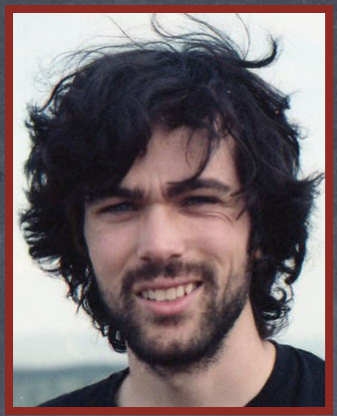
COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lecture 2 : Regular Expressions & DFAs



- **2019 T.A.s** :

Pouriya Alikhani	pouriya.alikhani@mail.mcgill.ca
Pierre-William Breau	pierre-william.breau@mail.mcgill.ca
Anirudha Jita	anirudha.jitani@mail.mcgill.ca
Justin Li	juan.y.li@mail.mcgill.ca
Yanjia Li	yanjia.li@mail.mcgill.ca
Shiquan Zhang	shiquan.zhang@mail.mcgill.ca

- **Office Hours** :

Claude : Wednesday 13:00-16:00 ENGMC 110N
Pouriya : Friday 13:00-14:00 ENGTR 3090
Pierre-William : Monday 15:00-16:00 ENGTR 3110
Anirudha : Monday 16:00-17:00 ENGTR 3090
Justin : Tuesday 15:00-16:00 ENGTR 3110
Yanjia : Friday 10:00-11:00 ENGTR 3110
Shiquan : Thursday 15:00-16:00 ENGTR 3110

COMP-330 Fall 2019 — Weekly Schedule

Mon 10:00	**Tue** 10:00	**Wed** 10:00	**Thu** 10:00	Yanjia TR-3110
Mon 10:30	**Tue** 10:30	**Wed** 10:30	**Thu** 10:30	
Mon 11:00	**Tue** 11:00	**Wed** 11:00	**Thu** 11:00	**Fri** 11:00
Mon 11:30	**Tue** 11:30	**Wed** 11:30	**Thu** 11:30	**Fri** 11:30
Mon 12:00	**Tue** 12:00	**Wed** 12:00	**Thu** 12:00	**Fri** 12:00
Mon 12:30	**Tue** 12:30	**Wed** 12:30	**Thu** 12:30	**Fri** 12:30
Mon 13:00	Claude MA-112 course	Claude MC-110N office hours	Claude MA-112 course	Pouriya TR-3090
Mon 13:30				
Mon 14:00				**Fri** 14:00
Mon 14:30	**Tue** 14:30		**Thu** 14:30	**Fri** 14:30
Pierre-W. TR-3110	Justin TR-3110		Shiquan TR-3110	**Fri** 15:00
				Fri 15:30
Anirudha TR-3090	**Tue** 16:00	**Wed** 16:00	TA meeting ?	**Fri** 16:00
	Tue 16:30	**Wed** 16:30		**Fri** 16:30

MC = MCENG = McConnell • TR = ENGTR = Trottier

COMP-330 Fall 2019 — Weekly Schedule

Mon 10:00	**Tue** 10:00	**Wed** 10:00	**Thu** 10:00	Yanjia TR-3110
Mon 10:30	**Tue** 10:30	**Wed** 10:30	**Thu** 10:30	
Mon 11:00	**Tue** 11:00	**Wed** 11:00	**Thu** 11:00	**Fri** 11:00
Mon 11:30	**Tue** 11:30	**Wed** 11:30	**Thu** 11:30	**Fri** 11:30
CSUS **Helpdesk** **TR-3090**	**CSUS**	**CSUS**	**CSUS**	
	Claude MA-112 course	Claude MC-110N office hours	Claude MA-112 course	Pouriya TR-3090
				CSUS **Helpdesk** **TR-3090**
Pierre-W. TR-3110	Justin TR-3110		Shiquan TR-3110	
Anirudha TR-3090	**Helpdesk** **TR-3090**	**Helpdesk** **TR-3090**	**Helpdesk** **TR-3090**	

MC = MCENG = McConnell • TR = ENGTR = Trottier

Post Correspondence Problem

Post Correspondence Problem

- **Theorem:**

  The Post Correspondence Problem cannot be **decided** by any algorithm (or computer program). In particular, no algorithm can identify in a finite amount of time some instances that have a **No** outcome. However, if a solution exists, we can ALWAYS find it.

Post Correspondence Problem

Post Correspondence Problem

- **Proof**:

  Reduction technique - if **PCP** was **decidable** then another **undecidable** problem (the halting problem) would be **decidable**.

The Halting Problem

The Halting Problem

- Notice that an algorithm is a piece of text.

The Halting Problem

- Notice that an algorithm is a piece of text.
- An algorithm can receive text as input.

The Halting Problem

- Notice that an algorithm is a piece of text.
- An algorithm can receive text as input.
- An algorithm can receive the text description of an algorithm as input.

The Halting Problem

- Notice that an algorithm is a piece of text.
- An algorithm can receive text as input.
- An algorithm can receive the text description of an algorithm as input.
- The Halting Problem:
  Given two texts A and B, consider A as an algorithm and B as an input. Will algorithm A halt (as opposed to loop forever) on input B?

The Halting Problem and **PCP**

The Halting Problem and **PCP**

- Any algorithm to decide **PCP** can be converted to an algorithm to decide the Halting Problem as well.

The Halting Problem and **PCP**

- Any algorithm to decide **PCP** can be converted to an algorithm to decide the Halting Problem as well.

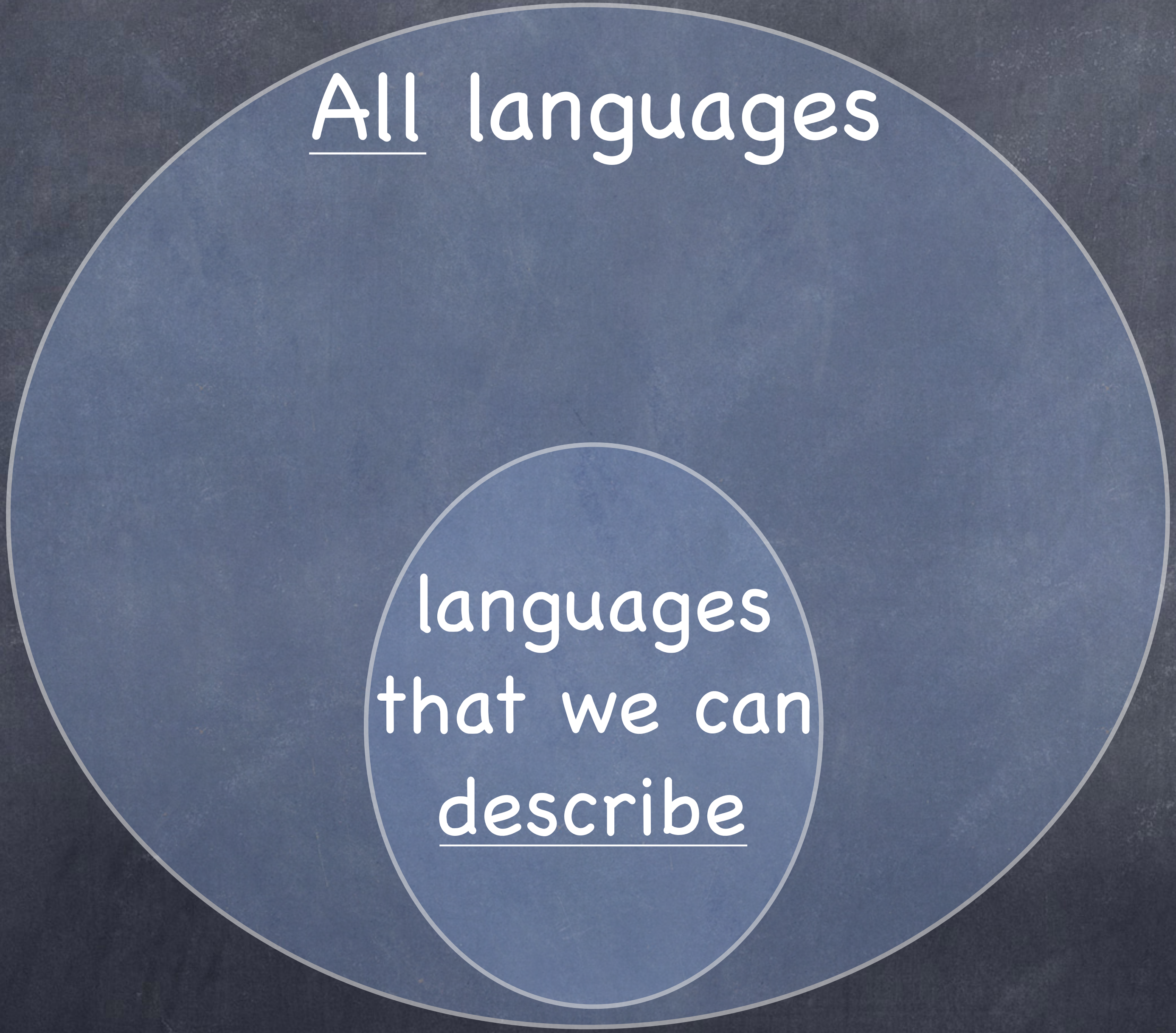
- **Conclusion**: **PCP** cannot be decided either.

