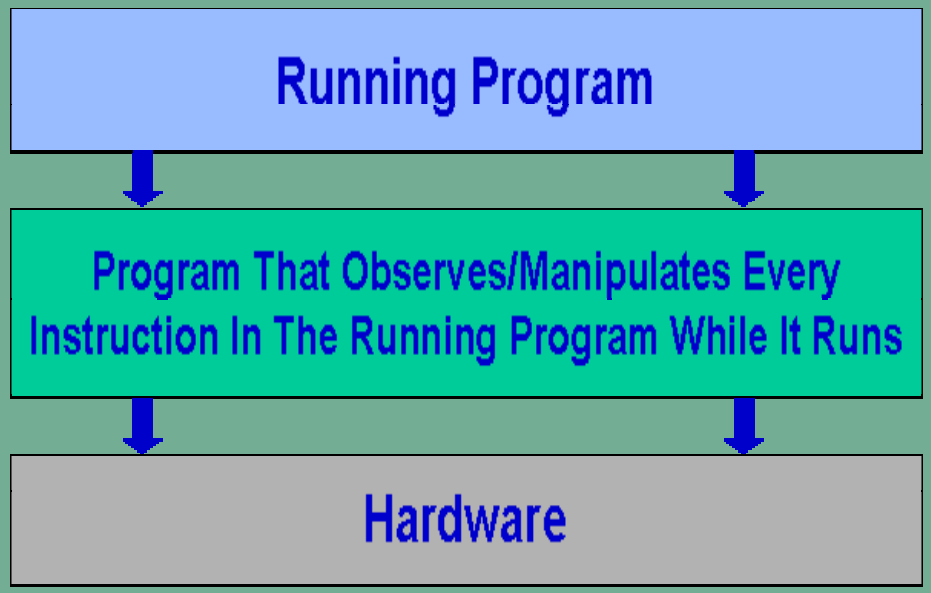
A New Facility for Dynamic Control of Program Execution: DELI

Giuseppe Desoli (STMicroelectronics*)
Nikolay Mateev (HP Labs)
Evelyn Duesterwald (IBM*)
Paolo Faraboschi (HP Labs)
Josh Fisher (HP Labs)

(* work done while at HP)

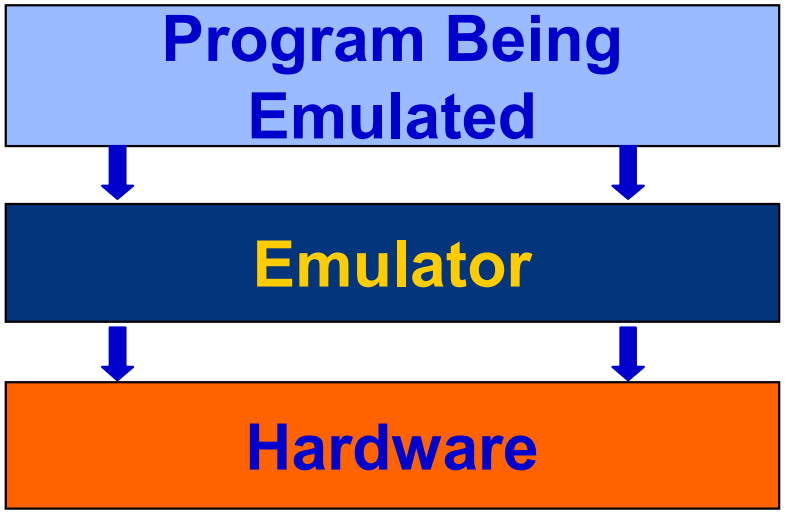
programs that process programs

	compile time	run time
persistent changes	compilers	dynamic loaders, **DELI**
transient changes	[?]	superscalar hardware



the ultimate control point in program execution

example: emulators



- emulators must stare at every target instruction as it runs (can't run on native hardware)
  - this is considered a *"necessary evil"*
  - serious (~100x) performance loss, when operating at the binary level
- an important "trick"
  - **cache, link** and then execute a private copy of fragments of the executable code
  - as long as you stay in the cache, you achieve nearly native speed

short history

the dynamo project at hpl

(1996-2001+)

hpl project whose goal was to optimize pa-risc binaries at run-time (Vasanth Bala, Evelyn Duesterwald)

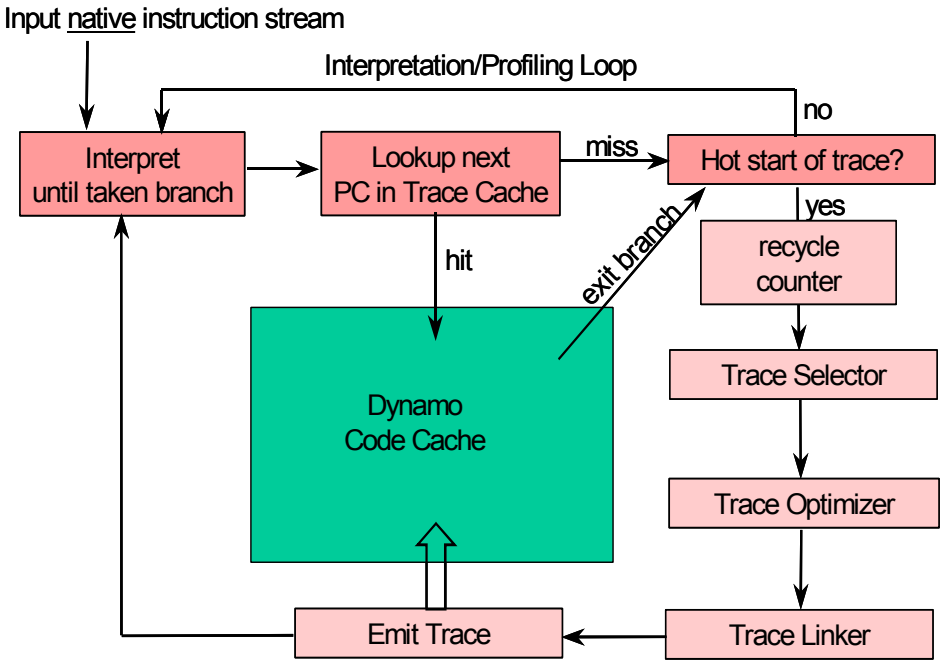
- in effect, they did native-native emulation, with optimization as the desired side-effect

produced a startling result

- **given a running program, you can run a second program ("dynamo") that speeds up the first one, more than enough to amortize the cost of running dynamo**

