Today

- ✓ Distributed System Basics
- ✓ Lamport's clock
- ✓ Efficient Communication software

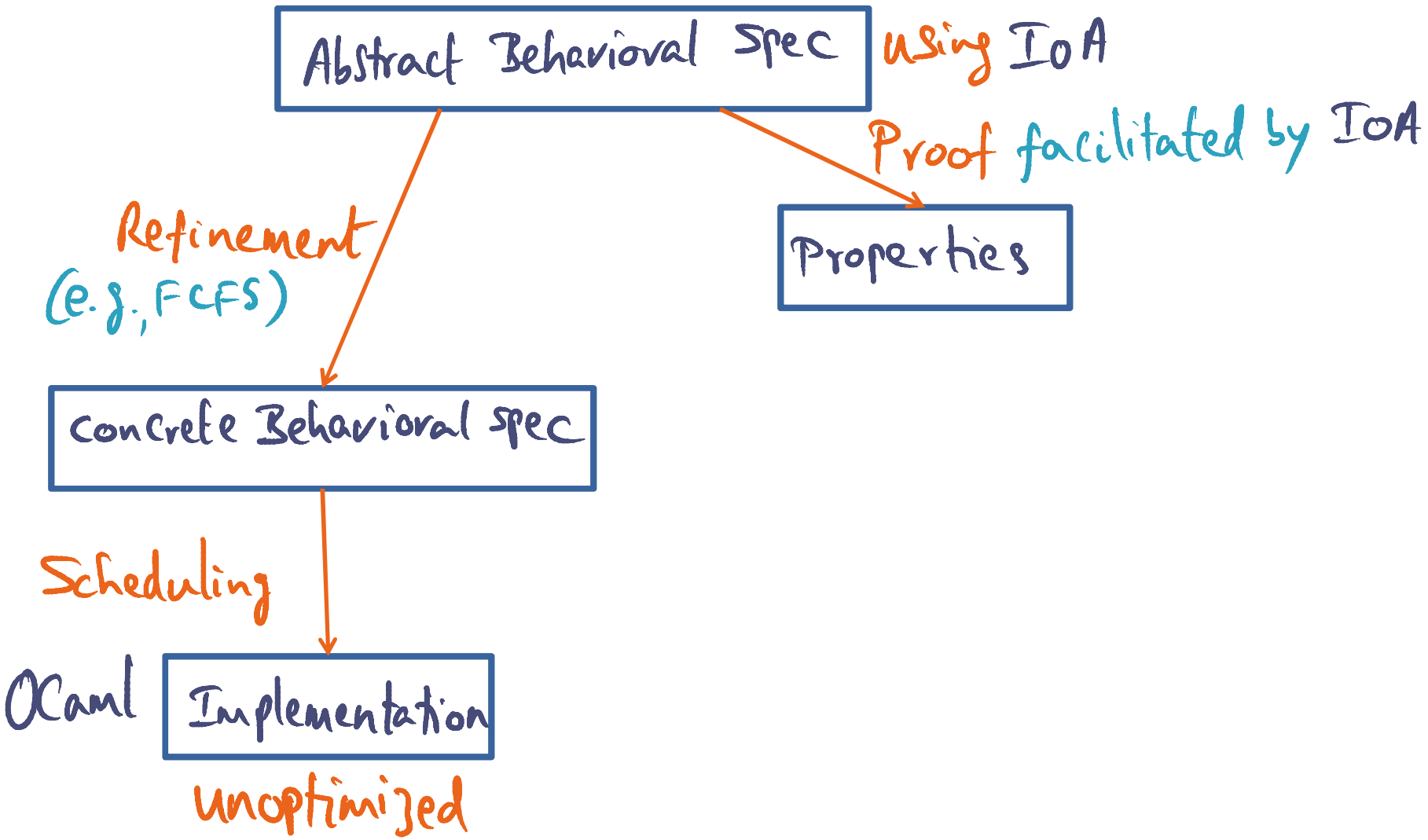
Therkath + Levy ✓
- * Application interface to the Kernel
- * Inside the Kernel

