Thread and Interrupt Safe Method Dispatch in PCL

Nikodemus Siivola
Steel Bank Studio Ltd

4th European Lisp Workshop, ECOOP 2007

Thread and Interrupt Safe Method Dispatch in SBCL

Nikodemus Siivola
Steel Bank Studio Ltd

4th European Lisp Workshop, ECOOP 2007

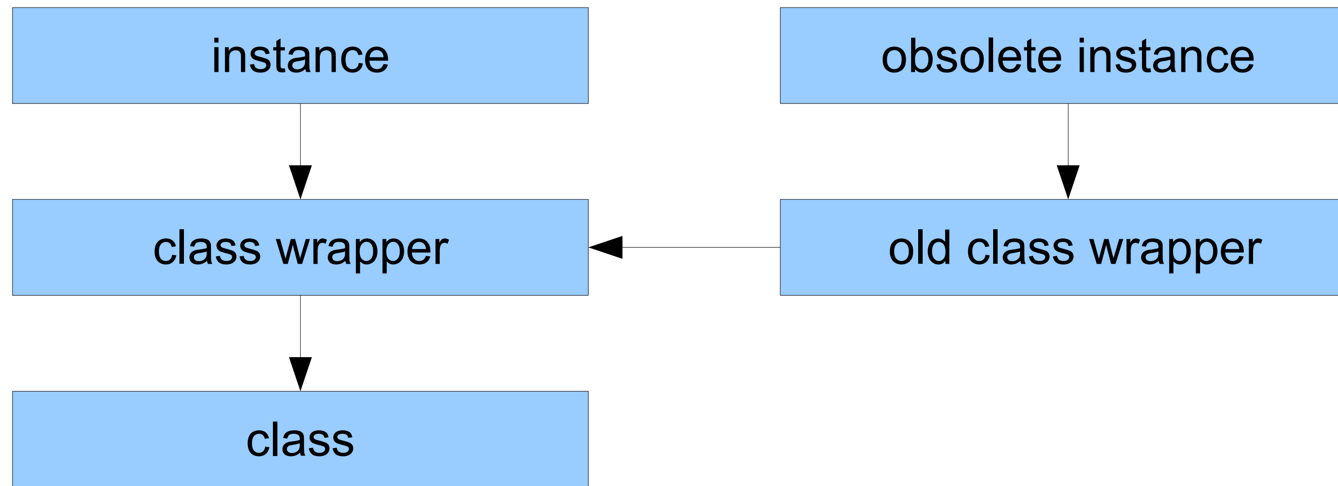
PCL Memoization Cache

Mapping from instances to...

- Effective methods
- Slot offsets
- Constant return values

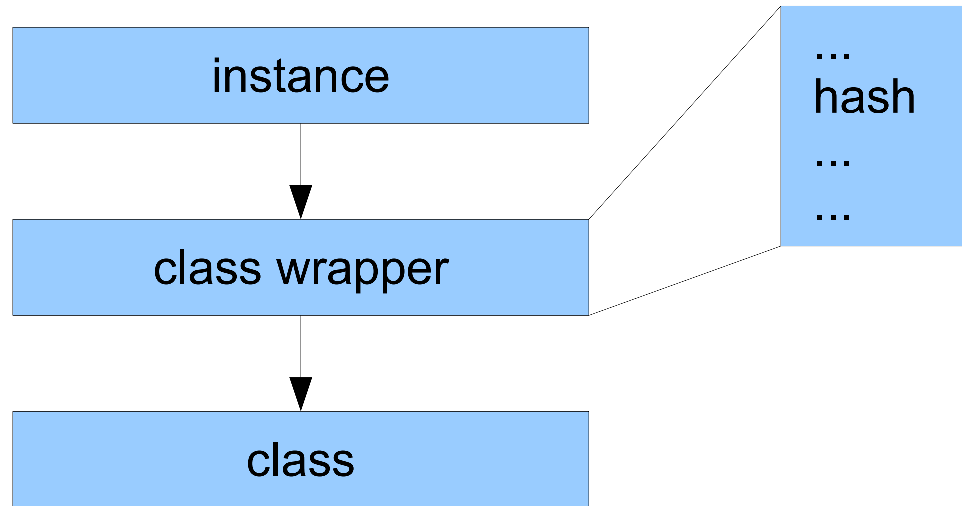
If not the heart, the lungs of method dispatch in PCL.

PCL Memoization Cache



...but that belongs to the realm of obsolete instance protocol.

PCL Memoization Cache



Where there's a hash there's a table...

PCL Memoization Cache

All the action is in the cache vector.

```
#(... wrapper-1 value-a ... wrapper-2 value-b ...)
```

Which is really just an open hash table.

N wrappers can form a compound key.

```
#(... wrapper-1 wrapper-2 value-a ... wrapper-3 wrapper-4 value-b ...)
```

N is constant for each table, of course.

N wrappers and the associated value are called a “cache line”.

```
(+ n 1) => line-size

(mod (* line-size (mix hashes)) (length cache-vector)) => index
```

...so how do we make this thread and interrupt safe?

PCL Memoization Cache

Reading is easy – not interesting.

Writing is tricky: compound key and value means many writes to do, plenty of opportunity for threads to stomp on each other.

Don't want to lock – too expensive.

Compare and swap to rescue!

PCL Memoization Cache

Invariant: keys never change once written.

This allows reads to remain simple and uninteresting. A thread that is walking along the vector doing a probe, reading keys, can trust that whatever it has read so far still holds.

Writes turn out to be easy as well: never overwrite a key and everything is good.

```
(compare-and-swap (svref vector index) +empty+ key) => old
```

If OLD is EQ to KEY or +EMPTY+, the write is considered a success. Of course, in the first case it wasn't really us who wrote the KEY, but as long as the key is whatever we want it to be we don't care.

Value we can just smash in place: as long as the vector is correctly aligned the write is atomic.

PCL Memoization Cache

Thread safe:

- Reads are consistent.
- Writes don't interfere with each other.
- When we run out of space in the vector, we allocate a new one, write the new cache line, and copy all the old ones – and then replace the whole thing at once.
- If multiple expansions occur in parallel, only one wins. This is is harmless as the data is memoization data: losers recompute and write to the new vector.

...what about interrupts?

PCL Memoization Cache

Interrupt safe:

- An interrupted read can unwind harmlessly: no state to speak of.
- An interrupted write will leave an incomplete cache line. This is safe: a later write that matches the already written wrappers will complete the line normally.
- An interrupted expansion will leave no state in the system: the new vector just becomes garbage.

This is pretty informal.

There's More

- Dispatch function updating.
- ADD-METHOD and REMOVE-METHOD.
- Bogus metacircles in COMPUTE-APPLICABLE-METHODS due to interrupts.

...and of course issues we've not yet identified.

Questions?

These slides and an informal implementation report:

`http://sb-studio.net/pub/`