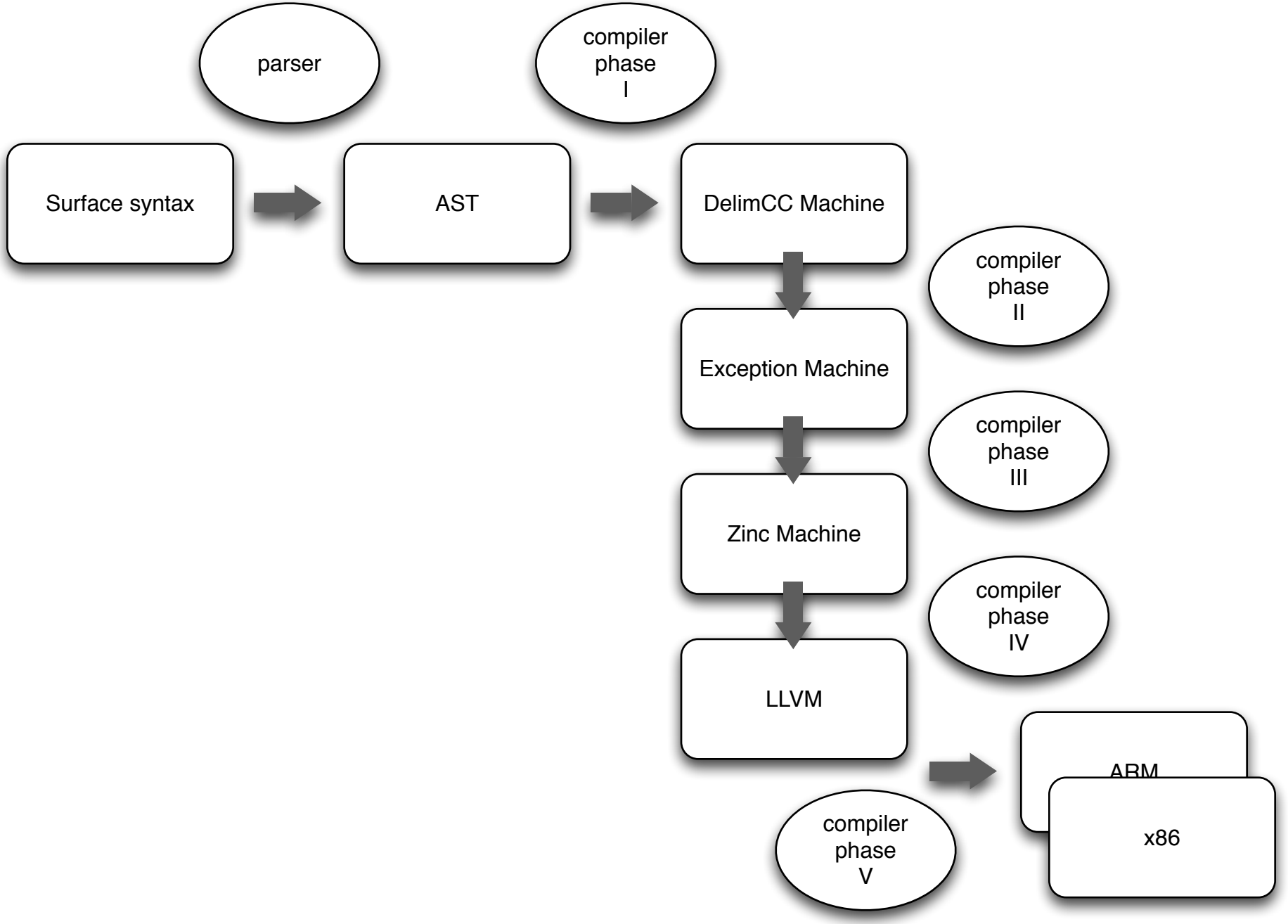
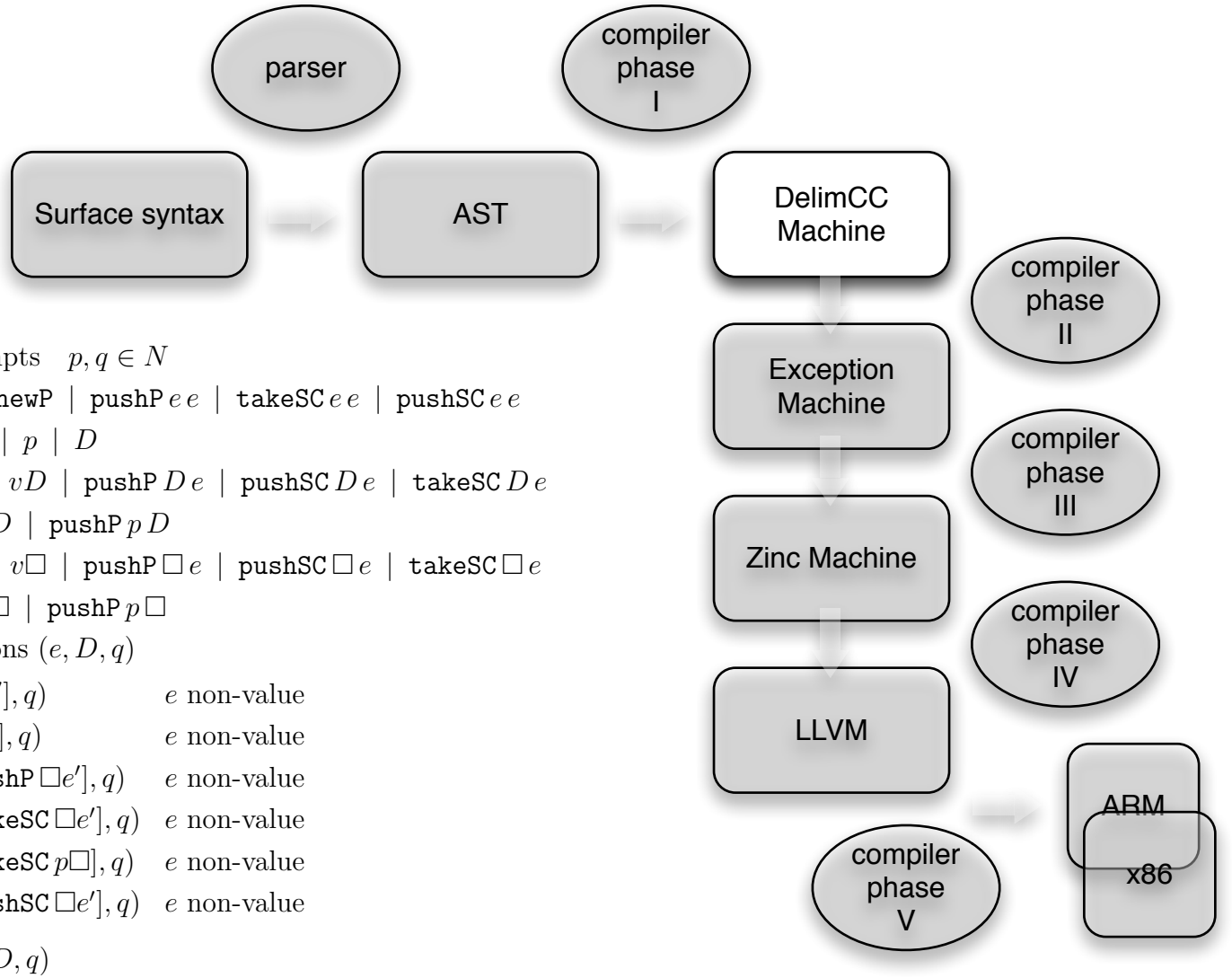

parser
compiler phase I
Surface syntax
AST
DelimCC Machine
compiler phase II
Exception Machine
compiler phase III
Zinc Machine
compiler phase IV
LLVM
compiler phase V
ARM
x86



Variables $x, y, \ldots$ Prompts $p, q \in N$

Expressions $e ::= v \mid e\,e \mid \texttt{newP} \mid \texttt{pushP}\,e\,e \mid \texttt{takeSC}\,e\,e \mid \texttt{pushSC}\,e\,e$

