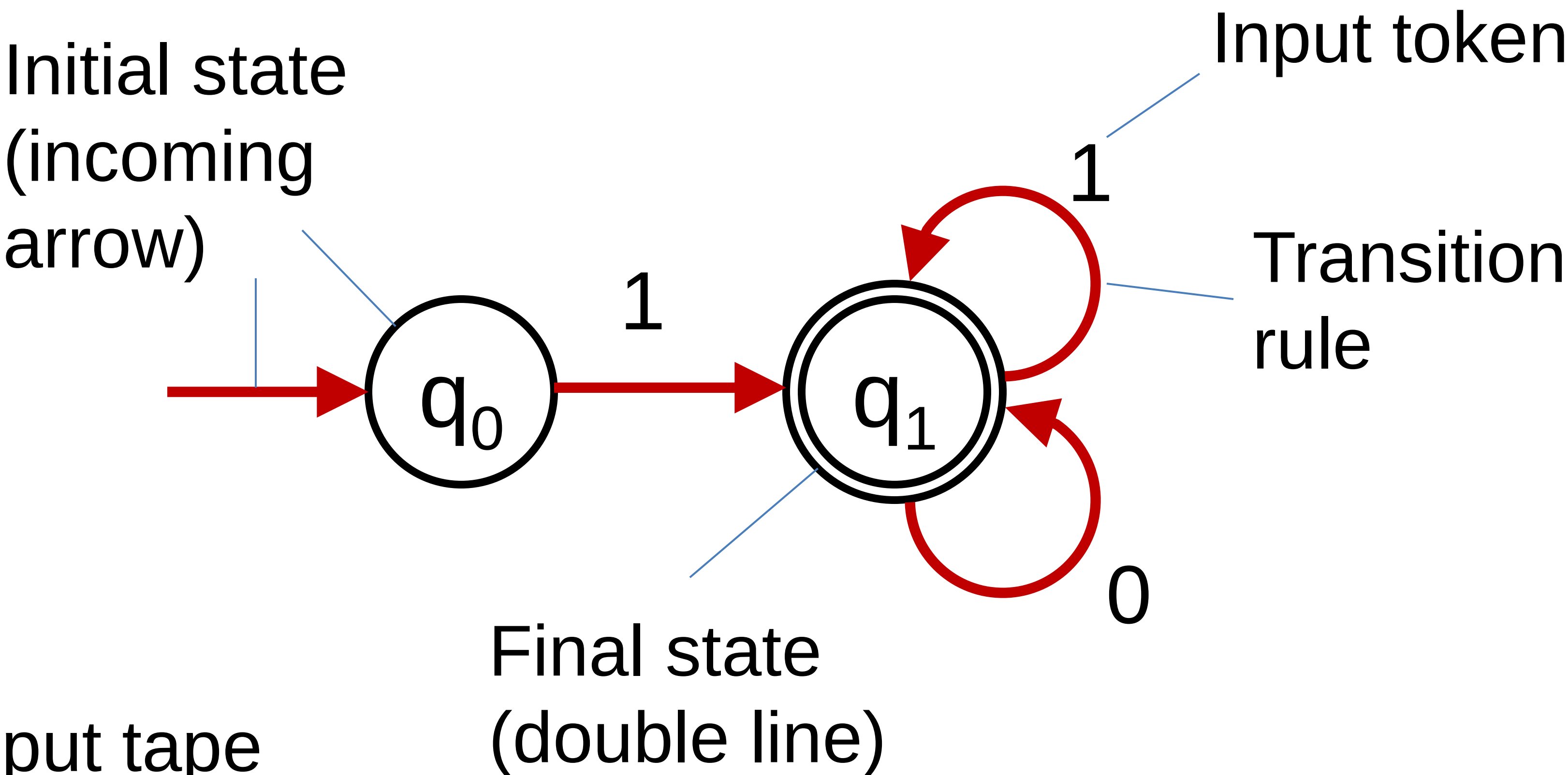
Mathematical Concepts (G6012)

Computing Machines II

Thomas Nowotny
Chichester I, Room CI-105
Office hours: Mondays 10:00-12:00
T.Nowotny@sussex.ac.uk

Graphic Representation



Input tape

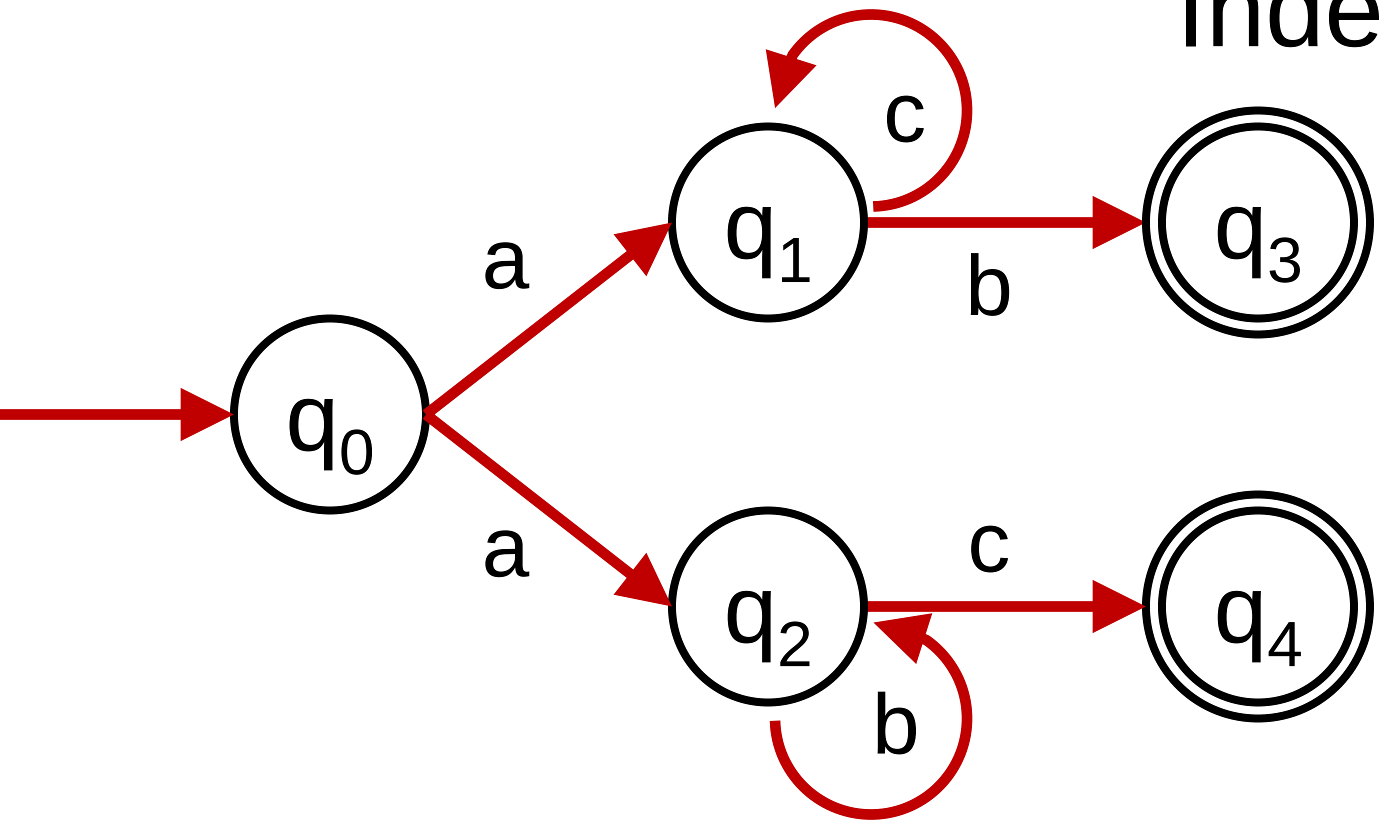
1	0	1	0

Outcomes of a FSA computation

- **Accepting computation**:
  Computation in which the machine reaches a final state and reads all the input.