Computability Theory

Computability Theory

All languages

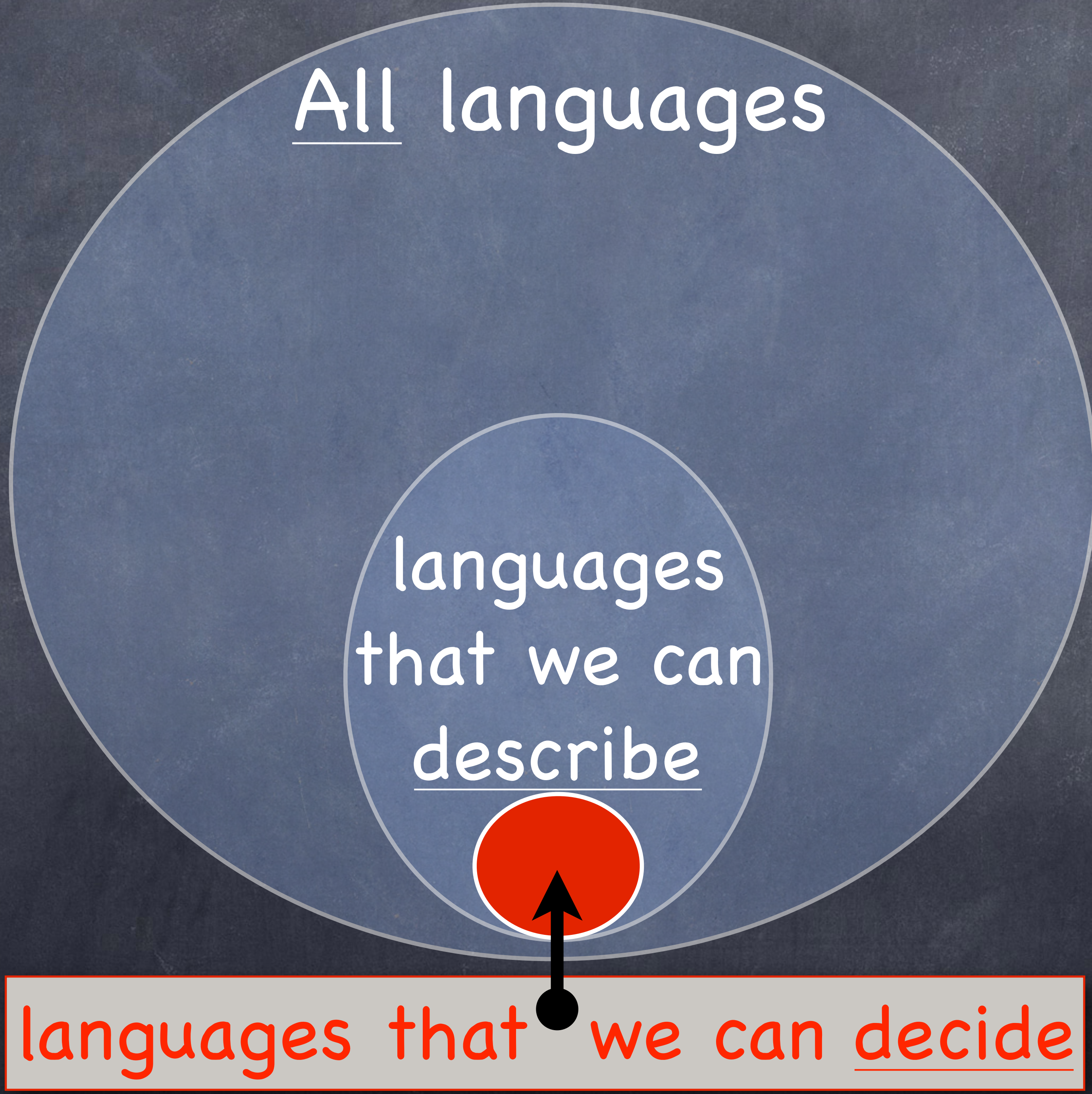
Computability Theory



All languages

languages that we can describe

Computability Theory



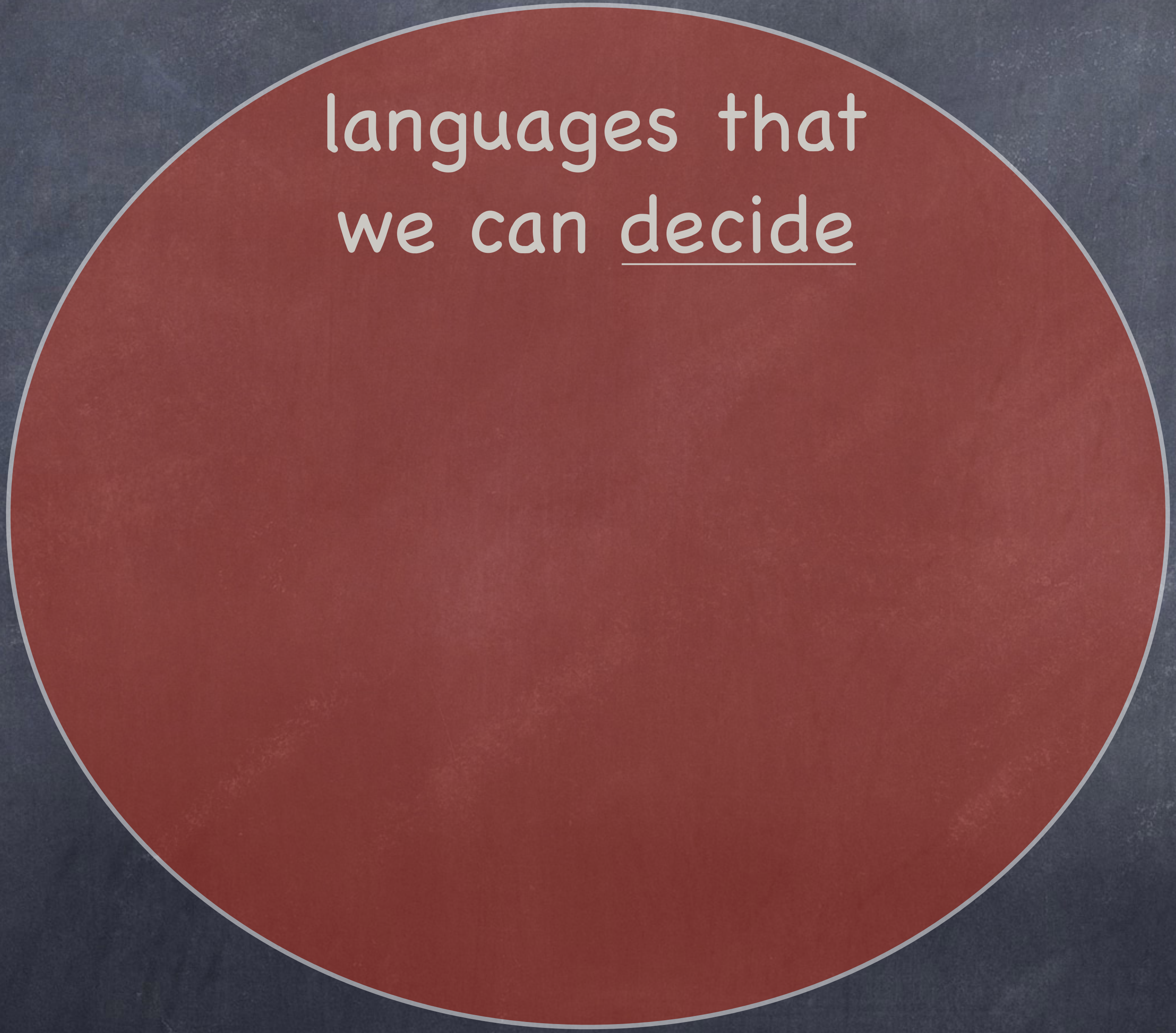
All languages

languages that we can describe

languages that we can decide

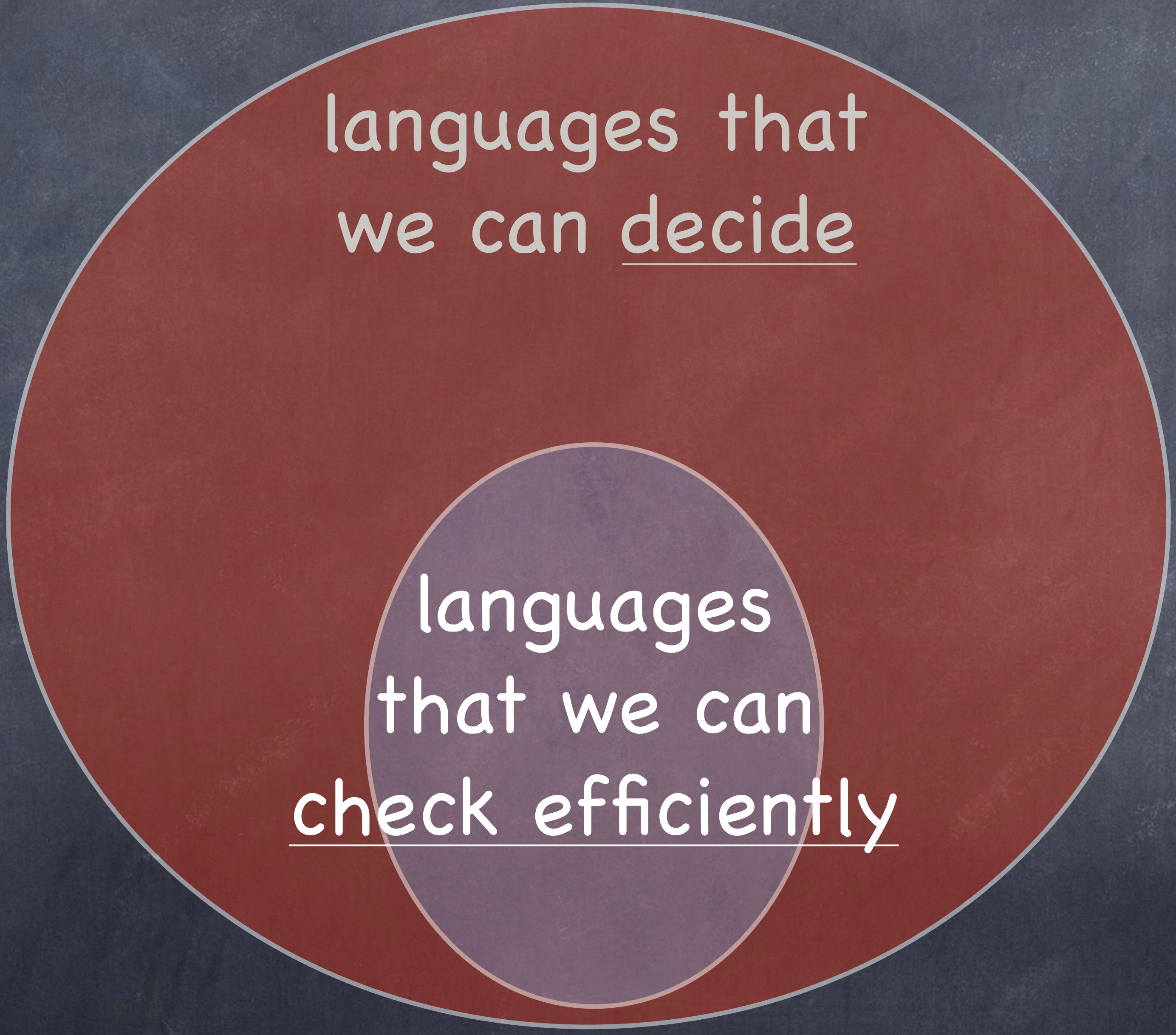
Complexity Theory

Complexity Theory



languages that we can decide

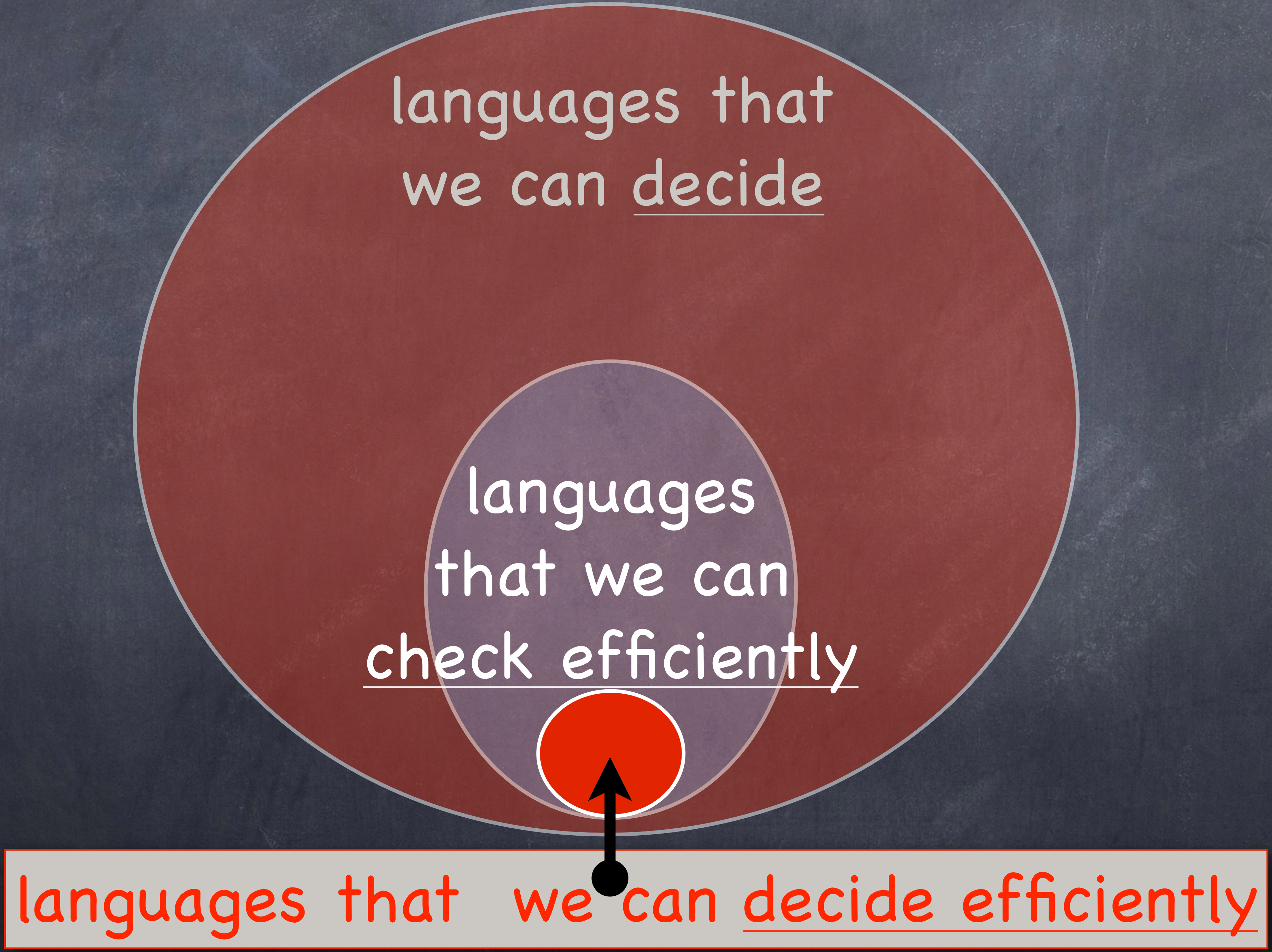
Complexity Theory



languages that we can decide

languages that we can check efficiently

Complexity Theory

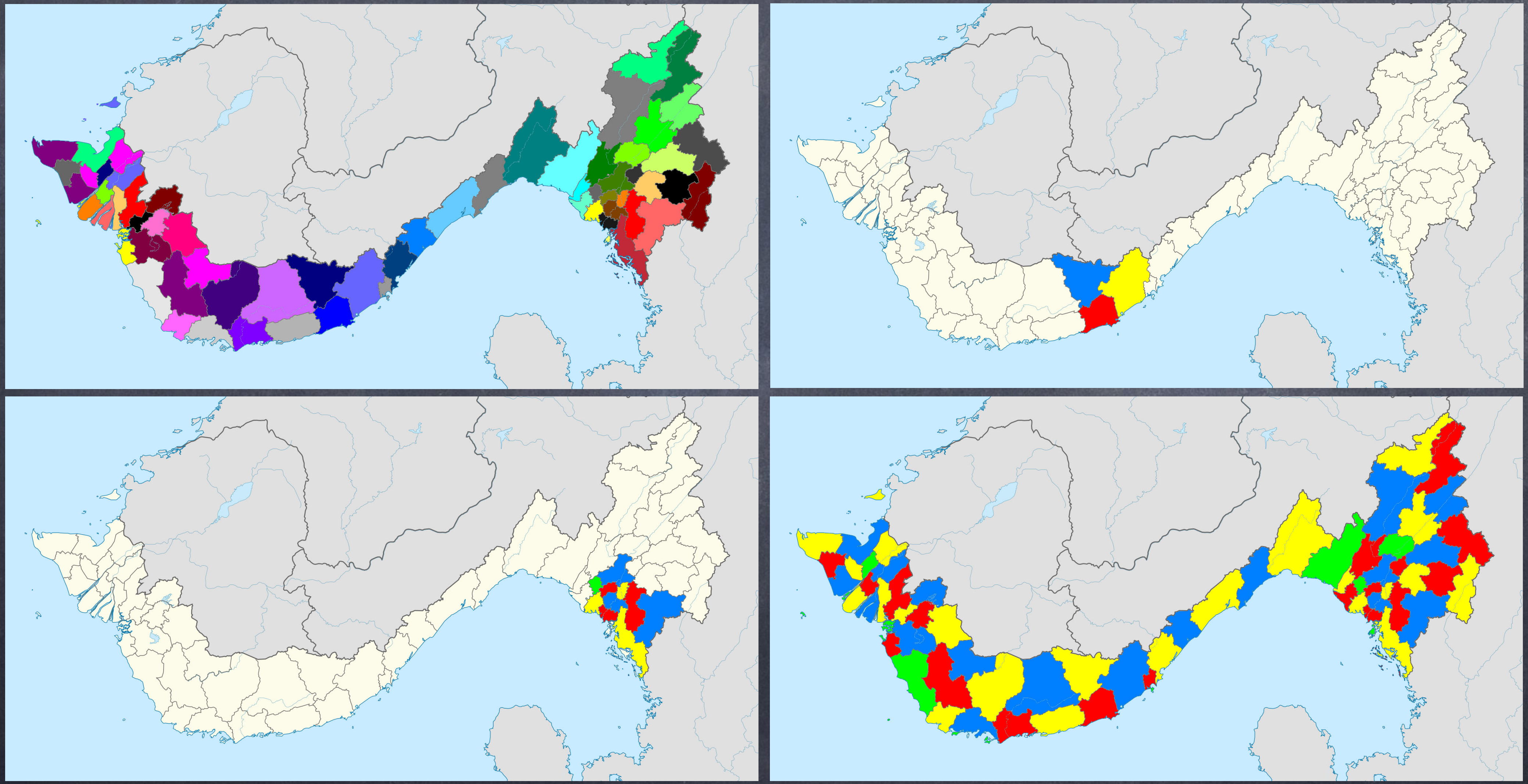


languages that we can decide

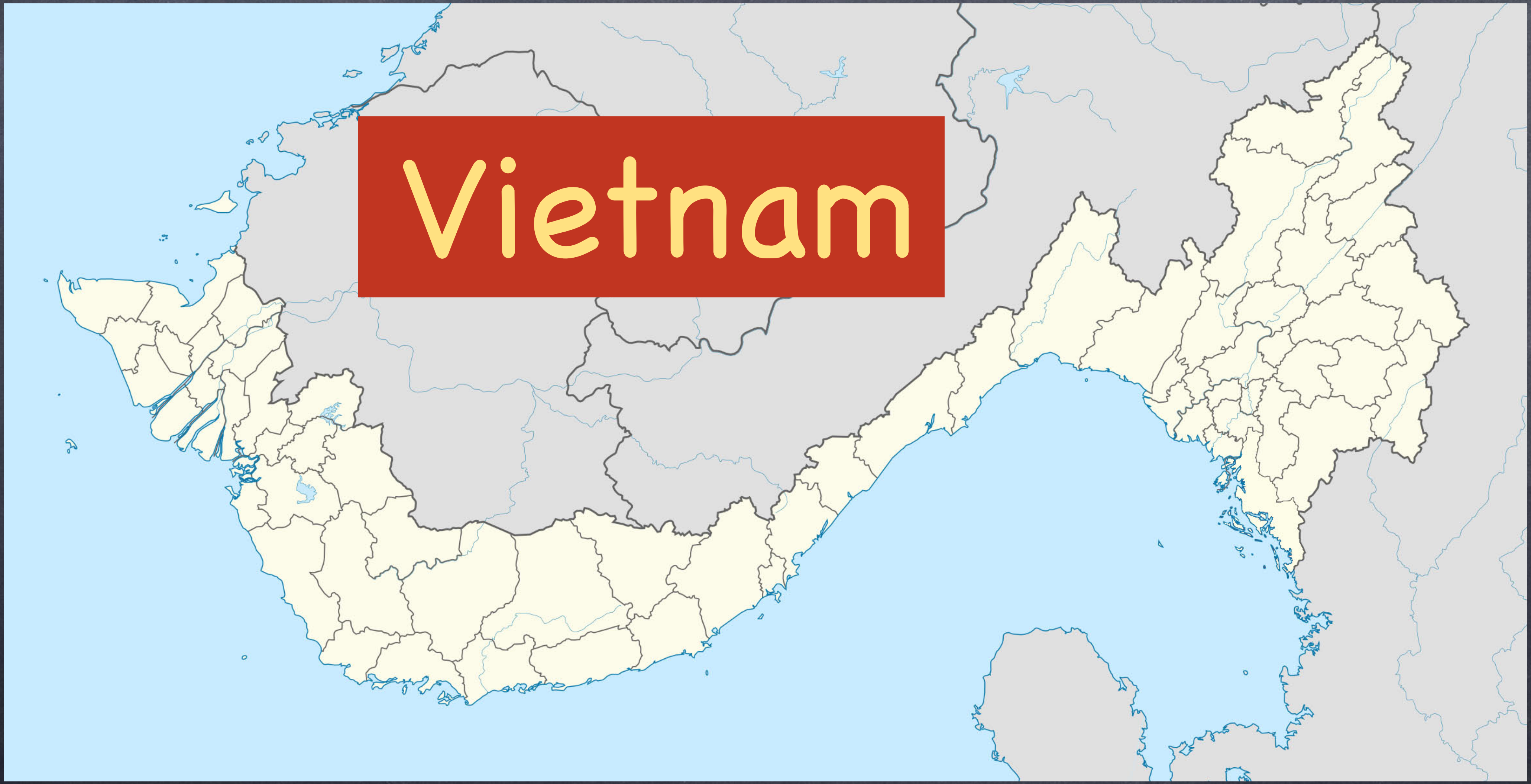
languages that we can check efficiently

languages that we can decide efficiently

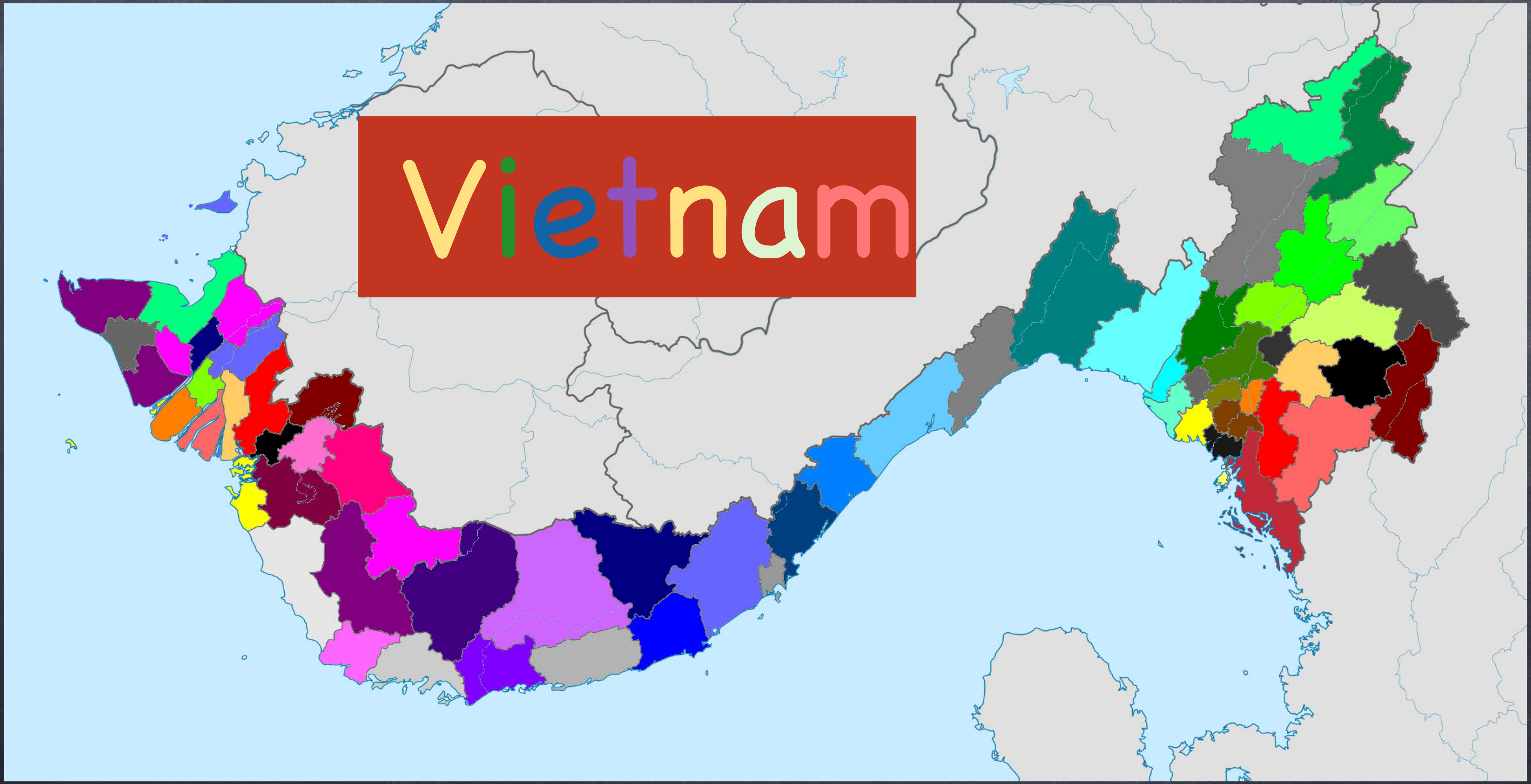
Not all problems were born equal...



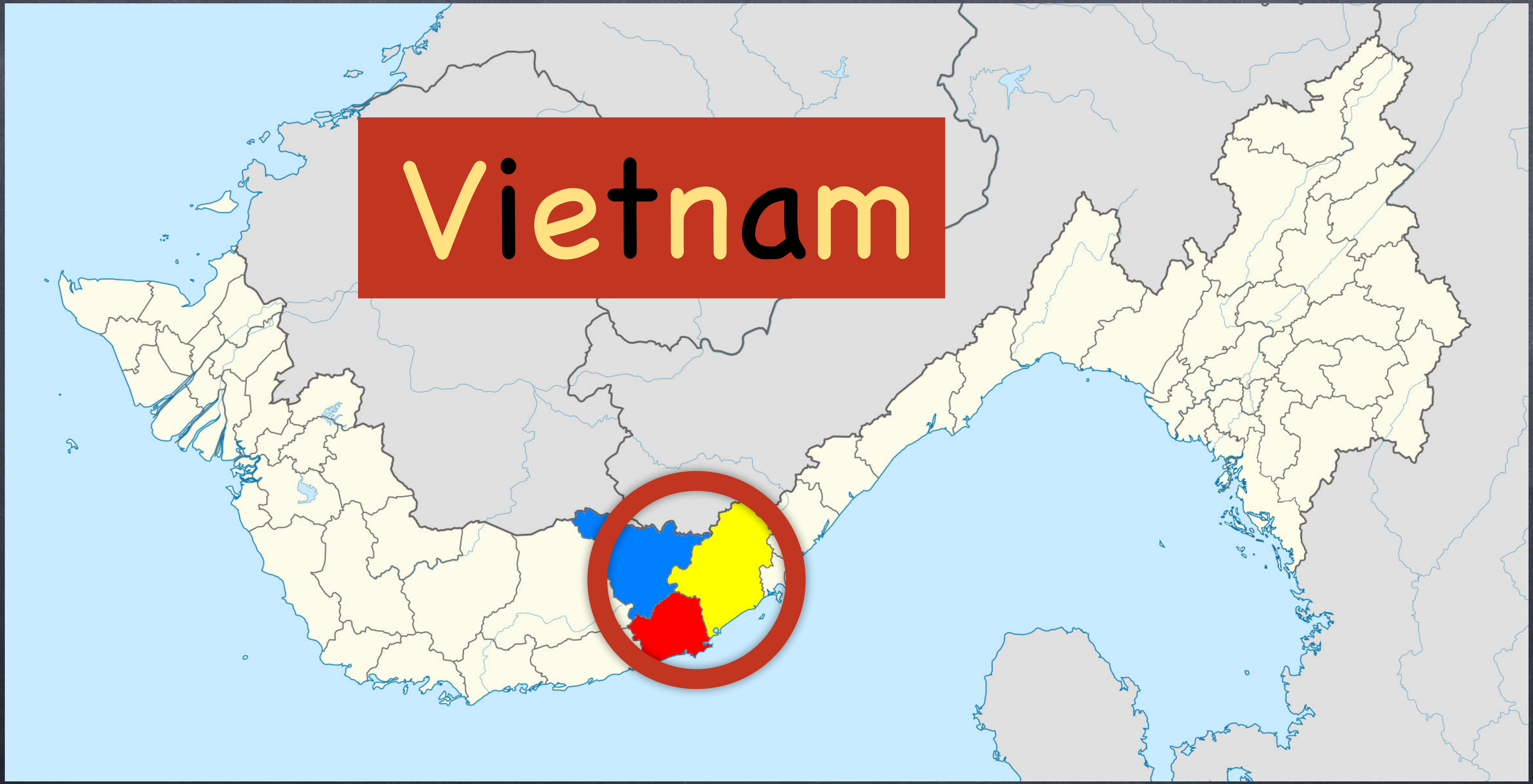
Is it possible to paint a colour on each region (province) of a map so that no neighbours are of the same colour?



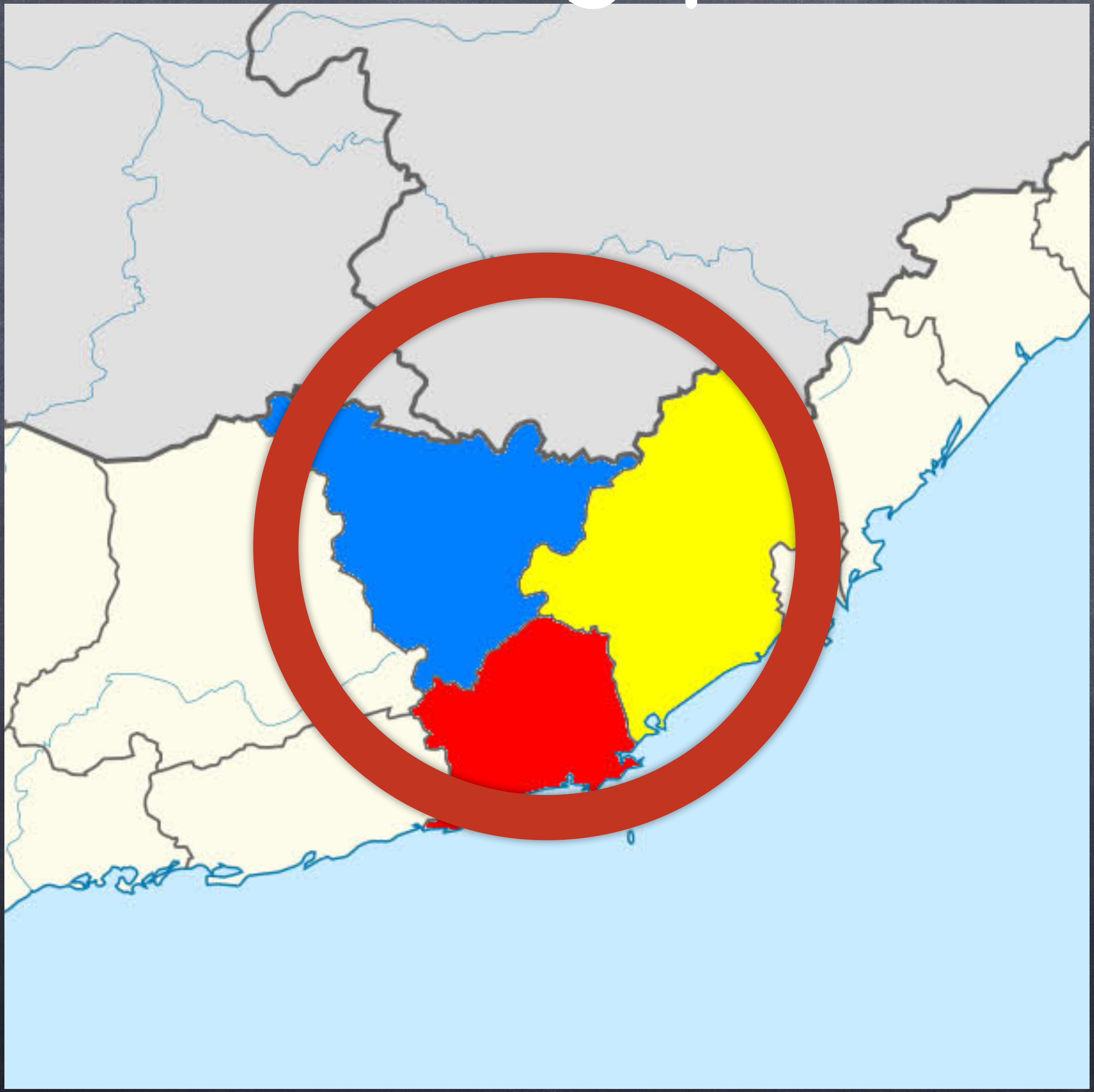
Obviously, **yes**, if you can use as many colours as you like...