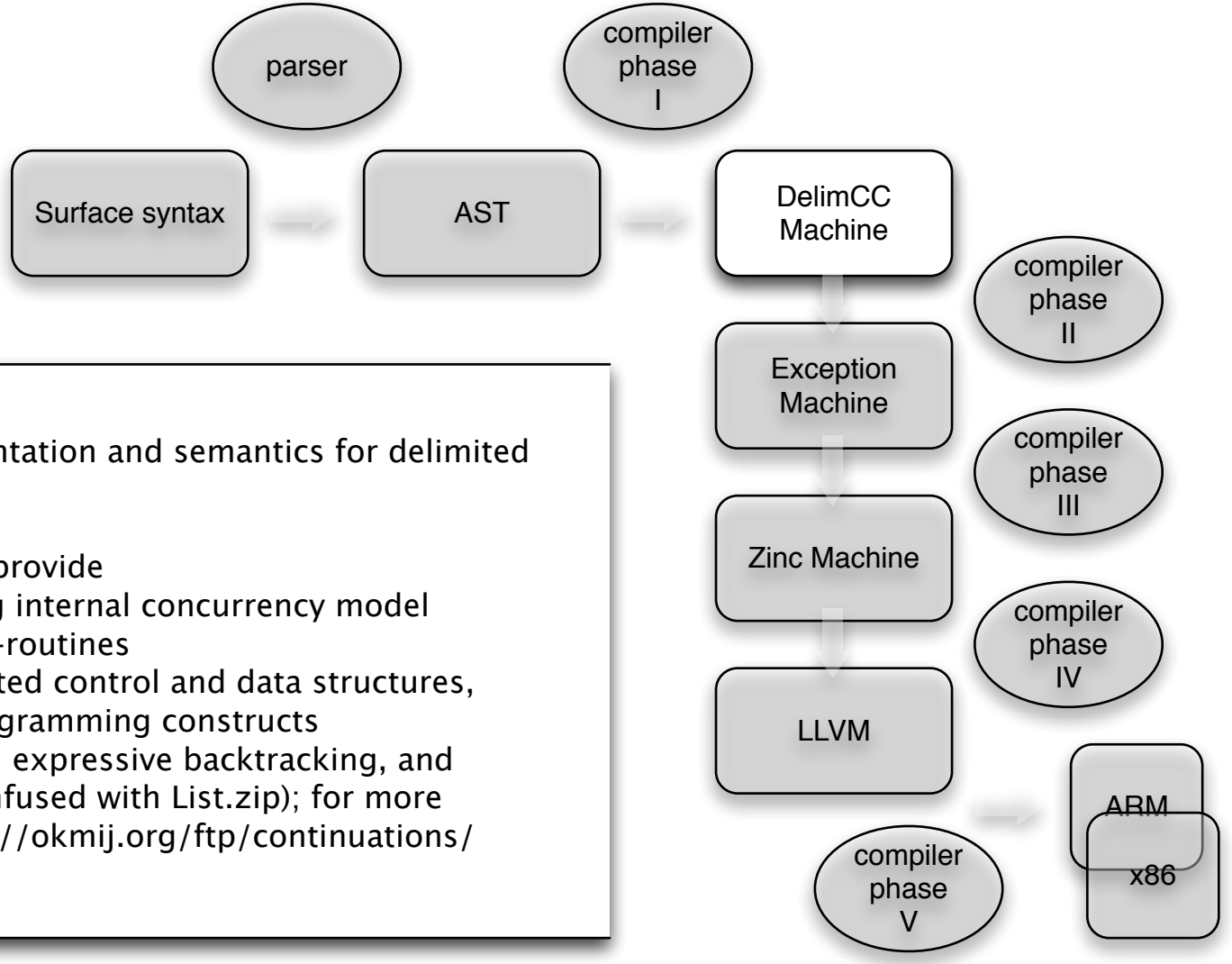
Values $v ::= x \mid \lambda x.\,e \mid p \mid D$

Contexts $D ::= \Box \mid De \mid vD \mid \texttt{pushP}\,D\,e \mid \texttt{pushSC}\,D\,e \mid \texttt{takeSC}\,D\,e$
$\mid \texttt{takeSC}\,p\,D \mid \texttt{pushP}\,p\,D$

Single Frame $::= \Box e \mid v\Box \mid \texttt{pushP}\,\Box\,e \mid \texttt{pushSC}\,\Box\,e \mid \texttt{takeSC}\,\Box\,e$
$\mid \texttt{takeSC}\,p\,\Box \mid \texttt{pushP}\,p\,\Box$

Transitions between configurations (e, D, q)

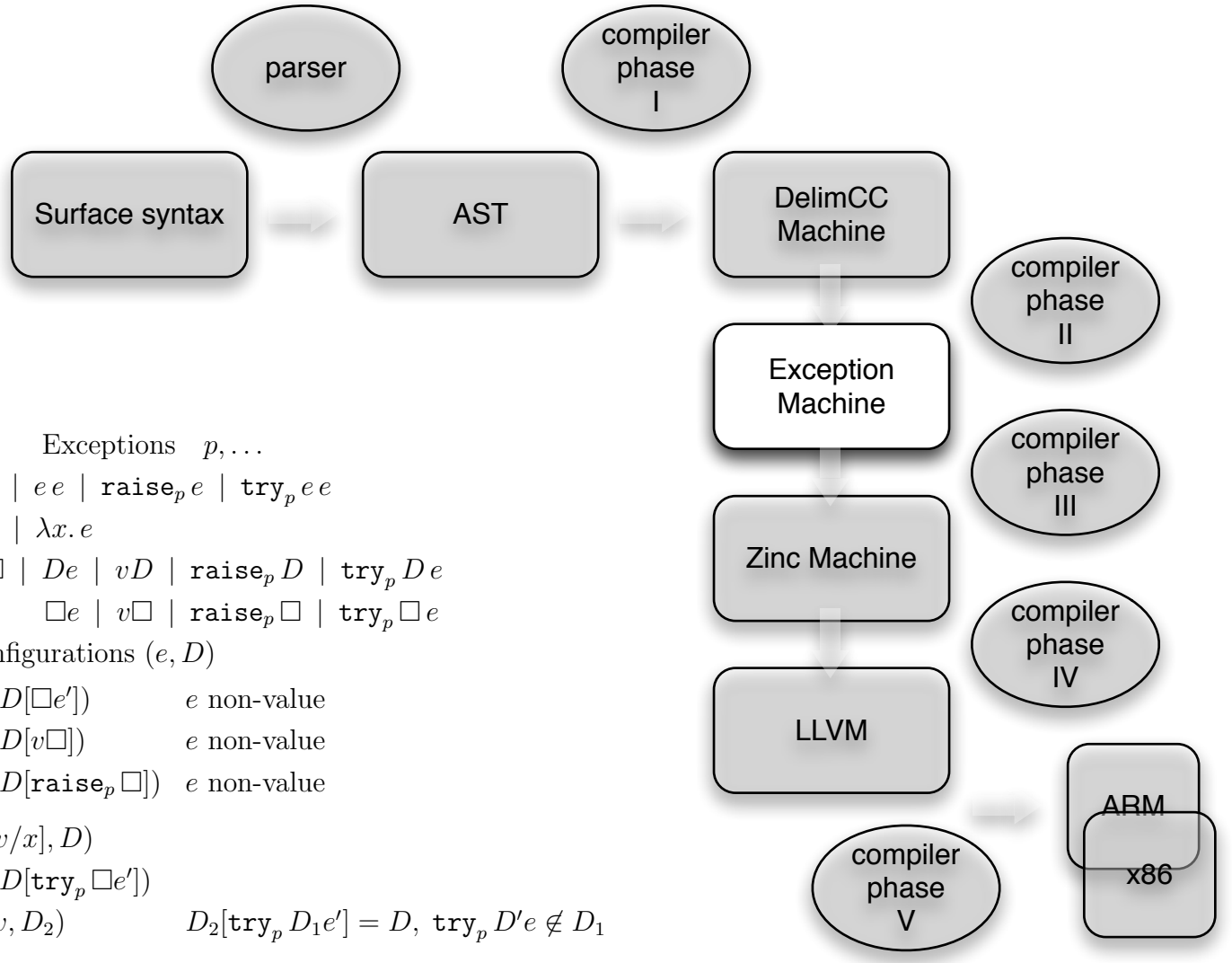
$$
\begin{array}{rcll}
(ee', D, q) & \mapsto & (e, D[\Box e'], q) & e \text{ non-value} \\
(ve, D, q) & \mapsto & (e, D[v\Box], q) & e \text{ non-value} \\
(\texttt{pushP}\,ee', D, q) & \mapsto & (e, D[\texttt{pushP}\,\Box e'], q) & e \text{ non-value} \\
(\texttt{takeSC}\,ee', D, q) & \mapsto & (e, D[\texttt{takeSC}\,\Box e'], q) & e \text{ non-value} \\
(\texttt{takeSC}\,pe, D, q) & \mapsto & (e, D[\texttt{takeSC}\,p\Box], q) & e \text{ non-value} \\
(\texttt{pushSC}\,ee', D, q) & \mapsto & (e, D[\texttt{pushSC}\,\Box e'], q) & e \text{ non-value} \\
\\
((\lambda x.\,e)v, D, q) & \mapsto & (e[v/x], D, q) & \\
(\texttt{newP}, D, q) & \mapsto & (q, D, q+1) & \\
(\texttt{pushP}\,pe, D, q) & \mapsto & (e, D[\texttt{pushP}\,p\Box], q) & \\
(\texttt{takeSC}\,pv, D, q) & \mapsto & (vD_1, D_2, q) & D_2[\texttt{pushP}\,pD_1] = D,\ \texttt{pushP}\,pD' \notin D_1 \\
(\texttt{pushSC}\,D'e, D, q) & \mapsto & (e, D[D'], q) & \\
\\
(v, D[D_1], q) & \mapsto & (D_1[v], D, q) & D_1 \text{ single frame} \\
(\texttt{pushP}\,pv, D, q) & \mapsto & (v, D, q) &
\end{array}
$$



Provides explicit representation and semantics for delimited continuations.