- **Non-accepting computation**:
  Computation in which either the machine gets stuck before end of input or finishes in a non-final state.

Indeterministic FSA



- Either *a* then zero or more *c*'s then b, or *a* then zero or more b's then c.
- More precisely: $\{ab^nc : n \geq 0\} \cup \{ac^nb : n \geq 0\}$
- Regular expression: (*ab**c*)|(*ac**b*)
- Nondeterministic!

Non-determinism

- What does it mean?
  - Machine has a choice of more than one legal move
  - Machine is able to explore all options
- Significance
  - Important theoretical idea
  - Nondeterminism arises with many computational models

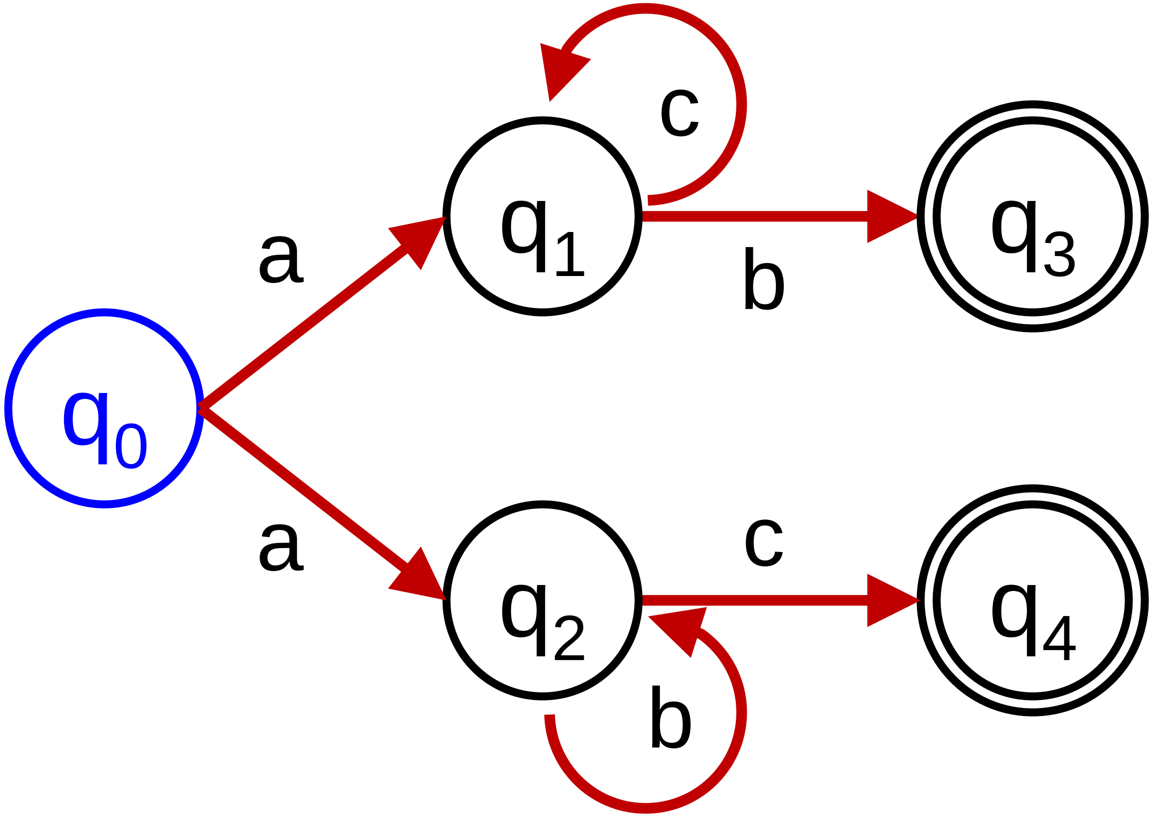
Deterministic versus nondeterministic FSA

- Deterministic FSA: There is never any choice in the computation
- However: Equivalence (!):
  - Nondeterministic FSA are equivalent to deterministic FSA, i.e. for every FSA there is an equivalent deterministic FSA
  - Prove by means of a construction:

Construction

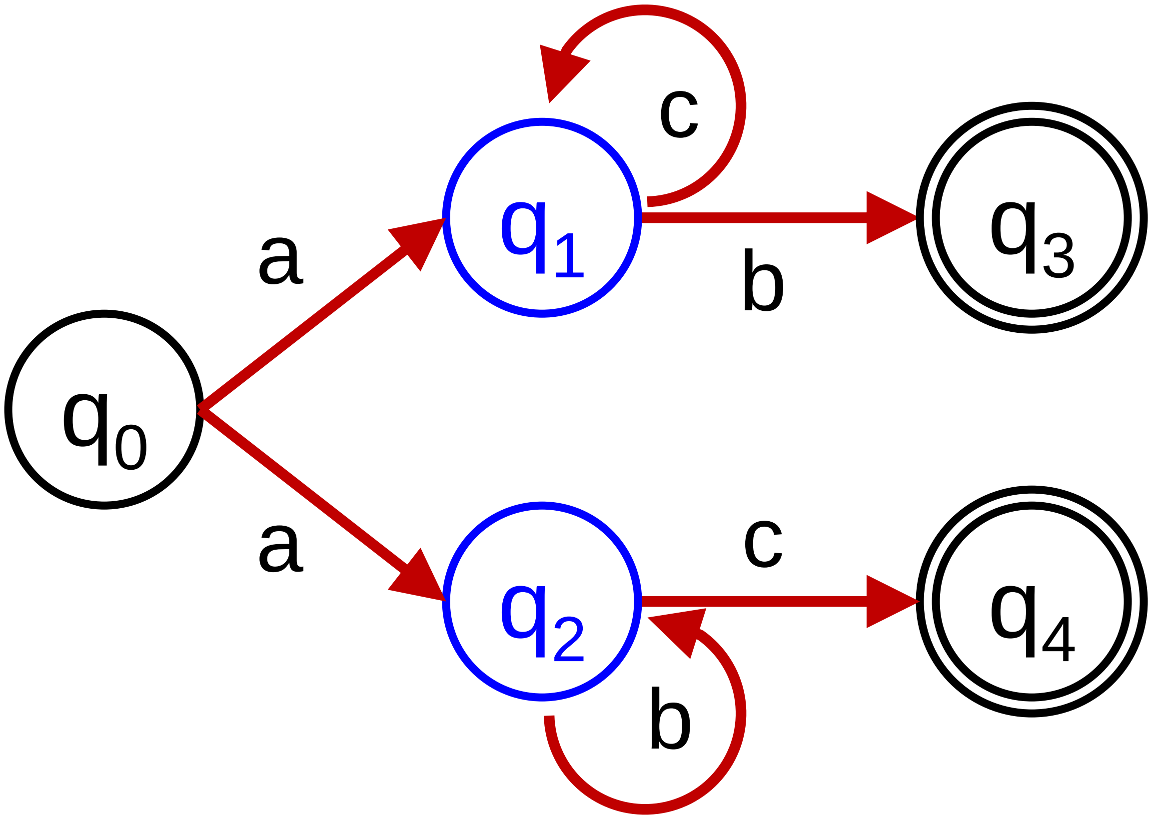
- What do we need to do?
  - Create deterministic machines that simulate nondeterministic machines
- Let's have a closer look at our nondeterministic example...

