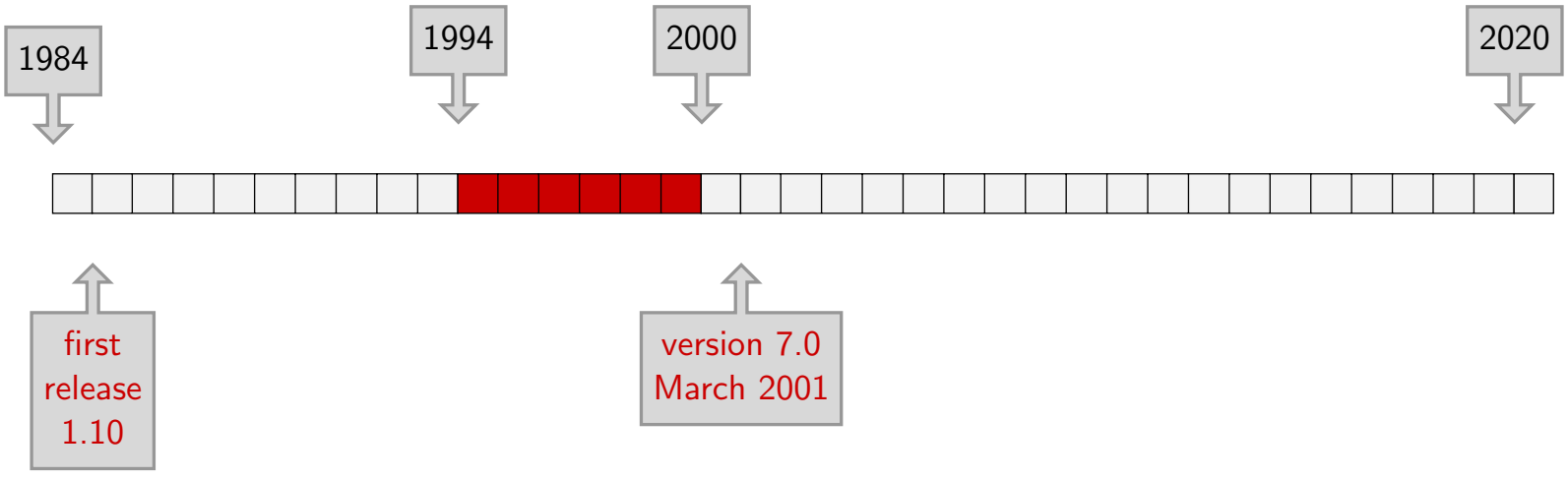
a Coq retrospective
—
at the heart of Coq architecture
the genesis of version 7.0

