Selective Functors in Build Systems

Jeremie Dimino, Jane Street

@diml

@dimenix

Jane Street

Jane Street

OPEN SOURCE

DUNE

BASE

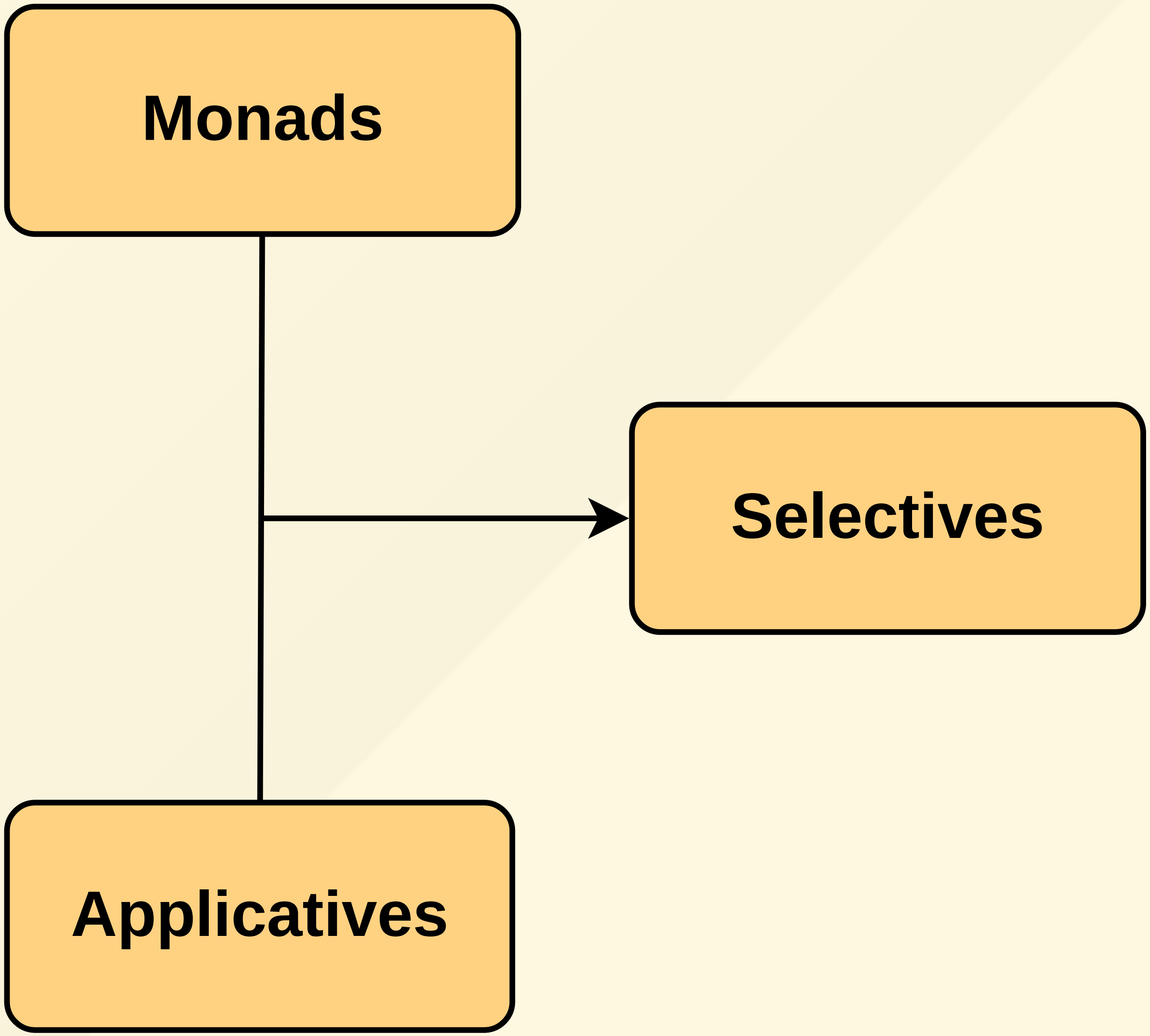
CORE

ASYNC

INCR_DOM

INCREMENTAL

What are selective functors?