Delimited continuations provide

- ✯ Target for interpreting internal concurrency model
- ✯ Green threads and co-routines
- ✯ Many other sophisticated control and data structures, as well as generic programming constructs most notably: fair and expressive backtracking, and Zippers (not to be confused with List.zip); for more information see: http://okmij.org/ftp/continuations/



Variables $x, y, \ldots$ Exceptions $p, \ldots$

Expressions $e ::= v \mid e\,e \mid \mathtt{raise}_p\, e \mid \mathtt{try}_p\, e\, e$

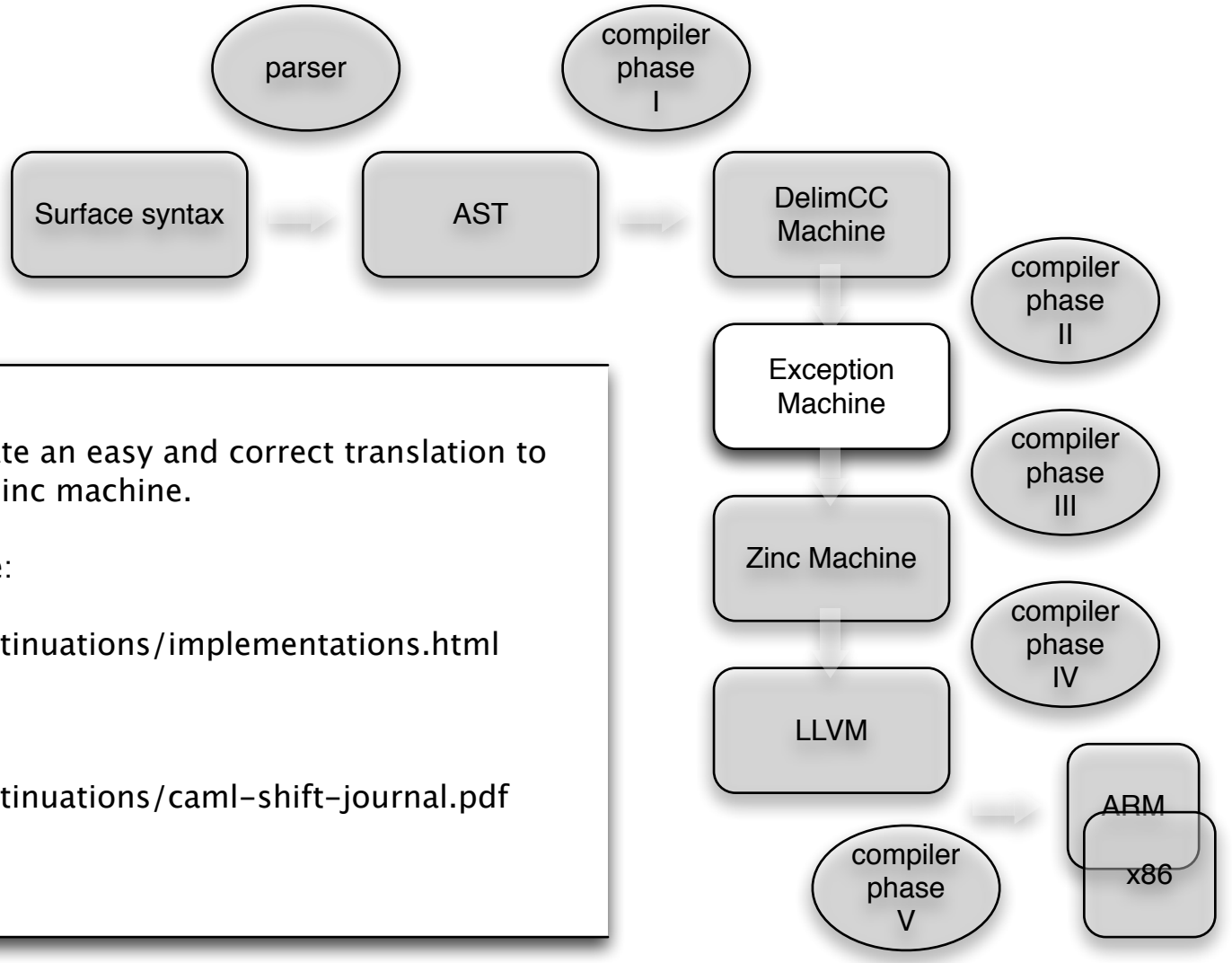
Values $v ::= x \mid \lambda x.\, e$

Contexts $D ::= \Box \mid De \mid vD \mid \mathtt{raise}_p\, D \mid \mathtt{try}_p\, D\, e$

Single Frame $::= \Box e \mid v\Box \mid \mathtt{raise}_p\, \Box \mid \mathtt{try}_p\, \Box\, e$

Transitions between configurations (e, D)

$$
\begin{array}{rcll}
(ee', D) & \mapsto & (e, D[\Box e']) & e \text{ non-value} \\
(ve, D) & \mapsto & (e, D[v\Box]) & e \text{ non-value} \\
(\mathtt{raise}_p\, e, D) & \mapsto & (e, D[\mathtt{raise}_p\, \Box]) & e \text{ non-value} \\
((\lambda x.\, e)v, D) & \mapsto & (e[v/x], D) & \\
(\mathtt{try}_p\, ee', D) & \mapsto & (e, D[\mathtt{try}_p\, \Box e']) & \\
(\mathtt{raise}_p\, v, D) & \mapsto & (e'v, D_2) & D_2[\mathtt{try}_p\, D_1 e'] = D,\ \mathtt{try}_p\, D' e \notin D_1 \\
(v, D[D_1]) & \mapsto & (D_1[v], D) & D_1 \text{ single frame} \\
(\mathtt{try}_p\, ve', D) & \mapsto & (v, D) &
\end{array}
$$



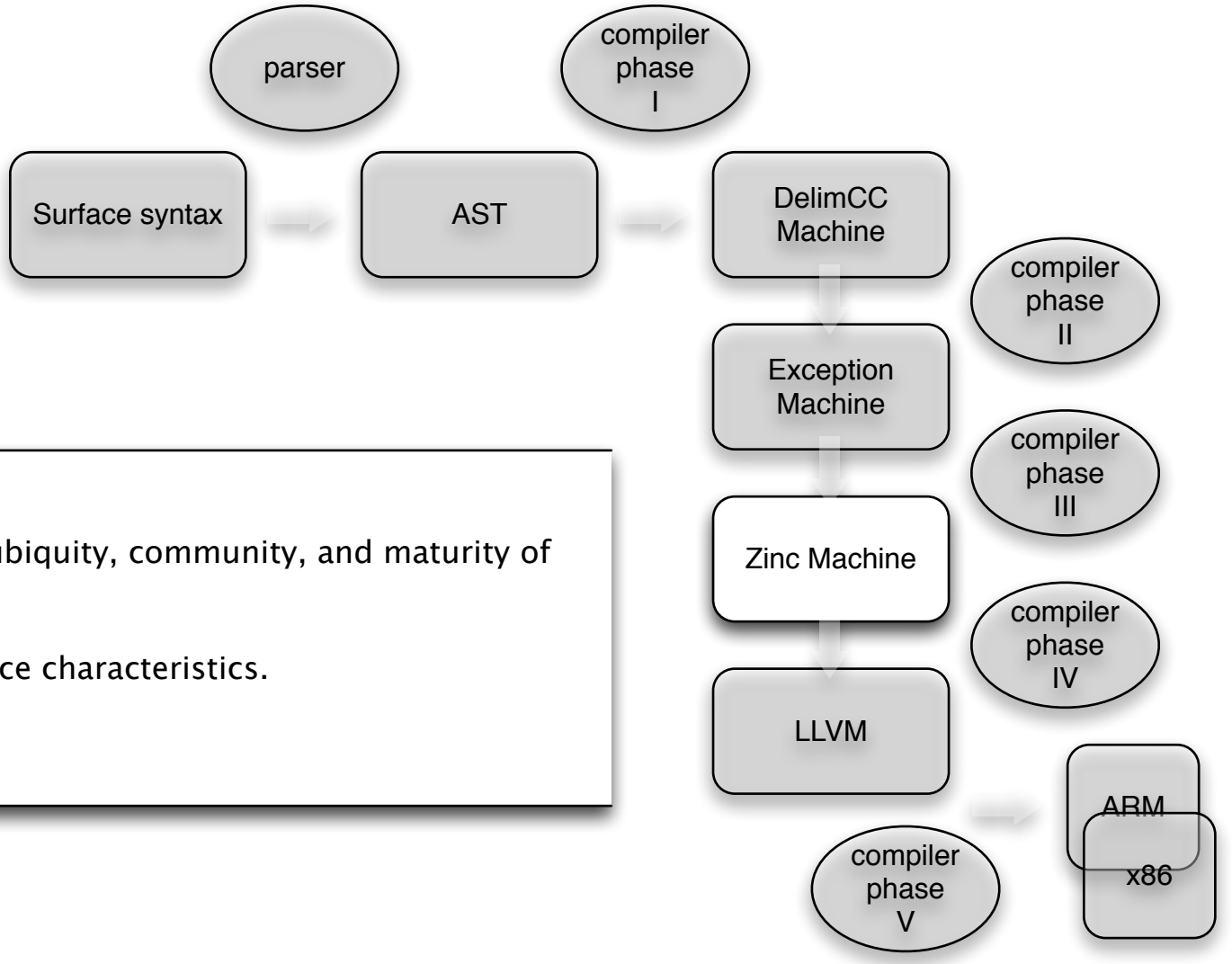
Principally here to facilitate an easy and correct translation to the target machine, the Zinc machine.

