

Incremental Interactive Computation

Matthew Hammer
hammer@cs.umd.edu

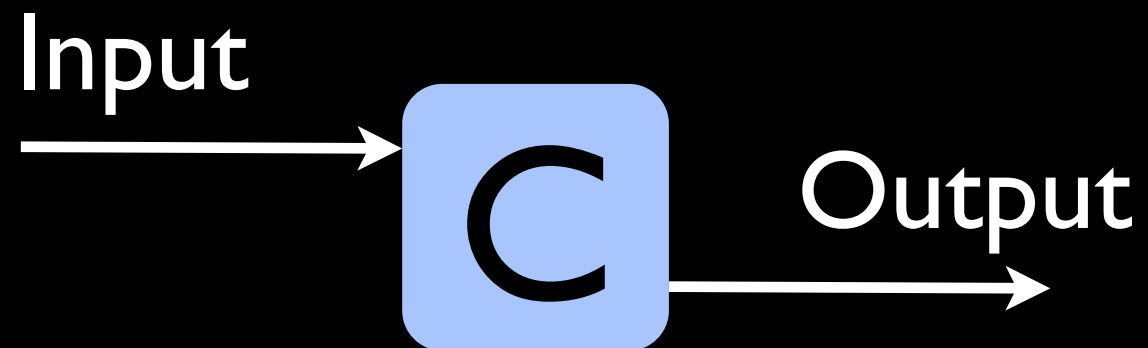
Incremental Interactive Computation

Matthew Hammer
hammer@cs.umd.edu

Batch Computation vs Interactive Computation

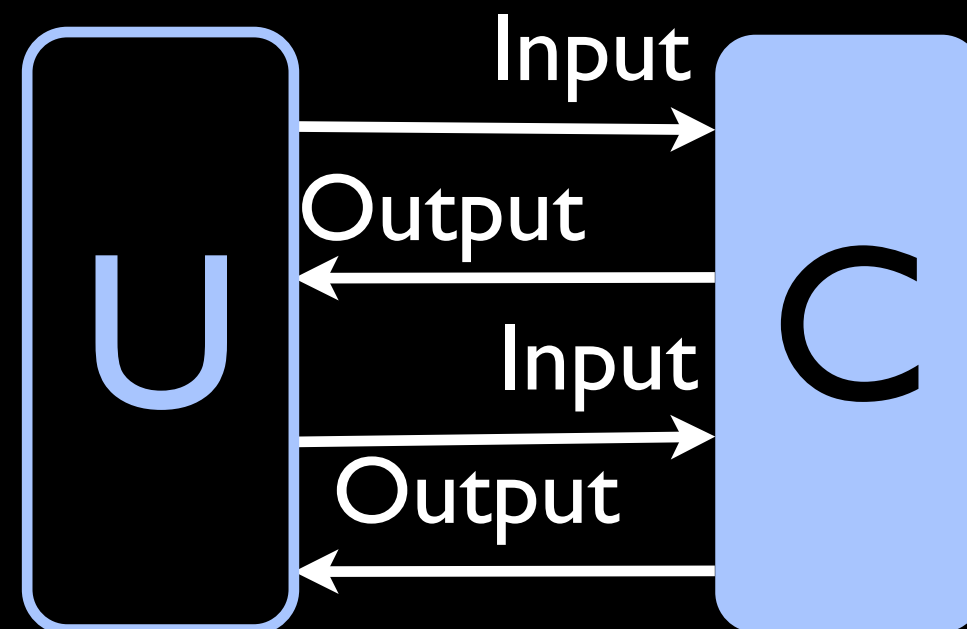
Batch Model

No time,
no dialogue

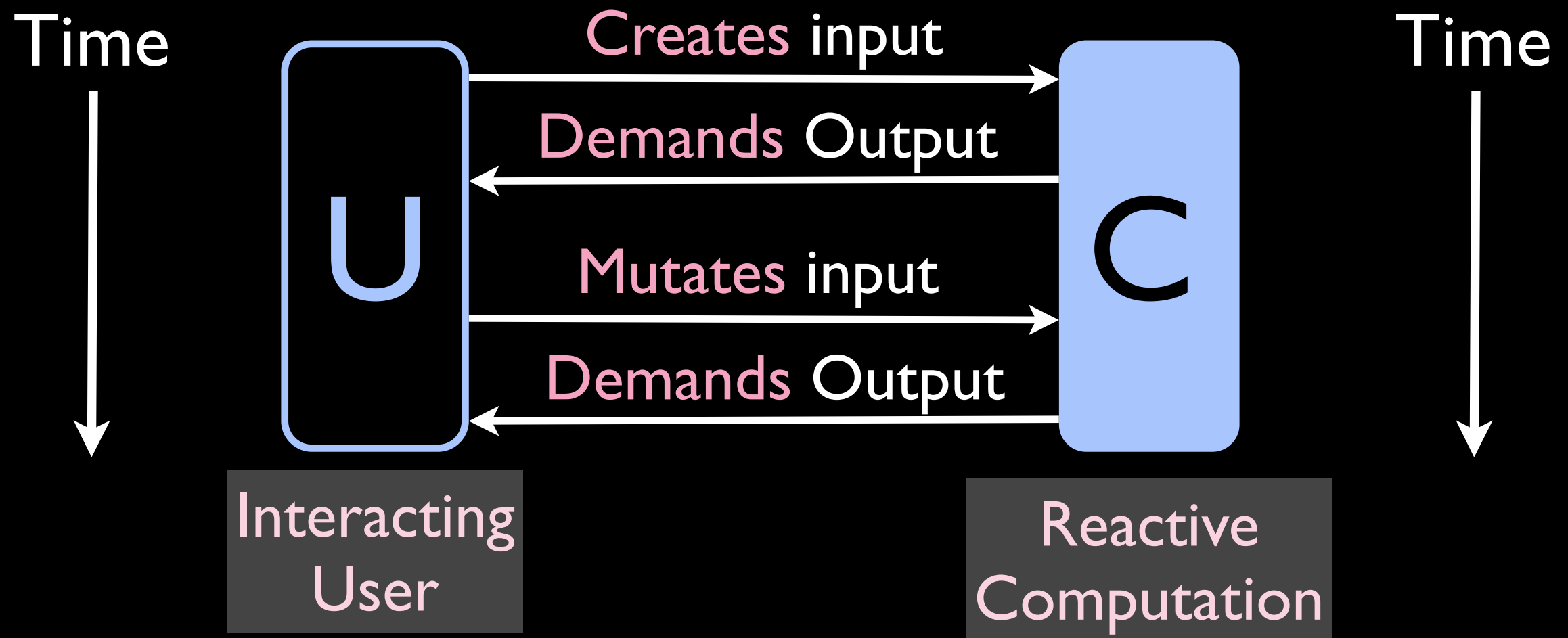


Interactive Model

Dialogue in time



Interaction is a Dialogue

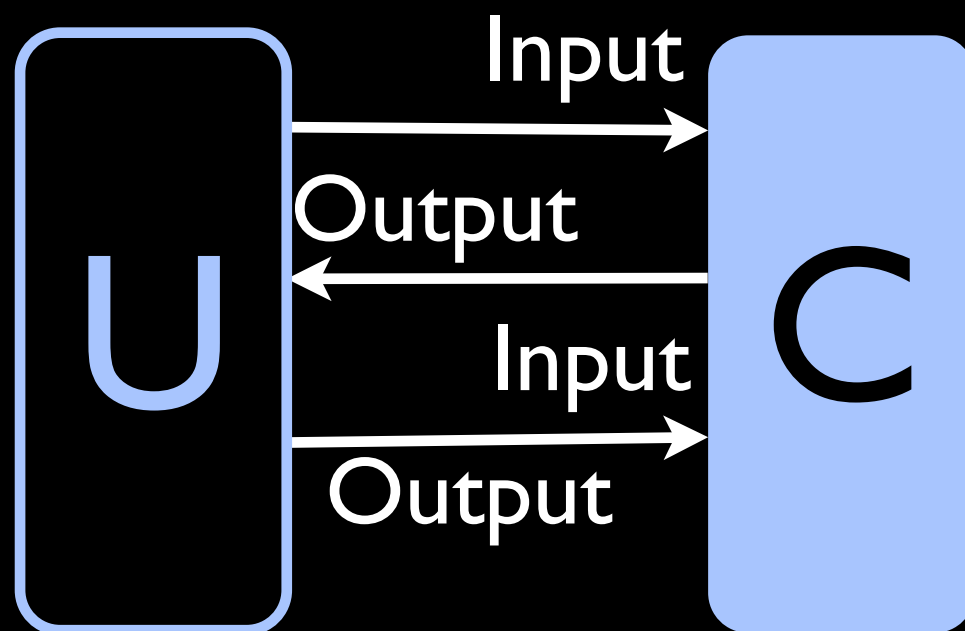


Elements of Interactive Dialogue

- User and system interact across time
- User mutates / changes input structure
- User demands / observes output structure
- The system maintains a correspondence between input and output structures
- I/O correspondence is computational

Incremental Interactive Computation

Incremental Interactive Computation



Claim: Interesting
interactive
systems consist of
incremental
computations

Example: Spreadsheets

Input
Structure

Cell Formulae

Incremental
Computation

Formula
evaluation

Output
