Tonic

Task Oriented Notation Inferred from Code

Radboud University Nijmegen

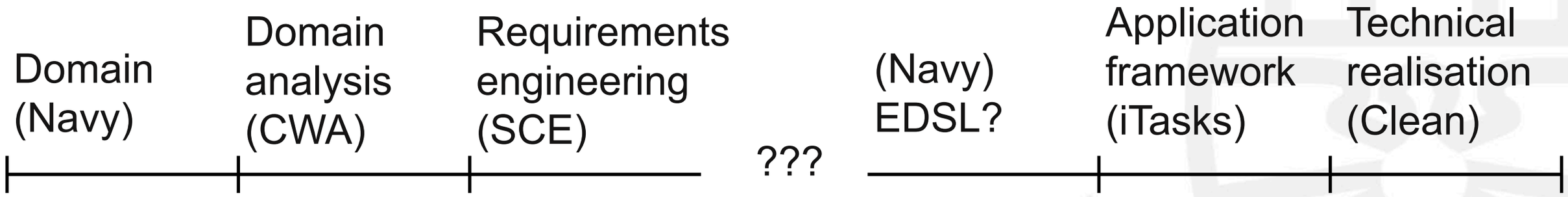
Context

- Royal Netherlands Navy wants to operate their future vessels with significantly less personnel
- Significant automatisation of on-board tasks required
- TNO is currently researching how this can be achieved

Challenge

- Navy is a complex socio-technical domain
- Automation requires sophisticated socio-technical solutions
- TNO recently started using iTasks for rapid prototyping of these solutions
  - Calamity response application for fire, leakage and resource conflict scenarios
  - iTasks: a framework in Clean implementing the Task-Oriented Programming paradigm

Bridging two extremes



CWA: Cognitive Work Analysis

SCE: Situated Cognitive Engineering

Visual/graphical programming

- Visual programming has been suggested as a way to bridge the gap between programmers and non-programmers
- Previous work in functional languages:
  - Visual Haskell (Reekie, 1994)
  - NiMo (Clerici and Zoltan, 2004)
  - GiN (Henrix, Plasmeijer and Achten, 2012)
  - Marama (Groenouwe, 2013)
- Previous work outside of FP:
  - Process charts (Gilbreth, 1921) and derivatives (flowcharts etc.)
  - Petri nets (Petri, 1939)
  - Object-modeling technique (Rumbaugh et al, 1991) and derivatives (UML etc.)
  - Business Process Model and Notation (BPMI, 2004)

A common idea

Provide a graphical syntax (often combined with some textual syntax) so non-programmers can write (simple) programs as well

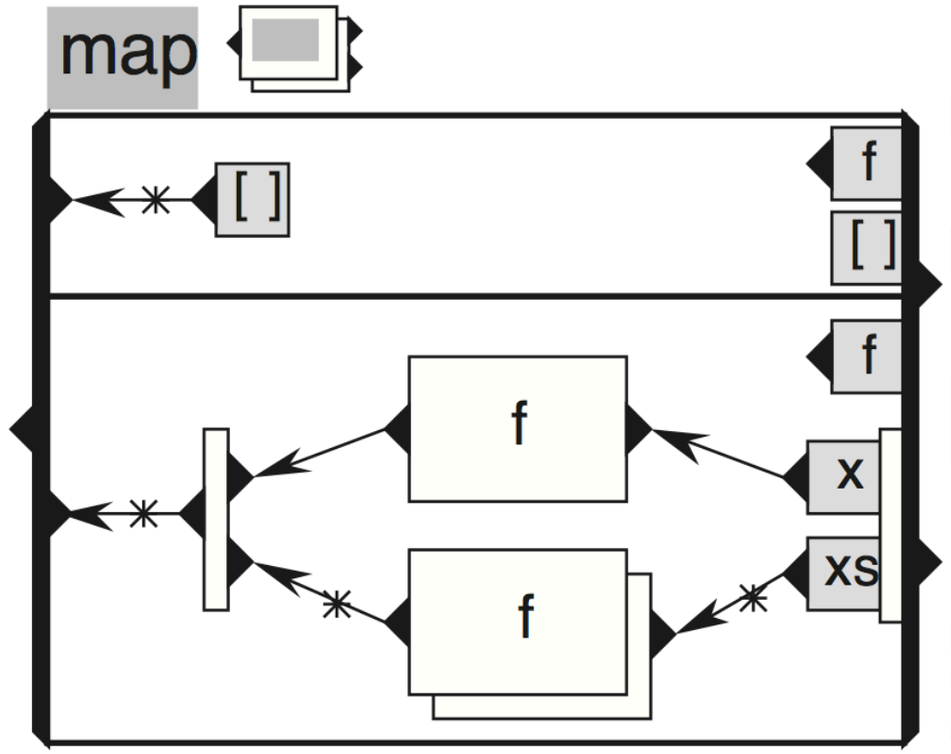
A common flaw

Programming is hard; a graphical syntax is not going to change that for the better

Nor is a visual syntax going to instantly teach a non-programmer how to write complex software

“A picture is worth a thousand words”

- Popular saying still holds: our spatial memory is strongly developed
- But programmers should still write code
  - (Until something truly better comes along)
- Why not visualize (parts of) code to provide insight for domain experts?
  - Visual Haskell does this as well to some extent:



Why?