Example revisited



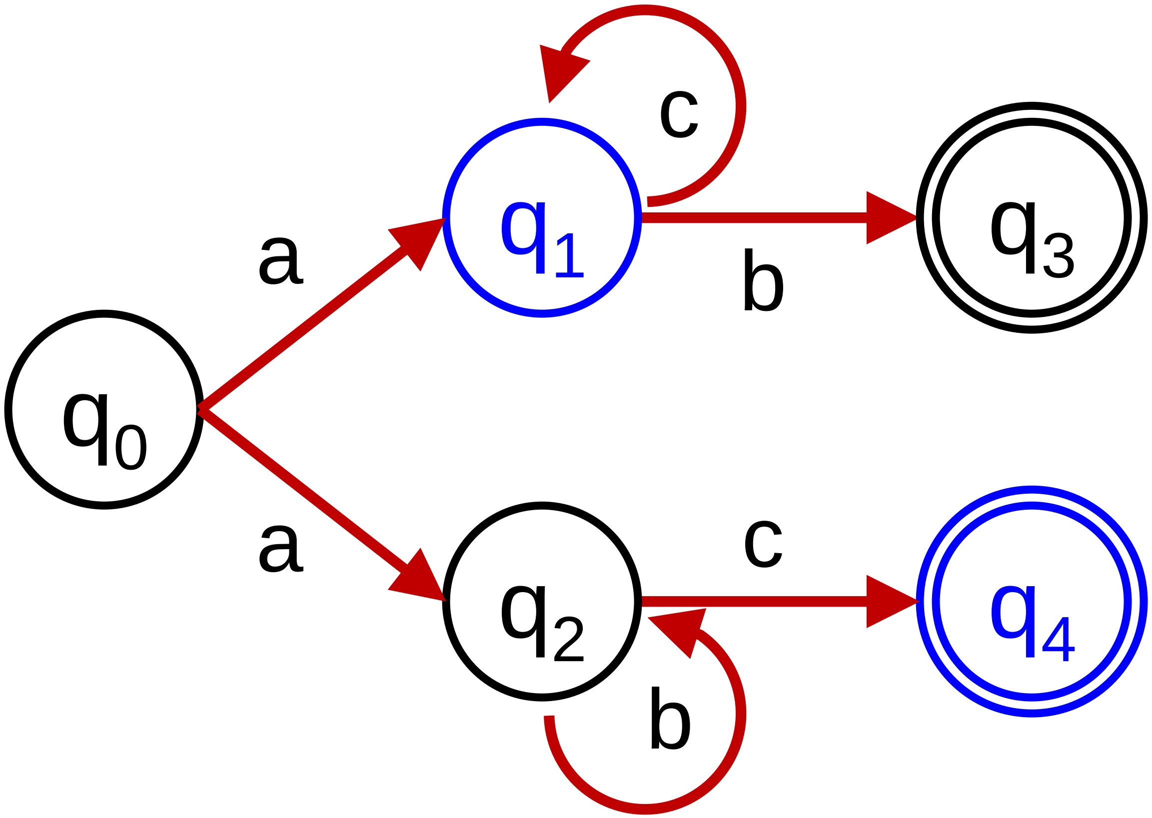
Suppose we see an ‘*a*’ first

Example revisited



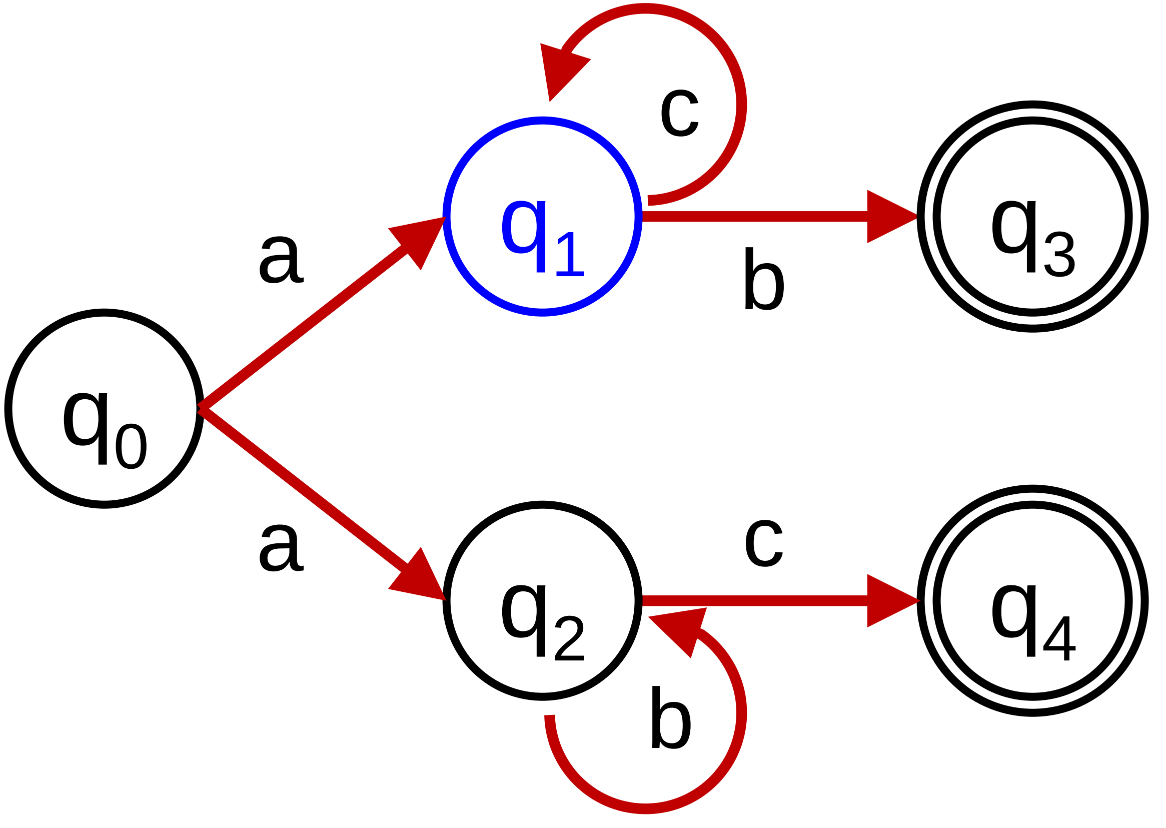
Suppose we see a ‘*c*’ next

Example revisited



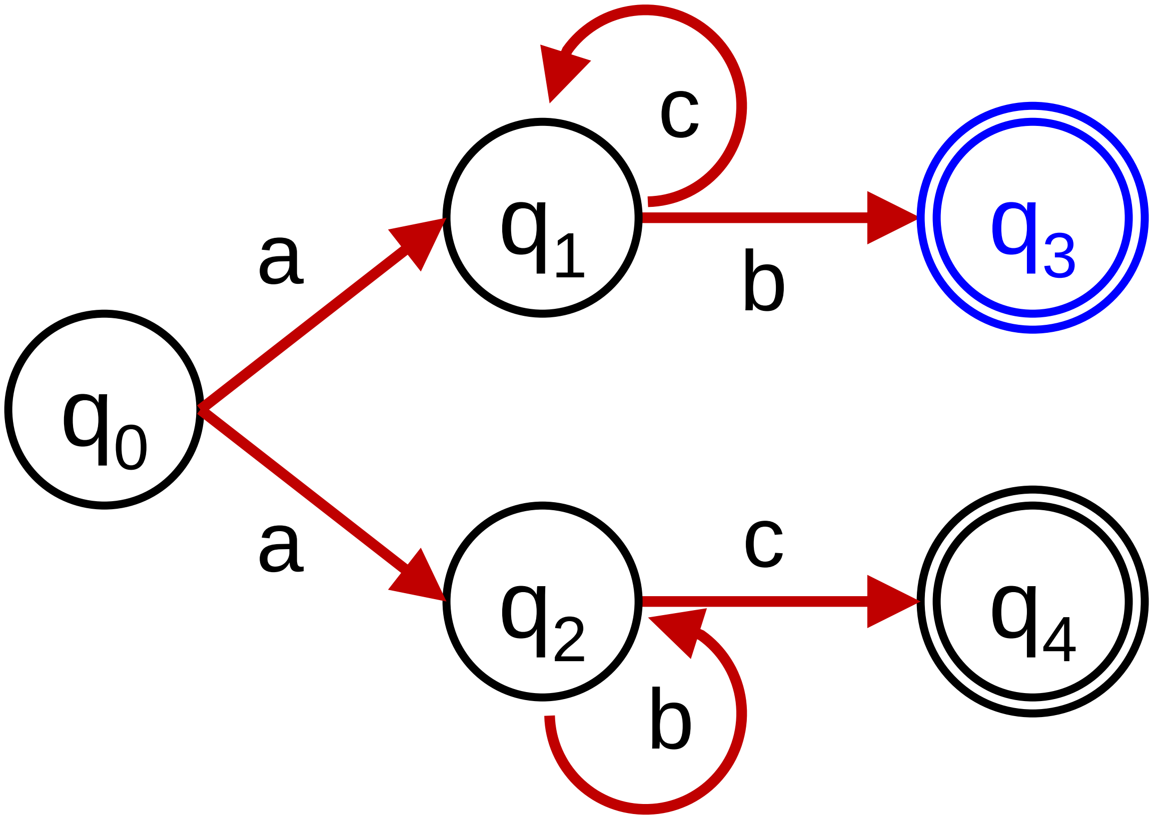
Suppose we see a ‘c’ next

Example revisited



And finally we see a ‘*b*’

