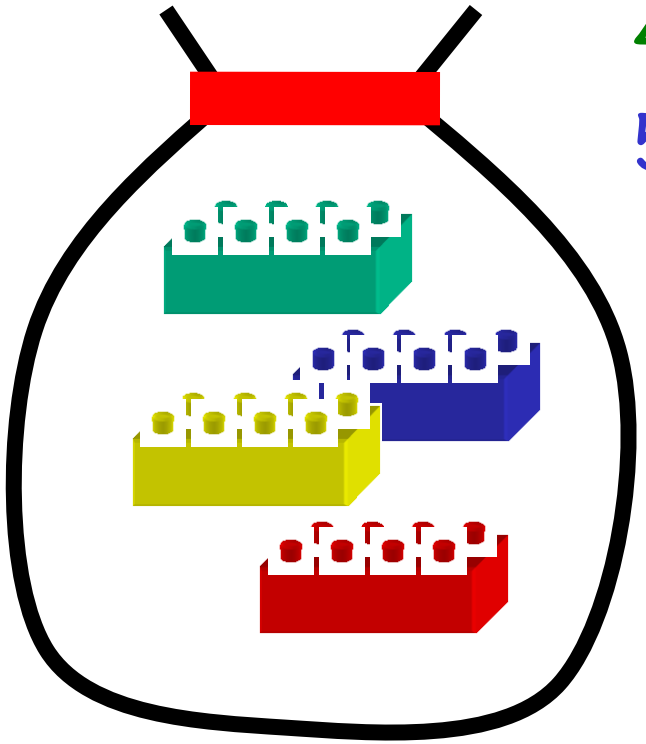
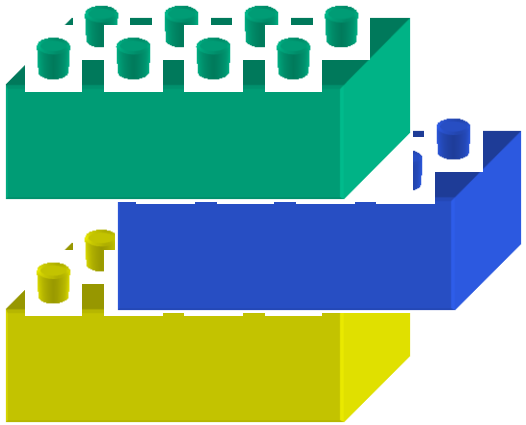
Why software components?

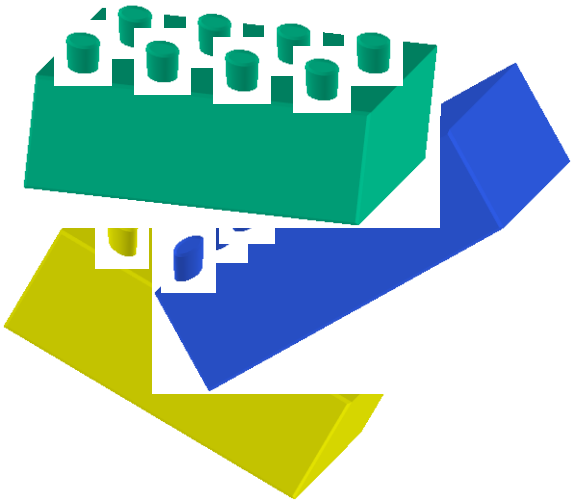
1. Ease design and development
2. Tuning to environment
3. Customization to user app
4. Extensibility
5. Verification and robustness





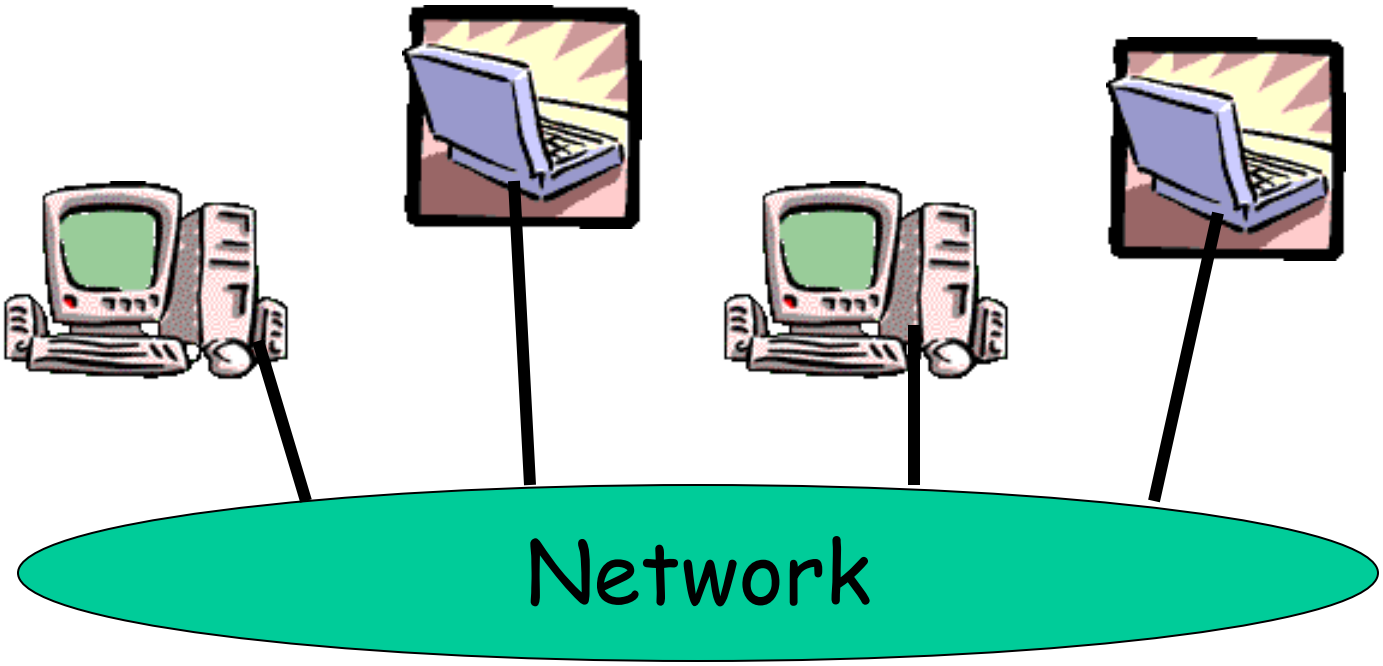
Why not?

