Component adaptation and assembly using interface relations

Or: a tool called Cake

.

Stephen Kell

Stephen.Kell@cl.cam.ac.uk

Computer Laboratory

University of Cambridge

There's something about software...

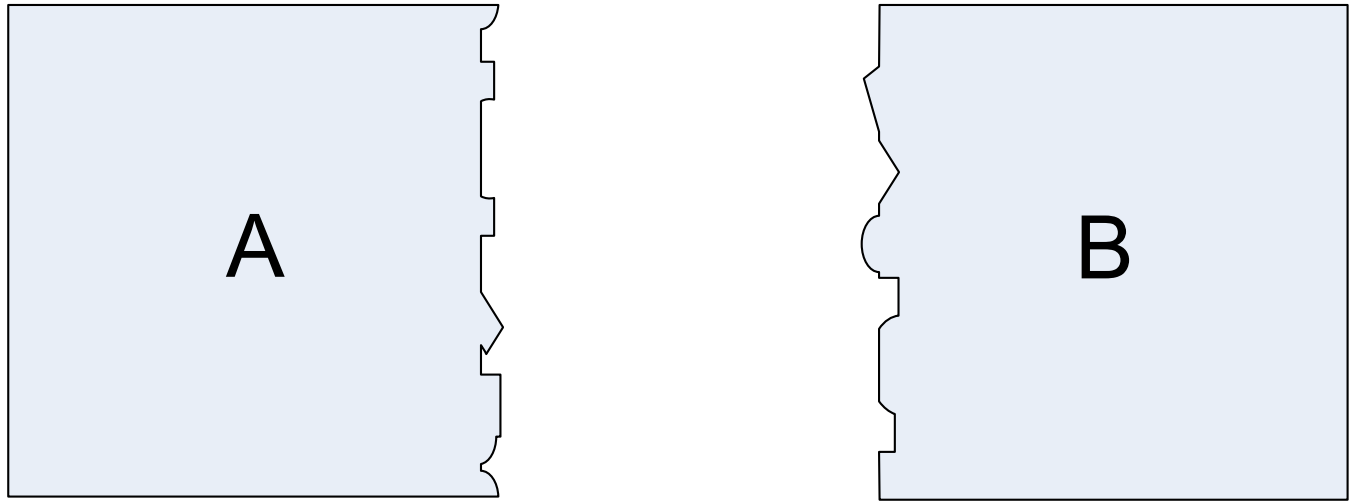
Software is expensive and *inflexible*.

Tools assume:

- ground-up
- perfect fit
- don't change
- never replaced

Reality: none of the above!

A problem



These programming tasks arise:

- as software evolves
- as new user requirements emerge
- as software *ecosystem* evolves
  - e.g. alternative components become available

This talk in one slide

Cake is a tool for tasks like these. It is:

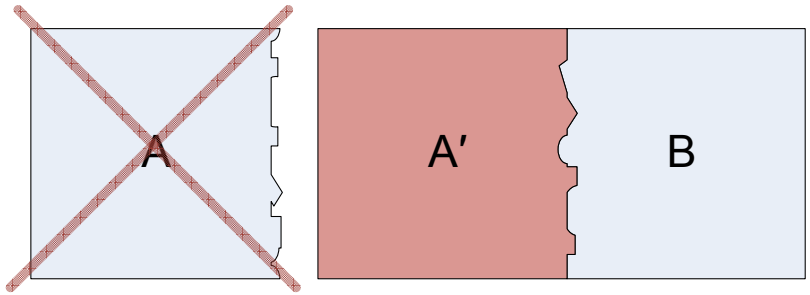
- a rule-based language
- . . . for describing adapters
- declarative
- black-box
- convenient

Key contributions:

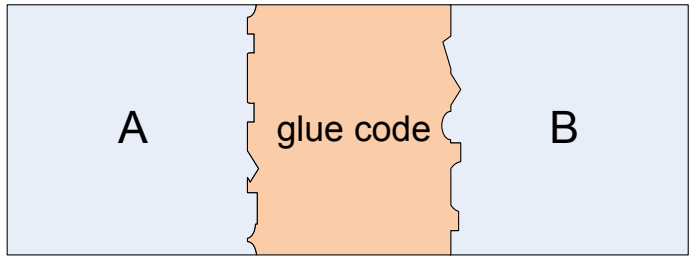
- expressive enough for realistic tasks
- . . . *context-sensitive*, *many-to-many* relations
- . . . evaluated on *real* tasks

