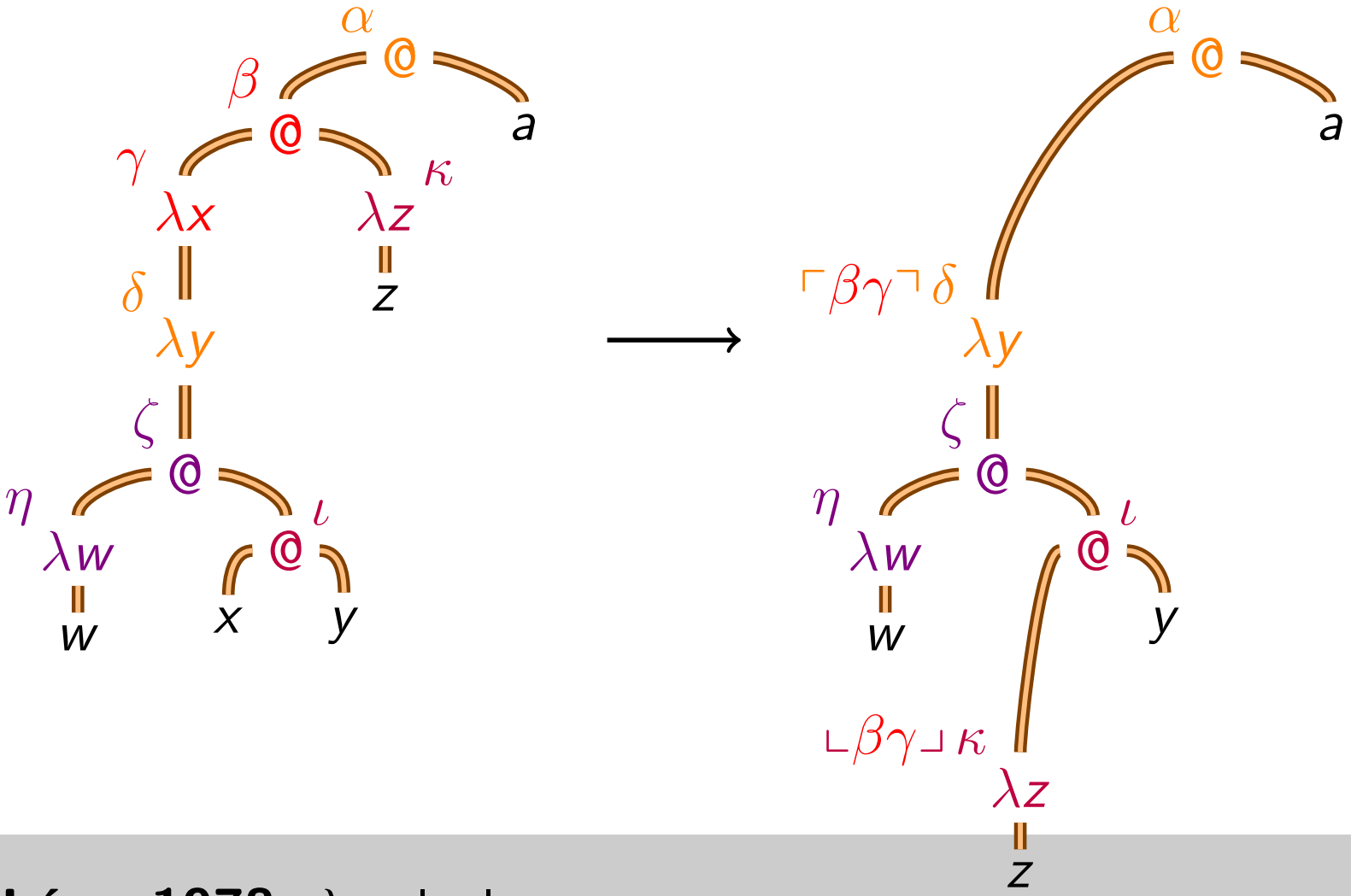
- transitivity of causal dependencies.
- recording in labels.

Redex creation



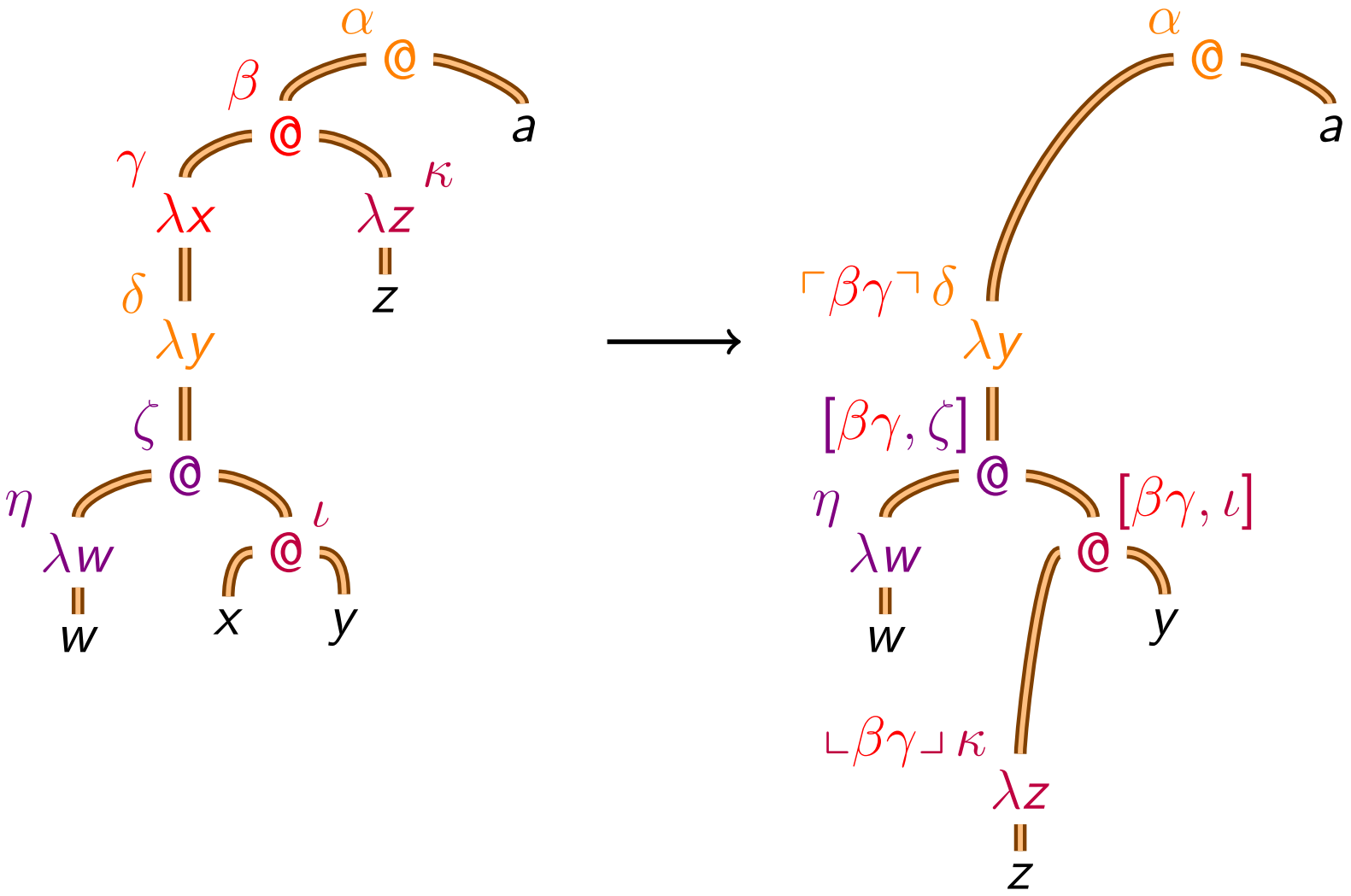
- **Lévy, 1978**: λ-calculus.
- **Maranget, 1992**: first-order rewriting.
- **van Oostrom, 1996**: higher-order rewriting.