Wetherall ✓
- * End-to-End QoS via Active Networks

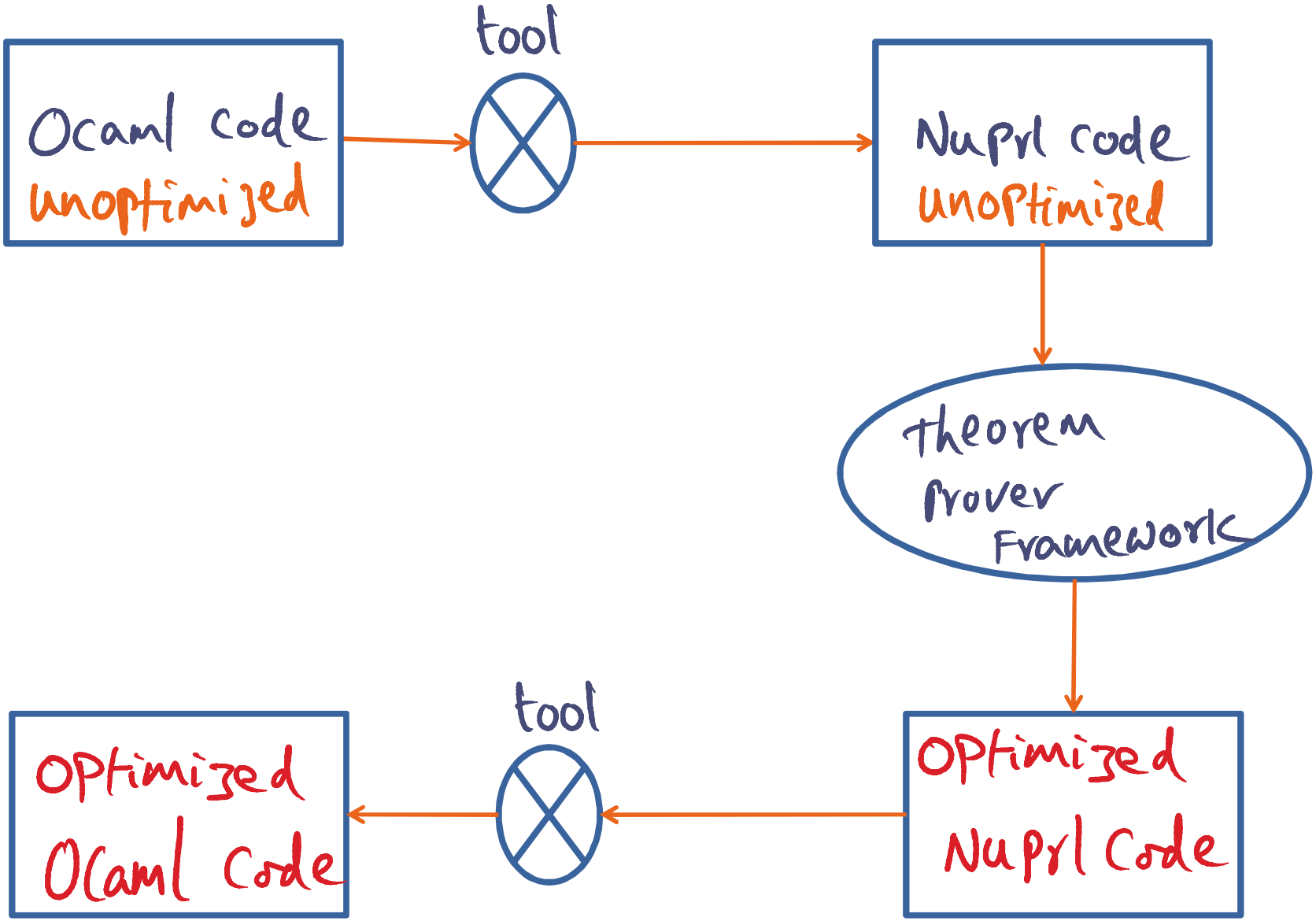
⇒ Synthesizing Network protocol Stacks

⇒ SPring Kernel (Distr. Objects Lesson 6)