- Configuration is hard
- Performance is bad
  - Abstraction barriers
  - Poor locality
  - Redundant code

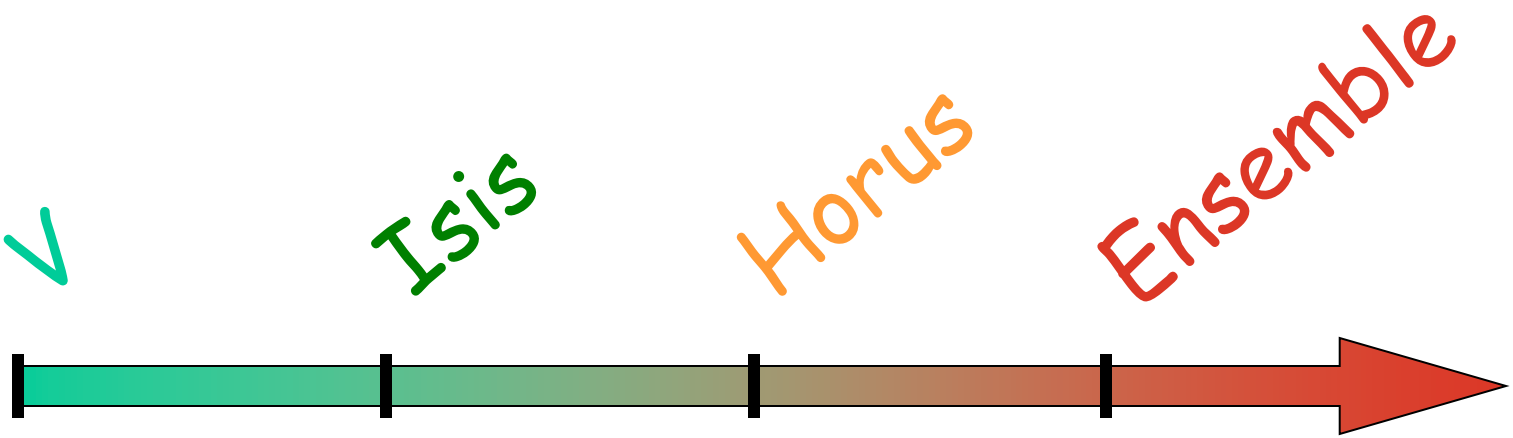


- *Not* reliable
- *Not* faster

Group Communication

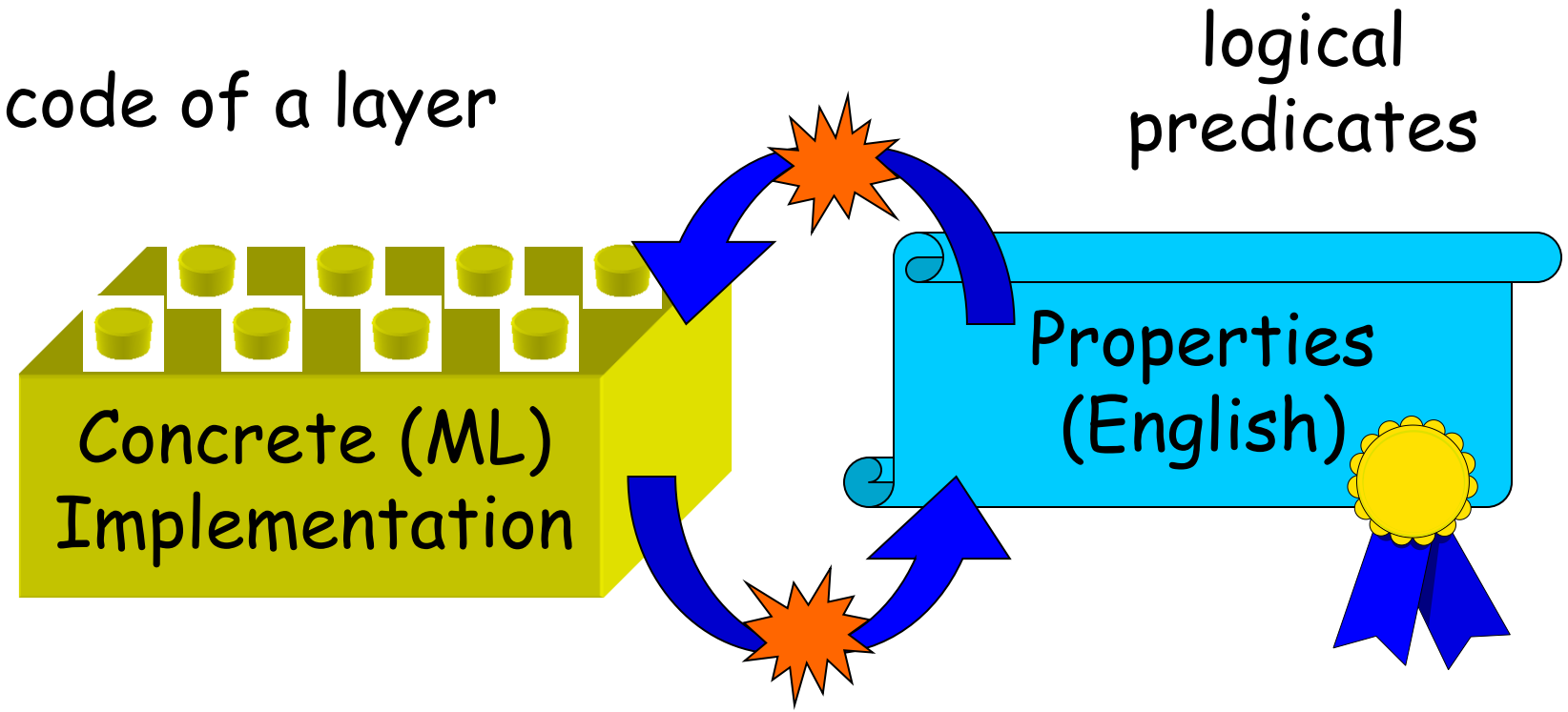


History