2-colouring problem

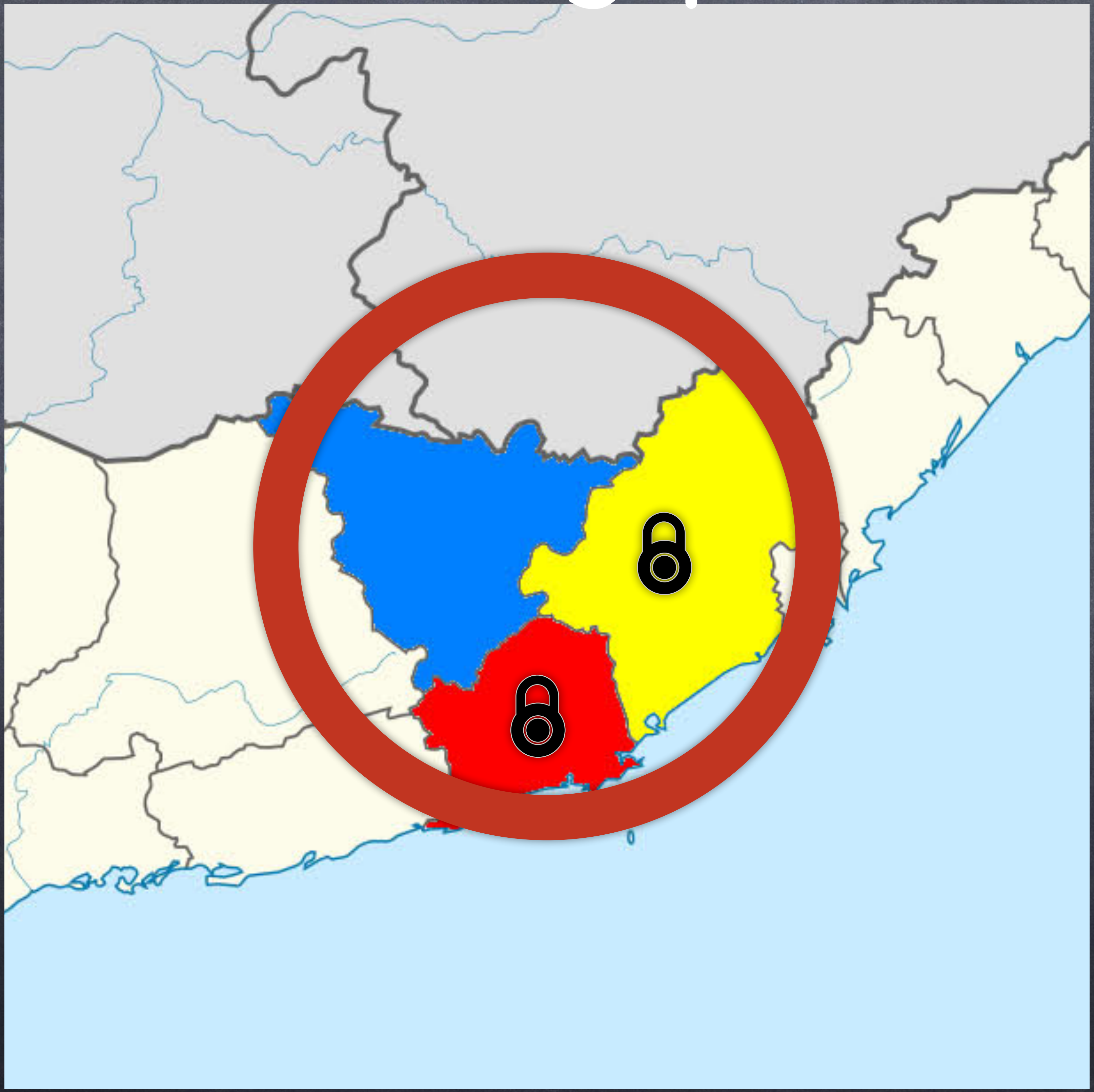


2-colouring problem

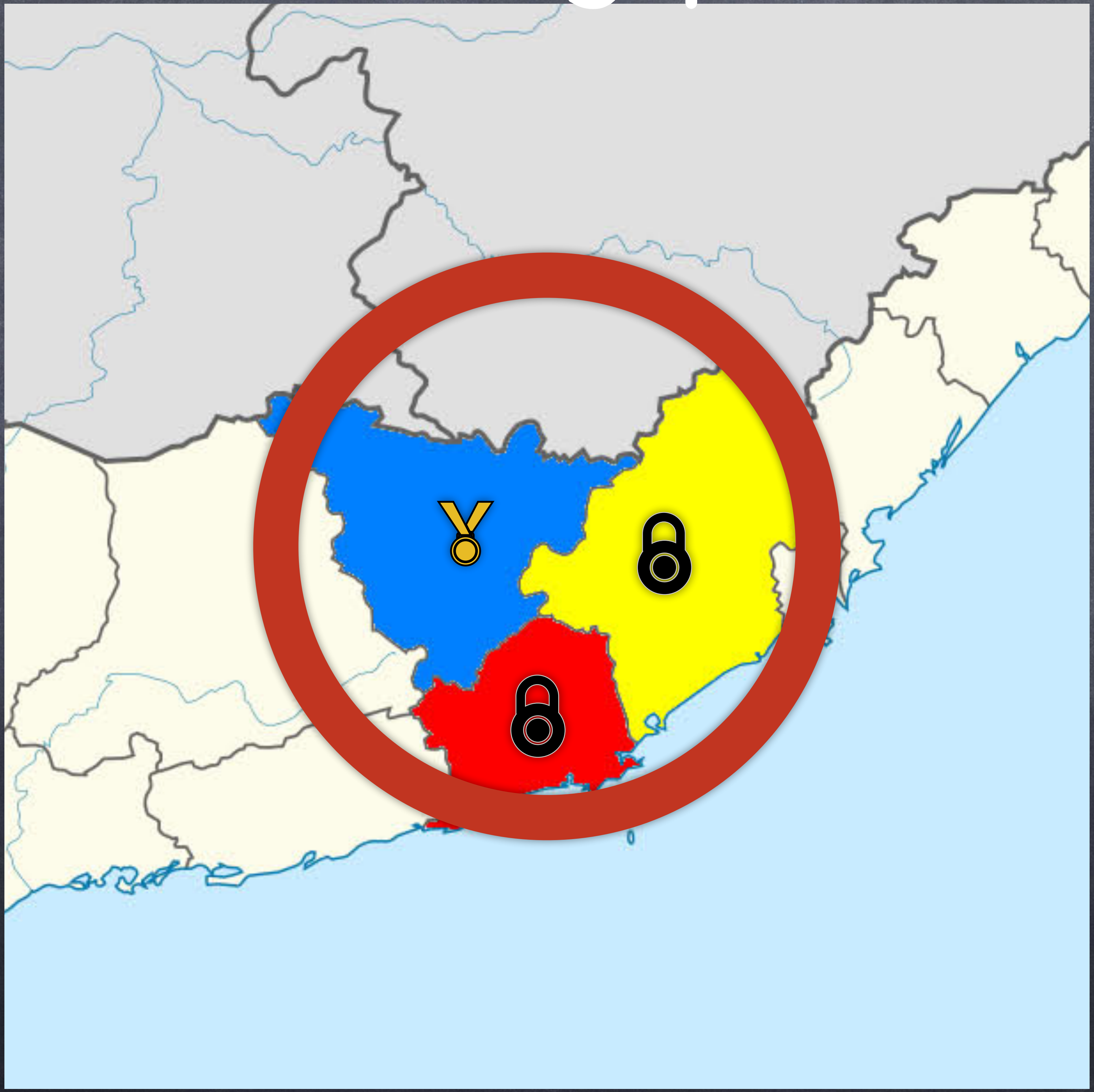


2-colouring problem

2-colouring problem

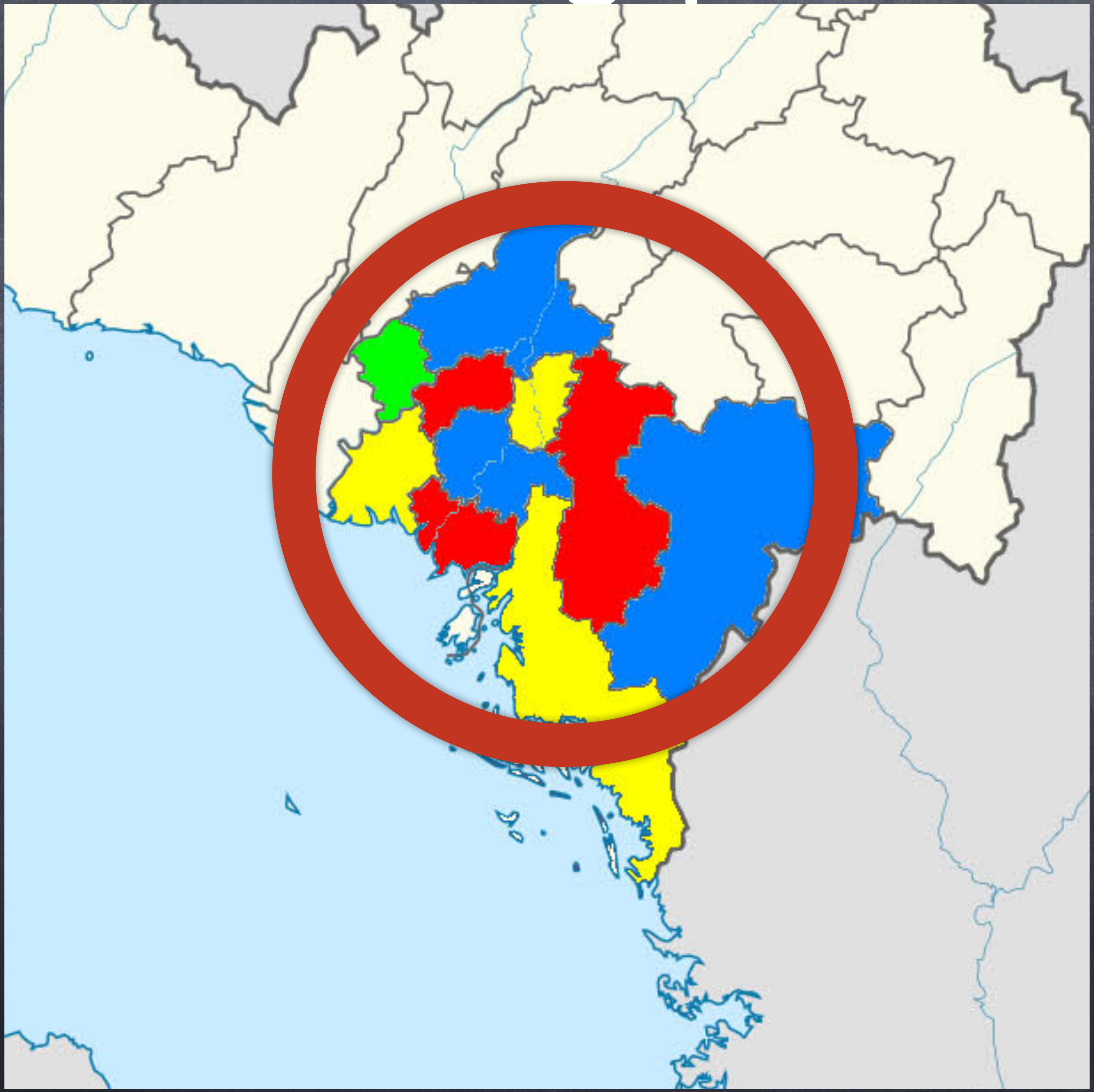


2-colouring problem

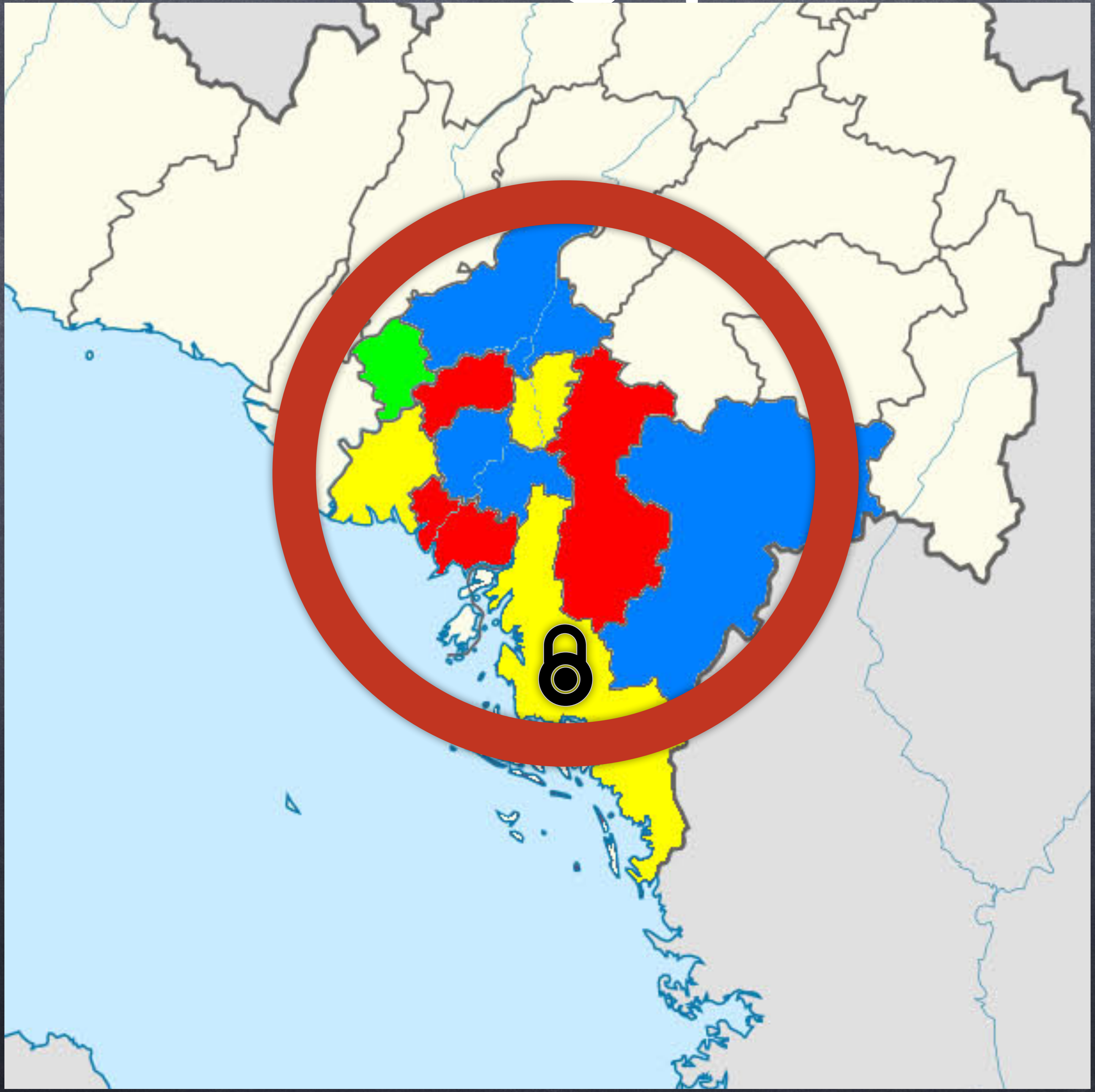


3-colouring problem

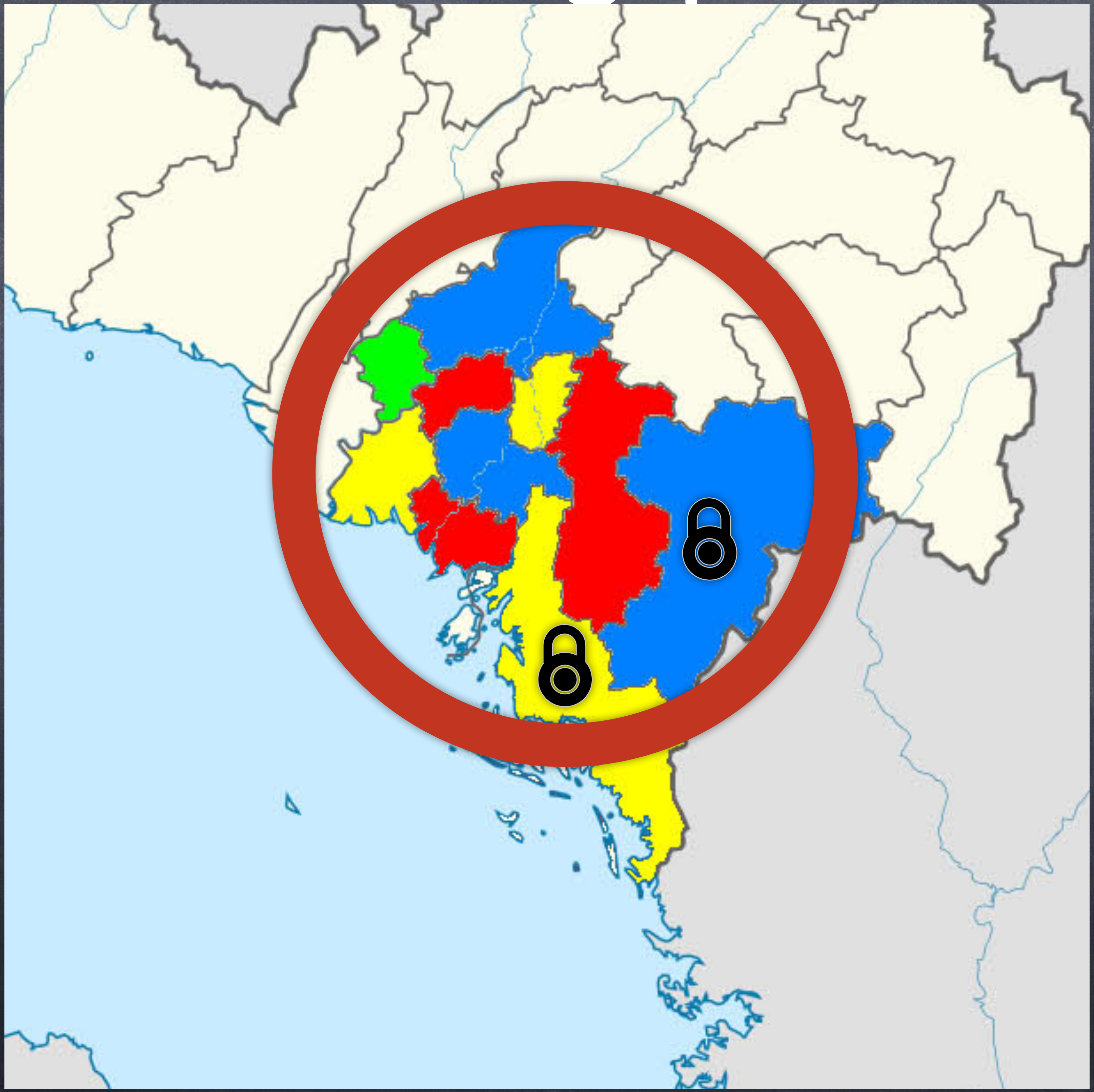
3-colouring problem



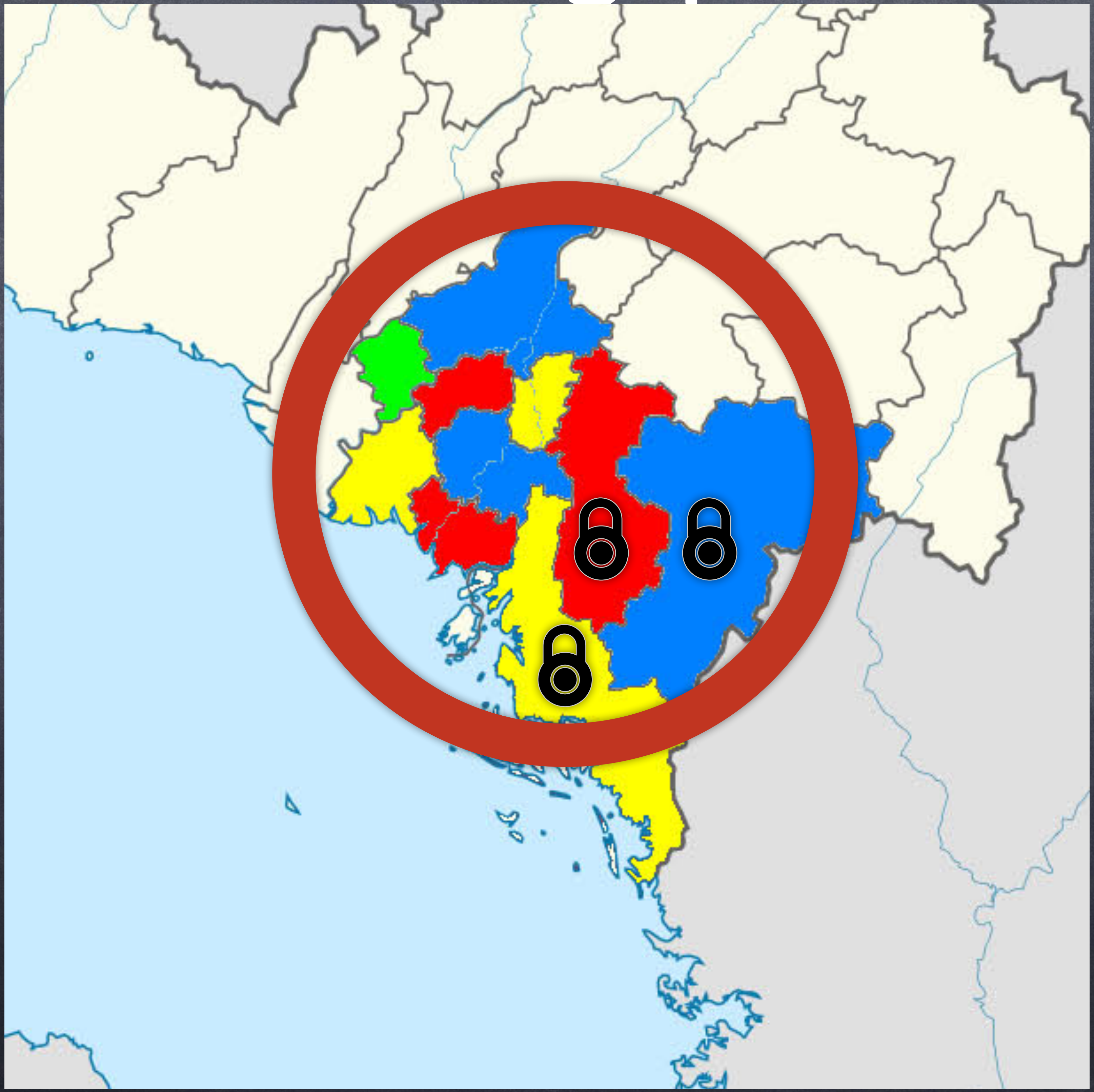
3-colouring problem



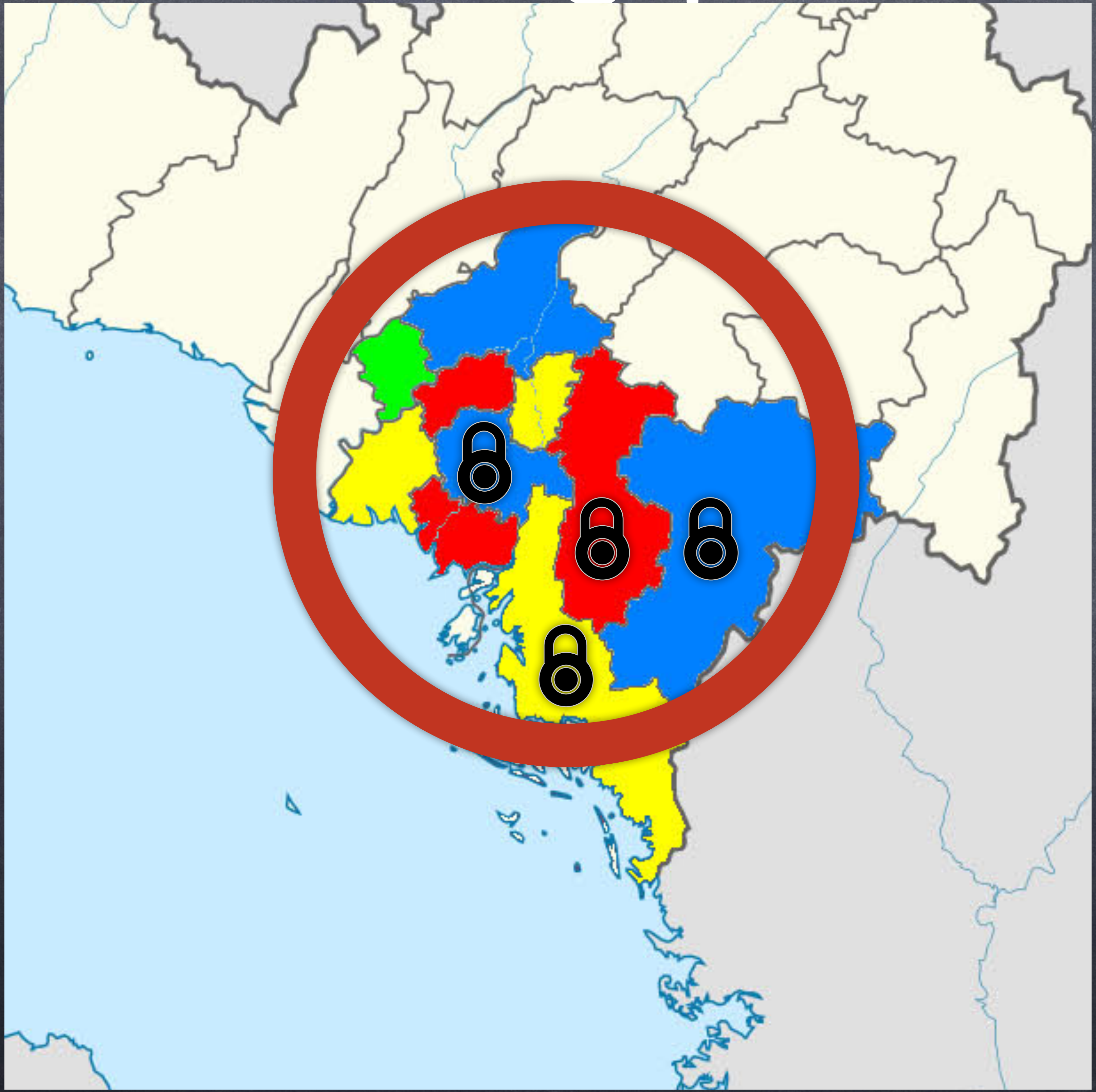
3-colouring problem



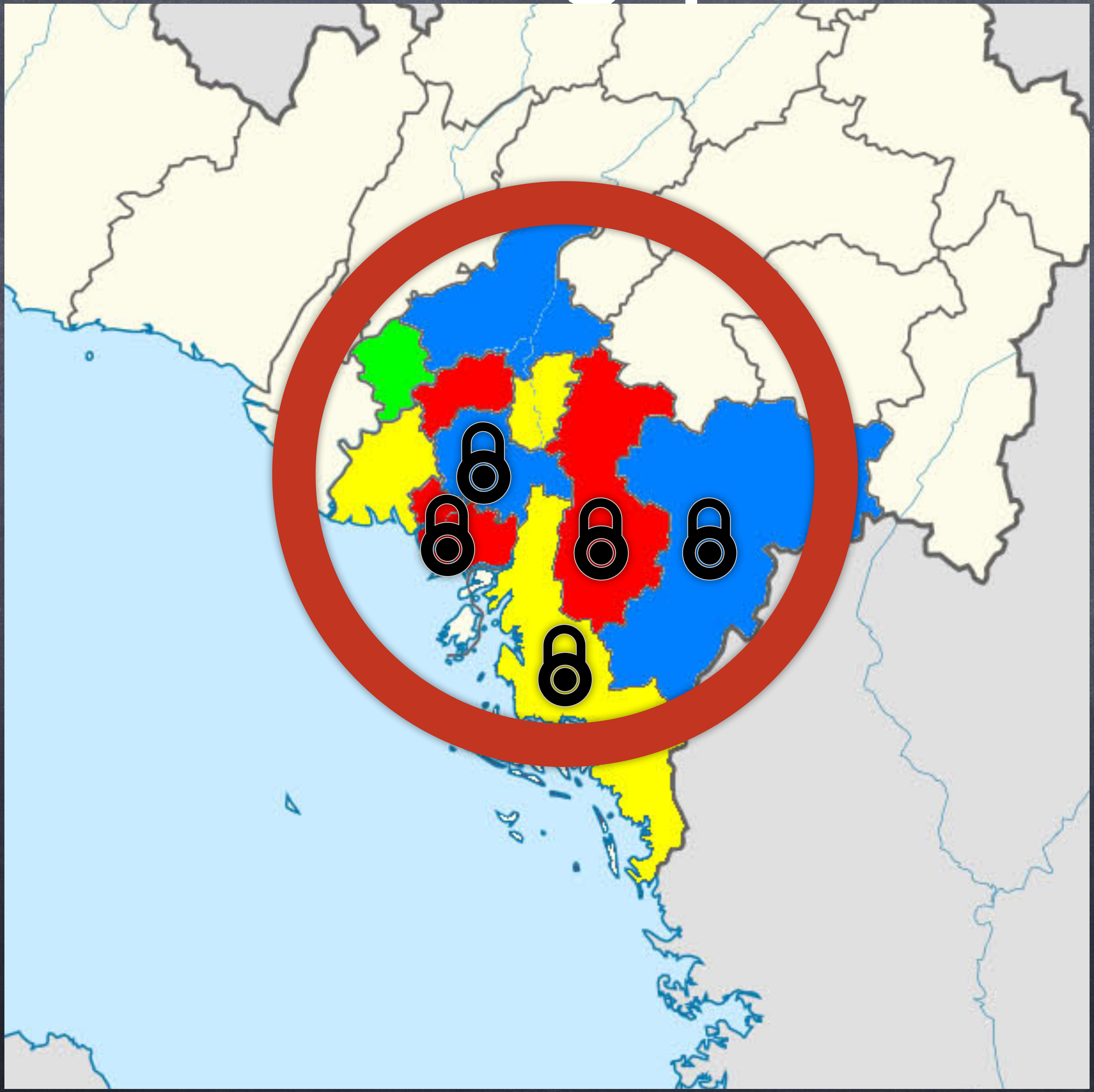
3-colouring problem



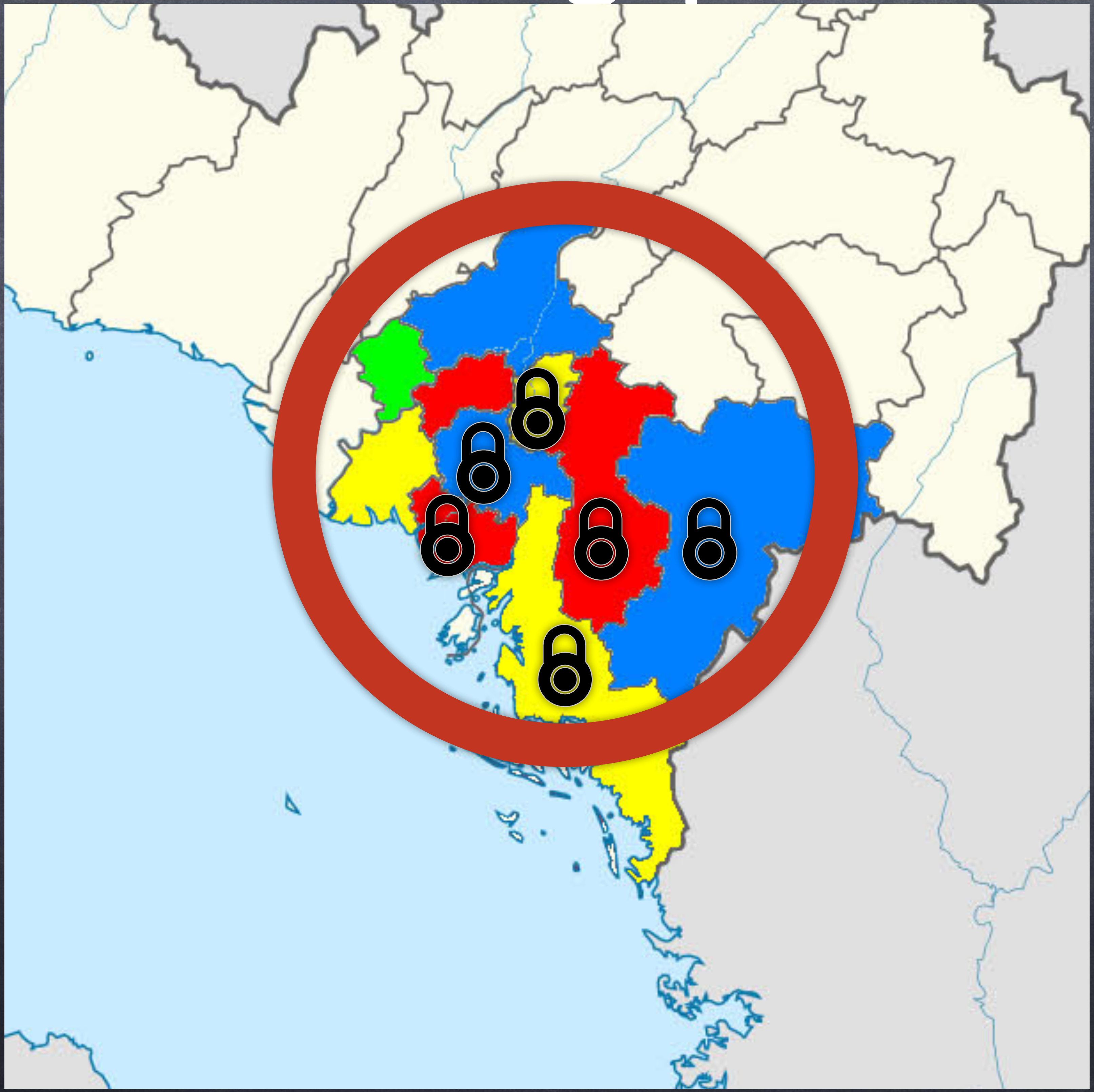
3-colouring problem



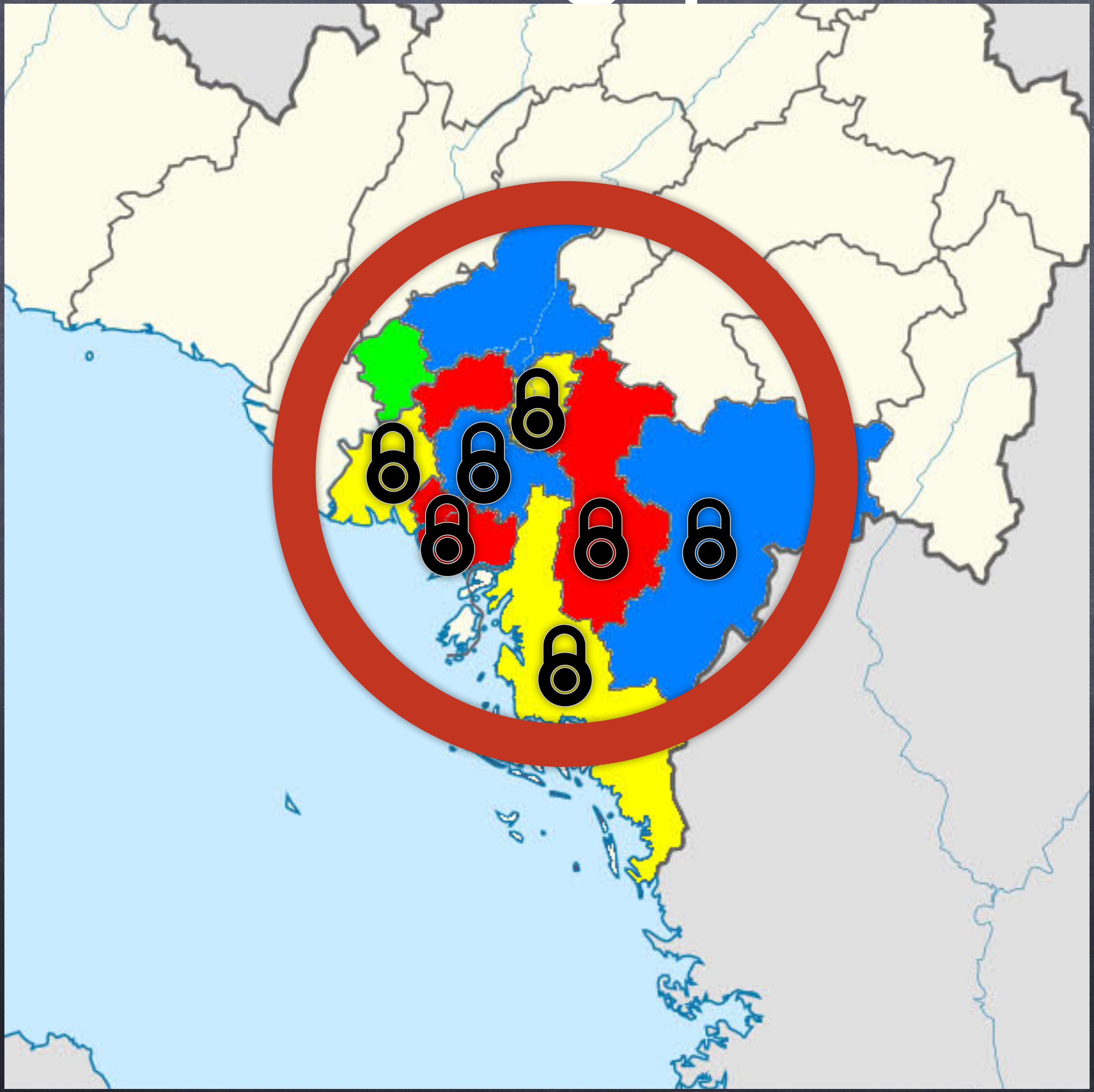
3-colouring problem



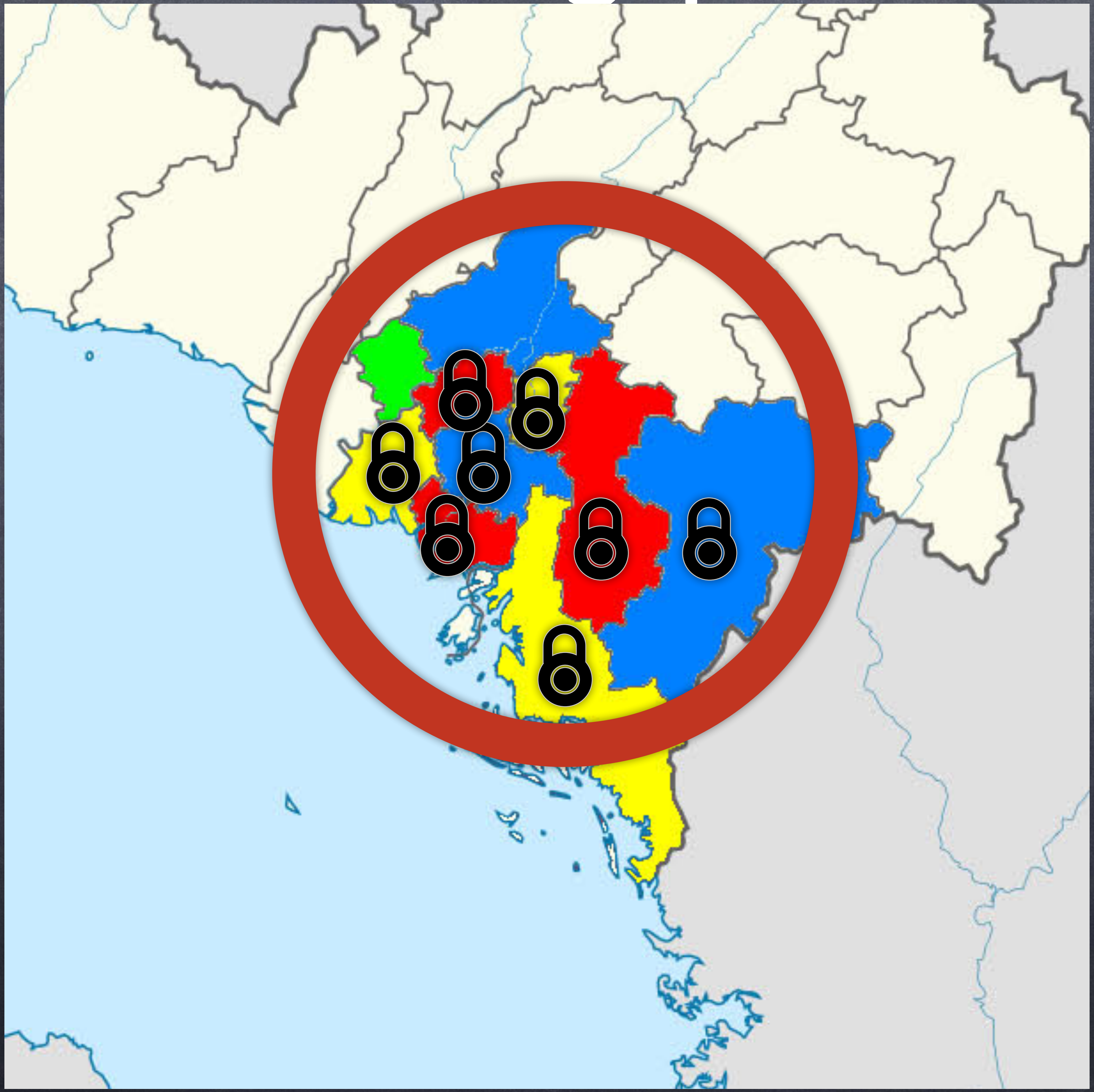
3-colouring problem



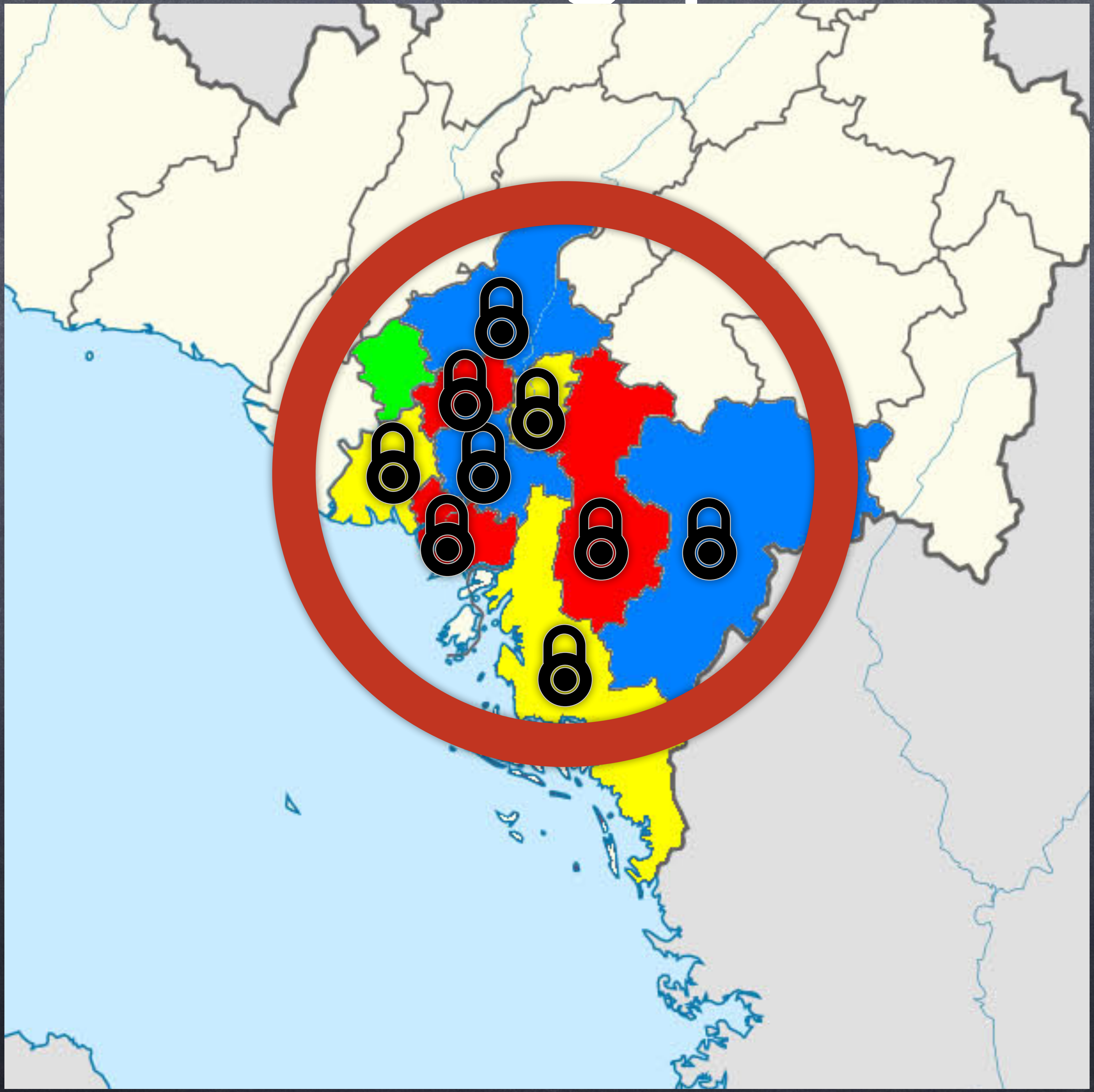
3-colouring problem



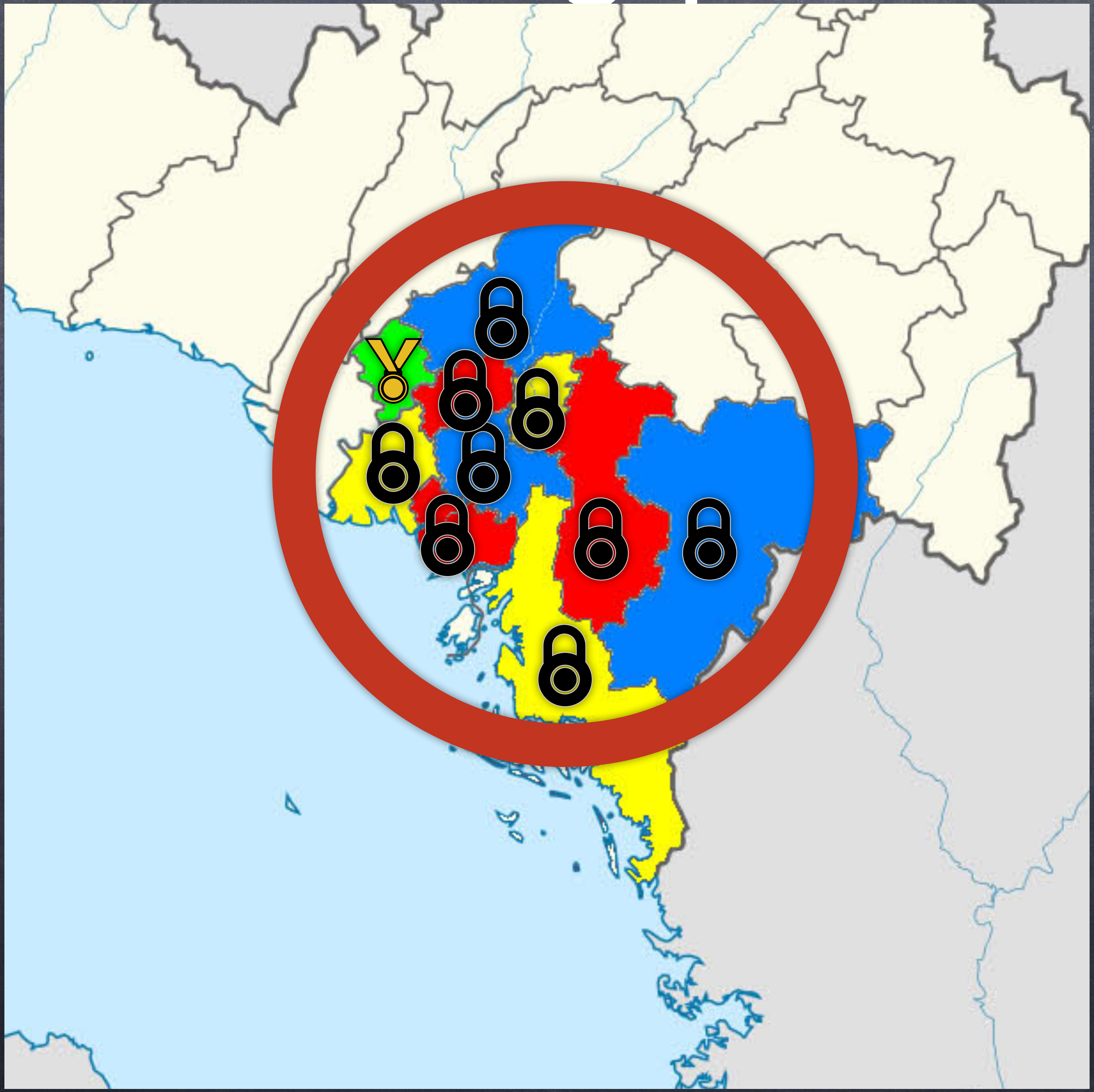
3-colouring problem



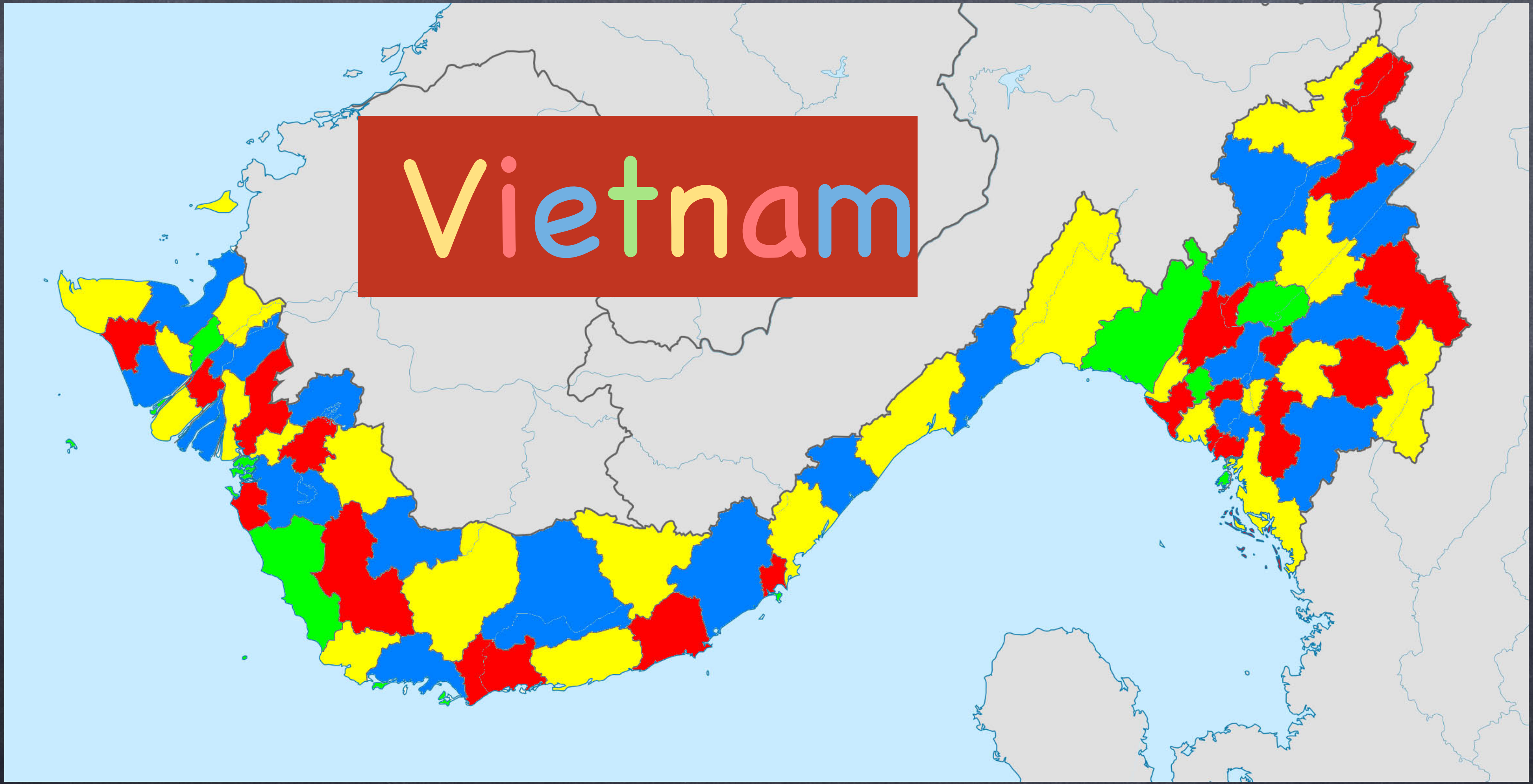
3-colouring problem



3-colouring problem



4-colouring problem



K-colouring of Maps (planar graphs)

http://people.math.gatech.edu/~thomas/FC/fourcolor.html

K-colouring of Maps (planar graphs)

- K=1, only the maps with zero or one region are 1-colourable.

http://people.math.gatech.edu/~thomas/FC/fourcolor.html

K-colouring of Maps (planar graphs)

- K=1, only the maps with zero or one region are 1-colourable.