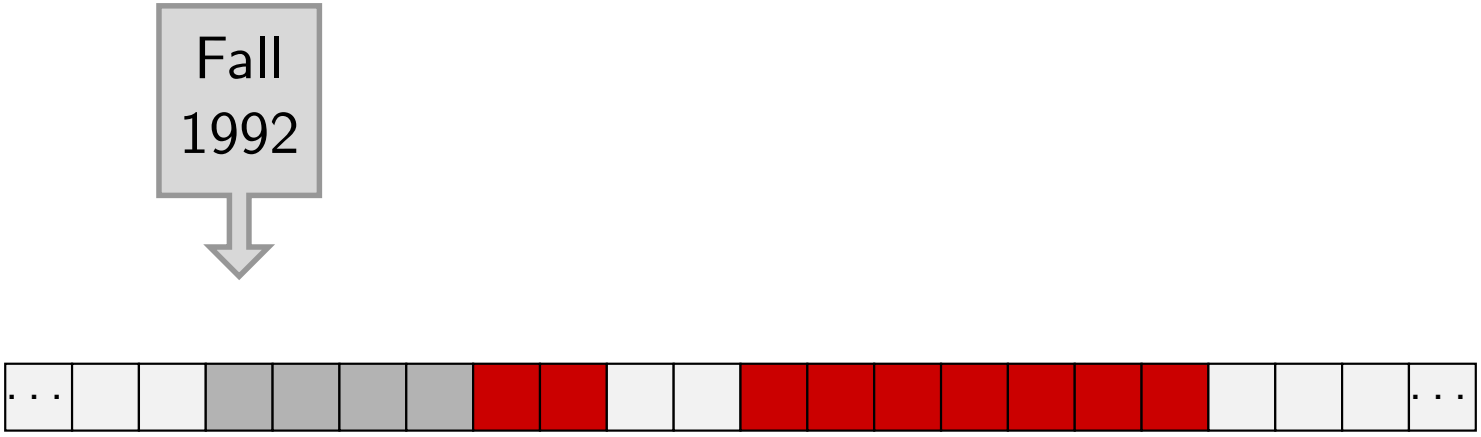
Jean-Christophe Filliâtre
CNRS