For more information see:

http://okmij.org/ftp/continuations/implementations.html

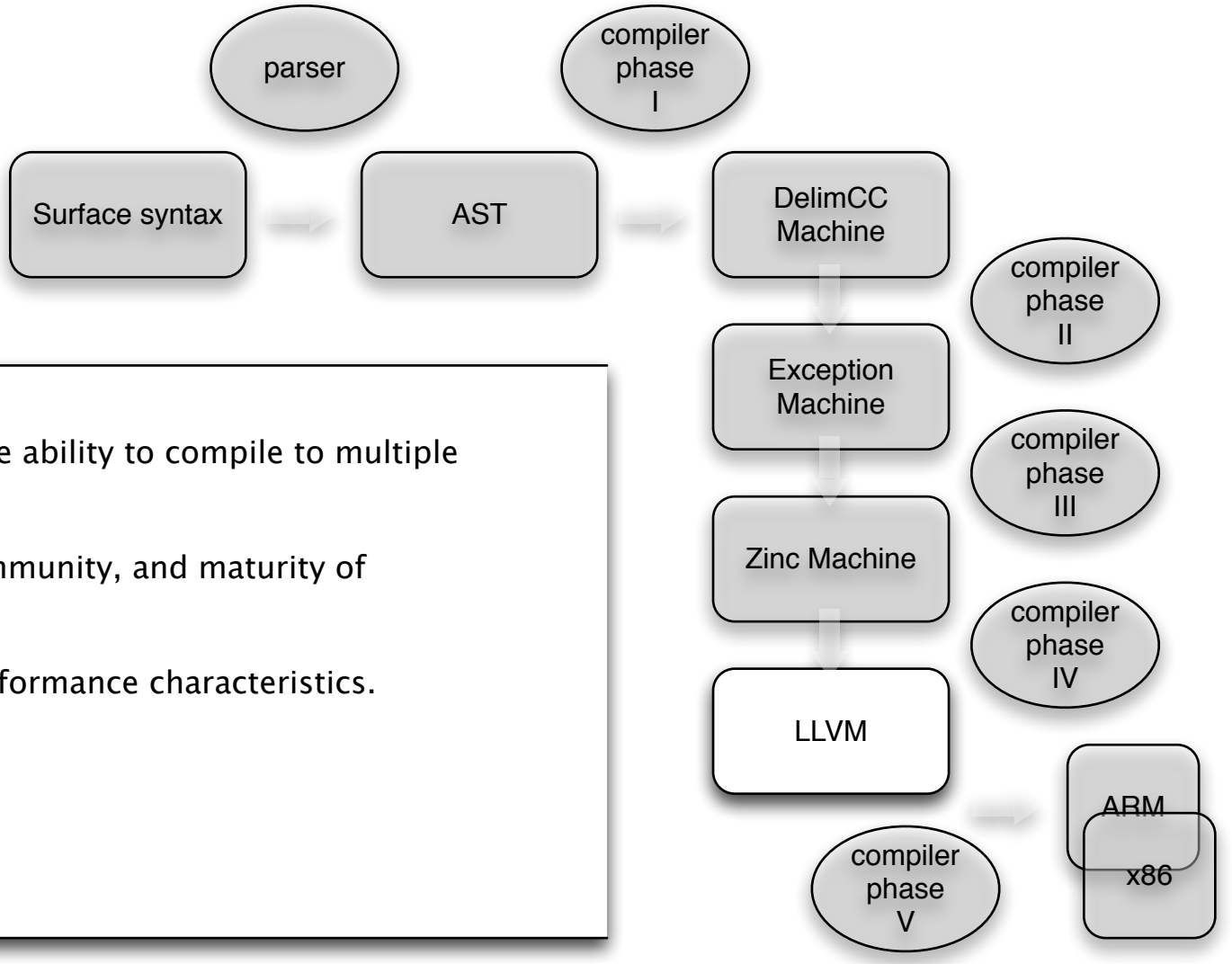
and especially

http://okmij.org/ftp/continuations/caml-shift-journal.pdf



Principally here due to ubiquity, community, and maturity of the technology.

It has known performance characteristics.

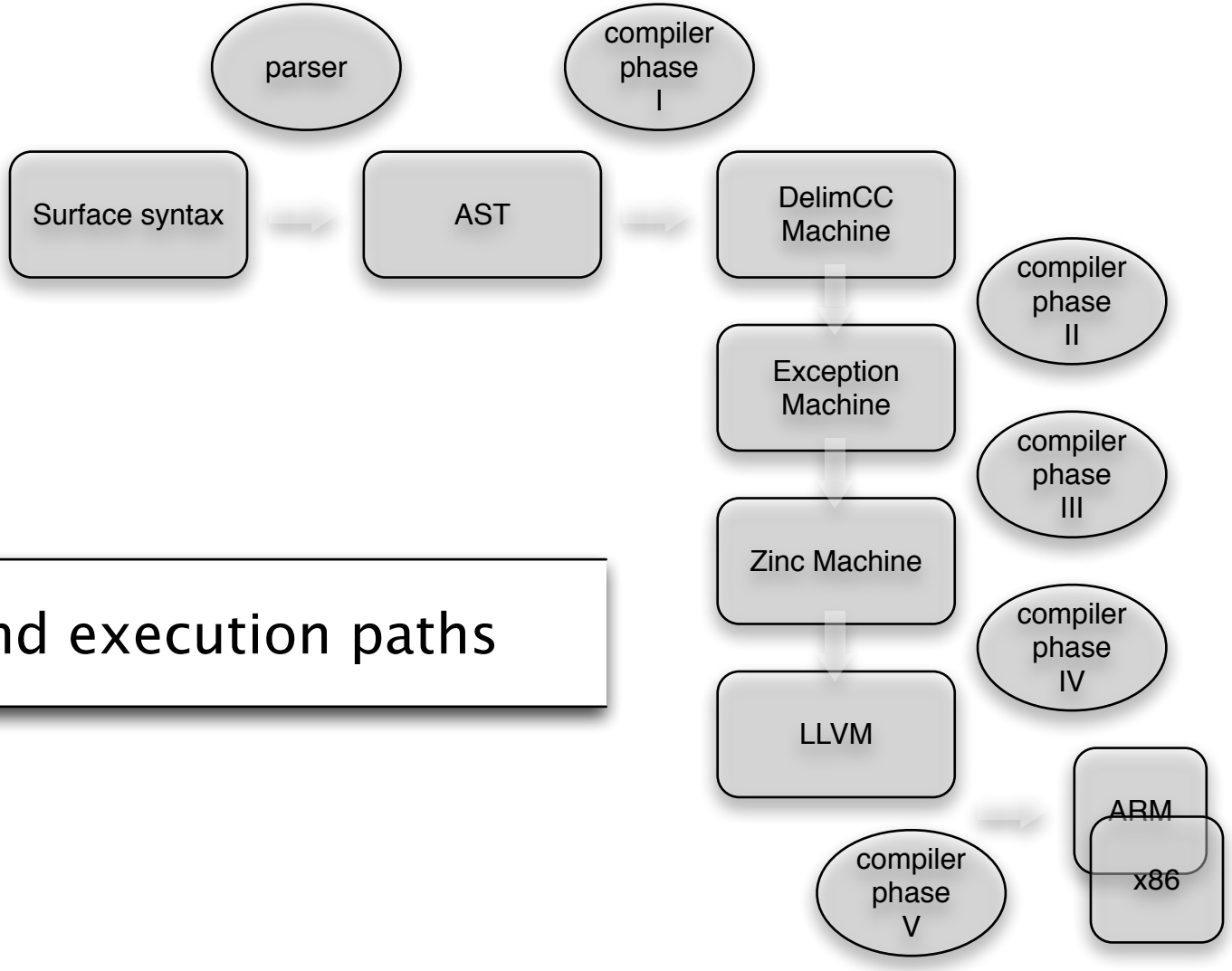


Principally here to provide ability to compile to multiple hardware targets.

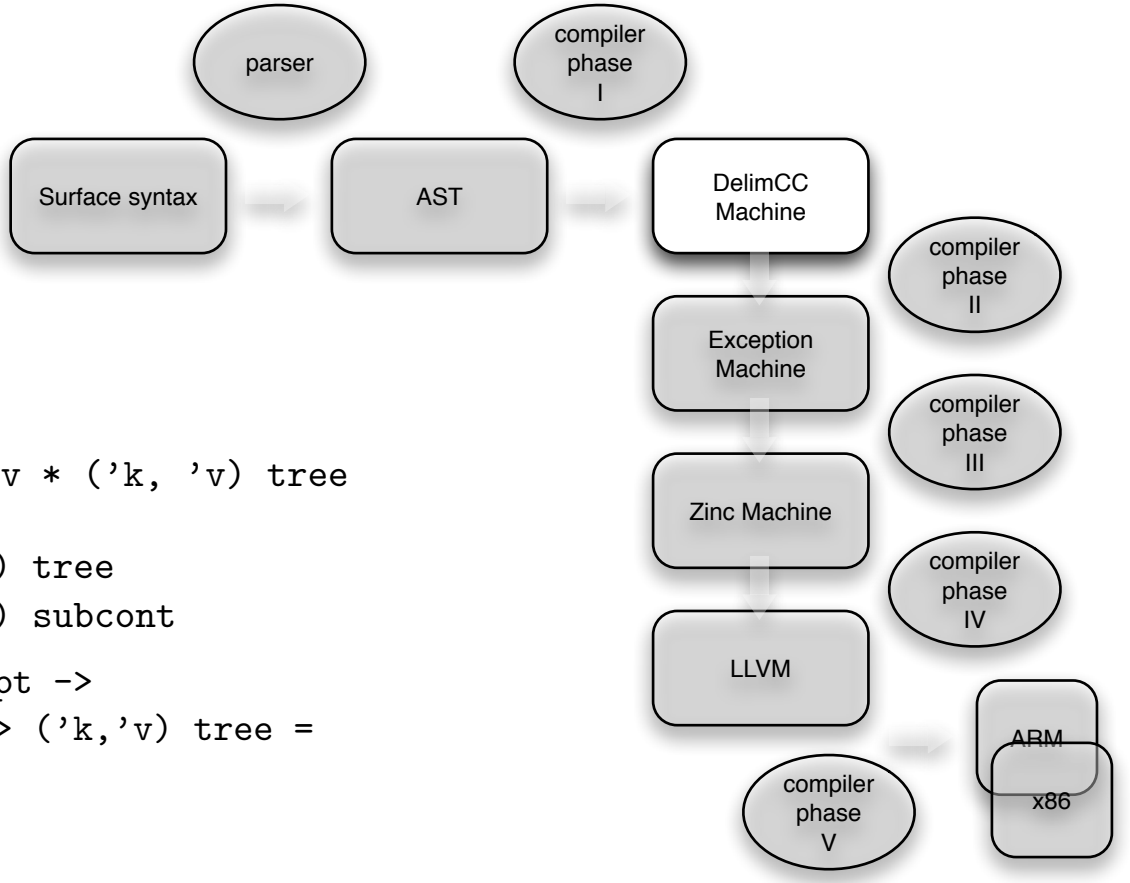
Also enjoys ubiquity, community, and maturity of the technology.

