facebook

GHC determinism

Bartosz Nitka
Software Engineer, Facebook
Sep 24, 2016

What is *determinism*?

Determinism

- Given the same
  - Source
  - Flags
  - Environment
- Produce identical interface files
- Interface files determine ABI

Why is it important?

Build systems with remote caching

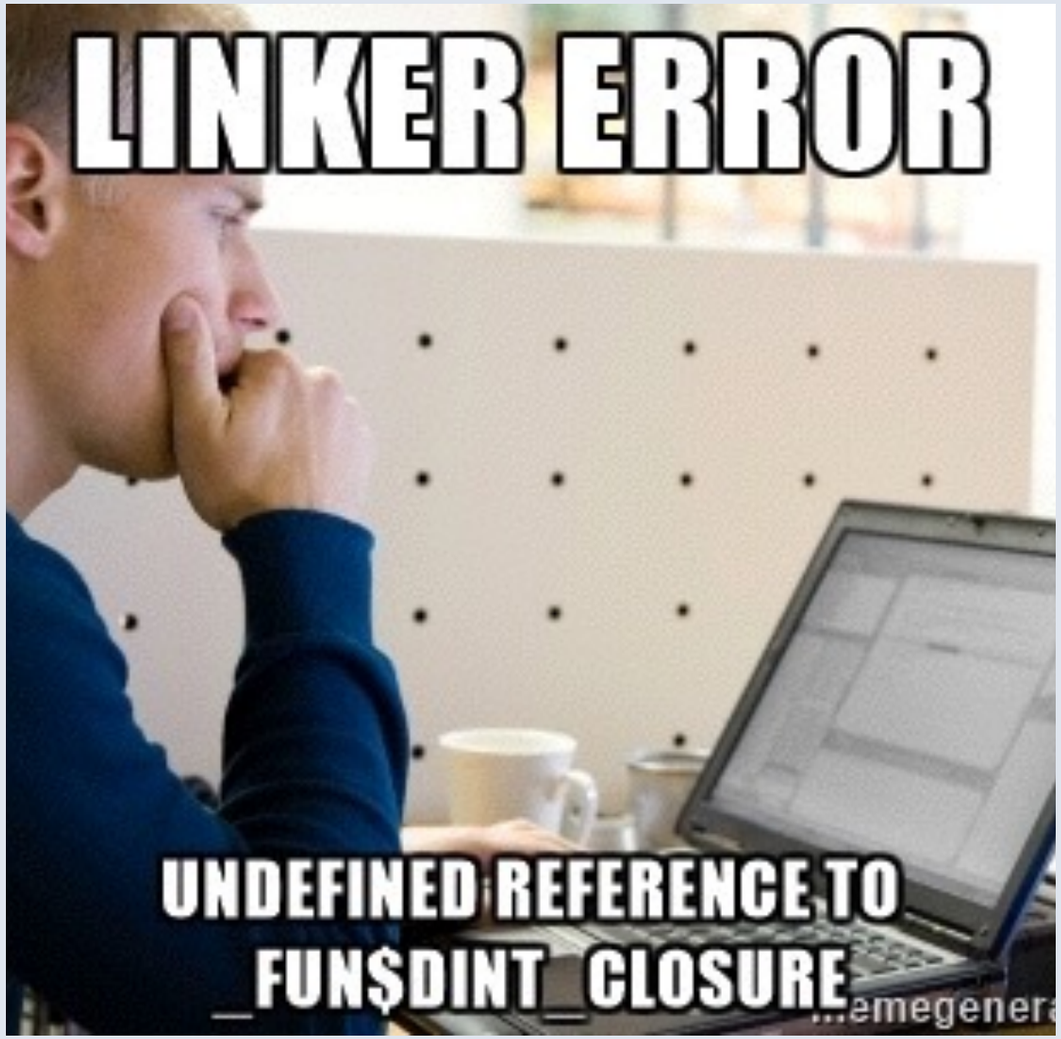
Buck, Bazel...

- Many developers and LOC
- Redundancy
- Idea:
  - Accurate dependency tracking
  - Minimal rebuilds
  - Shared cache

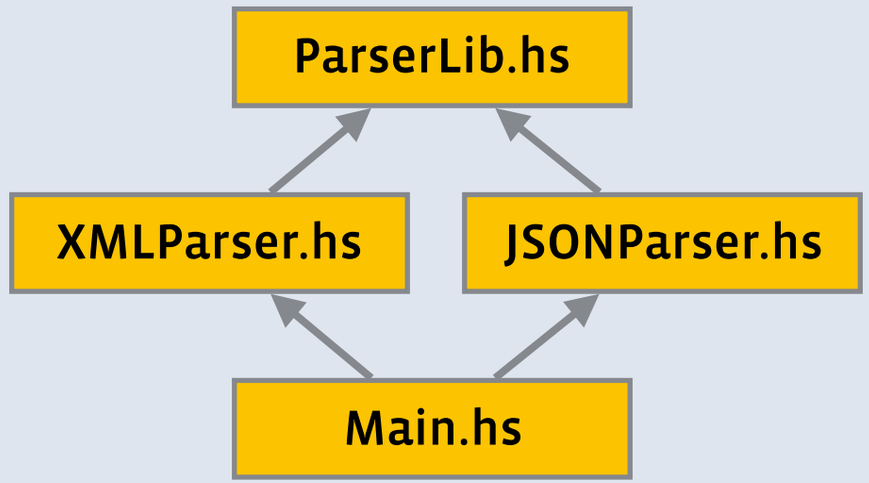
Build systems with remote caching

Buck, Bazel...

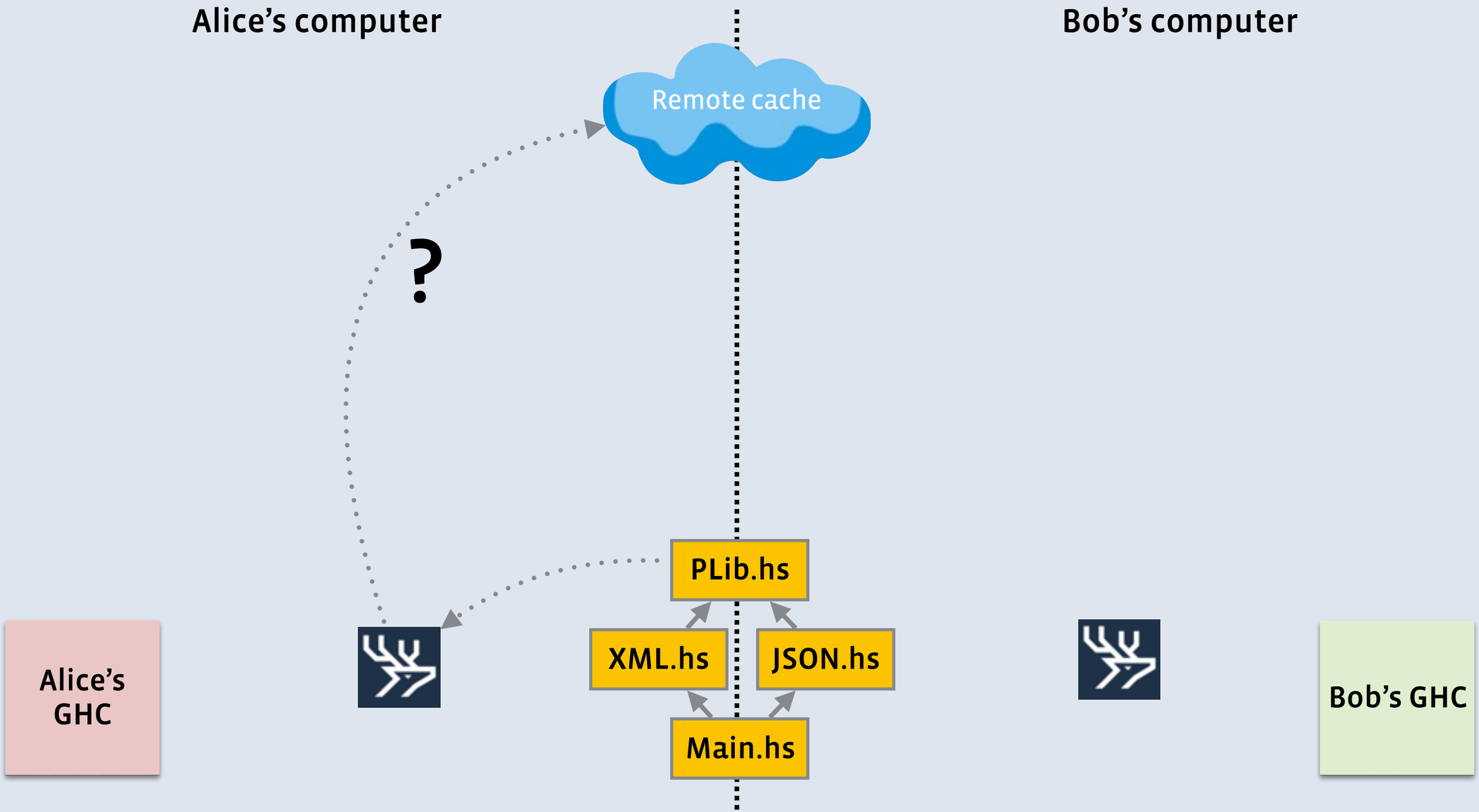
- Unfortunately things go horribly wrong with non-deterministic GHC



What happens if we try?



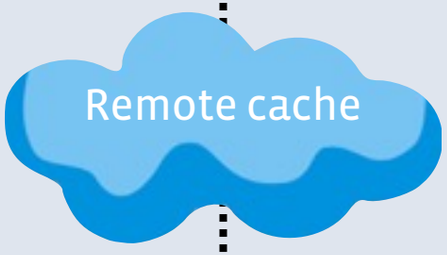
What happens if we try?



What happens if we try?