- Communication with non-programmers during development
- Automatic program documentation
- Provide insight in running (sociotechnical) processes
  - Identify bottlenecks and shortcomings
  - Discover better ways of cooperation previously unconsidered
- Debugging technical and non-technical problems

Tonic

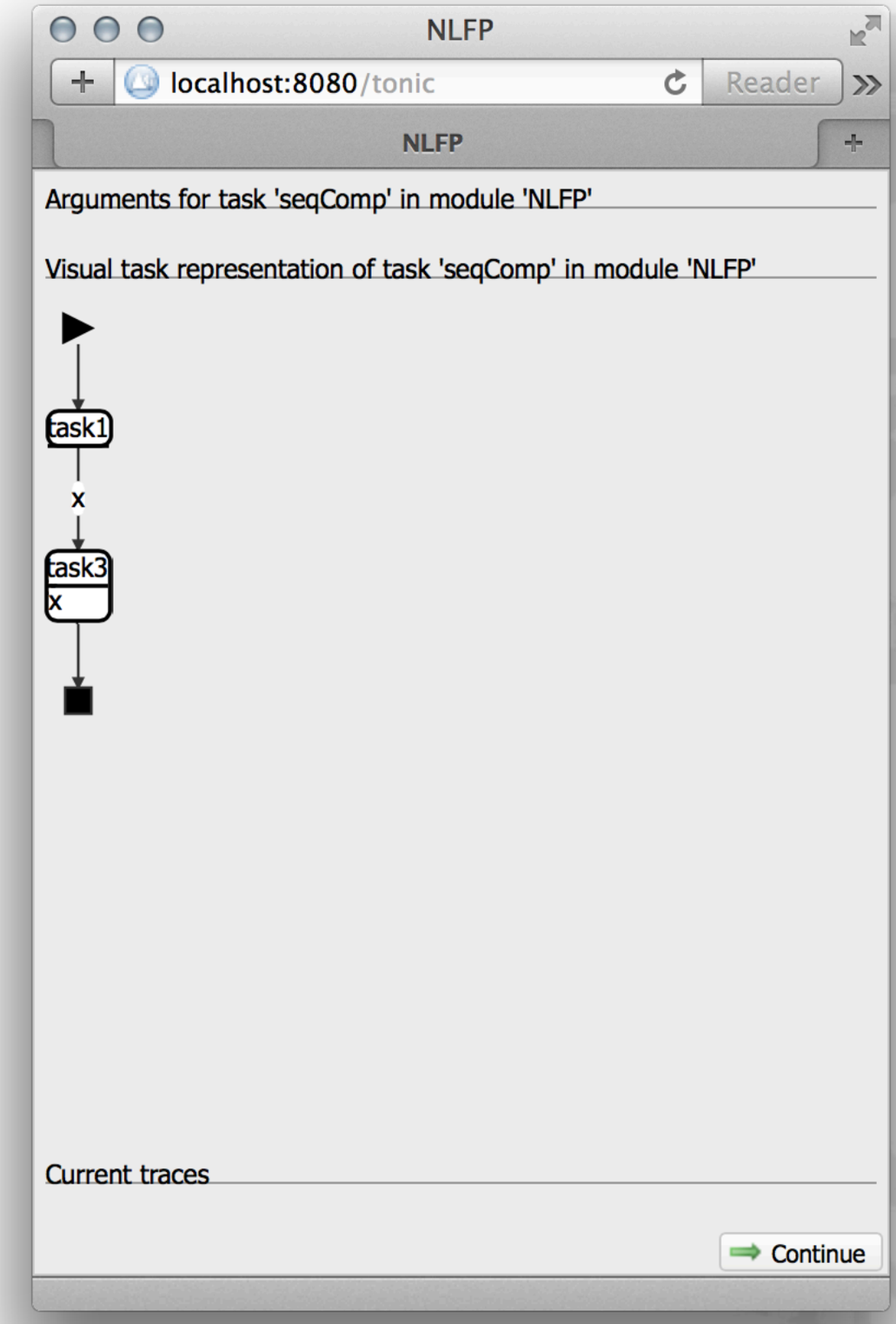
- Graphically represent iTasks programs on the Task level
  - Task is monad-ish
  - Don’t show every program detail; it’s not interesting for non-programmers
- Construct a static program representation compile-time
  - Implemented in the Clean compiler
- Overlay runtime information while the program is running
  - Annotate the program compile-time with traces to provide runtime information
  - Partly implemented as an iTasks library

How to visualize a program?

