Certified Web Services in Ynot:

Programming a web application in a proof assistant

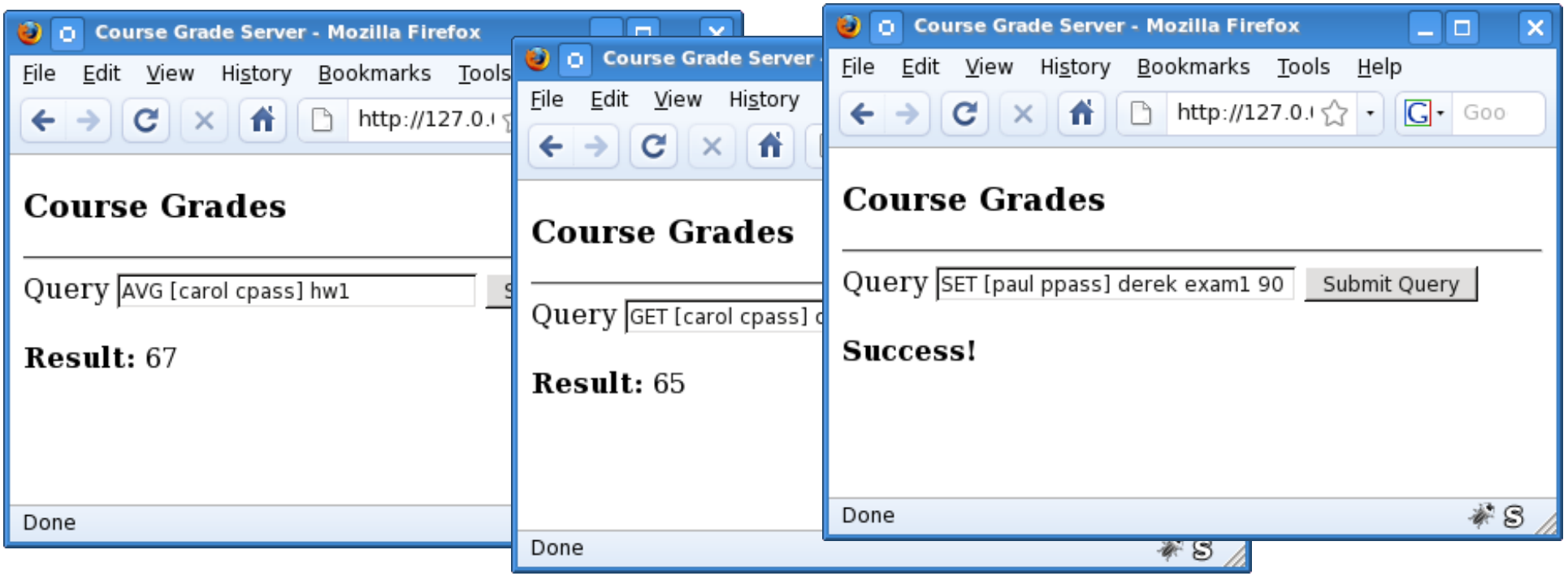
Ryan Wisnesky, Gregory Malecha, Greg Morrisett
Harvard University

Special thanks to Adam Chlipala

July 17, 2009

Goal

- Build a *course gradebook* with strong behavioral guarantees
  - Verify application logic, privacy, I/O behavior