Common approaches

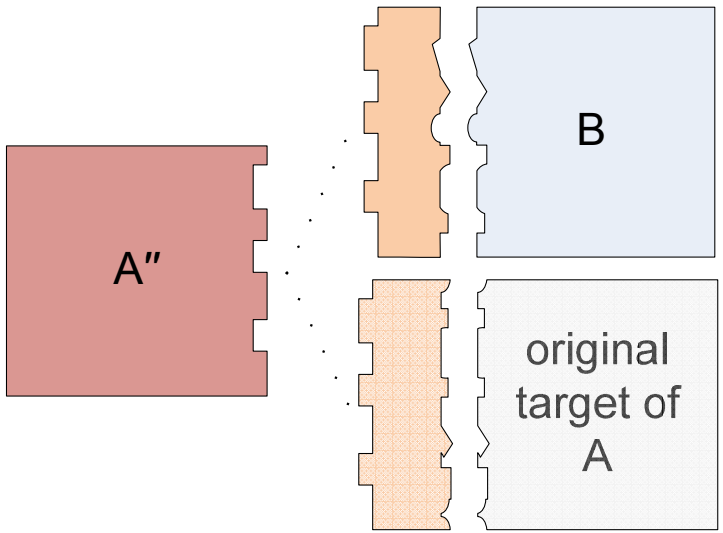
edit or patch



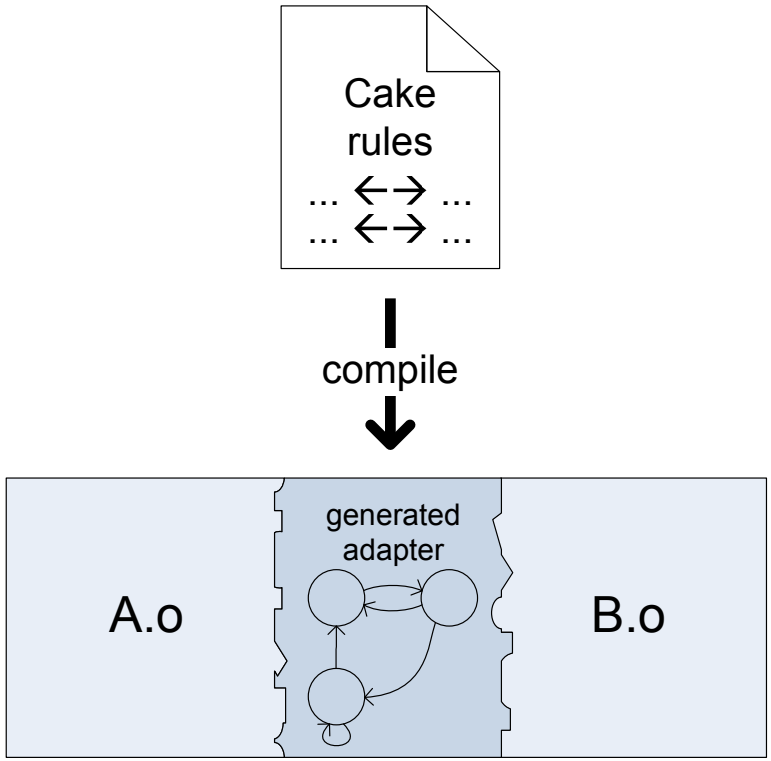
glue coding



abstraction layer

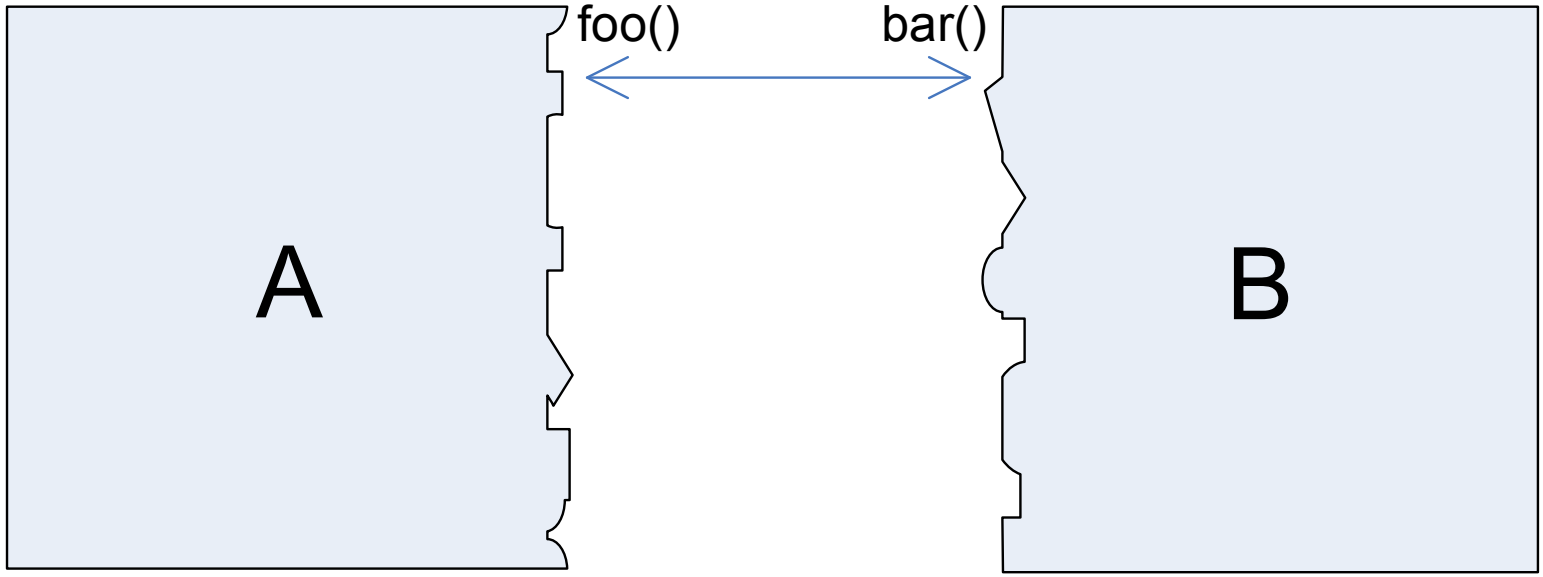


Overview



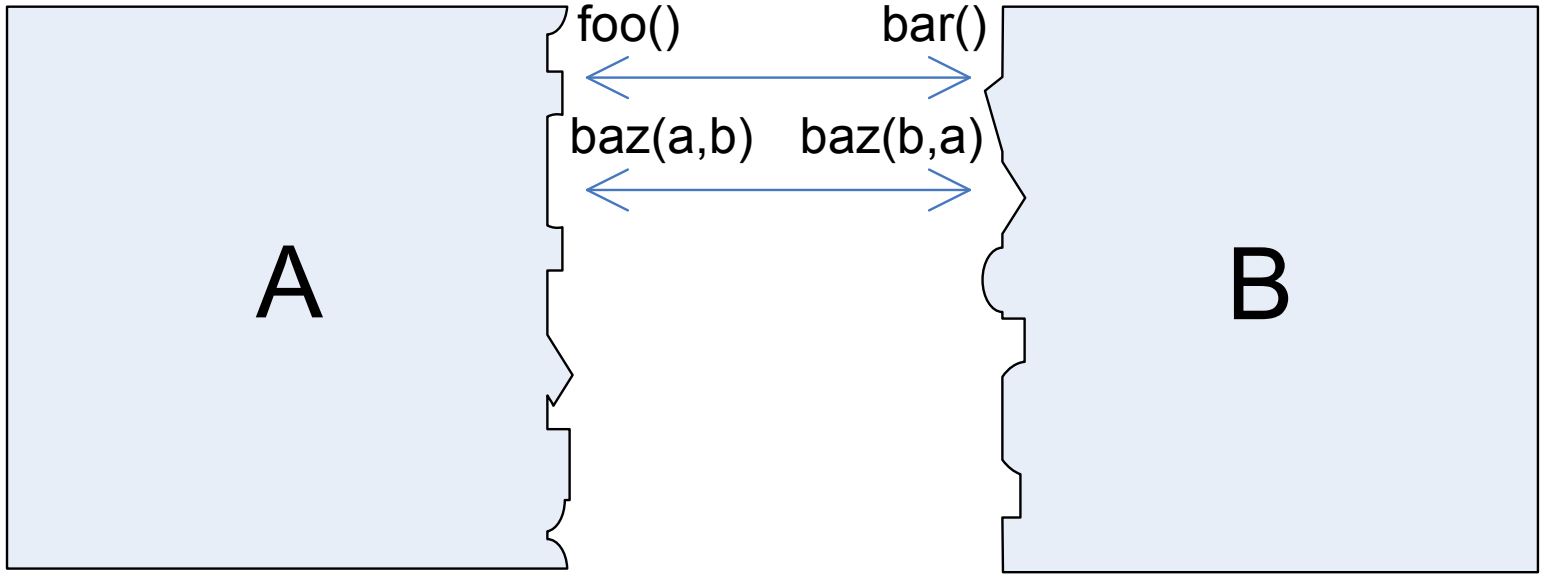

```
exists elf_reloc ("A.o") A; // assume that object files...