Alice's computer

Bob's computer



404

PLib.hs

XML.hs

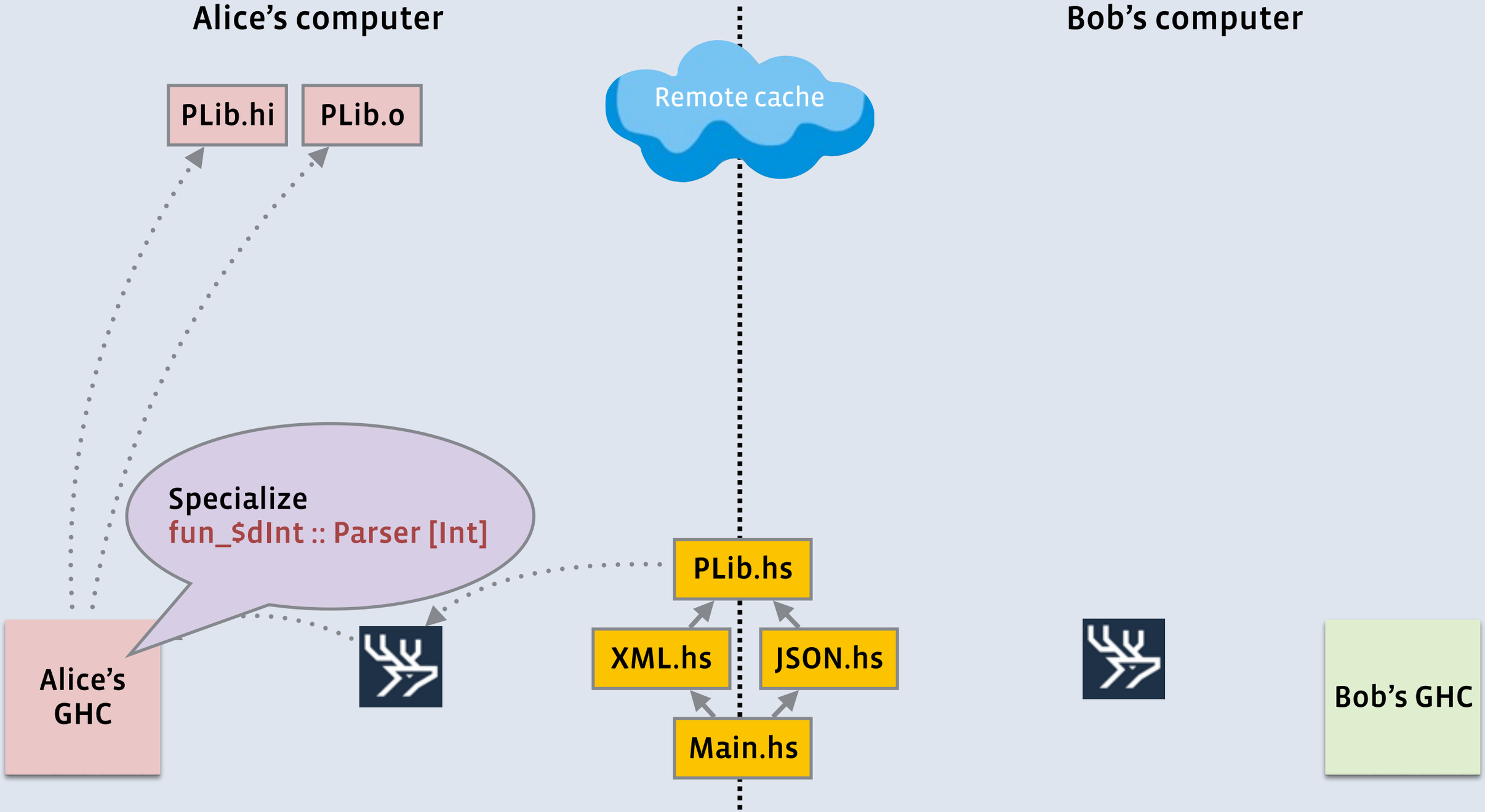
JSON.hs

Main.hs

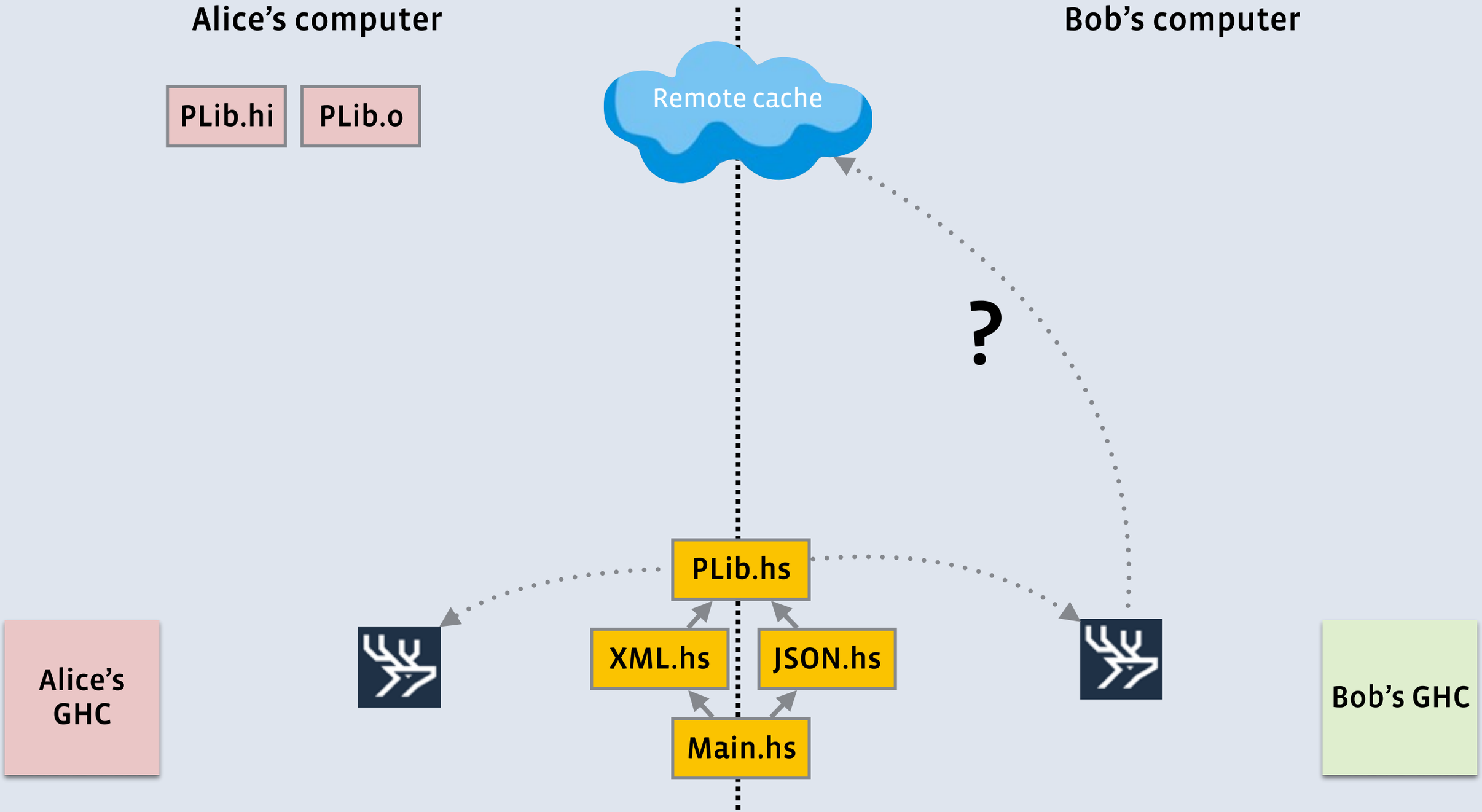
Alice's GHC

Bob's GHC

What happens if we try?



What happens if we try?



What happens if we try?

Alice's computer

Bob's computer

PLib.hi

PLib.o

Remote cache

404

PLib.hs

XML.hs

JSON.hs

Main.hs

Alice's GHC

Bob's GHC

What happens if we try?