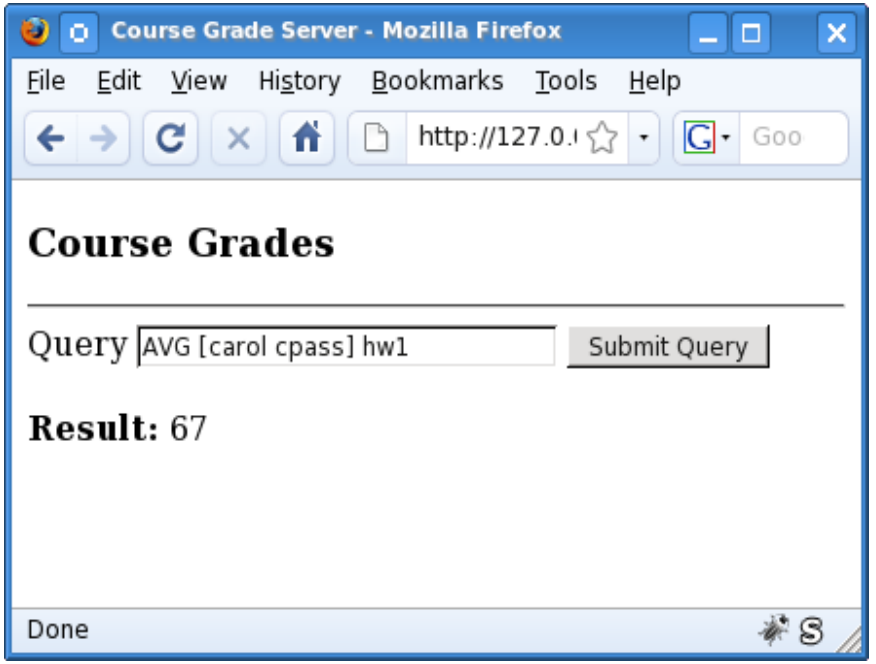


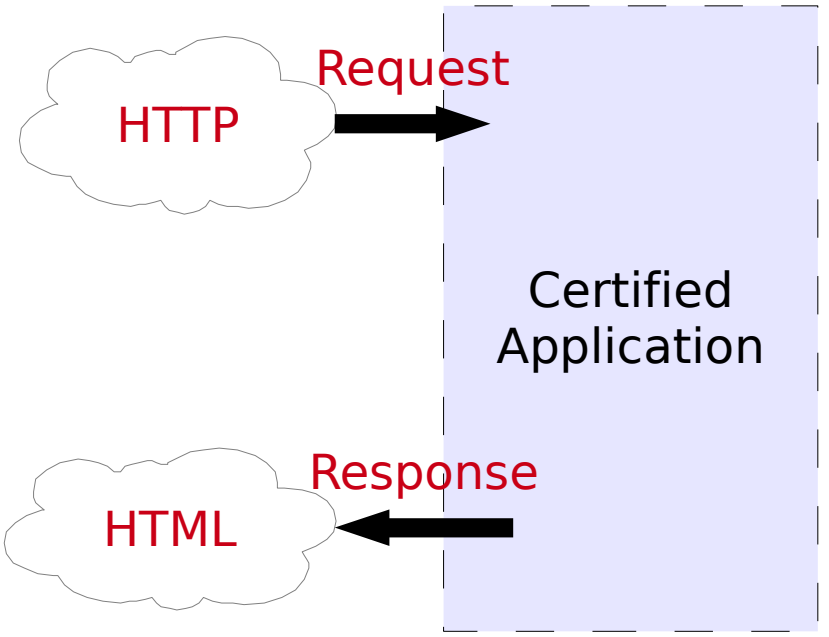
Gradebook Web Application

- Role-based access control:

	Read	Write	Average
Students	Self	None	All
TAs	Section	Section	All
Professors	All	All	All

```
/home>coqc gbsrv.coq
/home>gbsrv grades.dat
 Loading grades...
 Listening on port 80...
 Parsing request...
 Checking privacy...
 Computing class average...
```





Outline

- The Coq Language
  - Imperative programming in Coq
  - Our I/O and networking extensions
- The Gradebook server
  - Specification and implementation
  - Verification overhead

Coq

Core ML

+

Specifications as Types

=

Software that is correct by construction

Append in Coq

```
data List A = Nil
            | Cons A (List A)


++: ∀A, List A → List A → List A
Nil ++ L = L
(Cons a b) ++ L = Cons a (b ++ L)
```

Theorem Proving in Coq

```
Theorem append_associative:
 ∀ L1 L2 L3 ,
    (L1 ++ L2) ++ L3 = L1 ++ (L2 ++ L3).
Proof.
 Induction L1.
   ...
Qed.
```

Demo

Benefits of Coq

- Very small proof checker (100s of lines)
- Lightweight, pay as you go verification
- Specification, implementation, and proof of correctness written in the same language
- Extracts to ML

Limitations of Coq

- Coq code must be purely functional, terminating.
- But we need imperative features like general recursion and I/O.
  - Create a type of *imperative commands*
  - *Hoare logic* for reasoning about mutation
  - *Separation logic* for reasoning about memory