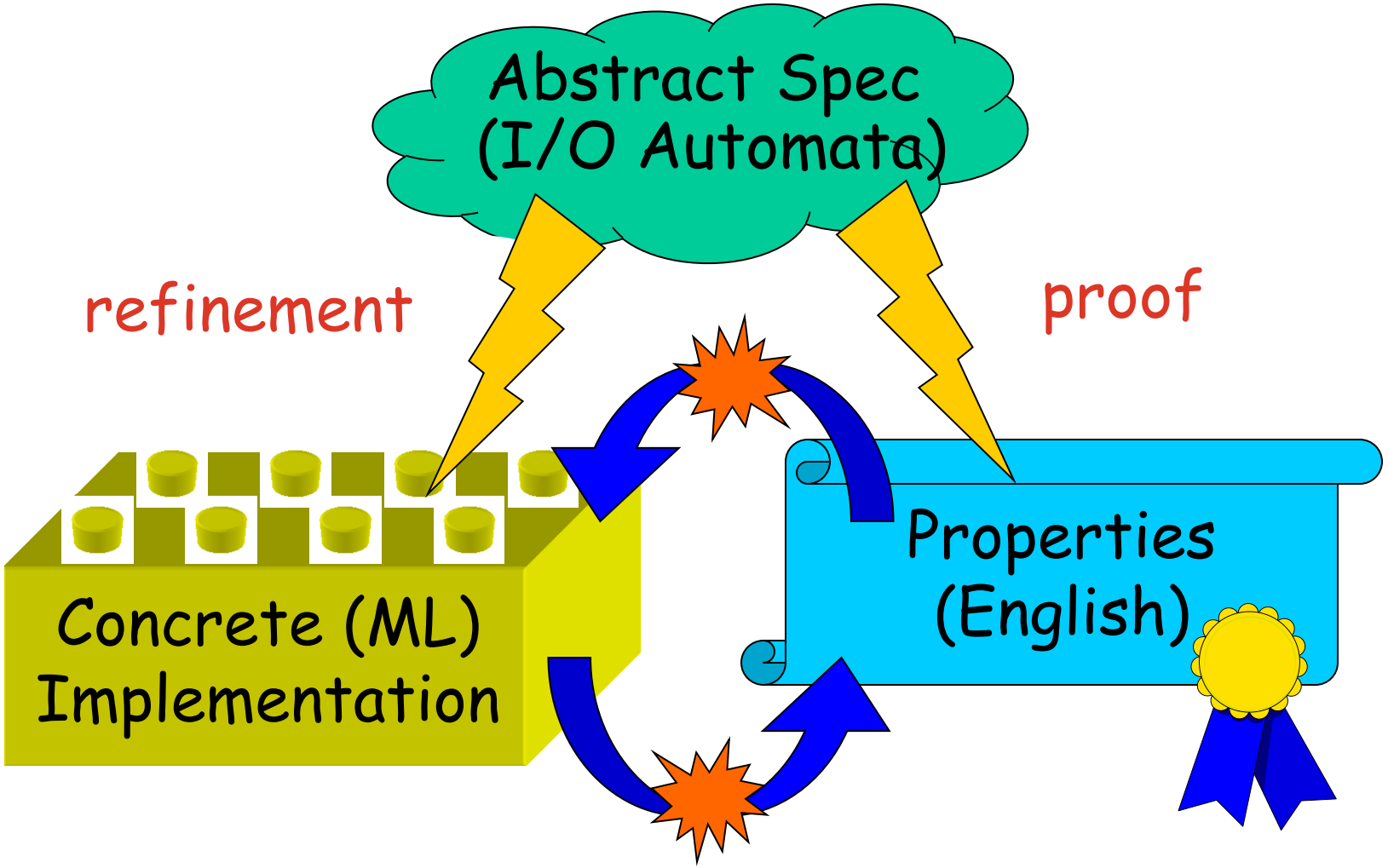


- Multicast
- Layers
- μProtocols
- Membership
- CATOCS
- ML
- Formal methods

Specification



Specification



Abstract IOA specification of totally ordered multicast

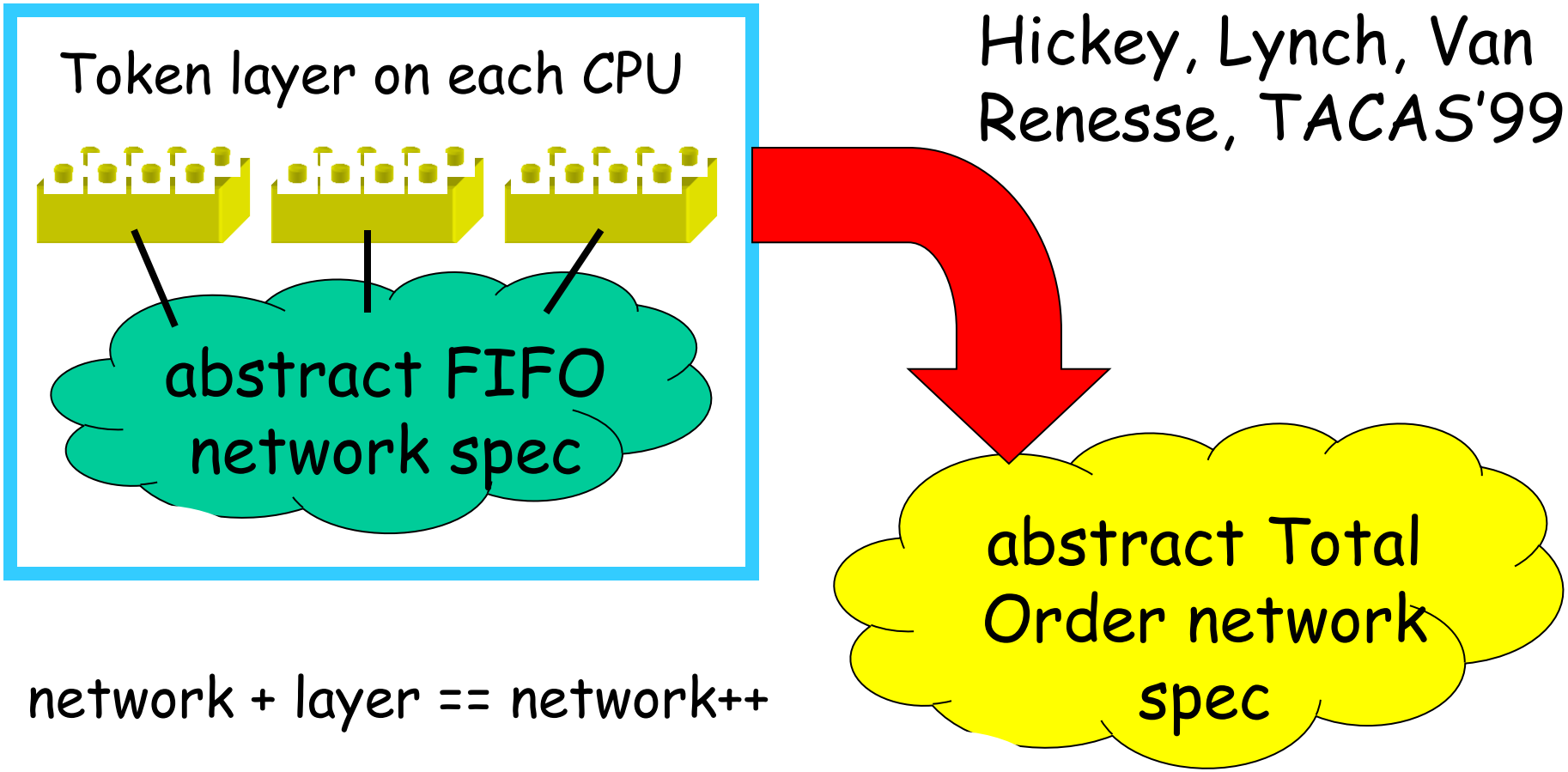