Example revisited



The input is consumed and we are in a final state

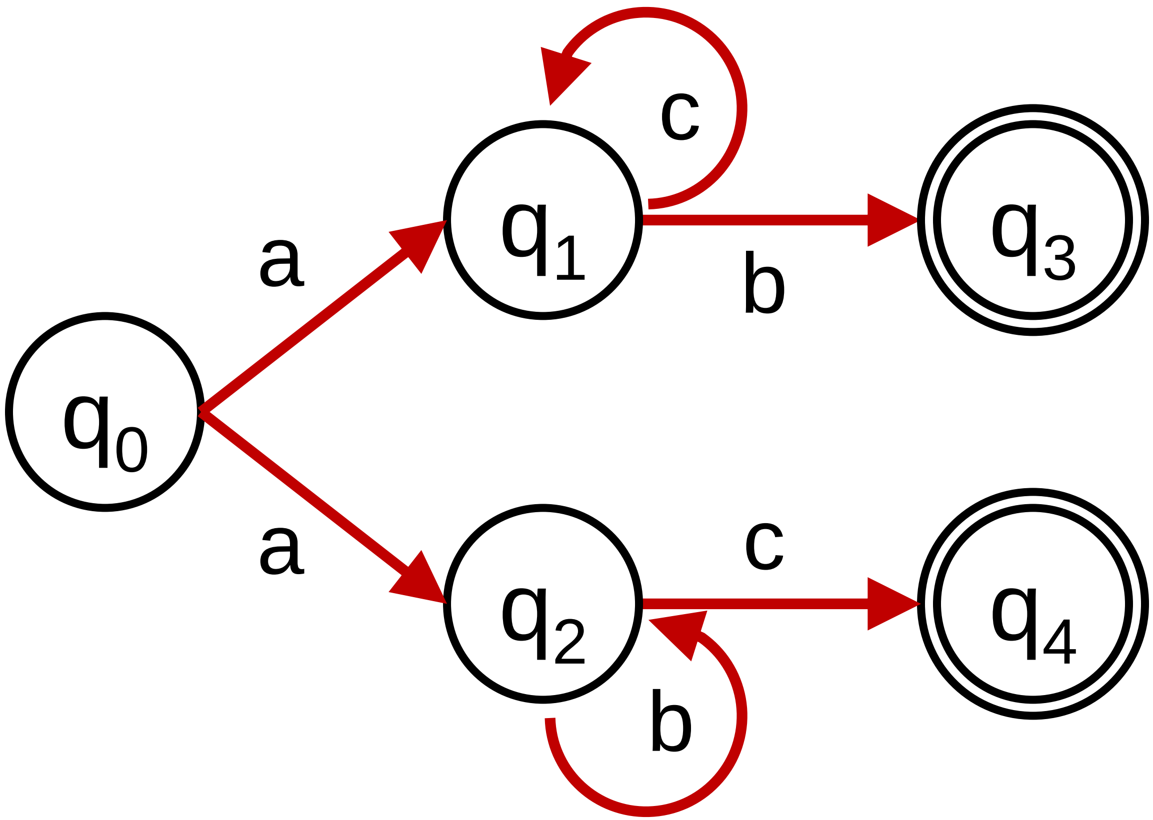
Simulating indeterminism

- Finiteness is crucial:
  - Finite number of states
  - Finite number of possible sets of states
  - 2^n possible subsets of n objects
  - Use subset to record all possible states that could be reached
  - Run all computations of a nondeterministic machine in parallel

Simulating indeterminism

- Build new deterministic machine
  - One state for every subset
  - New transitions based on original machine
  - Next state determined by what original machine would do

Our example



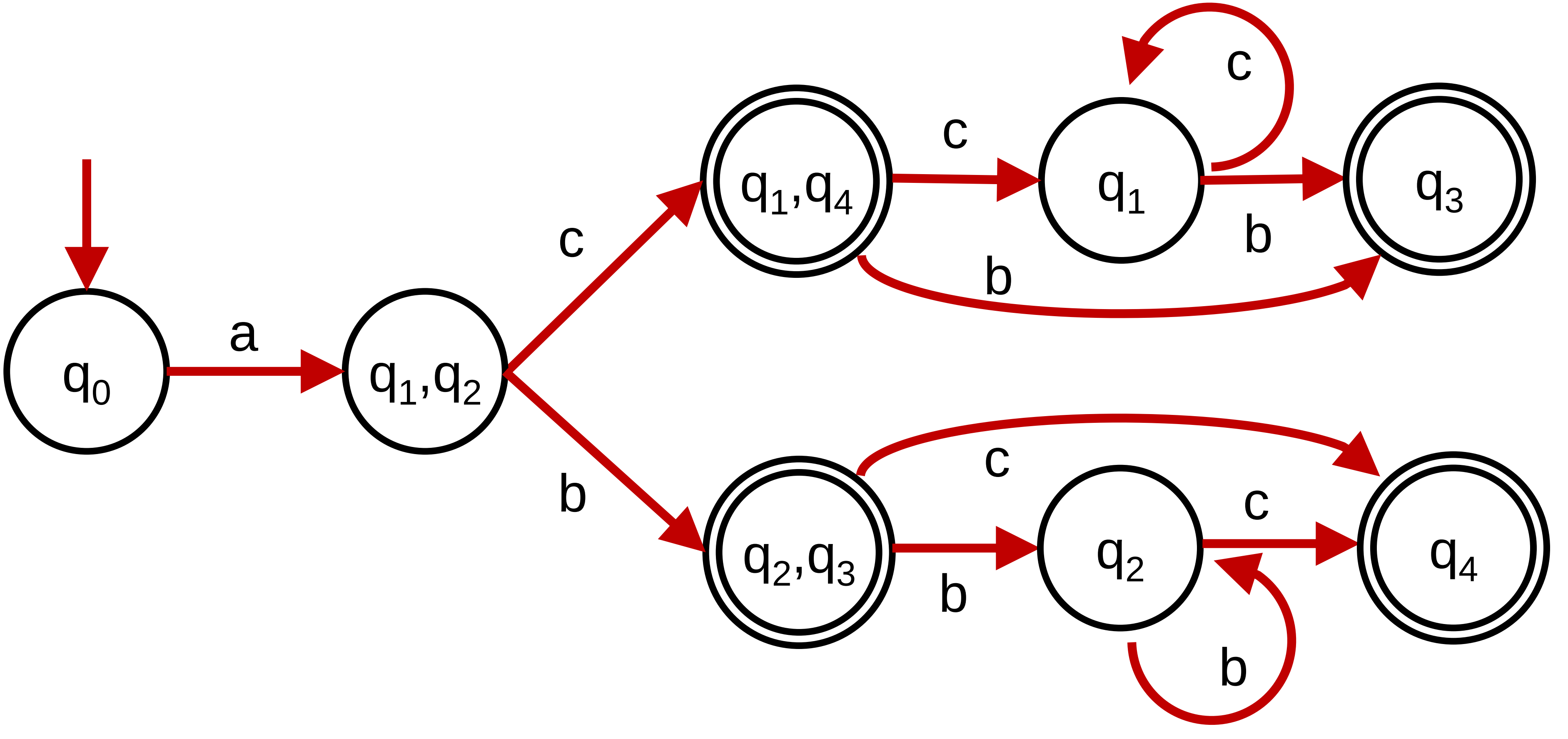
BB Constructing a deterministic machine

- States: $\{q_0\}$, $\{q_1,q_2\}$, $\{q_2,q_3\}$, $\{q_1,q_4\}$, $\{q_3\}$, $\{q_4\}$,$\{q_1\}$,$\{q_2\}$
- Transition from $\{q_0\}$ to $\{q_1, q_2\}$ on *a*
- Transition from $\{q_1,q_2\}$ to $\{q_1,q_4\}$ on *c*
- Transition from $\{q_1,q_4\}$ to $\{q_3\}$ on *b*
- Transition from $\{q_1,q_2\}$ to $\{q_2,q_3\}$ on *b*
- Transition from $\{q_2,q_3\}$ to $\{q_4\}$ on c
- And so on …