Monads
Selectives
Applicatives

Selective Functors

```haskell
class Applicative f => Selective f where
  select :: f (Either a b) -> f (a -> b) -> f b
```

Operator: `<*?`

Selective combinators

```haskell
whenS :: Selective f => f Bool -> f () -> f ()
branch :: Selective f => f (Either a b)
  -> f (a -> c) -> f (b -> c) -> f c
ifS :: Selective f => f Bool -> f a -> f a -> f a
(<||>) :: Selective f => f Bool -> f Bool -> f Bool
(<&&>) :: Selective f => f Bool -> f Bool -> f Bool
anyS :: Selective f => (a -> f Bool) -> [a] -> f Bool
allS :: Selective f => (a -> f Bool) -> [a] -> f Bool
fromMaybeS :: Selective f => f a -> f (Maybe a) -> f a
whileS :: Selective f => f Bool -> f ()
```

Limited form of dependance

```
bindBool :: Selective f => f Bool -> (Bool -> f a) -> f a
bindBool x f = ifS x (f False) (f True)
```

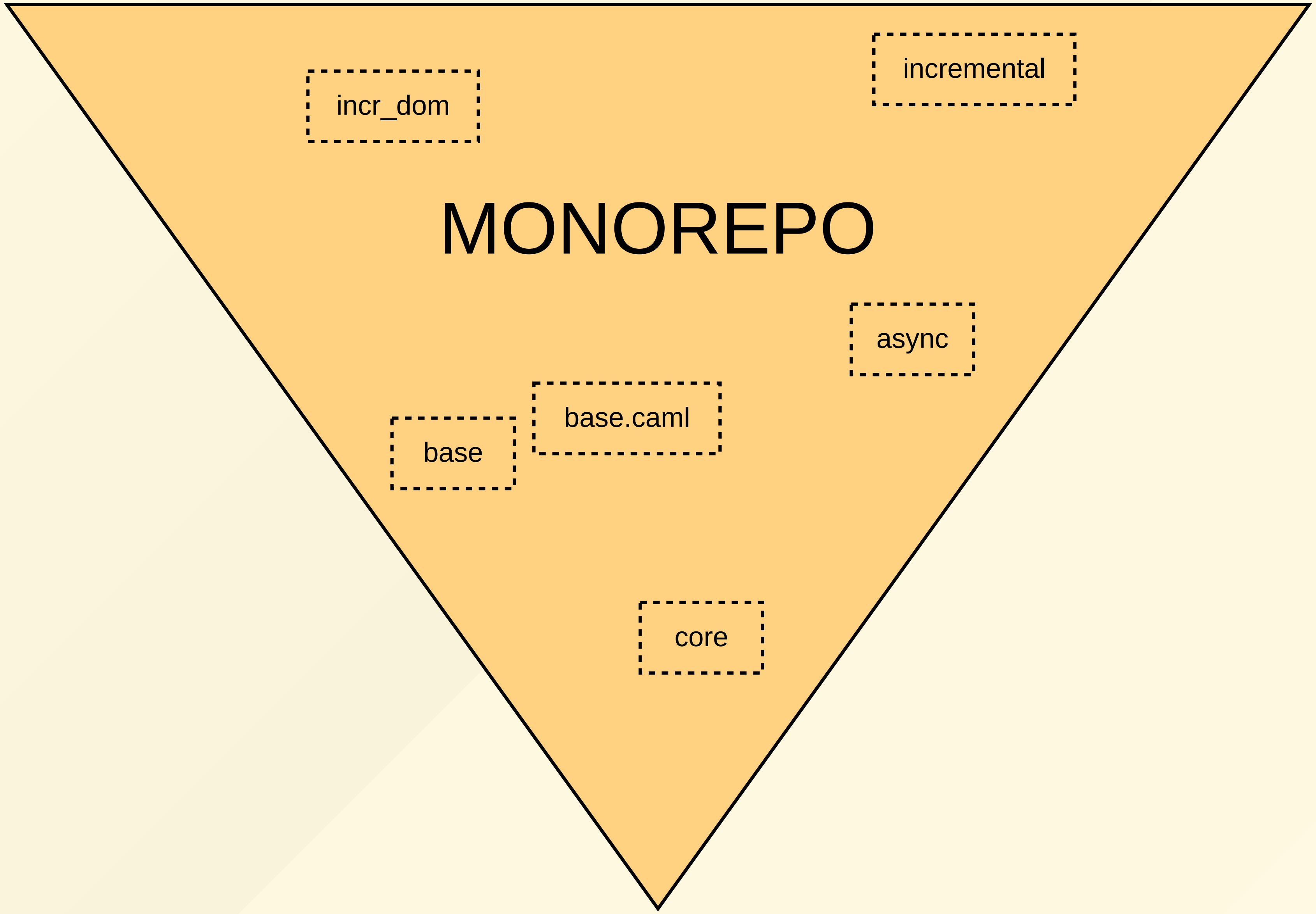
Works with any enumerable type.