Digging Deeper – From Spec to Implementation

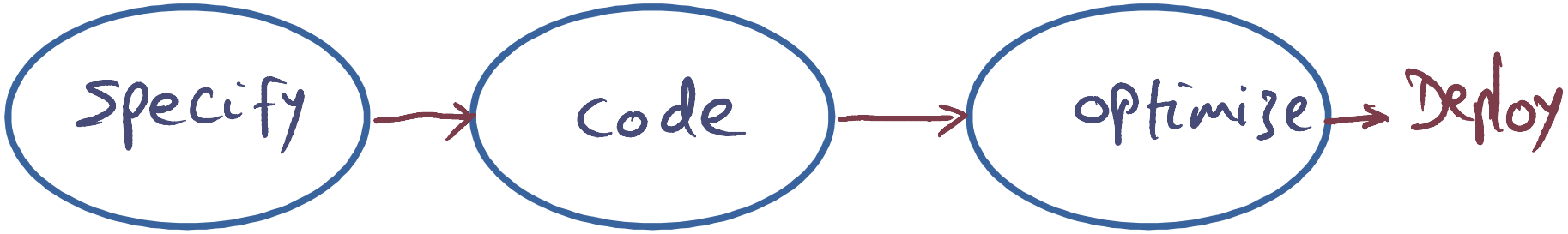


Digging Deeper – From Implementation to Optimization



Ensemble – Big Picture

Design Cycle



IOA
- C-like syntax
- composition operator

Ocaml
- object oriented
- efficient code similar to C
- nice complement to IOA

Nuprl
- optimization of Ocaml code
- output verified to be functionally equivalent

Putting this methodology to work

Start with IOA spec

- Abstract spec
- Concrete spec

Putting this methodology to work

Start with IOA spec

- Abstract spec
- concrete spec

How to synthesize the stack from concrete spec?

Putting this methodology to work

Start with IOA spec

- Abstract spec
- Concrete spec

How to synthesize the stack from concrete spec?

Getting to an unoptimized OCaml implementation

- Ensemble suite of microprotocols

Putting this methodology to work

Start with IOA spec

- Abstract spec
- Concrete spec

How to synthesize the stack from concrete spec?

Getting to an unoptimized OCaml implementation

- Ensemble suite of microprotocols
  - * flow control, sliding window, encryption, scatter/gather, etc.

Putting this methodology to work

Start with IOA spec

- Abstract spec
- Concrete spec

How to synthesize the stack from concrete spec?

Getting to an unoptimized Ocaml implementation

- Ensemble suite of microprotocols
  - * flow control, sliding window, encryption, scatter/gather, etc.
  - * well defined interfaces allowing composition
  - * facilitates component based design

Putting this methodology to work

Start with IOA spec

- Abstract spec
- Concrete spec

How to synthesize the stack from concrete spec?

Getting to an unoptimized OCaml implementation

- Ensemble suite of microprotocols
  - * flow control, sliding window, encryption, scatter/gather, etc.
  - * well defined interfaces allowing composition
  - * facilitates component based design

Recall original goal: mimic VLSI design

How to optimize the Protocol stack ?

How to optimize the Protocol stack ?

Layering Could lead to inefficiencies

How to optimize the Protocol stack ?

Layering Could lead to inefficiencies

- Analogy to VLSI component-based design breaks down for software

How to optimize the protocol stack?

Layering could lead to inefficiencies

- Analogy to VLSI component-based design breaks down for software

Several sources of optimization possible

- explicit memory management instead of implicit GC
- Avoid marshaling/unmarshaling across layers
- Buffering in parallel with transmission
- Header compression
- locality enhancement for common code sequences

How to optimize the protocol stack?

Layering could lead to inefficiencies

- Analogy to VLSI component-based design breaks down for software

Several sources of optimization possible

- explicit memory management instead of implicit GC
- Avoid marshaling/unmarshaling across layers
- Buffering in parallel with transmission
- Header compression
- locality enhancement for common code sequences

But how to automate the process?