BB Constructing a deterministic machine

- Initial state is { q_0 }
- Any set containing q_3 or q_4 is final

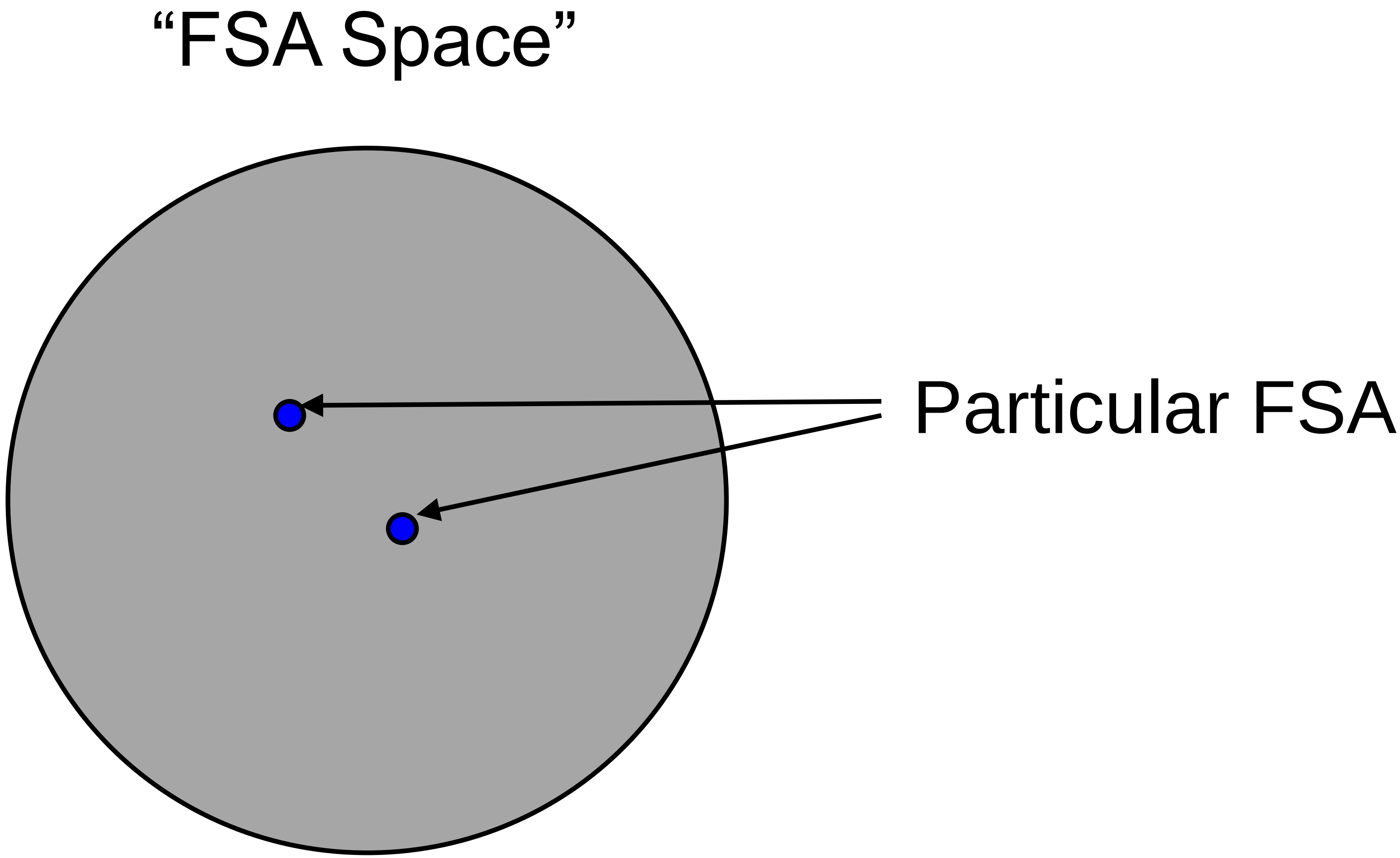
Equivalent deterministic FSA



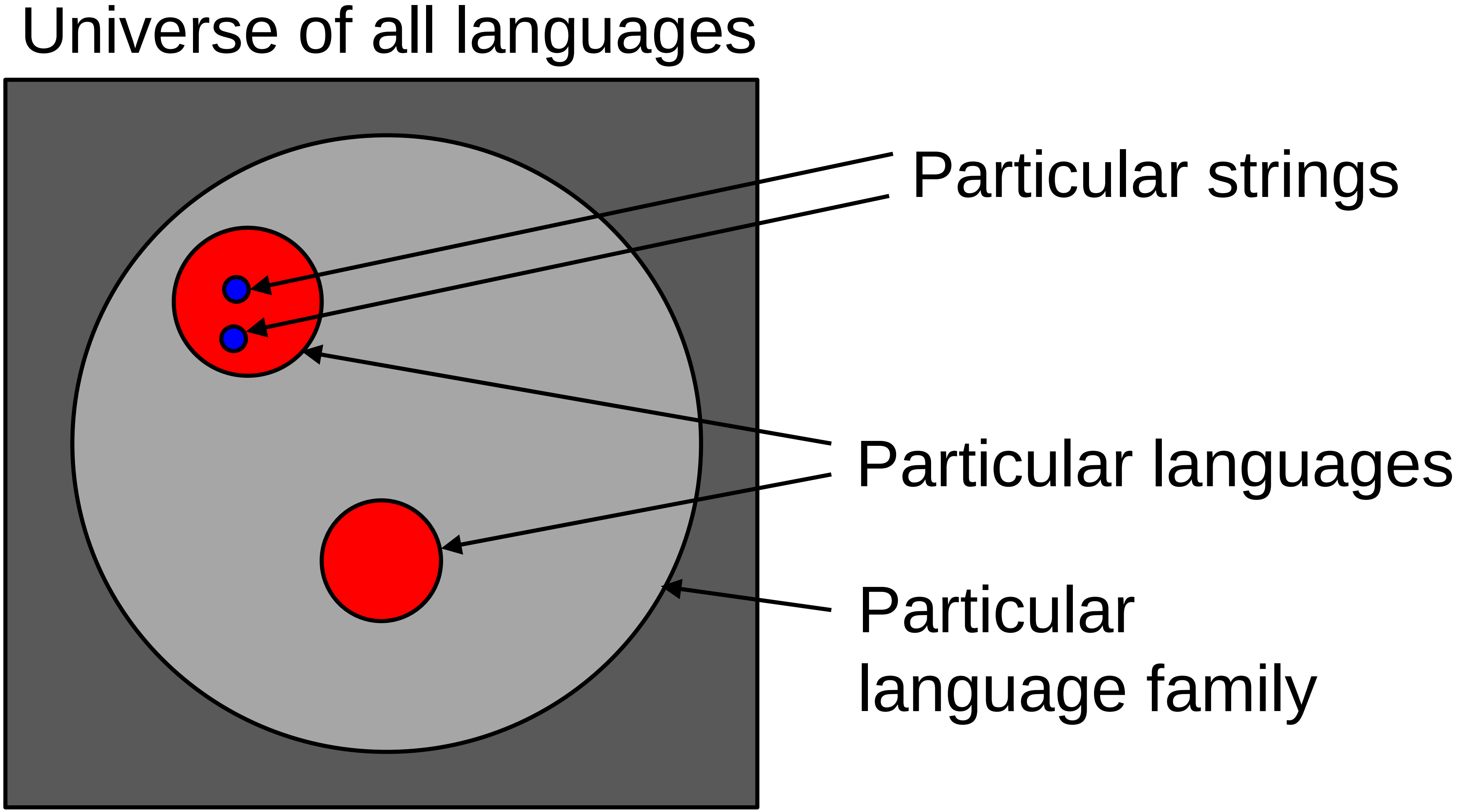
Stepping back …

- What did we just do?
  - We showed something very general
  - Two classes of machines are equivalent
  - Based on a general simulation
  - This is an important idea