- Visualizing a program is hard
- Visualization needs to meet various (sometimes conflicting) criteria
  - Easy to understand for non-programmers
  - Use as few graphical concepts as possible
  - Show as much as needed, but nothing more
- Allow ambiguity in notation?
  - Humans are very good at dealing with ambiguity

Sequential composition

```
task1 >>= \x -> task3 x
```



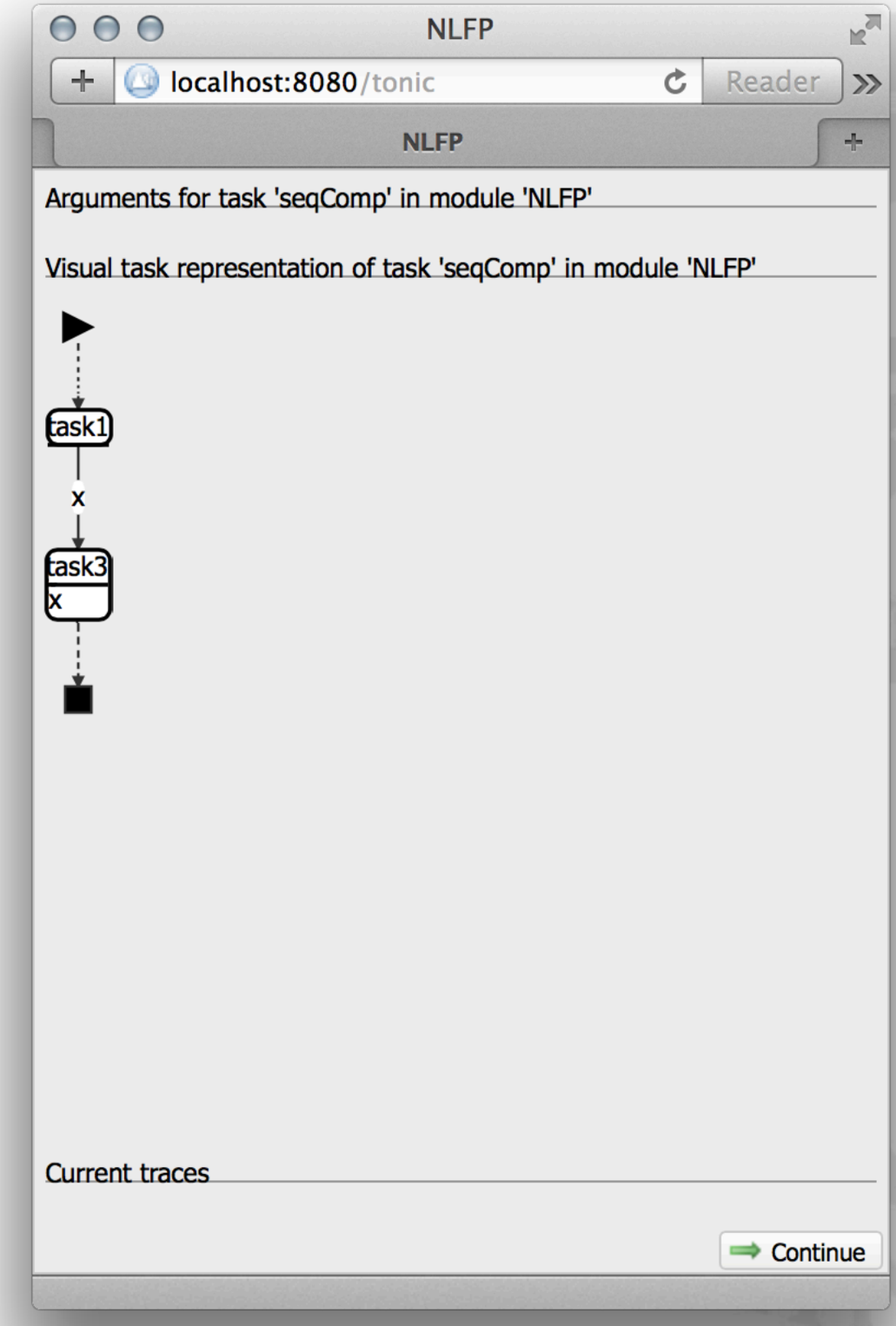
Note the current ambiguity in the meaning of the arrows