Toolchain has known performance characteristics.

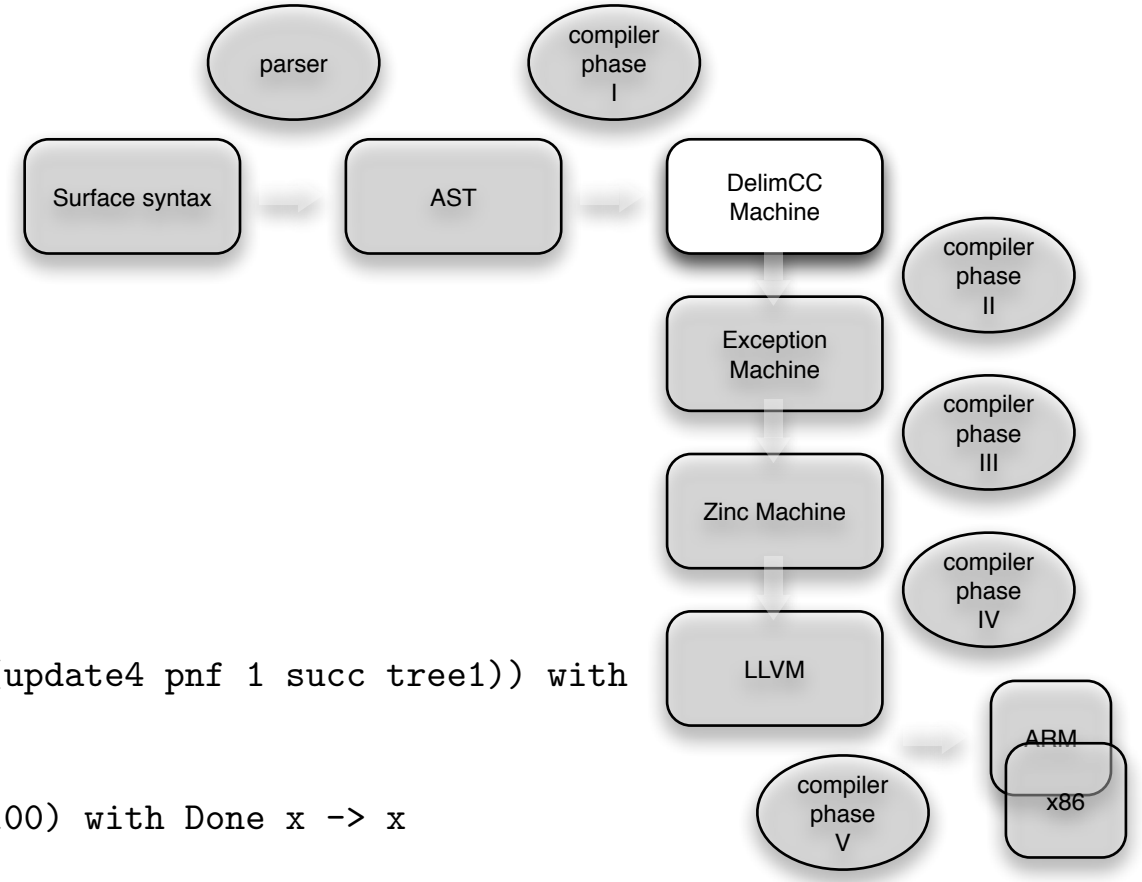
Examples and execution paths



Example: in place update on failed search

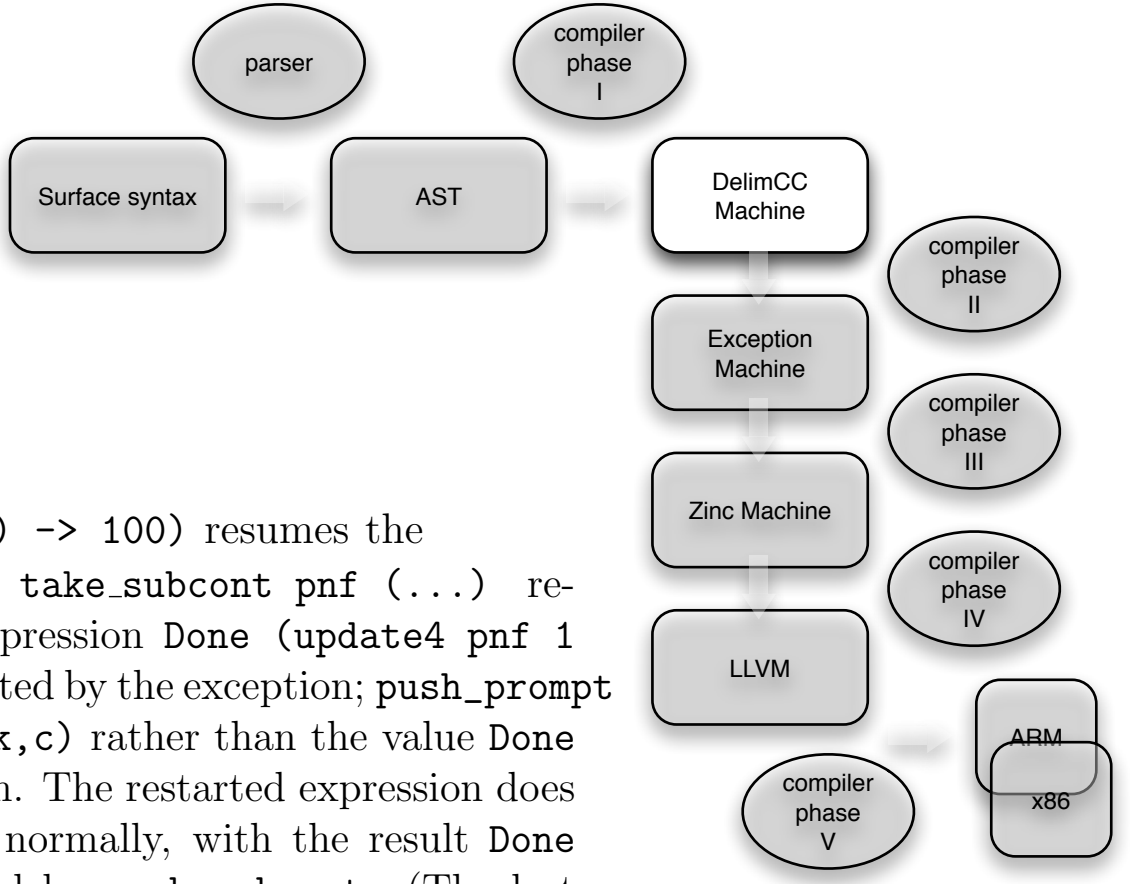



```
type ('k, 'v) tree =
  | Empty
  | Node of ('k, 'v) tree * 'k * 'v * ('k, 'v) tree

type ('k,'v) res = Done of ('k,'v) tree
  | ReqNF of 'k * ('v,('k,'v) res) subcont

let rec update4 : ('k,'v) res prompt ->
  'k -> ('v->'v) -> ('k,'v) tree -> ('k,'v) tree =
  fun pnf k f ->
    let rec loop = function
    | Empty -> Node(Empty,k,
                take_subcont pnf (fun c () -> ReqNF (k,c)),Empty)
    ...
    | Node (l,k1,v1,r) ->
        begin
          match compare k k1 with
          | 0 -> Node(l,k1,f v1,r)
          | n when n < 0 -> Node(loop l,k1,v1,r)
          | _            -> Node(l,k1,v1,loop r)
        end
    in loop
```