Redex creation



- **Lévy, 1978**: λ-calculus.
- **Maranget, 1992**: first-order rewriting.
- **van Oostrom, 1996**: higher-order rewriting.

Weak reduction: new creation case



- **Blanc, Lévy & Maranget, 2005**: weak λ-calculus.
- **Balabonski (Ch. 7)**: weak CRSs.

Causality and optimality

Axiomatic framework (Glauert & Khasidashvili 1996)

- Input: $\begin{cases} \text{Family relation on redexes,} \\ \text{Contribution relation between families,} \\ \ldots \text{ both enjoying good properties} \end{cases}$
- Output: needed complete family reduction is optimal.

Checking the axioms (Balabonski, Ch. 5, 7, 10 [PPDP'10])

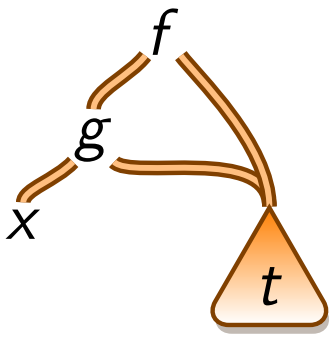
- Families and contribution defined by labels.
- Theorems of causal correctness and completeness of the labels.

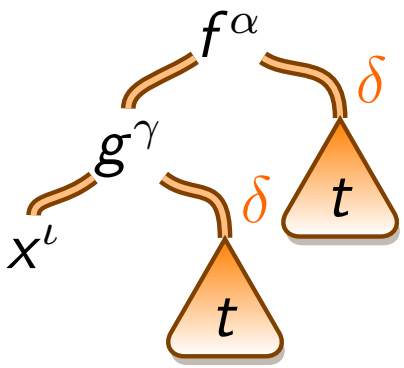
Sharing-via-Labelling

Labelled terms can represent acyclic graphs.

Labels and graphs

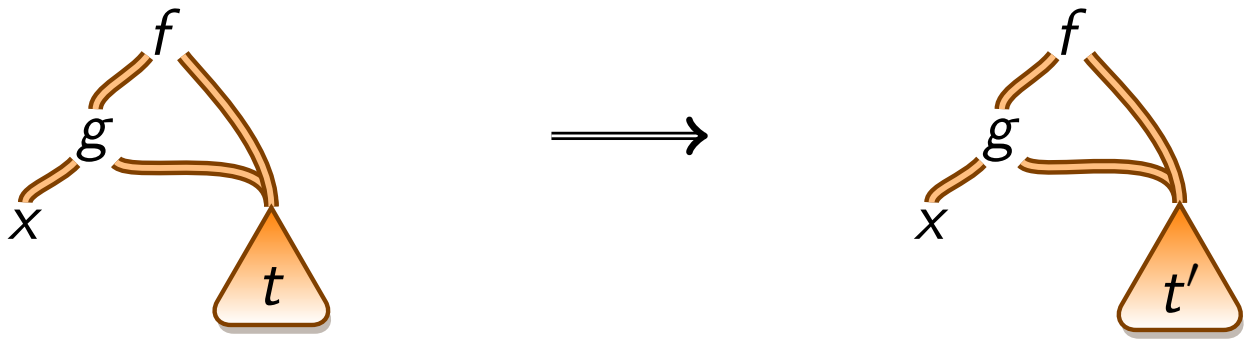
Maranget (1992)





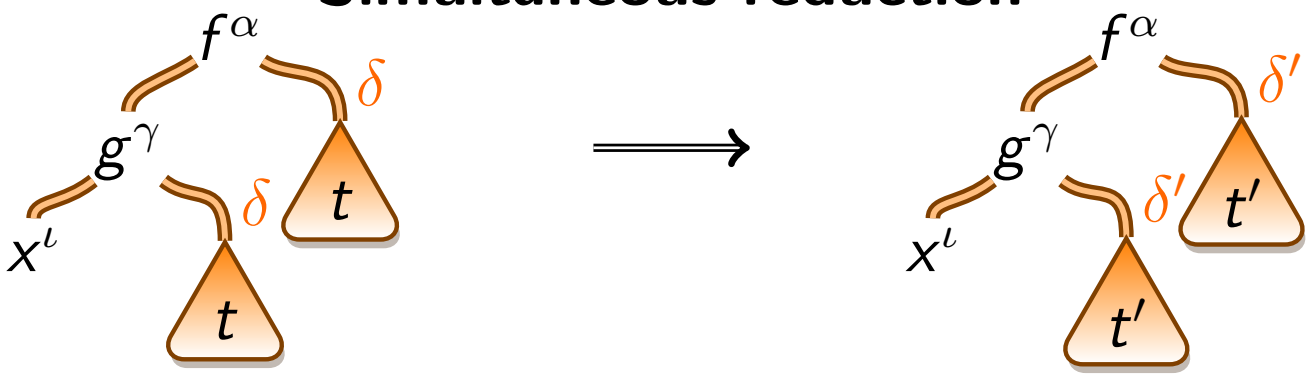
Labels and graphs

Maranget (1992)



Graph reduction

Simultaneous reduction



Requirements

The labelled reduction must **preserve** sharing.

Axioms for sharing-via-labelling

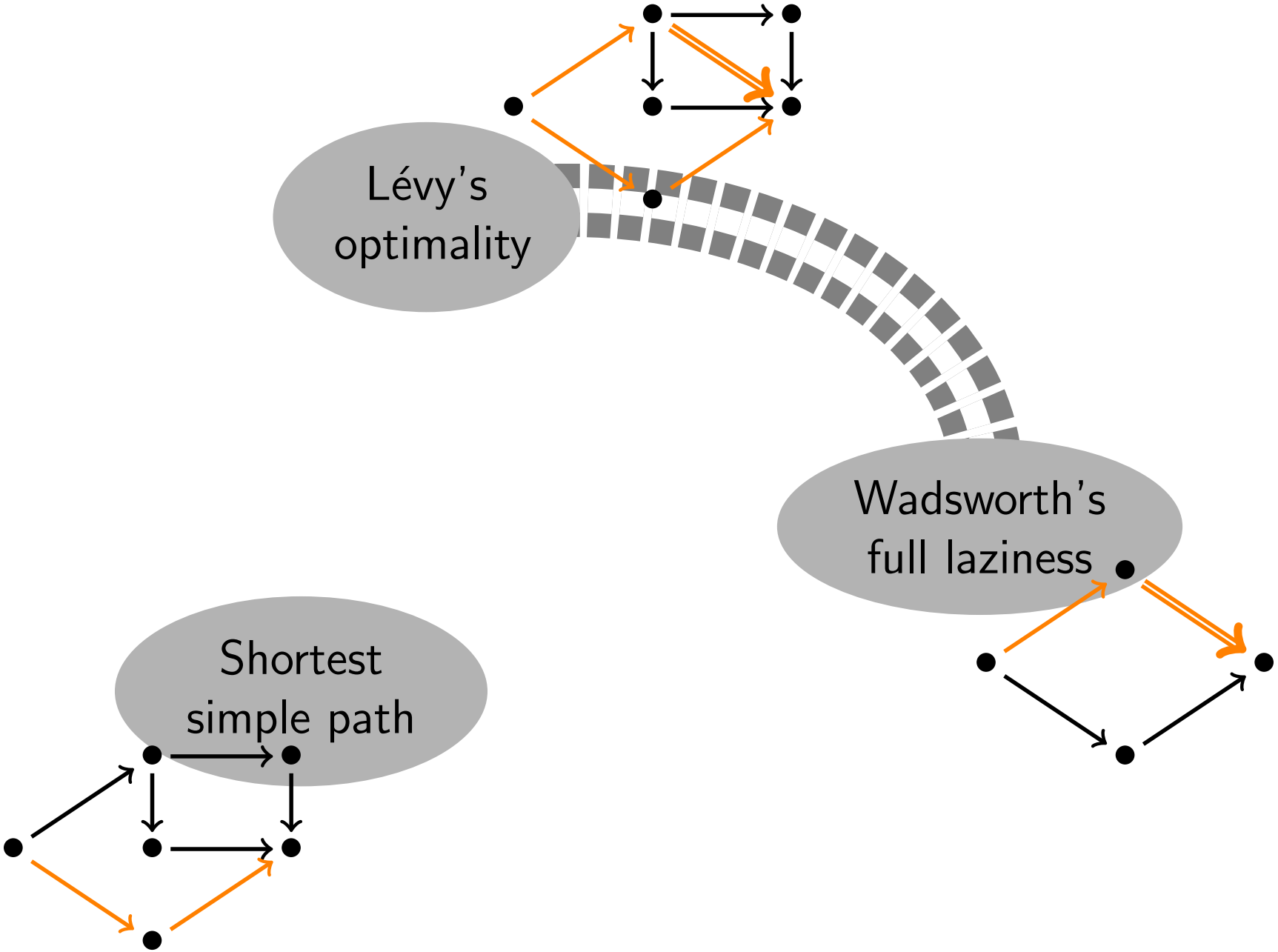
Axiomatic framework (Balabonski, Ch. 1 [RTA'12])