Sequential composition

```
task1 >>= \x -> task3 x
```

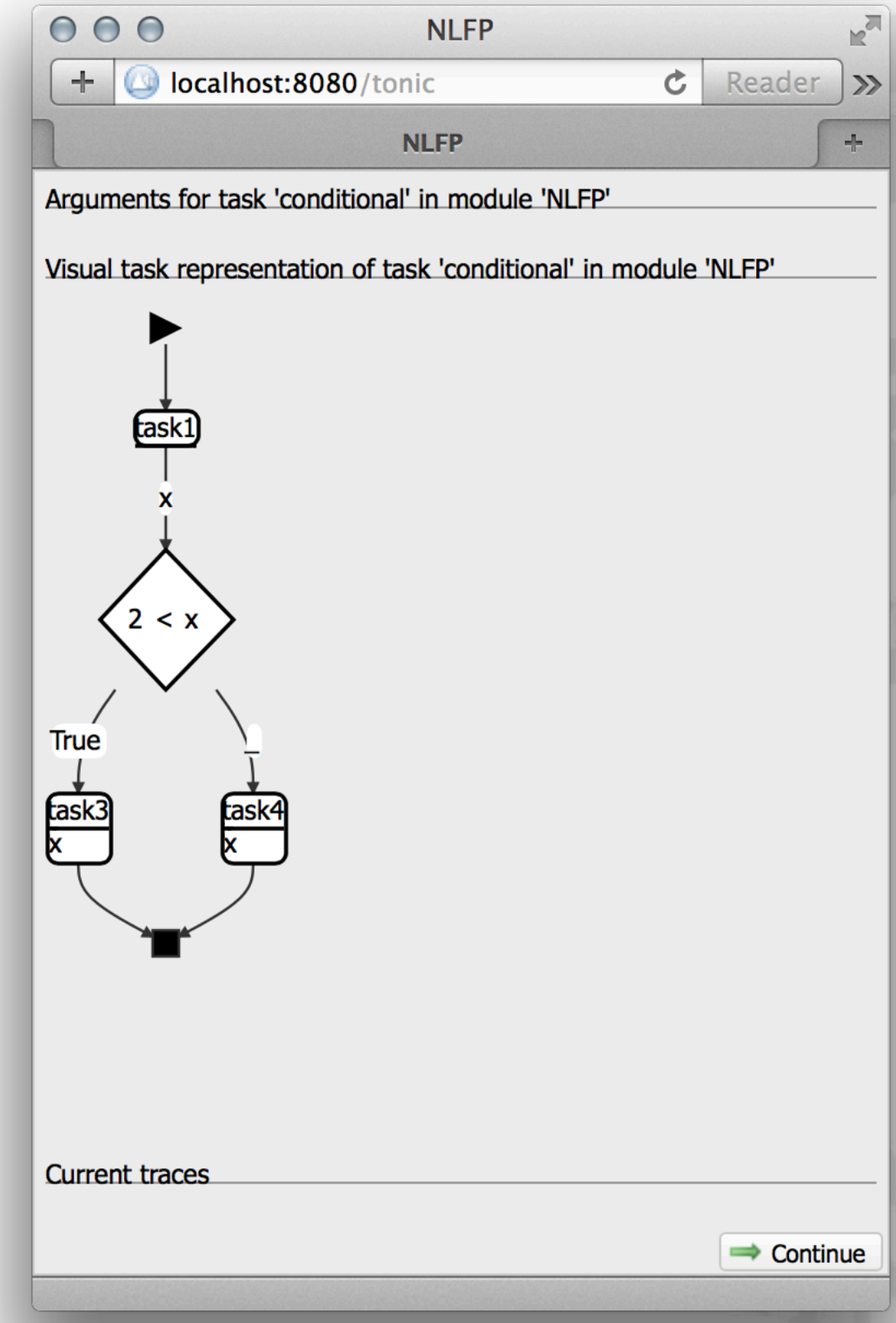
No more ambiguous arrows at the cost of additional notation.

Does this help a layman's understanding of the program? Or does it do the opposite?



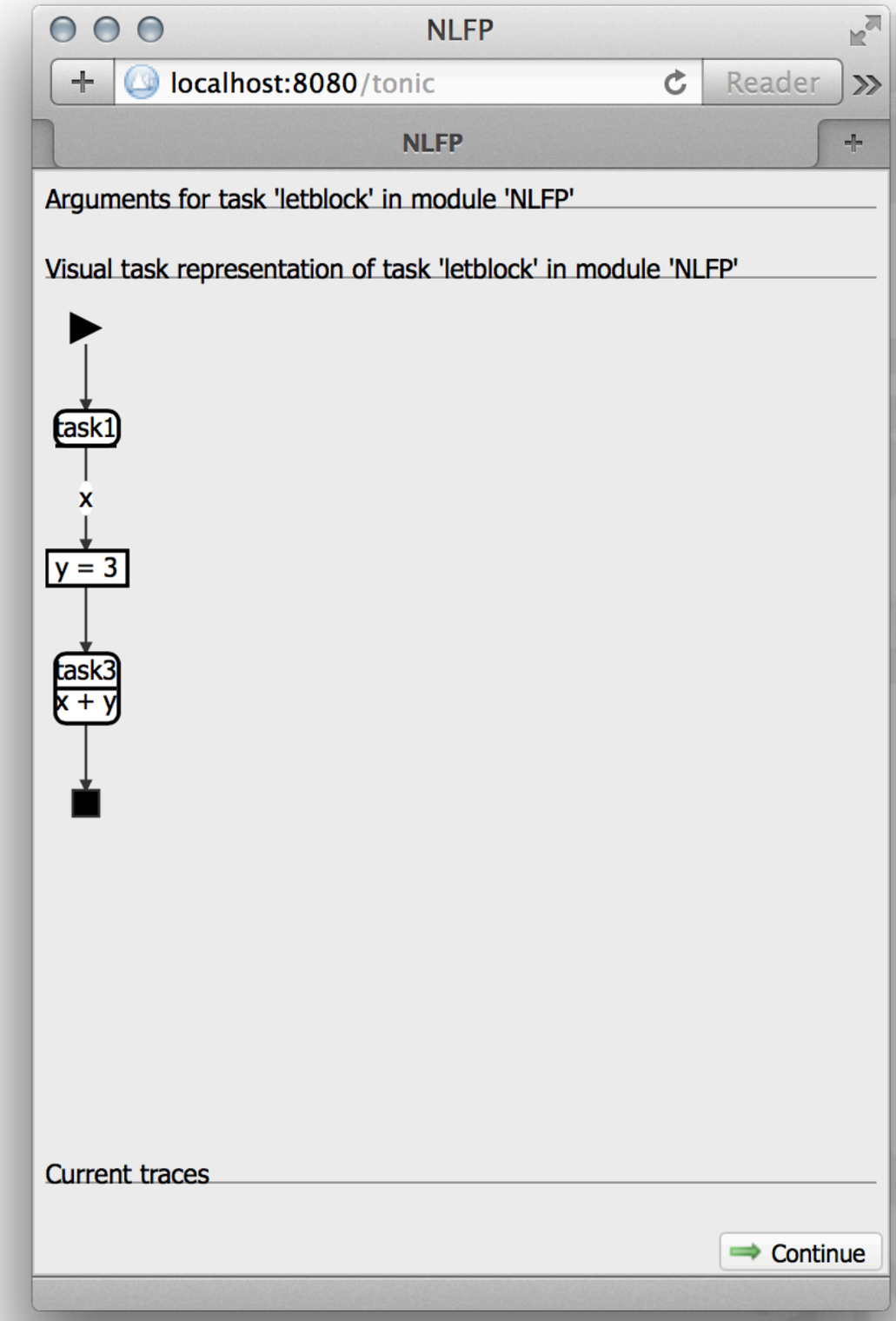
Conditional

```
task1 >>= \x ->
  if (x > 2)
     (task3 x)
     (task4 x)
```