Is it really worth it?

github.com/janestreet

- base
- core
- async
- incr_dom
- incrental
- ...

Over 100 packages


incr_dom
incremental
MONOREPO
async
base.caml
base
core

src/dune:

```
(library
 (public_name mylib)
 (libraries re lwt))

(rule (with-stdout-to m.ml (run gen/gen.exe)))
```

src/gen/dune:

```
(executable
 (name gen)
 (libraries ppxlib))
```

Dune's internals

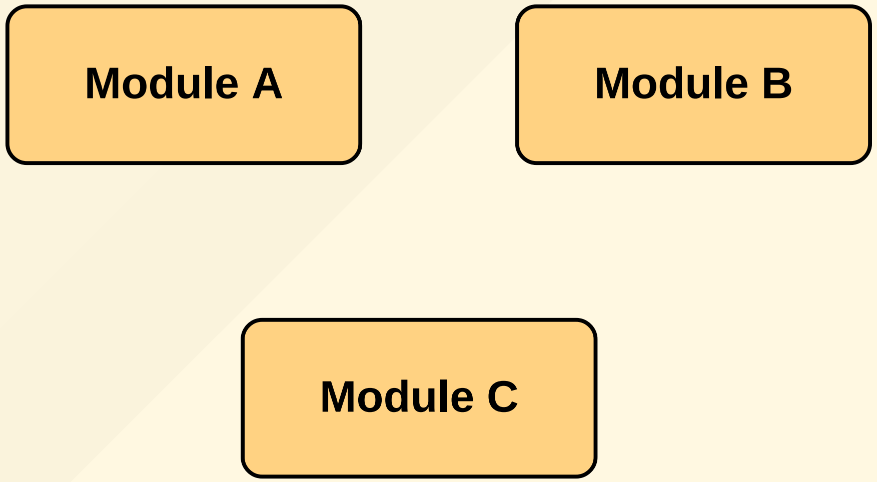
1. Generate rules
2. Run the build

The Build selective

```haskell
-- Action DSL
data Action = Run Path [String] | Chdir Path Action | ...

-- The Build selective
data Build a = Build a [Path]

-- A Build system rule
data Rule = Rule (Build Action) [Path]

-- Read the contents of a file
read :: Path -> Build String

-- Declare a file that the action will read
dep :: Path -> Build ()
dep fn = Build () [fn]
```