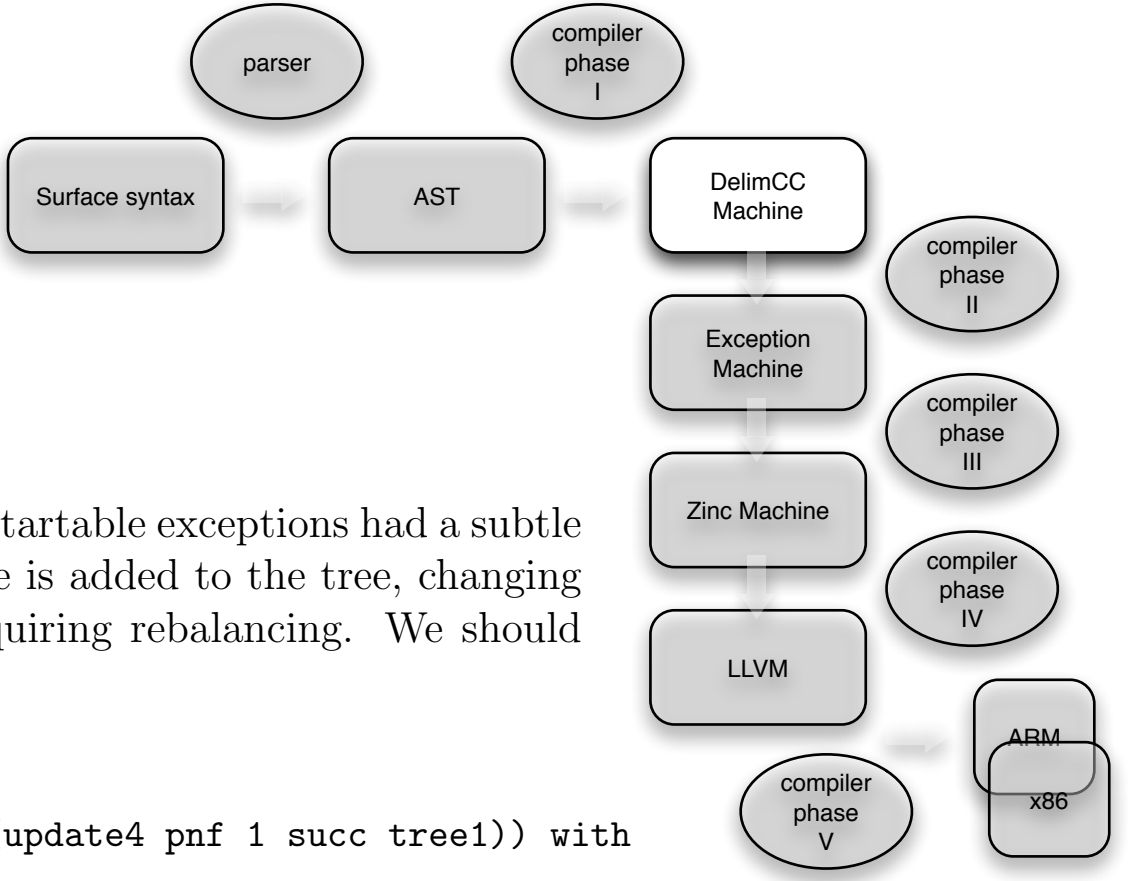
Example: client code

```
let pnf = new_prompt () in
match push_prompt pnf (fun () -> Done (update4 pnf 1 succ tree1)) with
  | Done tree    -> tree
  | ReqNF (k,c)  ->
      match push_subcont c (fun () -> 100) with Done x -> x
```



Example: client code

The function call `push_subcont c (fun () -> 100)` resumes the evaluation of `update4` as if the expression `take_subcont pnf (...)` returned `100`. We have started with the expression `Done (update4 pnf 1 succ tree1)`, whose evaluation was interrupted by the exception; `push_prompt` has caught the exception, yielding `ReqNF (k,c)` rather than the value `Done tree` expected as the result of our expression. The restarted expression does not raise any further exceptions, finishing normally, with the result `Done tree`. The result becomes the value yielded by `push_subcont`. (The last `Done x` pattern-match in the sample application is therefore total.)

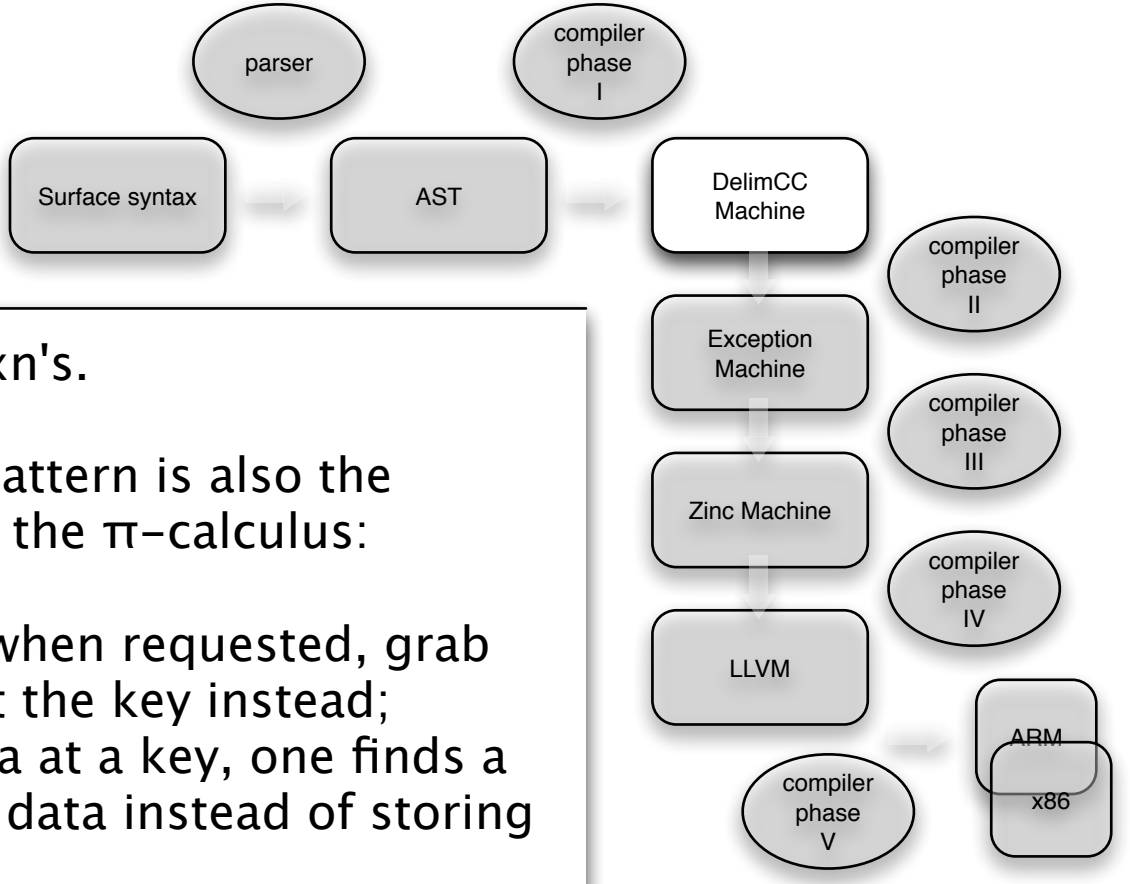


Example: client code

Our sample applications that relied on restartable exceptions had a subtle flaw. Upon the exception restart a new node is added to the tree, changing the height of its branch and potentially requiring rebalancing. We should have written

```
let pnf = new_prompt () in
match push_prompt pnf (fun () -> Done (update4 pnf 1 succ tree1)) with
  | Done tree    -> tree
  | ReqNF (k,c)  ->
     rebalance (match push_subcont c (fun () -> 100) with Done x -> x)
```

if we eventually discover that the key was missing and a new node has to be adjoined, we go 'back in time' and add the call to `rebalance` at the beginning.