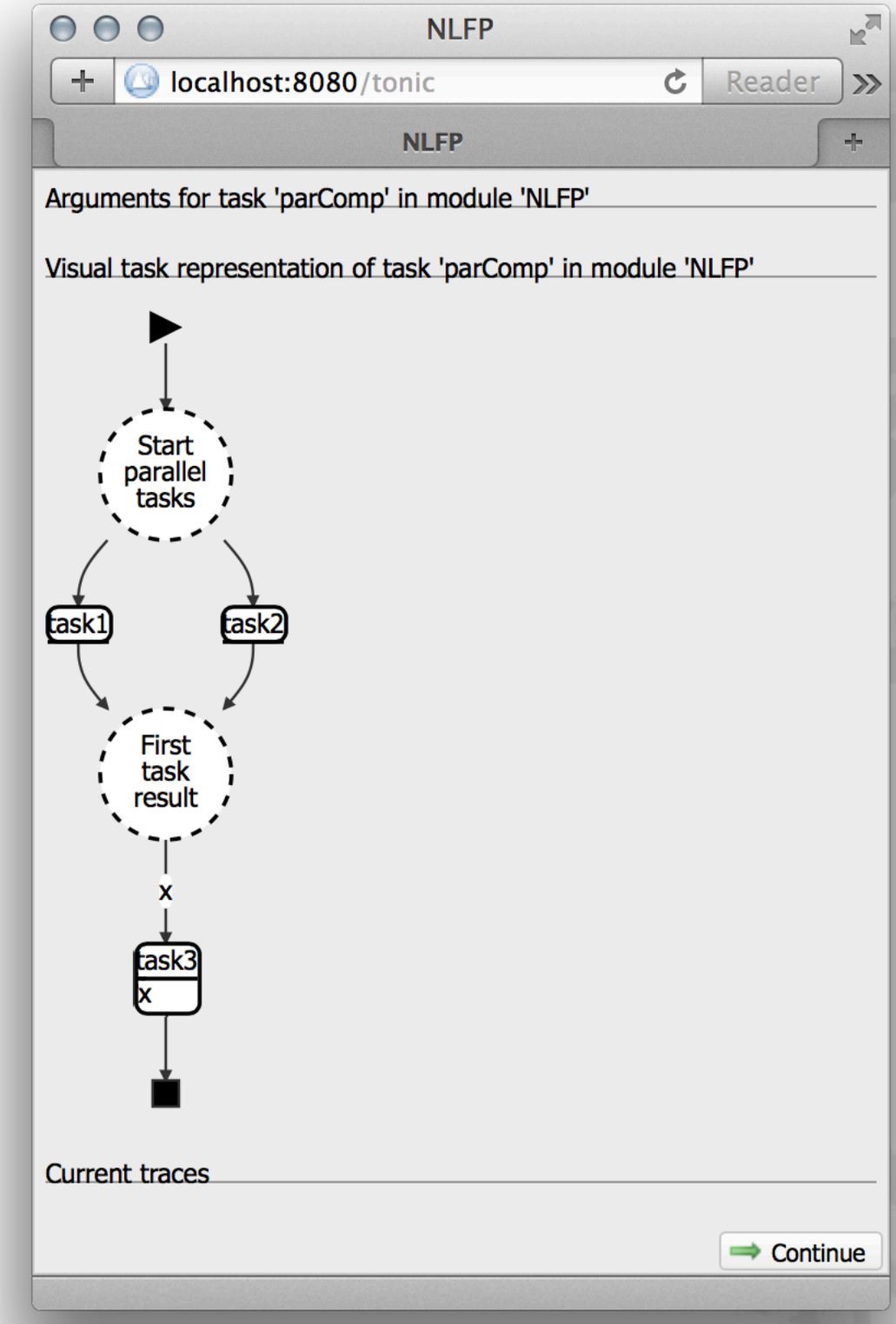
Let blocks

```
task1 >>= \x ->
  let y = 3
  in task 3 (x + y)
```