Swap

```
Definition swap (p1 p2 : ptr) (n1 n2 : Nat) :
  Cmd   (                    p1 → n1 * p2 → n2)
        (fun r : Unit => p1 → n2 * p2 → n1) :=
    v1 ← read p1  ;
    v2 ← read p2  ;
    write p1 ::= v2 ;
    write p2 ::= v1
```

Generate and solve proof obligations

Echo

```
Definition echo (lsock : Socket)
   (tr : Trace) :
   Cmd (traced tr)
       (fun r : Unit => ∃ msg rsock,
         traced [Sent lsock rsock msg ,
                 Recd lsock rsock msg ++
                      tr] ) :=
     (msg, rsock) ← recv sock ;
     send lsock msg rsock
```

Record events using traces.

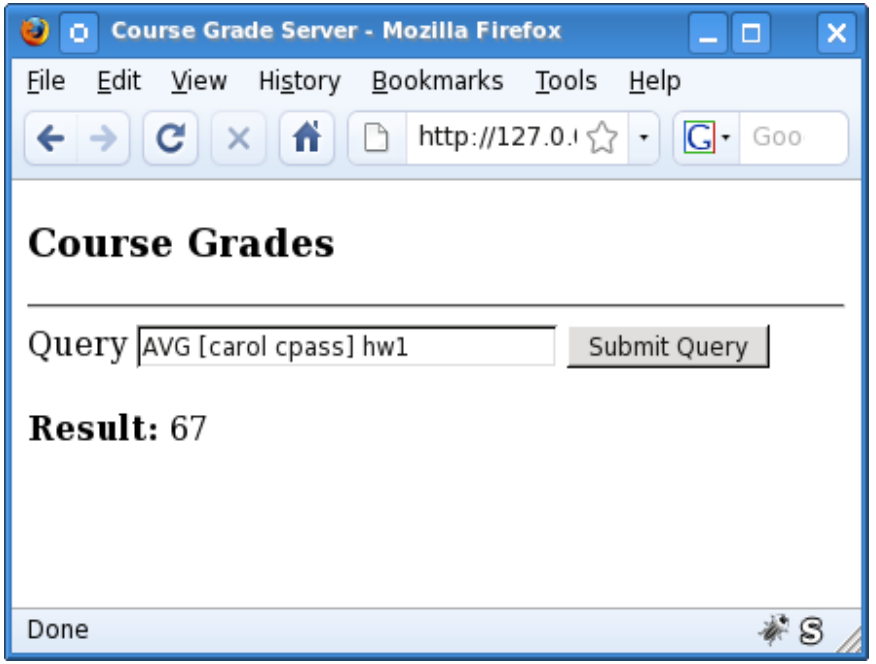
We do not verify...

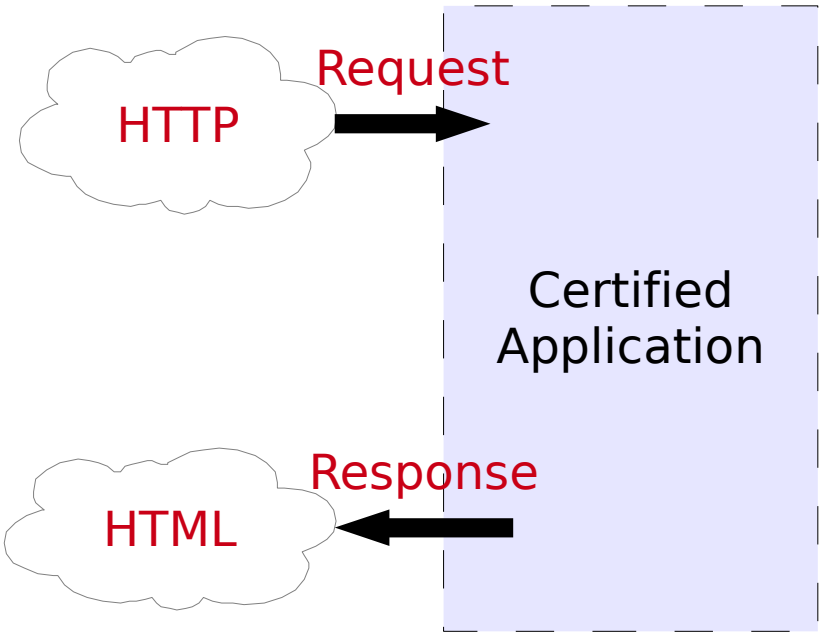
- Coq itself
- The ML implementations of our axioms
  - `read`, `write`, `recv`, `send`, etc
- Extraction from Coq to ML
- The ML compiler