Alice's computer

PLib.hi

PLib.o

Remote cache

Bob's computer

PLib.hi

PLib.o

No specialization

PLib.hs

XML.hs

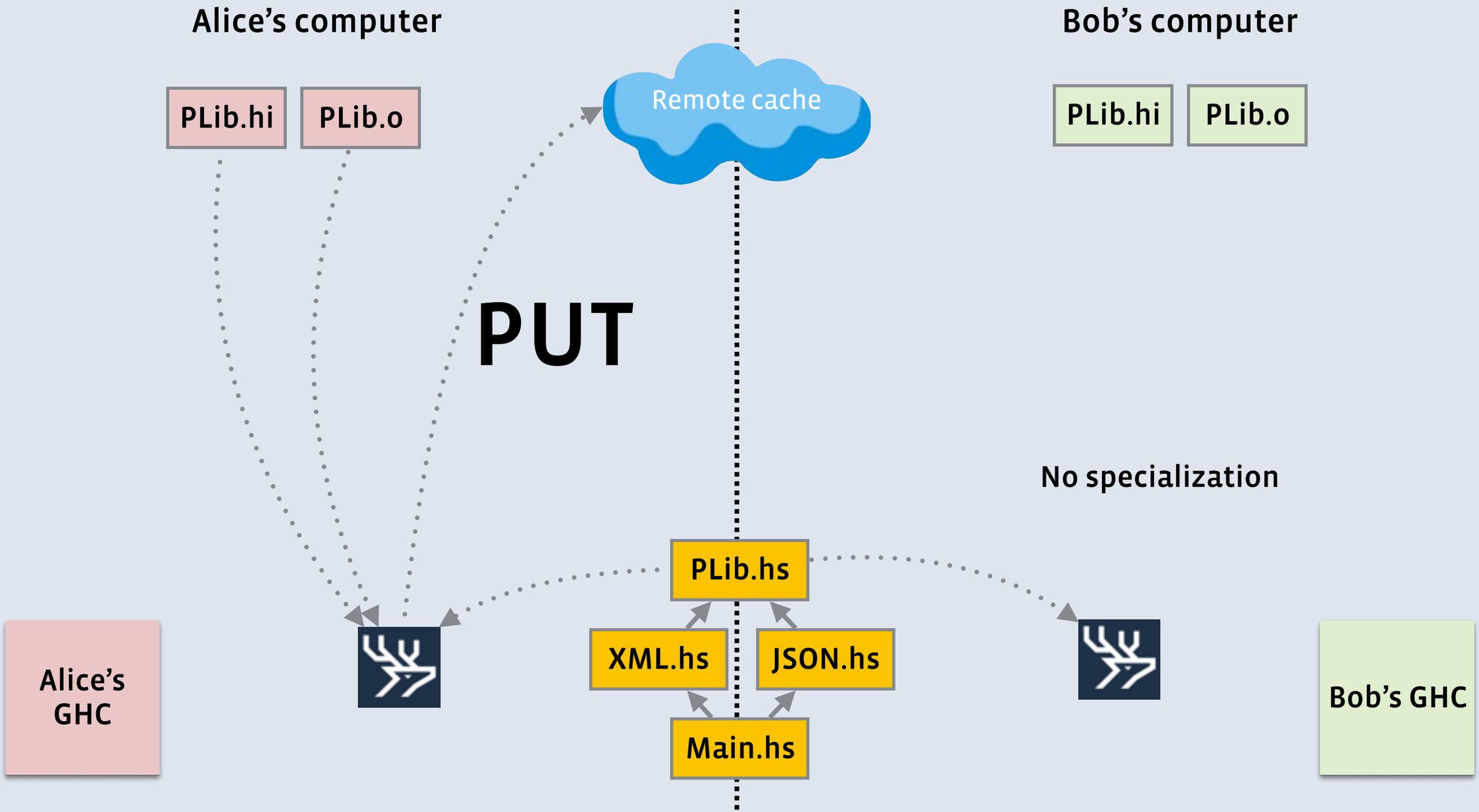
JSON.hs

Main.hs

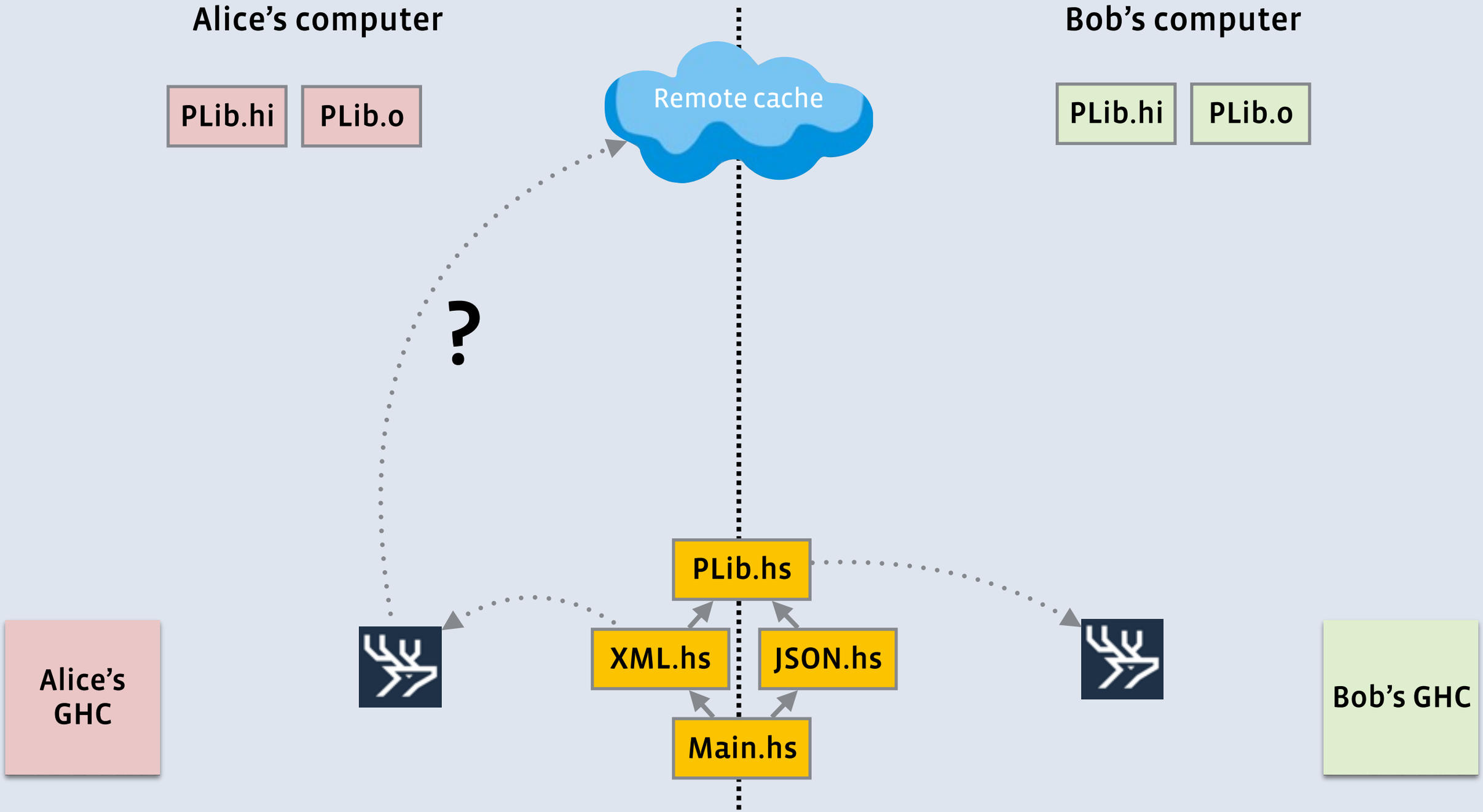
Alice's GHC

Bob's GHC

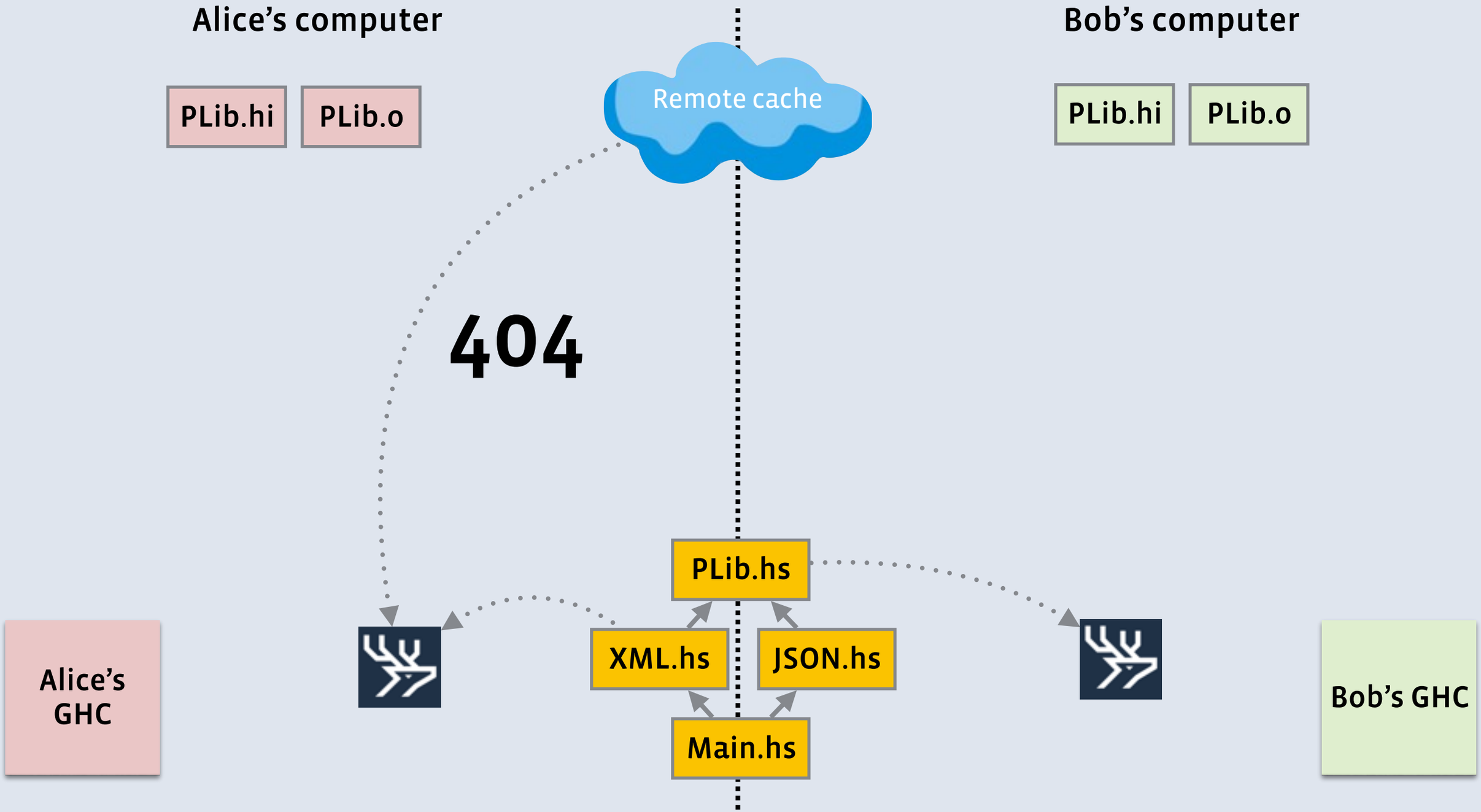
What happens if we try?