Parallel composition

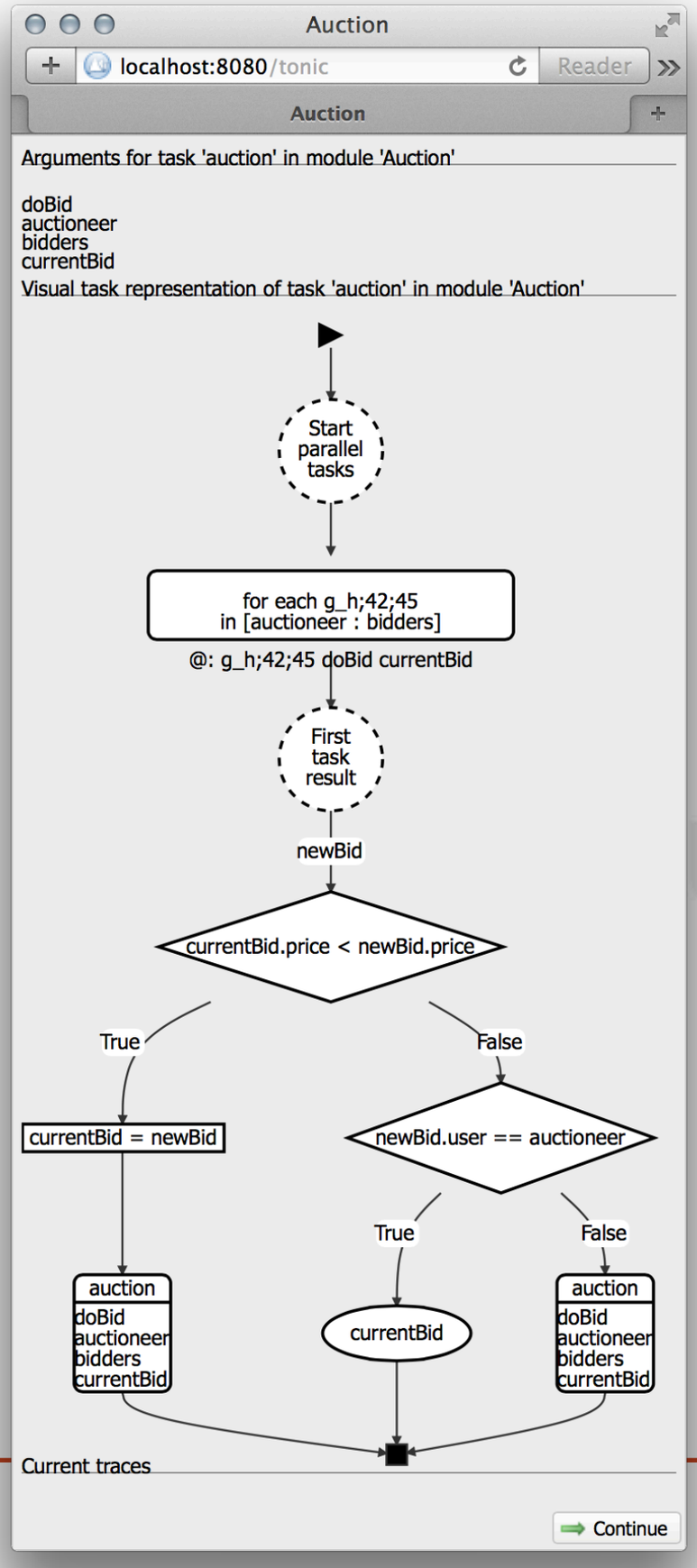
```
anyTask [task1, task2]
  >>= \x -> task3 x
```



English auction example: code

```
auction :: (Bid -> Task Bid) User [User] Bid -> Task Bid
auction doBid auctioneer bidders currentBid =
  anyTask [user @: doBid currentBid
           \\ user <- [auctioneer : bidders]] >>=
    \newBid ->
      if newBid.price > currentBid.price of
        (let currentBind = newBid
         in  auction doBid auctioneer bidders currentBid)
        (if (newBid.user == auctioneer)
           (return currentBid)
           (auction doBid auctioneer bidders currentBid))
```