Structure

Cell values

| | A | B | C | D |
|----|-----|--------------------------|-------|-------------|
| 1 | x | y | x^2 | $x \cdot y$ |
| 2 | 1,5 | 5,1 | 2,25 | 7,65 |
| 3 | 2,3 | 2,4 | 5,29 | 5,52 |
| 4 | 3,1 | 0,6 | 9,61 | 1,86 |
| 5 | 3,9 | -2,7 | 15,21 | -10,53 |
| 6 | 6,2 | -6 | 38,44 | -37,2 |
| 7 | 17 | -0,6 | 70,8 | -32,7 |
| 8 | | | | |
| 9 | | | | |
| 10 | a | $=(D7-A7*B7)/(C7-A7*A7)$ | | |
| 11 | b | | | |

Example: Word processing

Input
Structure

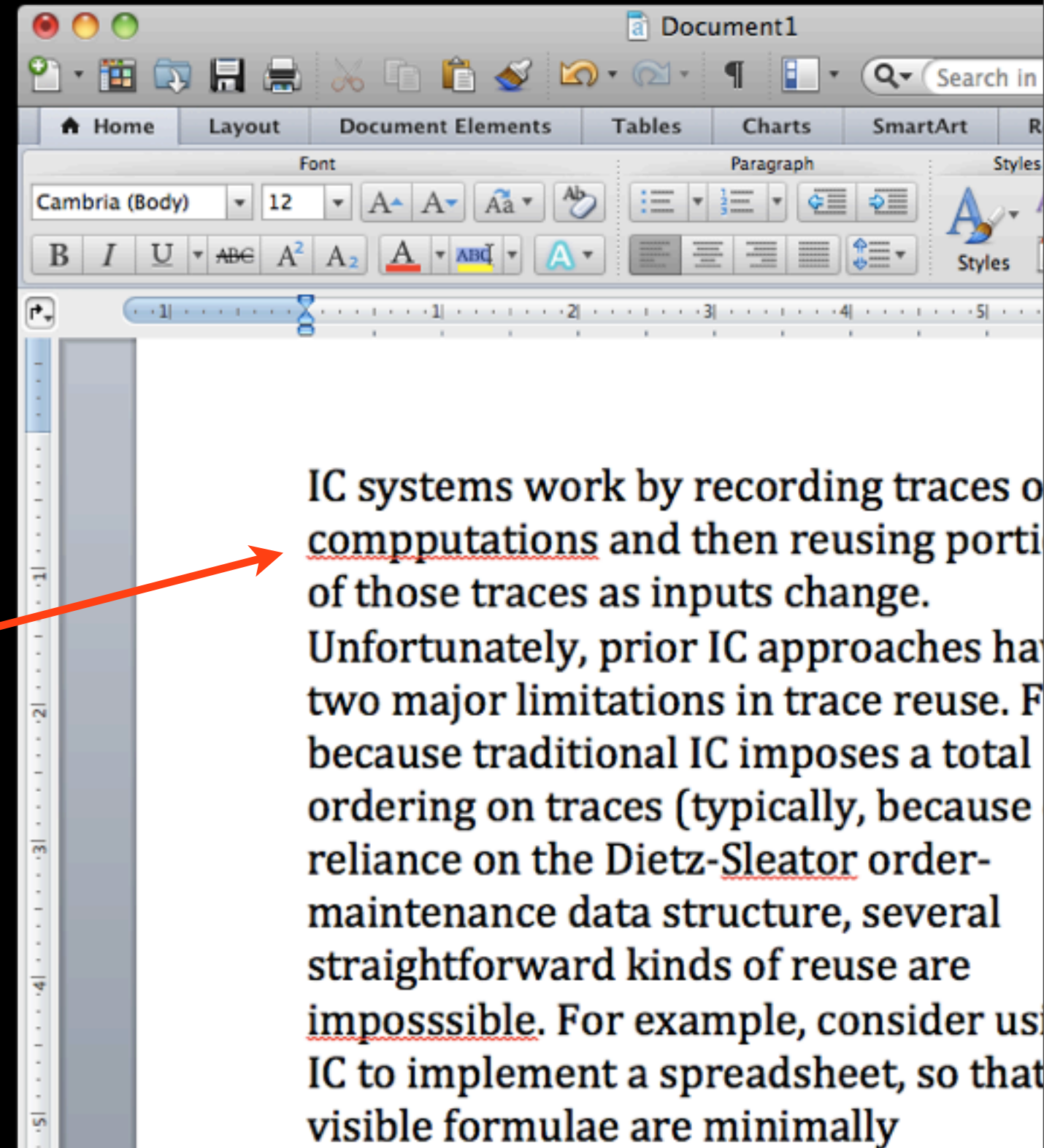
Document
Content

Incremental
Computation

Spell-check
each word

Output
Structure

Highlight
misspellings



Example: Programming

| | |
|-------------|--------------|
| Input | Text file |
| Computation | Parse tokens |
| Output | AST |

| | |
|-------------|-----------------|
| Input | AST |
| Computation | Lexical scoping |
| Output | Def/use edges |

| | |
|-------------|----------------|
| Input | AST + Def/use |
| Computation | Type inference |
| Output | Type errors |