- Input: $\begin{cases} \text{A labelled system,} \\ \text{An order on labels,} \\ \ldots \text{ both enjoying good properties.} \end{cases}$
- Output theorem: labelled reduction preserves sharing.

Checking the axioms

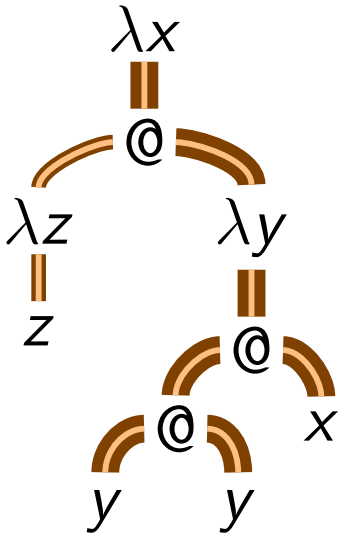
- Almost immediate.

Partial conclusion

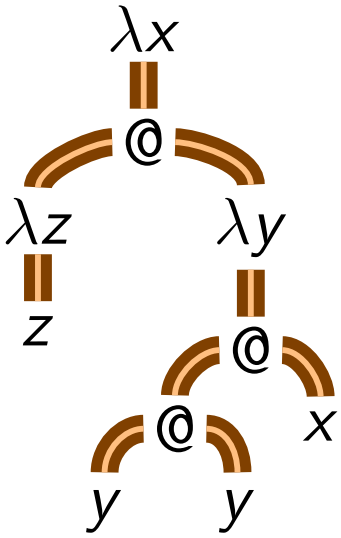


Application: comparing graph systems

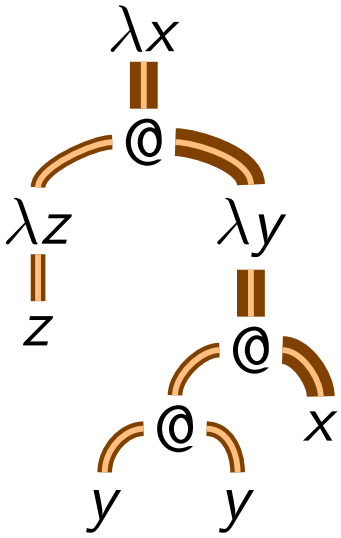
- Full laziness in the literature: 6 different sources.
- Translation to graphs: 3 different systems.



[Wad1971][PJ1987]
[Ses1997][SW2010]



[AF1997]



[BLM2005]

Theorem (Balabonski, Ch. 3 [POPL'12])

The three systems are bisimilar.

Optimality and sharing-via-labelling

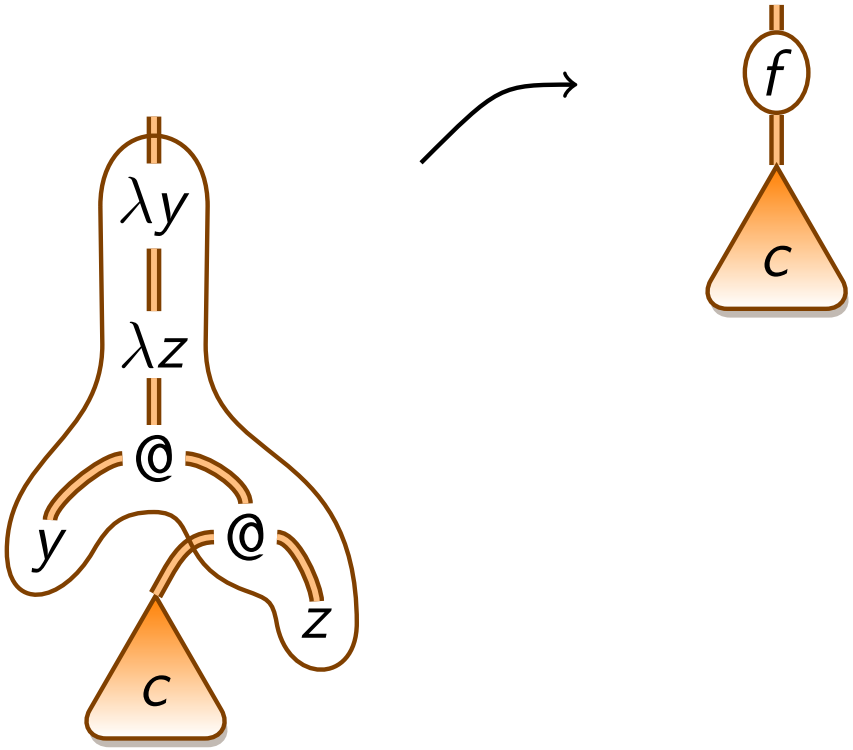
Second order	Weak second order	First order
	Theorem	**Theorem**
Sharing not preserved.	Sharing preserved.	Sharing preserved.
Lévy 1978. van Oostrom 1996.	Blanc, Lévy, Maranget 2005. Balabonski (Ch. 7).	Maranget 1992.