- K=2, easy to decide. Impossible as soon as 3 regions touch each other.

http://people.math.gatech.edu/~thomas/FC/fourcolor.html

K-colouring of Maps (planar graphs)

- K=1, only the maps with zero or one region are 1-colourable.
- K=2, easy to decide. Impossible as soon as 3 regions touch each other.
- K=3, No known efficient algorithm to decide. However it is easy to verify a solution.

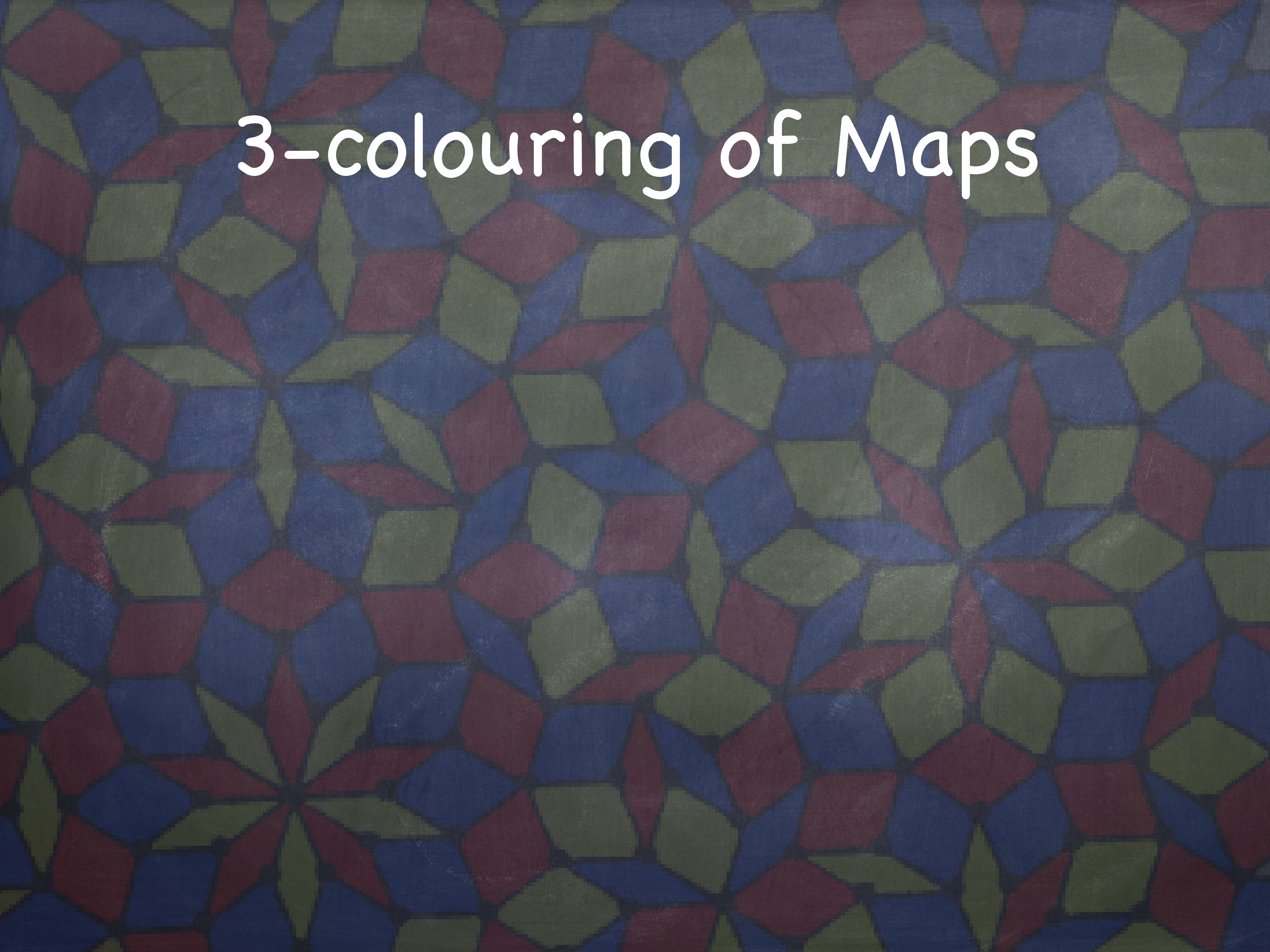
http://people.math.gatech.edu/~thomas/FC/fourcolor.html

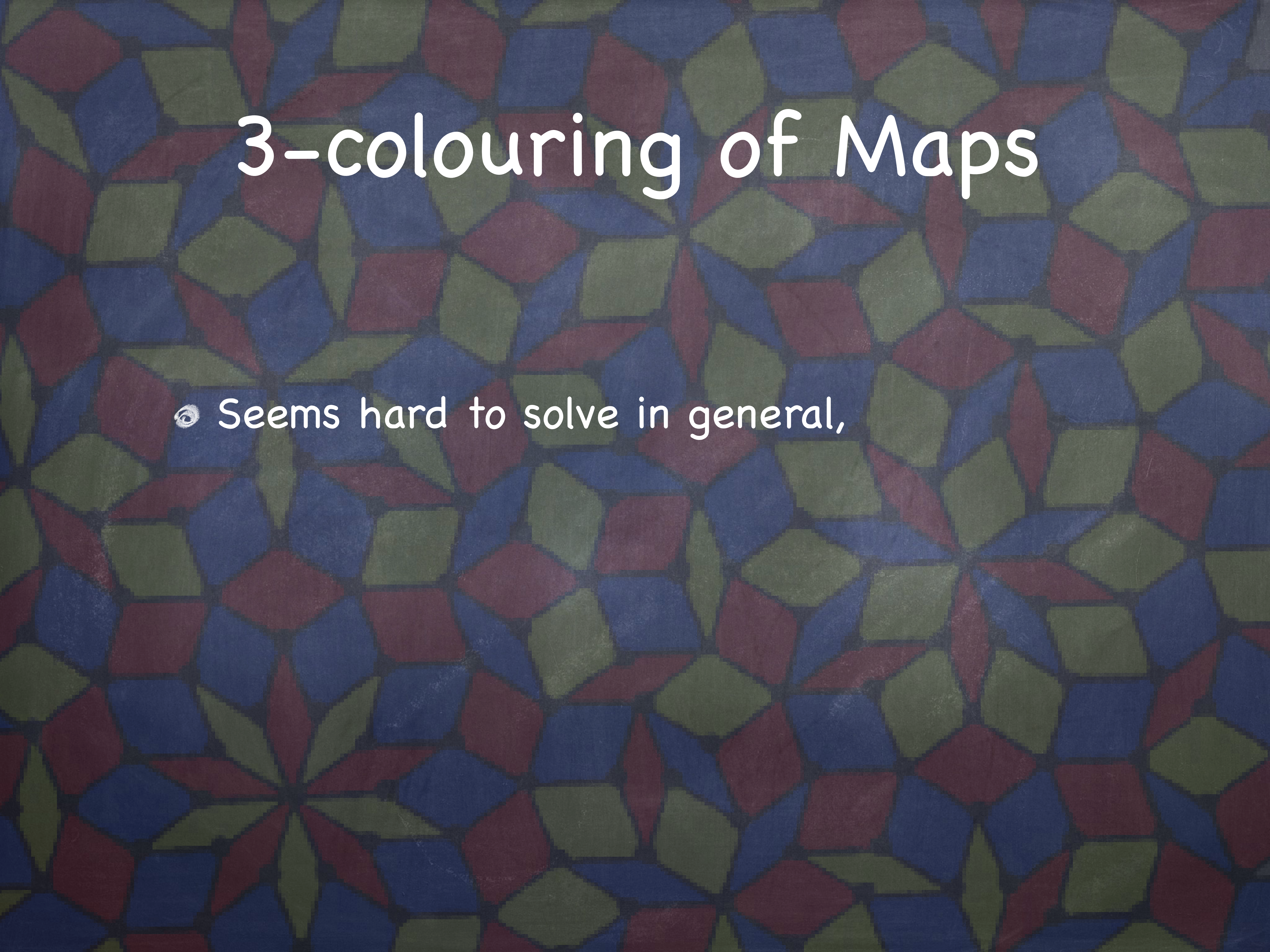
K-colouring of Maps (planar graphs)

- K=1, only the maps with zero or one region are 1-colourable.
- K=2, easy to decide. Impossible as soon as 3 regions touch each other.
- K=3, No known efficient algorithm to decide. However it is easy to verify a solution.
- K≥4, all maps are K-colourable. (looong proof) Not easy to find a K-colouring. However it is easy to verify a solution.

http://people.math.gatech.edu/~thomas/FC/fourcolor.html

3-colouring of Maps





3-colouring of Maps

- Seems hard to solve in general,

3-colouring of Maps

- Seems hard to solve in general,
- Is easy to verify when a solution is given,

3-colouring of Maps

- Seems hard to solve in general,
- Is easy to verify when a solution is given,
- Is a special type of problem (**NP**-complete) because an efficient solution to it would yield efficient solutions to MANY similar problems !