Computing Incrementally

1. Input changes are gradual

Full re-computation is
often **redundant**

2. Output observation is limited

Full re-computation is
often **overly eager**

Computing Incrementally

1. Input changes are gradual

2. Output observation is limited

Example:
Spreadsheet

Cells change slowly

Example:
Word processing

Document and
dictionary both
change slowly

Example:
Programming

Program changes
slowly

Computing Incrementally

1. Input changes are gradual

2. Output observation is limited

Example:
Spreadsheet

One worksheet is active, others hidden

Example:
Word processing

Viewport shows one or two pages

Example:
Programming

Viewport shows one file, module or function

Adapton

Programming Abstractions
for Incremental Interaction

Adapton Programming Abstractions

- **Mutable references:**

- Hold changing input structure

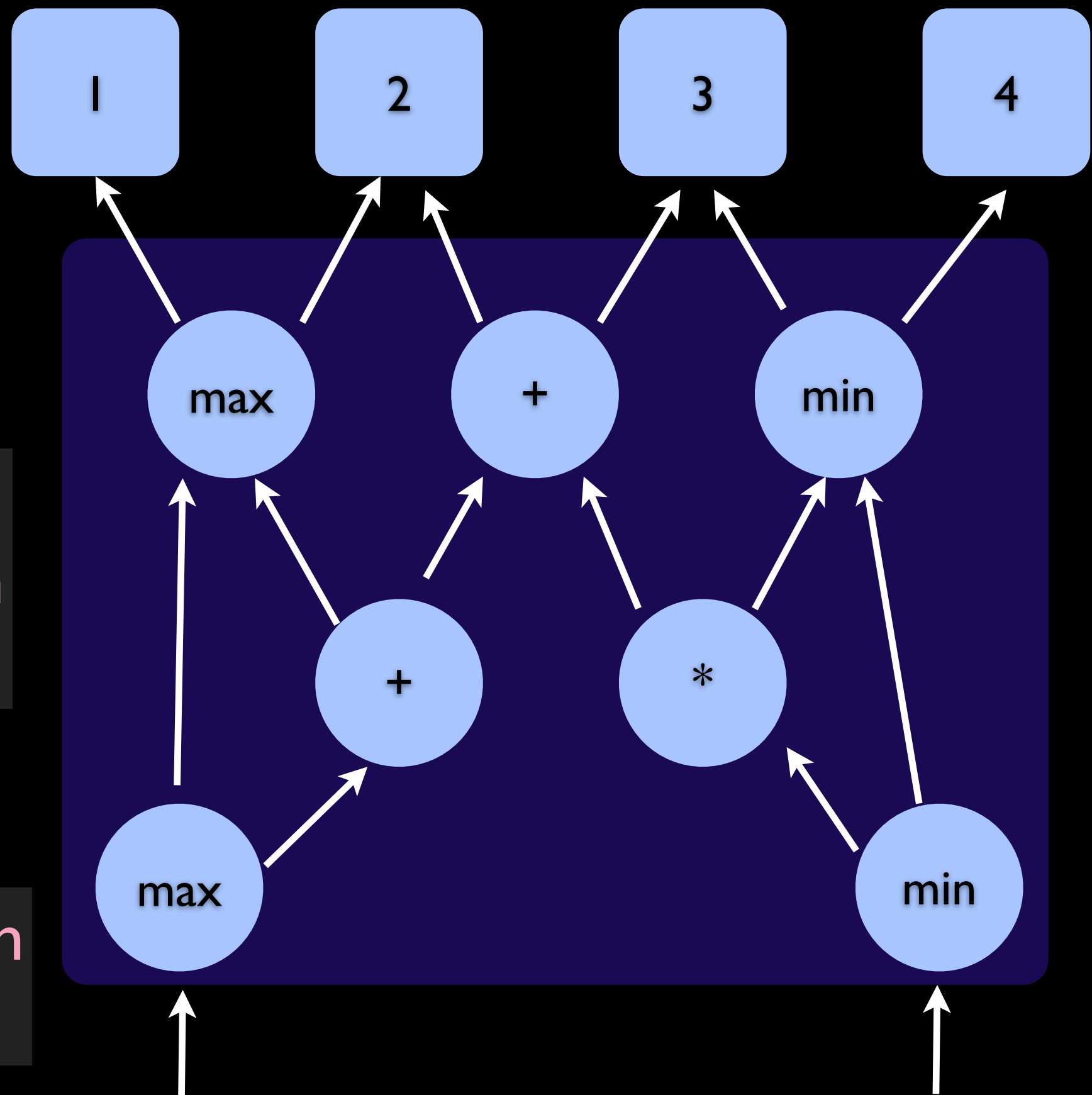
- **Lazy thunks:**

- Demand-driven computations
- Output structure

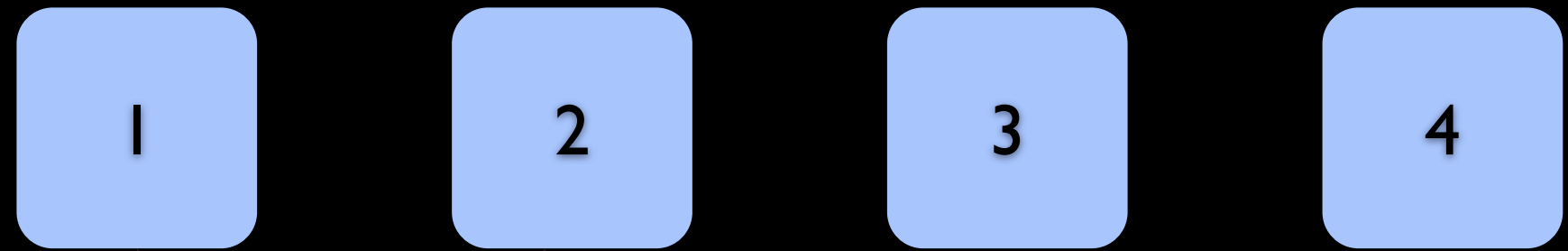
Mutable
references

Incremental
Computation
(thunks)