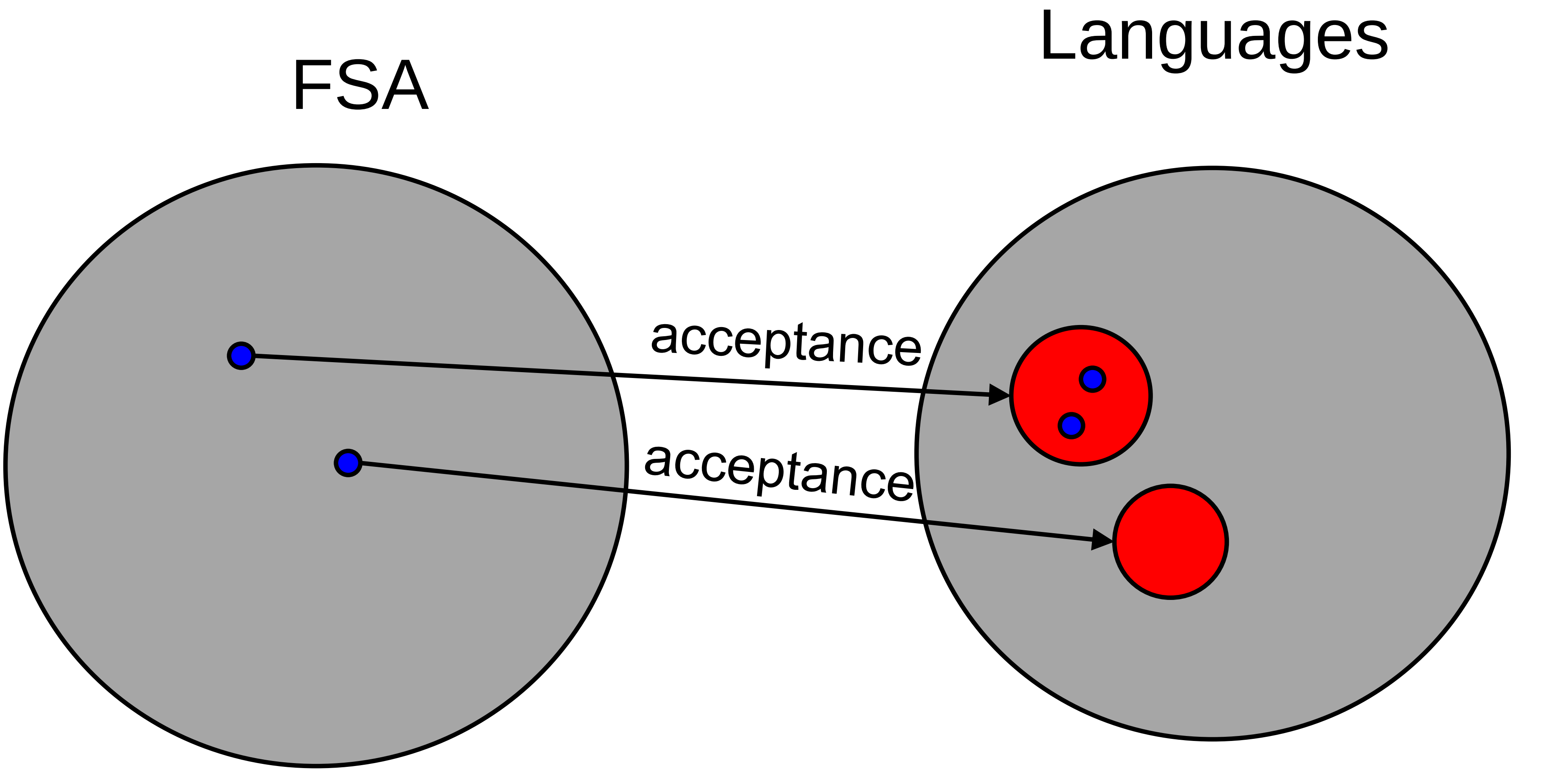
Finite State Automata (FSA)