Examples of **NP**-Complete Problems

Examples of **NP**-Complete Problems

- SAT: given a boolean formula, is there an assignment of the variables making the formula evaluate to true ?

Examples of **NP**-Complete Problems

- SAT: given a boolean formula, is there an assignment of the variables making the formula evaluate to true ?
- Travelling Salesman: given a set of cities and distances between them, what is the shortest route to visit each city once.

Examples of **NP**-Complete Problems

- SAT: given a boolean formula, is there an assignment of the variables making the formula evaluate to true ?
- Travelling Salesman: given a set of cities and distances between them, what is the shortest route to visit each city once.
- KnapSack: given items with various weights, is there of subset of them of total weight K.

NP-Complete Problems

NP-Complete Problems

- Many practical problems are **NP**-complete.

NP-Complete Problems

- Many practical problems are **NP**-complete.
- If any of them is easy, they are all easy.

NP-Complete Problems

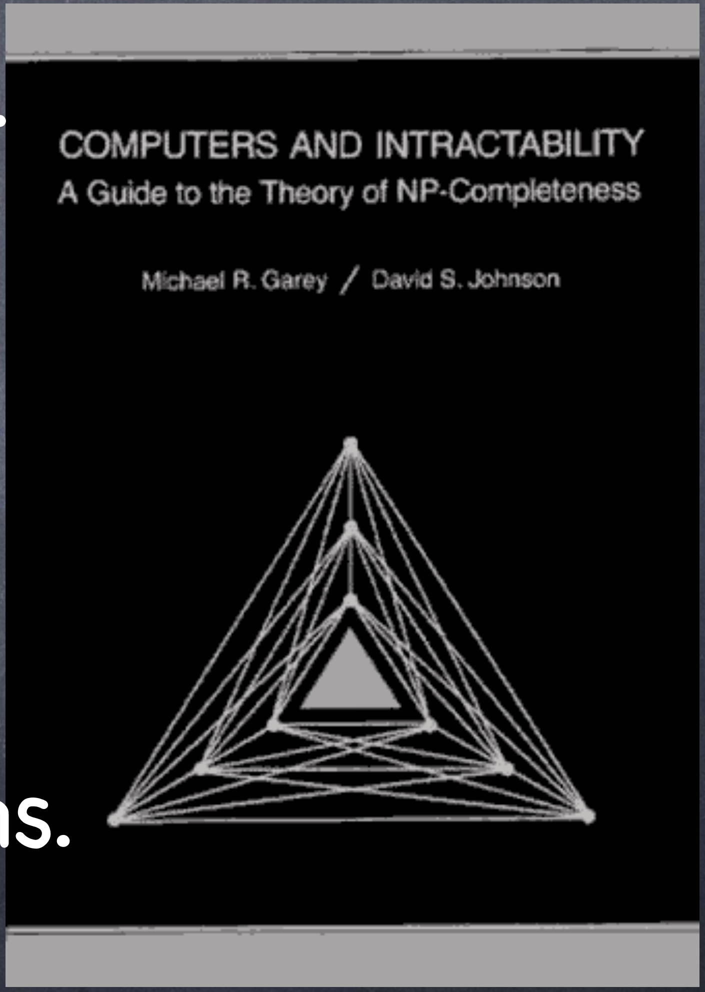
- Many practical problems are **NP**-complete.
- If any of them is easy, they are all easy.
- In practice, some of them may be solved efficiently in some special cases.

NP-Complete Problems

- Many practical problems are **NP**-complete.
- If any of them is easy, they are all easy.
- In practice, some of them may be solved efficiently in some special cases.
- Some books list hundreds of such problems.

NP-Complete Problems

- Many practical problems are **NP**-complete.
- If any of them is easy, they are all easy.
- In practice, some of them may be solved efficiently in some special cases.
- Some books list hundreds of such problems.



COMPUTERS AND INTRACTABILITY

A Guide to the Theory of NP-Completeness

Michael R. Garey / David S. Johnson

Tractable Problems (**P**)

Tractable Problems (**P**)

- 2-colorability of maps.

Tractable Problems (**P**)

- 2-colorability of maps.
- Primality testing. (but probably not factoring)

Tractable Problems (P)

- 2-colorability of maps.
- Primality testing. (but probably not factoring)
- Solving NxNxN Rubik's cube.

Tractable Problems (**P**)

- 2-colorability of maps.
- Primality testing. (but probably not factoring)
- Solving NxNxN Rubik's cube.
- Finding a word in a dictionary.

Tractable Problems (**P**)

- 2-colorability of maps.
- Primality testing. (but probably not factoring)
- Solving NxNxN Rubik's cube.
- Finding a word in a dictionary.
- Sorting elements.

Tractable Problems (**P**)

Tractable Problems (**P**)

- Fortunately, many practical problems are tractable. The name **P** stands for Polynomial-Time computable.

Tractable Problems (**P**)

- Fortunately, many practical problems are tractable. The name **P** stands for Polynomial-Time computable.
- Computer Science studies mostly techniques to approach and find efficient solutions to tractable problems.

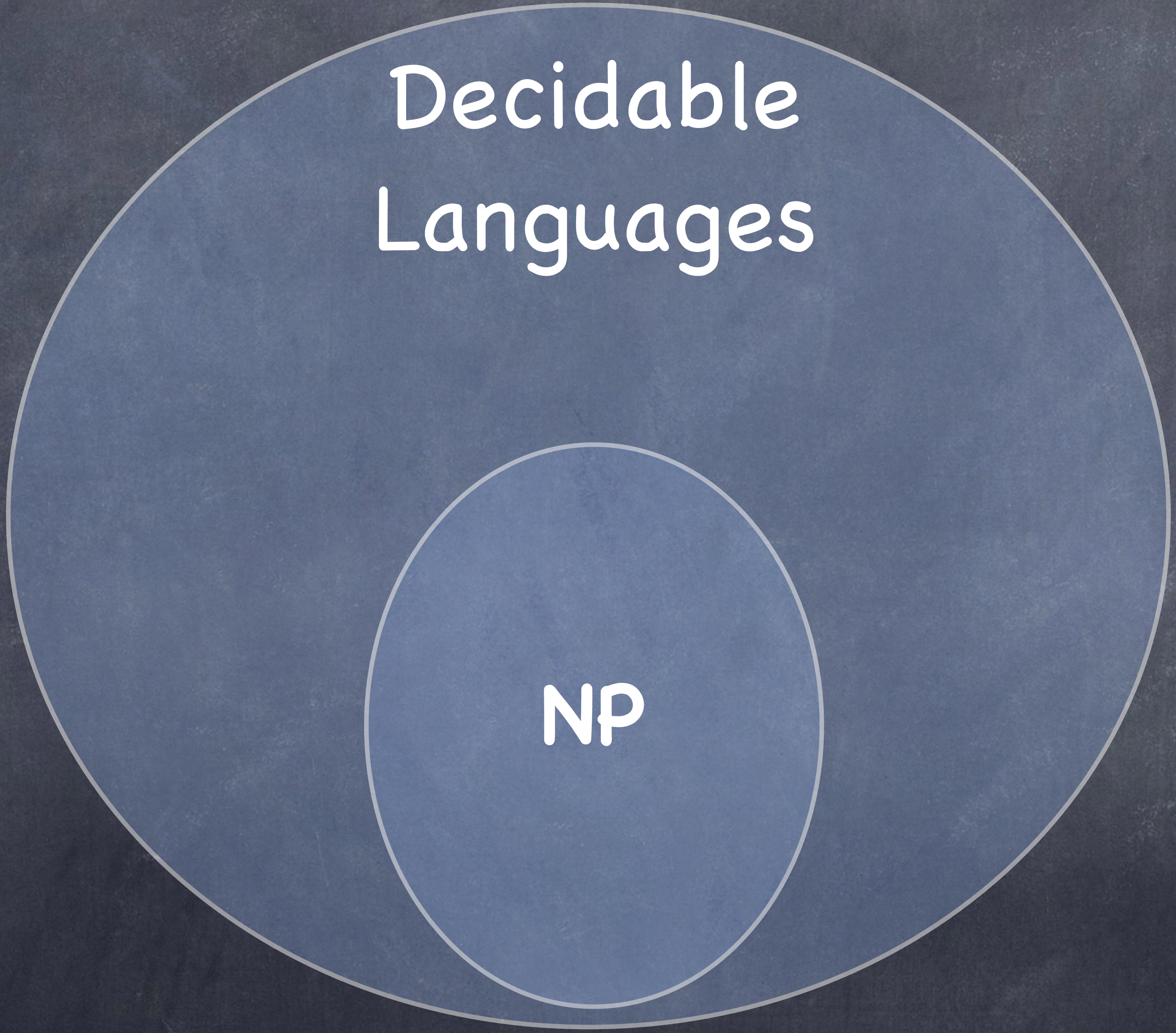
Tractable Problems (**P**)

- Fortunately, many practical problems are tractable. The name **P** stands for Polynomial-Time computable.
- Computer Science studies mostly techniques to approach and find efficient solutions to tractable problems.
- Some problems may be efficiently solvable but we might not be able to **prove** that…

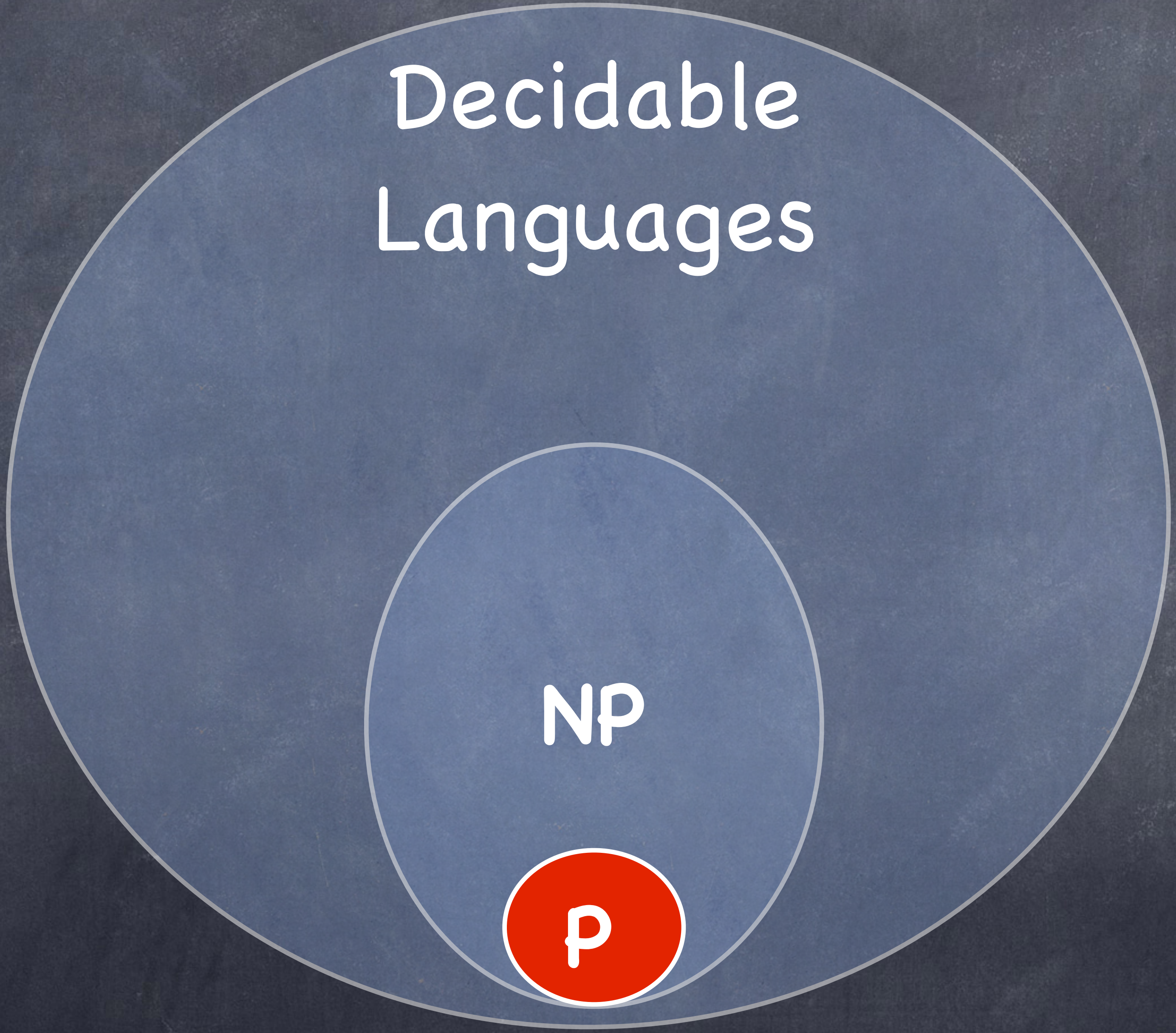
Complexity Theory

Decidable Languages

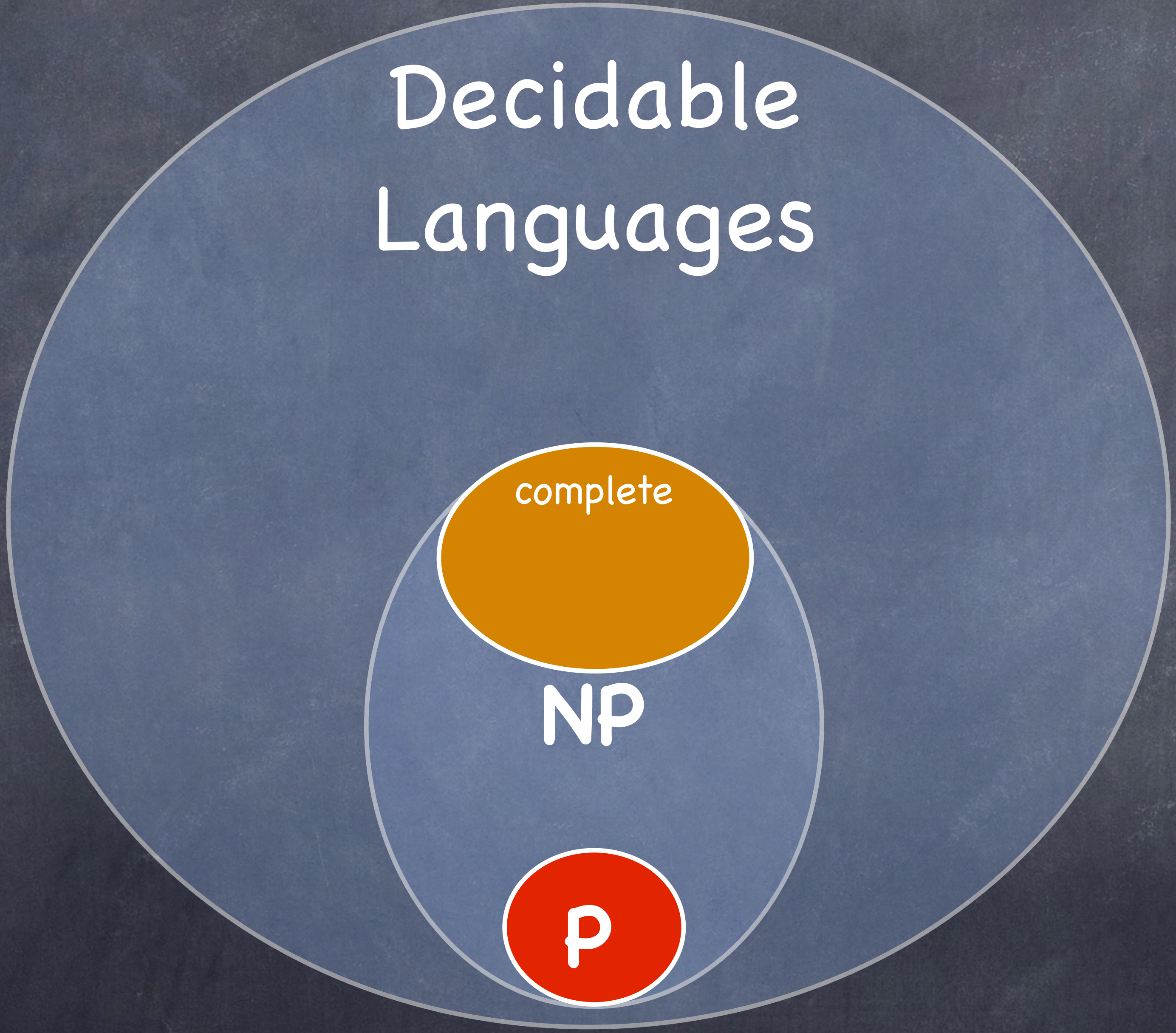
Complexity Theory



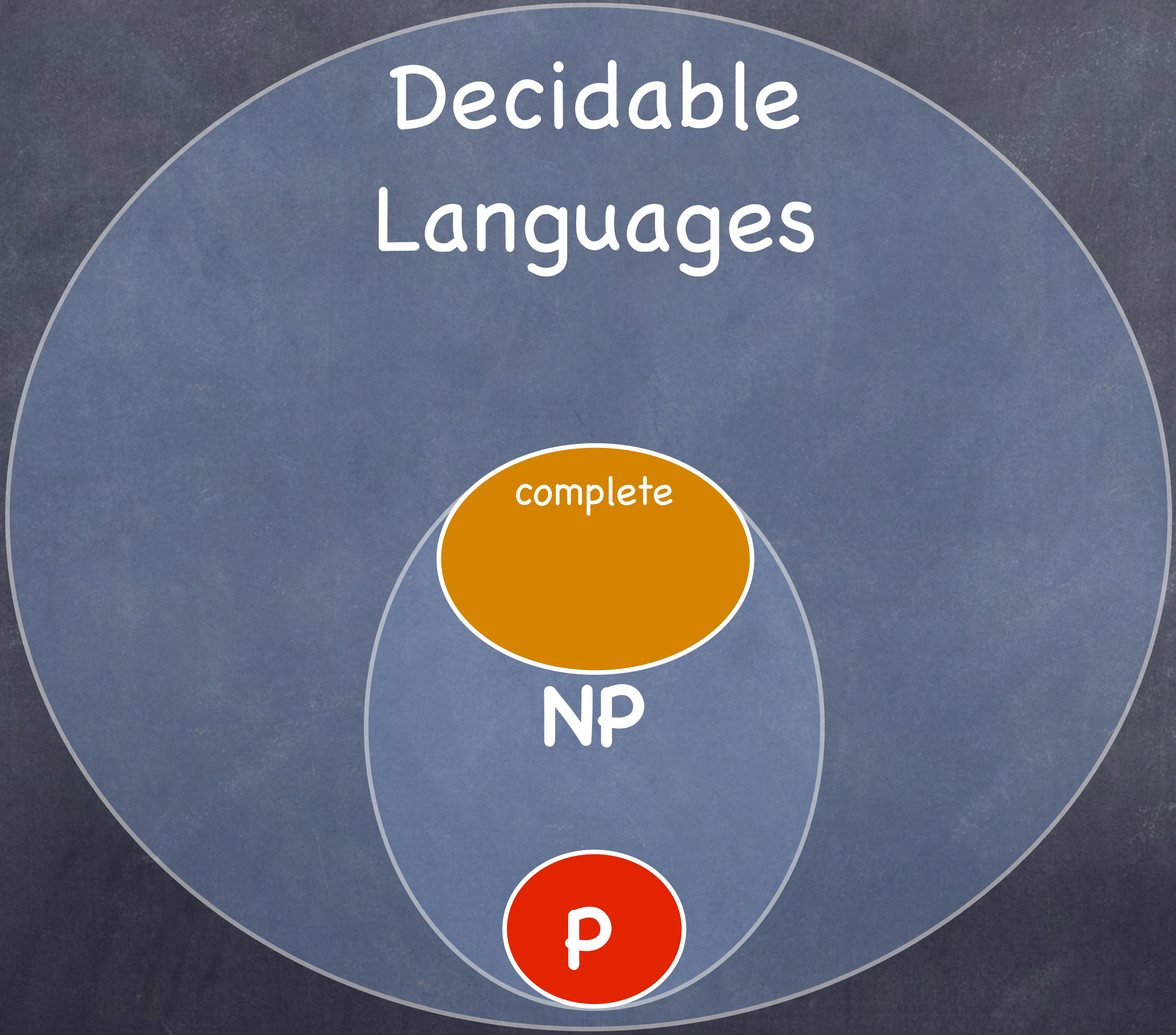
Complexity Theory



Complexity Theory



Complexity Theory



P = NP ?

Beyond **NP**-Completeness

Beyond **NP**-Completeness

- **PSpace** Completeness: problems that require a reasonable (Poly) amount of **space** to be solved but may use very long time though.

Beyond **NP**-Completeness

- **PSpace** Completeness: problems that require a reasonable (Poly) amount of **space** to be solved but may use very long time though.
- Many such problems. If any of them may be solved within reasonable (Poly) amount of time, then all of them can.

PSpace Completeness

PSpace Completeness

- Geography Game:

  Given a set of country names: Arabia, Cuba, Canada, France, Italy, Japan, Korea, Vietnam

PSpace Completeness

- Geography Game:

  Given a set of country names: Arabia, Cuba, Canada, France, Italy, Japan, Korea, Vietnam

- A two player game: One player chooses a name. The other player must choose a name that starts with the last letter of the previous name and so on. A player wins when his opponent cannot play any name.

Generalized Geography

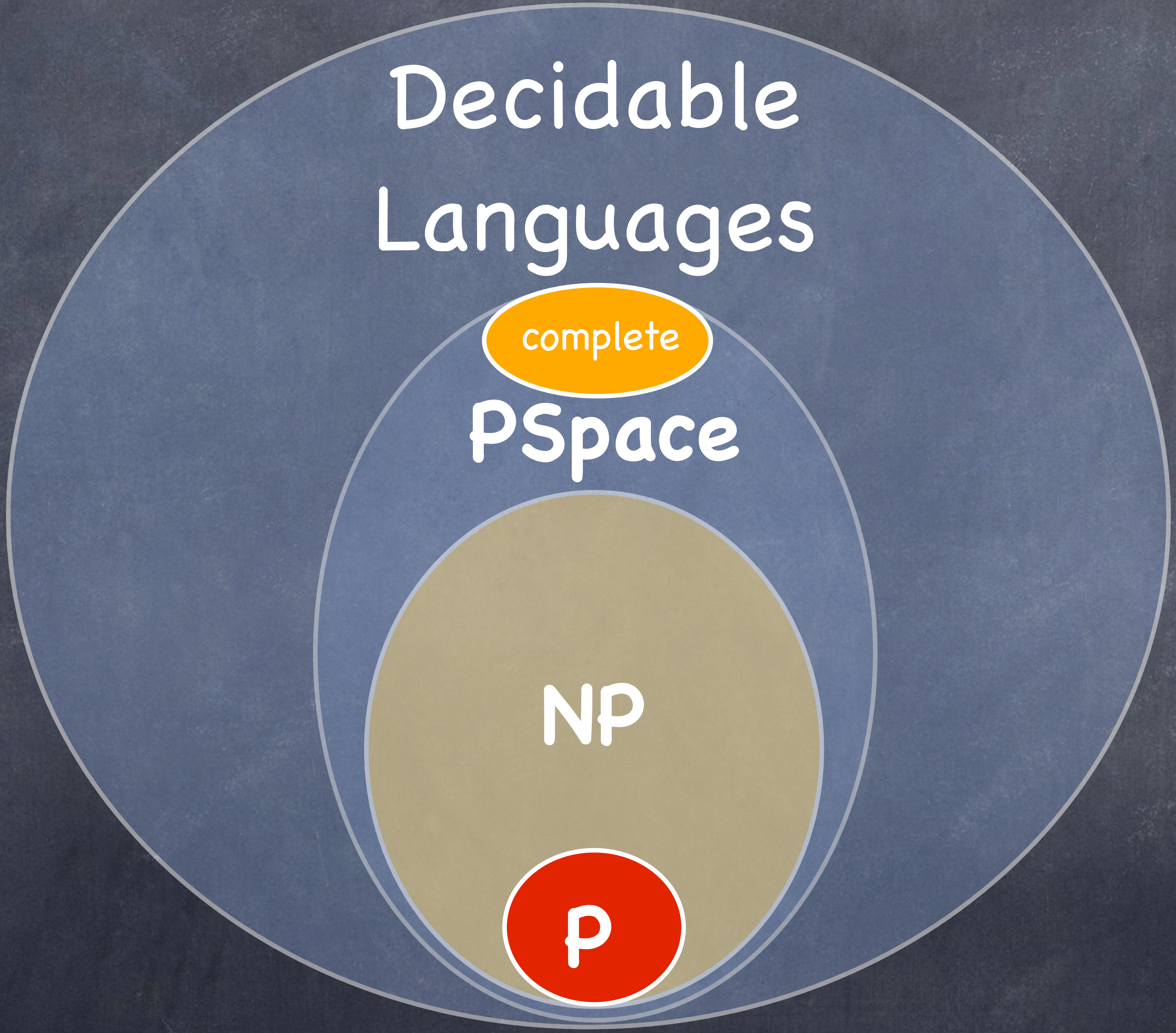
Generalized Geography

- Given an arbitrary set of names: $w_1, \ldots, w_n$.