some (already optimized) *specint95* programs went 23% faster

how dynamo works



dynamo speeds up native code, making up for its own overhead

the gains come from

- inlining small functions
- "straightening out" branches
- cache locality improvements
- eliminating redundant loads

dynamo uses *cache & link*

- when we realized that we could do this at 'native speed', we thought about a general layer, accessible for many different purposes

the "cambridge DELI"

the point of this talk

... this is not an emulator, and we did not rediscover cache & link

a dynamic control point is very powerful

but only an elaborate fantasy if it costs 100x performance

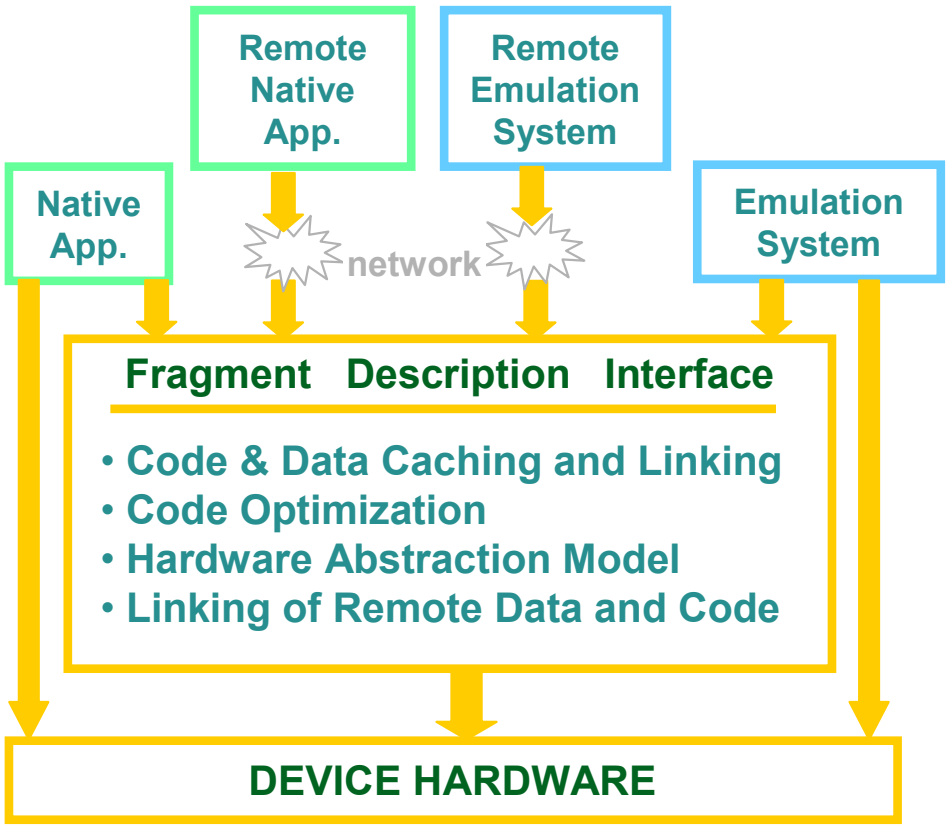
we can make this practical and open it up to many clients applications

the potential and variety of clients that take advantage of this is stunning

virtually every processor— gp or embedded— could include such a facility

if this works, there will be far-reaching changes in what computers do and how they do it

DELI: Dynamic Execution Layer Interface



a "control point" in program execution to **dynamically transform** code and data, enabling:

- dynamic optimization
- caching emulation
- transparent remote streaming
- virtualized hardware