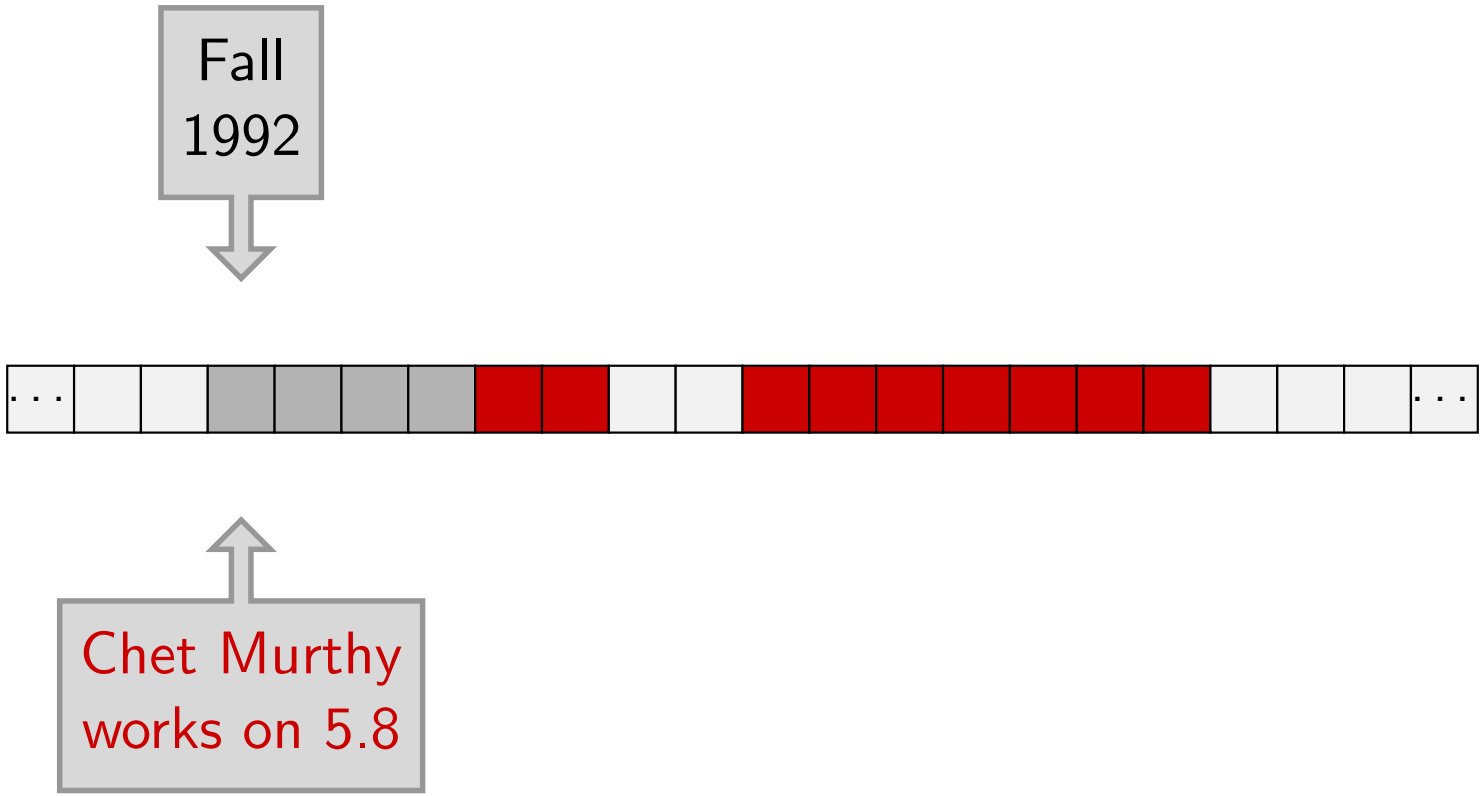
The Coq Workshop 2020

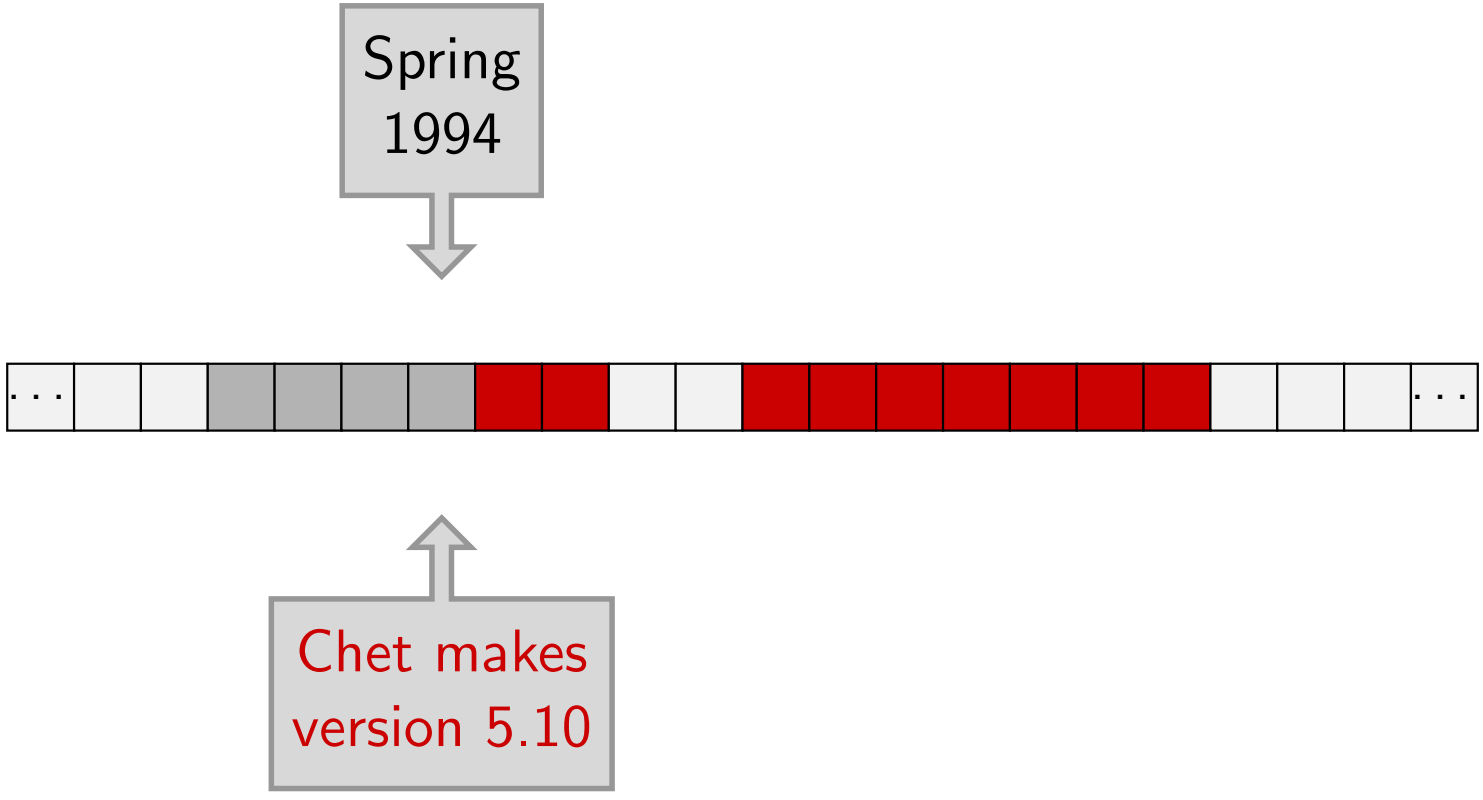