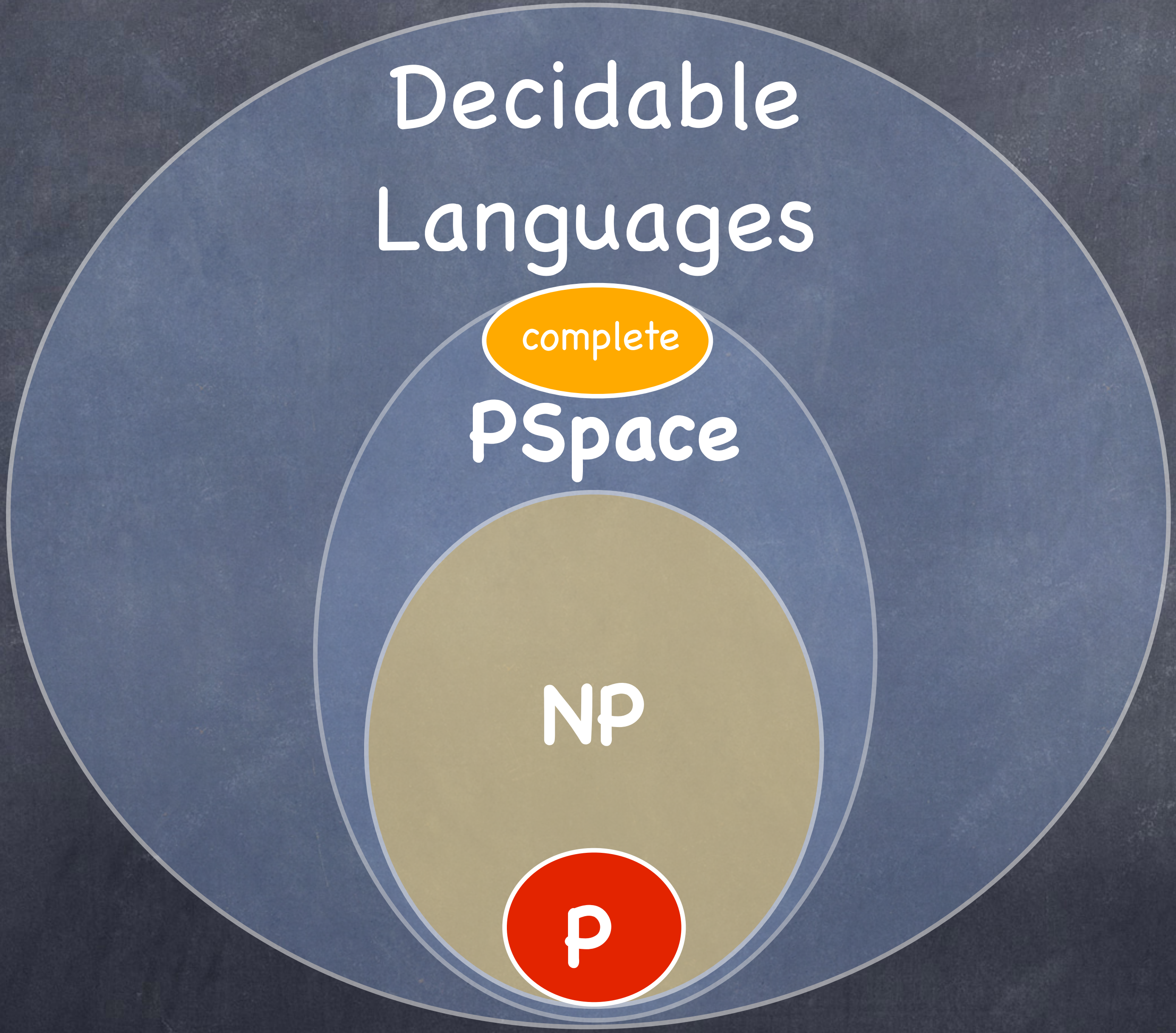
Generalized Geography

- Given an arbitrary set of names: w_1, ..., w_n.
- Is there a winning strategy for the first player to the previous game ?

Complexity Theory



Complexity Theory



NP = PSpace ?

Theoretical Computer Science

Theoretical Computer Science

- **Challenges of TCS:**

Theoretical Computer Science

- **Challenges of TCS:**

- **FIND** efficient solutions to many problems. (Algorithms and Data Structures)

Theoretical Computer Science

- **Challenges of TCS:**
- **FIND** efficient solutions to many problems. (Algorithms and Data Structures)
- **PROVE** that certain problems are **NOT** computable within a certain time or space.

Theoretical Computer Science

- **Challenges of TCS:**
- **FIND** efficient solutions to many problems.
  (Algorithms and Data Structures)
- **PROVE** that certain problems are **NOT** computable within a certain time or space.
- Consider new models of computation.
  (Such as a Quantum Computer)

COMP 330 Fall 2017: Lectures Schedule

1-2. Introduction

1.5. Some basic mathematics

2-3. Deterministic finite automata

+Closure properties,

3-4. Nondeterministic finite automata

5. Minimization+ Myhill-Nerode theorem

6. Determinization+Kleene's theorem

7. Regular Expressions+GNFA

8. Regular Expressions and Languages

9-10. The pumping lemma

11. Duality

12. Labelled transition systems

13. MIDTERM

14. Context-free languages

15. Pushdown automata

16. Parsing

17. The pumping lemma for CFLs

18. Introduction to computability

19. Models of computation

Basic computability theory

20. Reducibility, undecidability and Rice's theorem

21. Undecidable problems about CFGs

22. Post Correspondence Problem

23. Validity of FOL is RE / Gödel's and Tarski's thms

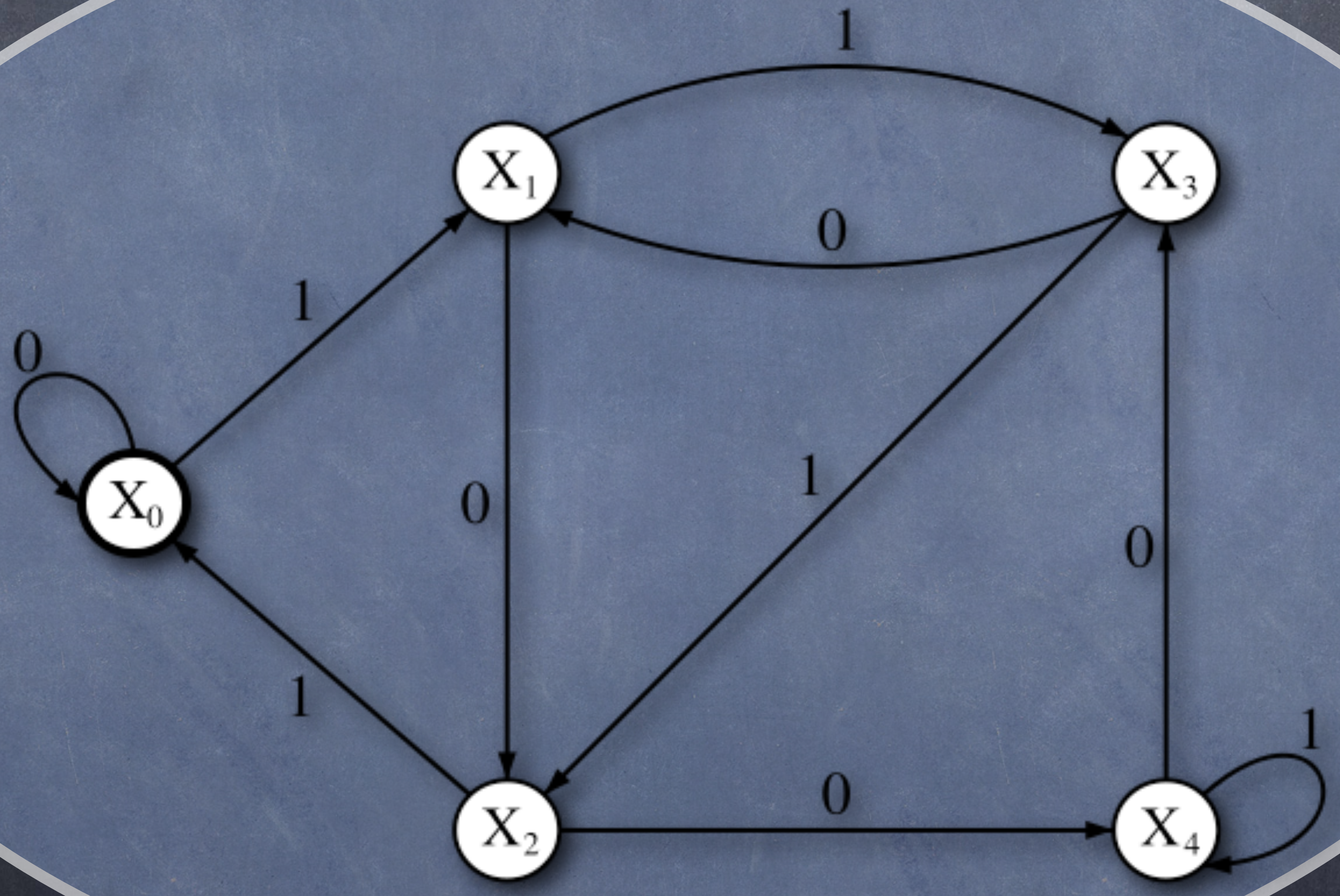
24. Universality / The recursion theorem

25. Degrees of undecidability

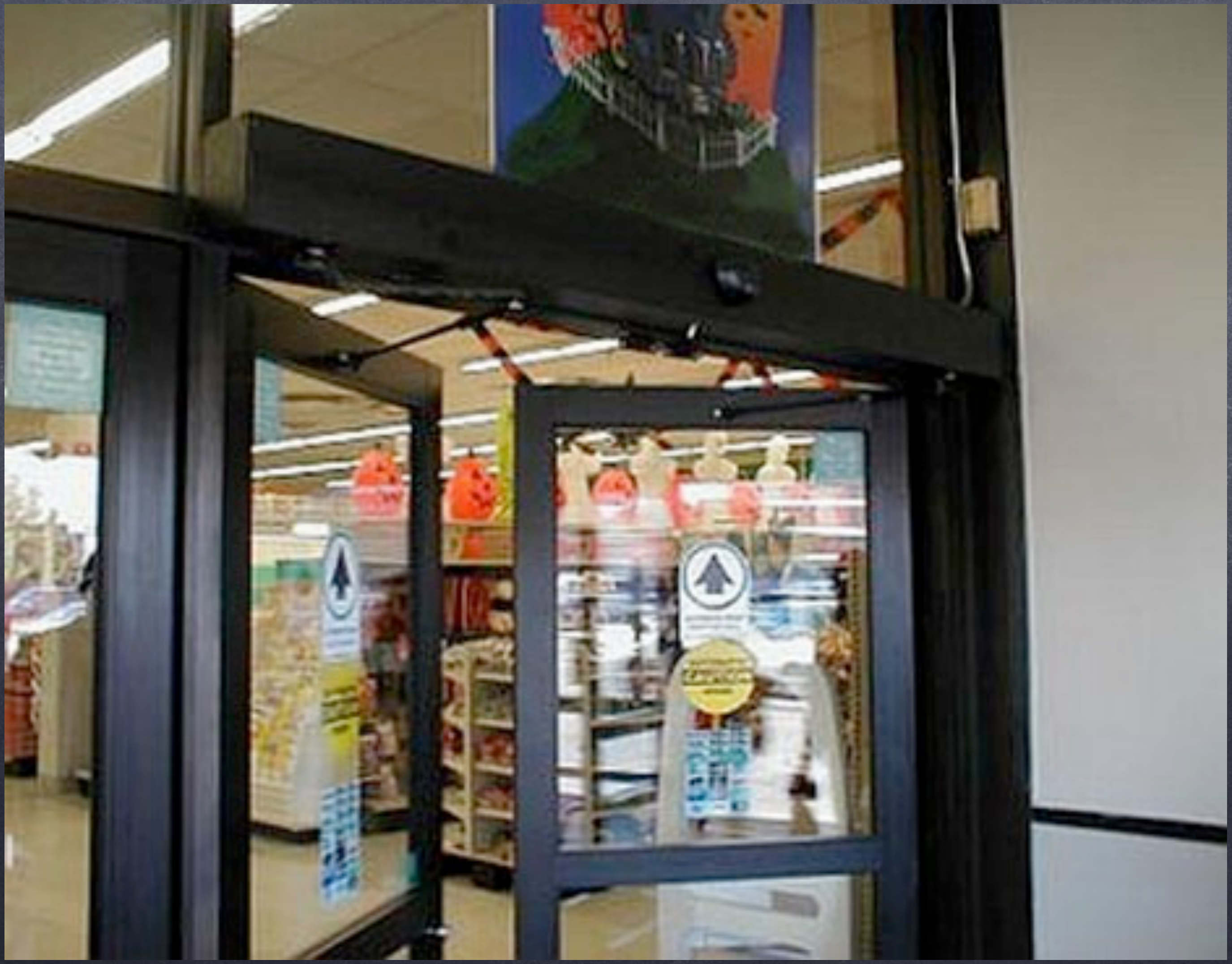
26. Introduction to complexity

Deterministic Finite Automata, and Regular expressions

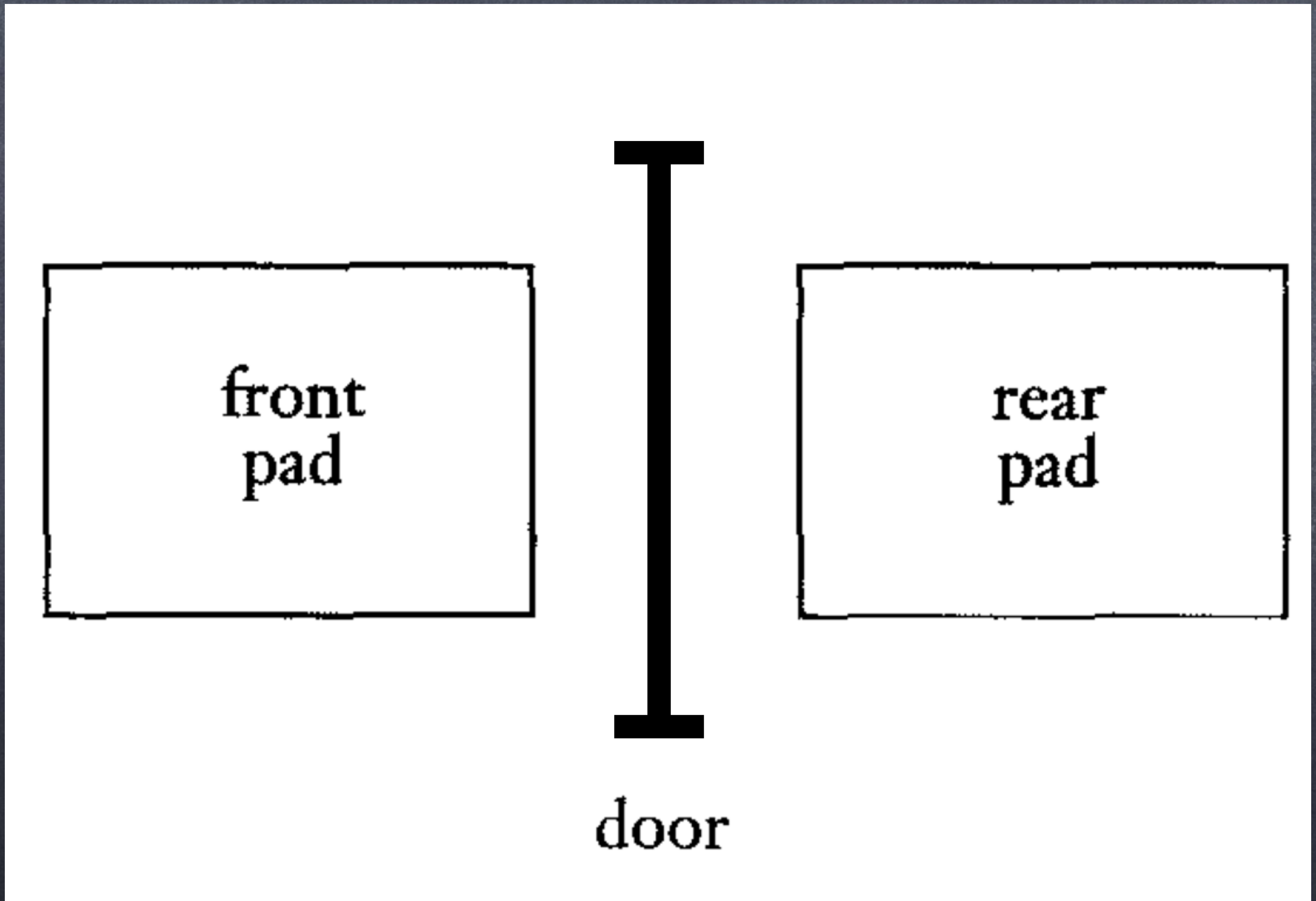
Deterministic Finite Automata, and Regular expressions