DELI facilitates

- efficient emulation
- program transformation
- program observation

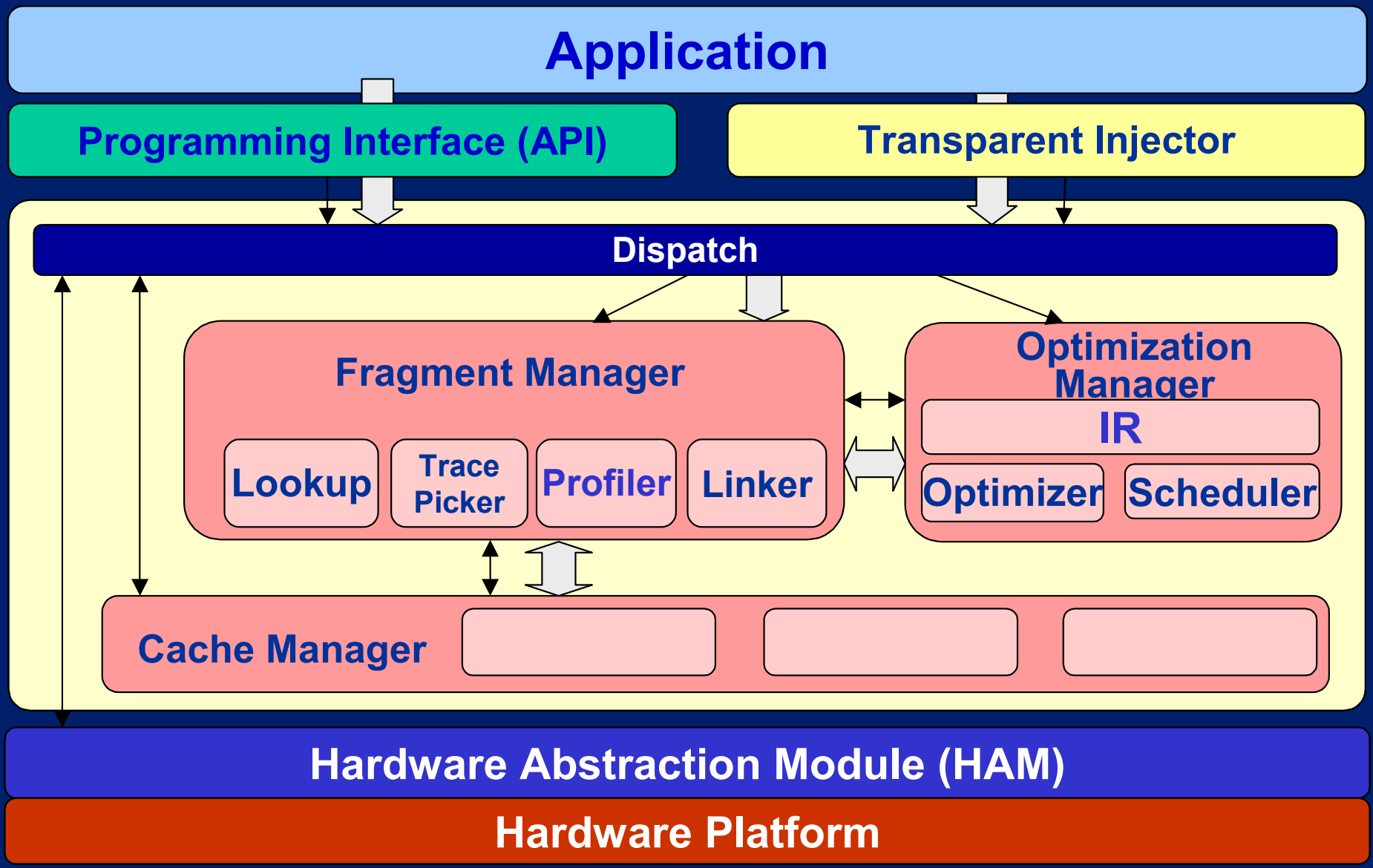
work in progress on different platforms

- x86, SuperH, ARM, ST200
- PocketPC, Linux

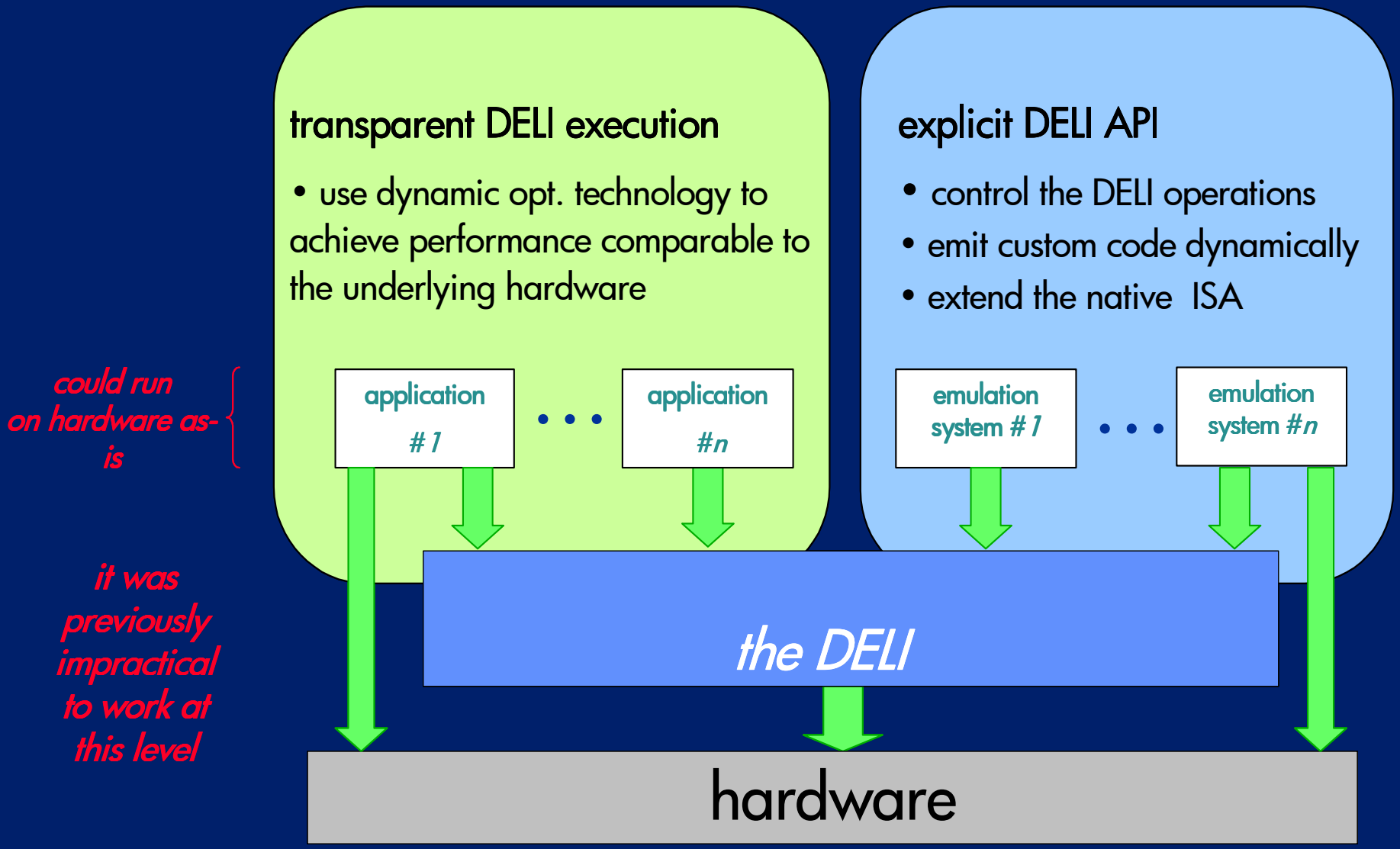
the DELI infrastructure

- provides a **uniform layer** for caching & linking of code
- facilitates the construction of emulators & virtual machines
- provides hardware virtualization
- integrates emulated and native code execution
- acts as a system service to support
  1. **observation**
  2. **transformation**
  3. **emulation**

