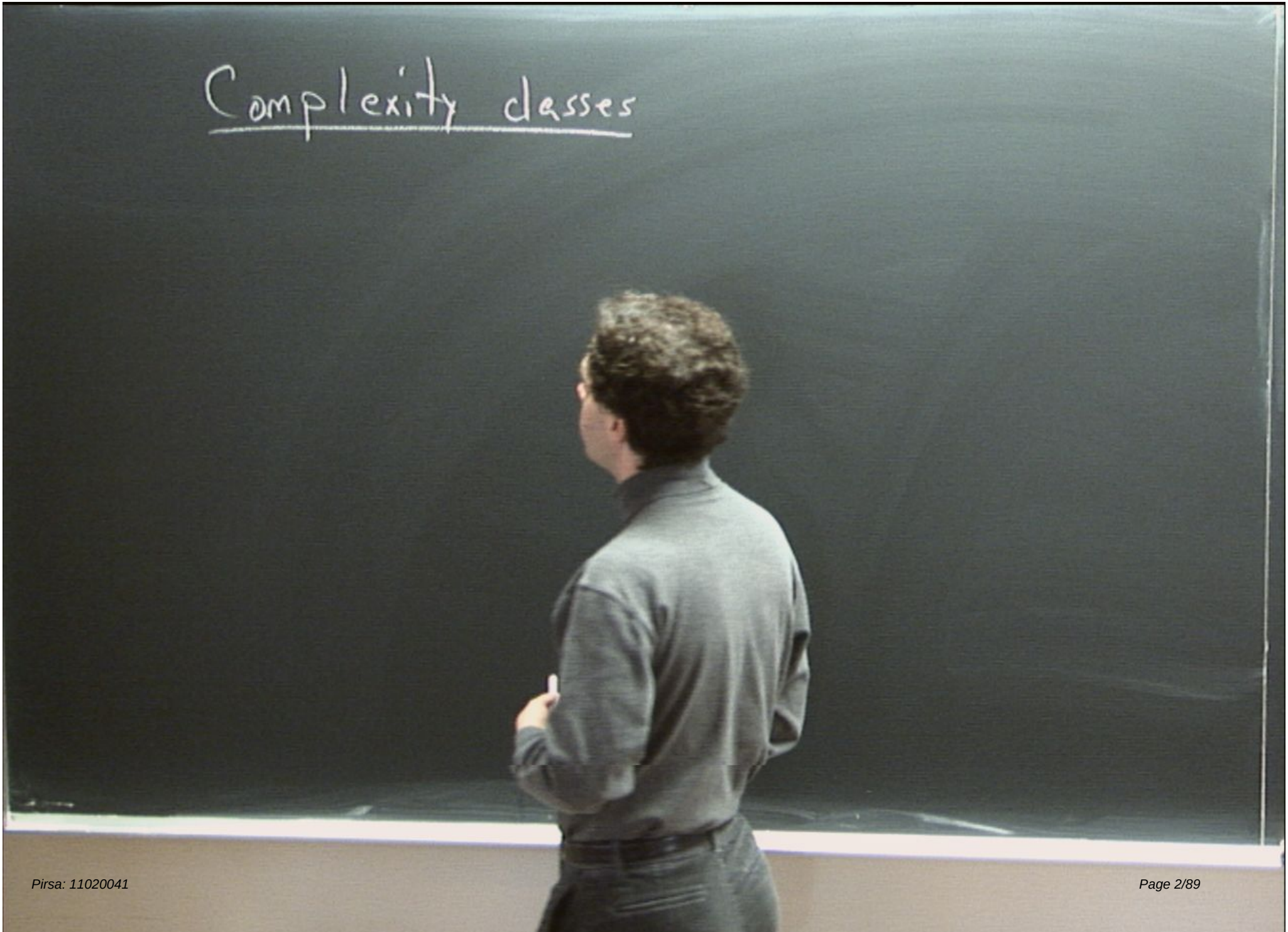
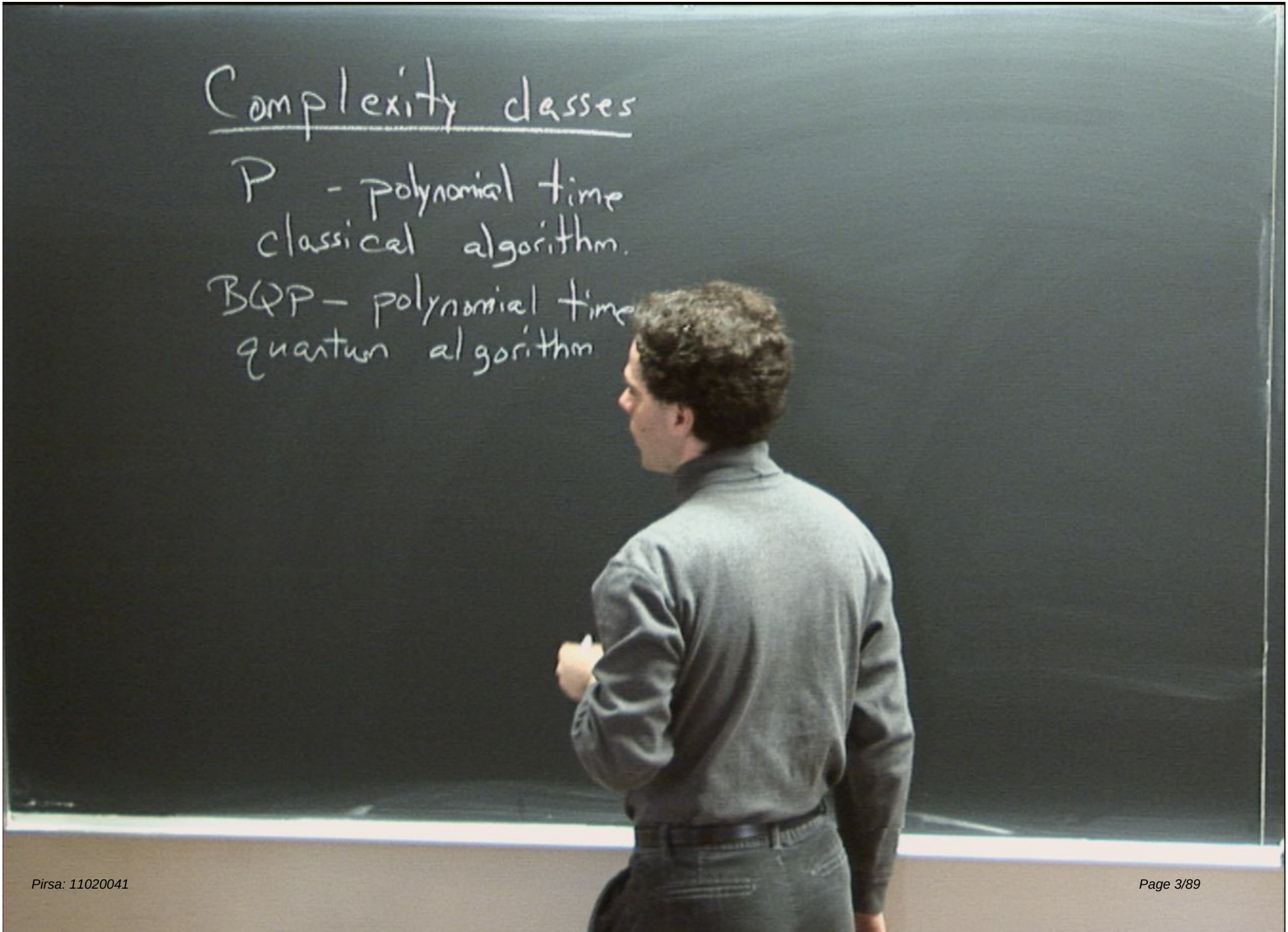
Title: Quantum Information Review - Lecture 6