Nuprl to the rescue

Static opt

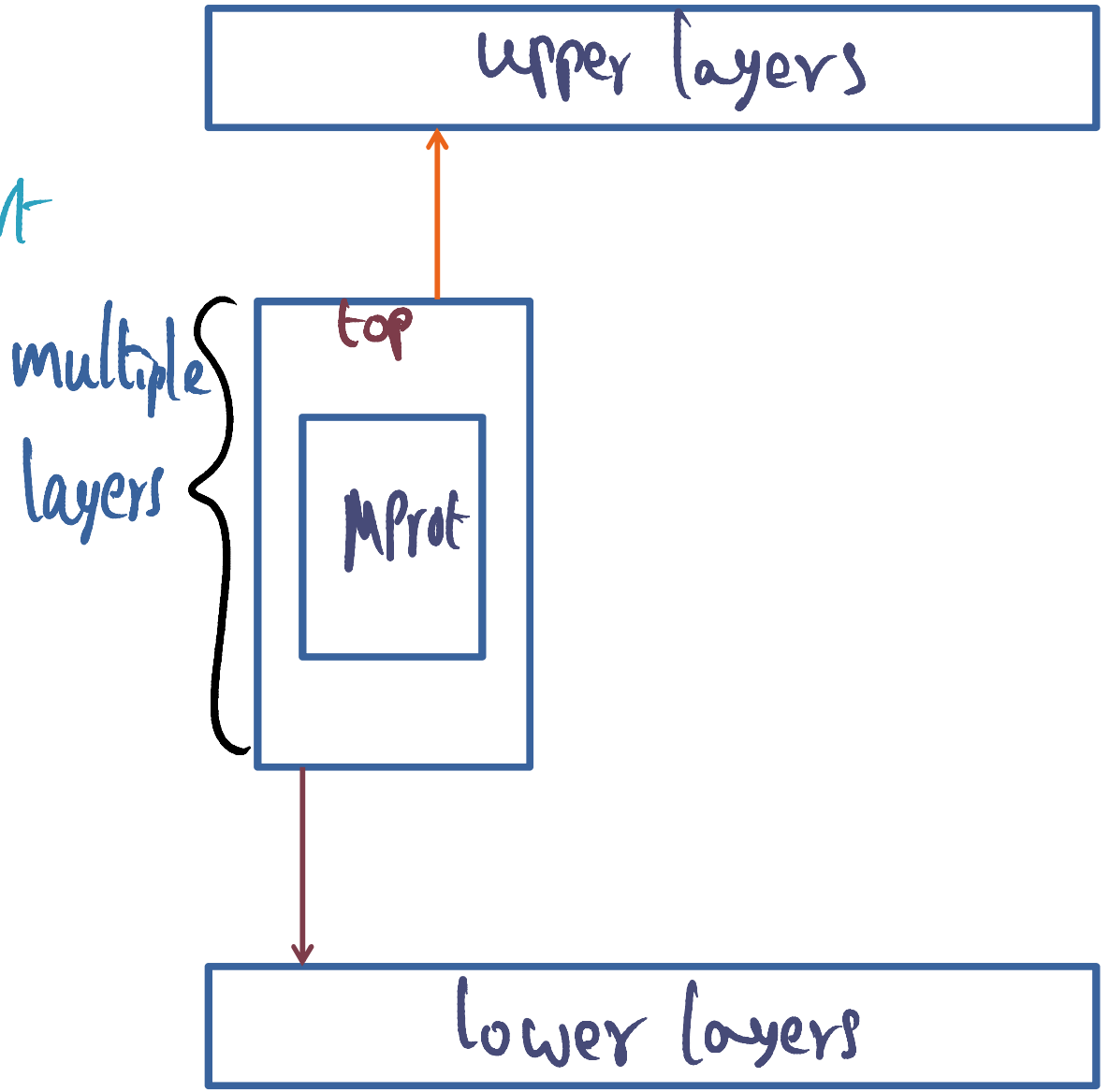
- Nuprl expert

  plus Ocaml expert

- layer by layer