OCaml compilation

- modules must be compiled in order
- the `ocamldep` tool computes dependencies



Exercise

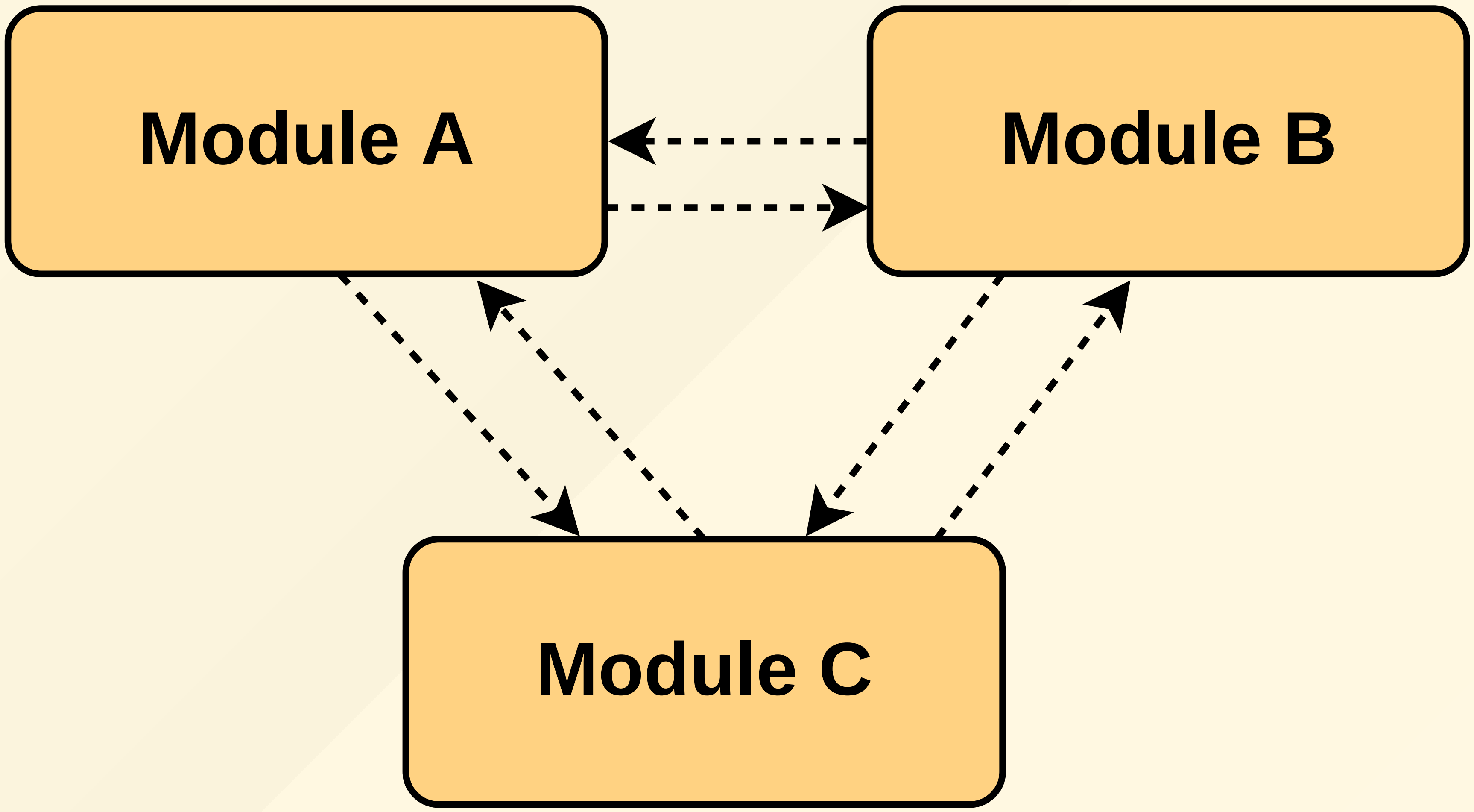
Compute the list of rules to build a library

```
-- Command that compiles a module
ocamlc :: ModuleName -> Action

-- Get the dependency of a module
ocamldep :: ModuleName -> Build [Path]

-- Declare dependencies and compile a module
compileModule :: ModuleName -> [ModuleName] -> Build Actio
compileModule m l = ??
```