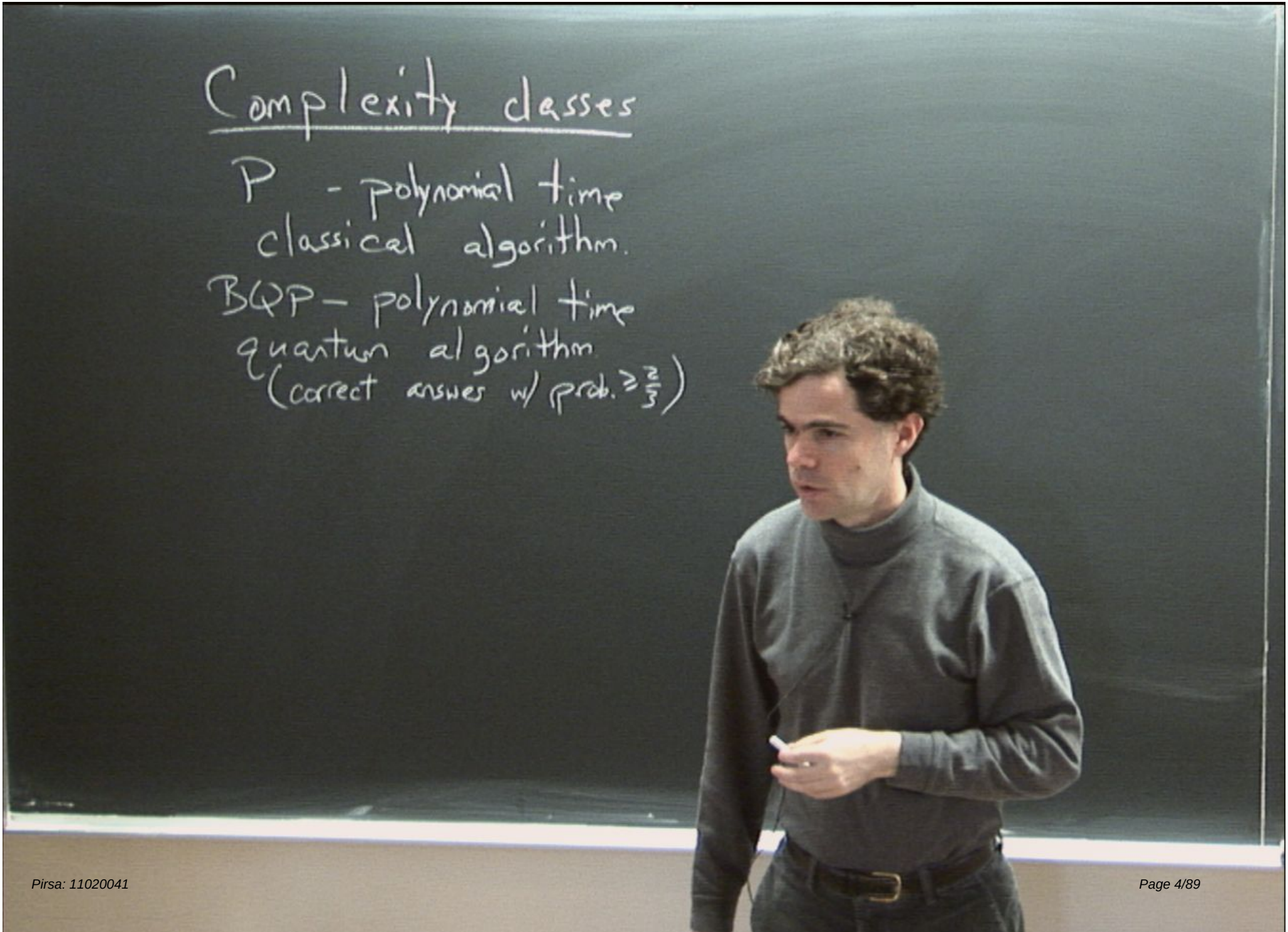
Date: Feb 22, 2011 09:00 AM

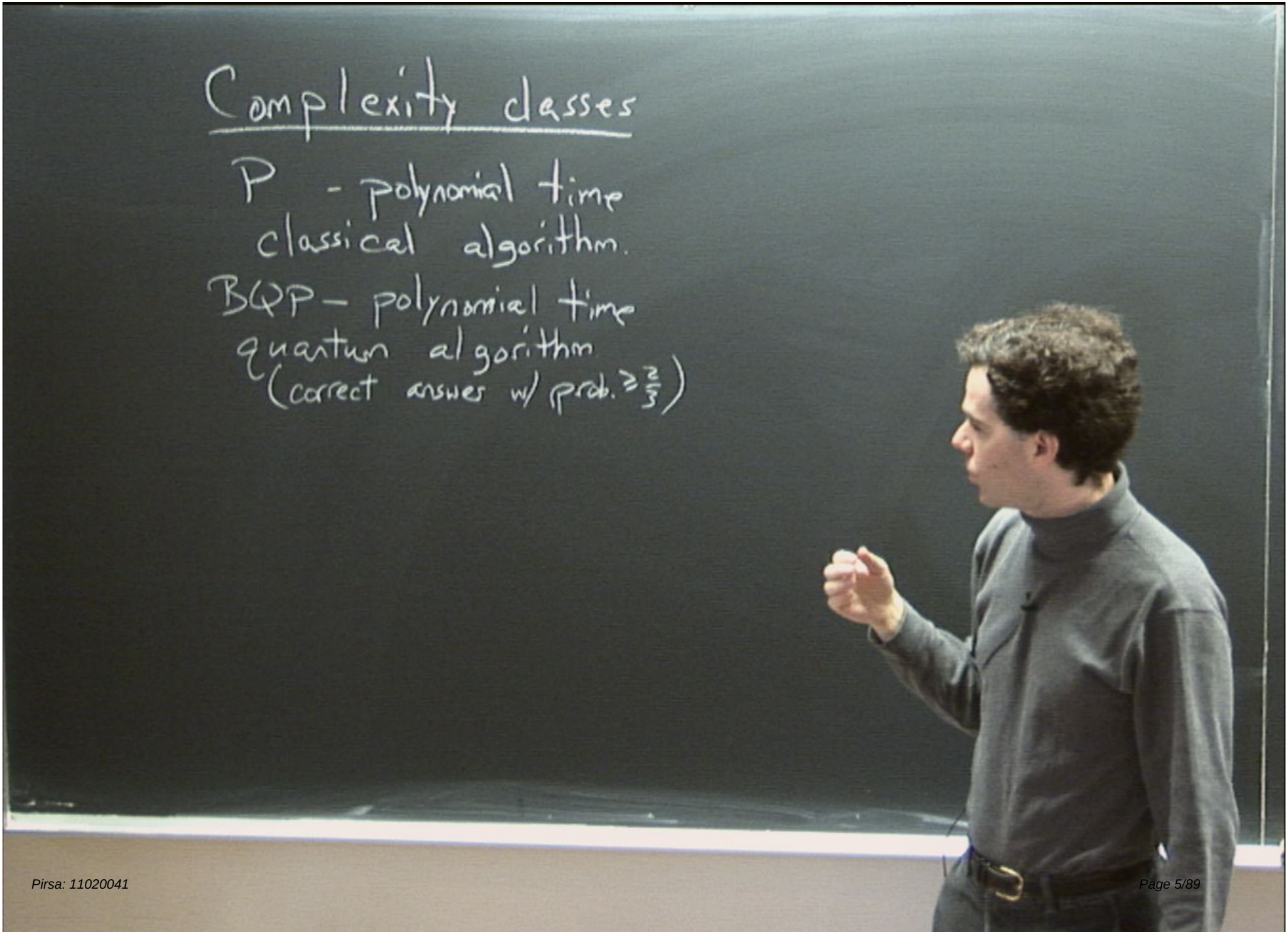