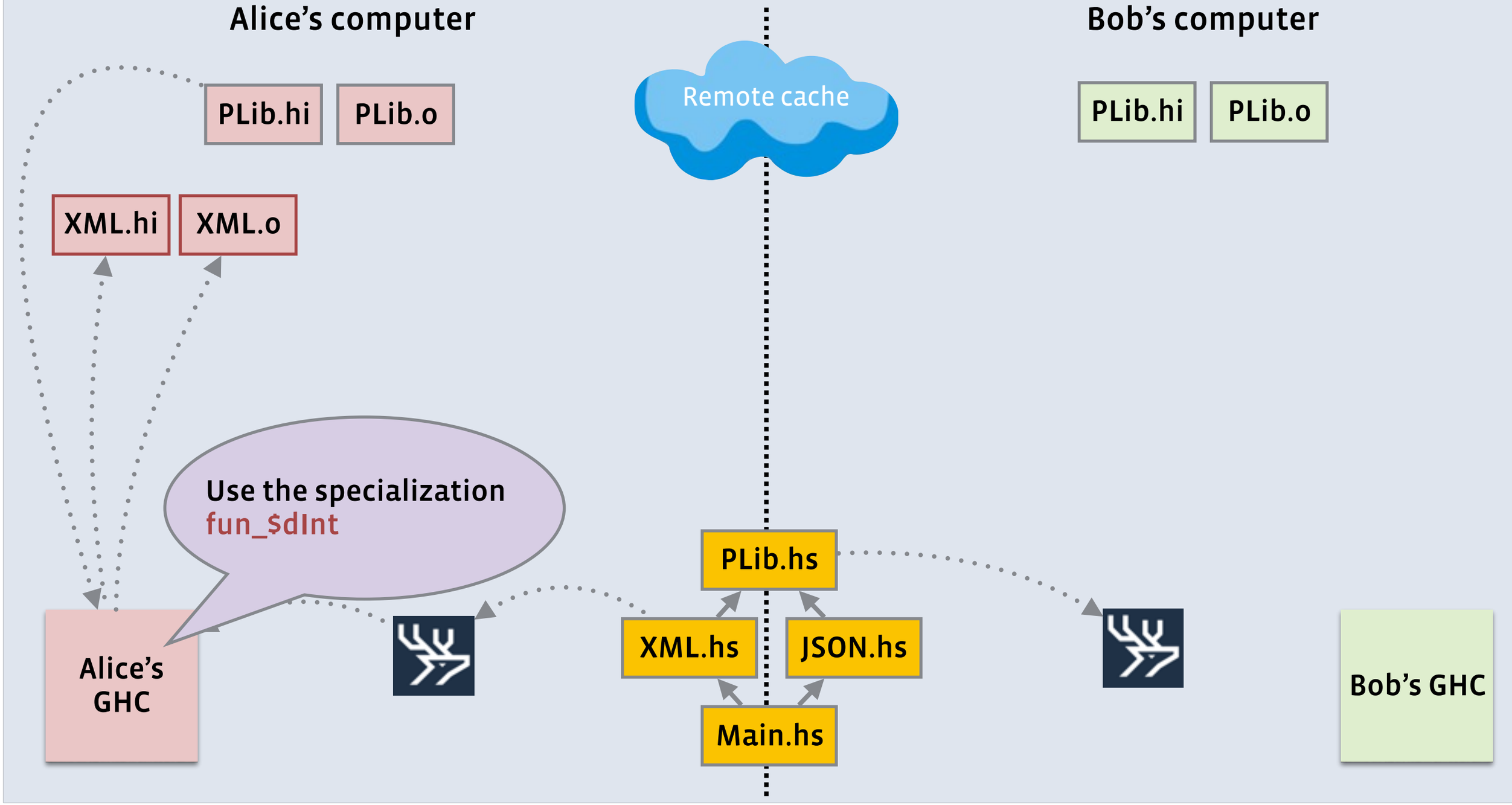
What happens if we try?



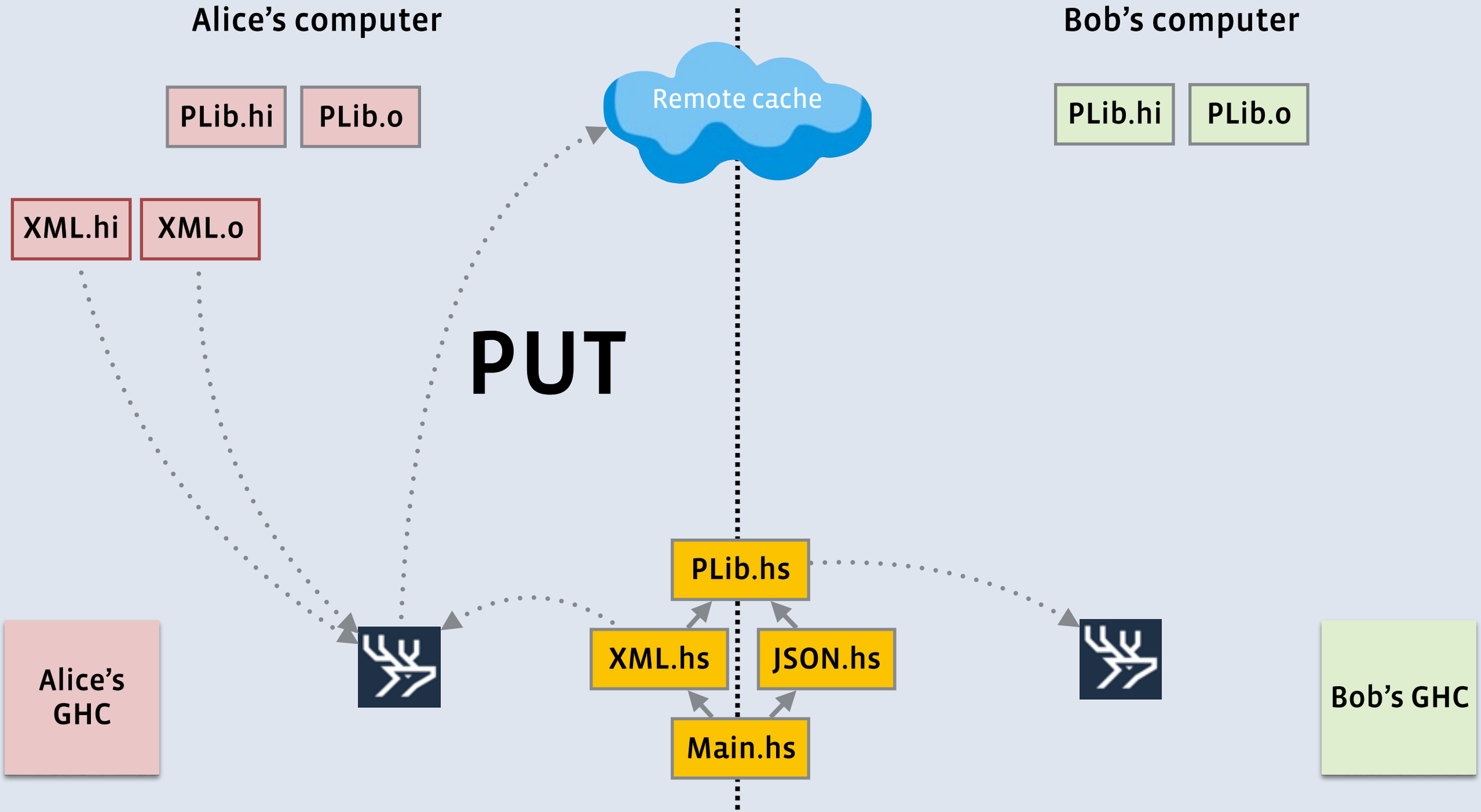
What happens if we try?



What happens if we try?



What happens if we try?



What happens if we try?