July 6, 2020

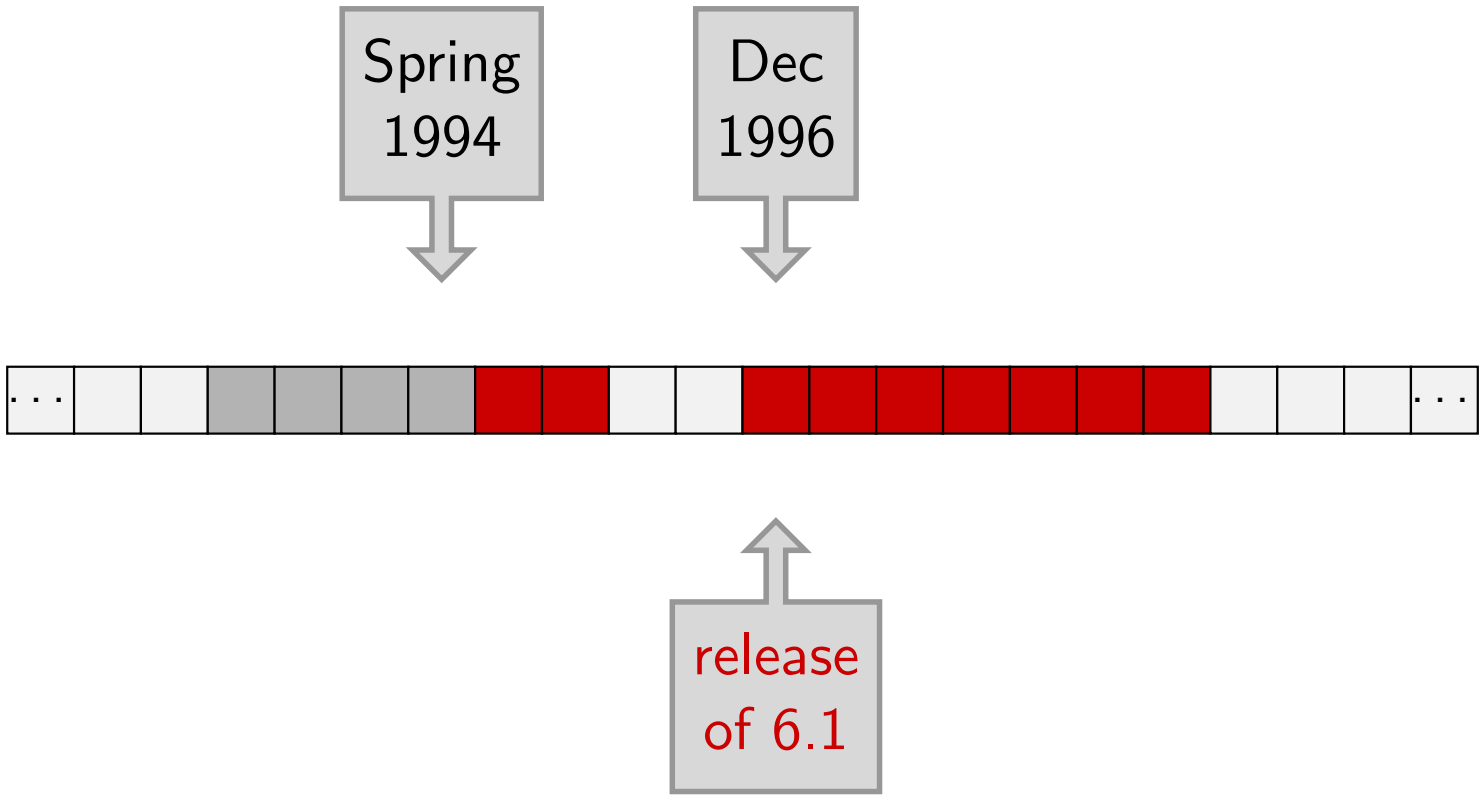
35 years of history