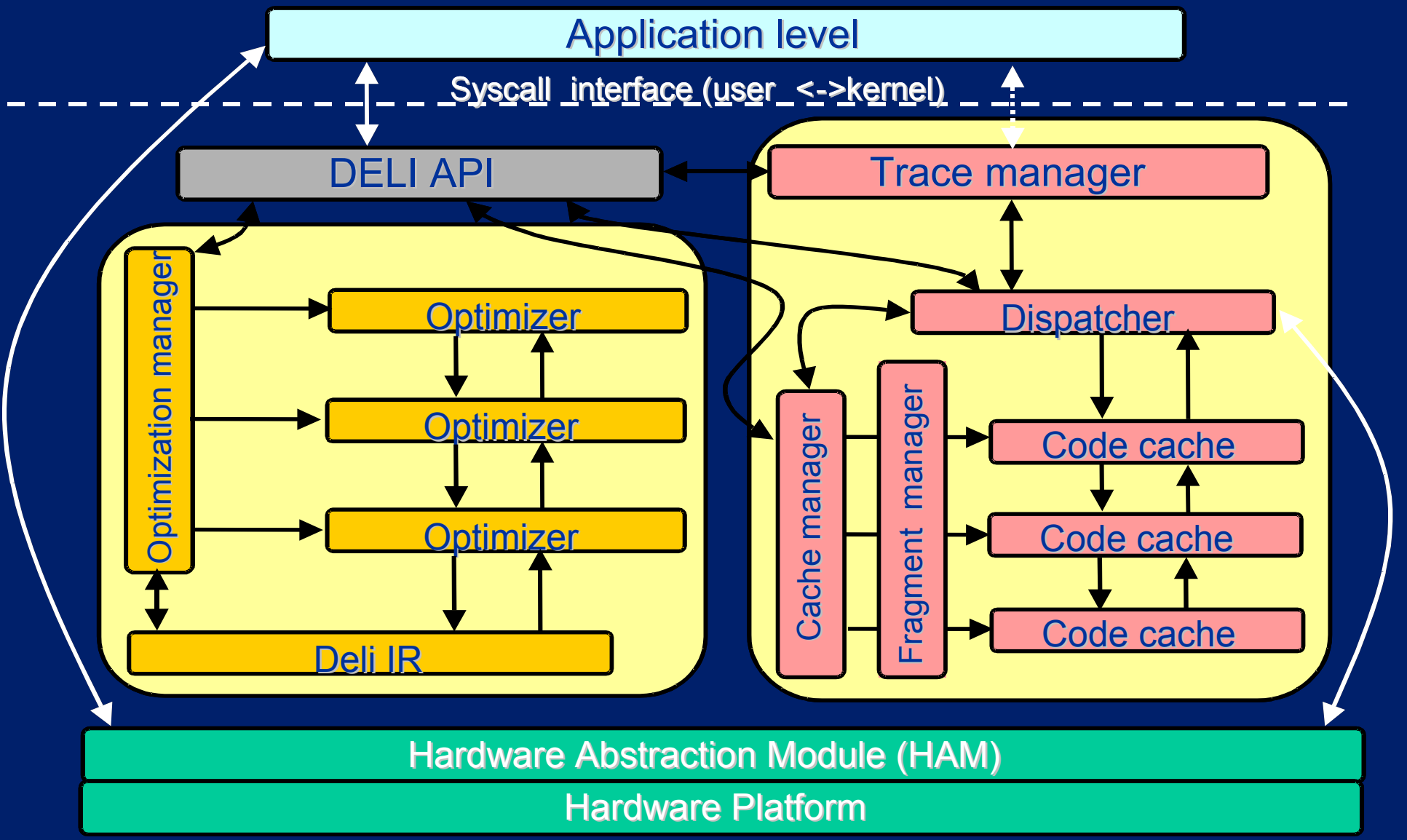
DELI components



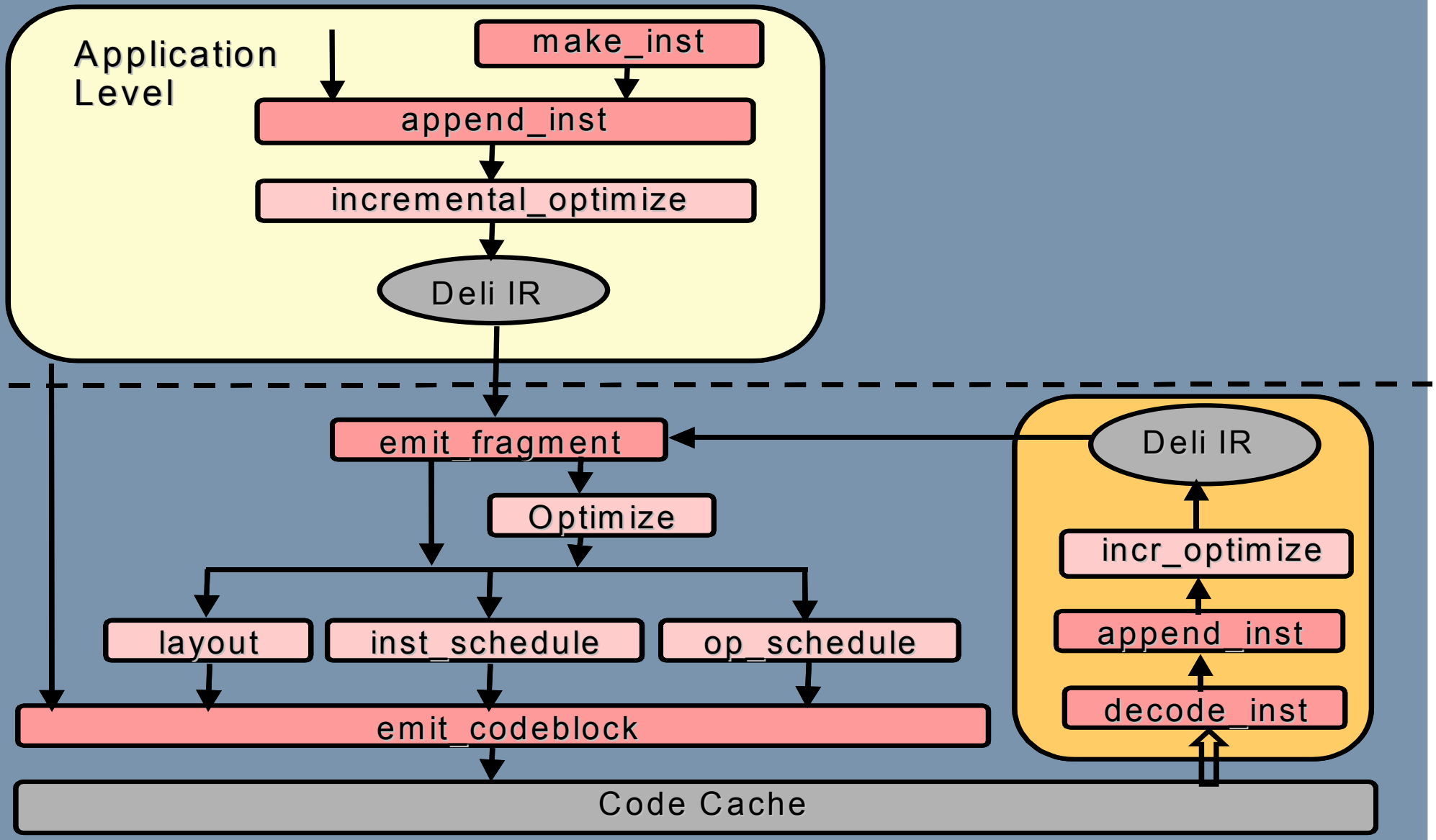
transparent or explicit?



how DELI works



the transformation infrastructure



how unique is DELI?

Transmeta *"code-morphing"*

Transitive *"dynamite"*

- transparent to the application and bundled with HW
- DELI exports API to application and supports mix native-emulated

VM-Ware: full-system emulation, transparent to the client

- DELI is not an emulator- it only provides support for it

AppStream: streams Java binaries to client machines running JVMs.