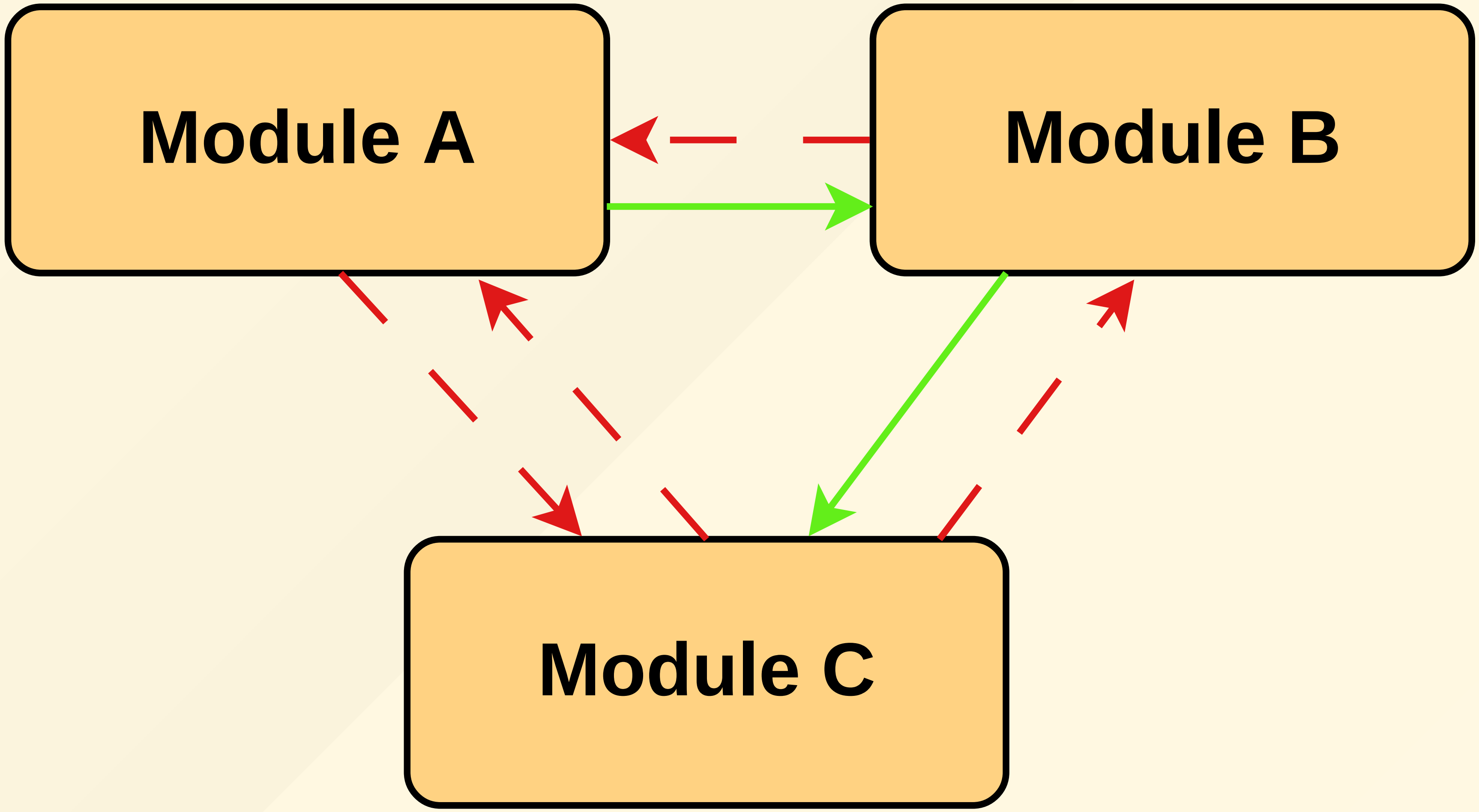
Solution

```
compileModule :: ModuleName -> [ModuleName] -> Build Actio
compileModule m l =
  foldr waitDep (ocamlc m) (filter (<> m) l)
  where
    isDep x = (mem x) <$> (ocamldep m)
    waitDep x acc =
      (ifS (isDep x) (dep x) (return ())) *> acc
```