URL: http://pirsa.org/11020041

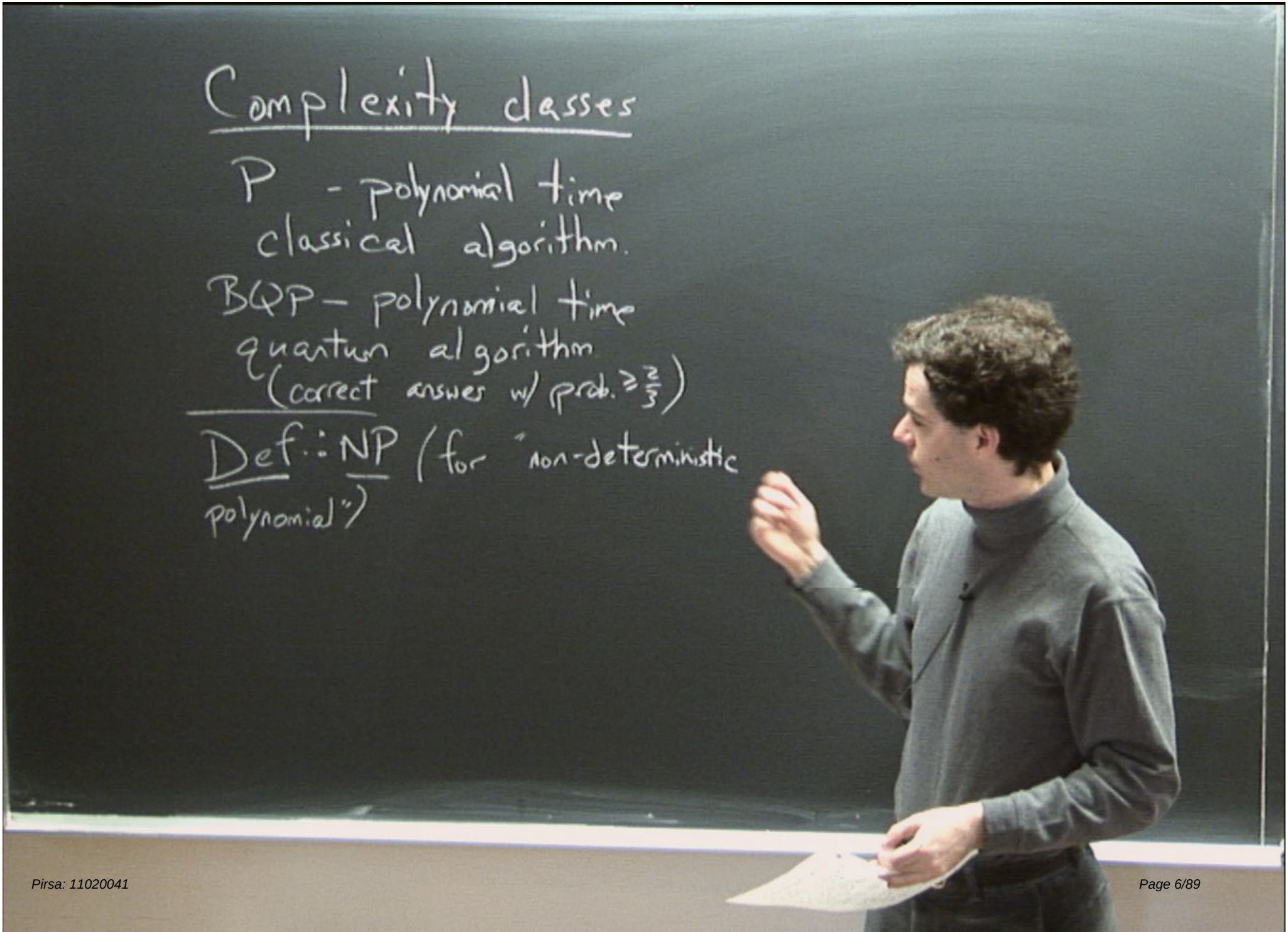
Abstract:

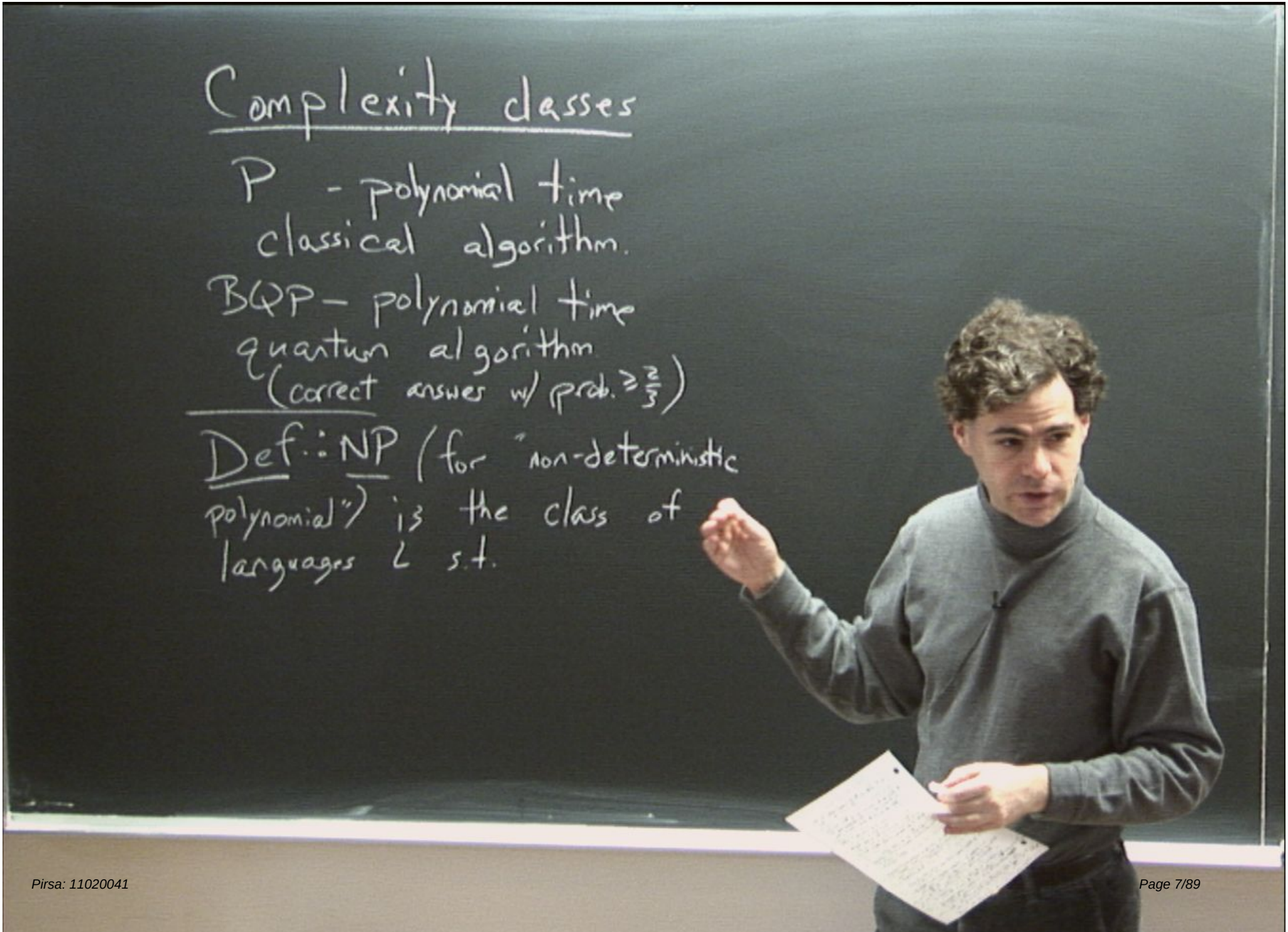

Complexity classes


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm (correct answer w/ prob. ≥ 2/3)


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm (correct answer w/ prob. ≥ 2/3)


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm (correct answer w/ prob. ≥ 2/3)
Def: NP (for "non-deterministic polynomial")


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm (correct answer w/ prob. ≥ 2/3)
Def.: NP (for "non-deterministic polynomial") is the class of languages L s.t.