exists elf_reloc ("B.o") B; // ... contain debug info
derive elf_reloc ("whole.o") = link[A, B]
{
  /* your rules here! */
};
```

Simple adaptations



foo (...) $\longleftrightarrow$ bar (...);

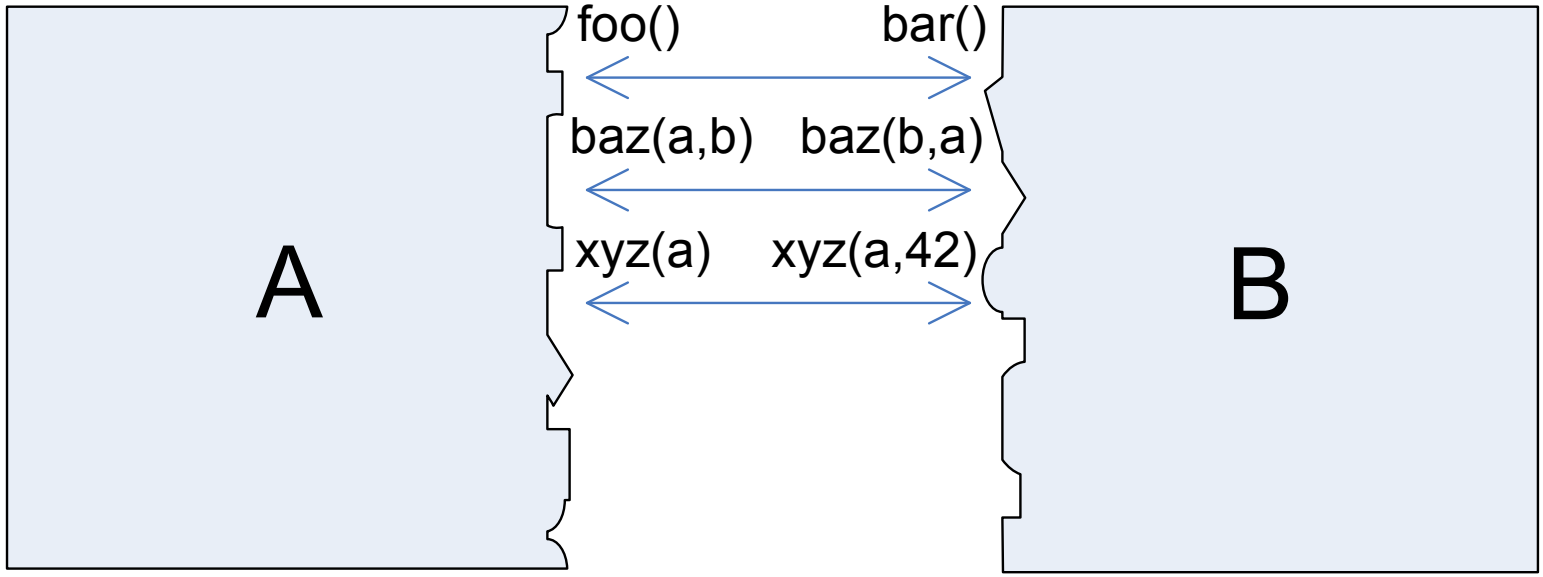
Simple adaptations



foo (...) $\longleftrightarrow$ bar (...);

baz(a, b) $\longleftrightarrow$ baz(b, a);