S: array[integer] of message
next: integer
deliv: array[process] of integer

} global state

action Multicast(m) { S[next++] := m; }

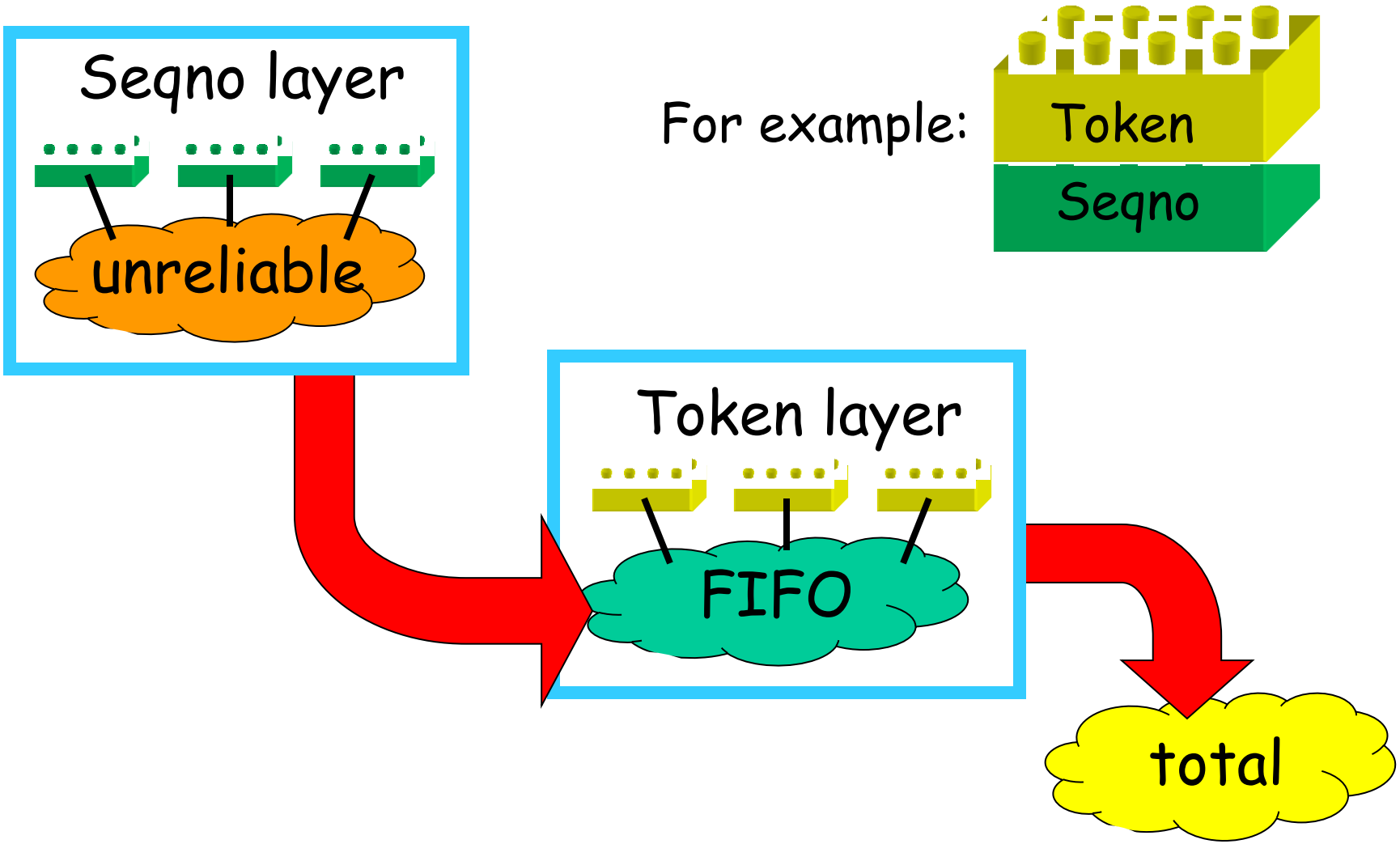
action Deliver(p, m)
precond: deliv[p] < next && m == S[deliv[p]]
  { deliv[p]++; }