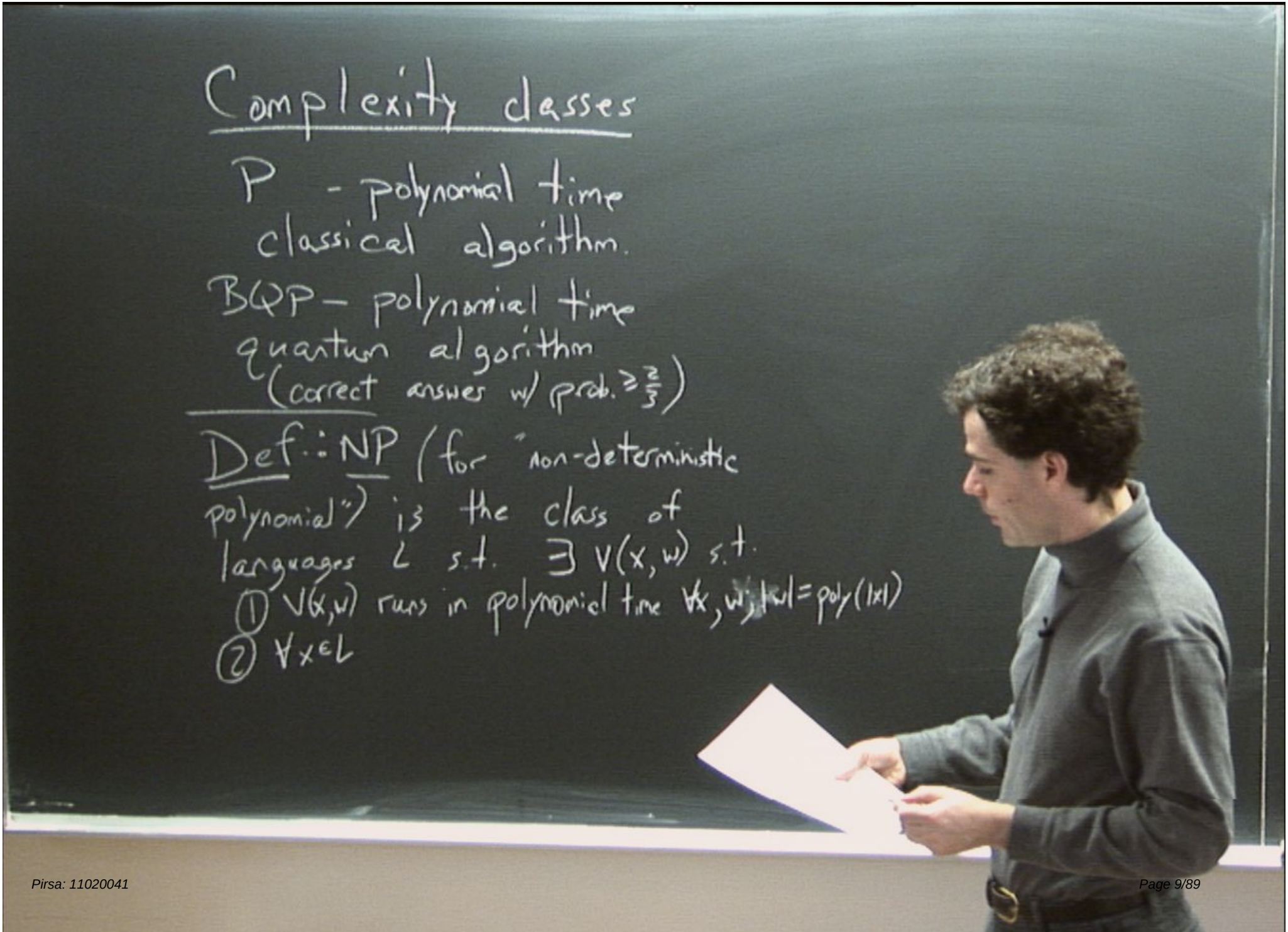
Complexity classes

P - polynomial time classical algorithm.

BQP - polynomial time quantum algorithm (correct answer w/ prob. $\geq \frac{2}{3}$)

Def.: NP (for "non-deterministic polynomial") is the class of languages L s.t. $\exists\, V(x, w)$ s.t.

① $V(x,w)$ runs in polynomial time $\forall x, w$; $|w| = \text{poly}(|x|)$


Complexity classes