Simple adaptations

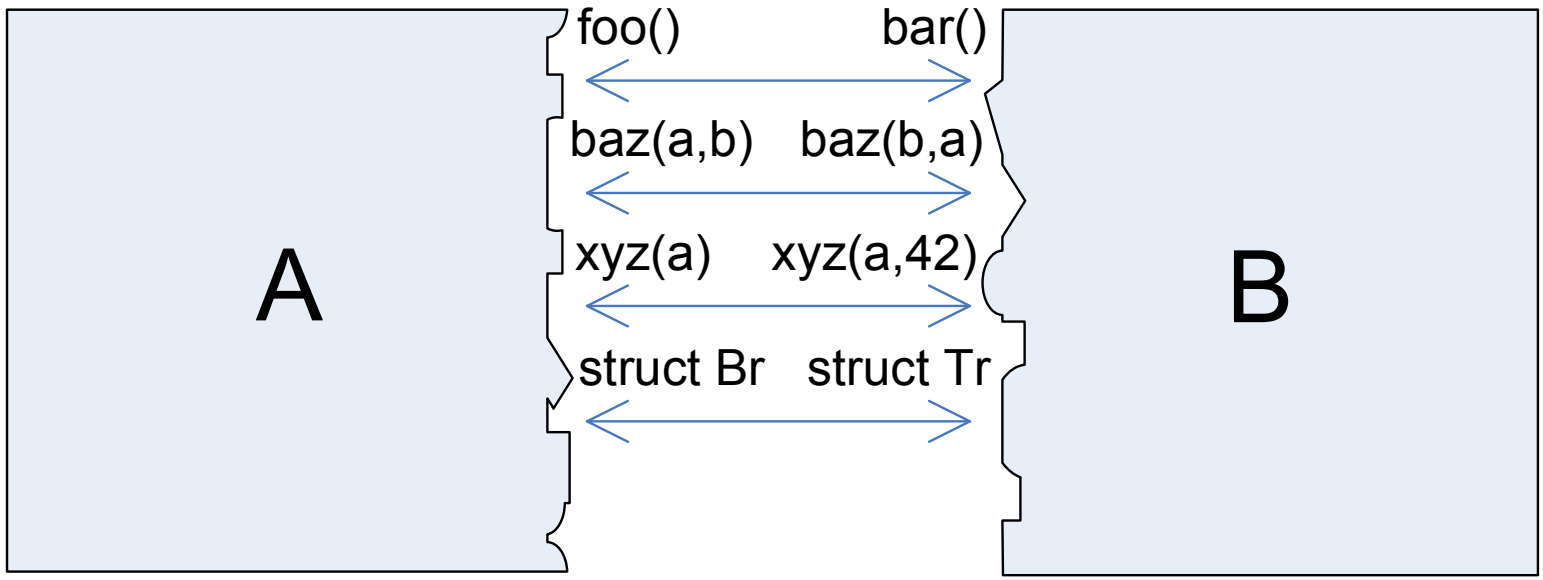


foo (...) $\longleftrightarrow$ bar (...);

baz(a, b) $\longleftrightarrow$ baz(b, a);

xyz(a) $\longleftrightarrow$ xyz(a, 42);

Simple adaptations



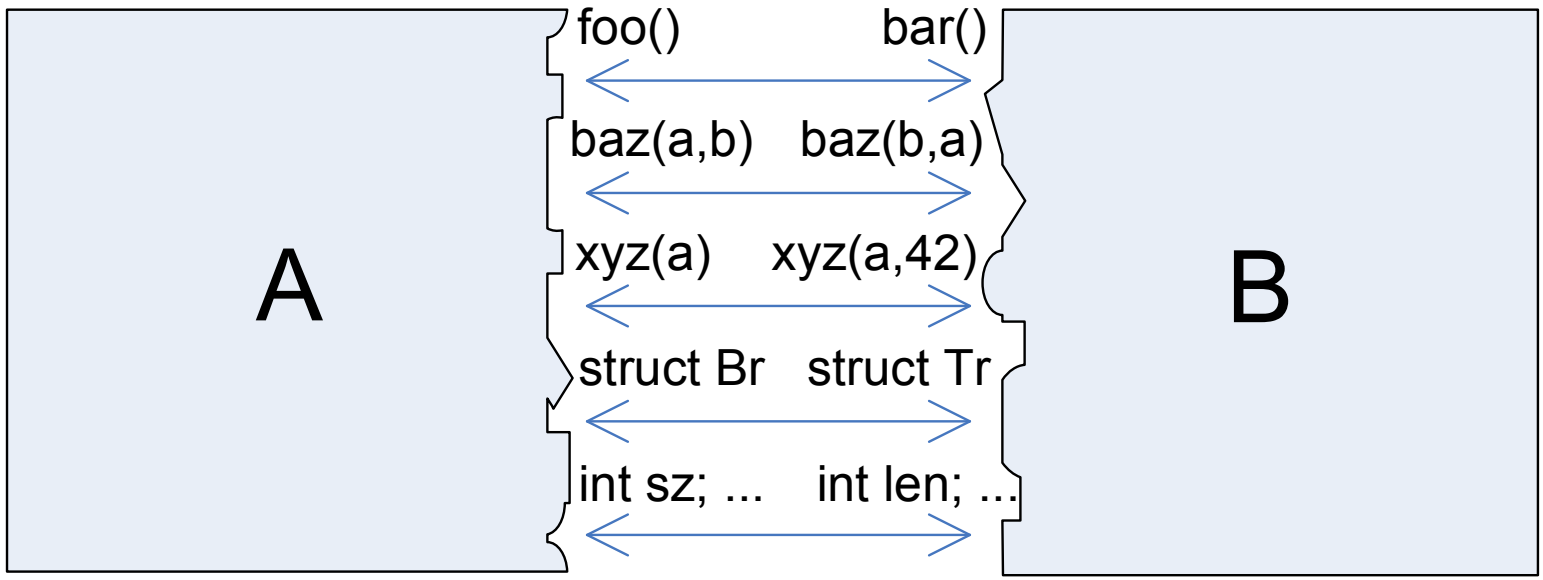
foo (...) $\longleftrightarrow$ bar (...);

baz(a, b) $\longleftrightarrow$ baz(b, a);

xyz(a) $\longleftrightarrow$ xyz(a, 42);

values Br $\longleftrightarrow$ Tr

Simple adaptations



foo (...) $\longleftrightarrow$ bar (...);

baz(a, b) $\longleftrightarrow$ baz(b, a);