Layer correctness



network + layer == network++

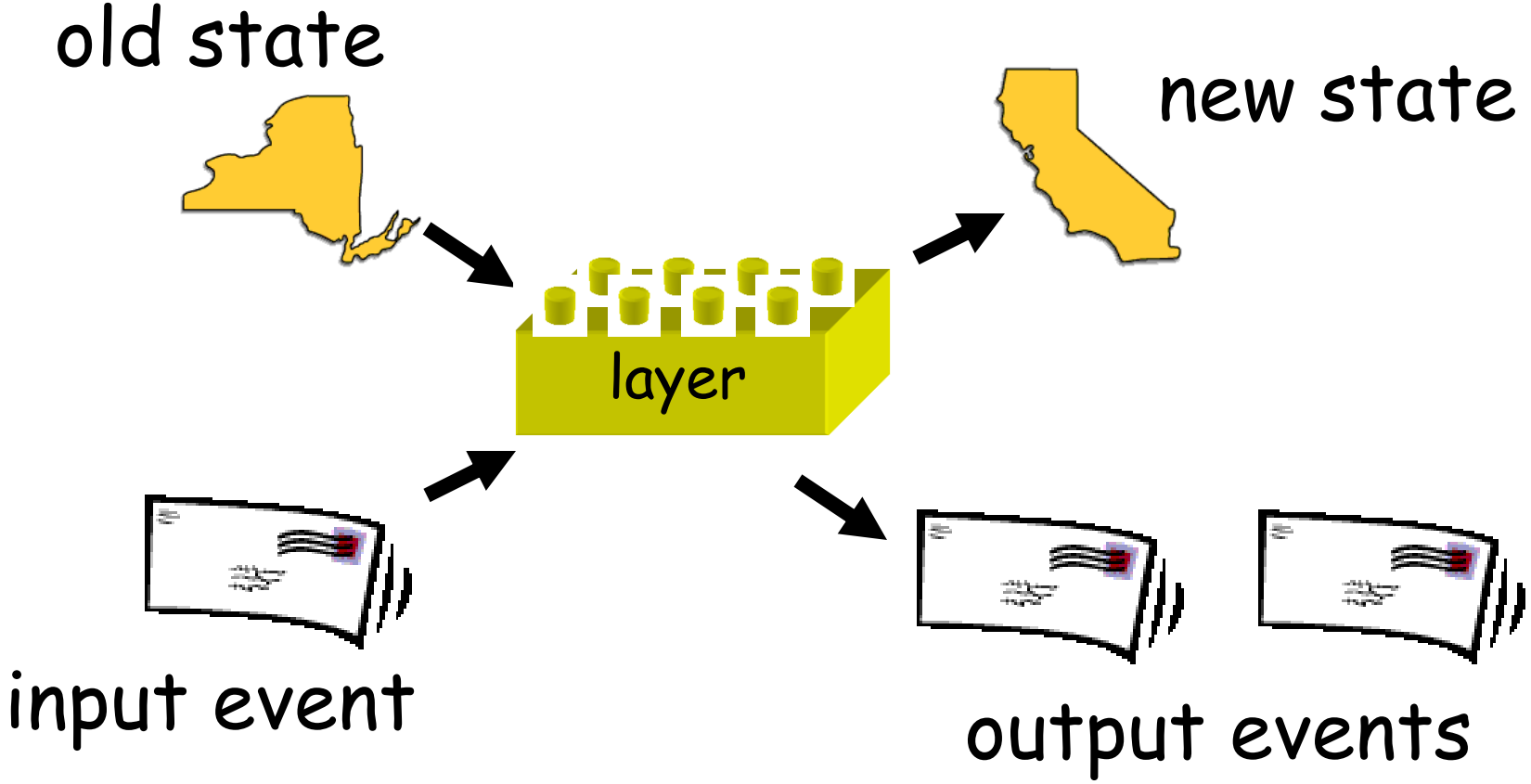
Stack correctness



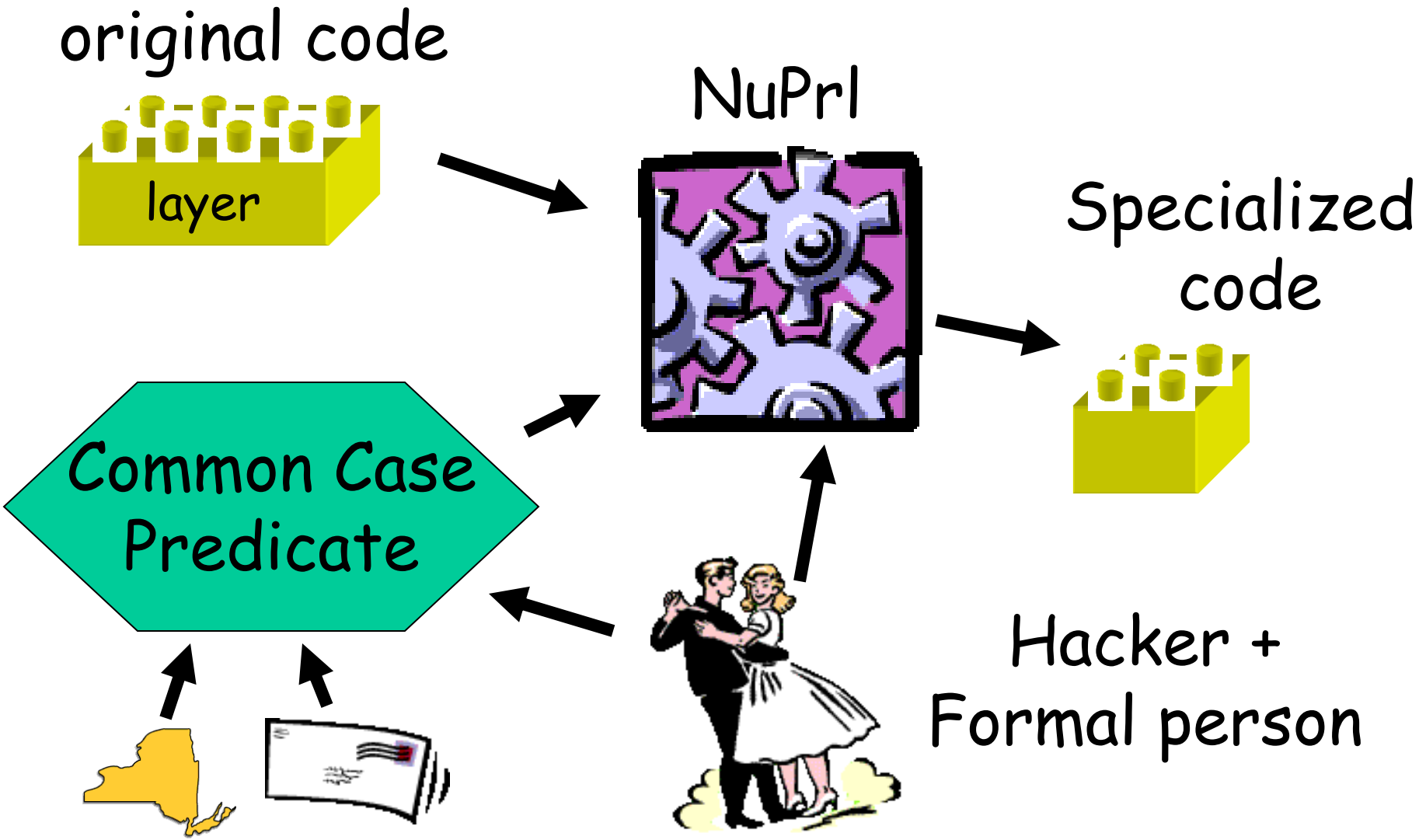
Efficiency?

- Ensemble stacks have many layers, improving clarity, but inefficient.
- 5 optimization techniques:
  1. Avoiding (in-line) garbage collection
  2. Avoiding marshaling
  3. Delaying non-critical message processing
  4. Identifying common paths
  5. Header compression

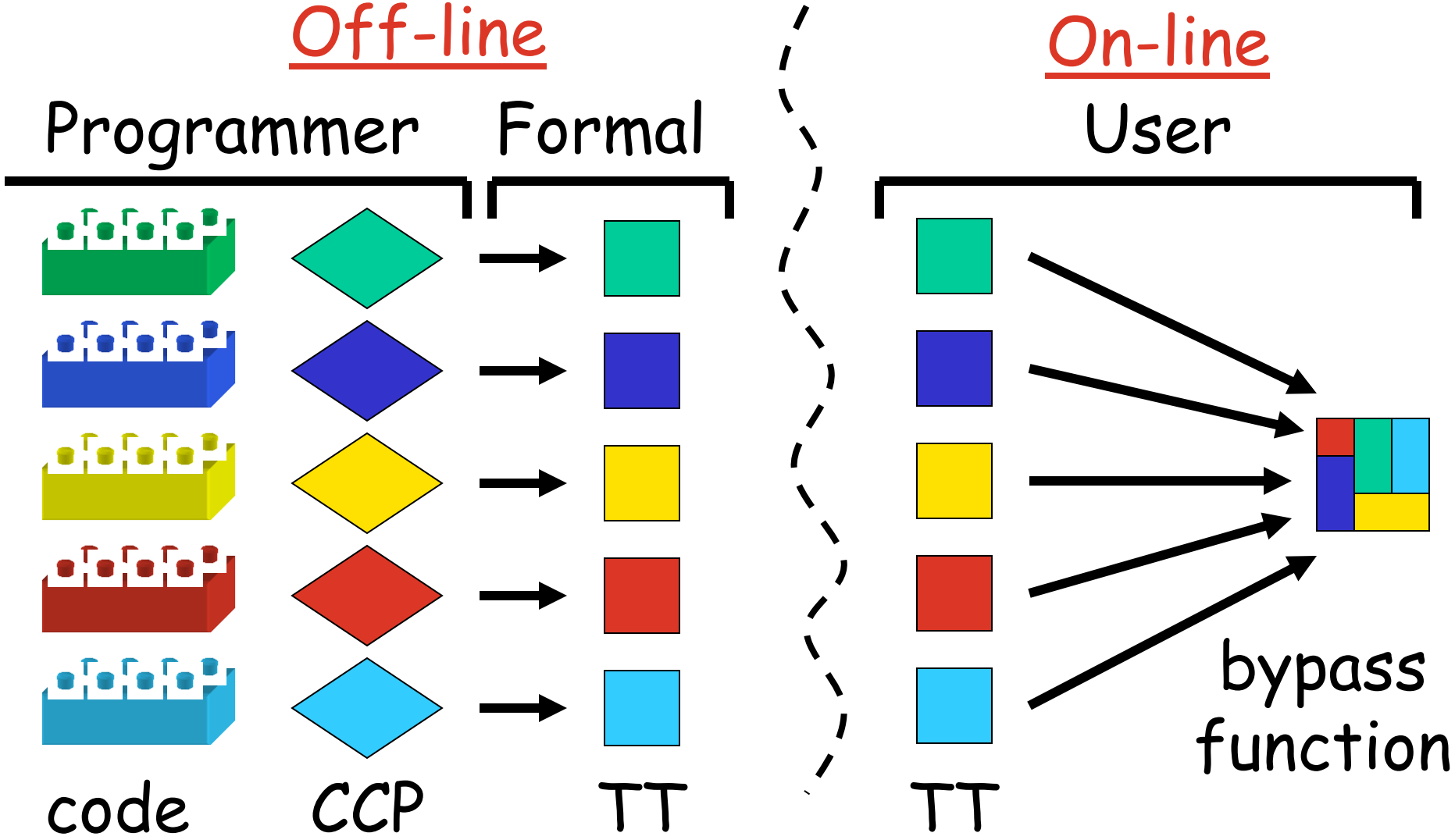
A protocol layer is a function!