P - polynomial time classical algorithm.
BQP - polynomial time quantum algorithm (correct answer w/ prob. ≥ 2/3)
Def.: NP (for "non-deterministic polynomial") is the class of languages L s.t. ∃ V(x,w) s.t.
① V(x,w) runs in polynomial time ∀x, w; |w| = poly(|x|)
② ∀x∈L

Complexity classes

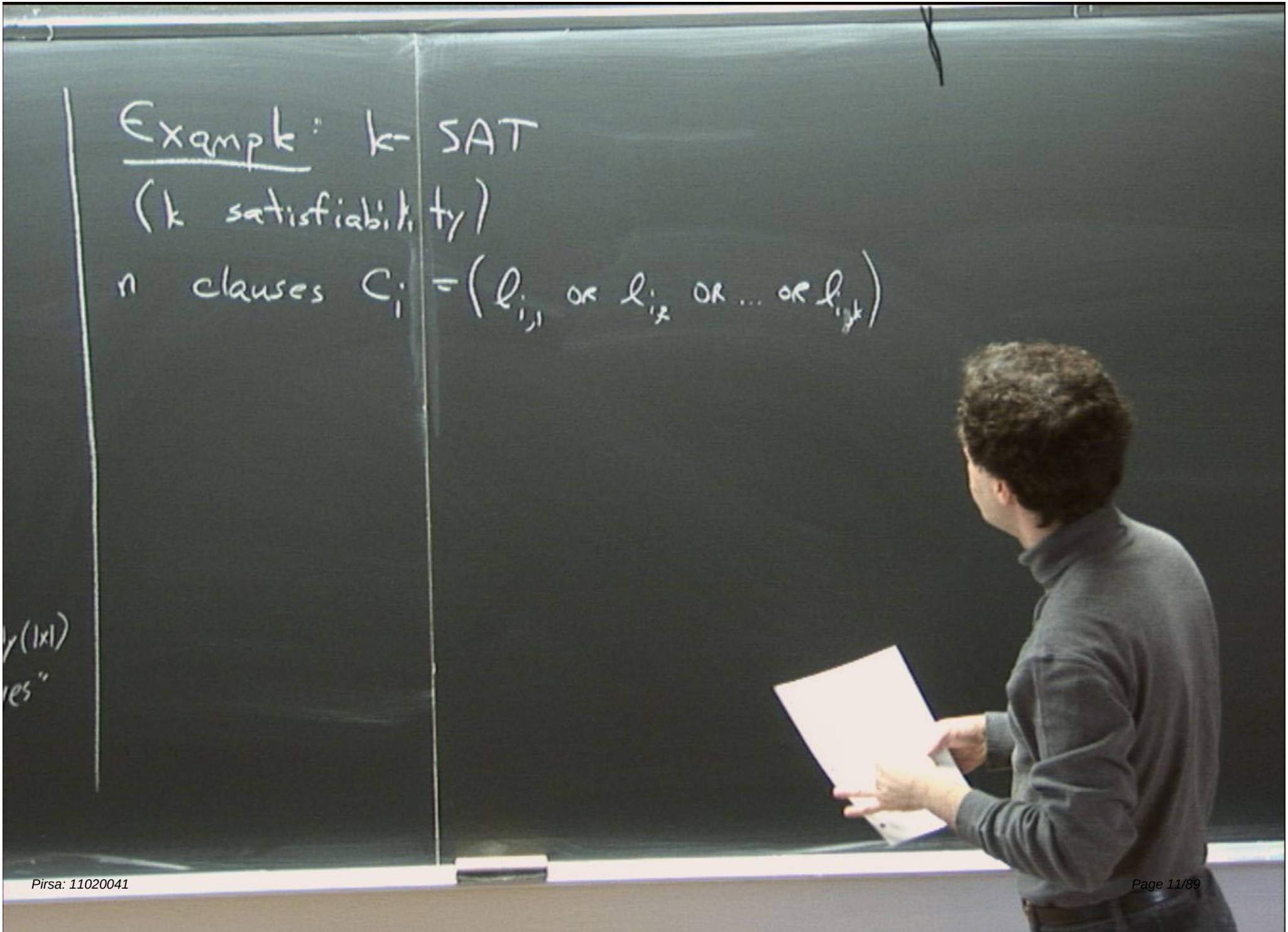
P - polynomial time classical algorithm.