Nuprl to the rescue

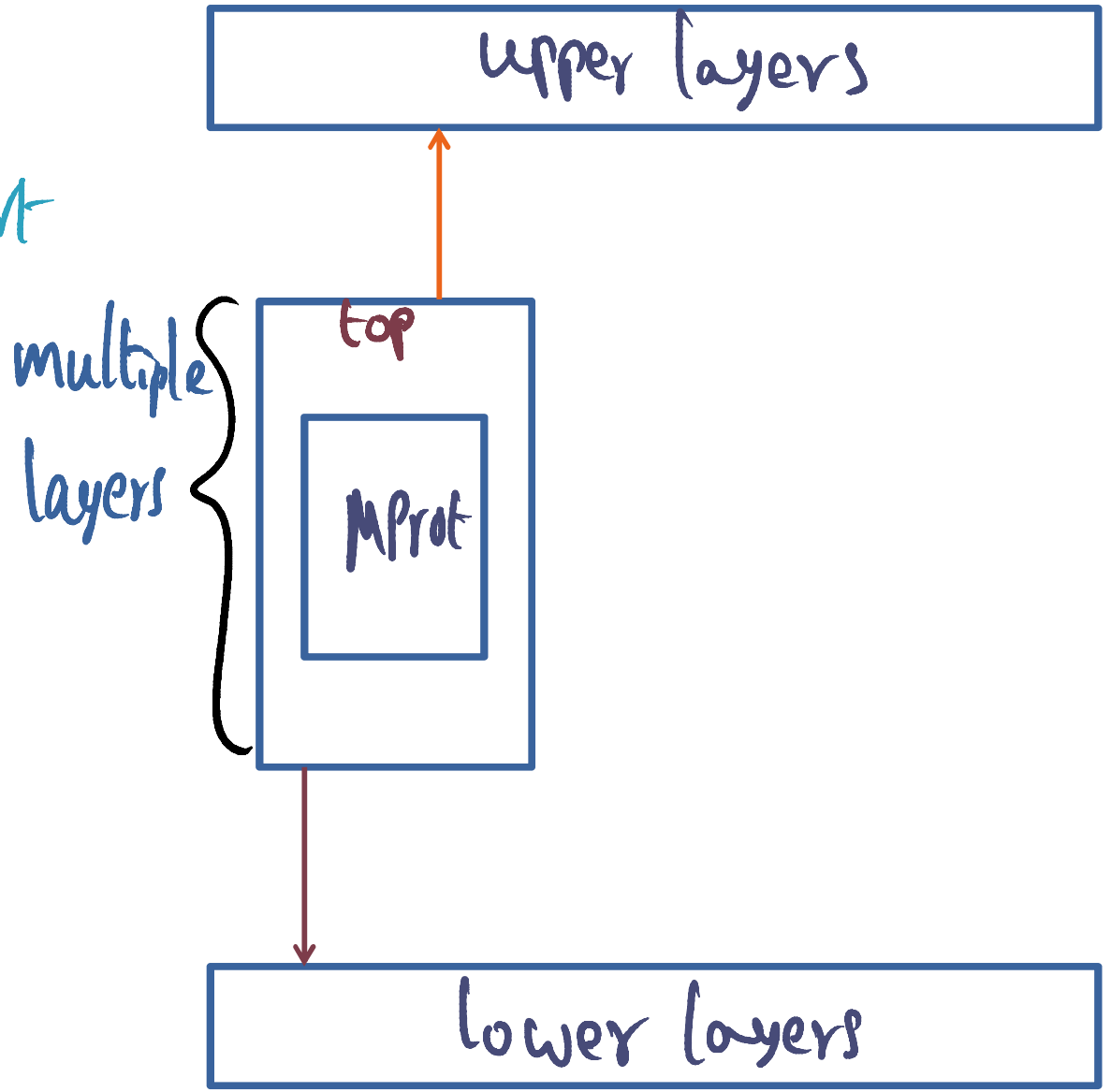
Static opt
- Nuprl expert
  plus Ocaml expert
- layer by layer