Alice's computer

Bob's computer

Remote cache

PLib.hi

PLib.o

XML.hi

XML.o

PLib.hi

PLib.o

Alice's GHC

PLib.hs

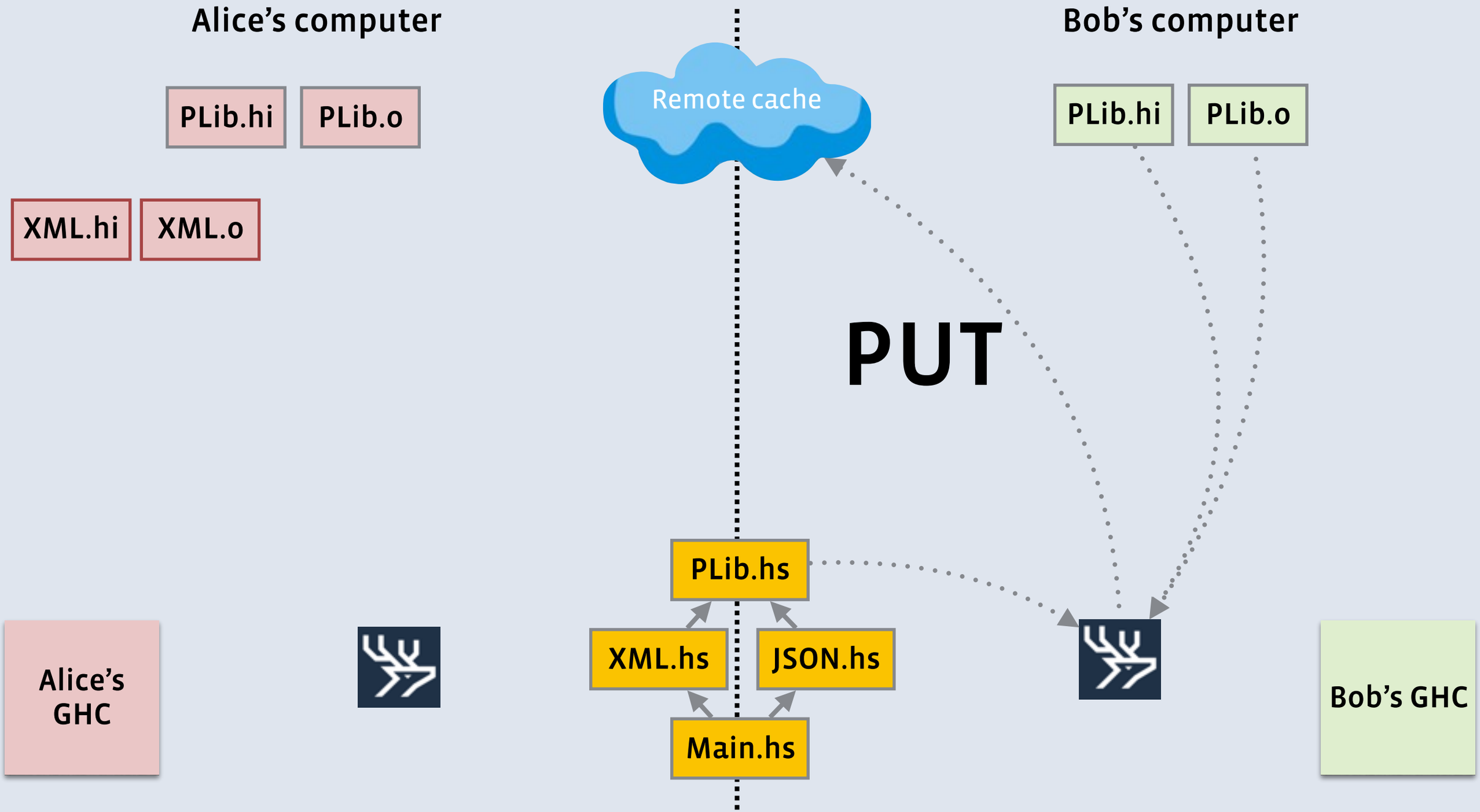
XML.hs

JSON.hs

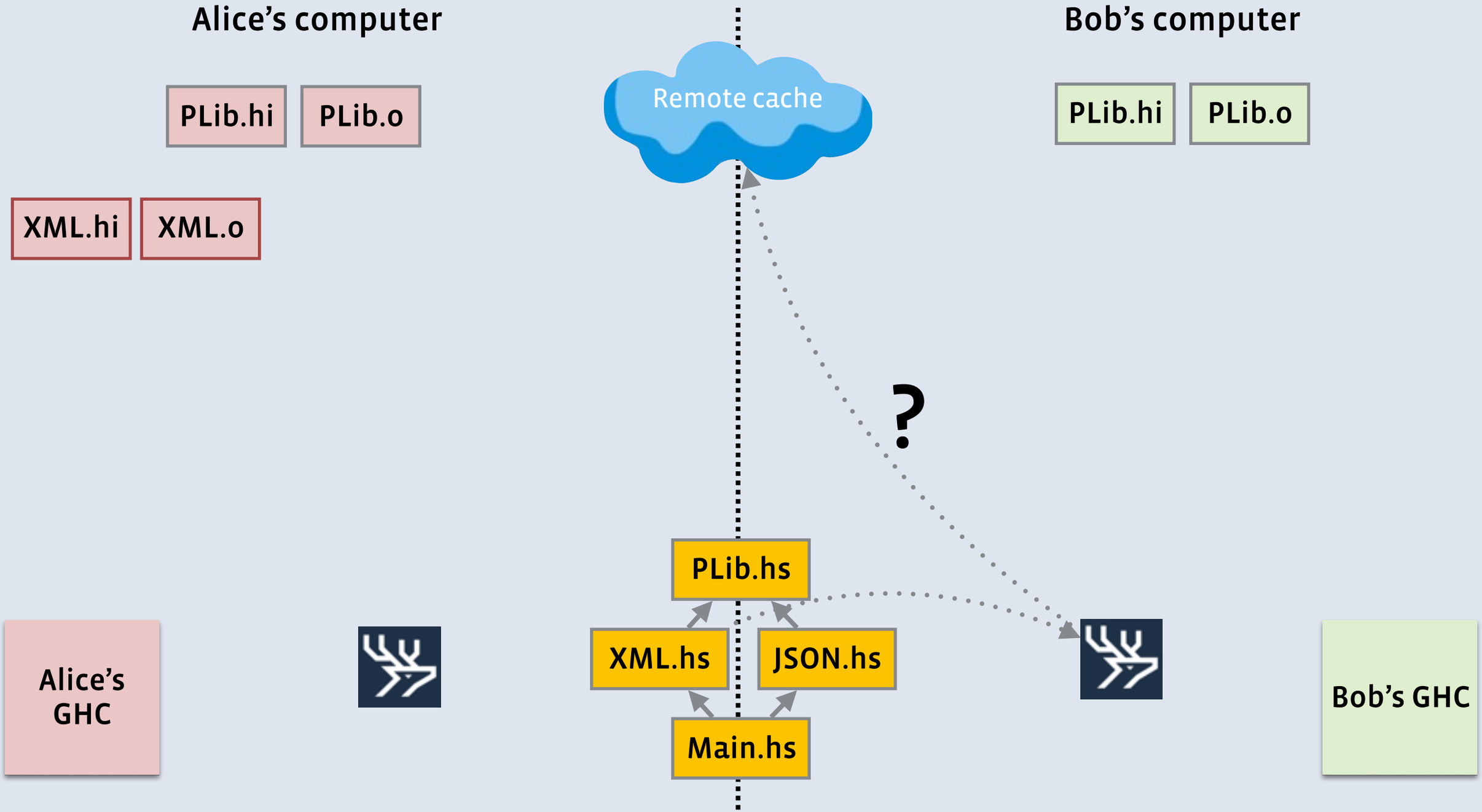
Main.hs

Bob's GHC

What happens if we try?



What happens if we try?



What happens if we try?

Alice's computer

Bob's computer

Remote cache

PLib.hi PLib.o

XML.hi XML.o

PLib.hi PLib.o

XML.hi XML.o

HIT

PLib.hs

XML.hs JSON.hs

Main.hs

Alice's GHC

Bob's GHC

What happens if we try?

Alice's computer

PLib.hi

PLib.o

XML.hi

XML.o

Remote cache

Bob's computer

PLib.hi

PLib.o

XML.hi

XML.o

JSON.h

JSON.o

Main.h

Main.o

PLib.hs

XML.hs

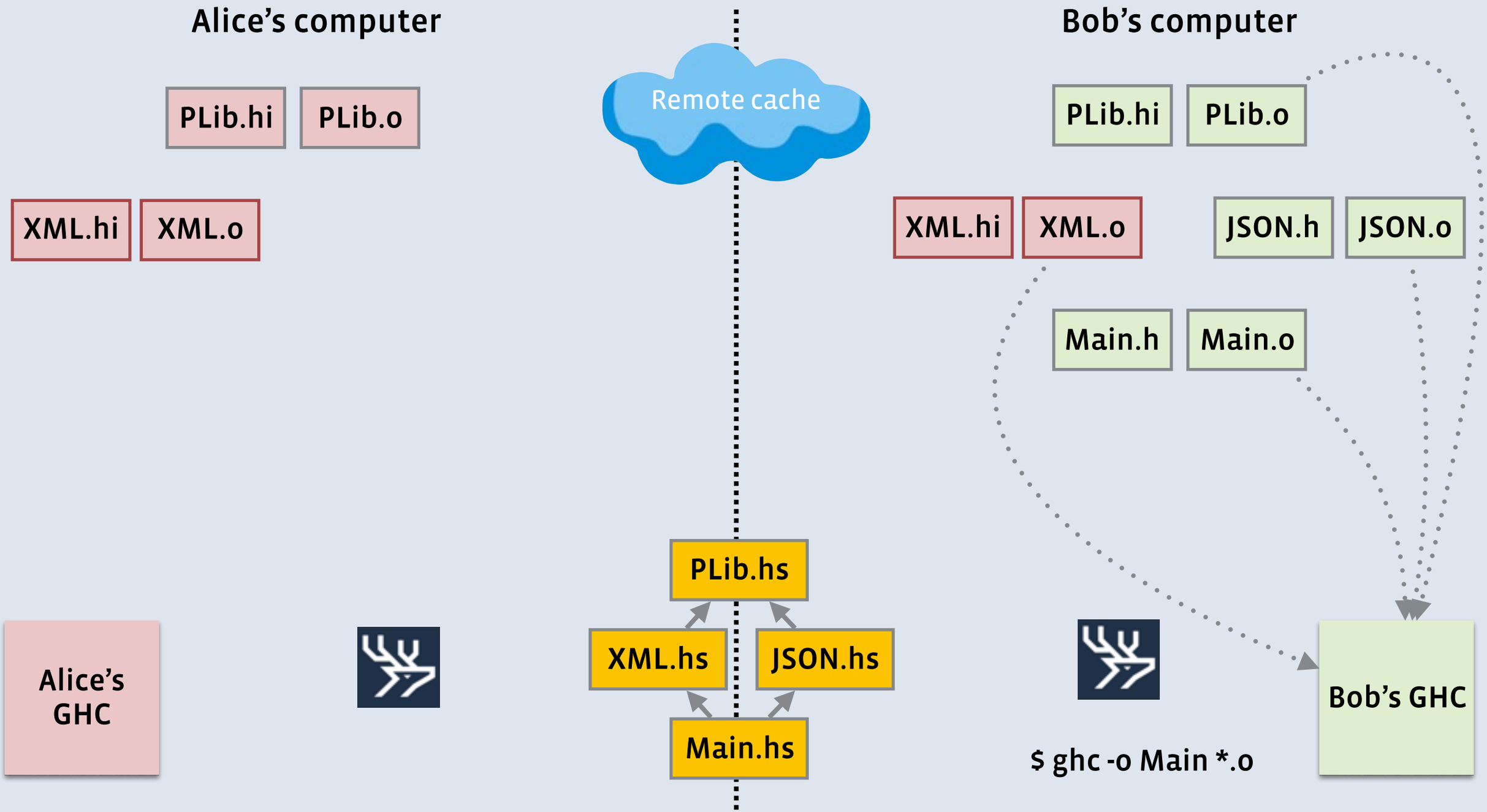
JSON.hs

Main.hs

Alice's GHC

Bob's GHC

What happens if we try?