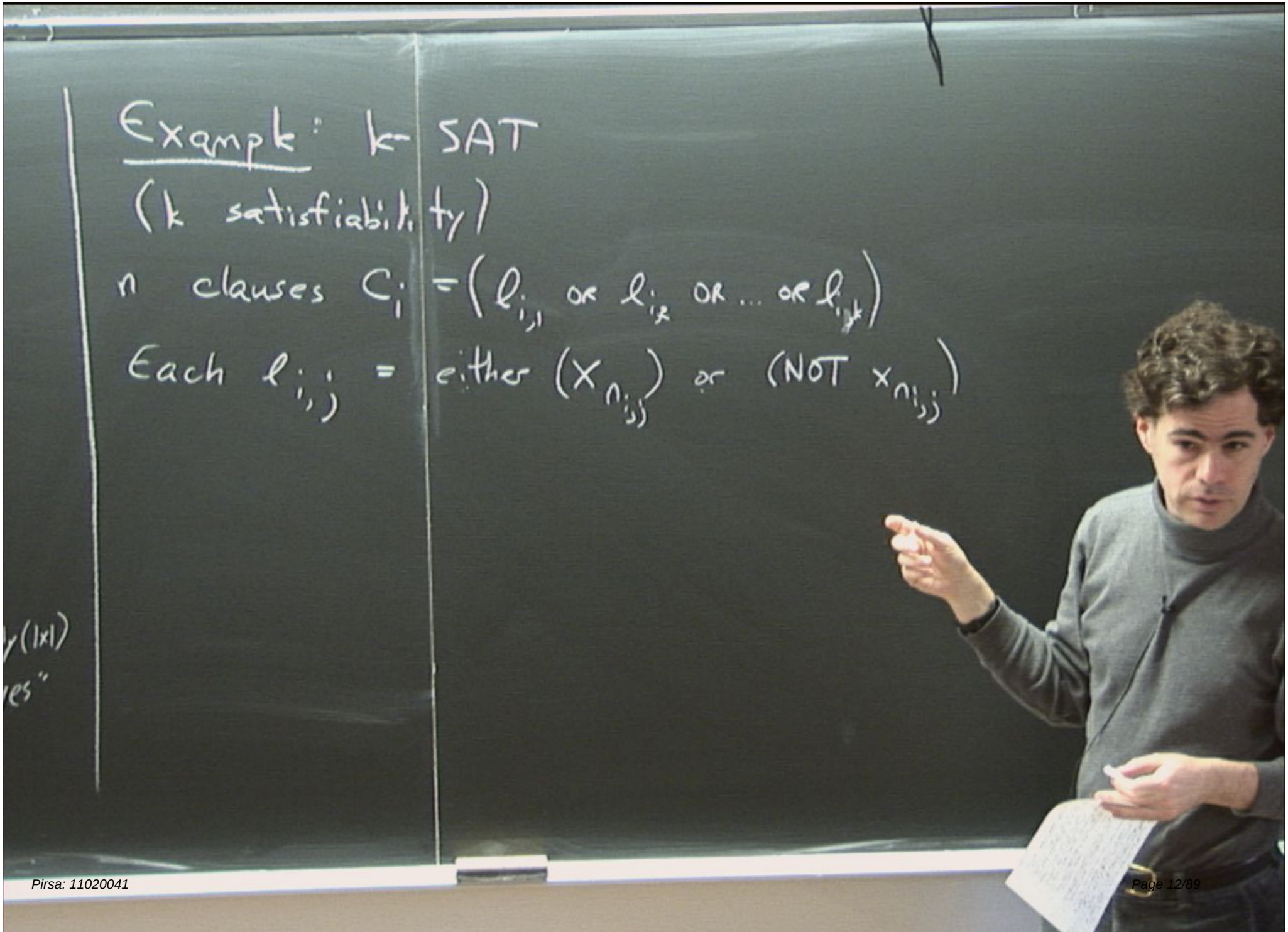
BQP - polynomial time quantum algorithm (correct answer w/ prob. $\geq \frac{2}{3}$)

Def.: NP (for "non-deterministic polynomial") is the class of languages L s.t. $\exists\, V(x,w)$ s.t.