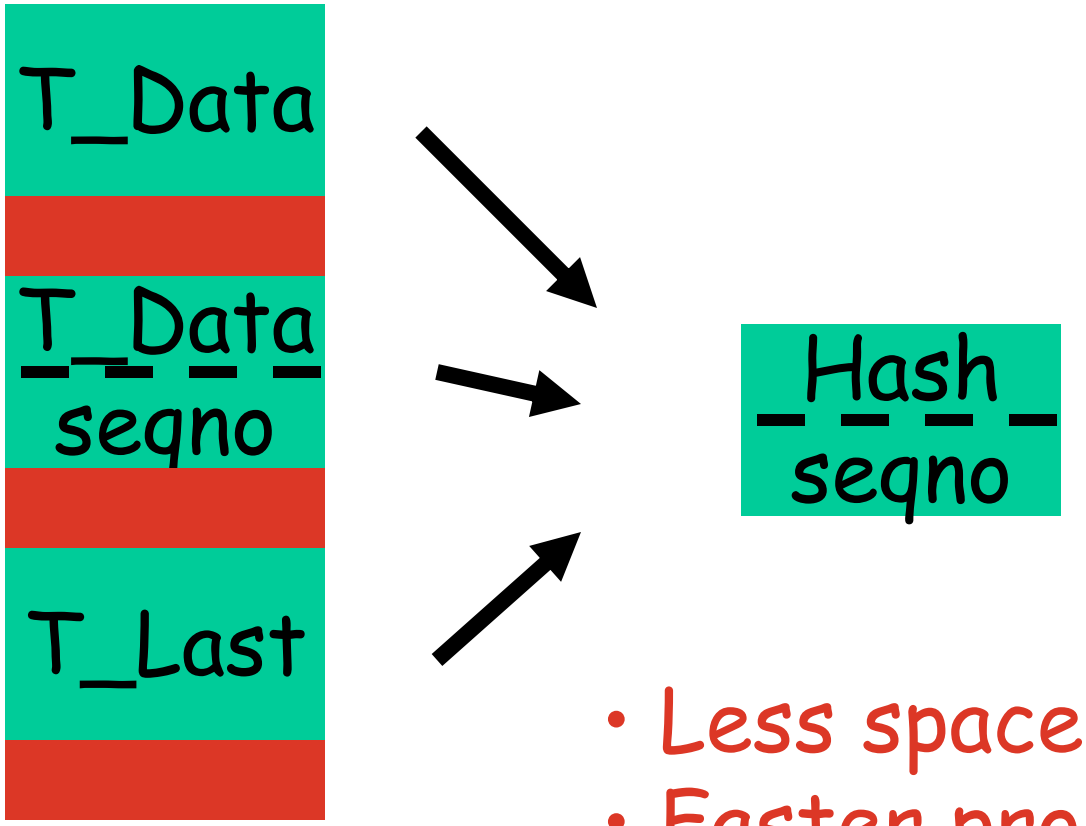
(off-line) partial evaluation



Two-phase optimization

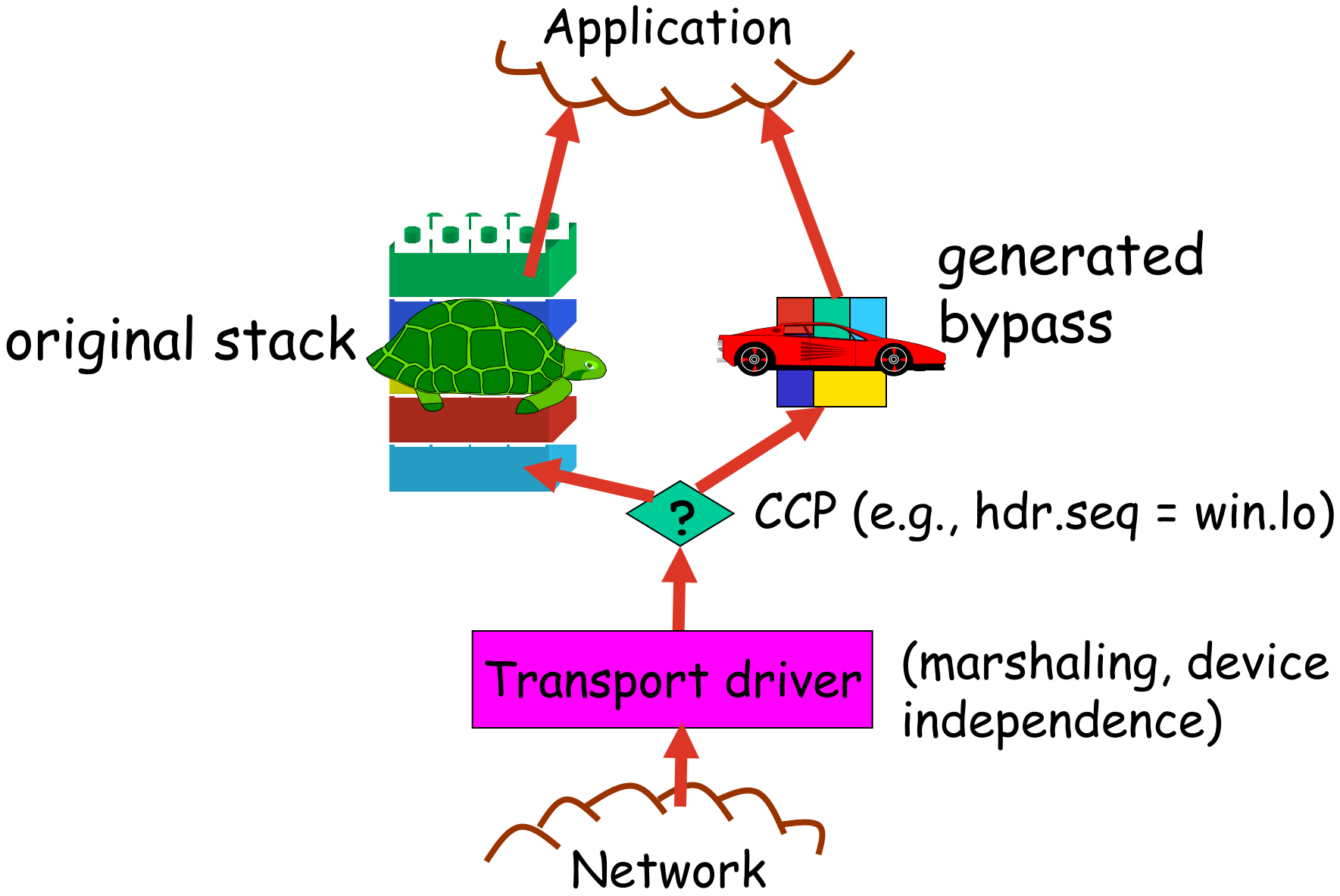


Header compaction

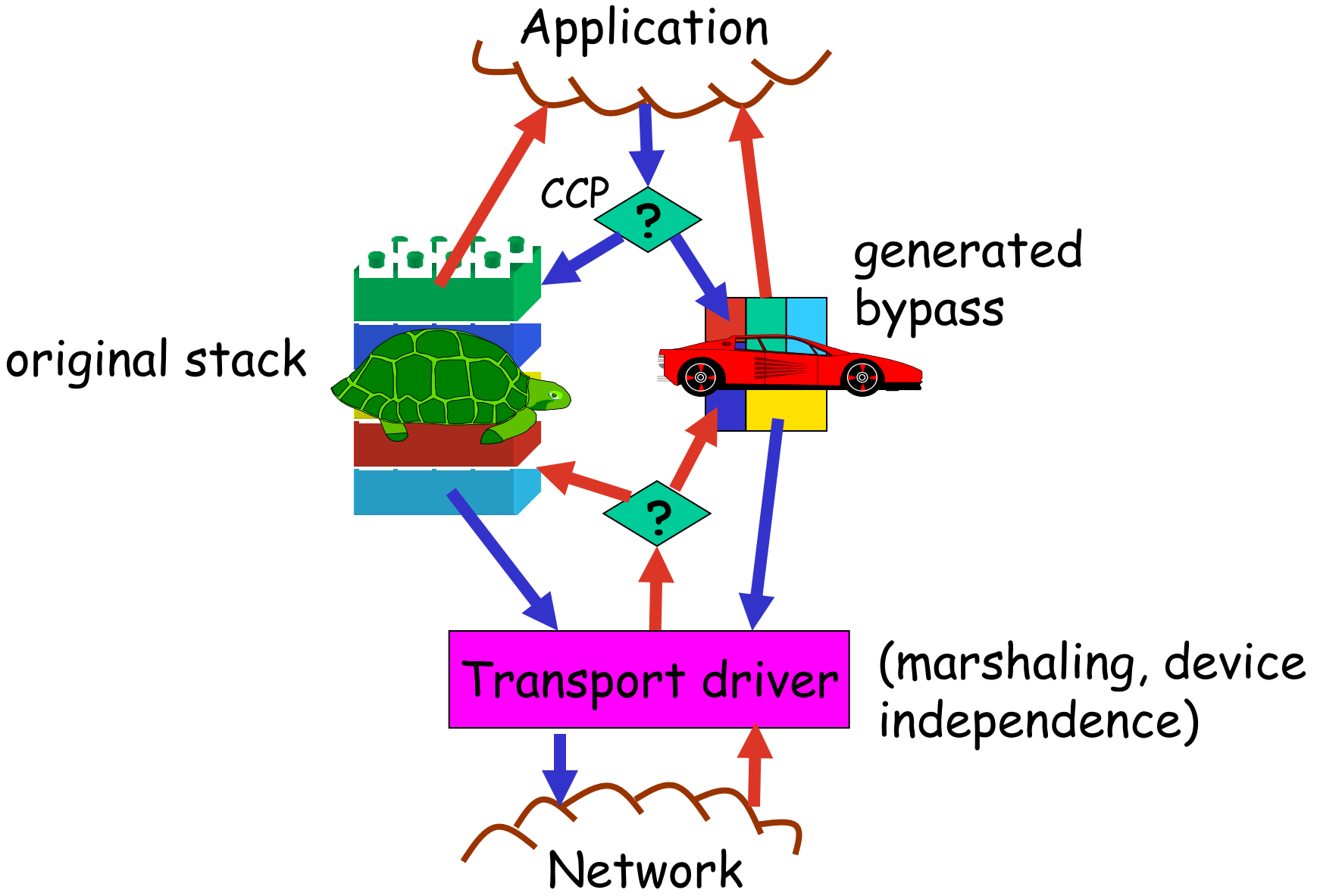


- Less space
- Faster processing

Architecture (deliver only)



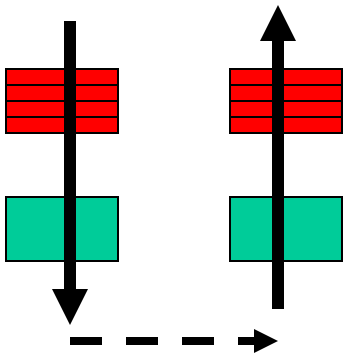
Architecture



Performance

- Three different versions:
  1. Original (ORIG)
  2. Hand-optimized (HAND)
  3. Machine-optimized (MACH)
- 300 MHz UltraSparc/Solaris 2.6
- OCaml 2.0 native code compiler

Code latency (μsec)



	10 layer		4 layer stack		
	ORIG	MACH	ORIG	MACH	HAND
Down Stack	20	9	13	2	2
Down Transport	27	8	6	6	+ 4
Up Transport	20	7	8	7	6
Up Stack	14	8	10	4	+ 2
Total	81	32	37	19	14

(See paper for CPU cycles and TLB misses)

Lessons learned

1. Design with formalization in mind
2. Use small, but not too small components
3. Use a language with formal semantics
4. Use IOA as a specification language
5. Use formal tool with in-house expertise

Final remarks

- See CD or Web for code samples, links to all code, as well as how to reproduce our results
- Still working on a machine-generated proof of correctness