Demand-driven
Outputs

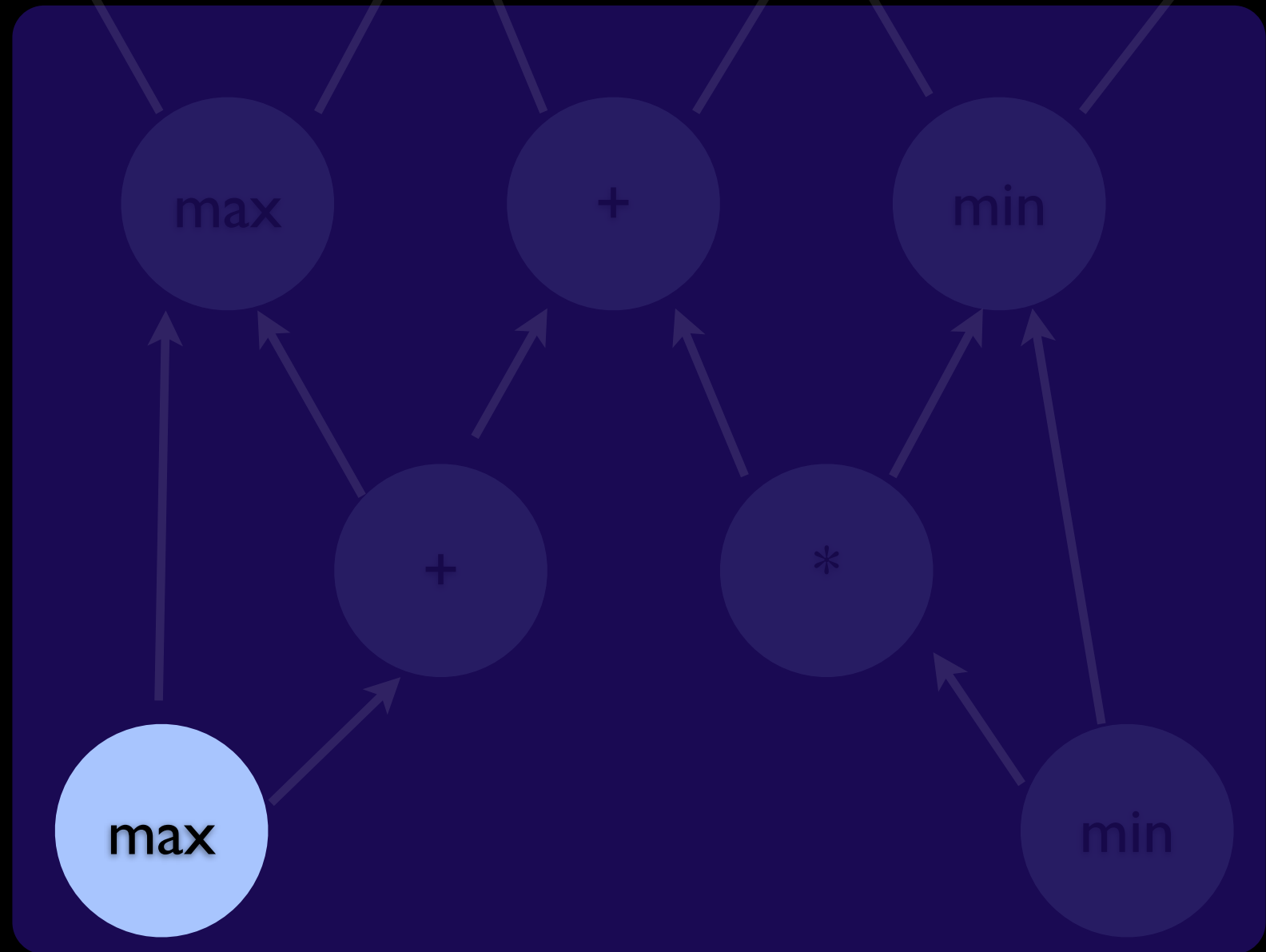


Mutable
references



Incremental
Computation
(thunks)

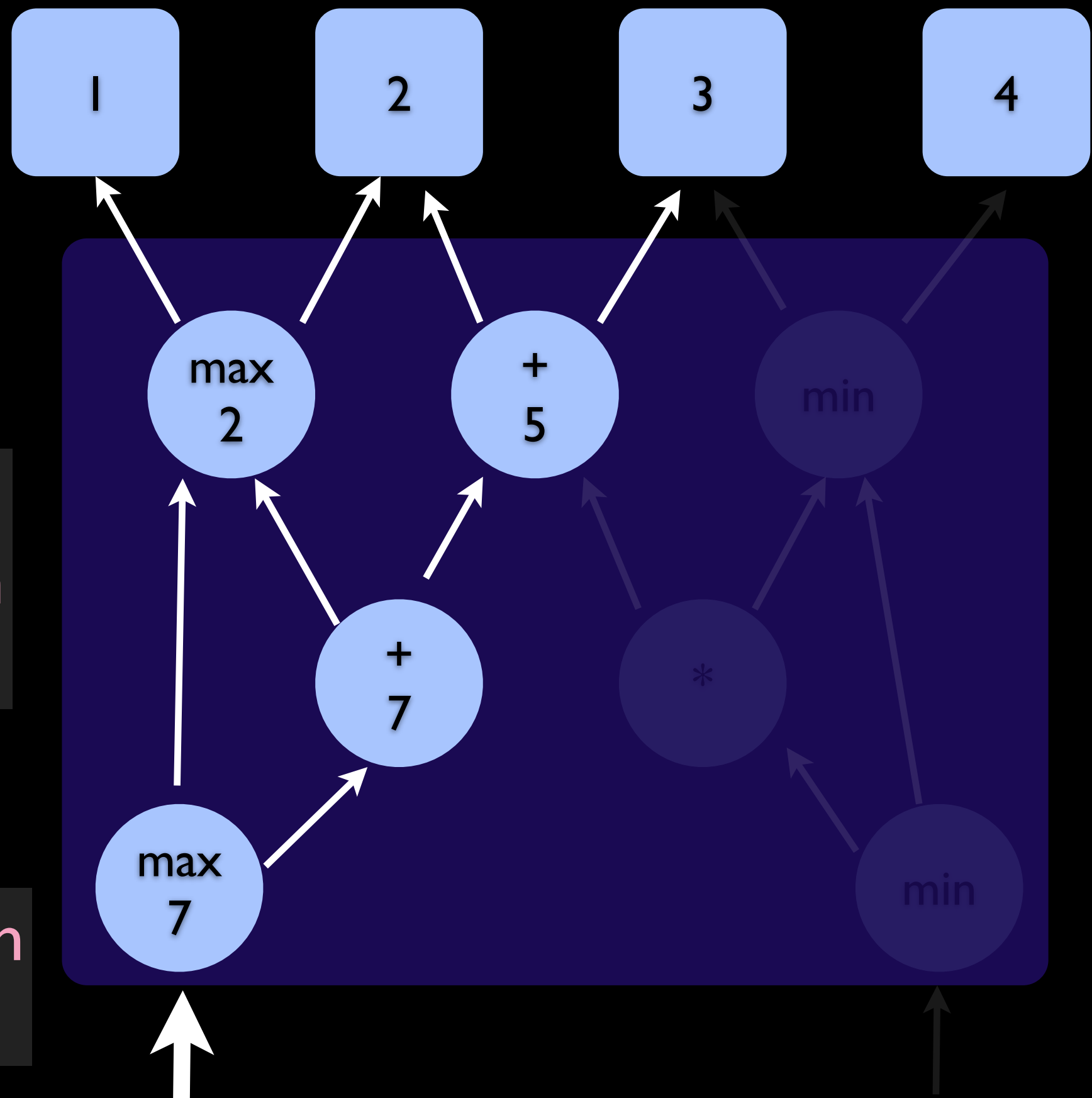
Demand-driven
Outputs



Mutable
references

Incremental
Computation
(thunks)