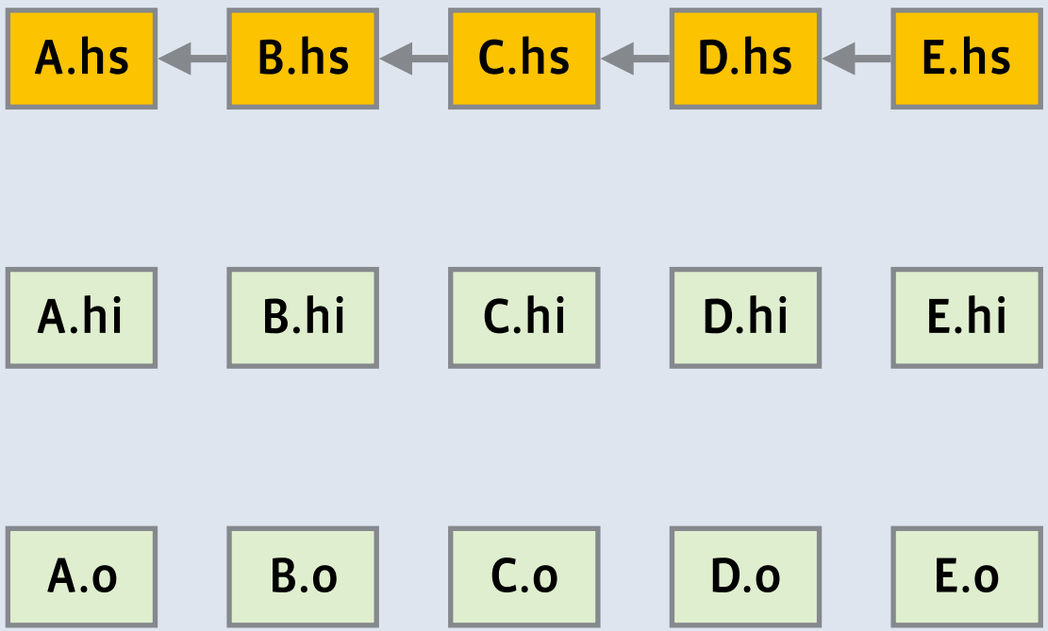
What happens if we try?

```
$ ghc -o Main *.o
Undefined symbols for architecture x86_64:
  "_ParserLib_commaSeparatedzudInt_closure", referenced from:
      _XMLParser_c_closure in XMLParser.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

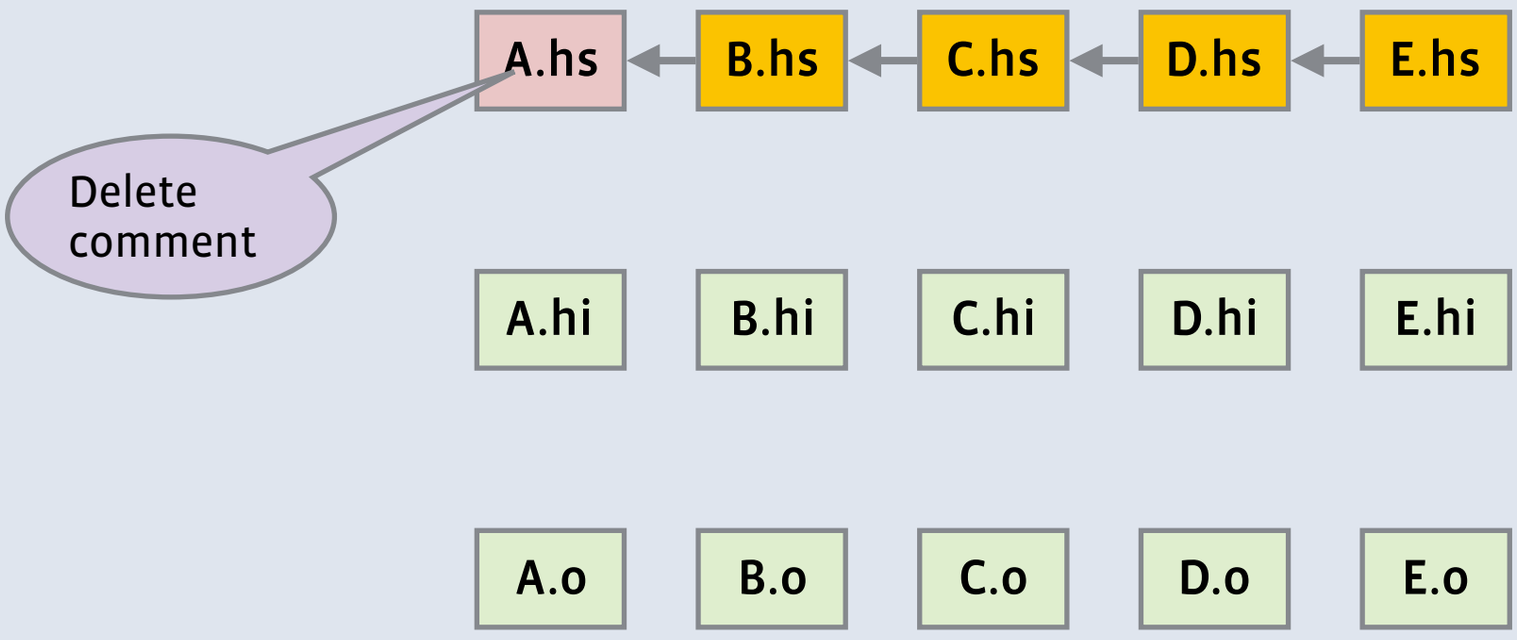
Incremental builds

- Clean build can take a long time
- We want short feedback cycles
- Solution:
  - Minimal recompile
  - GHC has a recompilation check

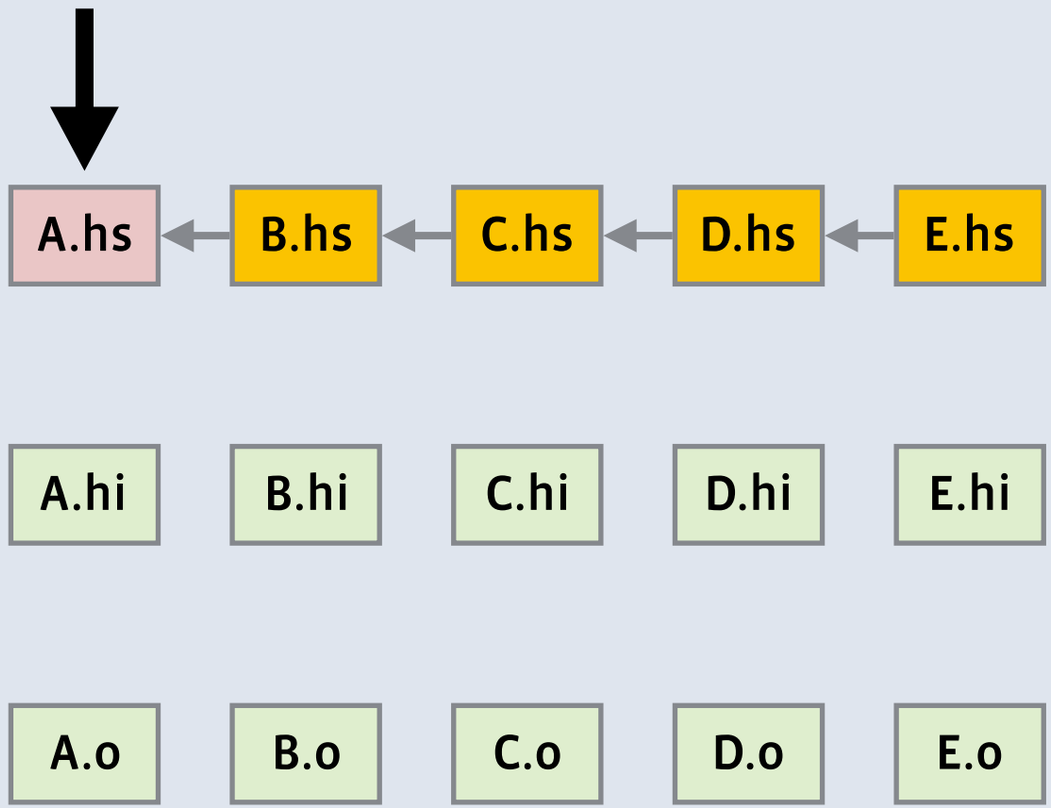
Incremental builds



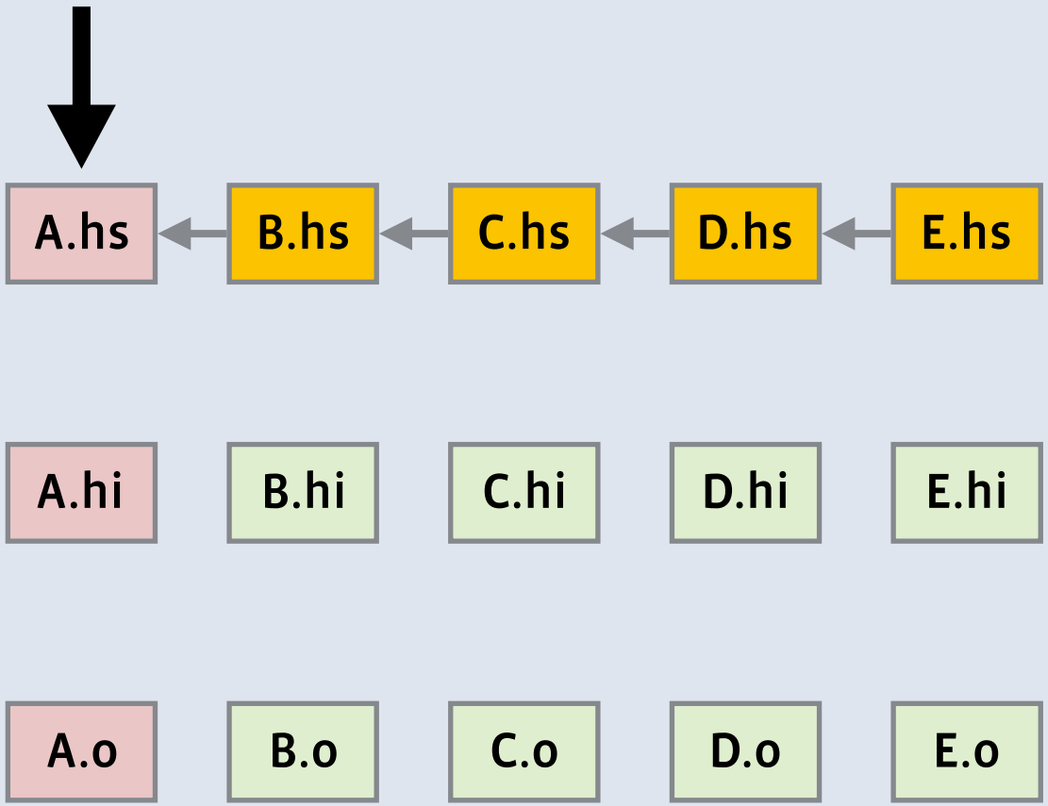
Incremental builds