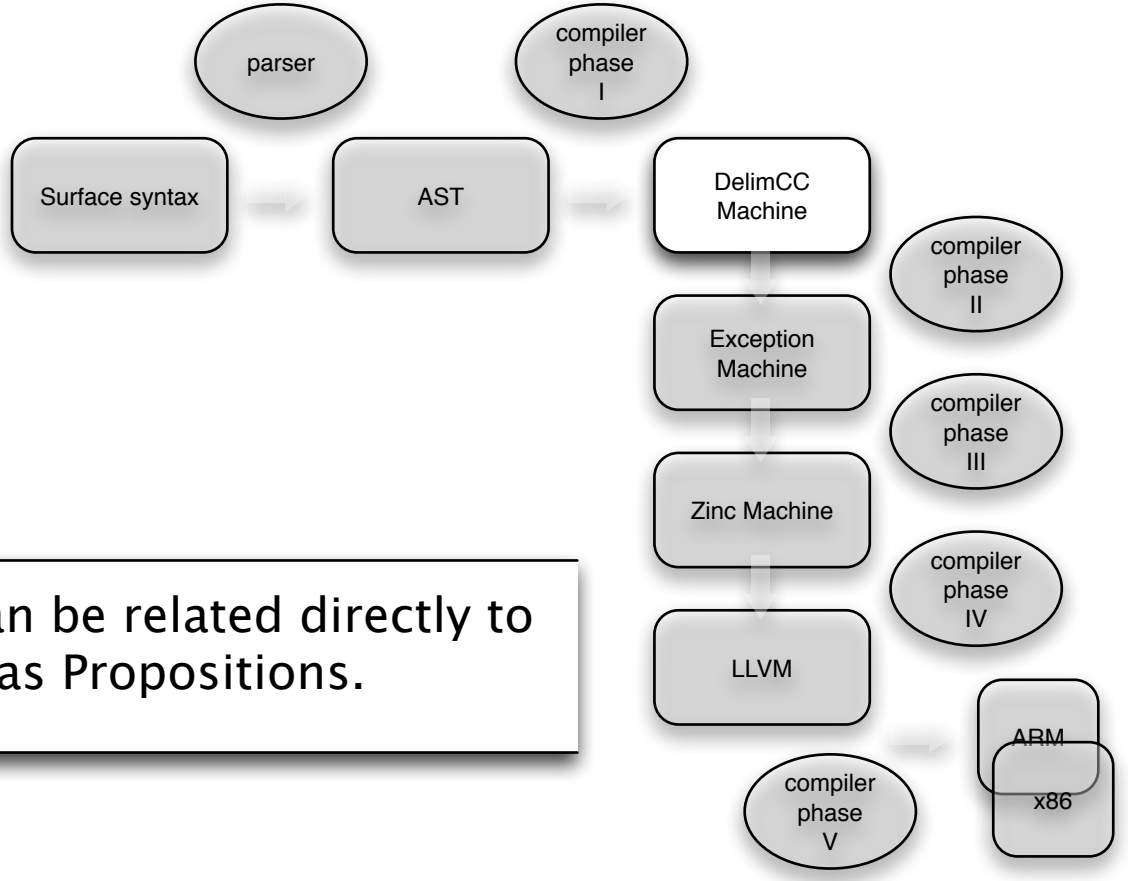


Time travel is the essence of txn's.

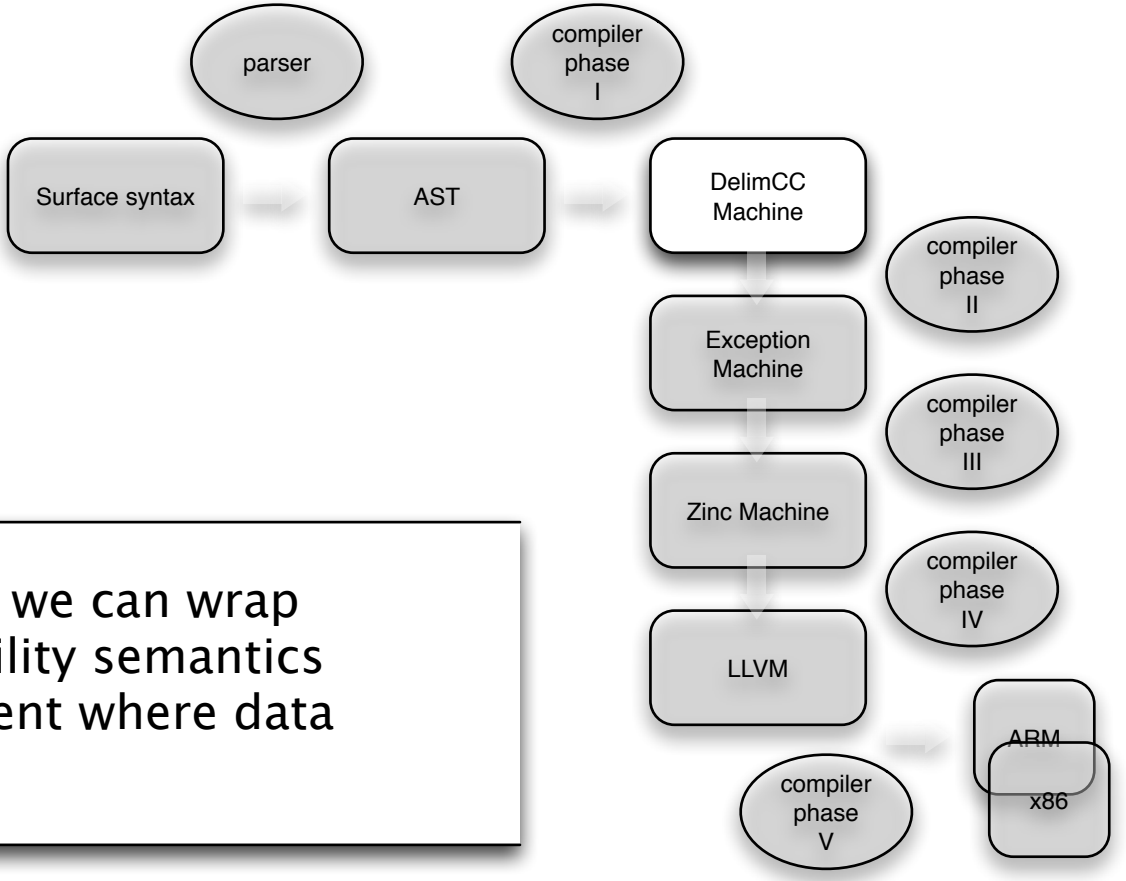
The key-match-continuation pattern is also the essence of an interpretation of the π-calculus:

* if data isn't present at a key when requested, grab the continuation and store it at the key instead;
* if, on an attempt to store data at a key, one finds a continuation, apply that to the data instead of storing it.

Generalizing the π-calculus: all major distributed programming models are realized by combinations on the synchronization between continuation and data and whether either are ephemeral or persisted.