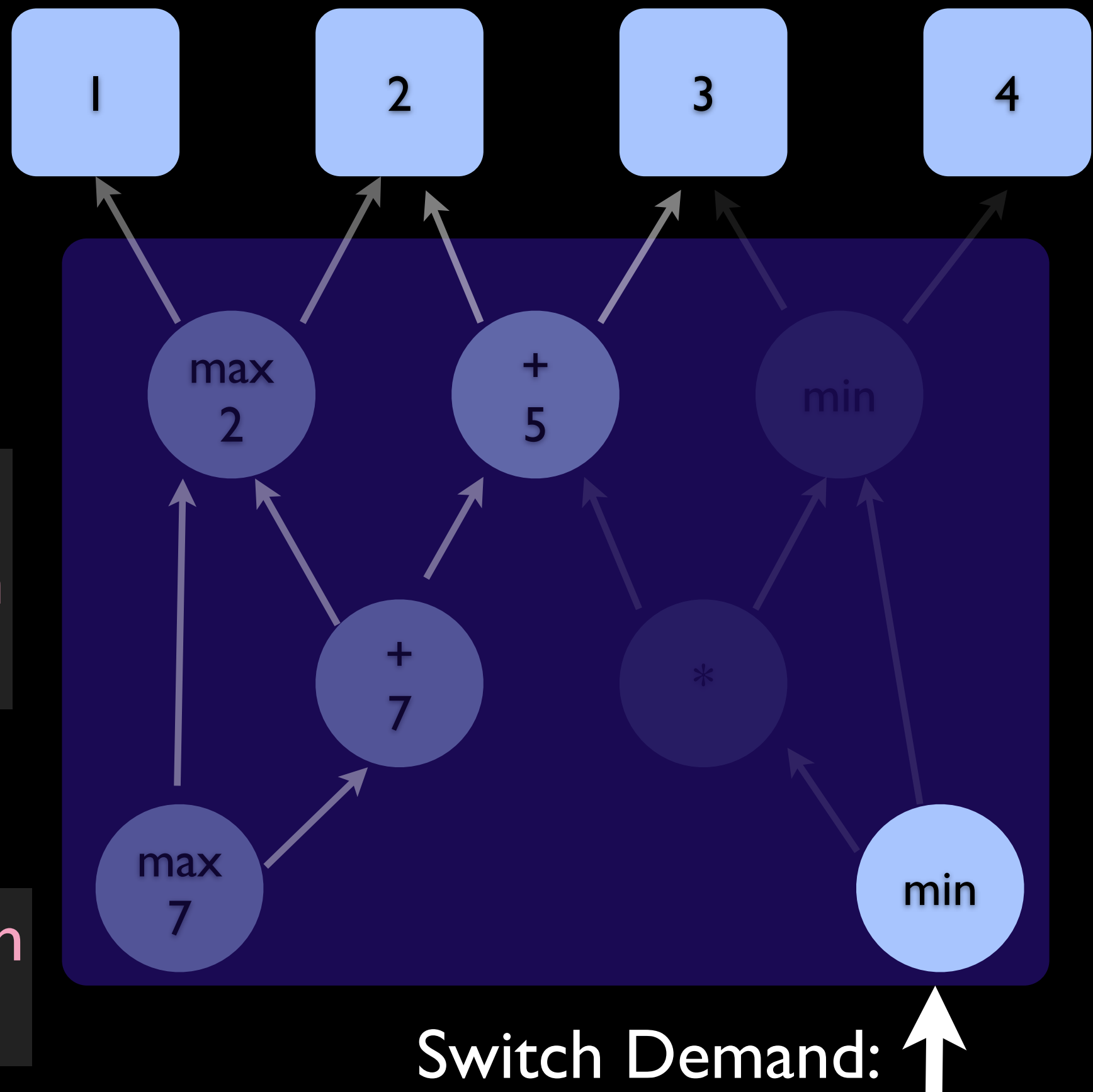
Demand-driven
Outputs



Mutable
references

Incremental
Computation
(thunks)

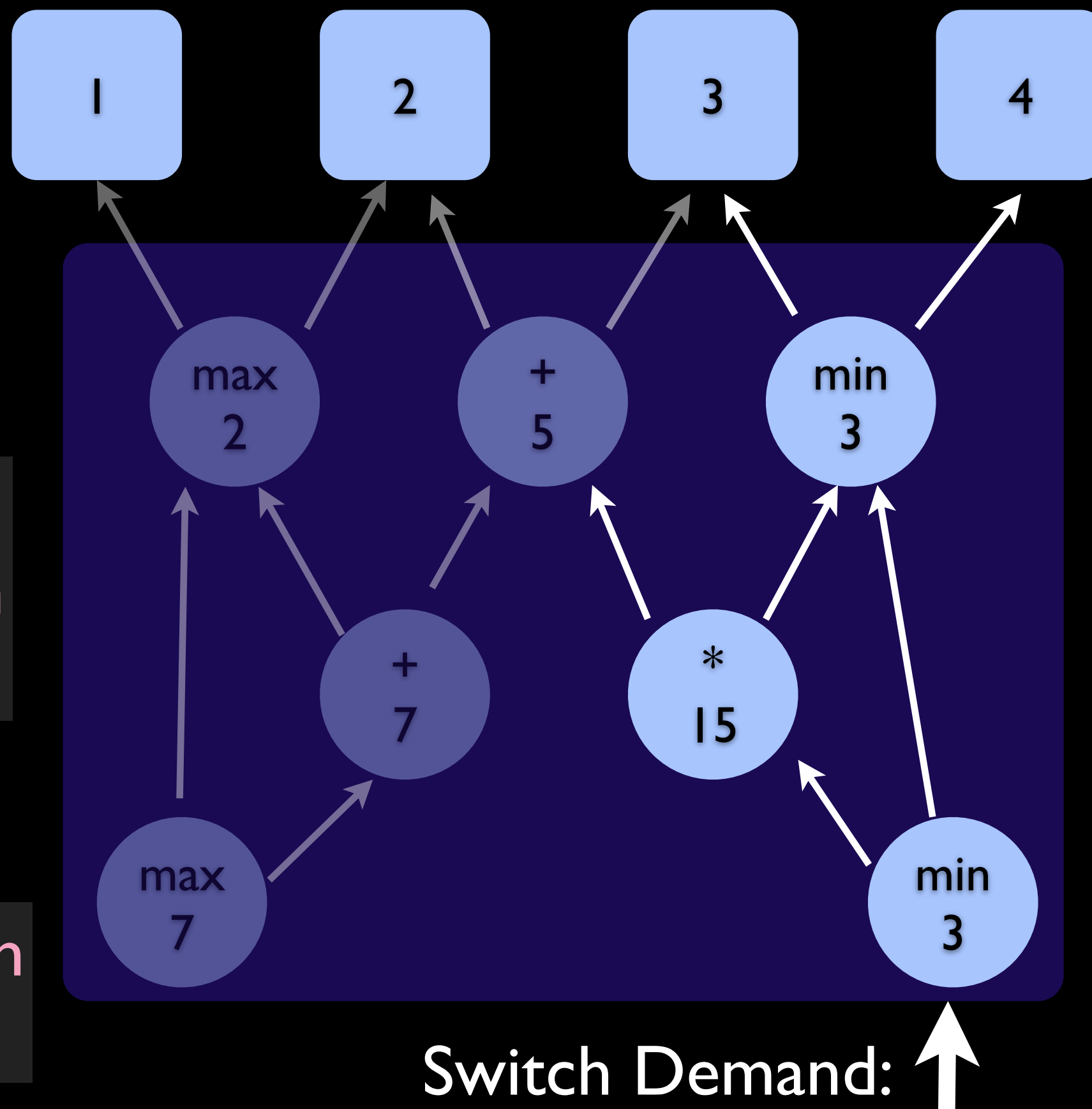
Demand-driven
Outputs



Mutable
references

Incremental
Computation
(thunks)