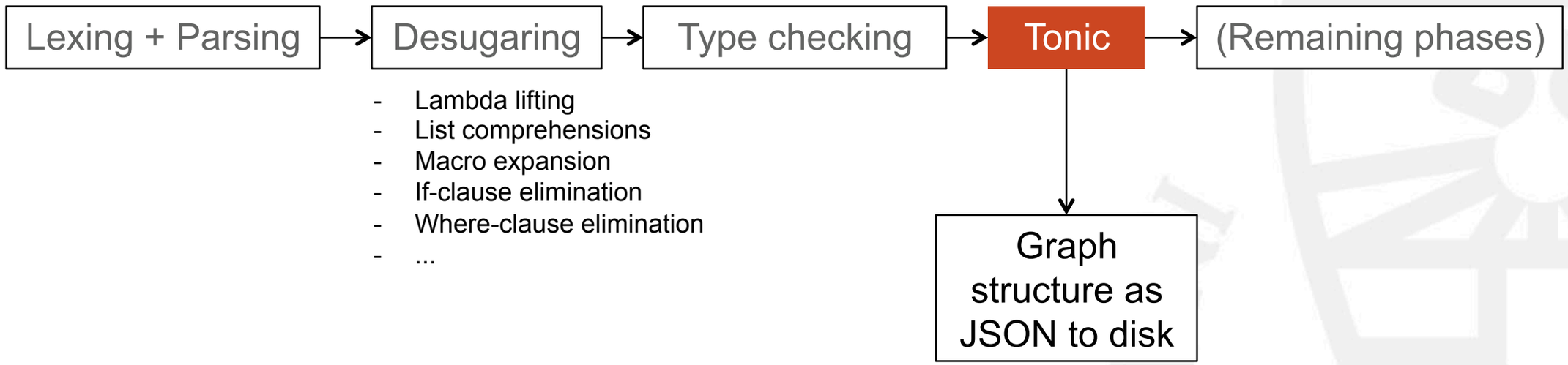
English auction example: rendering



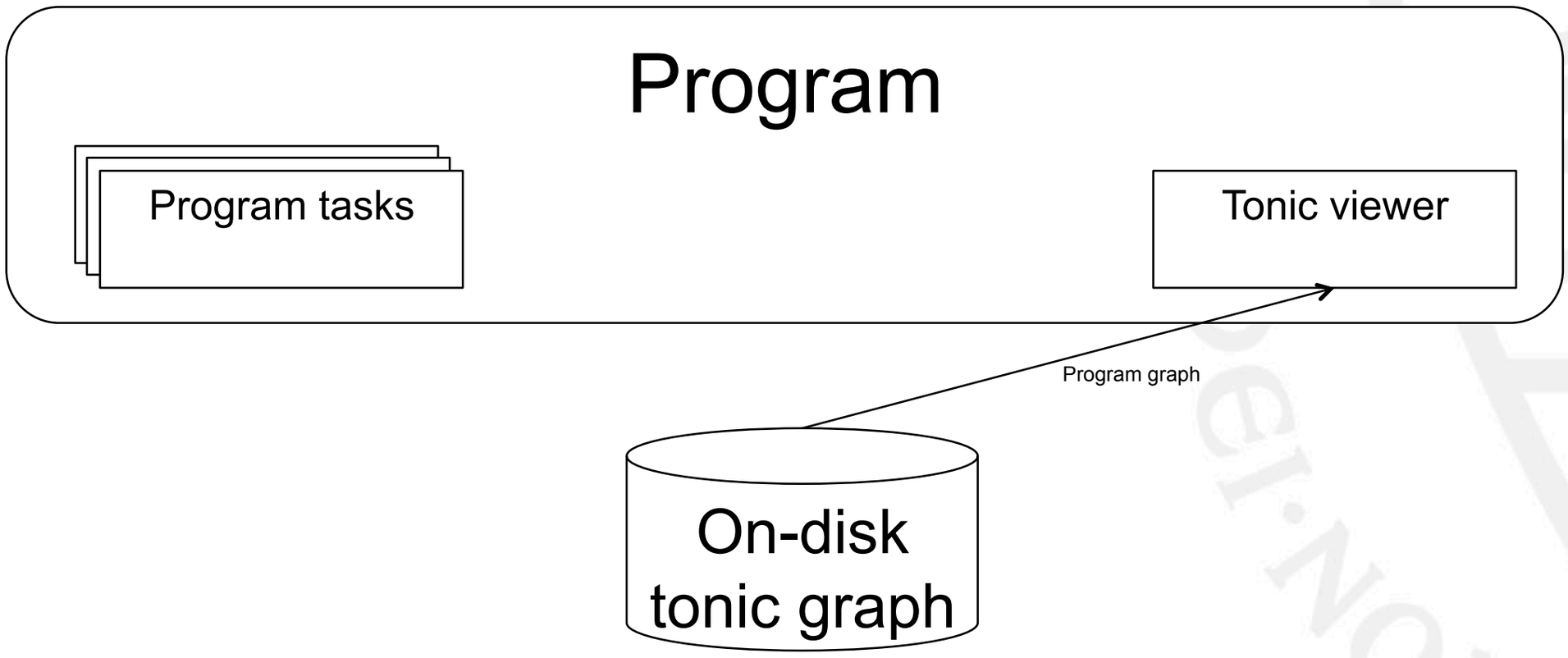
What to visualize?

- Visualize:
  - iTasks primitives and often used combinators (with special visual syntax)
    - Sequential and parallel task composition
    - Task assignment
    - User interaction (step)
    - Shares (shared memory mechanism)
  - Task application
    - If- and case blocks
    - Let blocks and beta-reducible lambdas
- Don’t visualize
  - Functions that are not a task
- Experimentation still required to determine exactly what (not) to show
  - Task function arguments?
  - Data types?
  - Custom visualizations for certain tasks?