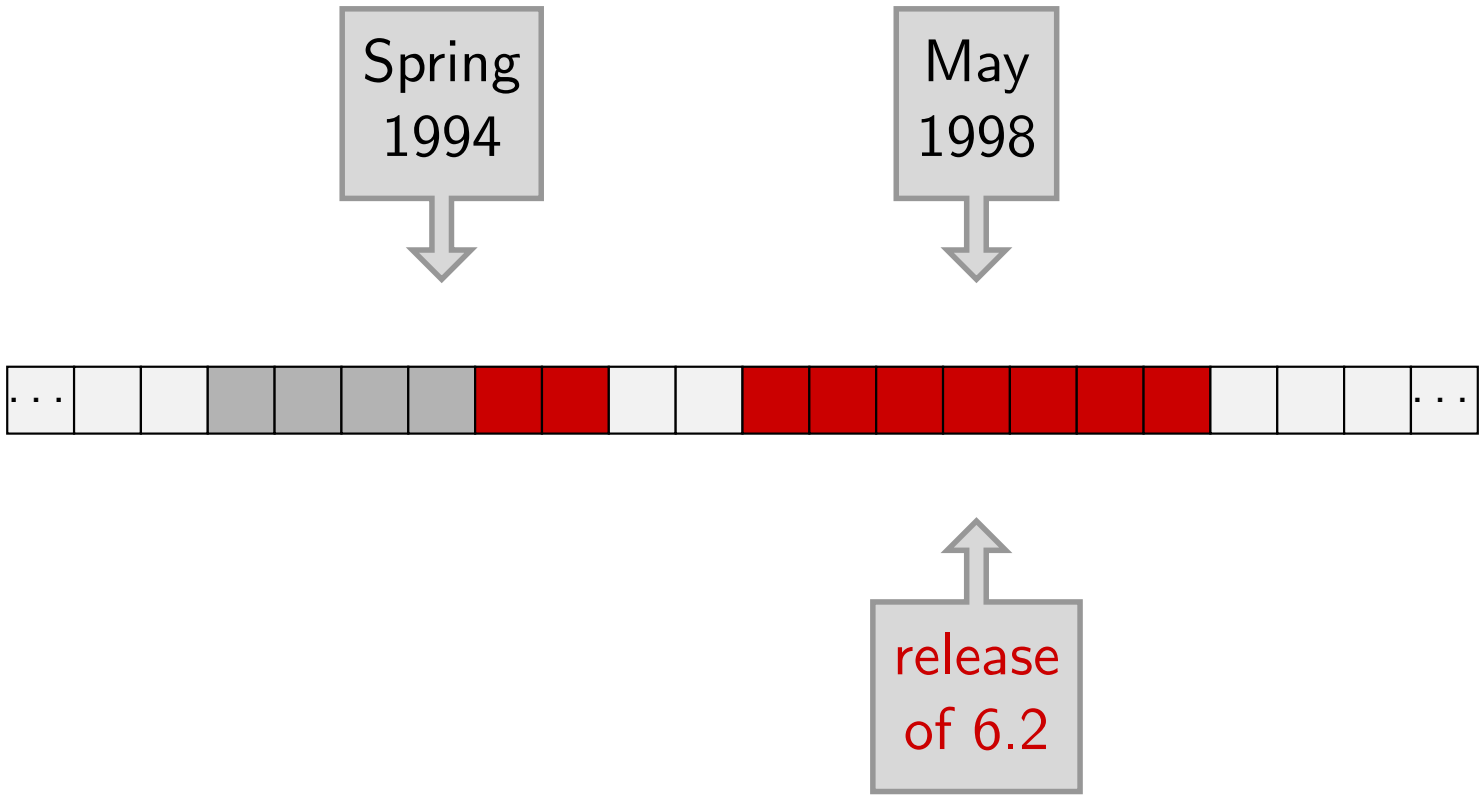


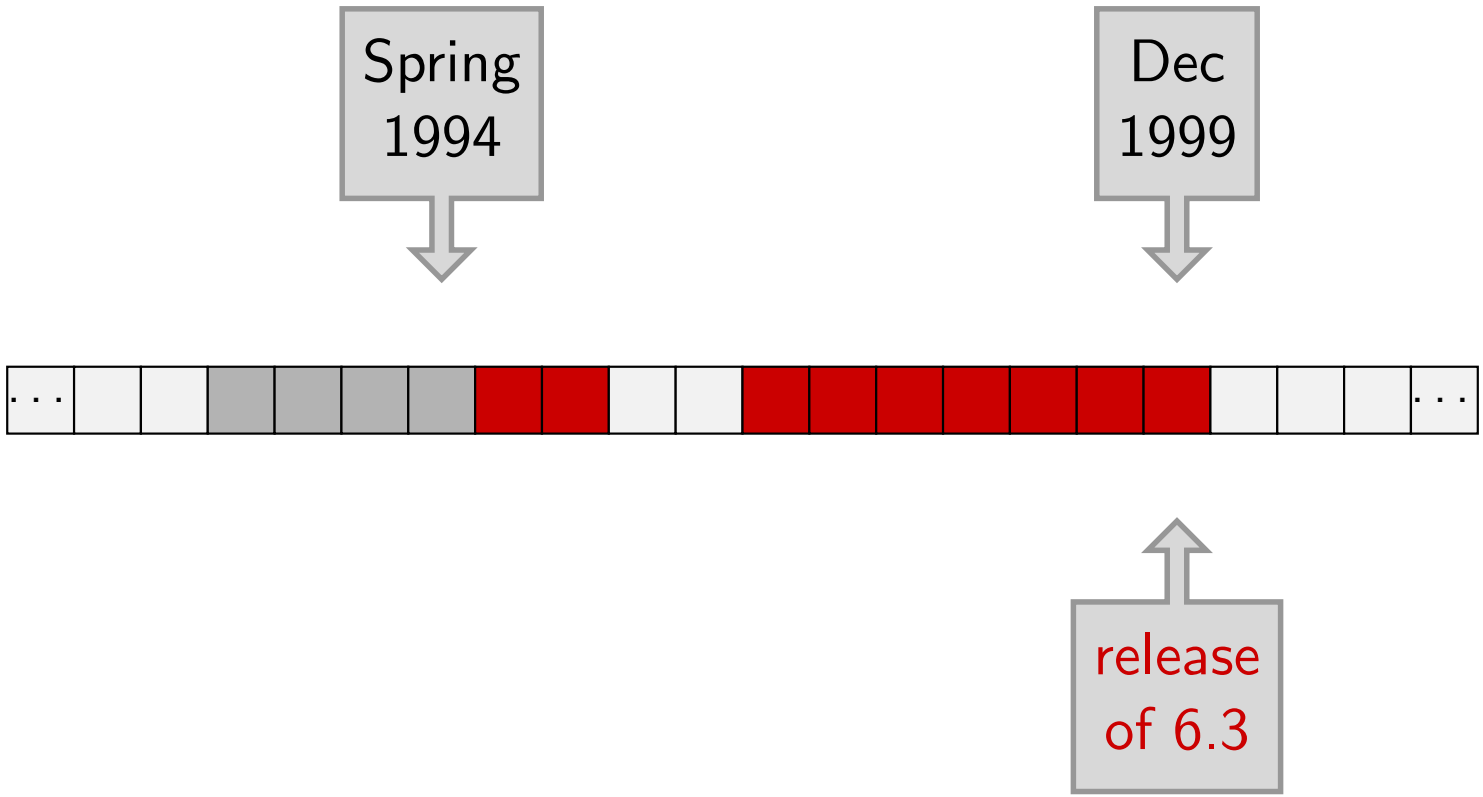

Fall
1992
. . .
. . .