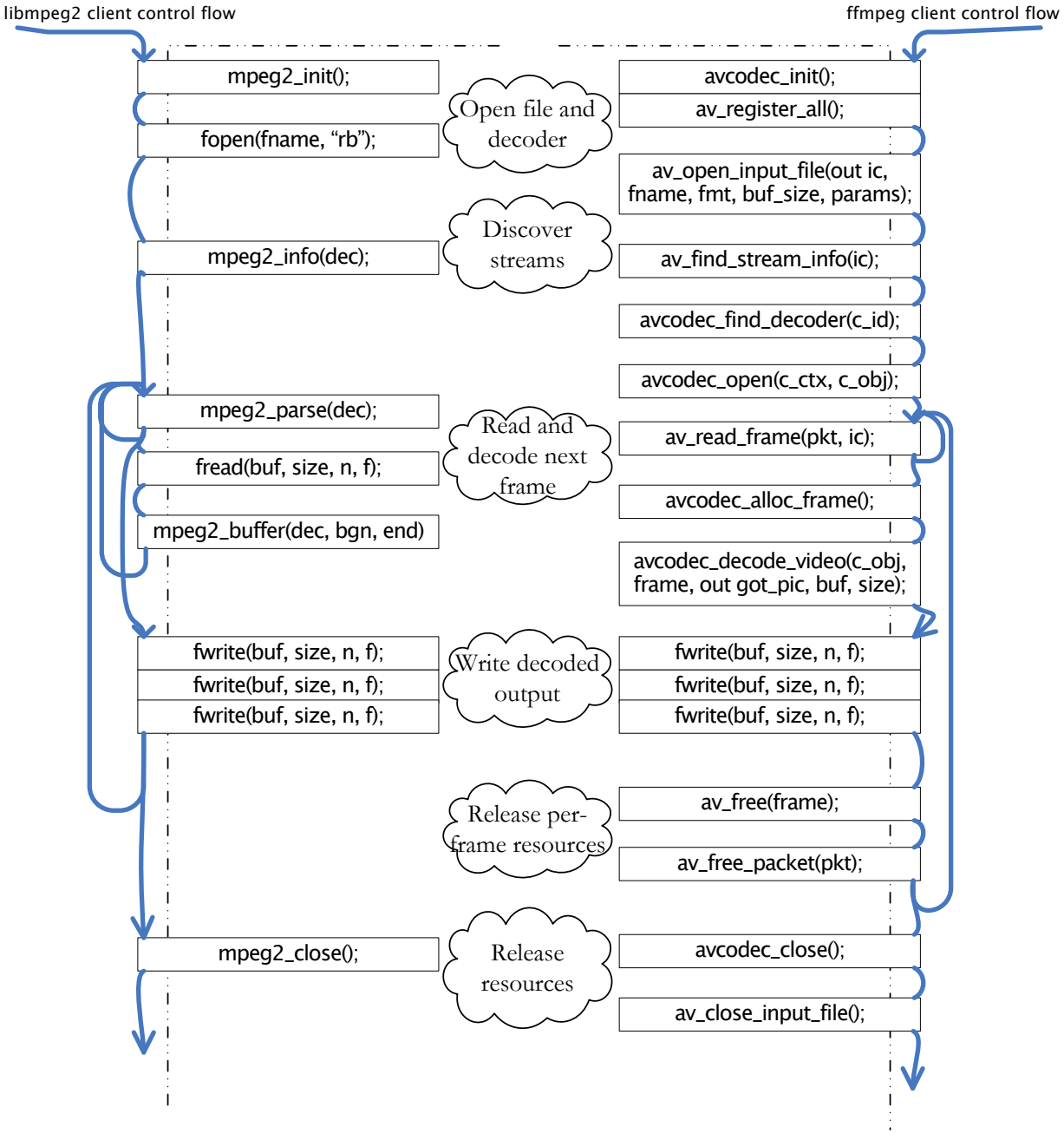
xyz(a) $\longleftrightarrow$ xyz(a, 42);

values Br $\longleftrightarrow$ Tr

{ sz $\longleftrightarrow$ len; };

The Cake compiler generates wrapper functions from rules.