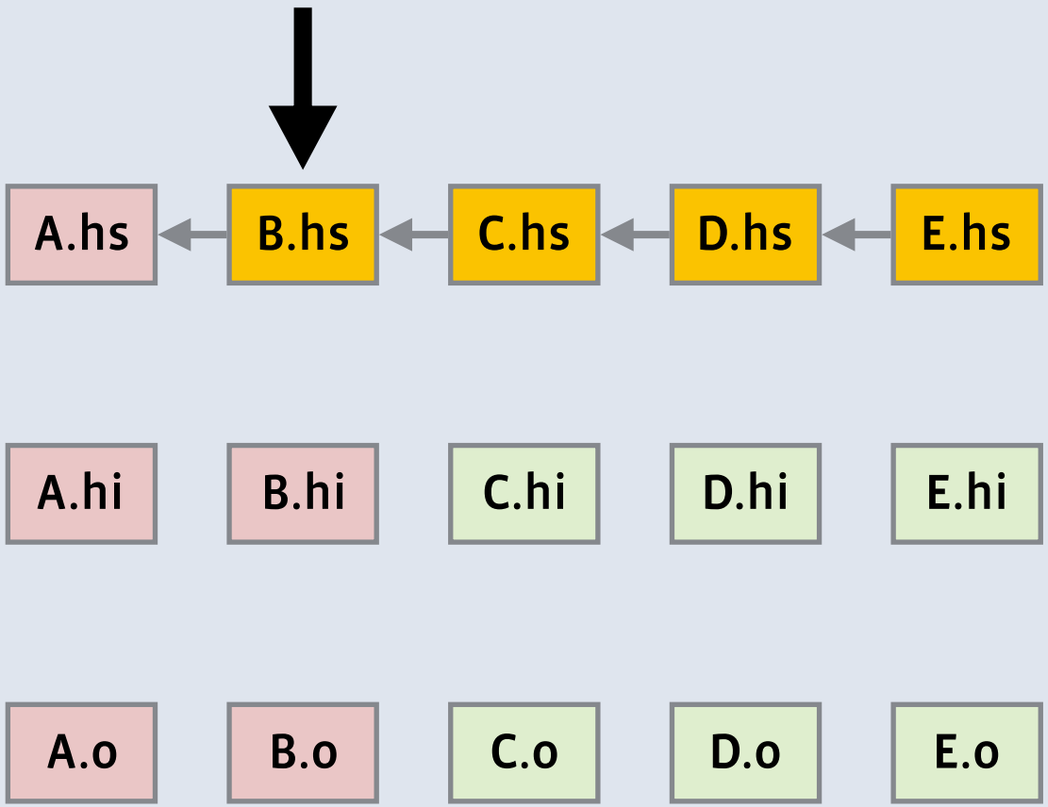
Incremental builds



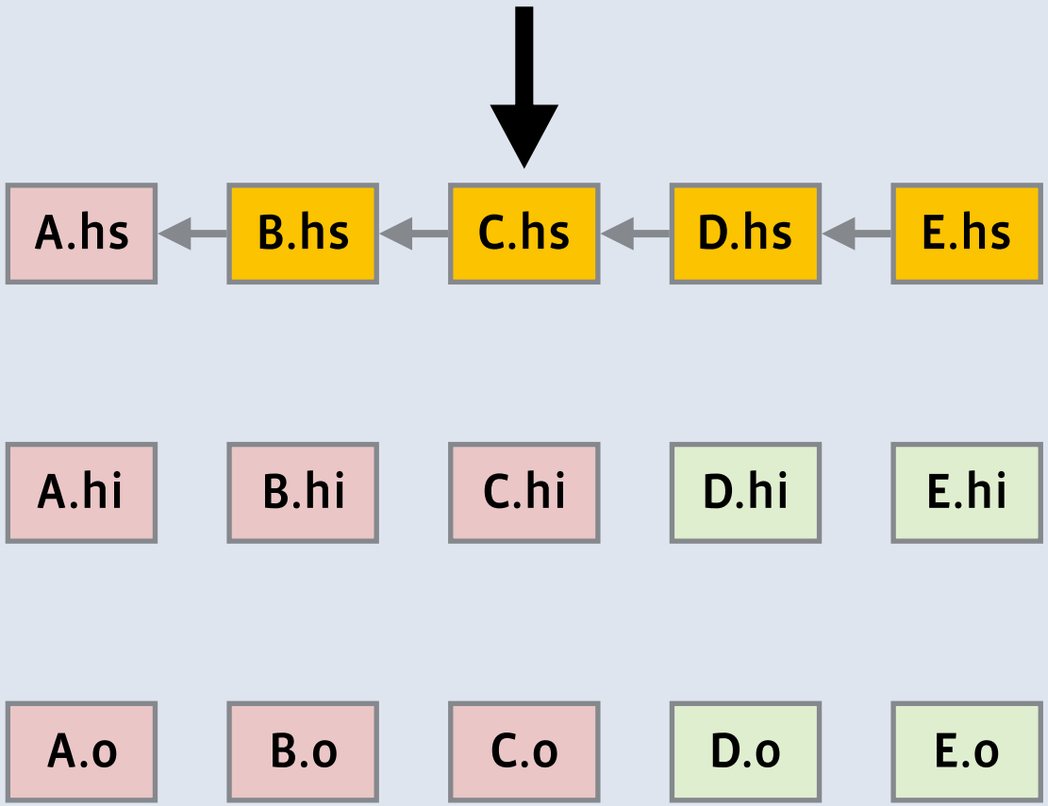
Incremental builds



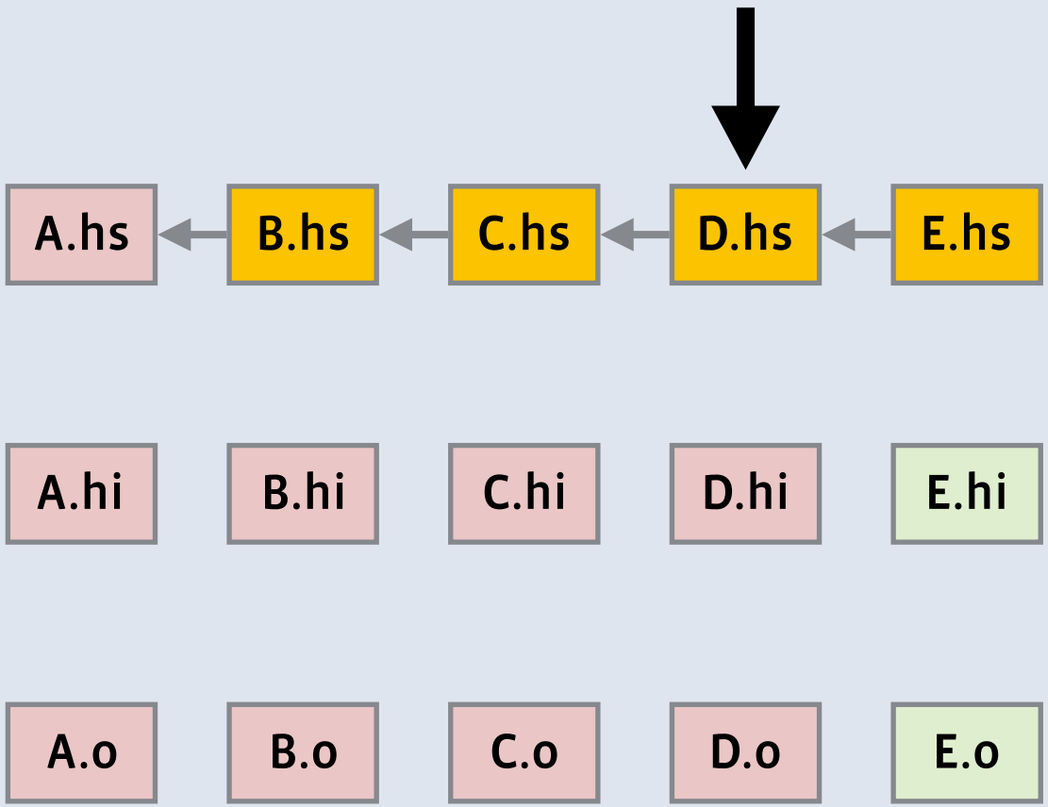
Incremental builds



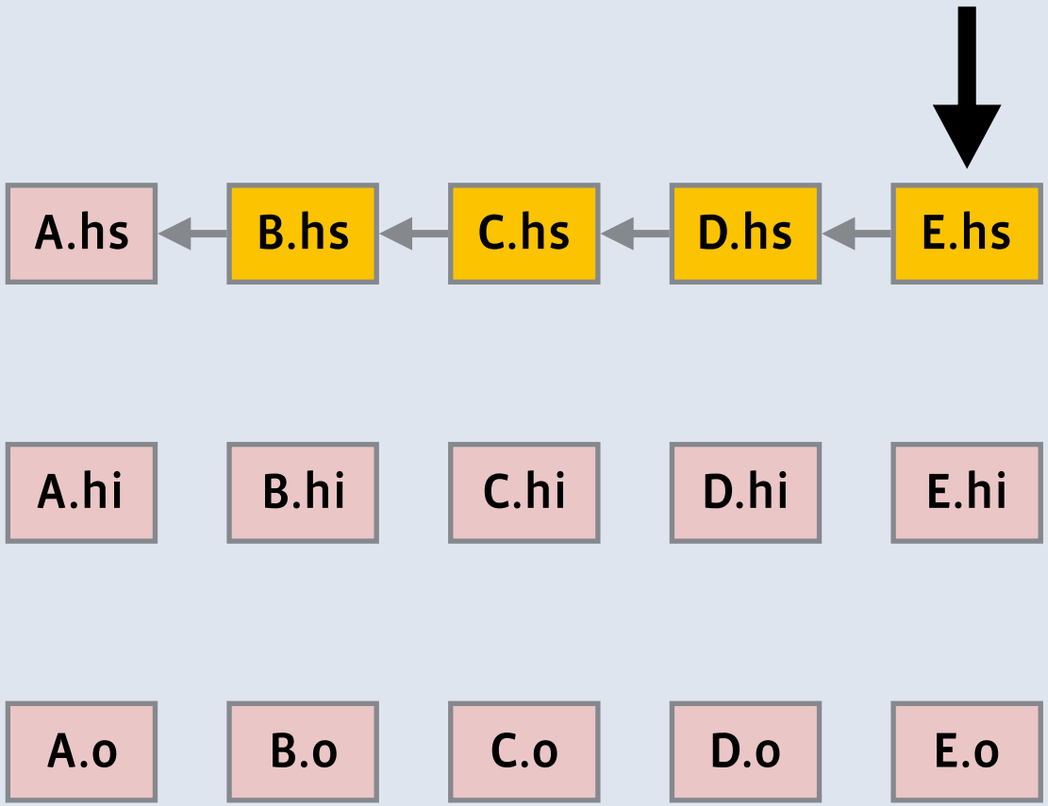
Incremental builds



Incremental builds



Incremental builds



Other

- Binary distributions (Debian, Nix) can recompile GHC without recompiling the world
- hot-swapping
- bit-for-bit determinism

What makes it non-deterministic?

Determinism

- #4012 opened 24 Apr 2010
  - > 200 comments
  - “Are all tickets matching the regex #[4012]+ related” — Joachim Breitner

Uniques