Lambda-lifting

Hiding higher-order

Turning functions into black boxes

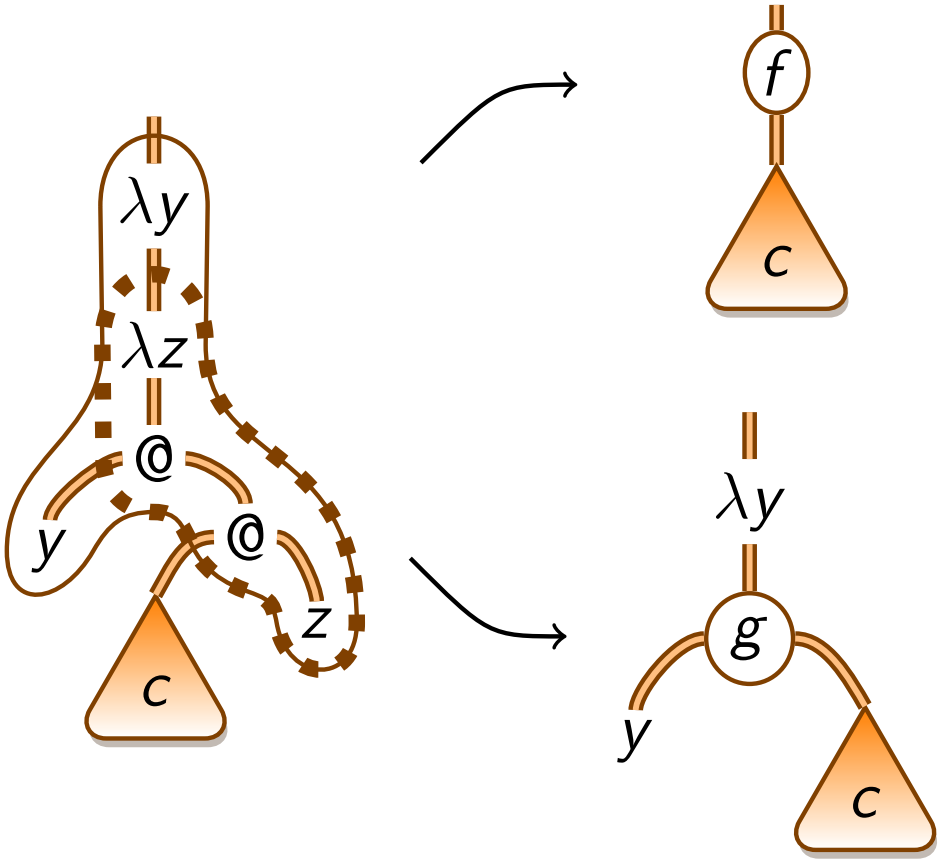
Hughes (1982); Johnsson (1985)



$$f \quad \triangleq \quad \lambda y.\lambda z.@(y, @([], z))$$

Turning functions into black boxes

Hughes (1982); Johnsson (1985)

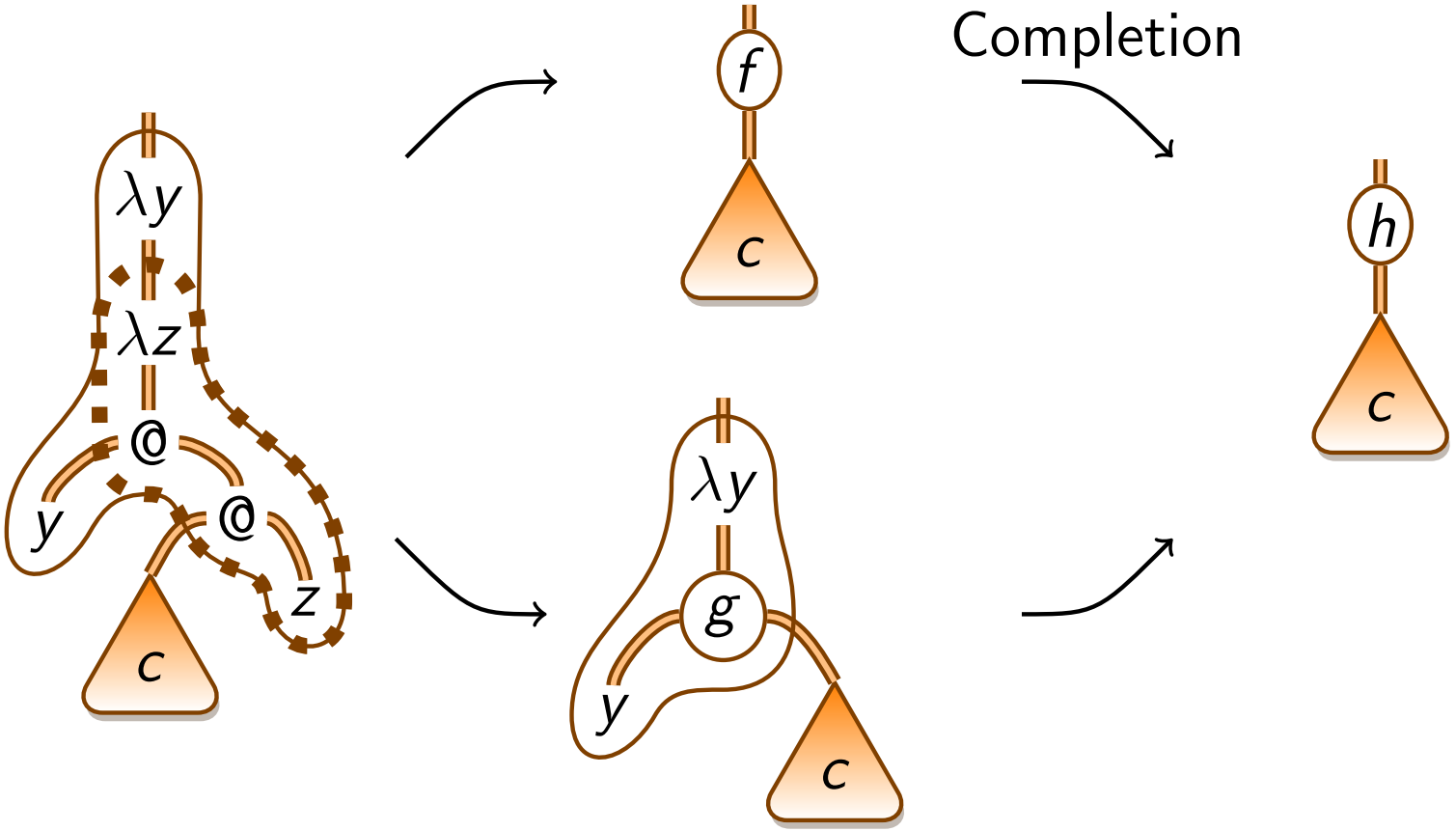


$$f \triangleq \lambda y.\lambda z.@(y, @([], z))$$

$$g \triangleq \lambda z.@([], @([], z))$$

Turning functions into black boxes

Hughes (1982); Johnsson (1985)