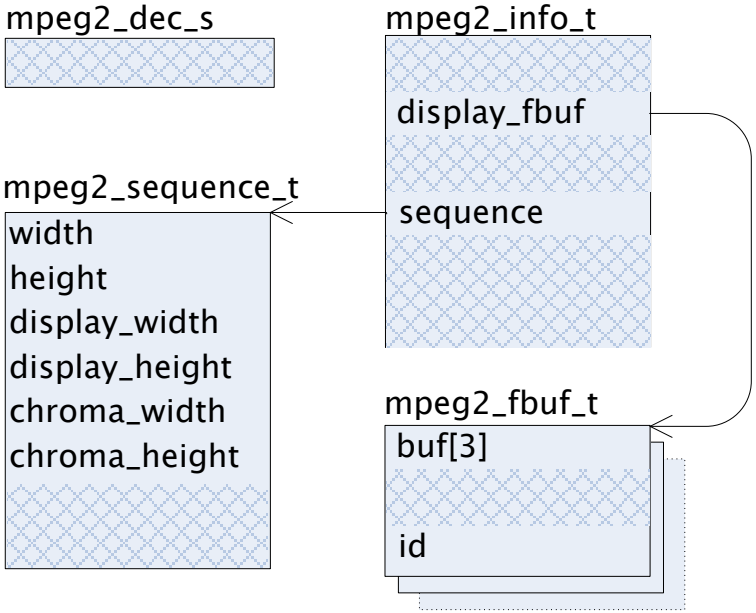
More complex adaptations

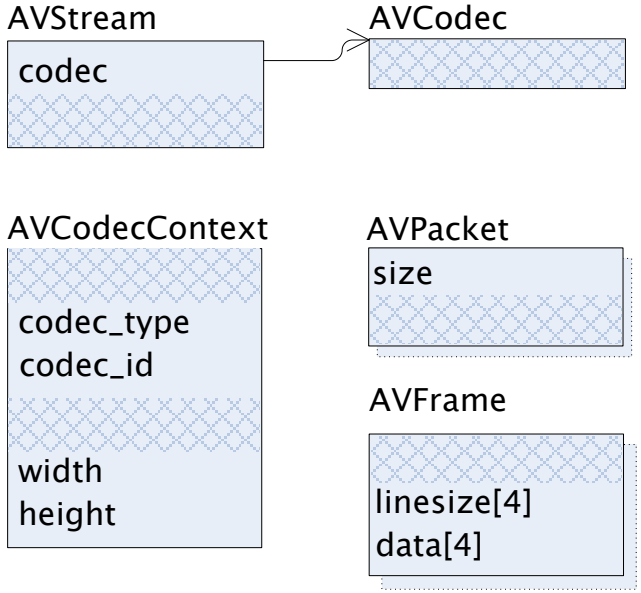


Real interfaces correspond less simply:

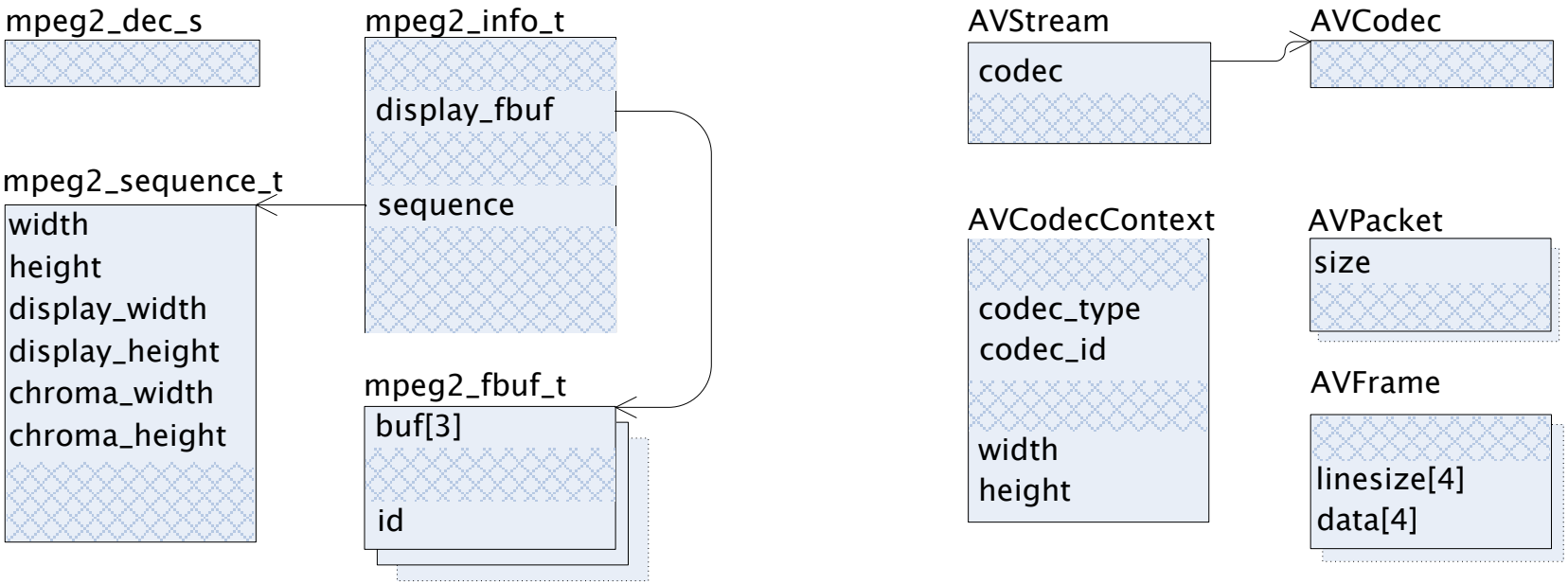
- non-1-to-1 mappings
- context-sensitive
- data, not just code

Many-to-many mappings





Many-to-many mappings



values (dec: mpeg2_dec_s, info: mpeg2_info_s,
seq: mpeg2_sequence_s, fbuf: mpeg2_fbuf_s)

$\longleftrightarrow$

(ctxt : AVCodecContext, vid_idx: int, frame: AVFrame,
p: AVPacket, s: AVStream, codec: AVCodec)

{ fbuf $\longleftrightarrow$ frame
{ buf[0] **as** line[seq.height] **ptr** $\longleftrightarrow$ data[0] **as** line[ctxt.height] **ptr**;
} } /* $\longleftarrow$ *- more rules go here...* */

This creates an *association* at run time (like a join table).