Nuprl to the rescue

Static opt
- Nuprl expert
  plus Ocaml expert
- layer by layer

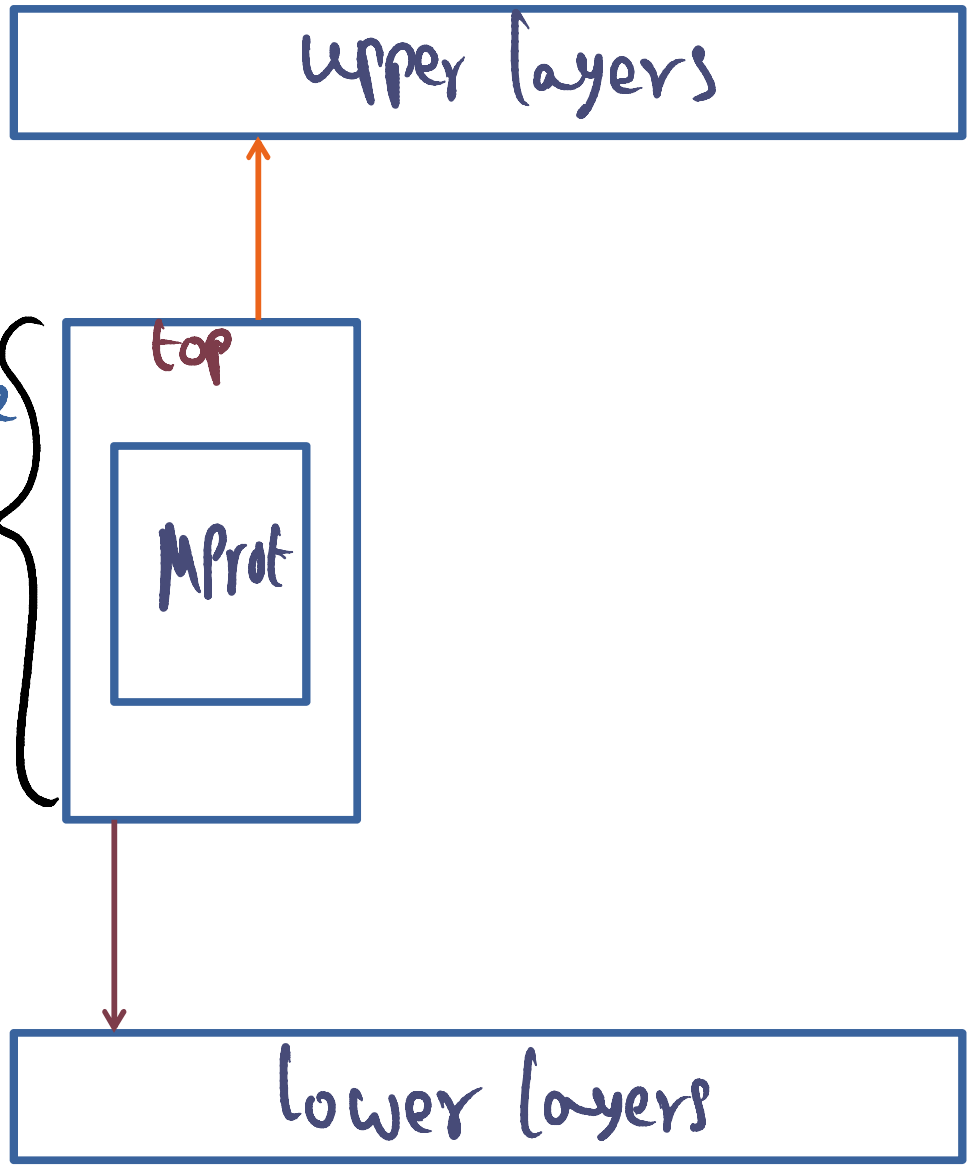
Dynamic opt
- Collapse layers



Nuprl to the rescue