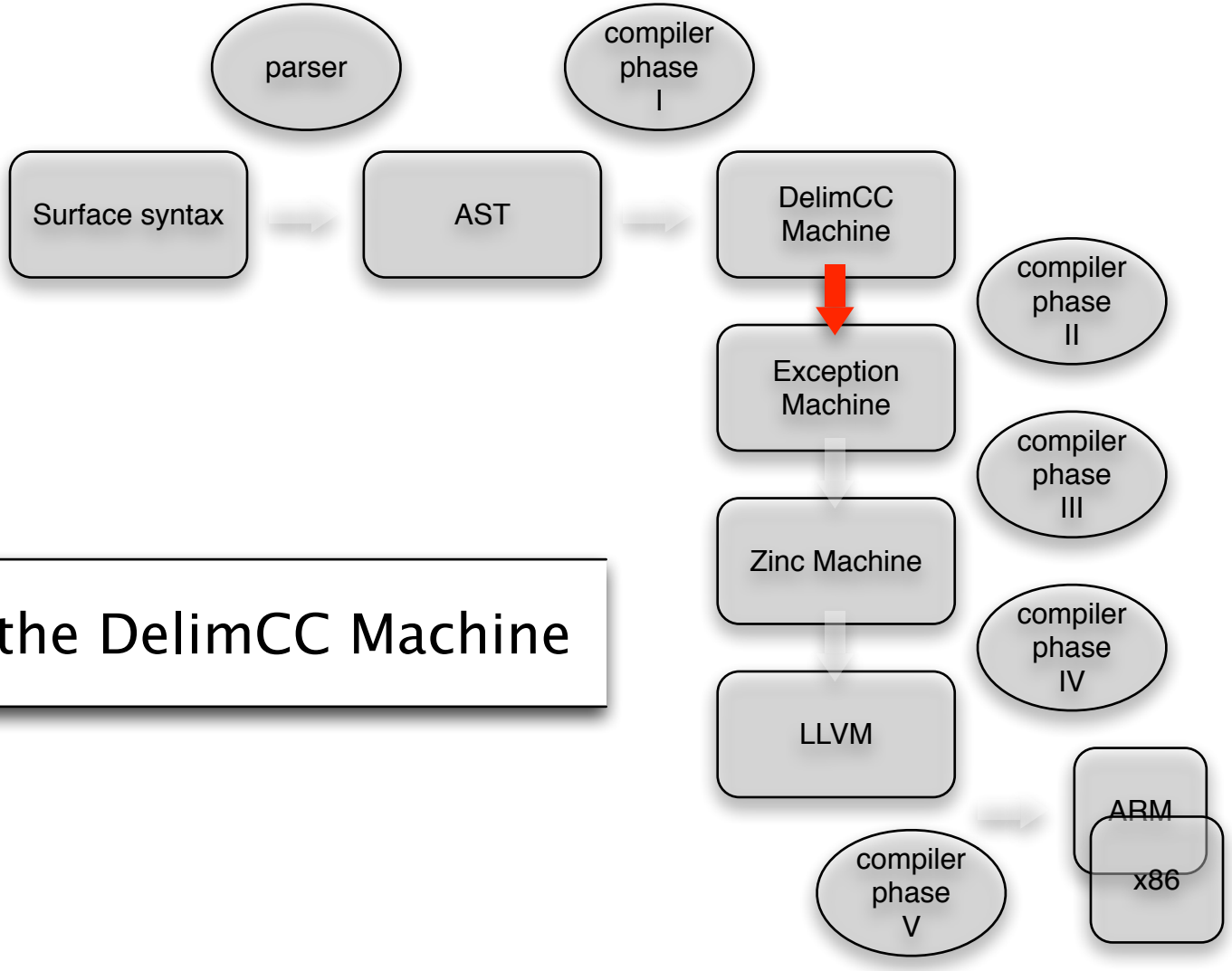


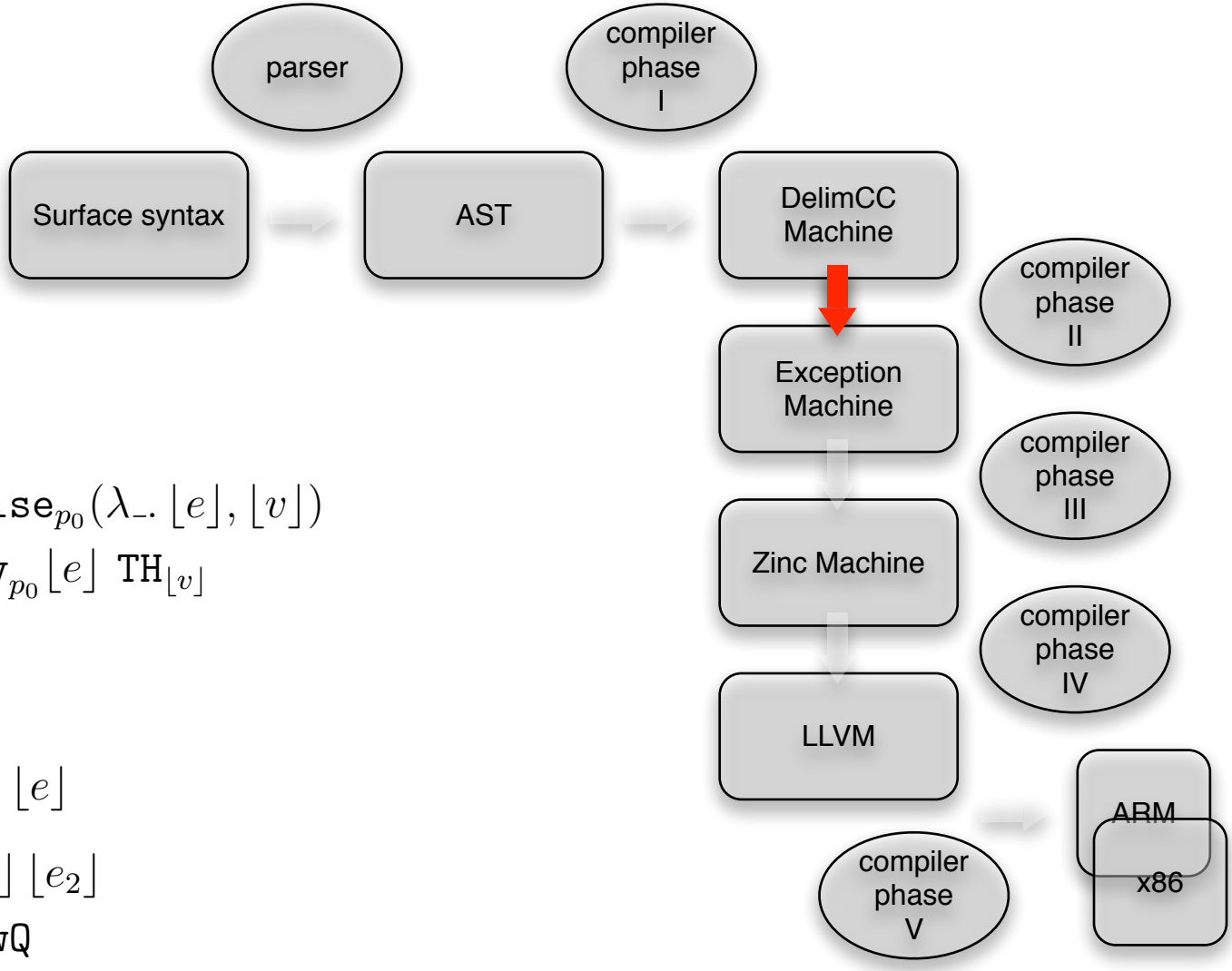
This generalized mechanism can be related directly to Lindley's encoding of Sessions as Propositions.



As the example makes clear, we can wrap transactional and other visibility semantics around a synchronization event where data meets continuation.

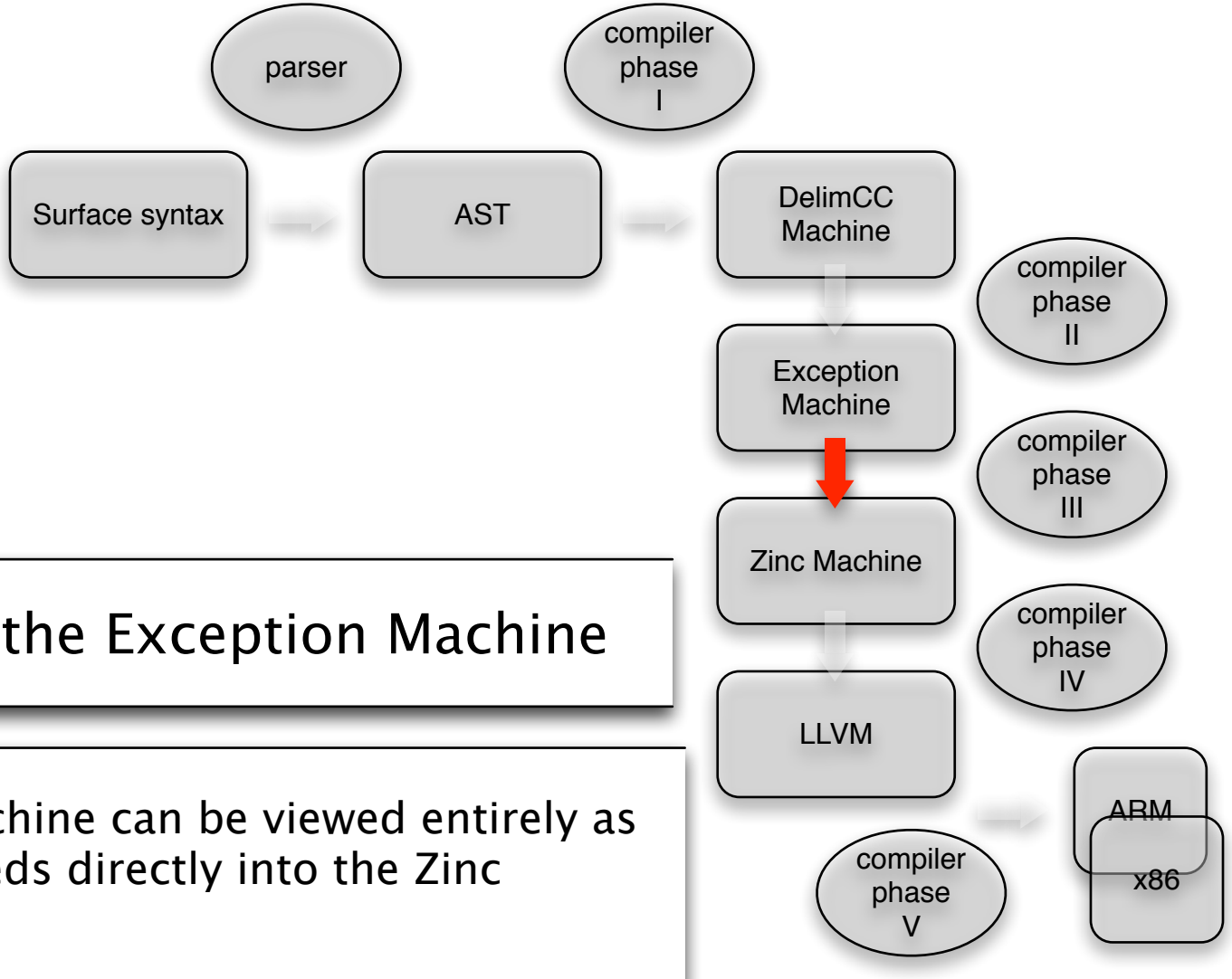
Eliminating the DelimCC Machine





$$
\begin{array}{lcll}
\lfloor \mathtt{takeSC}\, v\, (\lambda\_.\, e) \rfloor & = & \mathtt{raise}_{p_0}(\lambda\_.\, \lfloor e \rfloor, \lfloor v \rfloor) & \\
\lfloor \mathtt{pushP}\, v\, e \rfloor & = & \mathtt{try}_{p_0} \lfloor e \rfloor\ \mathtt{TH}_{\lfloor v \rfloor} & \\
\lfloor x \rfloor & = & x & \\
\lfloor p \rfloor & = & q & \\
\lfloor \lambda x.\, e \rfloor & = & \lambda x.\, \lfloor e \rfloor & \\
\lfloor e_1\, e_2 \rfloor & = & \lfloor e_1 \rfloor\, \lfloor e_2 \rfloor & \\
\lfloor \mathtt{newP} \rfloor & = & \mathtt{newQ} & \\
\lfloor \mathtt{pushP}\, e\, e' \rfloor & = & \lfloor (\lambda x.\, \mathtt{pushP}\, x\, e')e \rfloor & e \text{ non-value}, x \text{ fresh} \\
\lfloor \mathtt{takeSC}\, e\, \lambda\_.\, e' \rfloor & = & \lfloor (\lambda x.\, \mathtt{takeSC}\, x\, \lambda\_.\, e')e \rfloor & e \text{ non-value}, x \text{ fresh}
\end{array}
$$

$$\mathtt{TH}_q = \lambda y.\, \mathtt{if}\ (\lambda y_2.\, (q, y_2))(\mathtt{snd}\, y)\ \mathtt{then}\ \mathtt{fst}\, y\, ()\ \mathtt{else}\ \mathtt{raise}_{p_0}\, y$$



Eliminating the Exception Machine

The exception machine can be viewed entirely as a fiction that embeds directly into the Zinc Machine.

The Zinc machine already contains a register for exception handler pointers.