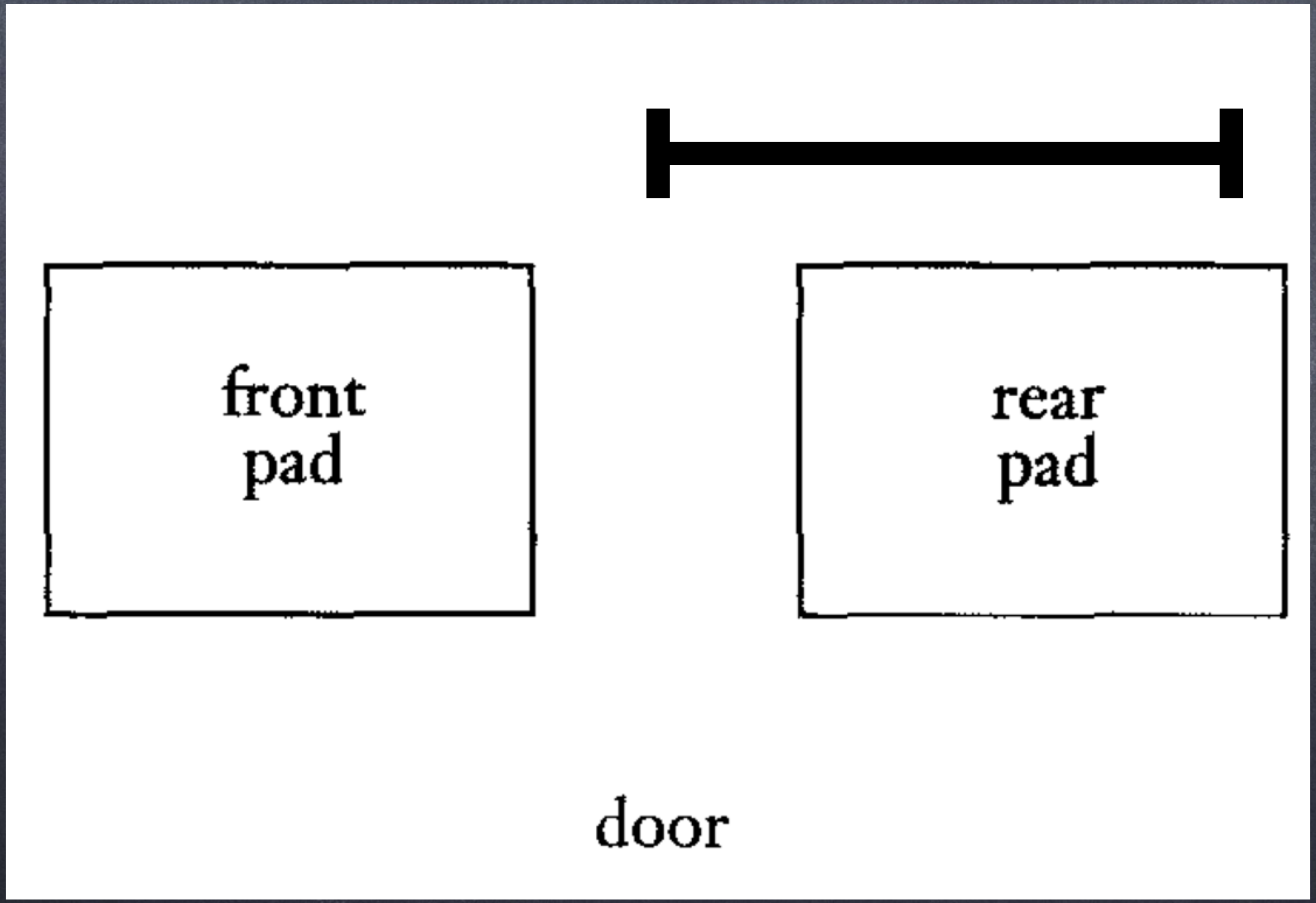
Swing doors



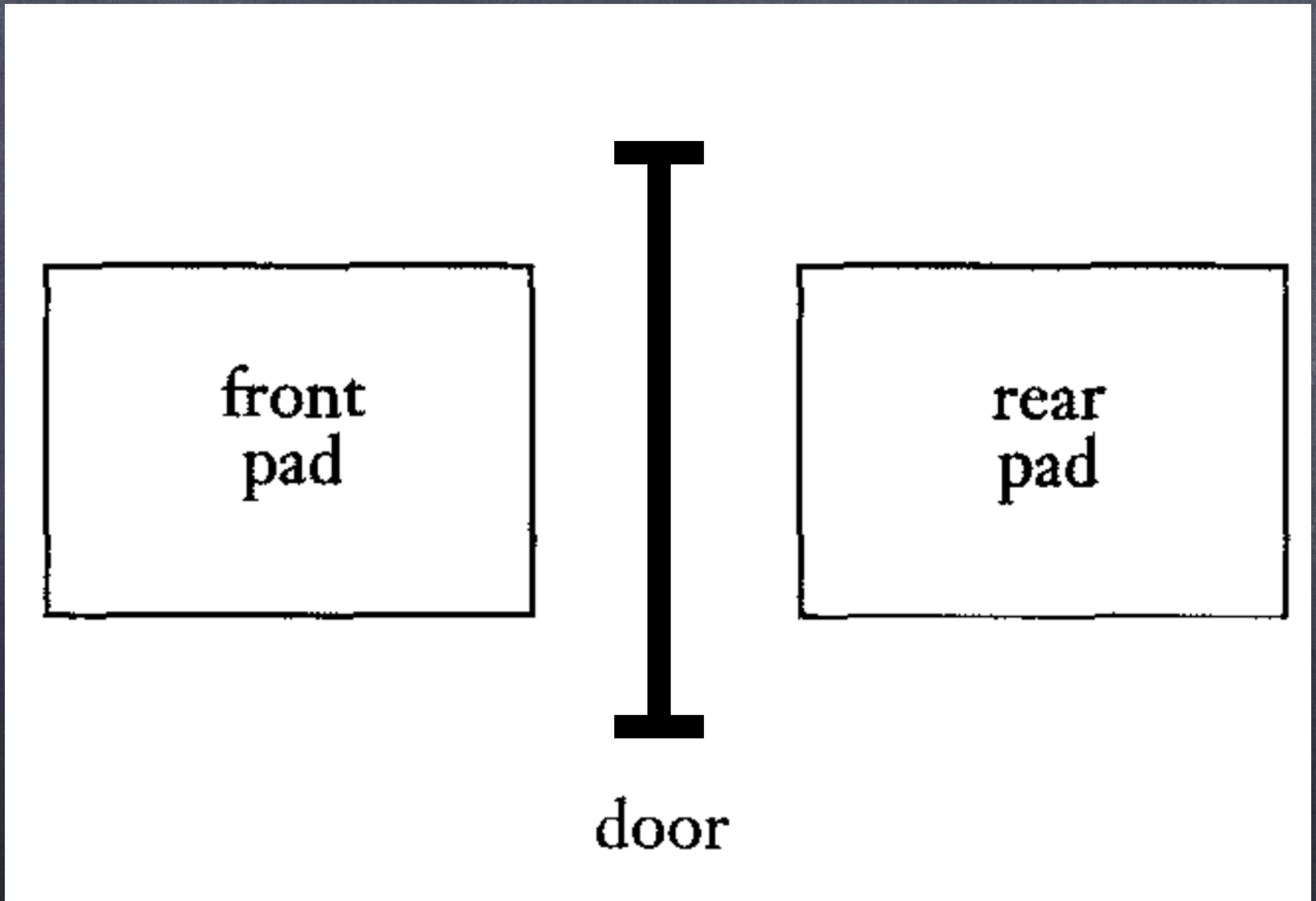
Swing doors



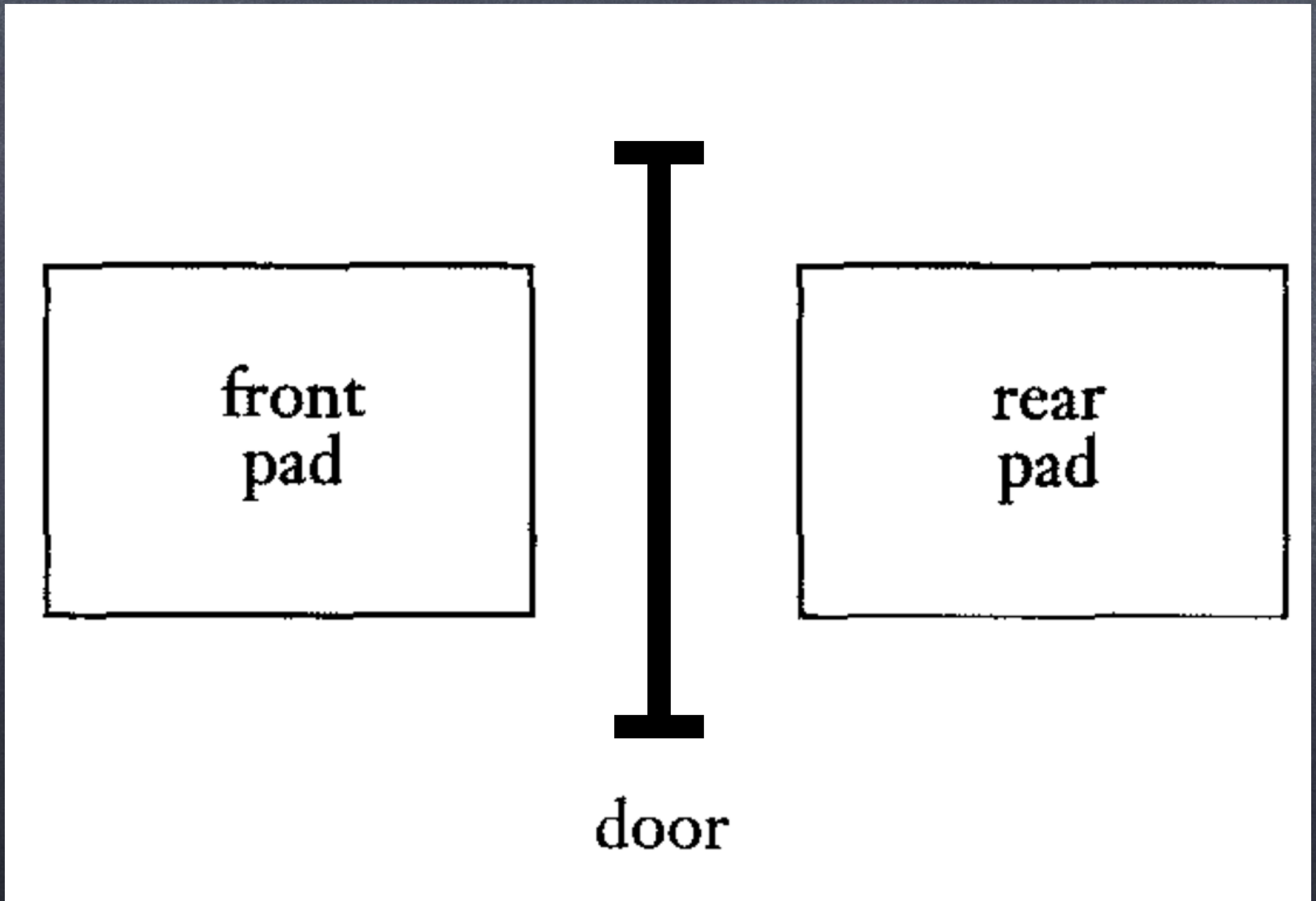
Swing doors



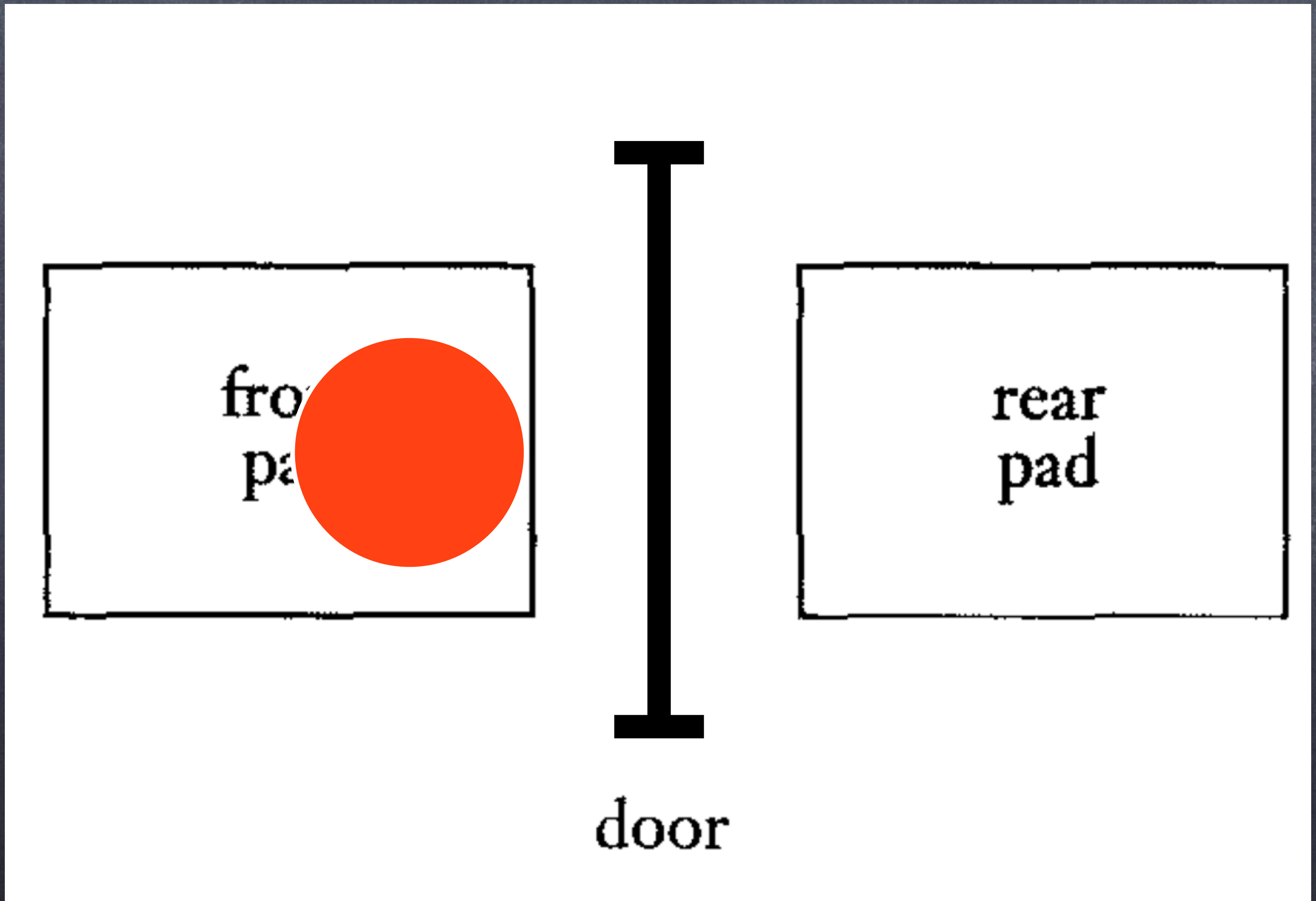
Swing doors



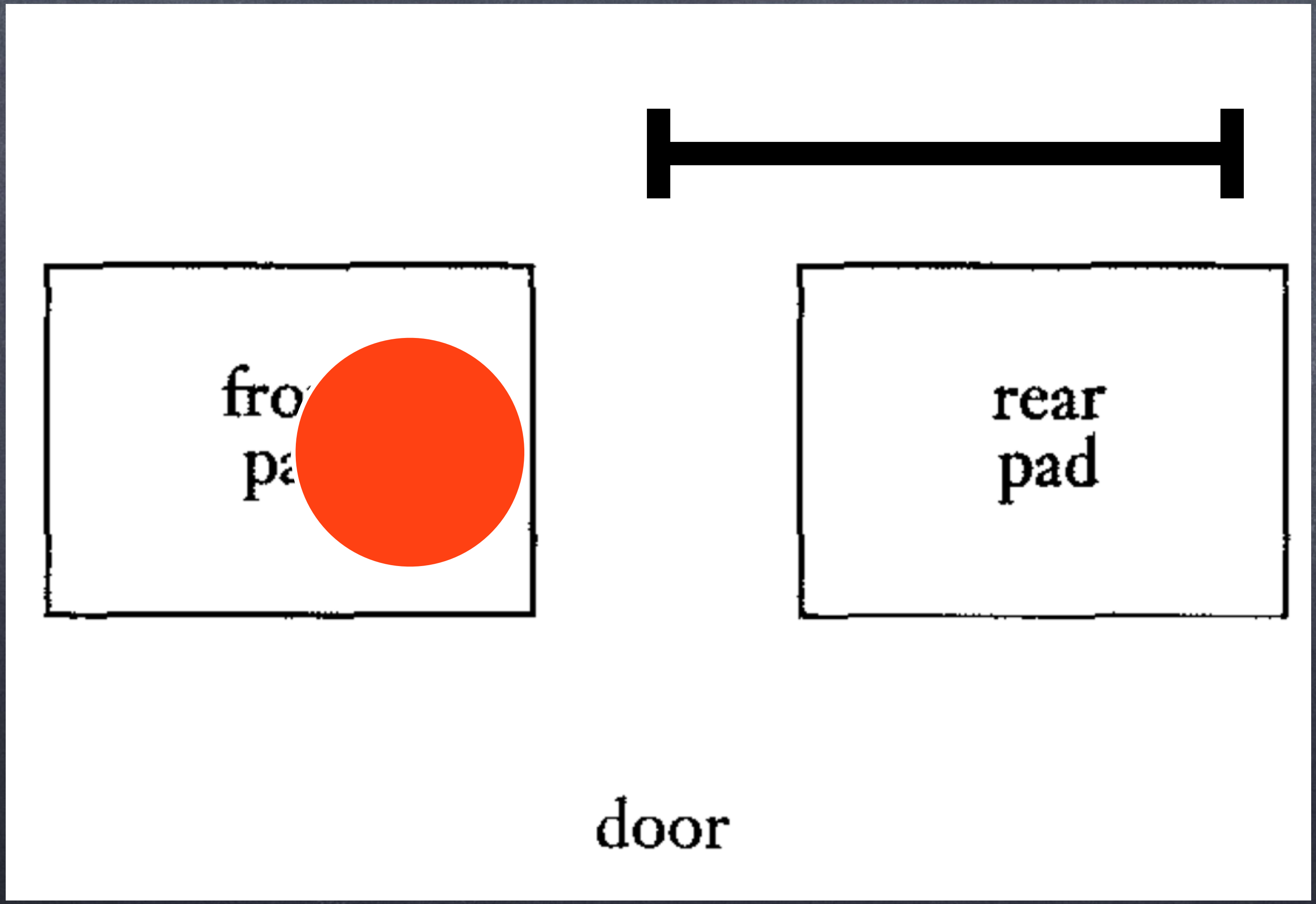
Swing doors



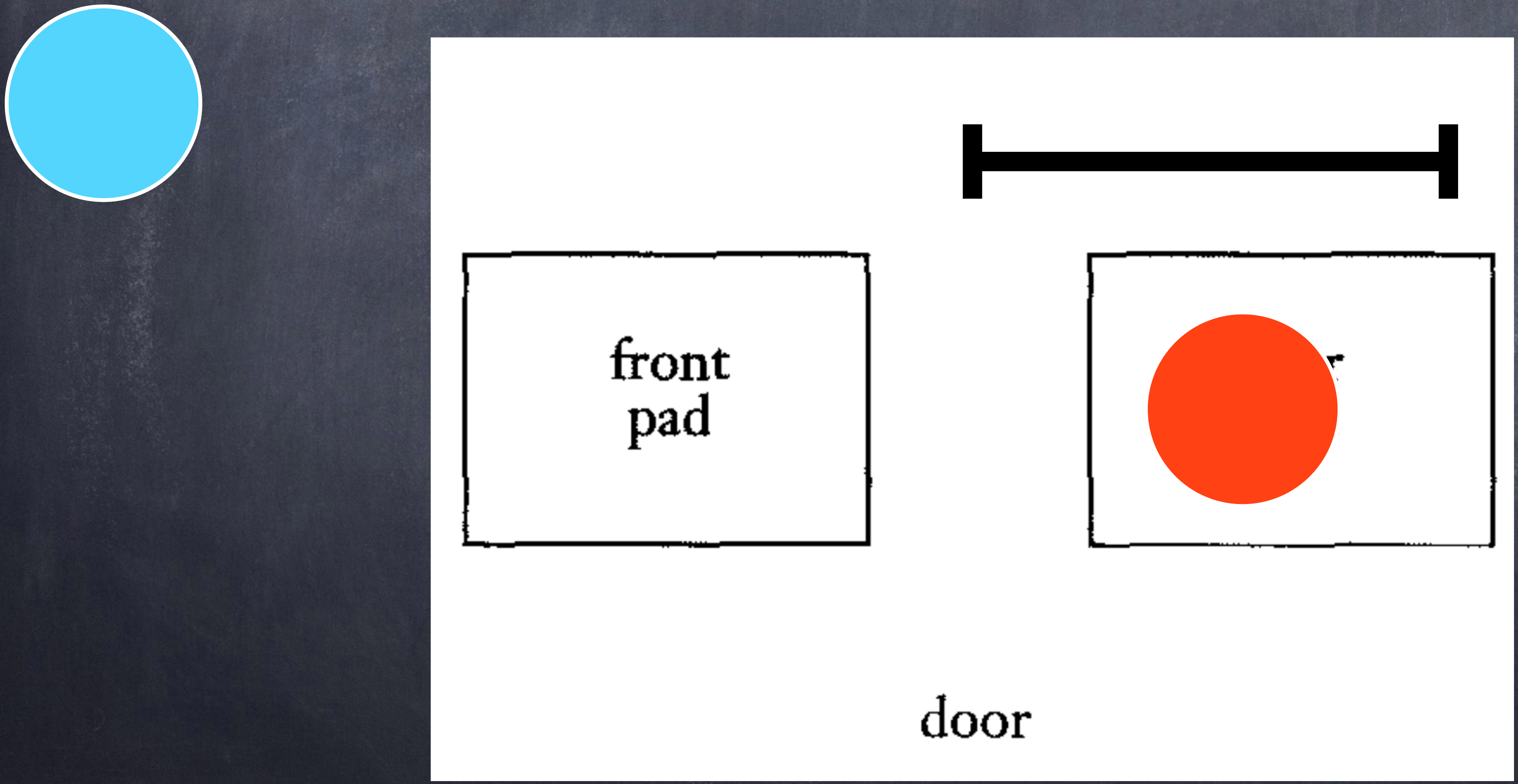
Swing doors



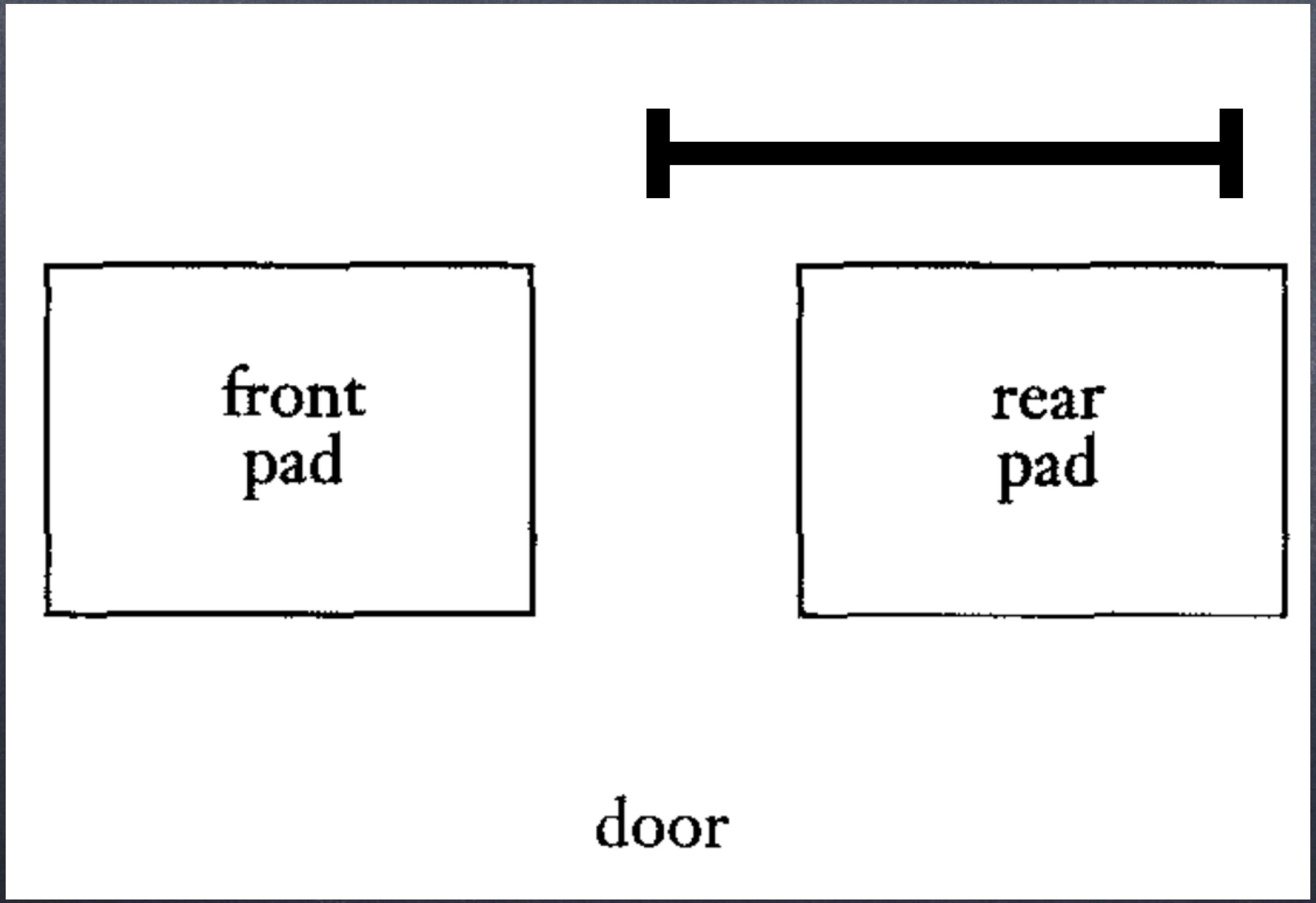
Swing doors



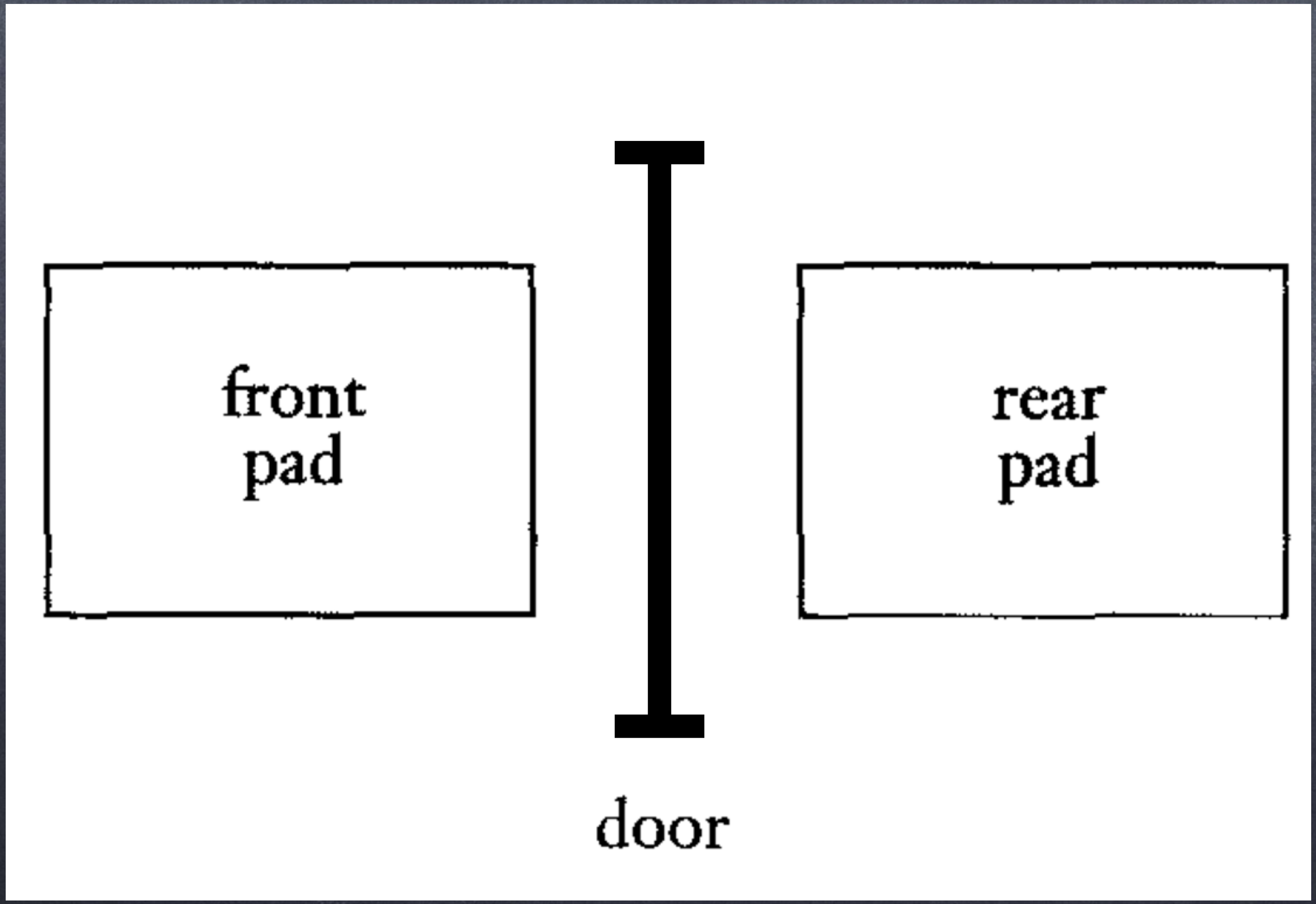
Swing doors



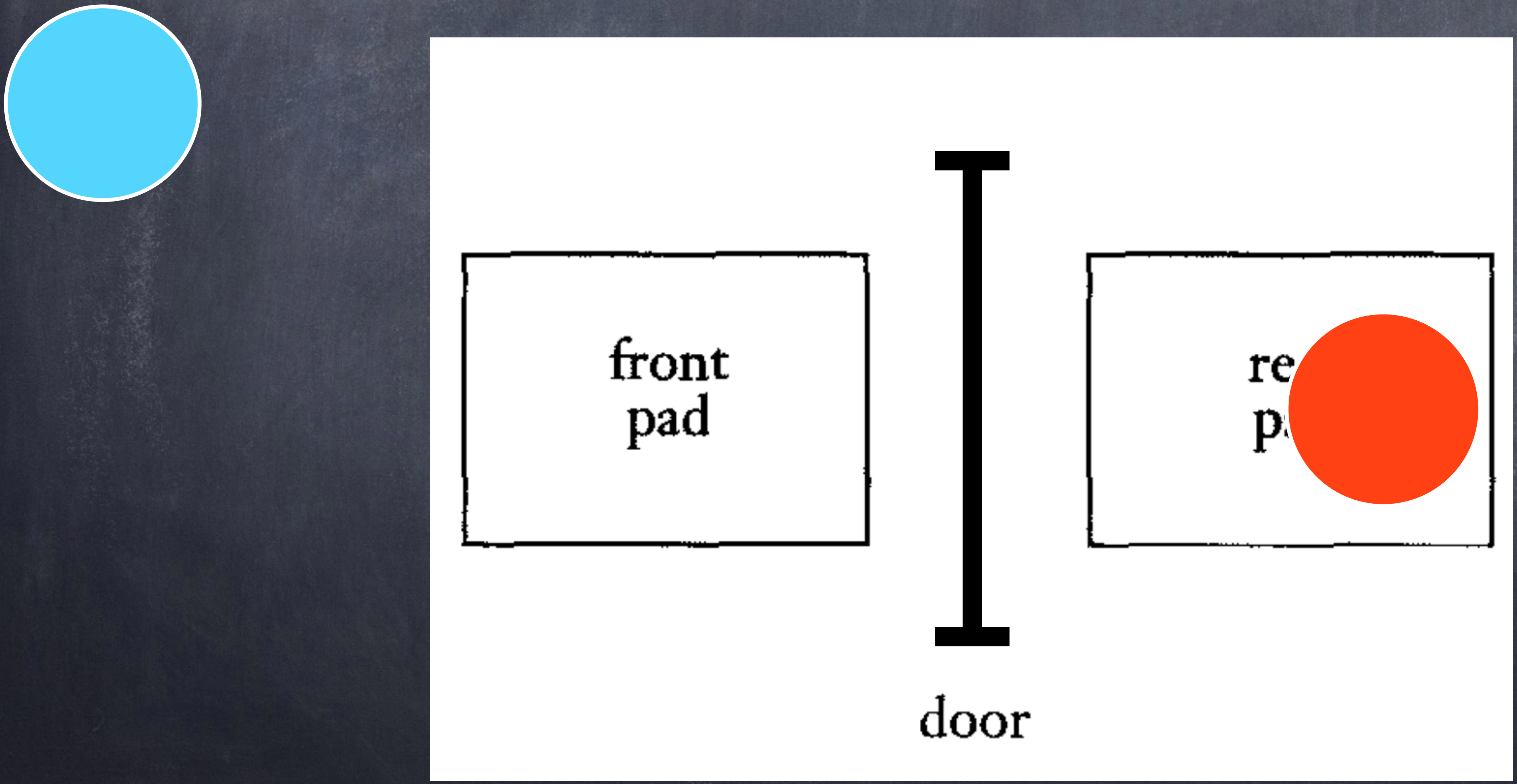
Swing doors



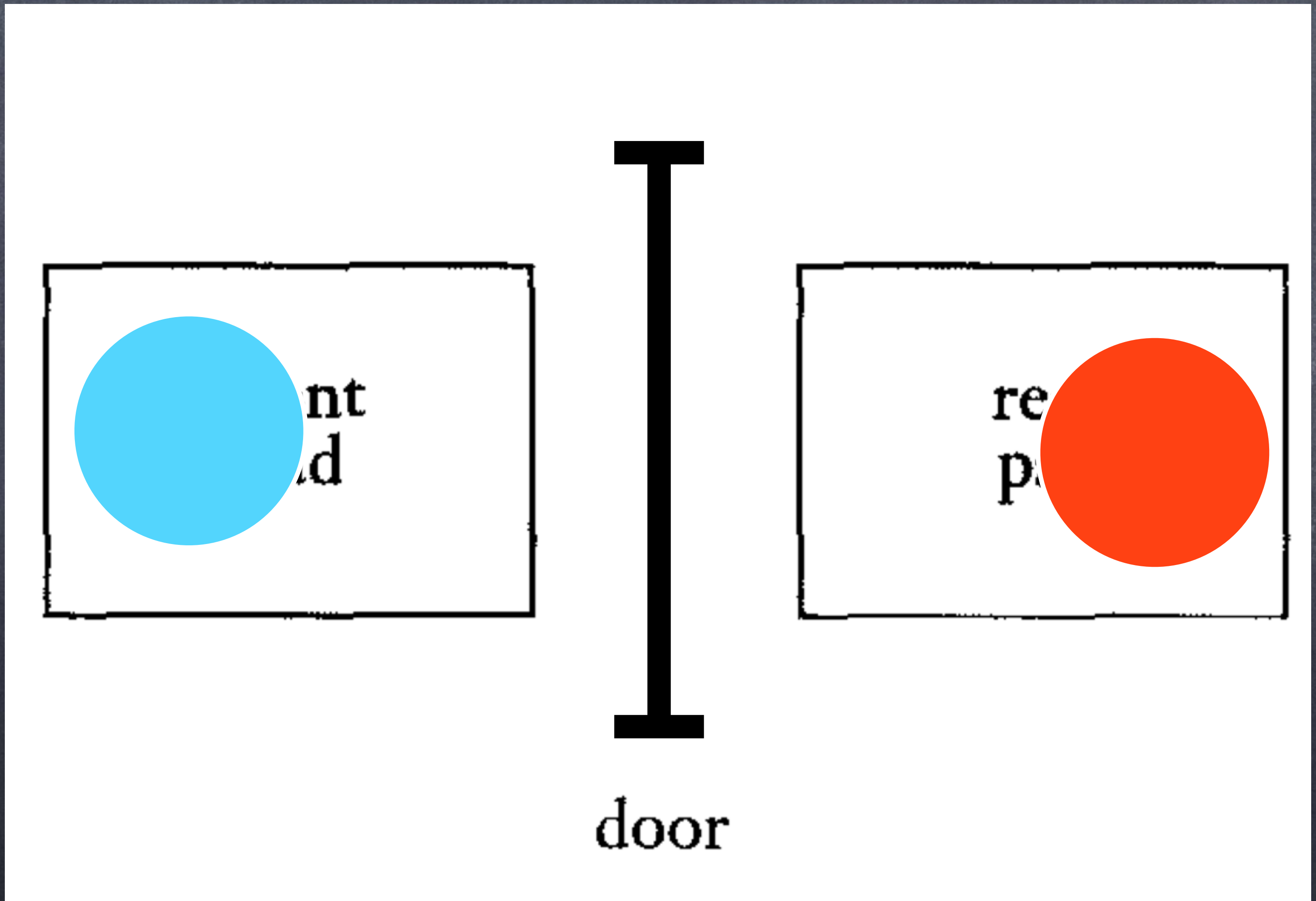
Swing doors



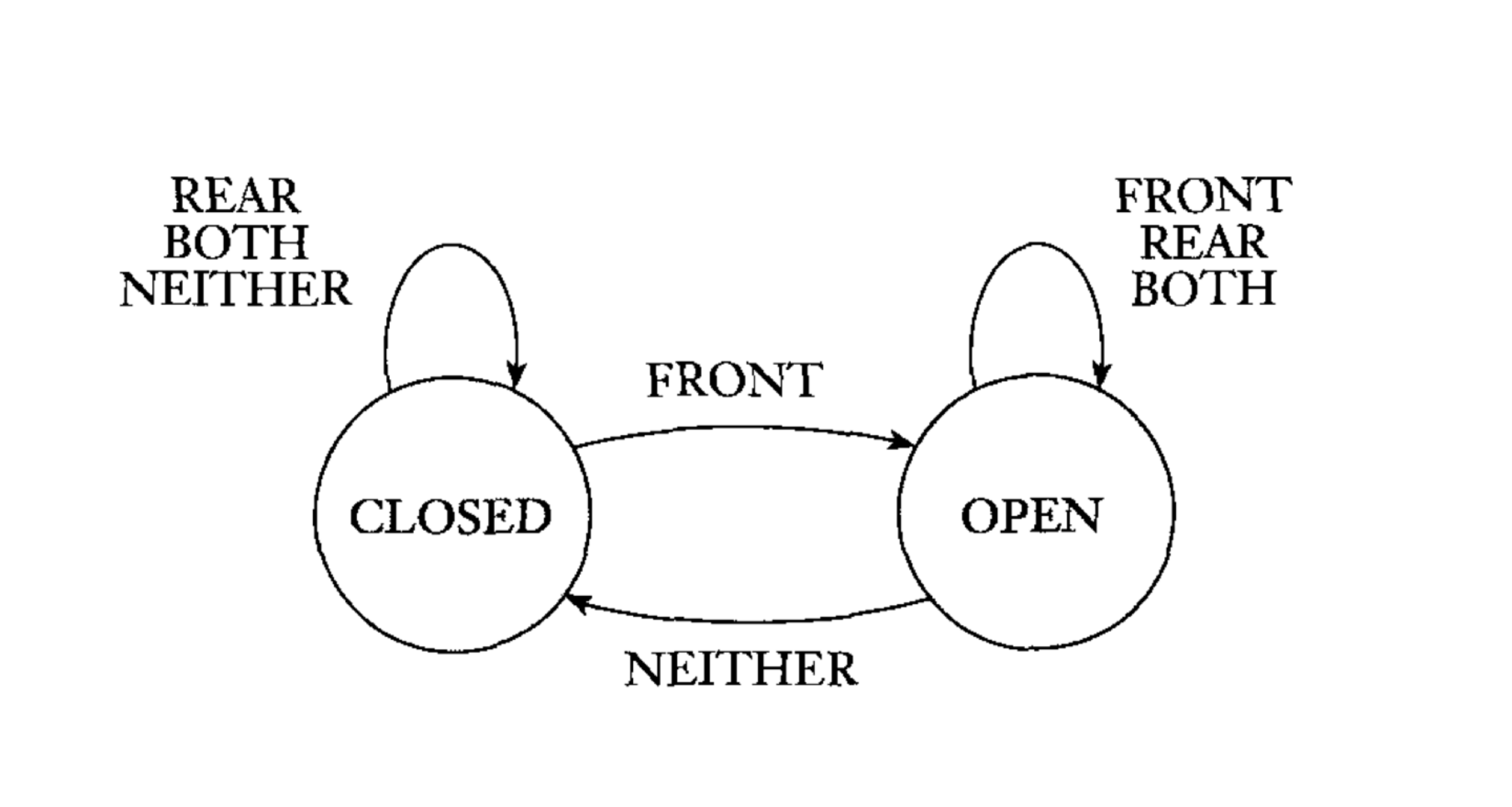
Swing doors



Swing doors

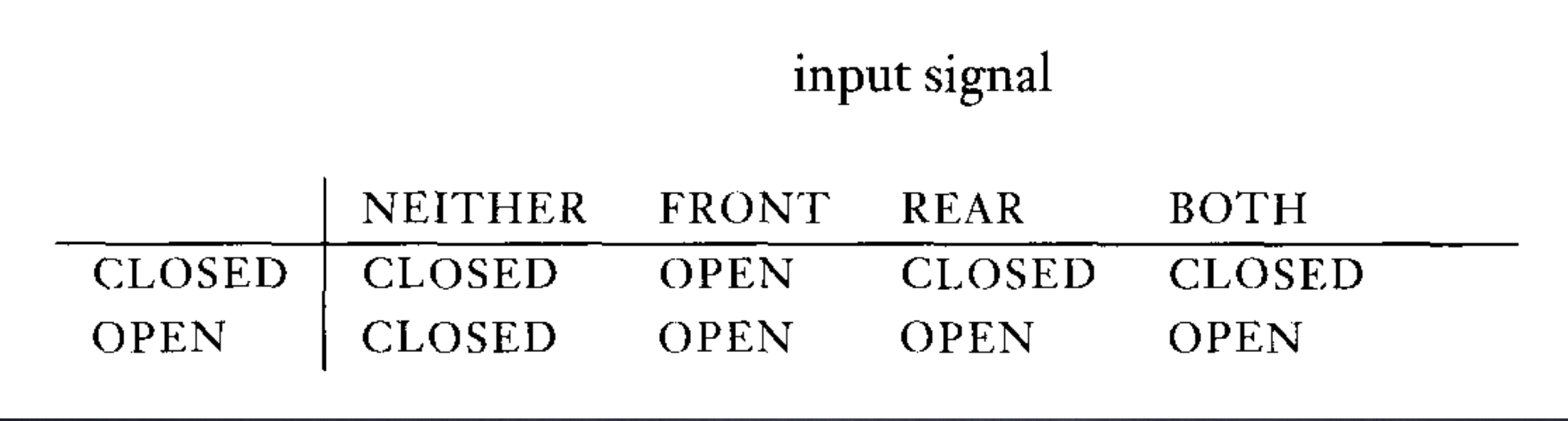


Swing doors



REAR
BOTH
NEITHER
FRONT
REAR
BOTH
FRONT
CLOSED
OPEN
NEITHER

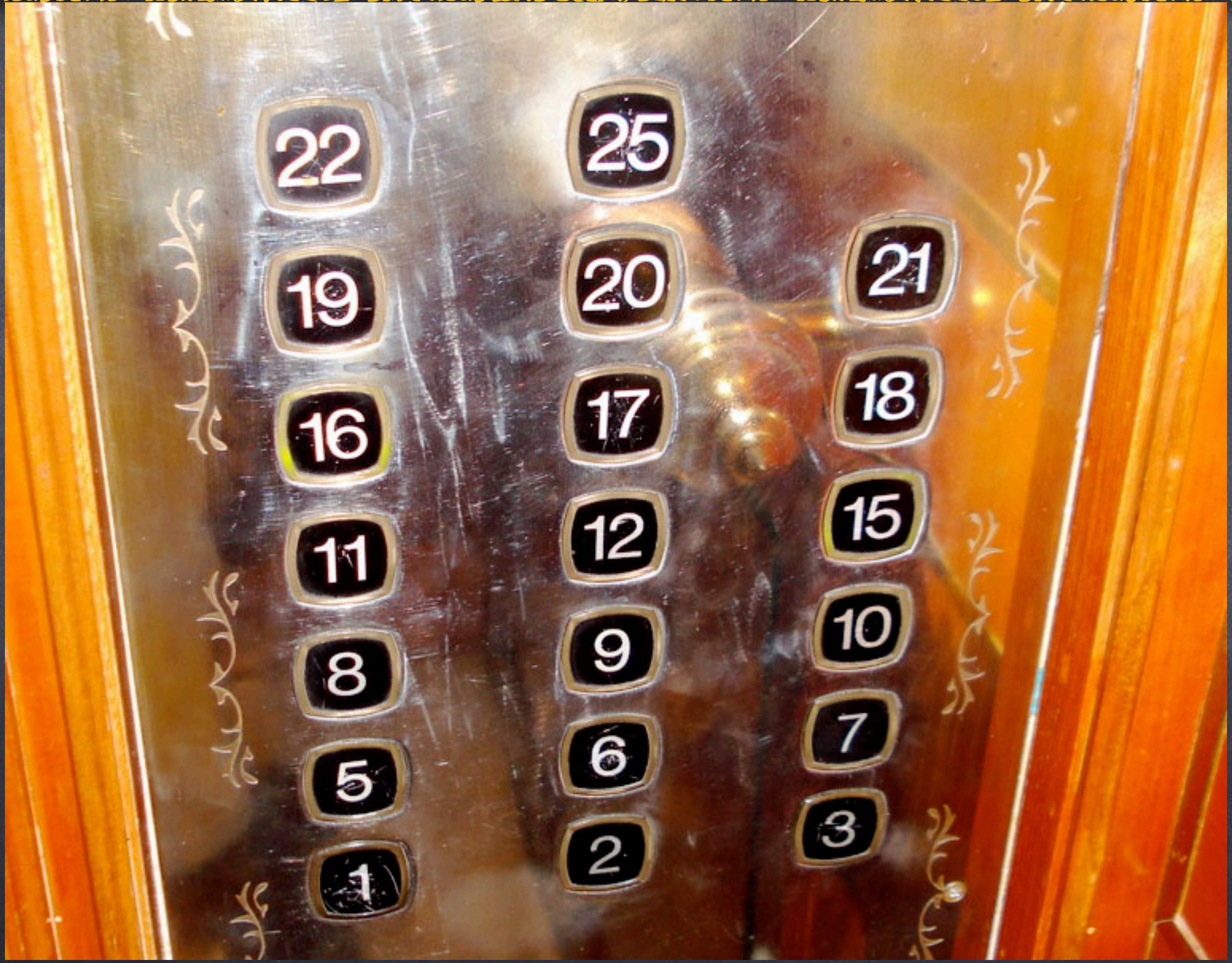
Swing doors



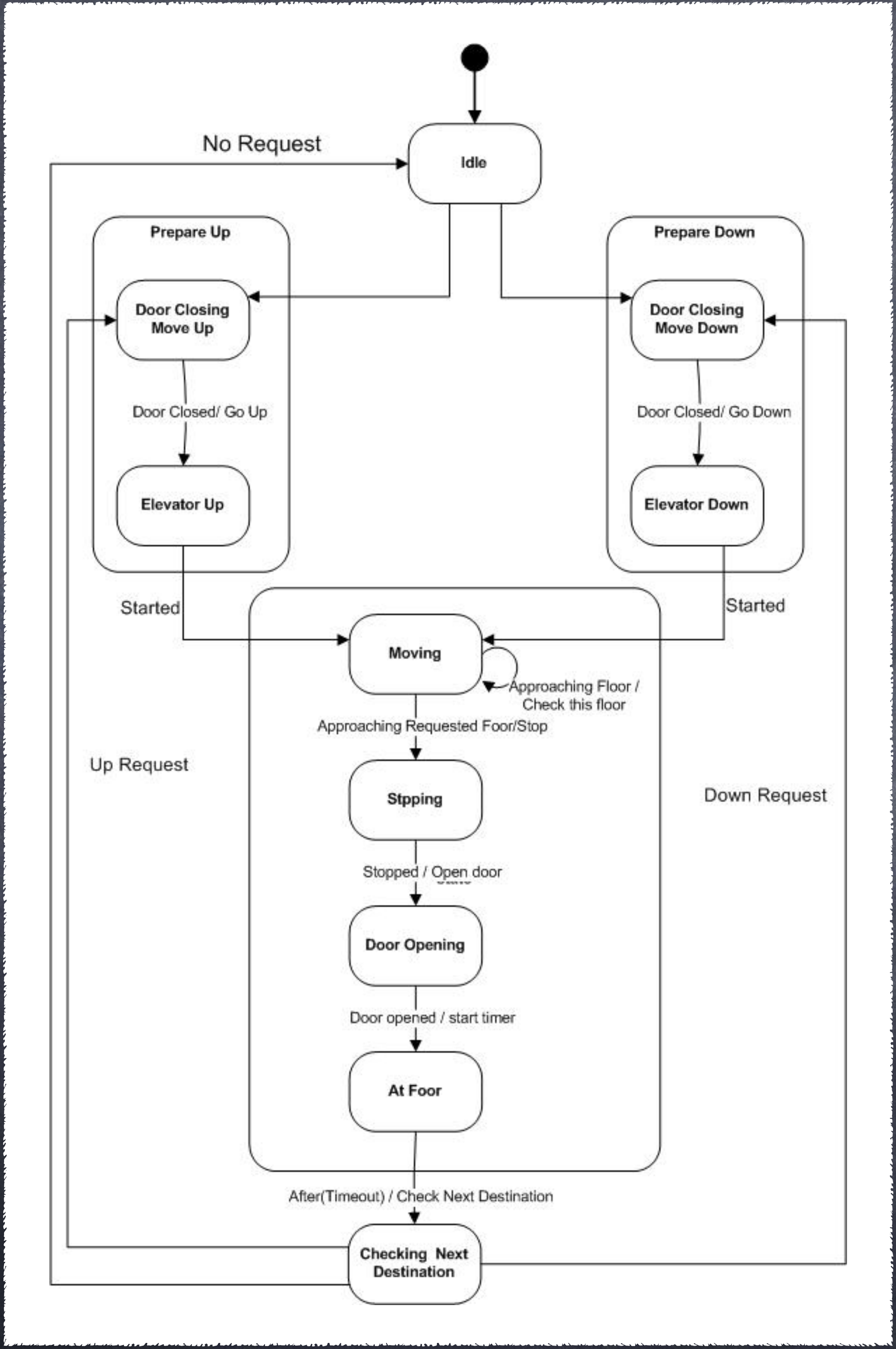
input signal

	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Elevators



Elevators



Idle

No Request

Prepare Up

Door Closing Move Up

Door Closed/ Go Up

Elevator Up

Prepare Down