Static opt
- Nuprl expert
  plus Ocaml expert
- layer by layer

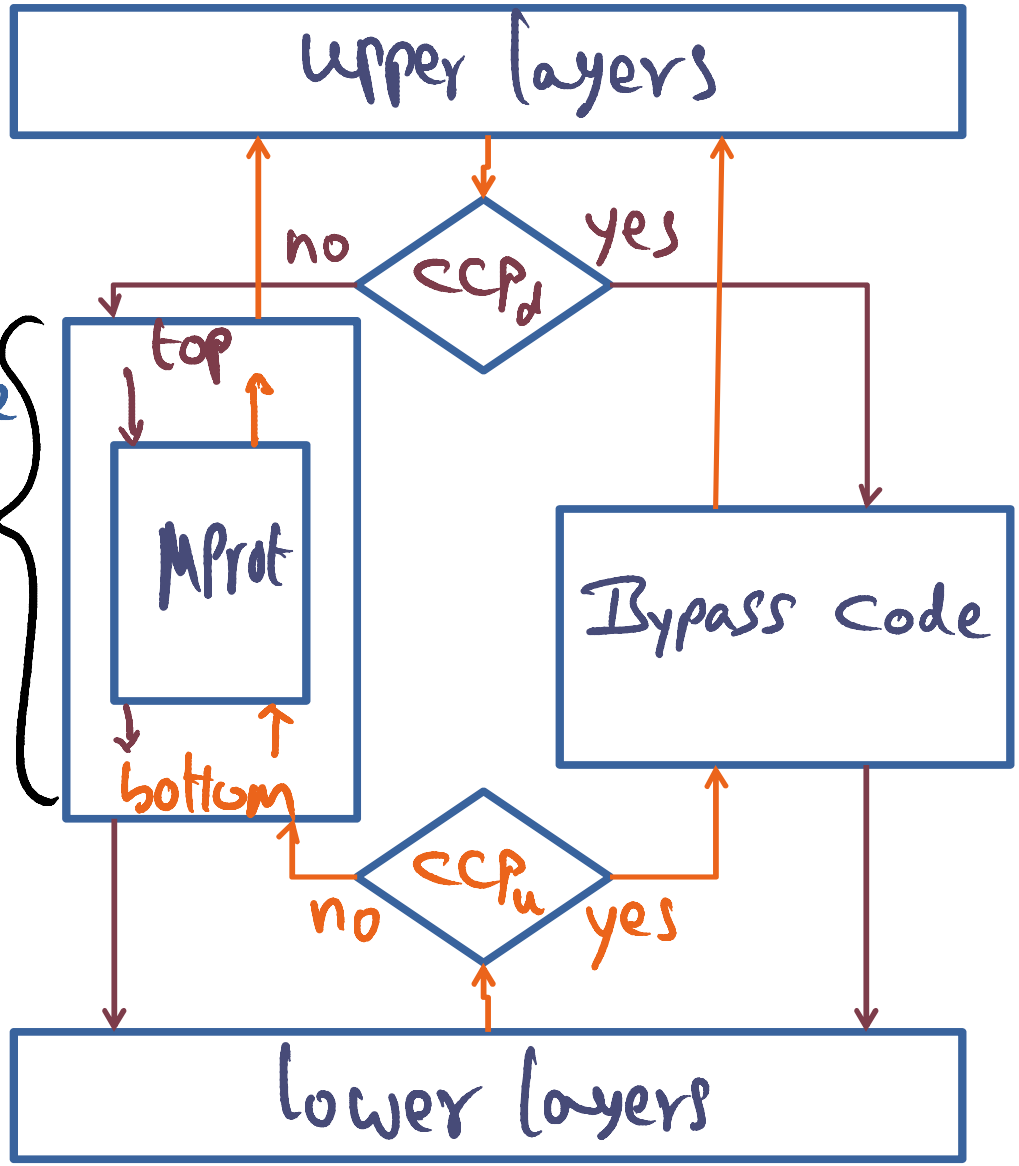
Dynamic opt
- Collapse layers
- generate bypass
  code if CCP satisfied