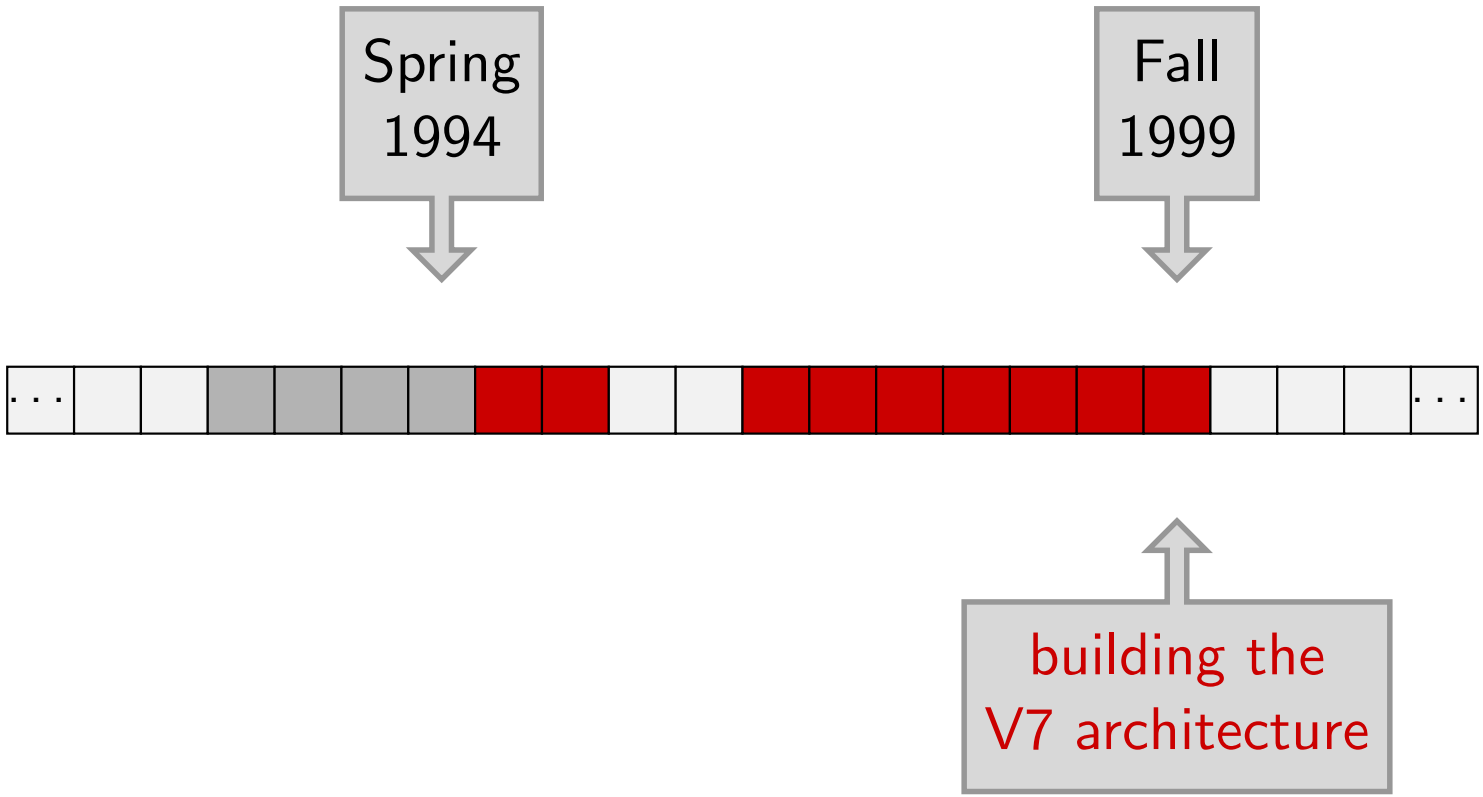

Fall
1992
Chet Murthy
works on 5.8


Spring
1994
Chet makes
version 5.10


Spring
1994
Dec
1996
release
of 6.1


Spring
1994
May
1998
release
of 6.2


Spring
1994
Dec
1999
release
of 6.3


Spring
1994
Fall
1999
building the
V7 architecture

what was in Chet's 5.10

what was in Chet's 5.10

- efficiency
  - de Bruijn indices, space-efficient terms
  - hash-consed identifiers
  - efficient rollback mechanism (more later)
- extensibility
  - user-extensible grammar (parser, pretty-printer)
  - mechanisms to declare new tables/operations
- separate compilation
  - a Coq file is a separate module
  - it is compiled to a `.vo` file

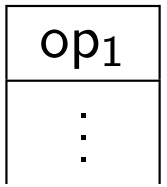
rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

rollback mechanism

1. when declaring a table, provide `freeze`/`unfreeze` operations

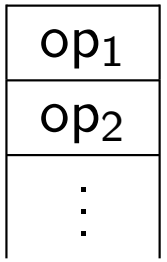
2. a single stack of all operations (with a little bit of dynamic typing under the hood)



rollback mechanism

1. when declaring a table, provide `freeze`/`unfreeze` operations

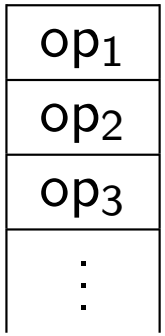
2. a single stack of all operations (with a little bit of dynamic typing under the hood)



rollback mechanism

1. when declaring a table, provide `freeze`/`unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