Context-sensitive mappings

Trace of start-up calls appropriate for the two libraries:

```
mpeg2_init()   ↦ dec
fopen("...", "rb") ↦f
mpeg2_info(dec) ↦info
mpeg2_parse(dec) ↦STATE_BUFFER
```

```
avcodec_init() ↦()
av_register_all() ↦()
av_open_file(...) ↦avf
av_find_stream_info(avf) ↦()
avcodec_find_decoder(...) ↦
codec
av_codec_open(...) ↦()
```

Problem!

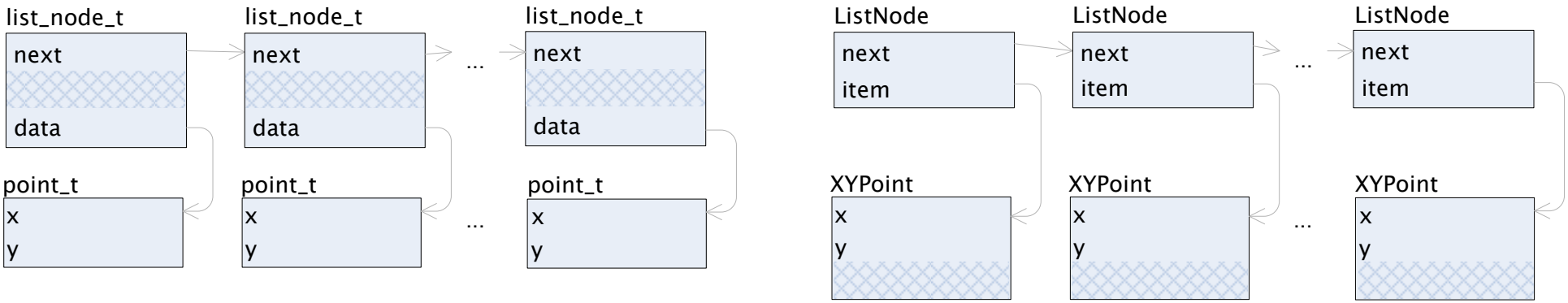
- one `_info` call wants decoder object; other wants file.

Solution: *context predication* with name binding.

let dec = mpeg2_init(), ...,
let f = fopen(fname, "rb"), ...,
mpeg2_info(dec) ⟶{ /* *now both f and dec are available* */ };

Complex object structures

"Passing objects" often means passing *object graphs*.



The Cake runtime traverses object structures automatically.

```
values list_node_t ⟷ ListNode
{  data ⟷ item;  };
values point_t ⟷ XYPoint;
```

Incidentally, note the following:

- name-matching is Cake's default policy
- can insert stub code for value transformation (not shown)

Implementation

Compiler:

- accepts Cake source file
- emits wrapper functions as C++ code
- consumes DWARF debugging information

Runtime:

- allocator instrumentation
- dynamic points-to analysis
- "split heap" management, association tracking, ...

Status: compiler still lagging language design, but WIP...

Evaluation

Compare Cake implementations with pre-existing adapters:

1. p2k: a filesystem adapter from the NetBSD OS
2. ephy-webkit: abstraction layer from Epiphany browser
3. XCL (subset of): compatibility layer for XCB X11 library

Summary outcome:

- less code (code *written*; various syntactic measures)
  - p2k: approx 70% reduction
  - ephy-webkit: approx 65% reduction
  - XCL: approx 30% reduction
- less scattering of concerns

Comparison (1)

```
int seek(struct puffs_usermount *pu,
  puffs_cookie_t opc, off_t oldoff,
  off_t newoff, struct puffs_cred *pcr)
{
   kauth_cred_t cred; int rv;

   cred = cred_create(pcr);
   VLE(opc);



   rv = RUMP_VOP_SEEK(
      opc, oldoff, newoff, cred);

   VUL(opc);
   cred_destroy(cred);

   return rv;
}
```

```
int remove(struct puffs_usermount *pu,
  puffs_cookie_t opc, puffs_cookie_t targ,
  struct puffs_cn *pcn)
{
   struct componentname *cn; int rv;

   cn = makecn(pcn);
   VLE(opc);
   rump_vp_incref(opc);
   VLE(targ);
   rump_vp_incref(targ);

   rv = RUMP_VOP_REMOVE(
      opc, targ, cn);

   AUL(opc);
   AUL(targ);
   freecn(cn, 0);
   return rv;
}
```

Comparison (2)

```
// rules concerning functions
p2k_node_seek(_, vn, oldoff, newoff, cred)
  ⟶ RUMP_VOP_SEEK(vn, oldoff, newoff, cred);
p2k_node_remove(_, vn as vnode_bump, tgtvn as vnode_bump,
    cn) ⟶ RUMP_VOP_REMOVE(vn, tgtvn, cn);

// rules concerning values
values puffs_cookie_t ⟶ ({VLE(that); that}) vnode;
values puffs_cookie_t ⟵ ({VUL(that); that}) vnode;
values vnode_bump ⟶ ({VLE(that); rump_vp_incref(that);
   that}) vnode; // also bump refcount
values vnode_bump ⟵ vnode; // unlock not required
values puffs_cred (cred_create(this)) ⟶ kauth_cred;
values puffs_cred ⟵ (cred_destroy(this)) kauth_cred;
values puffs_cn (makecn(this)) ⟶ component_name;
values puffs_cn ⟵ (freecn(this, 0)) component_name;
```

+ these rules contribute to *other* wrapper functions (28 total)

Related work

Similar tools with narrow domains or less expressiveness:

- Nimble (Purtilo, 1990), BCA (Keller, 1998)
- Jigsaw (Bracha, 1993), Knit (Reid, 2000)
- Swig (Beazley, 1996)
- C++ concept maps (Jarvi, 2007)
- Twinning (Nita, 2010)

Work focused on formalisation rather than implementation:

- Yellin & Strom, 1994; Bracciali, 2003
- subject-oriented composition (Ossher, 1995)

Clean-slate approaches to similar problems:

- Flexible Packaging (Deline, 2001)

Future work, conclusions, . . .