Nuprl to the rescue

Static opt
- Nuprl expert
  plus Ocaml expert
- layer by layer

Dynamic opt
- Collapse layers
- generate bypass
  code if CCP satisfied

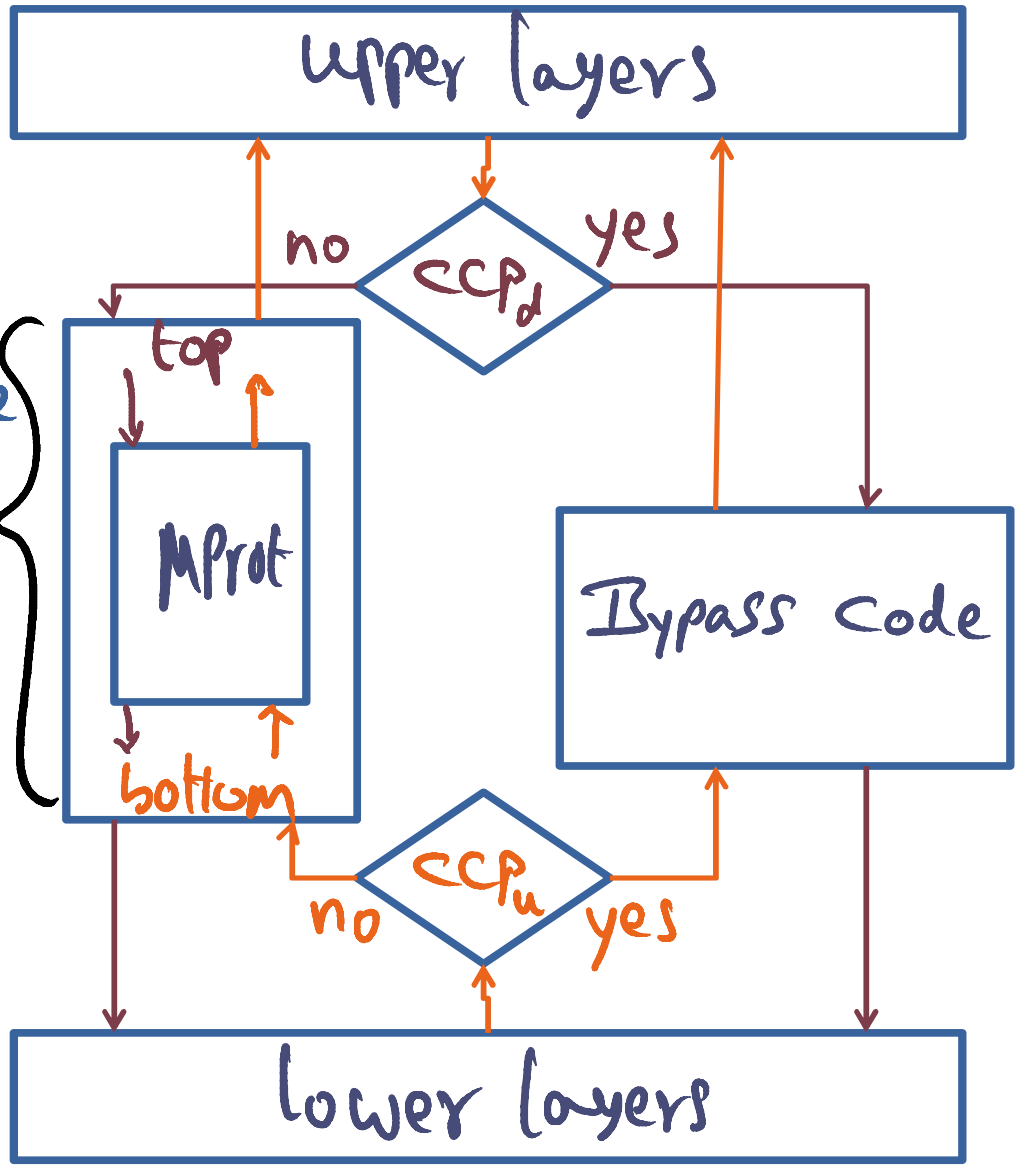


Nuprl to the rescue

Static opt
- Nuprl expert plus Ocaml expert
- layer by layer

Dynamic opt
- Collapse layers
- generate bypass code if CCP satisfied

Final step
- Convert back to Ocaml



Key Takeaways

- Do we lose out on performance for this convenience of component-based design?
  - Recall this is the same question that came up when we wanted to go for a micro-kernel based design of an OS
- This experiment concerns synthesizing individual subsystems of an OS from modular components
- Key to the success of this experiment is putting theory with practice
- Ensemble is one concrete example of putting this methodology to work for designing protocol stacks
  - Proof again by construction similar to Liedtke
- Specialization is not new (SPIN, Exokernel, Thekkath & Levy)
- Cornell experiment is achieving this at a subsystem level systematically