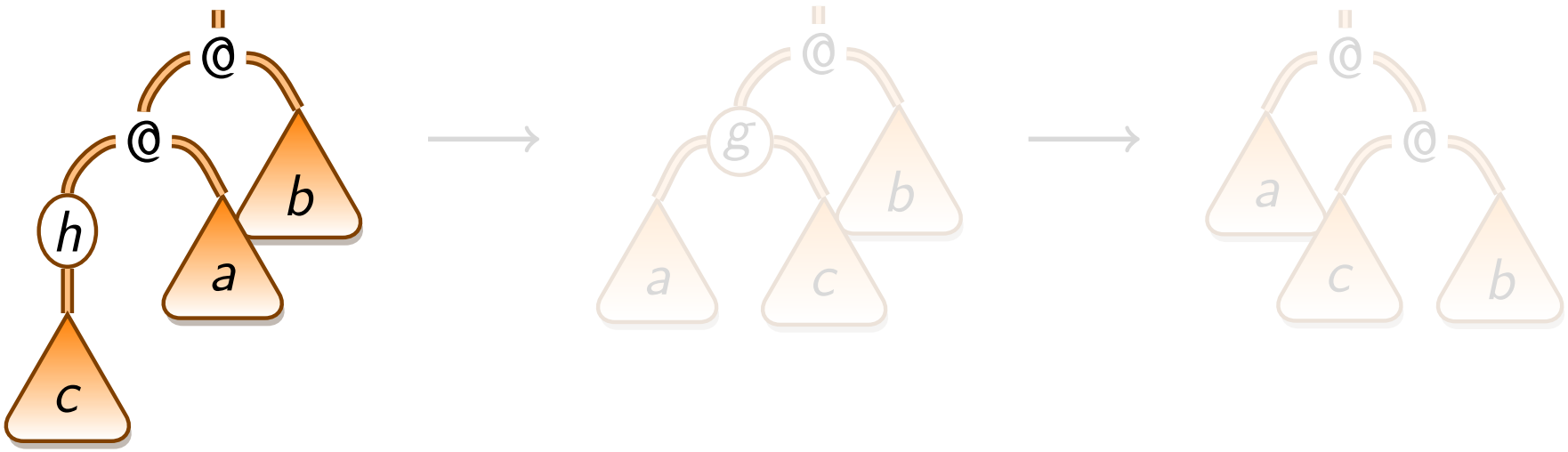


$$
\begin{aligned}
f &\triangleq \lambda y.\lambda z.@(y, @([], z)) \\
g &\triangleq \lambda z.@([], @([], z)) \\
h &\triangleq \lambda y.g(y, [])
\end{aligned}
$$

The target first-order system

$$g \triangleq \lambda z.@([], @([], z))$$

$$h \triangleq \lambda y.g(y, [])$$



Features of the system

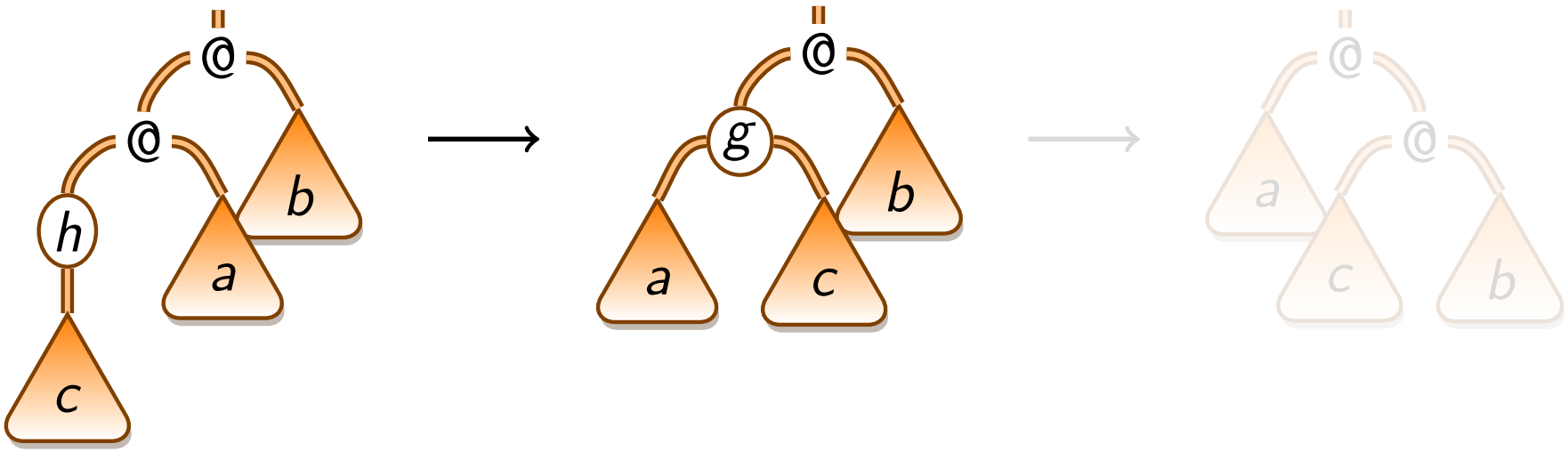
- Left-linear
- Orthogonal

The target first-order system

$$g \triangleq \lambda z.@([], @([], z))$$

$$h \triangleq \lambda y.g(y, [])$$

$$@(h(Z_1), Z_0) \longrightarrow g(Z_0, Z_1)$$

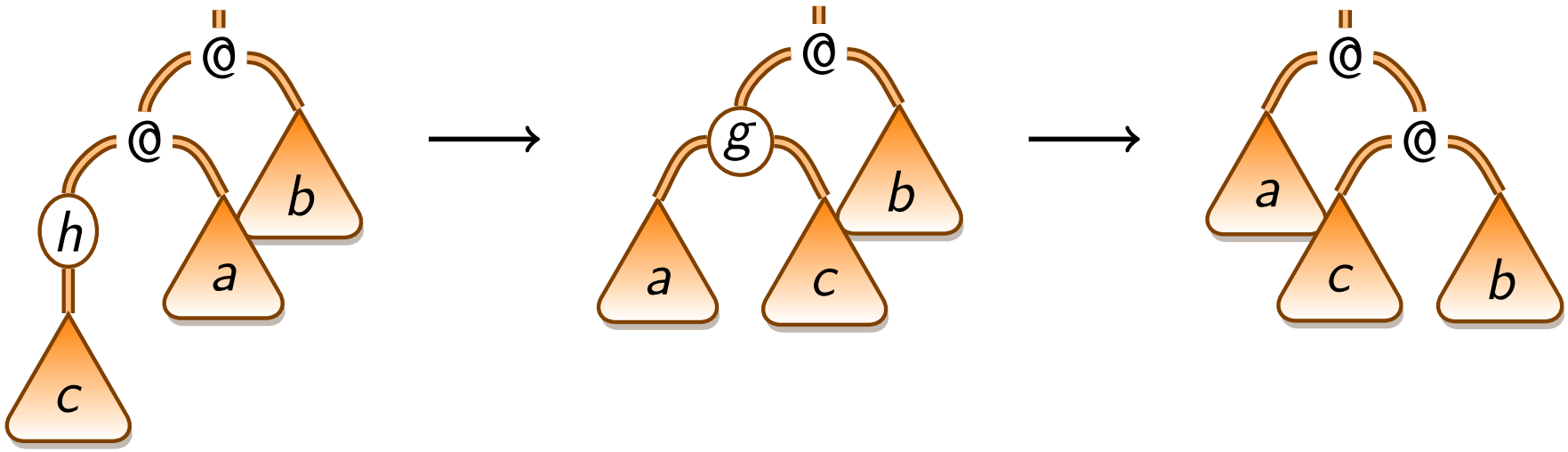


Features of the system

- Left-linear
- Orthogonal

The target first-order system

$g \triangleq \lambda z.@([], @([], z))$	$@(g(Z_1, Z_2), Z_0) \longrightarrow @(Z_1, @(Z_2, Z_0))$
$h \triangleq \lambda y.g(y, [])$	$@(h(Z_1) \quad , Z_0) \longrightarrow g(Z_0, Z_1)$