```
type Unique = Int
data Var = Var String Unique

instance Eq Var where
  (Var _ u1) == (Var _ u2) = u1 == u2

instance Ord Var where
  (Var _ u1) `compare` (Var _ u2) = u1 `compare` u2

-- Useful optimization
  -- Fast maps, sets, substitutions…
```

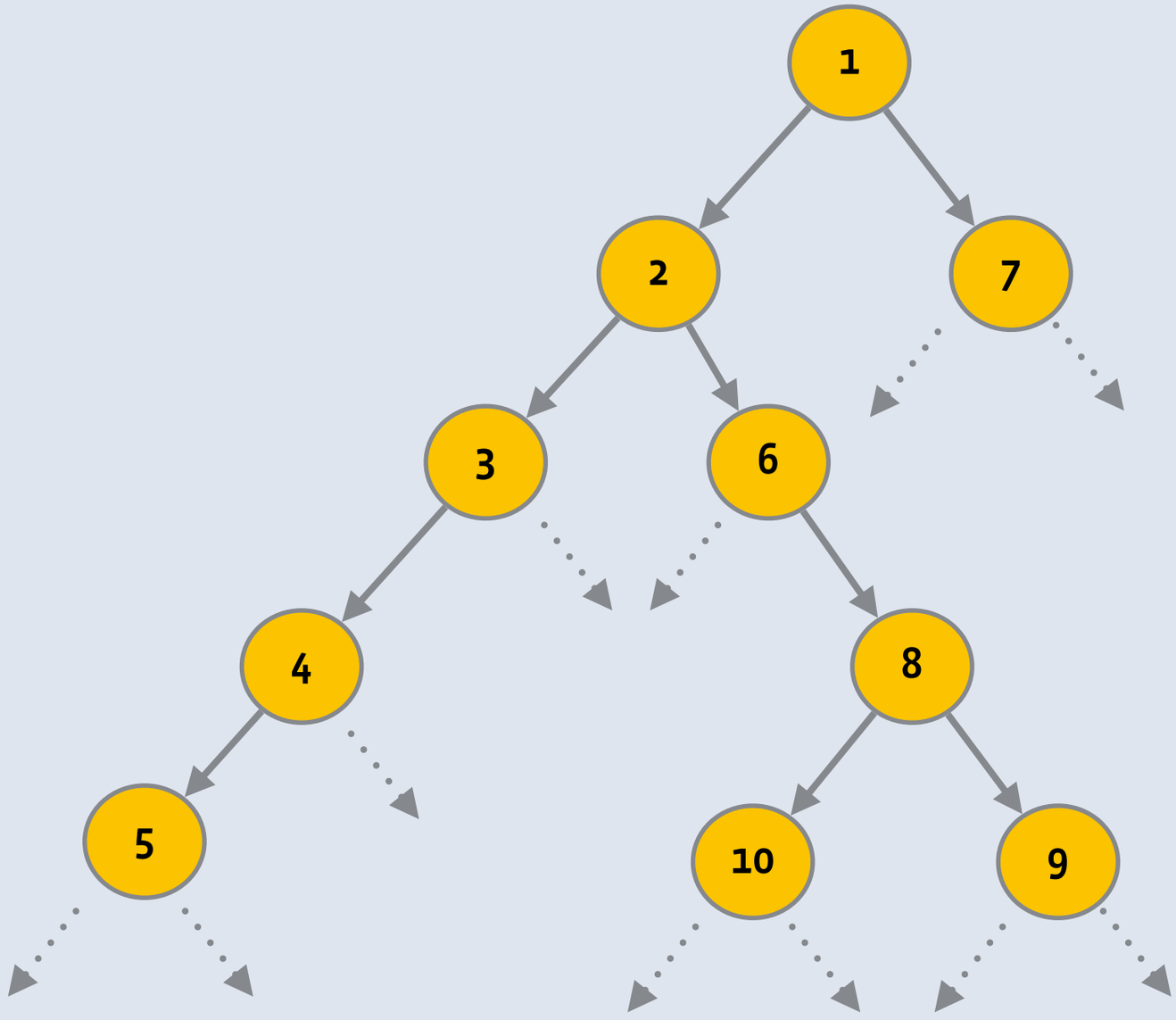
Uniques, how do I get one?

```
data UniqSupply =
  MkSplitUniqSupply Int UniqSupply UniqSupply

-- Lazily generated tree with each node incrementing a
global counter when forced.
-- Really just a nice way of pulling it out of IO.

takeUniqFromSupply :: UniqSupply -> (Unique,
UniqSupply)
takeUniqFromSupply (MkSplitUniqSupply v us _) = (v,
us)
```

UniqSupply



Uniques

- To get determinism, either:
  - stable across compilations
  - not affecting the ABI

Option 1: stable Uniques

Unique nondeterminism

- Global UniqueSupply

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files
- Type checking interface files is lazy

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files
- Type checking interface files is lazy
  - The interface file for the file currently being compiled may or may not exist

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files
- Type checking interface files is lazy
  - The interface file for the file currently being compiled may or may not exist
    - ghc --make: any subset of interface files

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files
- Type checking interface files is lazy
  - The interface file for the file currently being compiled may or may not exist
    - ghc --make: any subset of interface files
- we want to parallelize with multiple threads

Unique nondeterminism

- Global UniqueSupply
- Need to allocate Uniques when type-checking interface files
- Type checking interface files is lazy
  - The interface file for the file currently being compiled may or may not exist
    - ghc --make: any subset of interface files
- we want to parallelize with multiple threads
- and still get the same Uniques!

Option 2: ABI doesn't depend on Uniques

Nondeterministic uniques

```
showSquared a = show (a * a)
-- What's the inferred type?
```

Nondeterministic uniques

```
showSquared a = show (a * a)
-- What's the inferred type?
```

```
showSquared
  :: (Num a, Show a)
  => a -> String
```

```
showSquared
  :: (Show a, Num a)
  => a -> String
```

Nondeterministic uniques

```
showSquared a = show (a * a)
-- What's the inferred type?
```

```
showSquared
  :: (Num a, Show a) => a -> String
```

```
showSquared
  :: (Show a, Num a) => a -> String
```

They are operationally different!

Nondeterministic uniques, why?

```
showSquared a = show (a * a)
-- What's the inferred type?
```

```
showSquared
  :: (Num a, Show a) => a -> String
```

```
showSquared
  :: (Show a, Num a) => a -> String
```

```
constraints :: VarSet
```

```
constraints :: VarSet
```

Nondeterministic uniques, why?

```
showSquared a = show (a * a)
-- What's the inferred type?
```

```
showSquared
  :: (Num a, Show a) => a -> String
```

```
showSquared
  :: (Show a, Num a) => a -> String
```

constraints :: VarSet

```
makeCtx
  :: VarSet -> [Var]
makeCtx constraints =
  [Num, Show]
```

constraints :: VarSet

```
makeCtx
  :: VarSet -> [Var]
makeCtx constraints =
  [Show, Num]
```

Nondeterministic uniques, why?

```
showSquared a = show (a * a)
-- What's the inferred type?
```

```
showSquared
  :: (Num a, Show a) => a -> String
```

```
constraints :: VarSet

makeCtx :: VarSet -> [Var]

makeCtx constraints =
  [ Var "Num" 1
  , Var "Show" 2 ]
```

```
showSquared
  :: (Show a, Num a) => a -> String
```

```
constraints :: VarSet

makeCtx :: VarSet -> [Var]

makeCtx constraints =
  [ Var "Show" 1
  , Var "Num" 2 ]
```

The fix

- No easy fix
  - >100 commits

Other examples

- SCCs
- Free variables
  - let floating
  - worker-wrapper
    - Arrow, ApplicativeDo, RecursiveDo desugaring
- Anchoring of instances, rules and type families
- Order of record fields
- …

How do we keep it fixed?

The long-term fix

- Combination of types and convention

The long-term fix

- Types

```
216 instance Ord Unique where
217     a  < b = ltUnique a b
218     a <= b = leUnique a b
219     a  > b = not (leUnique a b)
220     a >= b = not (ltUnique a b)
221     compare a b = nonDetCmpUnique a b
222
```

```
112 foldUniqSet = foldUFM
```

```
80 nameSetElems     = uniqSetToList
```

```
89 nameSetAny :: (Name -> Bool) -> NameSet -> Bool
90 nameSetAny = uniqSetAny
91
92 nameSetAll :: (Name -> Bool) -> NameSet -> Bool
93 nameSetAll = uniqSetAll
94
```

The long-term fix

- Convention

```
1233  zonkTyCoVarsAndFV :: TyCoVarSet -> TcM TyCoVarSet
1234  zonkTyCoVarsAndFV tycovars =
1235    tyCoVarsOfTypes <$> mapM zonkTyCoVar (nonDetEltsUFM tycovars)
1236    -- It's OK to use nonDetEltsUFM here because we immediately forget about
1237    -- the ordering by turning it into a nondeterministic set and the order
1238    -- of zonking doesn't matter for determinism.
```

The long-term fix

- If all else fails, test suite
- Testing
  - -dunique-increment=-1
  - that's all the regression tests
- Mind-map almost done

What's coming to GHC 8.0.2?

GHC 8.0.2

- Observable determinism
  - GHC
  - stackage - 1200 packages
- Open a Trac ticket!
  - DIY non-determinism in TH

Thanks

- Simon Marlow
- Simon Peyton Jones
- Gemma Silvers
- Jon Coens
- Ben Gamari
- Thomas Miedema
- Herbert Valerio Riedel

Questions?

Other examples

- Uniques in error messages for unreachable code
- Uniques used to get a unique name for automatic deriving
- Nondeterministically abstracted RULES for specialization
- Unique order leaking to TemplateHaskell
- …