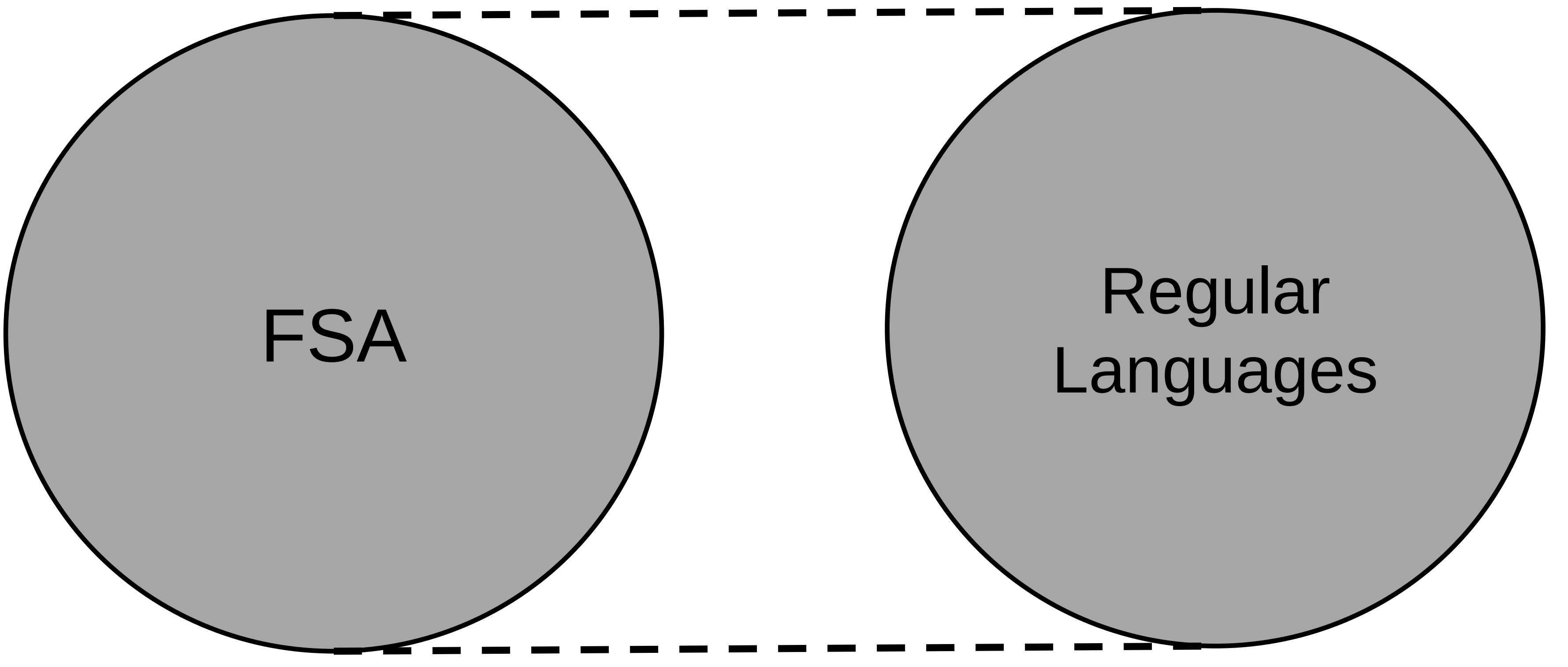
Languages



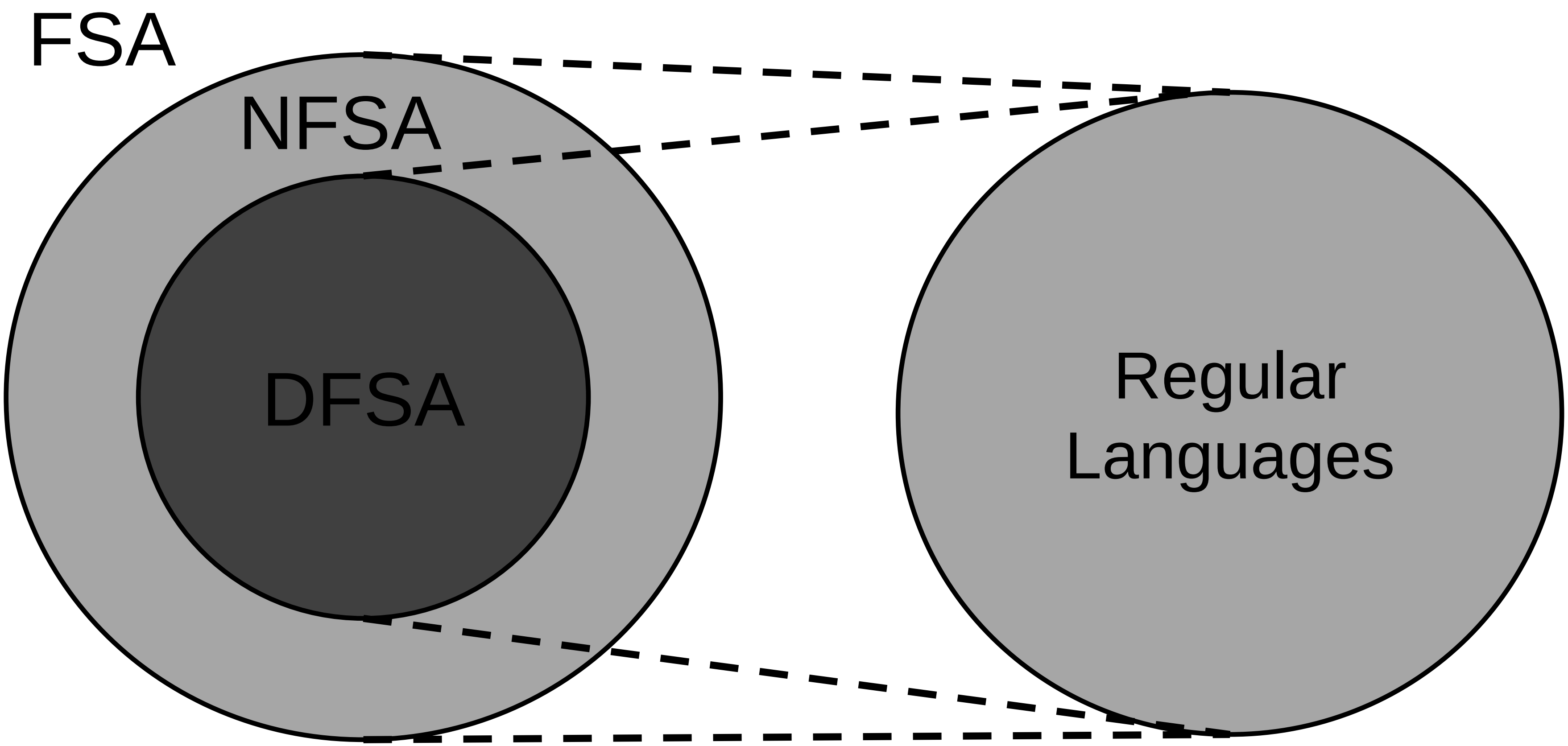
Universe of all languages
Particular strings
Particular languages
Particular
language family


FSA
Languages
acceptance
acceptance

FSA accept Regular Languages,
and only Regular Languages



Non-deterministic FSA (NFSA) and deterministic FSA (DFSA) accept the same family of languages – all Regular Languages



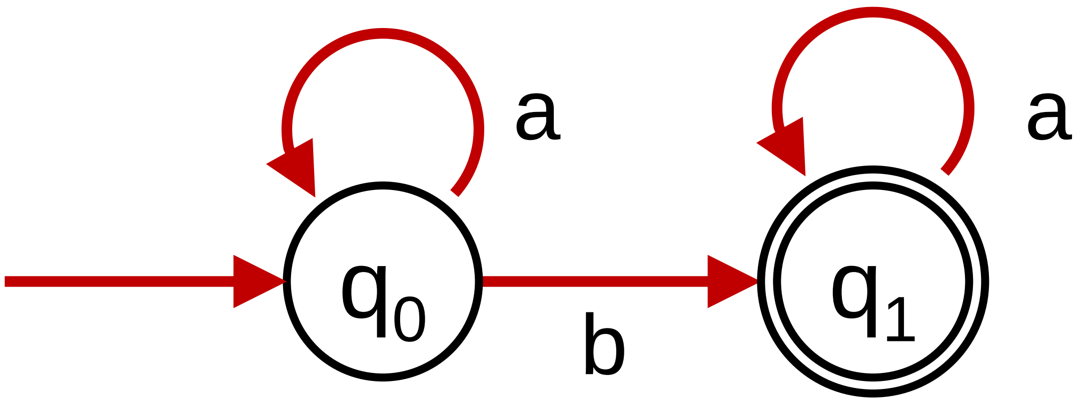
FSA summary 1

- FSA recognize (accept) the class of regular languages
  (which are closed under union, intersection, complement, and concatenation)
- FSA are equivalent to regular expressions
- But FSA have limited power (more on this later)

Finite State Memory

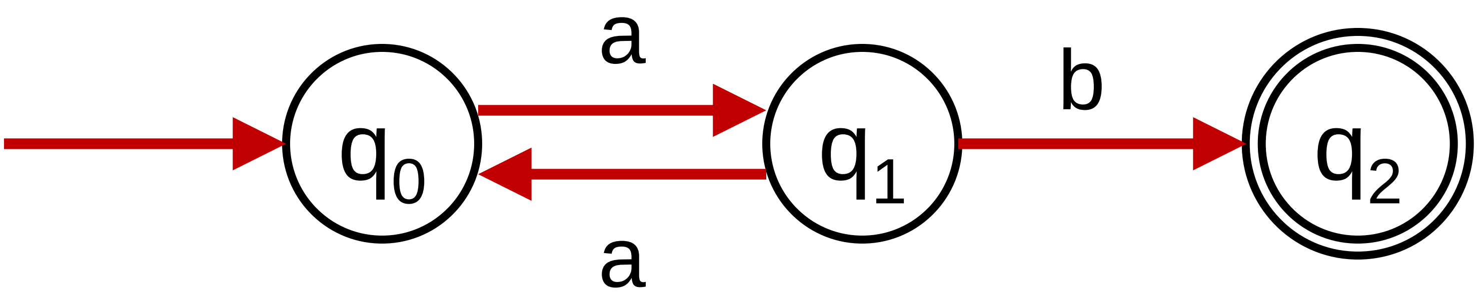
- An FSA makes decisions about the entire input
- But it cannot look again at any input that it already has consumed
- Needs to remember & can only use states to do that
- Memory limit: Finite number of states