Gradebook Web Application

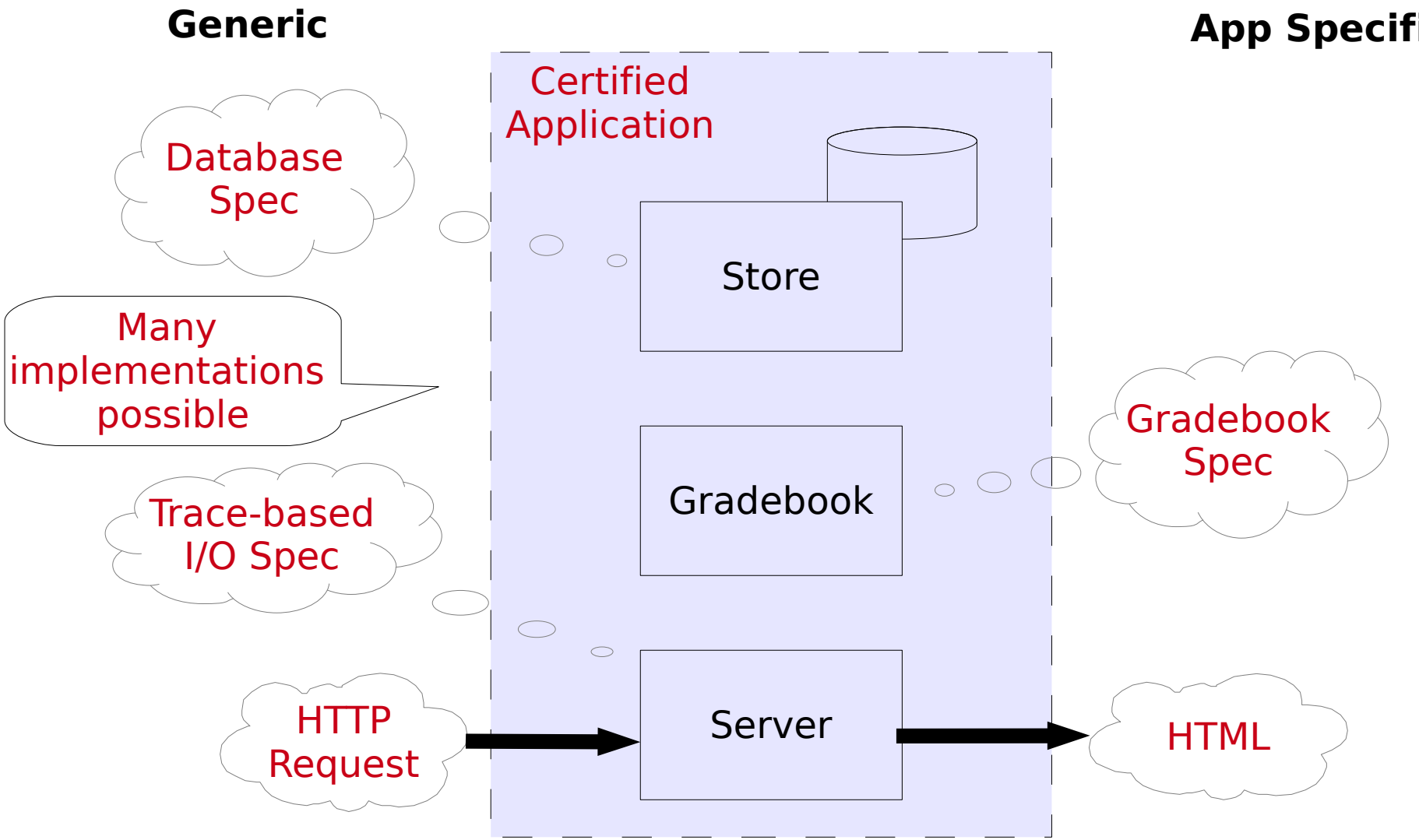
	Read	Write	Average
Students	Self	None	All
TAs	Section	Section	All
Professors	All	All	All

```
/home>coqc gbsrv.coq
/home>gbsrv grades.dat
 Loading grades...
 Listening on port 80...
 Parsing request...
 Checking privacy...
 Computing class average...
```