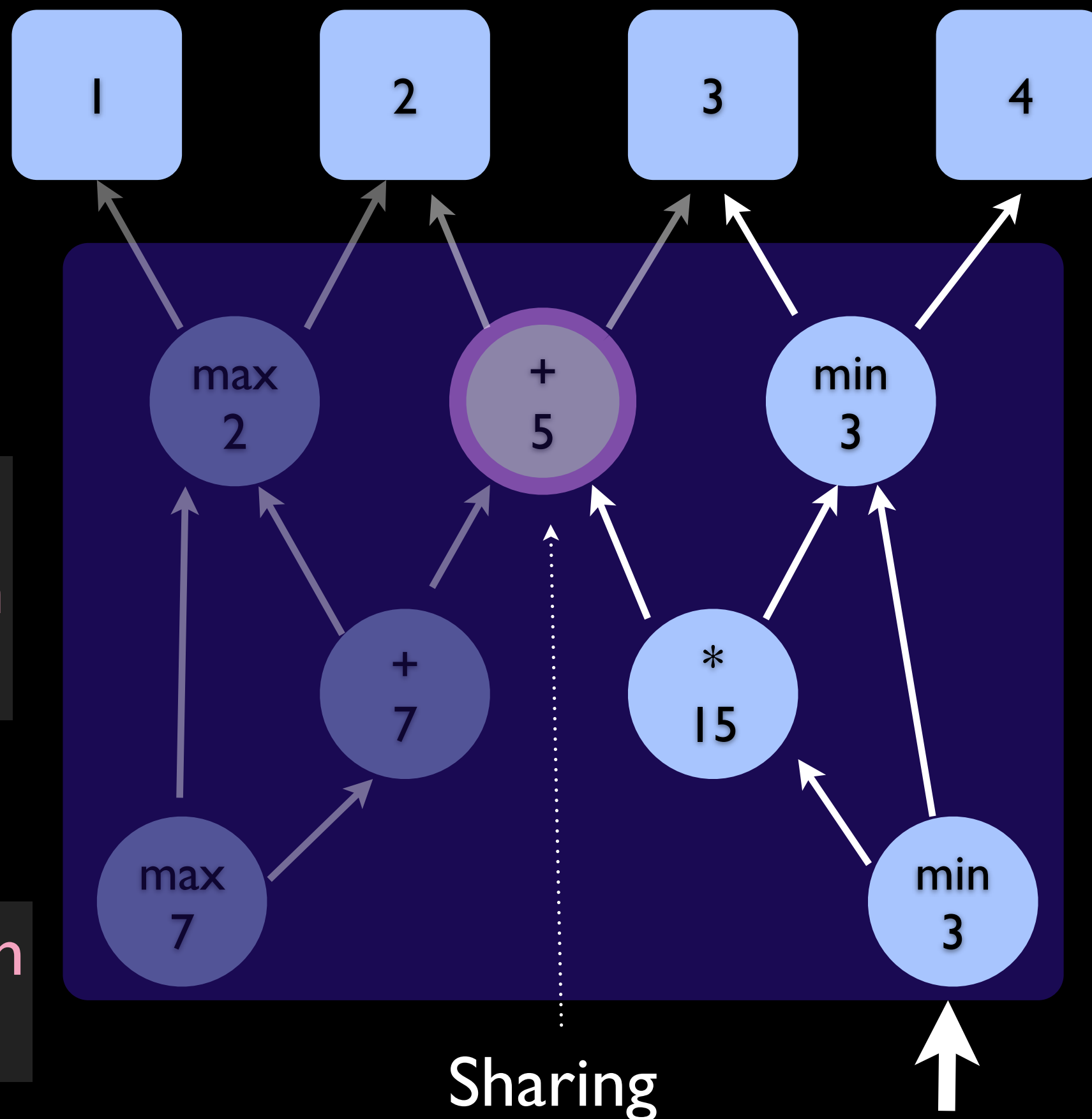
Demand-driven
Outputs



Mutable
references

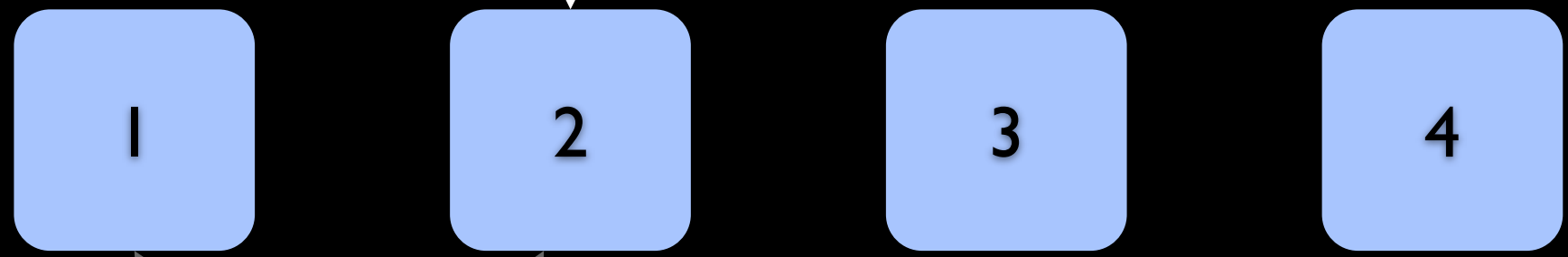
Incremental
Computation
(thunks)

Demand-driven
Outputs

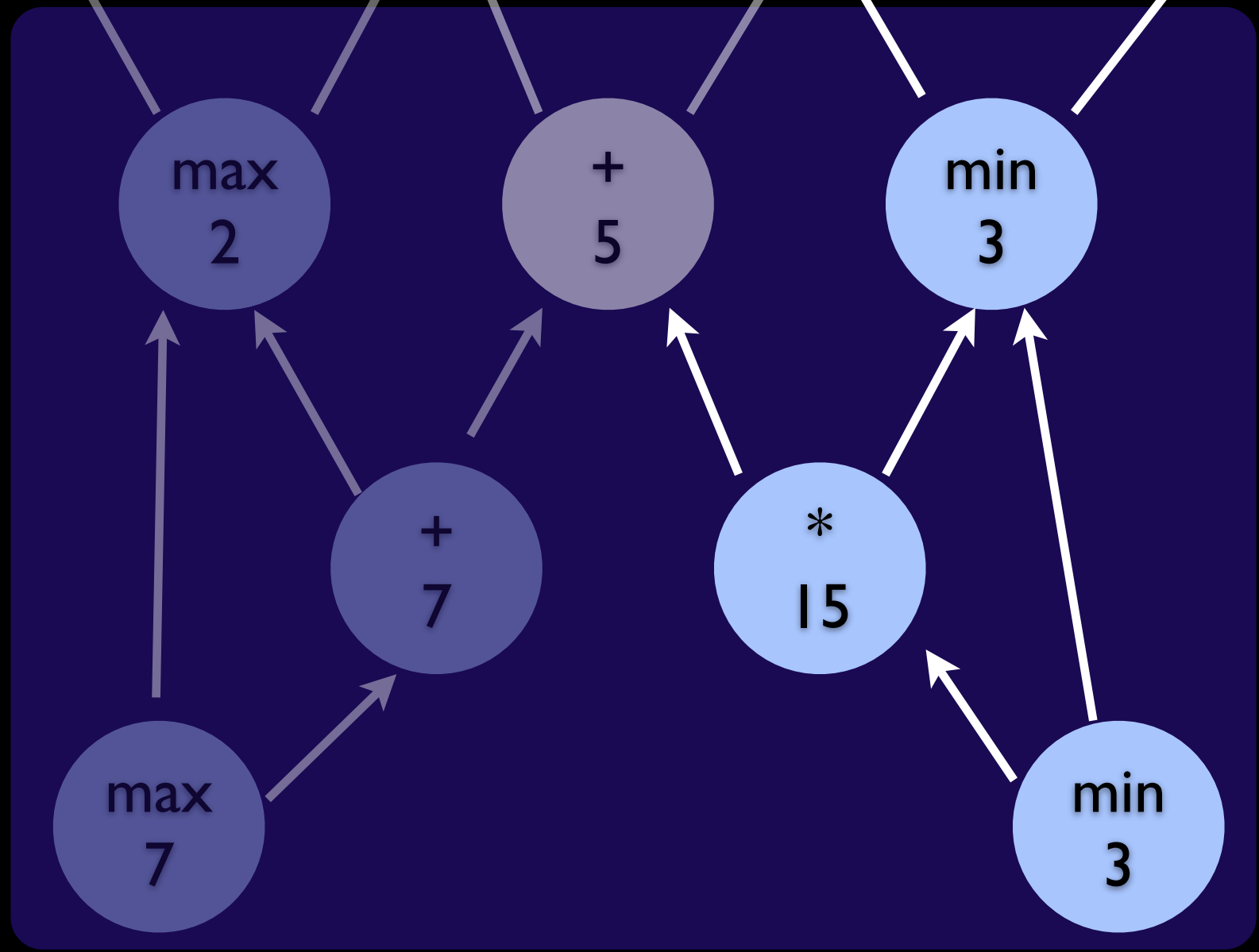


Input Change: ↓

Mutable
references



Incremental
Computation
(thunks)