- requires Java, and modification of the client-side JVM
- DELI can support streaming of legacy native binaries

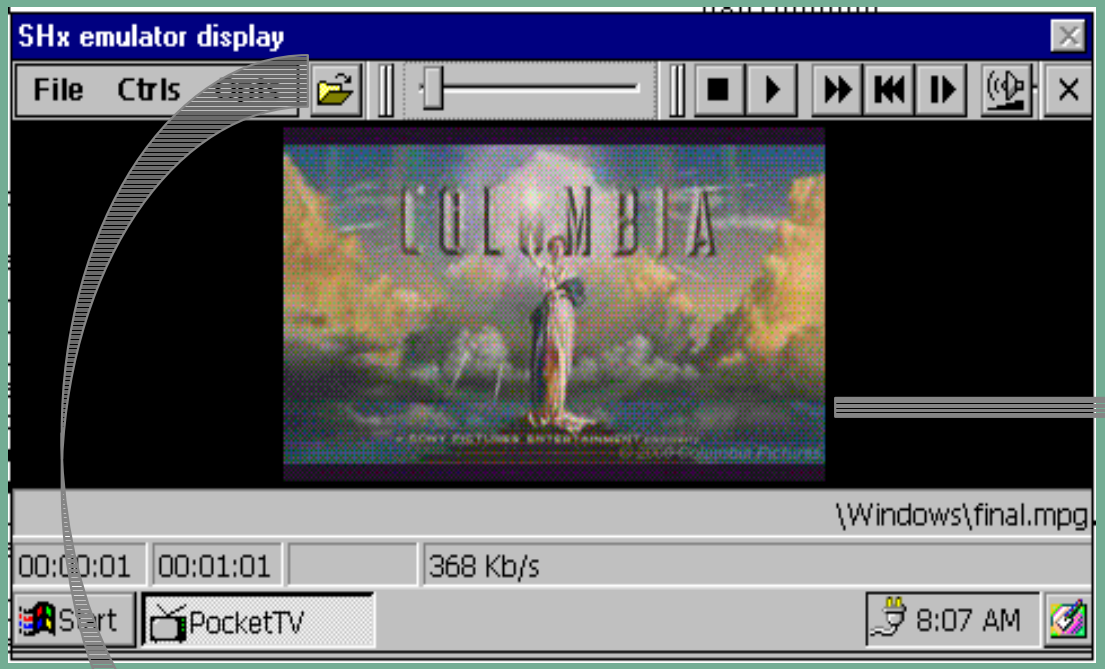
emulation with DELI:

the "*DELIverX*" prototype

- efficient emulation of embedded cores on a VLIW processor
  - leverages OS + SDK (e.g. WinCE) without a port
  - augments the OS functionality with no OS changes
- integration of native and emulated code
  - allows native implementation of compute-intensive tasks
  - enables integration of legacy GUIs with native computation engines
  - facilitates "incremental" migration across product generations

example:
emulation in rich-media devices

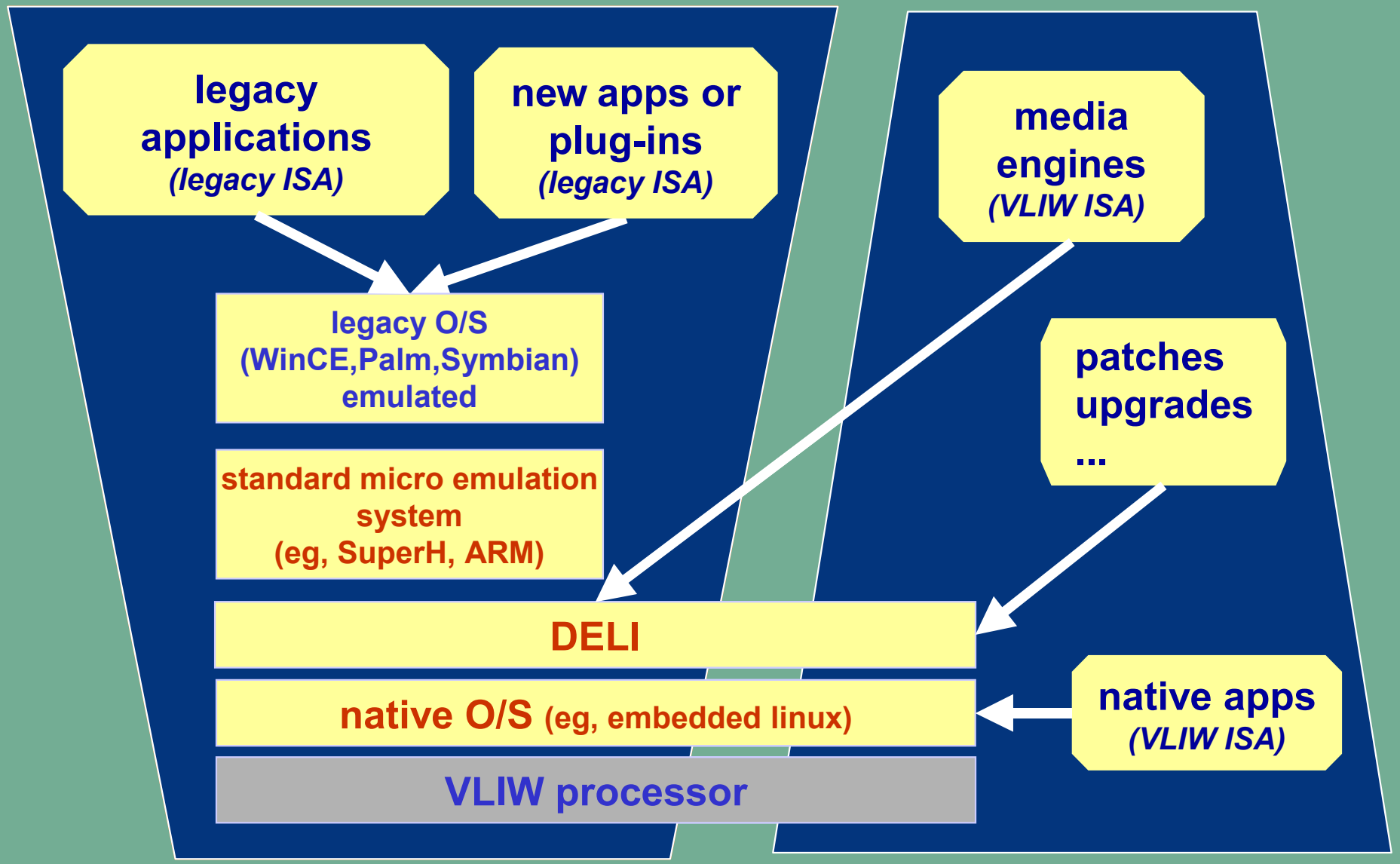
PocketTV running on emulated PocketPC /SuperH