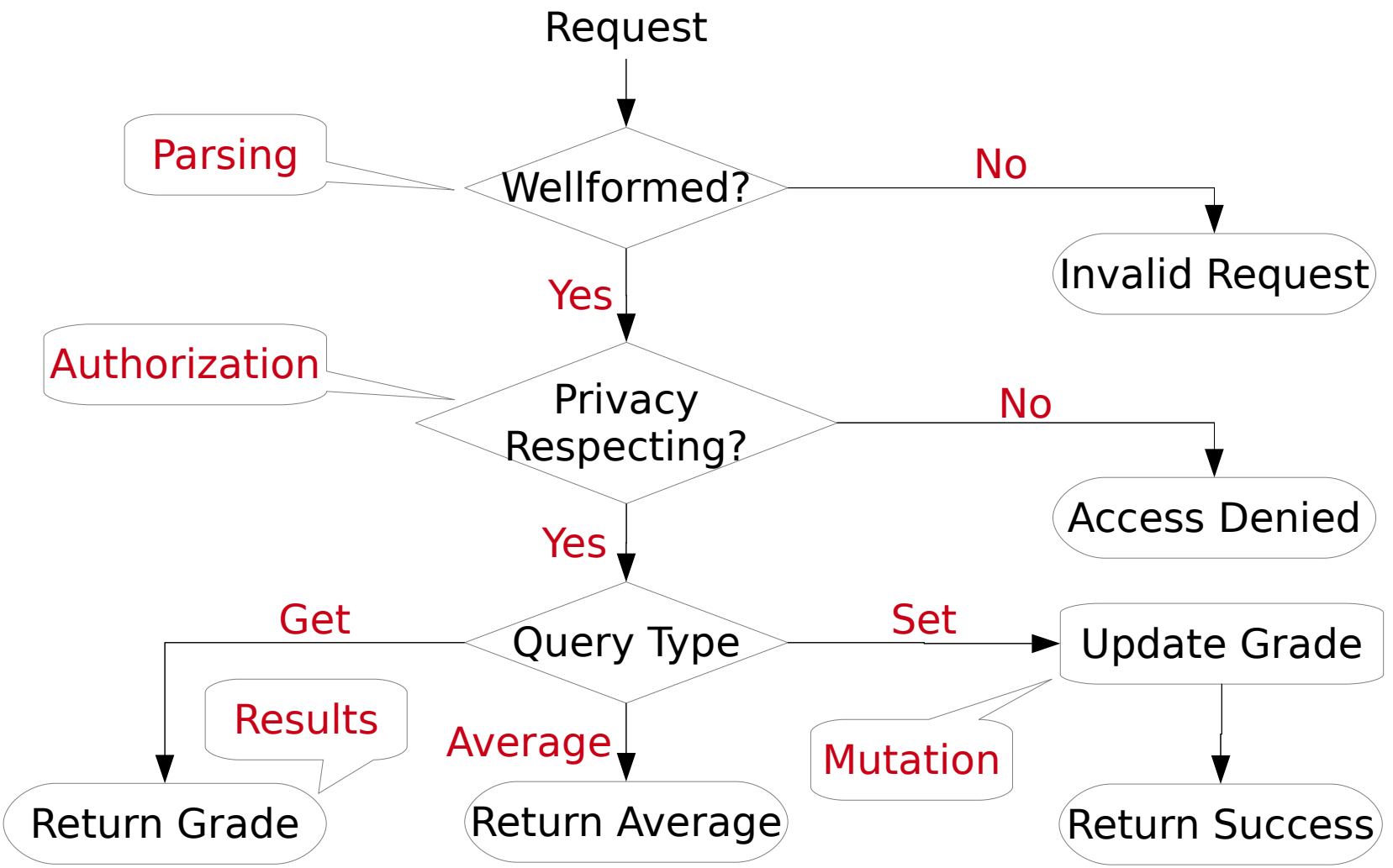




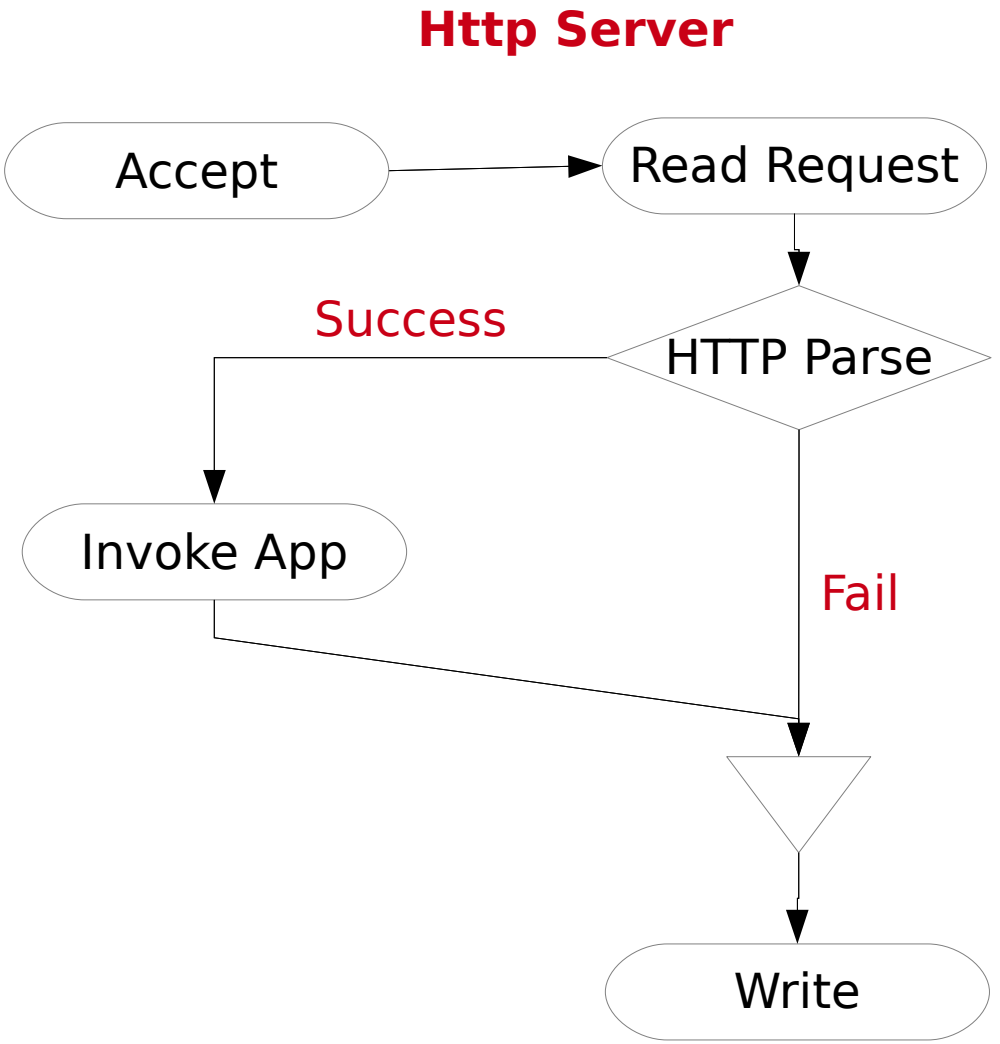
Architecture



Gradebook Specification



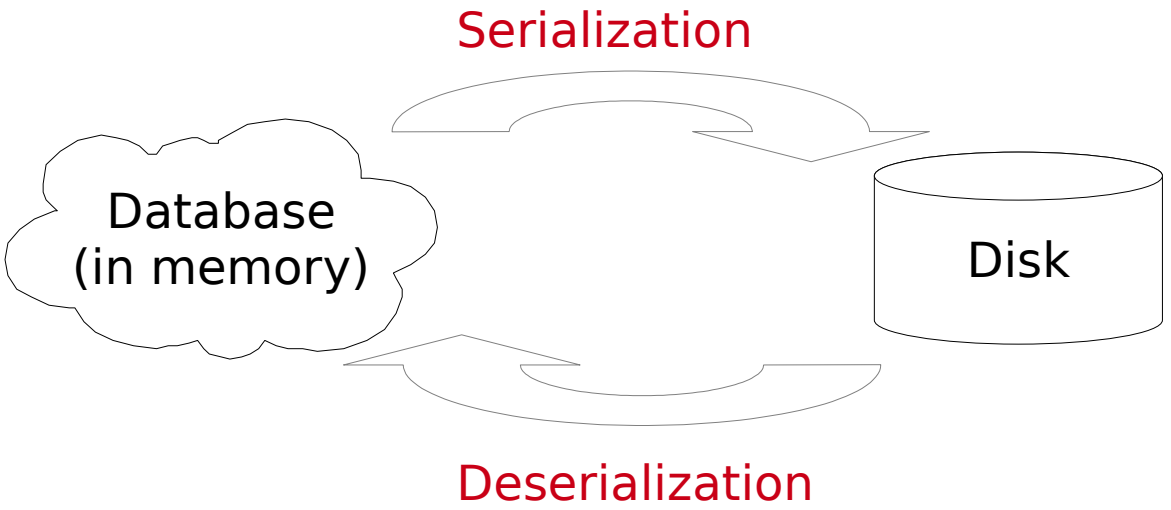
Application Server Spec



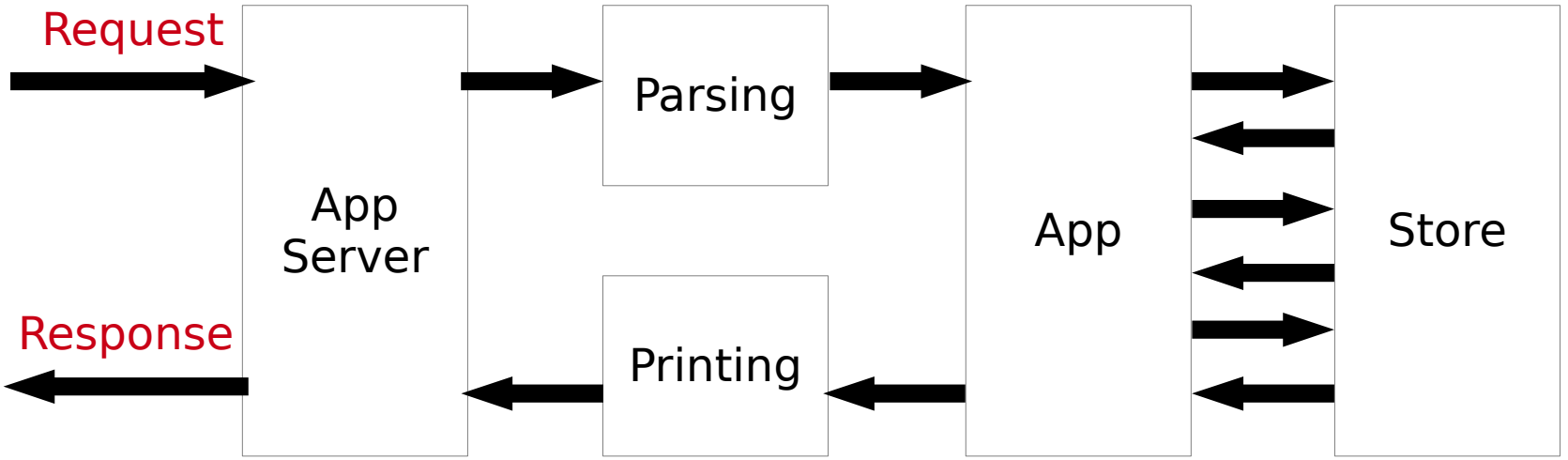
Store

- Queries: Select, update, etc
- Verify isomorphism between grades and tuples
- Linked-list implementation

Theorem: `deserialize (serialize x) = x`



Verification Overhead



	App Server	Parsing	App	Store
Specification (LOC)	414	184	231	154
Implementation	223	269	119	113
Proofs	231	82	564	99
Overhead	1.04	.3	4.74	.88
Compile-time (m:ss)	1:21	0:55	0:32	0:23

Conclusion

- You can program verified imperative software in Coq.
- Language level, “correctness by construction” techniques are scaling up.
- Future directions:
  - Concurrency
  - Other effects
  - Failure modes

ynot.cs.harvard.edu

Compilation