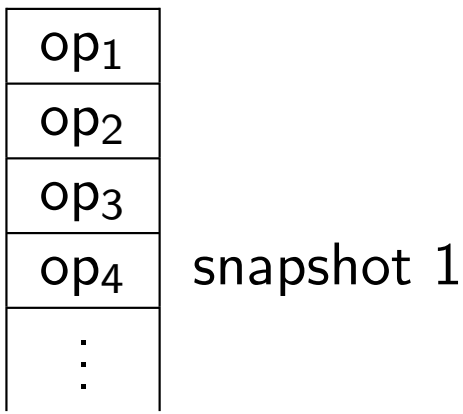


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

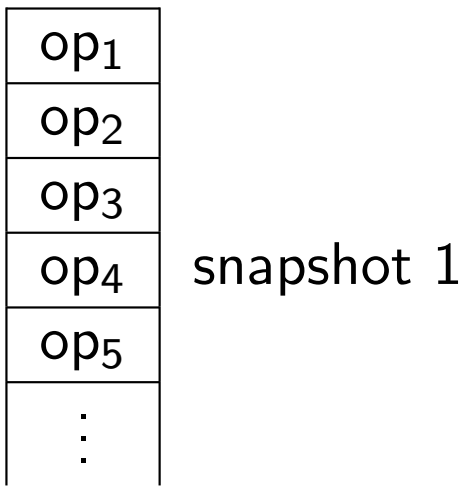


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

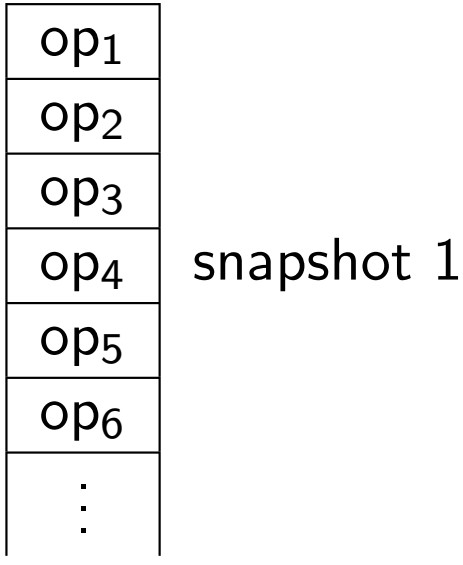


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

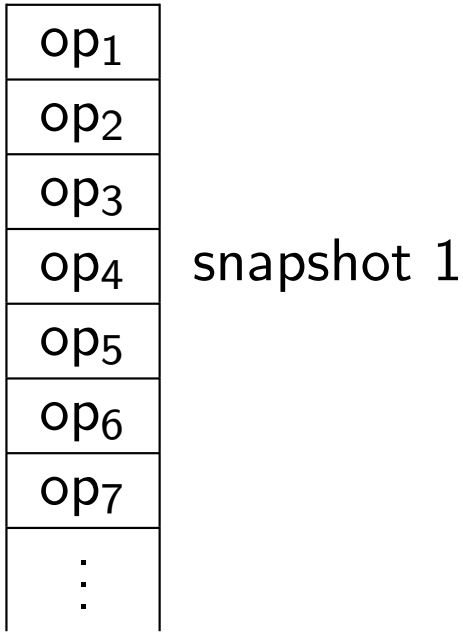


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

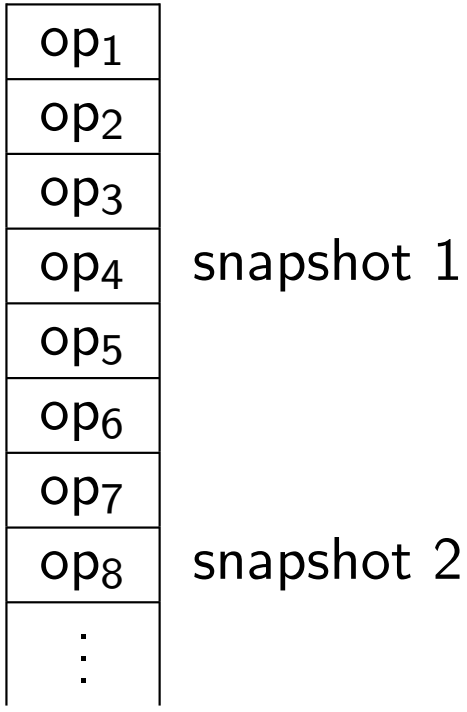


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

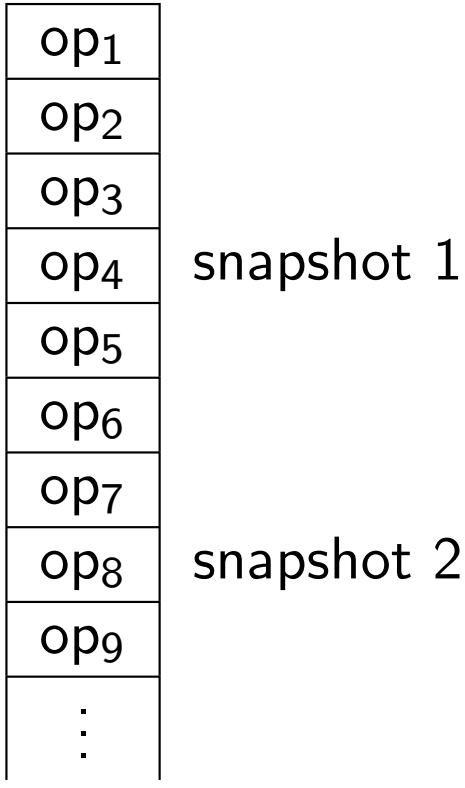


rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`



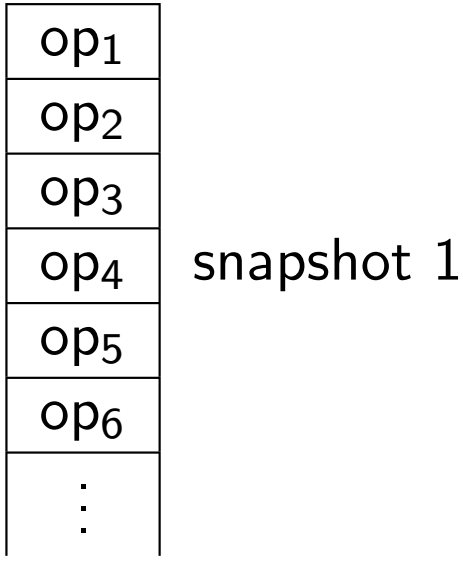
rollback mechanism

1. when declaring a table, provide `freeze/unfreeze` operations

2. a single stack of all operations (with a little bit of dynamic typing under the hood)

3. from time to time, take snapshots of all tables using `freeze`

4. to move back in time, roll back to the previous snapshot and redo some operations



```
let table = ref ...purely functional data structure...
let freeze () = !table          (* O(1) *)
let unfreeze v = table := v     (* O(1) *)
```

assuming some flavor of balanced trees,
this is only a $O(\log N)$ overhead factor (time and space)

may even be less than that for space
(depends on snapshot frequency)

why changing something that works fine?

despite all its marvels, 5.10 had no such thing as a kernel of trust
(and subsequently the versions 6)

the trusted computing base was a little bit everywhere

in particular,

- the rollback mechanism comes first
- CIC declarations are operations like any others

Coq version 7
a new architecture, with a kernel

1. implement a purely functional type checker for the CIC
   (the kernel)

2. then the rollback mechanism
   (outside the kernel)

3. last, declare a table holding the current typing environment
   (in a reference)

even like this, the kernel is not small (7,800 loc at that time)

not convenient to put all that behind a single abstraction barrier (and no such thing as OCaml `-pack` back in 1999)

files for CIC terms

```
type constr = ...
...
```

can be ill-formed/ill-typed

files for CIC terms

```
type constr = ...
...
```

can be ill-formed/ill-typed

files for CIC environments

```
type env = ...
val add_constant:
  env -> constant -> env
...
```

just a data structure

environments can be ill-formed

files for CIC terms

```
type constr = ...
...
```

can be ill-formed/ill-typed

files for CIC environments

```
type env = ...
val add_constant:
  env -> constant -> env
...
```

just a data structure

environments can be ill-formed

files for typing rules