Tonic in the compiler



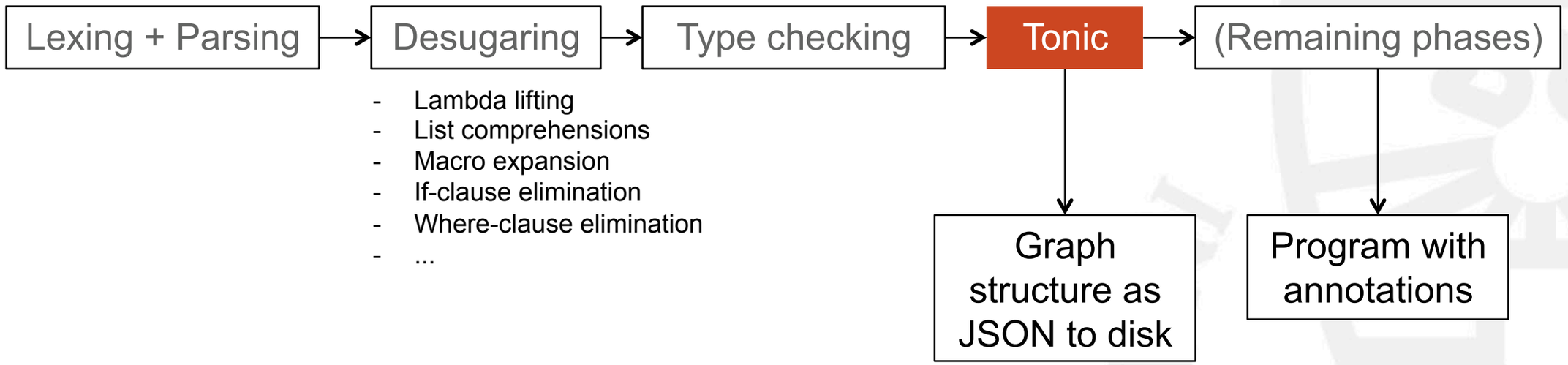
Program visualization



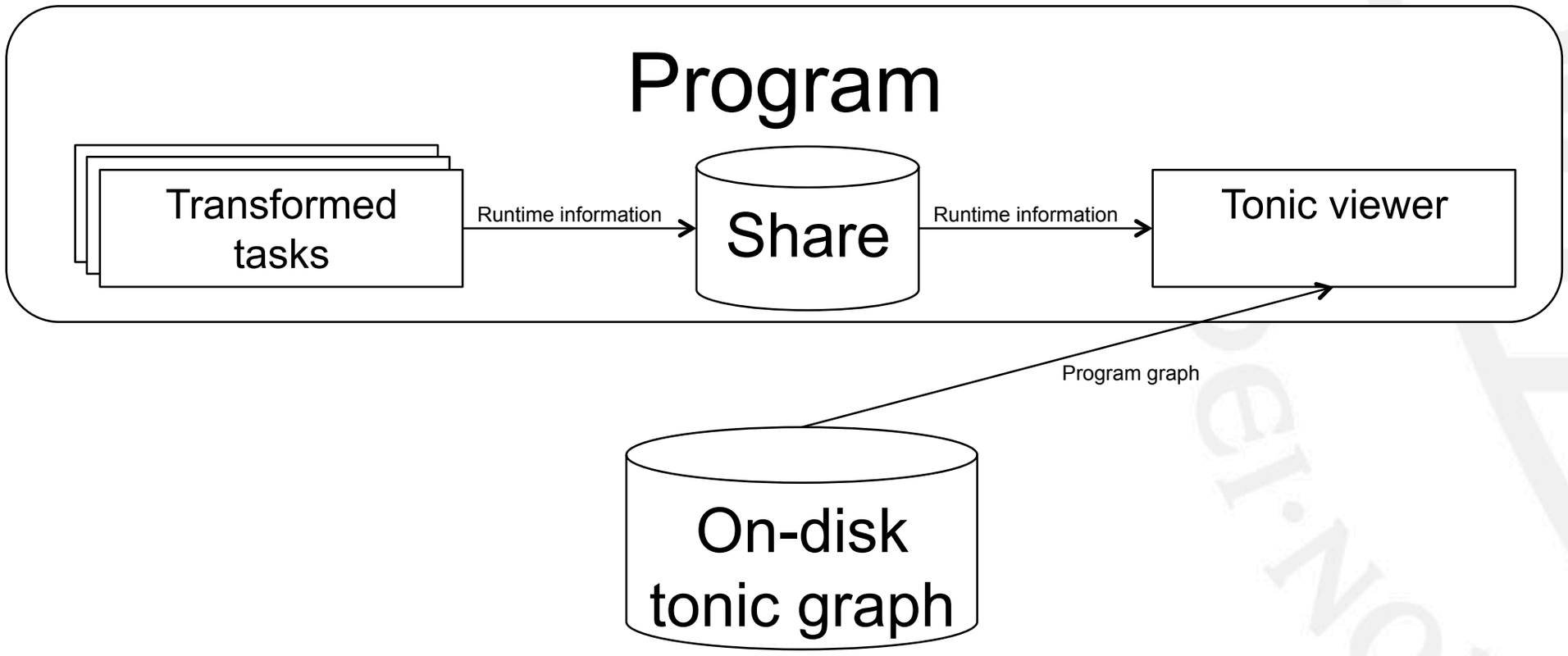
Dynamic behavior

- We want visualize dynamic program behavior as well
  - Current (sub)task being executed
  - Value of variables
  - Dynamic program graphs
- Tonic compiler applies semantics-preserving program transformations
  - Adds trace functions that gather relevant data

Tonic in the compiler



Runtime data gathering and visualization



Current status

- Static representation is generated and can be rendered in an iTasks application
  - Works for simple applications
- Automatic program annotation with traces still in very early stages
  - Current experiments annotate program manually
- Runtime information is not rendered yet

Future work

- Visualize more complex programs
- Evaluate annotations at the right time (i.e. deal with laziness)
- Overlay runtime information graphically
- Transform generated visual program graphs at run time
- Lessen dependency on custom Tonic compiler
- Generalize work to, e.g., monads and applicative functors

Thank you

Questions?