Native thread decodes mpeg video/audio stream with a large speedup with respect to actual target core

Emulated application controls user interface and native thread is seen as a library in the emulated environment

emulation in an embedded appliance



"Lx": a vliw architecture for media-oriented applications

joint hp - stmicro family of vliw embedded cores

- base vliw instr. set + extensions
- efficient 32-bit integer ISA
- extensible (for fp, simd, special ops)
- many gp/predicate registers
- simple predication through "select"
- static branch prediction
- precise interrupts
- explicit speculation and prefetching

customizable and scalable for a specific application domain

variable number of clusters and registers, cache sizes, operation latencies, special operations

the st200 (L*x*) vliw architecture

DELIverX target: the ST210 core

HP & STMicroelectronics joint project

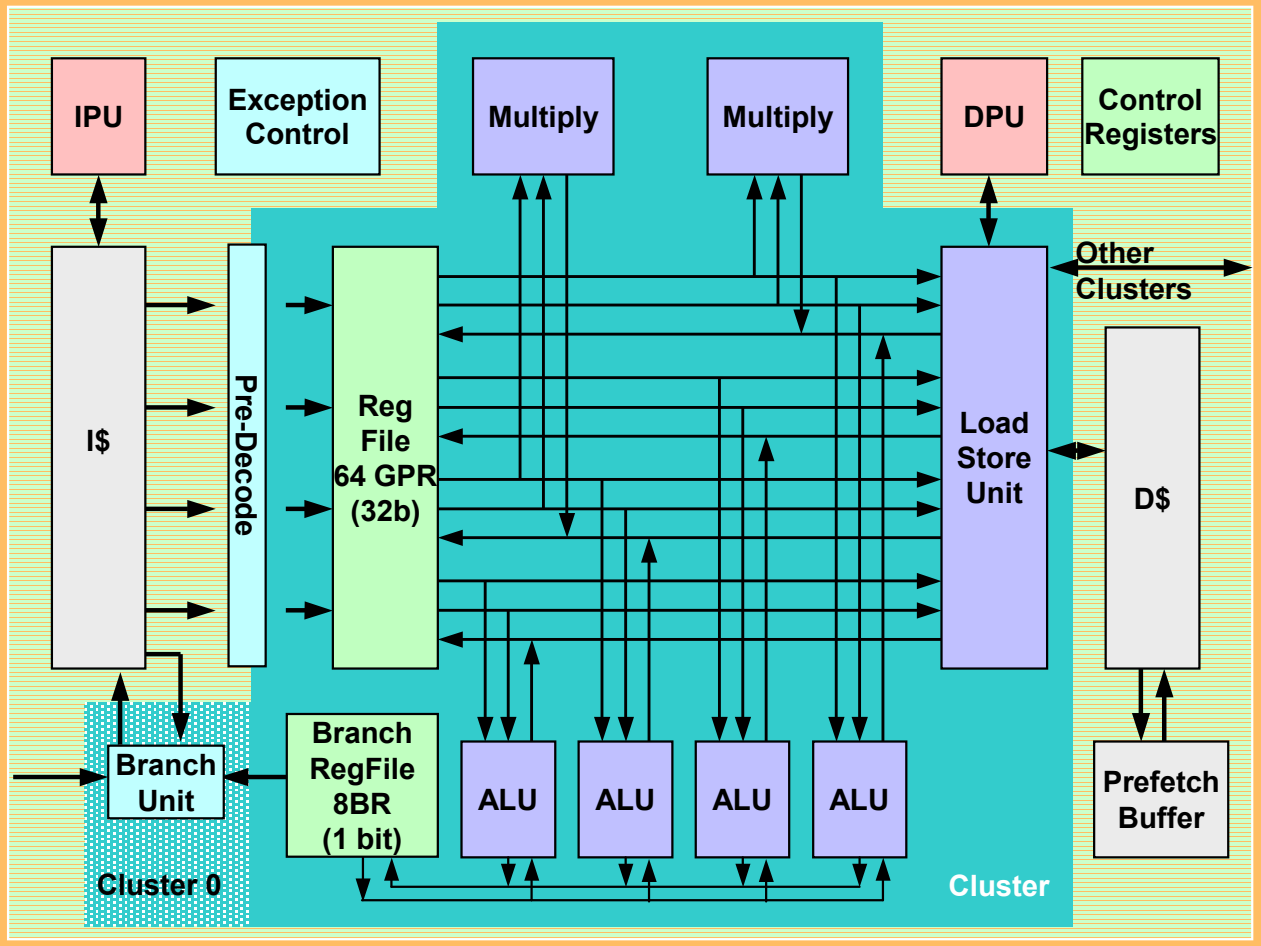
4-way VLIW

250MHz (0.18µ)