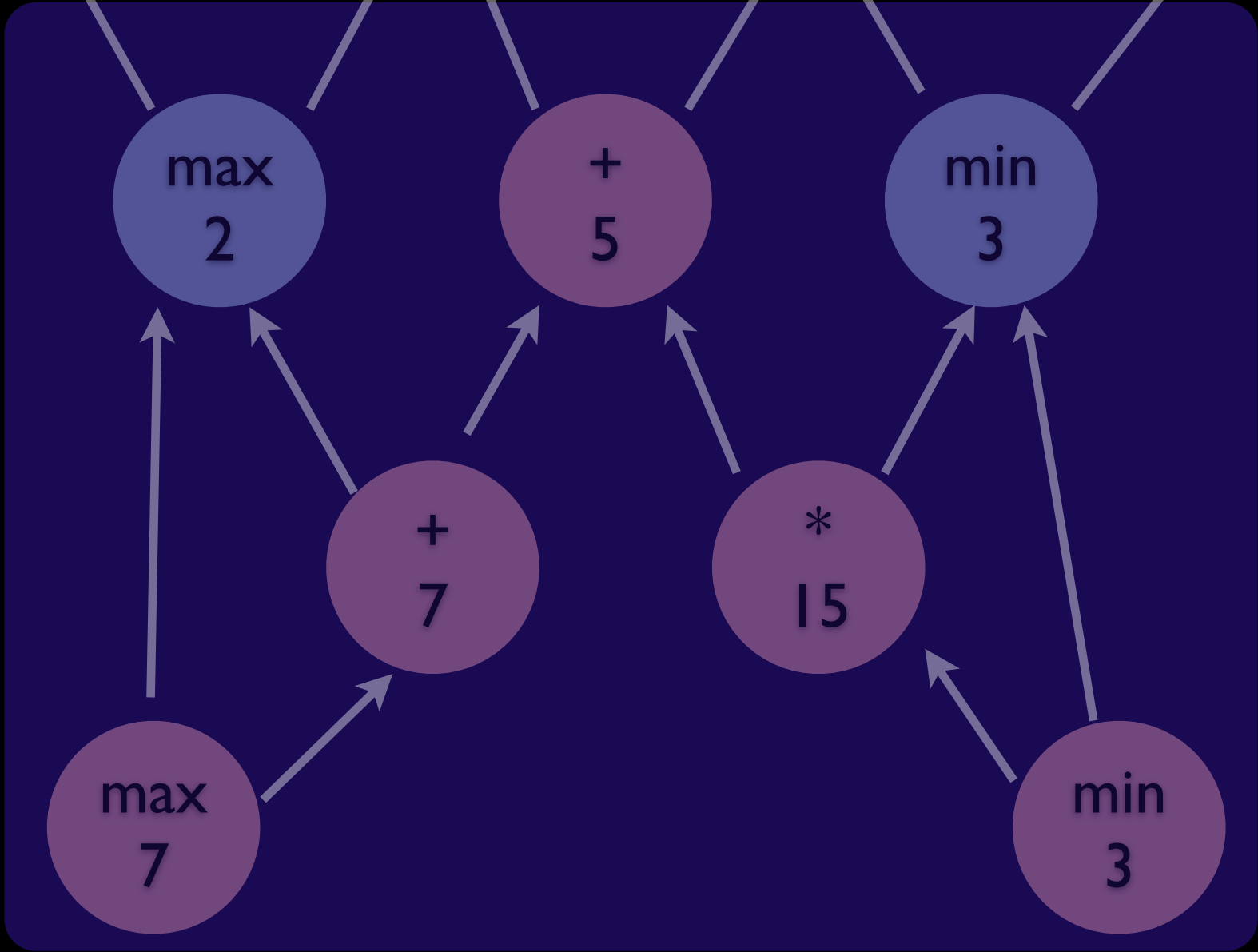
Demand-driven
Outputs



Mutable references

Incremental Computation (thunks)

Demand-driven Outputs

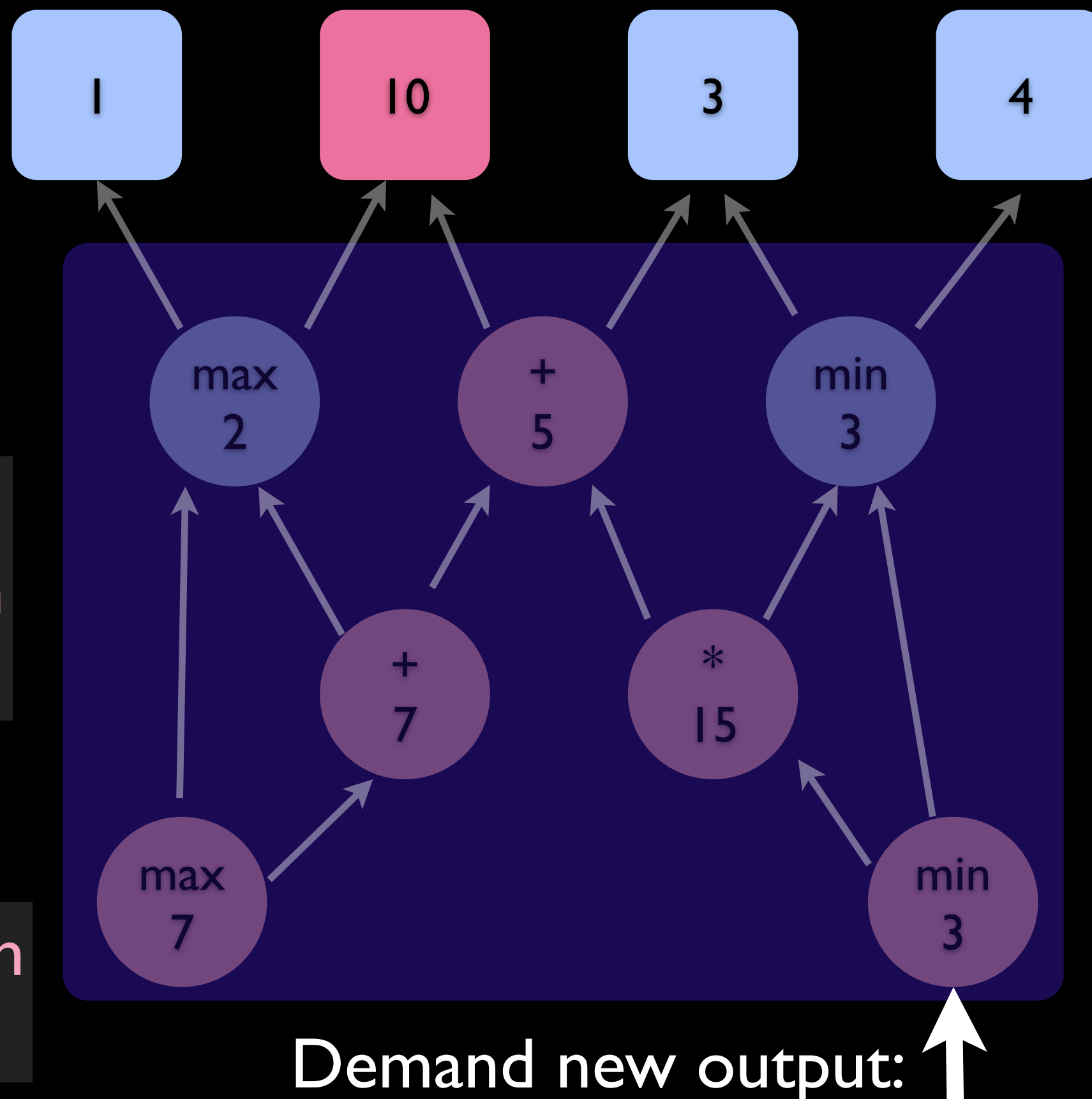


Some previous results are affected

Mutable
references

Incremental
Computation
(thunks)