So far, Cake is

- a simpler way of writing short modular adapters
- a convenient tool for binary composition
- a step towards more compositional development

Cake is a platform for lots of potential future work.

- styles (for abstracting heterogeneous object code)
- white-box complement
- improved bidirectionality
- semi-automatic generation of Cake rules

Things I didn't have time to mention

More language features:

- input versus output parameters
- error discovery & handling
- design for heterogeneity
- stub language (algorithms, lambdas, . . .)
- annotations, memory management adaptations, . . .

Repositories:

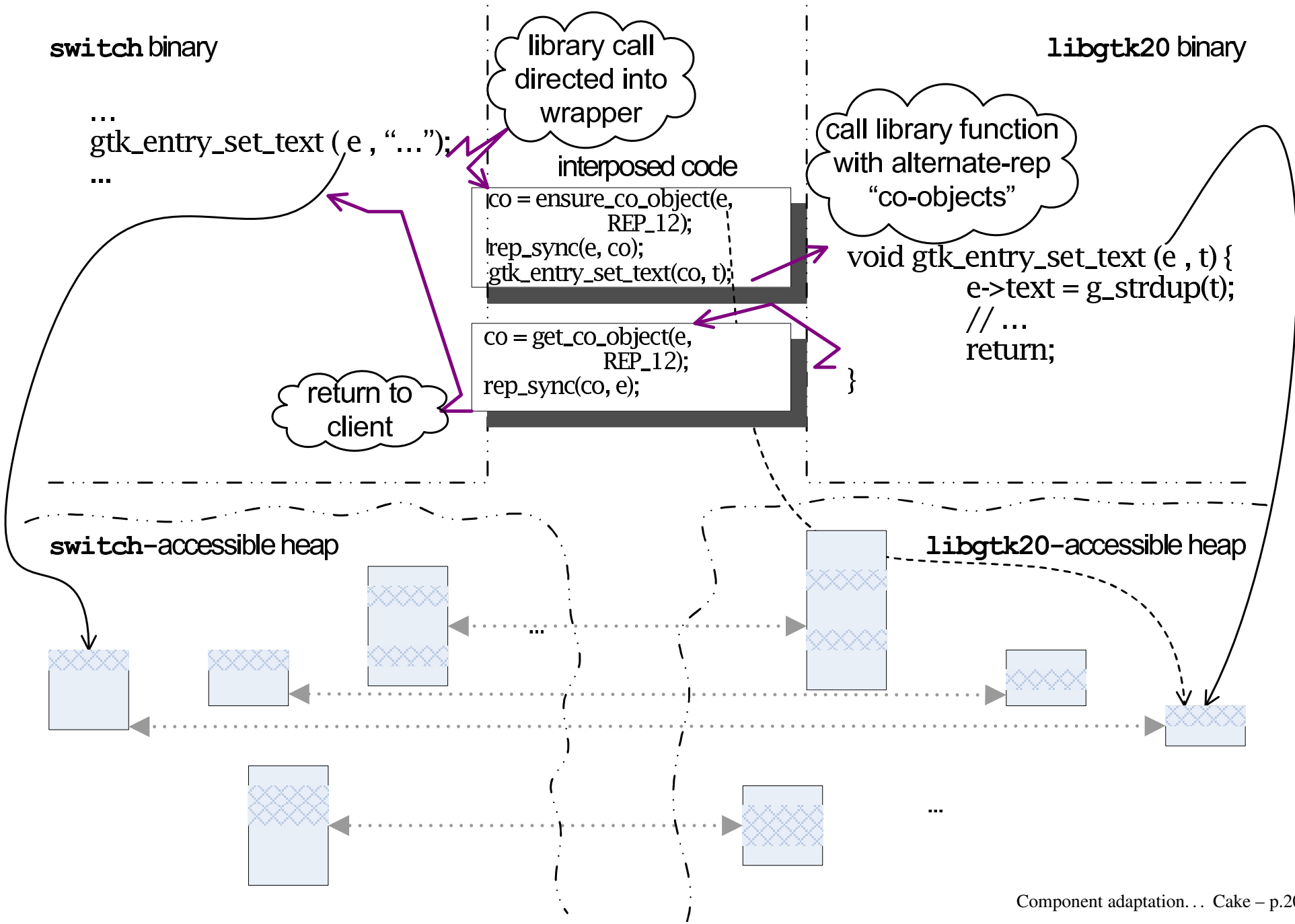
- http://www.cl.cam.ac.uk/~srk31/cake/

Thanks for your attention. Questions?

How Cake wins

- separate treatment of values from treatment of functions
- separate general from special cases
- name-matching
- black-box, binary, language-independent
- designed to accommodate heterogeneity
- make previously edit-requiring tasks black-boxable
- generates sequence recognition code automatically
- maintains object mappings (co-object relation) at run time
- transitive treatment of object structures
- potential for bidirectional rules

Partially split heap



Numbers

p2k

	C	adjusted	Cake	remaining C
LoC (nb nc)	605	523	133	54
tokens	3469	3137	1131	347
semicolons	358	277	69	33

- wins: rule localisation (hugely), allocation

ephy

	C	adjusted	Cake	remaining C
LoC (nb nc)	525	513	161	0
tokens	2529	2455	784	0
semicolons	175	163	70	0

- wins: rule localisation, many-to-many, graph exploration, pattern-matching

Numbers (2)

XCL

	C	adjusted	Cake	remaining C
LoC (nb nc)	380	315	189	42
tokens	2581	2328	1543	232
semicolons	187	148	107	19

- problems: abstraction gap, cross-rule commonality, more data types, more special cases, smaller subset of code (increasing returns?)