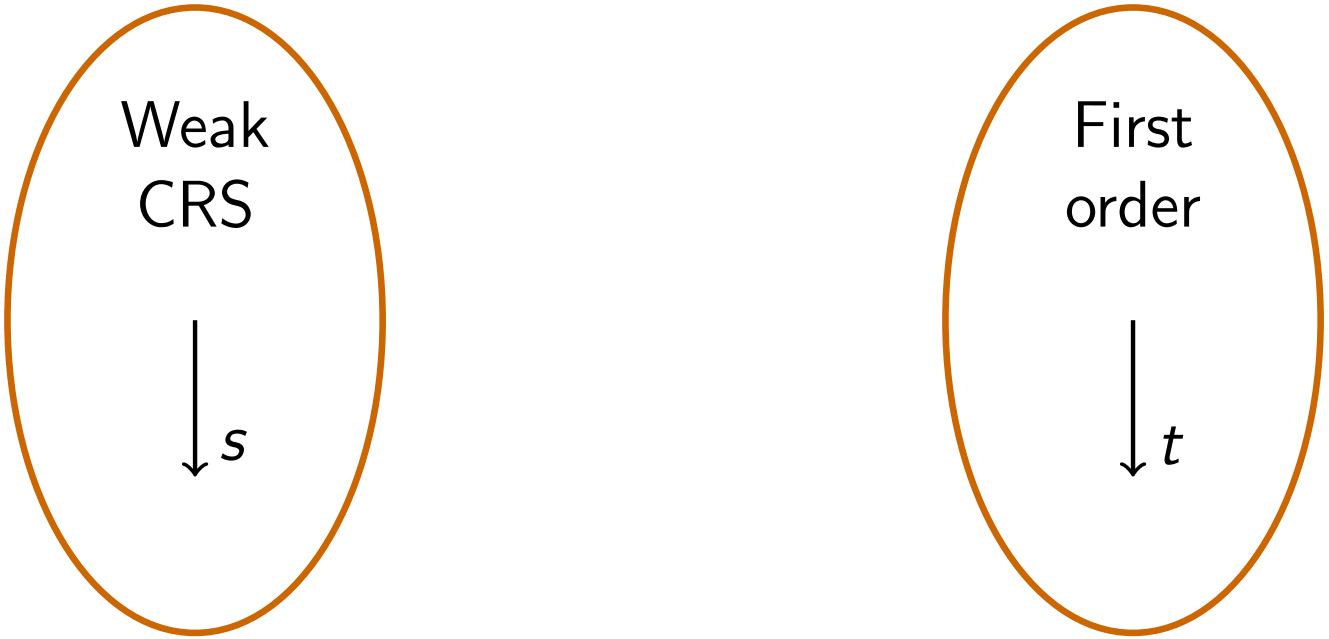


Features of the system

- Left-linear
- Orthogonal

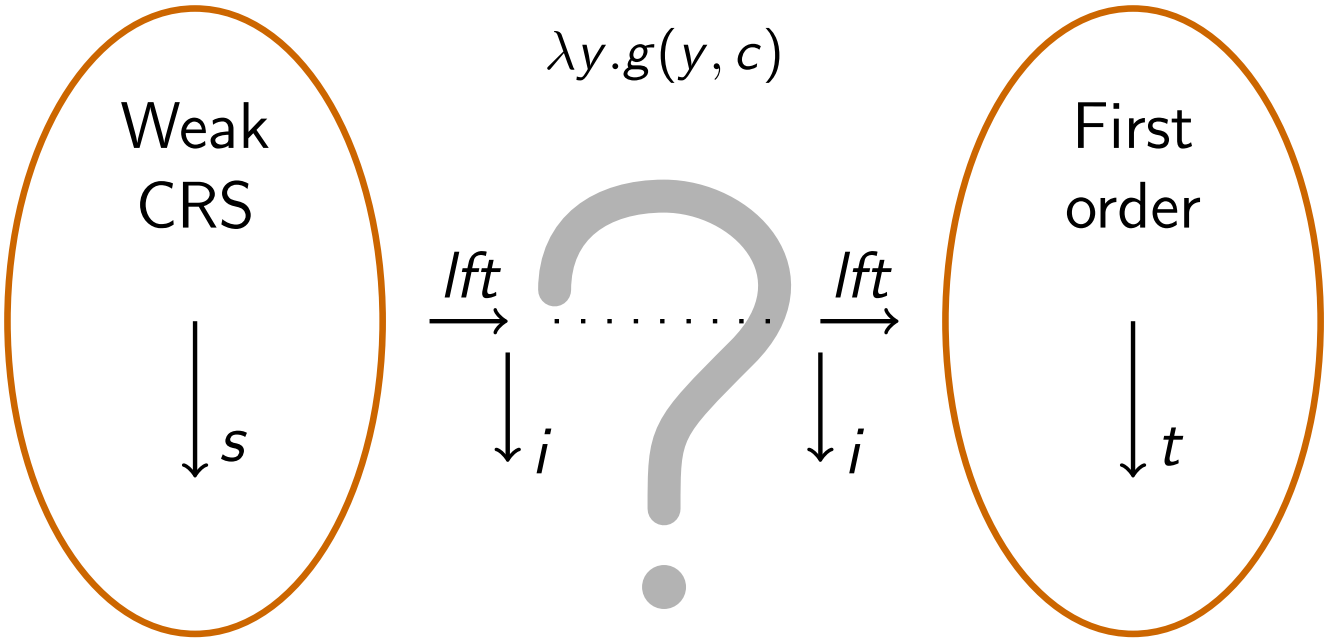
Internalization of the transformation

Balabonski (Ch. 8, 9) [POPL'12]