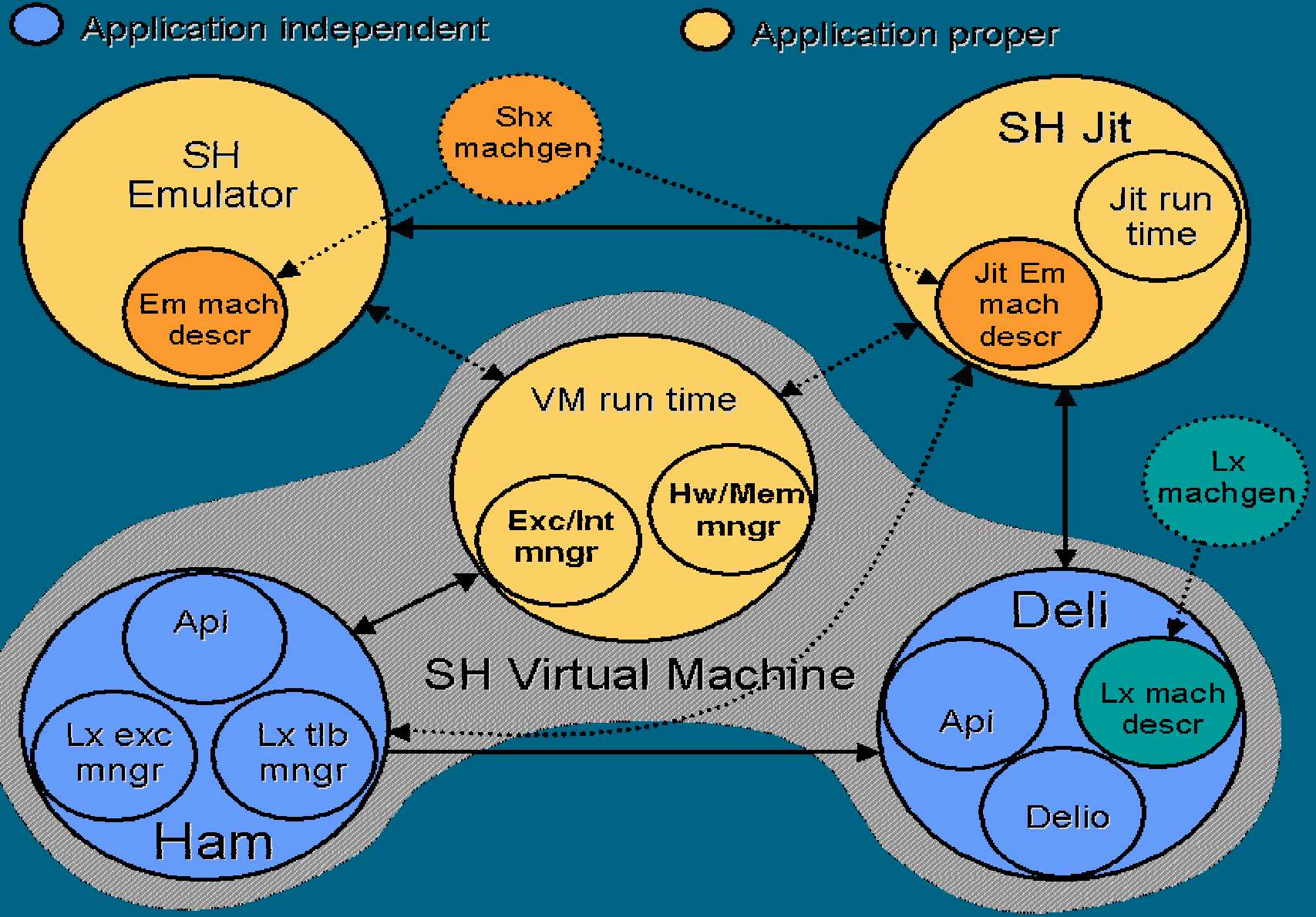
scalable

customizable

compiler-driven



DELIverX components

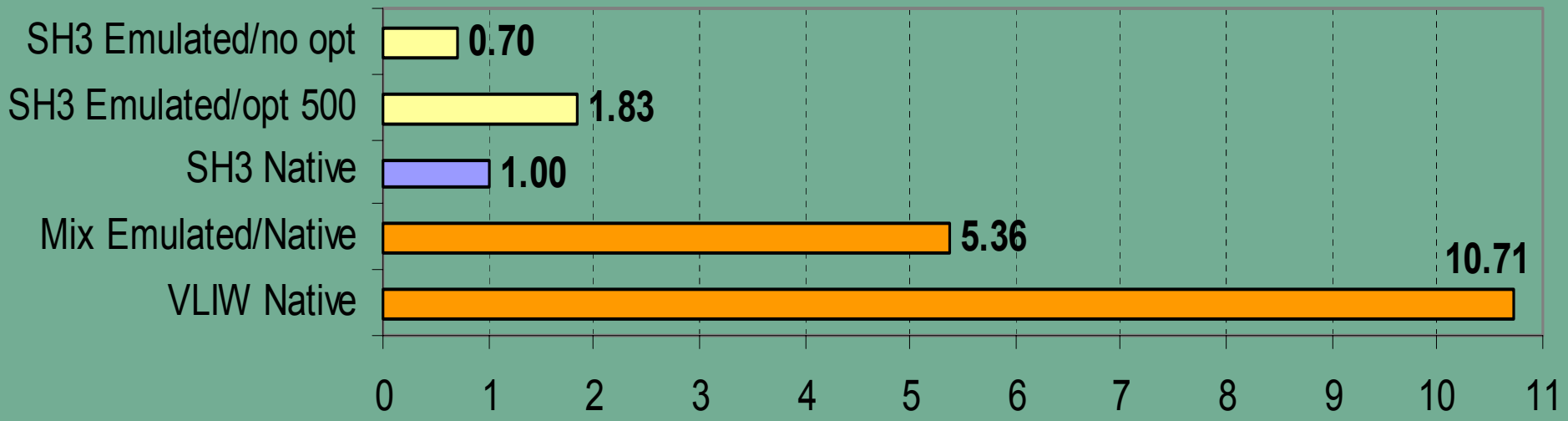


DELIverX performance

mixing native and emulated code

value proposition for developers:

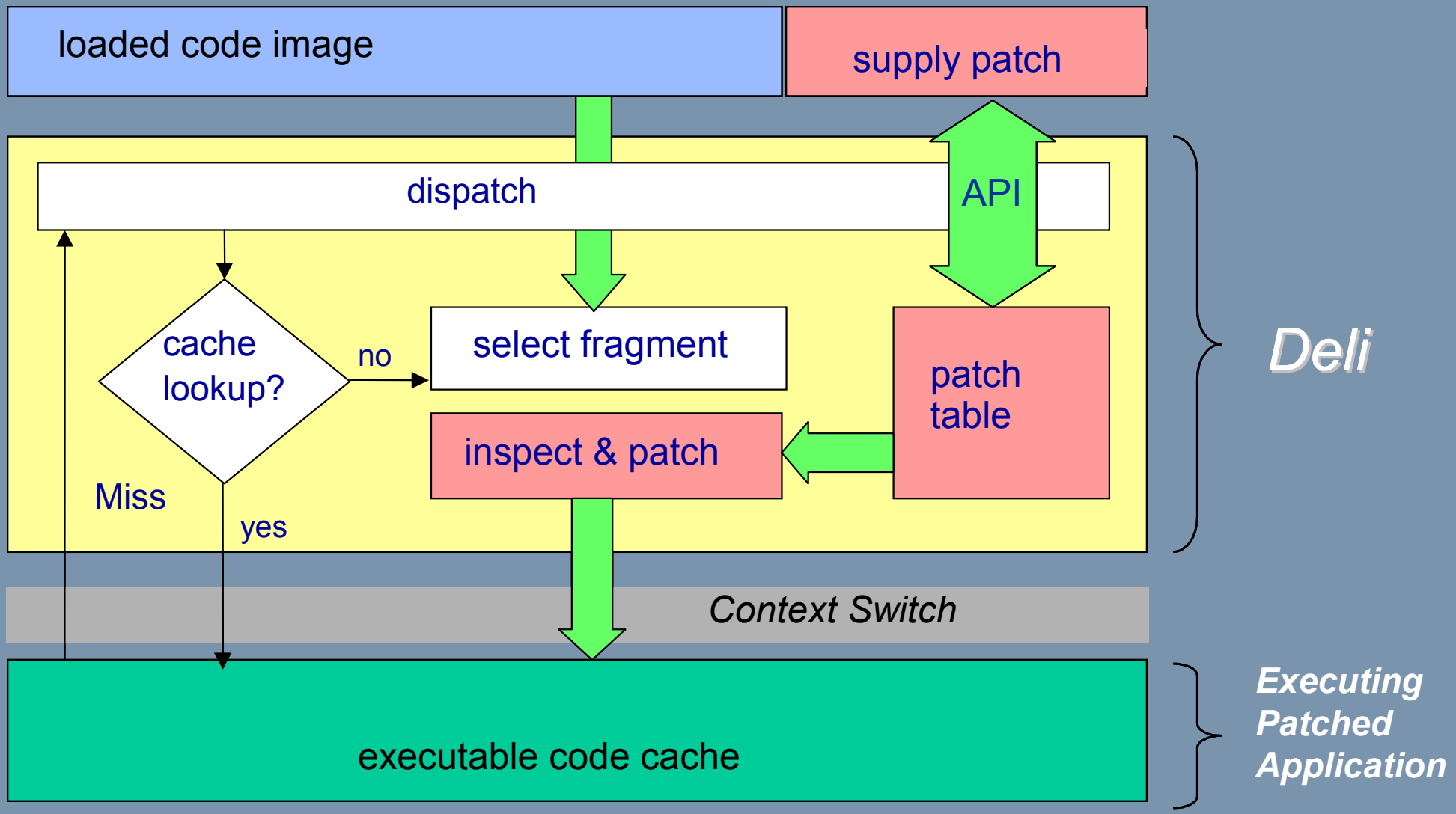
- scale performance by selectively porting the compute-intensive kernels of applications
- without giving up the benefits of a legacy operating system and GUI



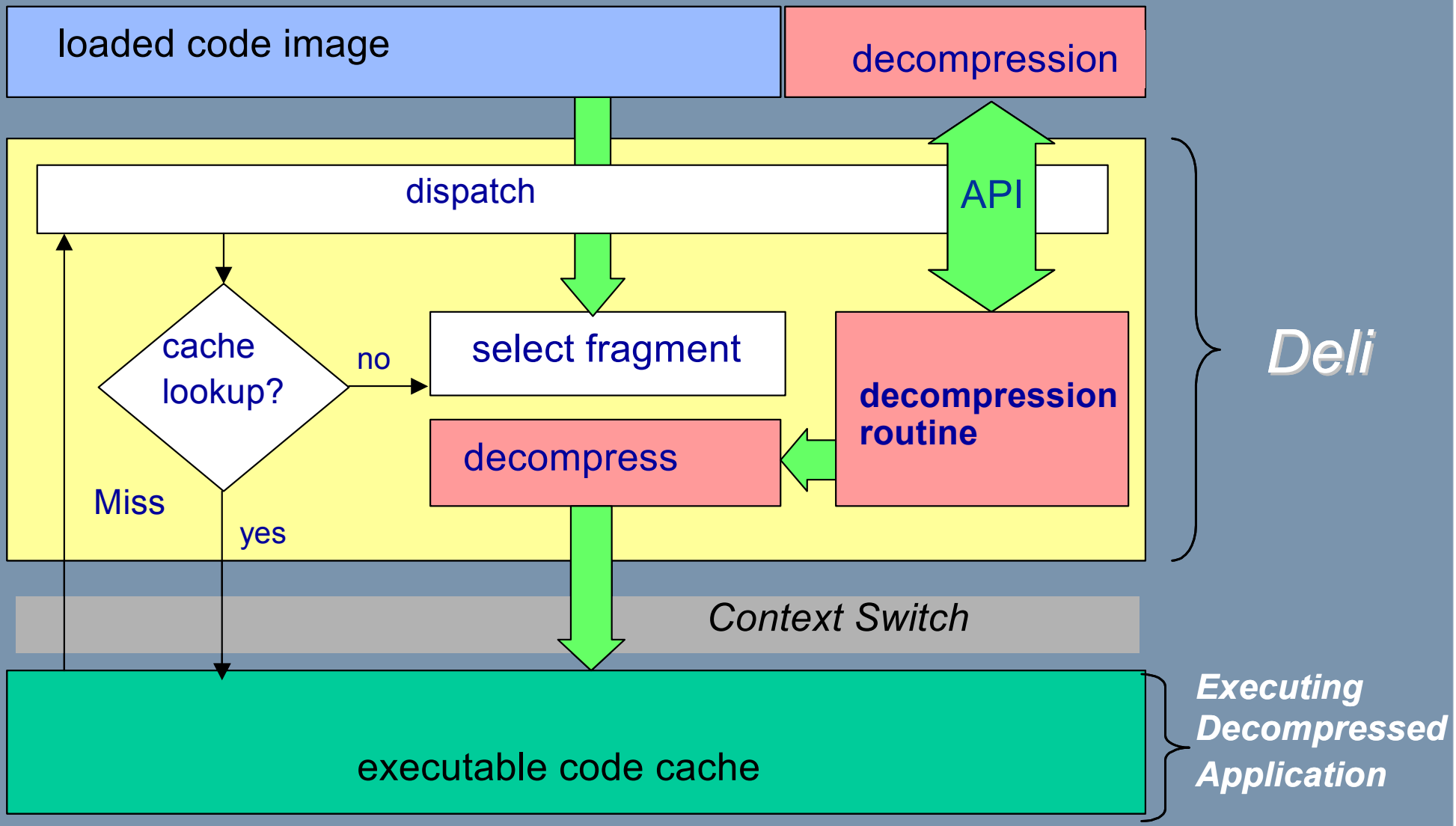
transforming and observing programs

- **emulation** is just one of the many DELI clients
- many types of **code transformations** are practical at run-time
  - dynamic code patching
  - streaming of data & code
  - code decompression
  - code decryption
  - polymorphic virus detection
- many types of **code observations** are useful at run-time
  - sand-boxing
  - profiling
  - digital rights management

transformation example: dynamic code patching



transformation example: code decompression (or decryption)



summary

DELI is a general layer, accessible for many purposes

other alternatives have a specific usage in mind, or point products

we have shown feasibility, achievable performance, and basic infrastructure

future work ideas

hardware support
compiler annotations
more applications