```
val type_constr:
  env -> constr -> constr
...
```

can be misused

finally, wrap everything behind an abstraction barrier

```
type safe_env
val empty: safe_env
val add_constant: safe_env -> constant -> safe_env
...
```

finally, wrap everything behind an abstraction barrier

```
type safe_env
val empty: safe_env
val add_constant: safe_env -> constant -> safe_env
...
```

whose implementation is trivial

```
type safe_env = env
let empty = Env.empty
let add_constant env c =
  let c = type_constant env c in
  Env.add_constant env c
...
```

global environment

outside the kernel, declare a global, mutable environment

```
let global_env = ref Kernel.empty
let add_constant c =
  global_env := Kernel.add_constant !global_env c
...
```

and declare it as a table

```
let freeze () = !global_env
let unfreeze v = global_env := v
let _ = declare_table "typing env" freeze unfreeze
```

how to trust the disk?

one more issue: Coq's `Require` loads declarations from `.vo` files

and all this machinery is outside of the kernel

how to trust the disk?

one more issue: Coq's `Require` loads declarations from `.vo` files

and all this machinery is outside of the kernel

the solution is borrowed from OCaml's compiler

- when writing a file to the disk,
  - include MD5 checksums of loaded modules
  - include its own checksum
- when loading a file,
  - verify that assumptions and reality coincide

if I had to do it again

if I had to do it again

- I would consider hash-consing+memoisation seriously
- I would consider a more defensive API for the kernel, with terms that are always well-typed

conclusion

many, many thanks

deep thanks to

- Chet, for his code, for inspiring me
- Christine, for a one-in-a-lifetime opportunity

and to all the other Coq developers in 1994–1999

- Bruno Barras
- Cristina Cornes
- Yann Coscoy
- Judicaël Courant
- David Delahaye
- Daniel de Rauglaudre
- Eduardo Giménez
- Hugo Herbelin
- Gérard Huet
- Patrick Loiseleur
- César Muñoz
- Catherine Parent-Vigouroux
- Amokrane Saïbi
- Benjamin Werner

- if you see young interns who like coding and who are willing to contribute, give them a chance

- if you see young interns who like coding and who are willing to contribute, give them a chance

- your code won't be the best cathedral ever;
  accept this idea and make the best compromise you can

- if you see young interns who like coding and who are willing to contribute, give them a chance
- your code won't be the best cathedral ever;
  accept this idea and make the best compromise you can
- postdoc is a sweet spot,
  where you can combine experience with time