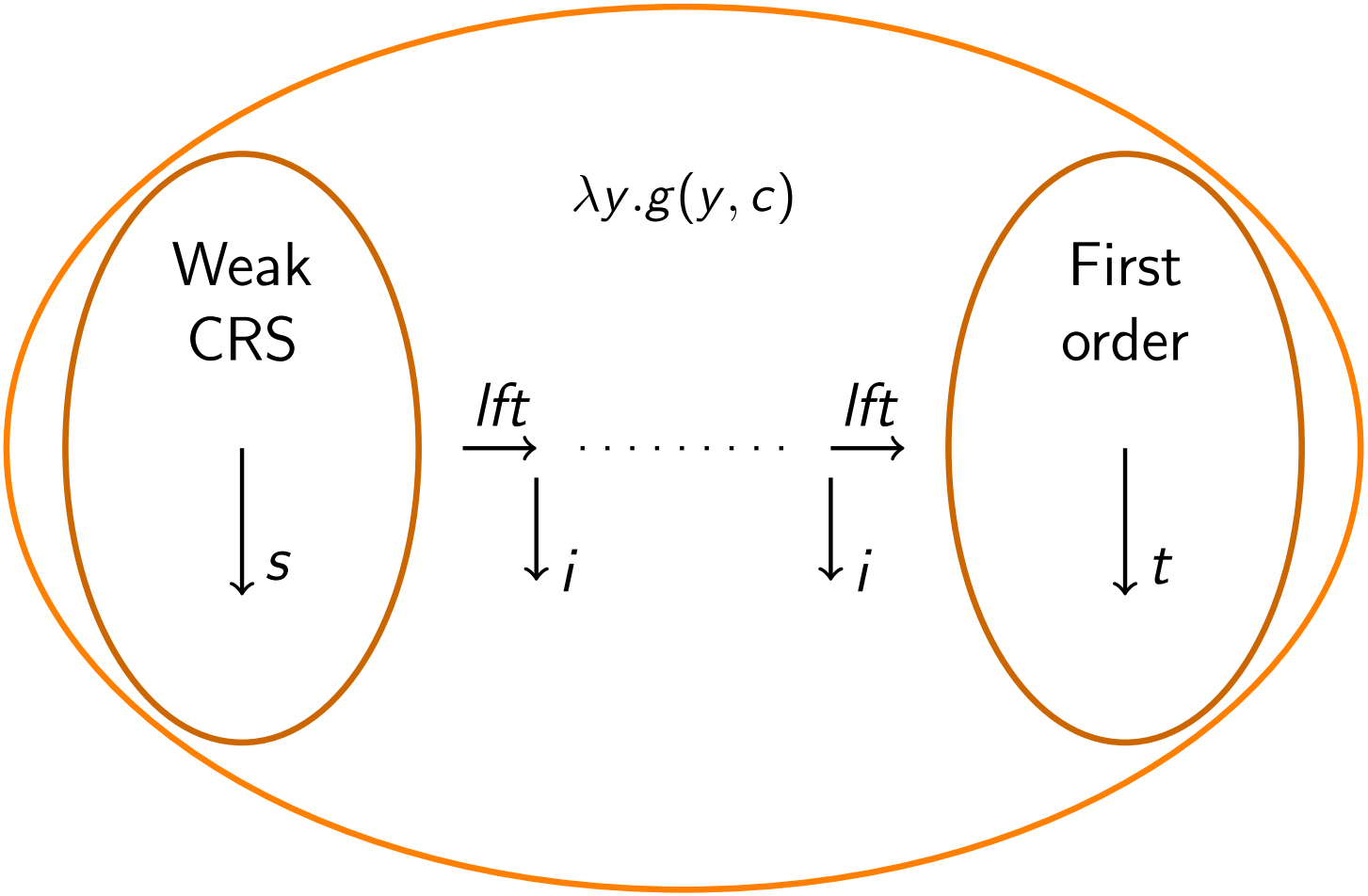
Internalization of the transformation

Balabonski (Ch. 8, 9) [POPL'12]



Internalization of the transformation

Balabonski (Ch. 8, 9) [POPL'12]



Higher-order rewriting
(still CRS)

Bisimulations

Any $\begin{tikzcd} \cdot \arrow[r, "lft"] \arrow[d, "i"'] & \cdot \\ \cdot & \end{tikzcd}$ can be closed as $\begin{tikzcd} \cdot \arrow[r, "lft"] \arrow[d, "i"'] & \cdot \arrow[d, orange, "i"] \\ \cdot \arrow[r, orange, two heads, "lft"'] & \cdot \end{tikzcd}$

Any $\begin{tikzcd} \cdot \arrow[r, "lft"] & \cdot \arrow[d, "i"] \\ & \cdot \end{tikzcd}$ can be closed as $\begin{tikzcd} \cdot \arrow[r, "lft"] \arrow[d, orange, "i"'] & \cdot \arrow[d, "i"] \\ \cdot \arrow[r, orange, two heads, "lft"'] & \cdot \end{tikzcd}$

Theorem (Balabonski, Ch. 8, 9 [POPL'12])

Source and target systems are **bisimilar**.

This means: the λ-lifting transformation is **correct**.

Application: shortest simple path

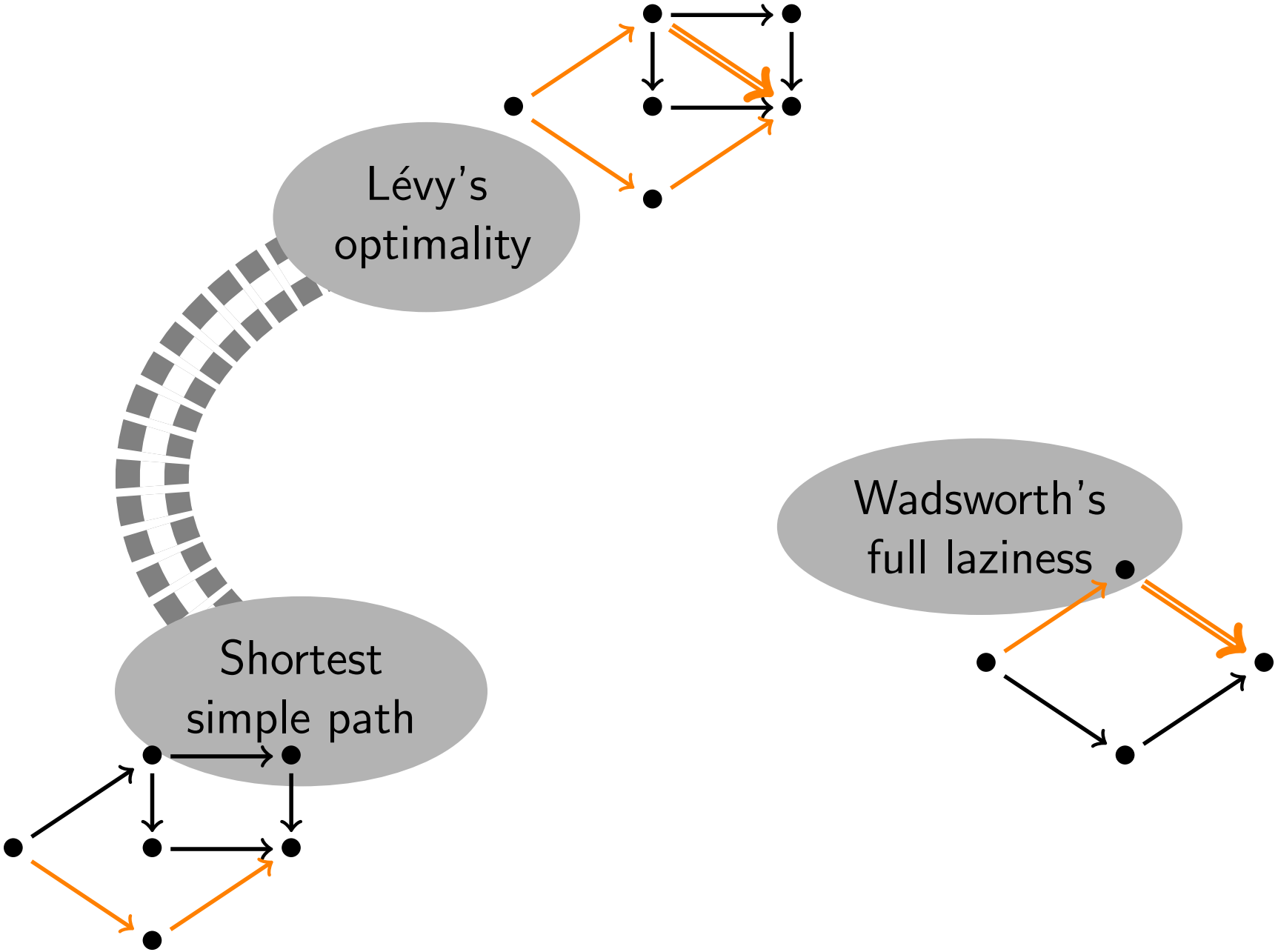
Theorem (Khasidashvili 1993)

Innermost needed reduction
is optimal
in orthogonal first-order rewriting systems.

Transfer (Balabonski, Ch. 10)

Innermost needed reduction
is optimal
in **weak** orthogonal **second-order** rewriting systems.