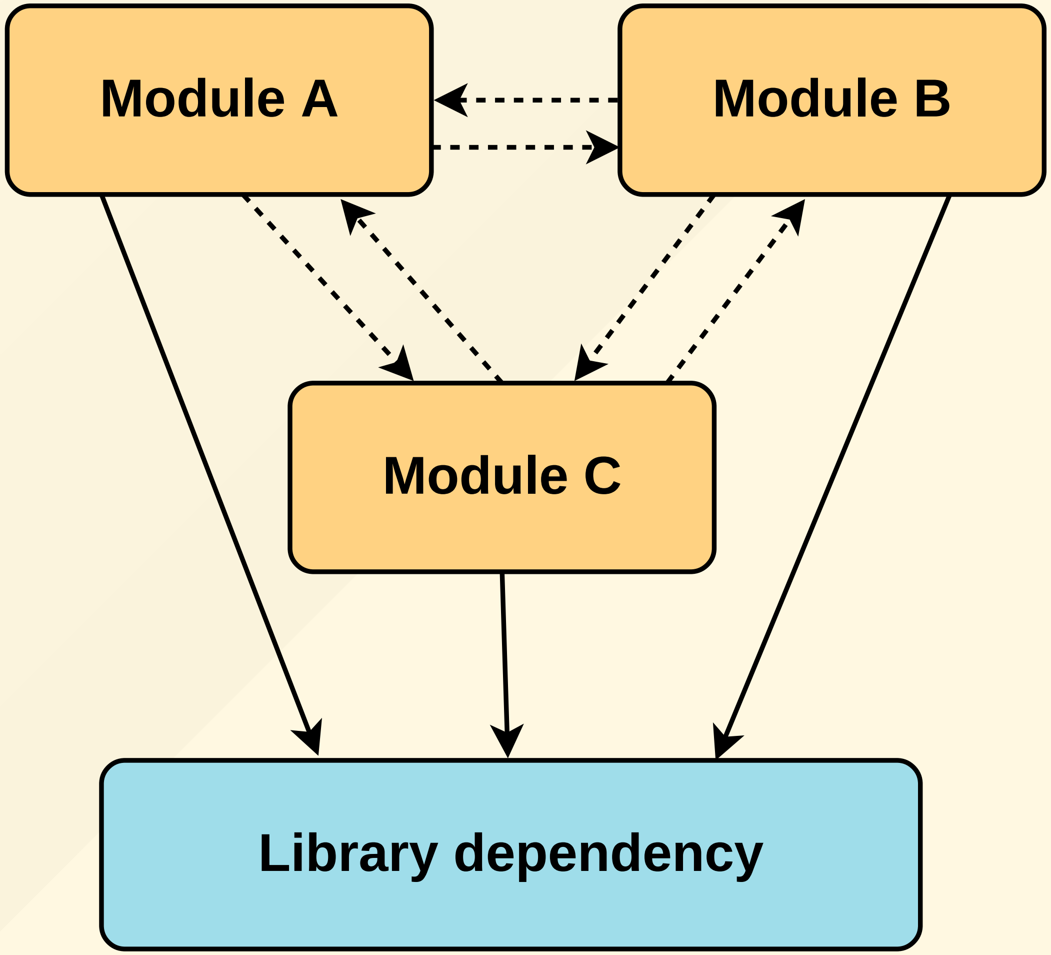
Compiling a library


$$\begin{tikzcd}
\text{Module A} \arrow[rr, dashed, shift right] \arrow[dr, dashed, shift right] & & \text{Module B} \arrow[ll, dashed, shift right] \arrow[dl, dashed, shift right] \\
& \text{Module C} \arrow[ul, dashed, shift right] \arrow[ur, dashed, shift right] &
\end{tikzcd}$$

Compiling a library



Unconditional dependencies



The end

discuss.ocaml.org

opensource.janestreet.com