Door Closing Move Down

Door Closed/ Go Down

Elevator Down

Started

Started

Moving

Approaching Floor / Check this floor

Approaching Requested Foor/Stop

Stpping

Stopped / Open door

Door Opening

Door opened / start timer

At Foor

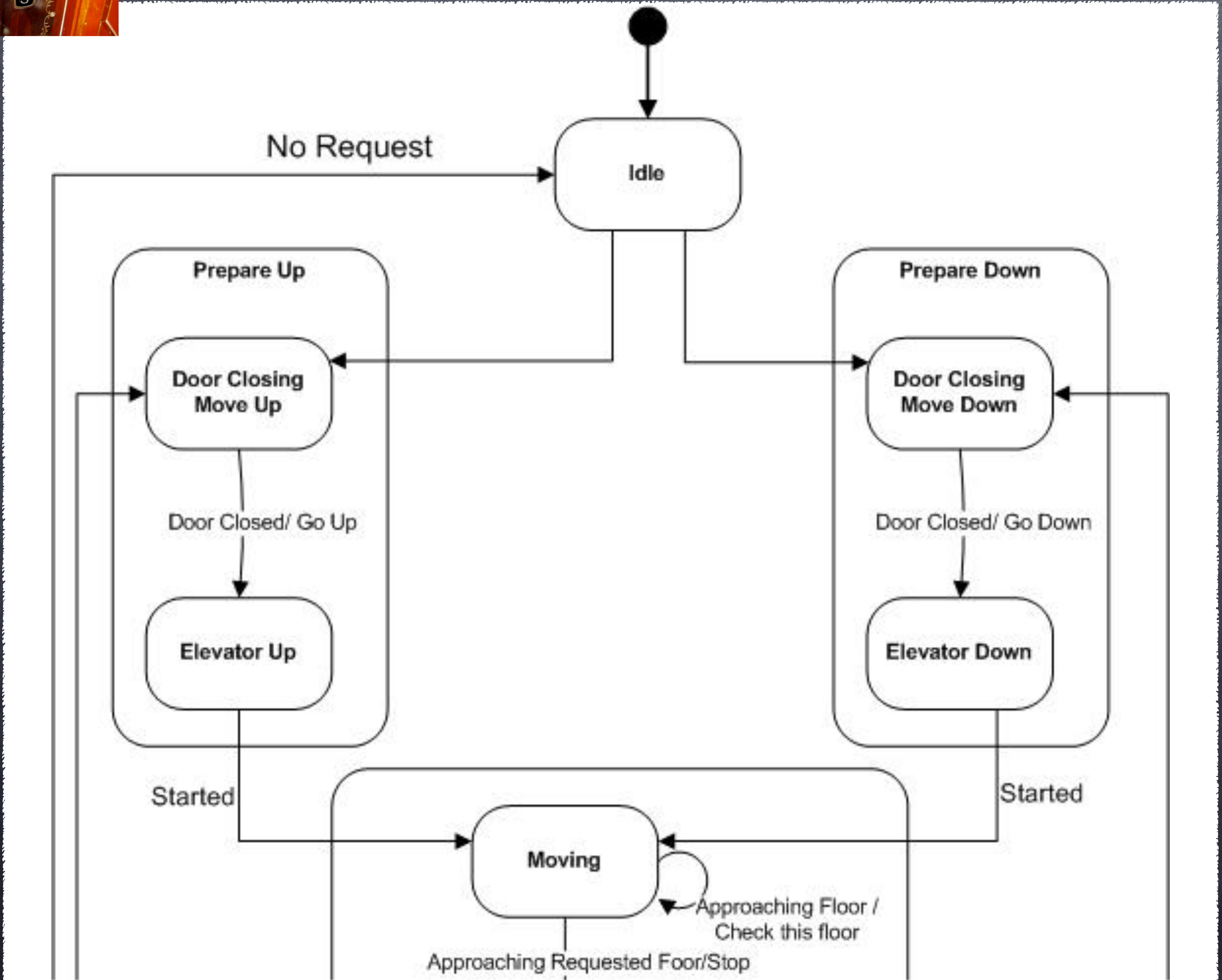
After(Timeout) / Check Next Destination

Checking Next Destination

Up Request

Down Request

Elevators



Idle

No Request

Prepare Up

Door Closing
Move Up

Door Closed/ Go Up

Elevator Up

Prepare Down

Door Closing
Move Down

Door Closed/ Go Down

Elevator Down

Started

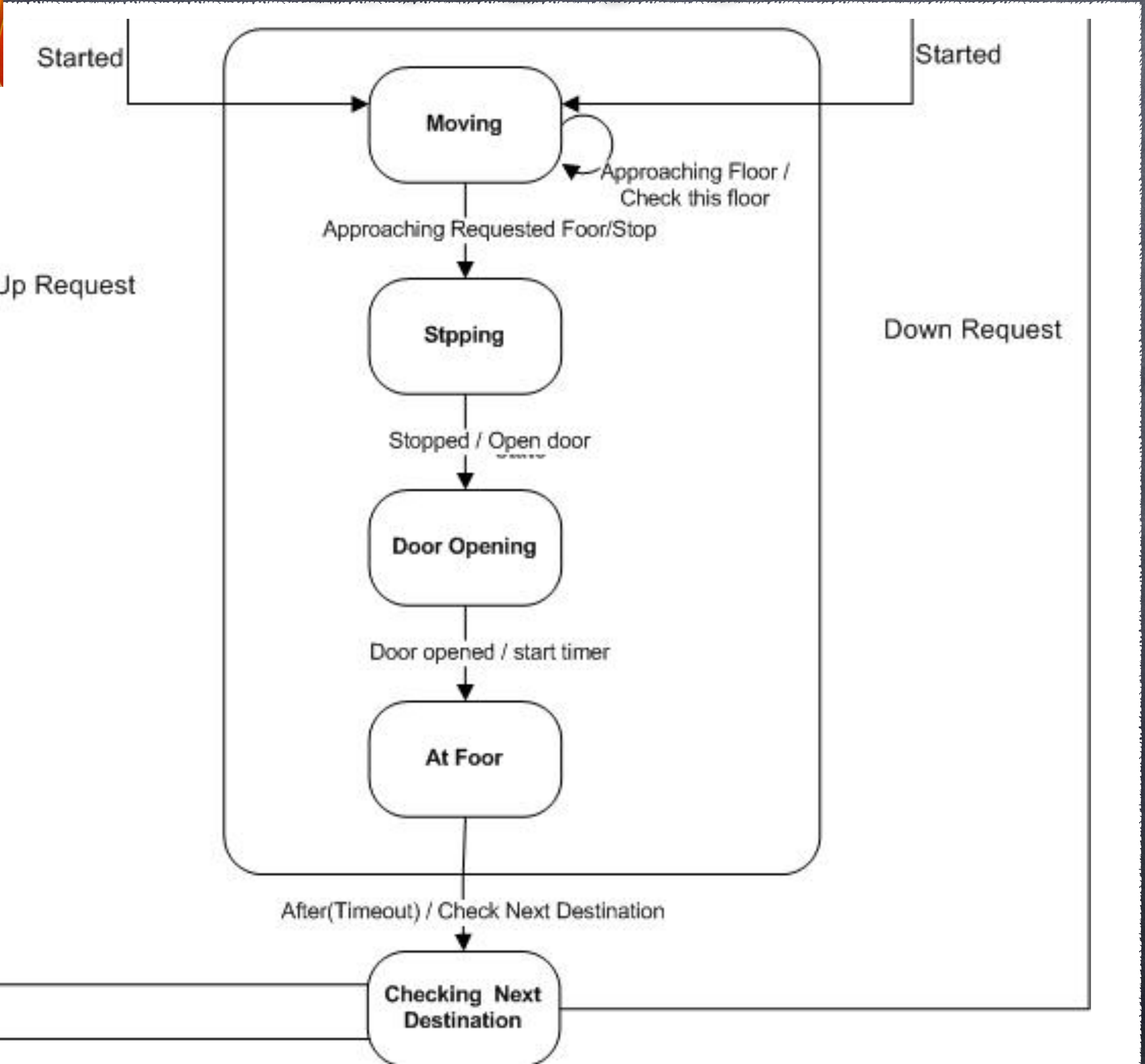
Started

Moving

Approaching Floor /
Check this floor

Approaching Requested Foor/Stop

Elevators


Started
Started
Moving
Approaching Floor / Check this floor
Approaching Requested Foor/Stop
Up Request
Stpping
Down Request
Stopped / Open door
Door Opening
Door opened / start timer
At Foor
After(Timeout) / Check Next Destination
Checking Next Destination

COMP-330
Theory of Computation

Fall 2017 -- Prof. Claude Crépeau

Lecture 2 : Regular Expressions & DFAs