Partial conclusion



Application: unreachable optimality

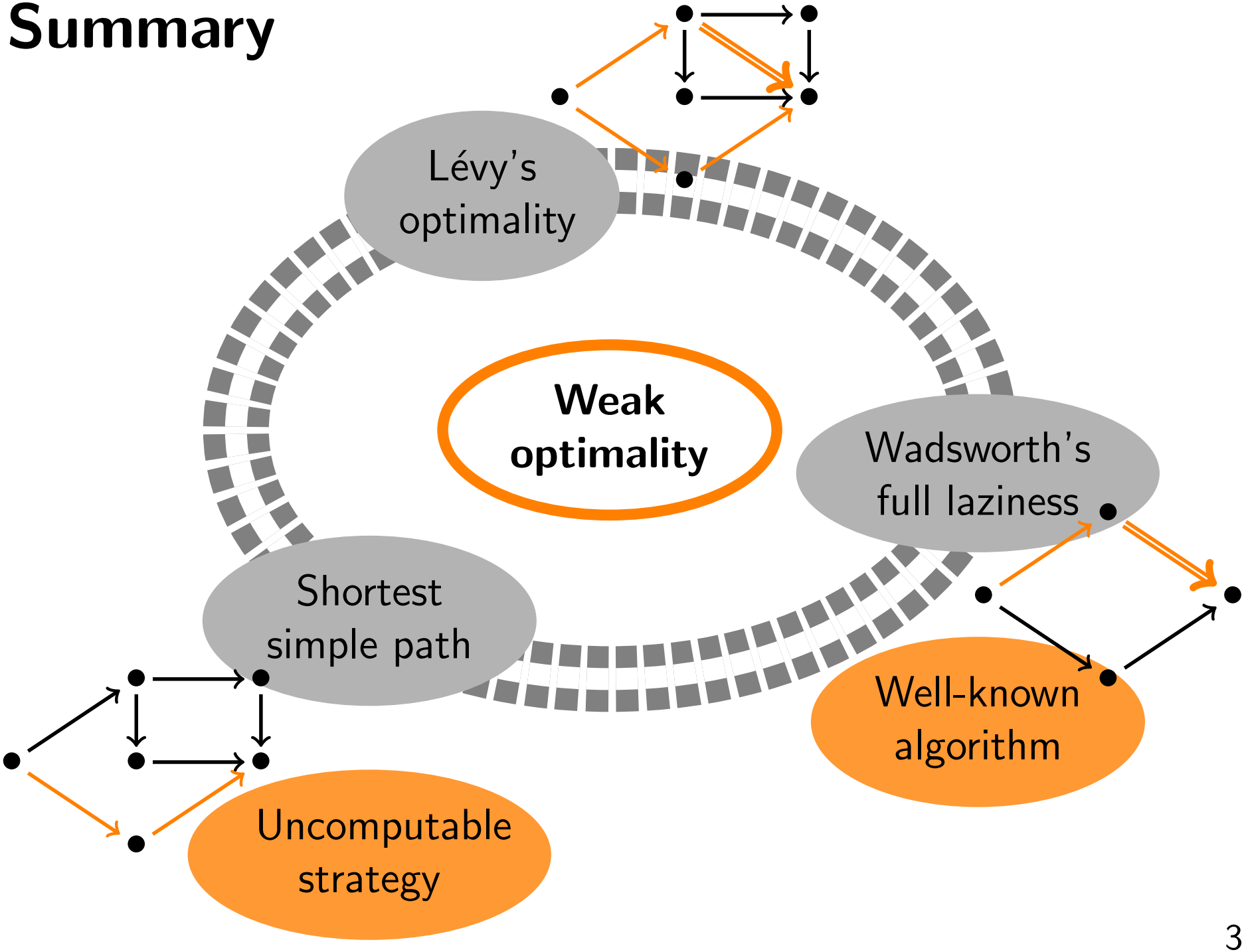
Theorem: uncomputability (Barendregt et al. 1976)

Optimal strategies for the λ-calculus
cannot be computable.

Theorem: uncomputability (Balabonski, Ch. 10)

Optimal strategies for the **weak** λ-calculus
cannot be computable.

Summary


Lévy's optimality
Weak optimality
Wadsworth's full laziness
Shortest simple path
Well-known algorithm
Uncomputable strategy

Further work: beyond optimality

Remark

Non-weak reduction allow shorter reduction paths.

Hint (Balabonski, Ch.11)

Sharing-via-labelling is compatible with non-weak reduction.

Further work: tight complexity bounds

Goal

Link the number of reduction steps to the actual complexity.

Problems

- Cost of sharing.
- Size of the terms/graphs.

Hint (Balabonski, Ch. 4)

Labels controls the growth of the graphs.

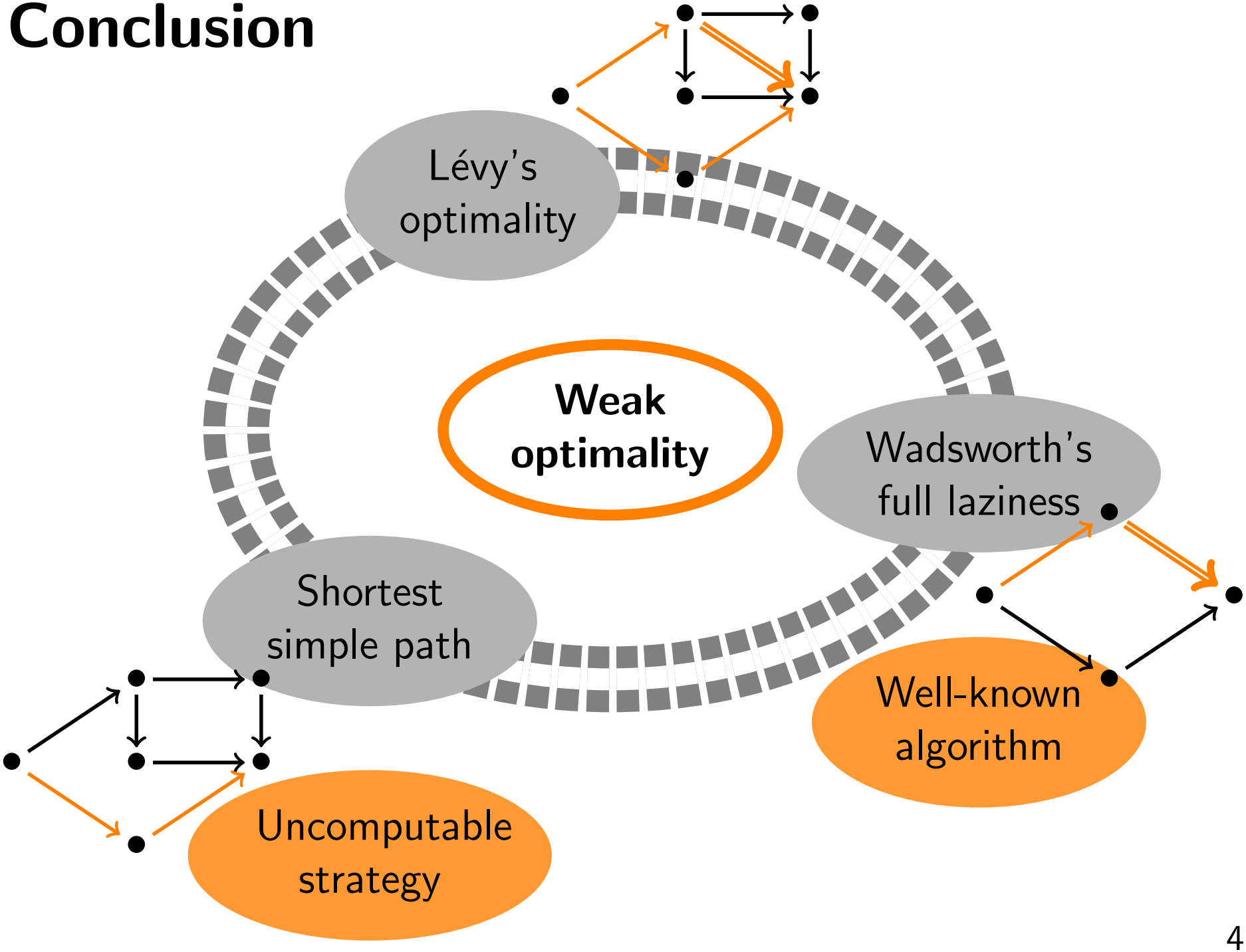
Further work: causality and non-stability

Why does

$$(1-1) \quad \times \quad (2-2)$$

equal 0?

Conclusion


Lévy's optimality
Weak optimality
Wadsworth's full laziness
Shortest simple path
Well-known algorithm
Uncomputable strategy