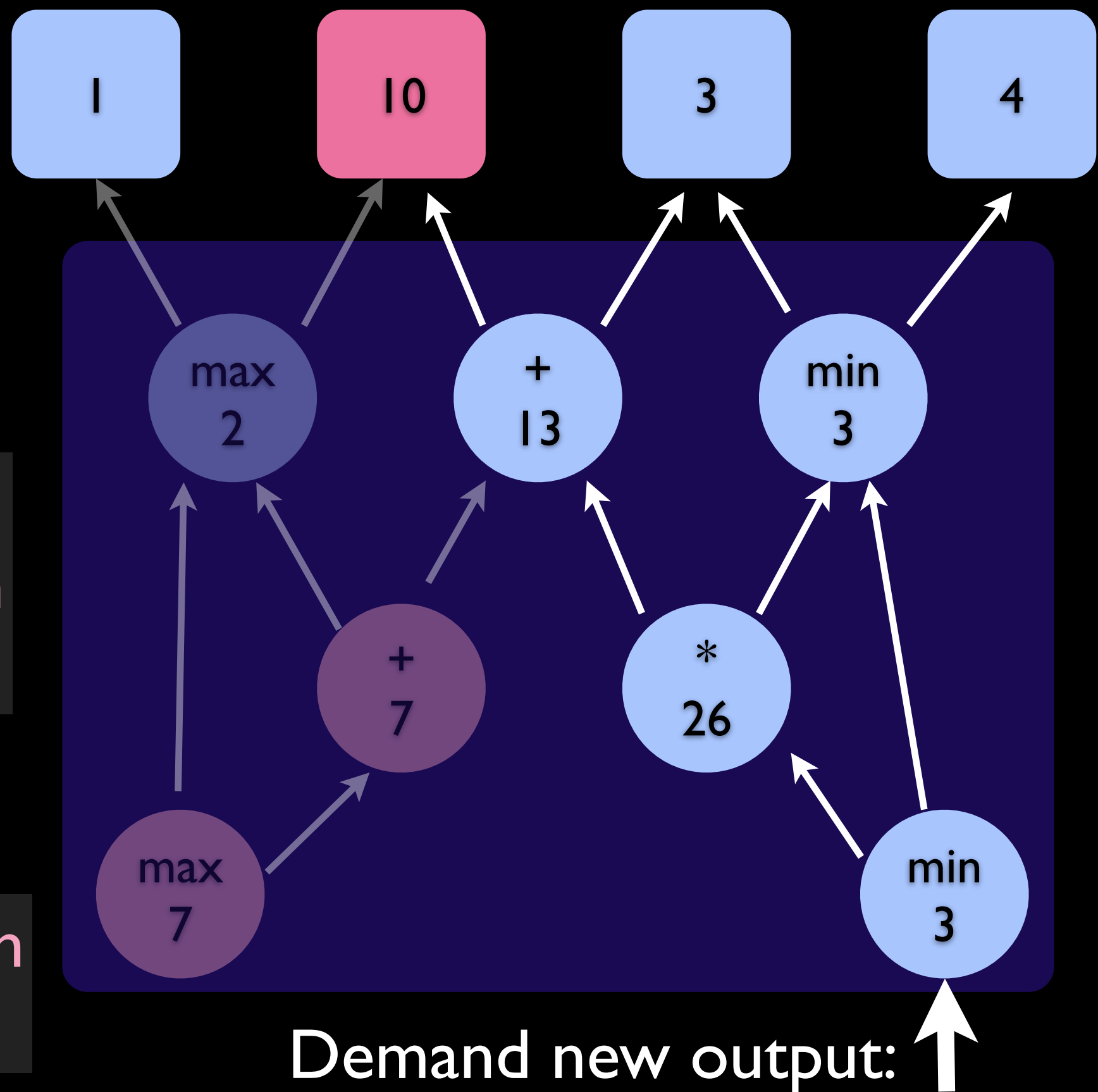
Demand-driven
Outputs



Mutable
references

Incremental
Computation
(thunks)

Demand-driven
Outputs



Lazy Structures

- **Spreadsheet example:**
 - Each thunk returns a **single number**
- **Lazy Lists:**
 - Each thunk returns **`Nil** or **`Cons**
 - **`Cons** holds **head value** and a **thunk tail**
- Laziness can be applied to **trees, graphs,** and essentially any other data structure
- **Inputs:** Special thunks are **mutable**

Background: Lazy Lists

```
type 'a thunk = unit -> 'a
```

```
let force : 'a thunk -> 'a  
  = fun t -> t ()
```

Apply unit argument



```
type 'a lazylist = [  
  | `Nil  
  | `Cons of 'a * ('a lazylist thunk)  
]
```

```
let rec from_list l =  
  match l with  
  | [] -> `Nil  
  | h::t -> `Cons(h, fun() -> from_list t)
```

```
type 'a lzlzlist = [  
  | `Nil  
  | `Cons of 'a * ('a lzlzlist thunk)  
]
```

```
let rec merge l1 l2 = function  
  | l1, `Nil -> l1  
  | `Nil, l2 -> l2  
  
  | `Cons(h1,t1), `Cons(h2,t2) ->  
  
    if h1 <= h2 then  
      `Cons(h1, fun()->merge (force t1) l2)  
  
    else  
      `Cons(h2, fun()->merge l1 (force t2))
```

Mergesort Example

| | |
|------------|----------------------------------|
| Input | [3,5,8,2,1,7] |
| Singletons | [[3], [5], [8], [2], [1], [7]] |
| Merge #1 | [[3,5], [2,8], [1,7]] |
| Merge #2 | [[3,5], [1,2,7,8]] |
| Merge #3 | [[3,5], [1,2,7,8]] |
| Merge #4 | [[1, 2, 3, 5, 7, 8]] |
| Flatten | [1, 2, 3, 5, 7, 8] |

Course Project

Project: Interactive Program Analysis

- Assume: Interactive “Structure Editor”
 - Programmer manipulates AST directly
- Learn: Adaption IC framework
- Build: Incremental Program Analysis
 - Example: Use/def information
 - Example: Type Inference
 - Example: Control-flow analysis

Background: Structure Editors

- Philosophical claims:
 - Programs consist of **rich structure**
 - Rich interaction **exposes this structure**

- Example prototypes:
 - **Haskell** -- <http://www.youtube.com/watch?v=v2ypDcUM06U>
 - **Citris** -- <http://www.youtube.com/watch?v=47UcOspbZ2k>
 - **TouchDevelop** -- <http://www.youtube.com/watch?v=a6GRg2glKpc>