Two State Memory



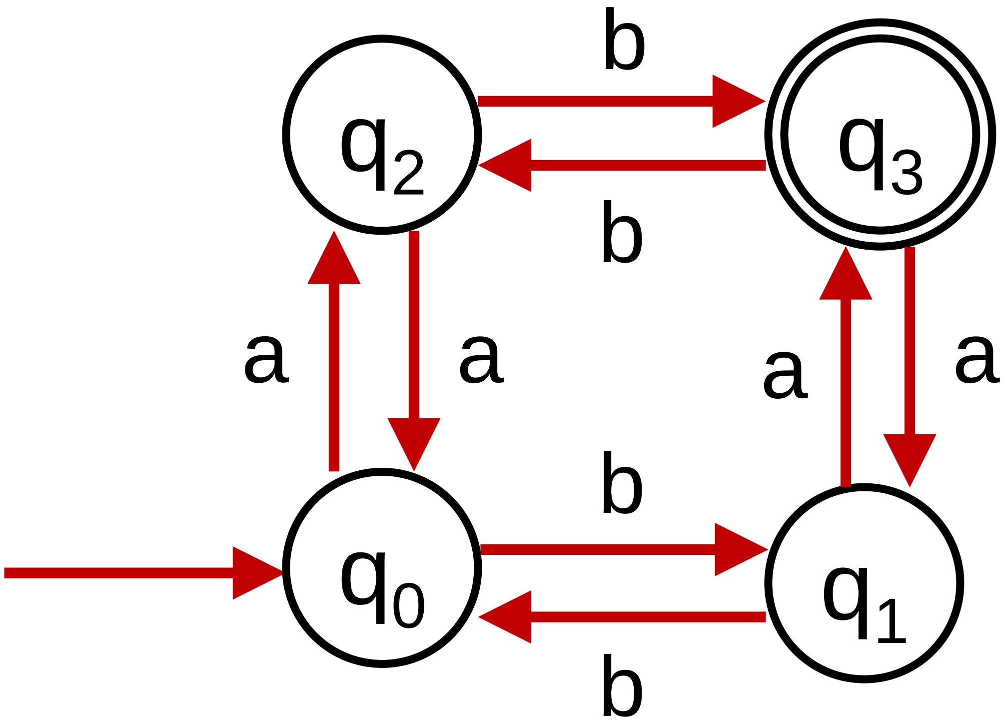
- In state q_0 when seen some a's
- In state q_1 when seen some a's then b and (possibly) more a's
- Each state constitutes a memory of what has been seen

Three State Memory



- In q_0 when seen even number of a's
- In q_1 when seen odd number of a's
- In q_2 when seen odd a's then b
- "Three memory items"

Four State Memory



- q_0: even a’s and even b’s; q_1 even a’s and odd b’s
- q_2: odd a’s and even b’s; q_3 odd a’s and odd b’s

Finite State Memory

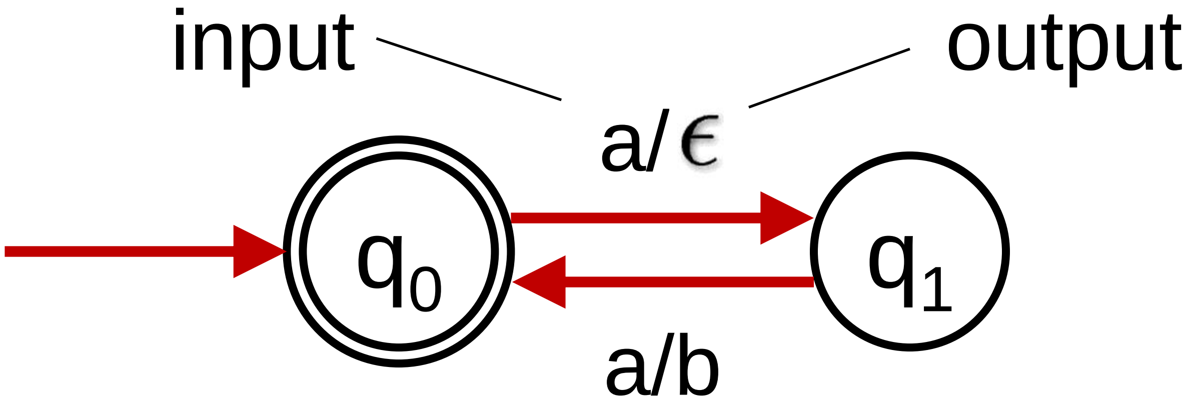
- We see that FSA can remember properties of the input
- However, the maximum number of memories is limited by the number of states

FINITE STATE TRANSDUCERS

Finite State Transducers

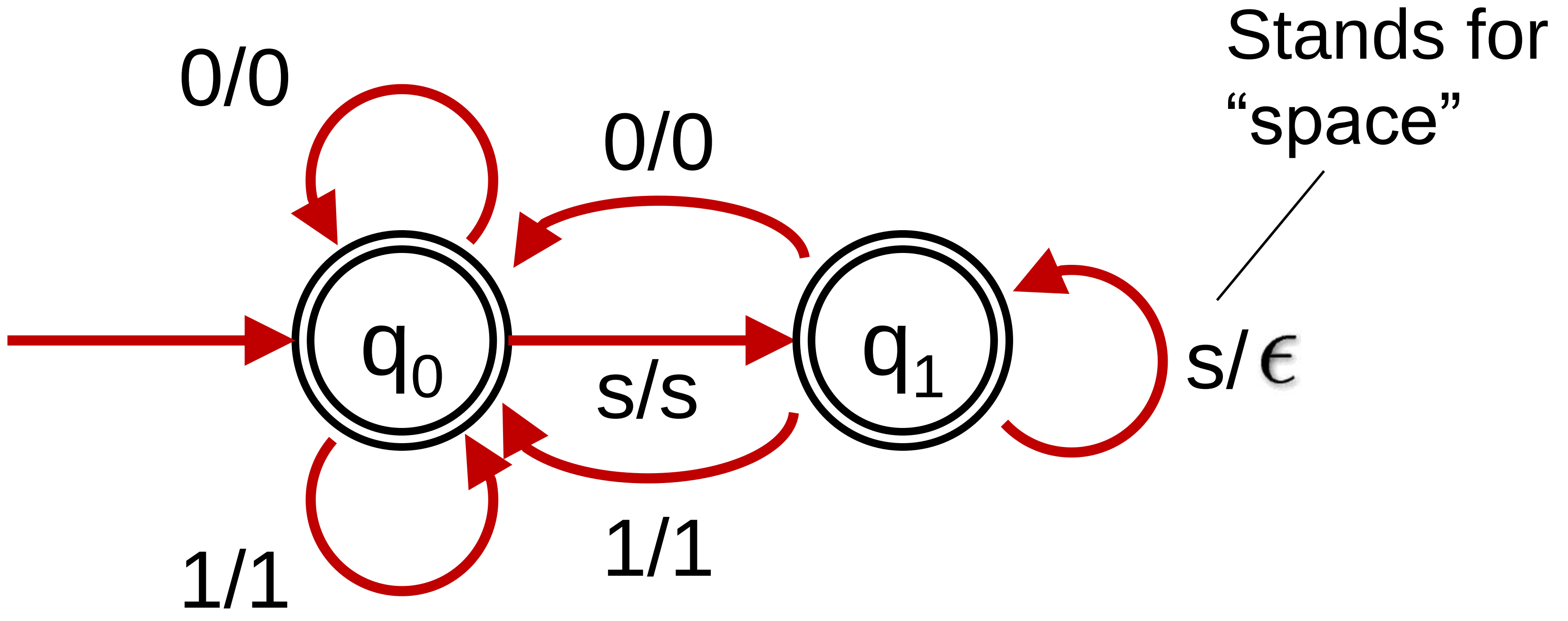
- Slightly enhance machine
- Also known as Mealy's automata
- Each input symbol is mapped to an output symbol, i.e. we have two tapes: Input tape and output tape
- Machine becomes a translator

Example FS Transducer



- Reminder: ϵ is the empty string
- What does it do?
- … translates strings of a's into strings of b's with half the length.

Another example



- What does it do?
- Cleans up white space: Removes all but one space from input

Application: Two phases of compilation

- Lexical analysis:
  - Identifying the sequence of tokens of characters in input file: Finite State Transducers are adequate for this
- Syntax analysis:
  - Checking syntax and determining structure: Finite State Machines/Transducers are not adequate due to possible nesting of statements
  - Needs more powerful machines – see later.

FSA summary 2

- Finite state machines are not only relevant to language processing
- State can be interpreted in a general sense: Current status of a system (but must be a finite number of states)
- What happens next can only depend on current state and next input
- Inputs could be non-linguistic: This can be applied in a variety of situations