1. $V(x,w)$ runs in polynomial time $\forall x, w$; $|w| = \text{poly}(|x|)$
2. $\forall x \in L$, $\exists w_x$ s.t. $|w_x| = \text{poly}(|x|)$, $V(x,w_x) = \text{"yes"}$
3. $\forall x \notin L$, $\forall w\ (|w| = \text{poly}(|x|))$, $V(x,w) = \text{"no"}$

w_x is the witness for a yes instance x.


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

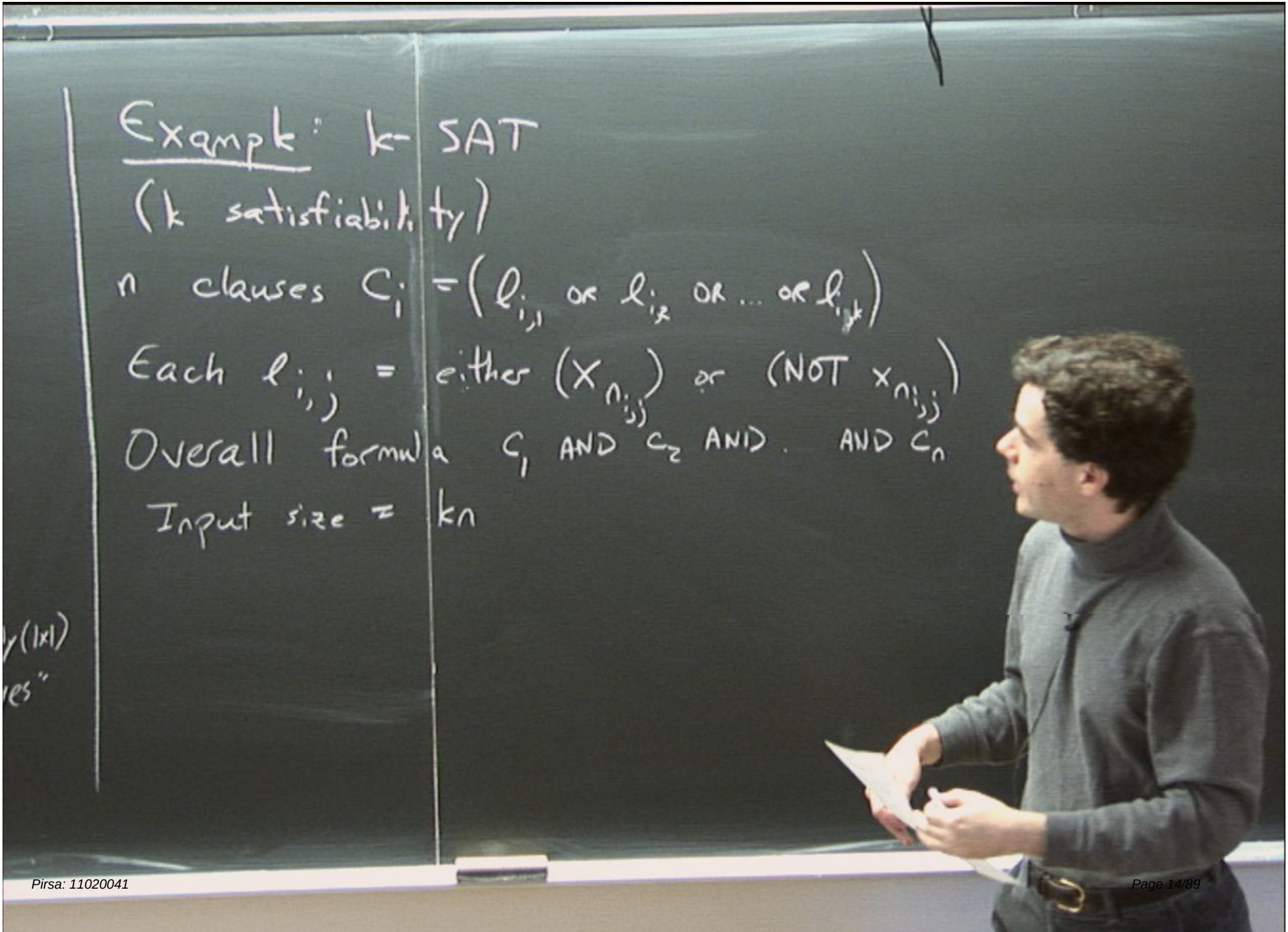
Example: k-SAT

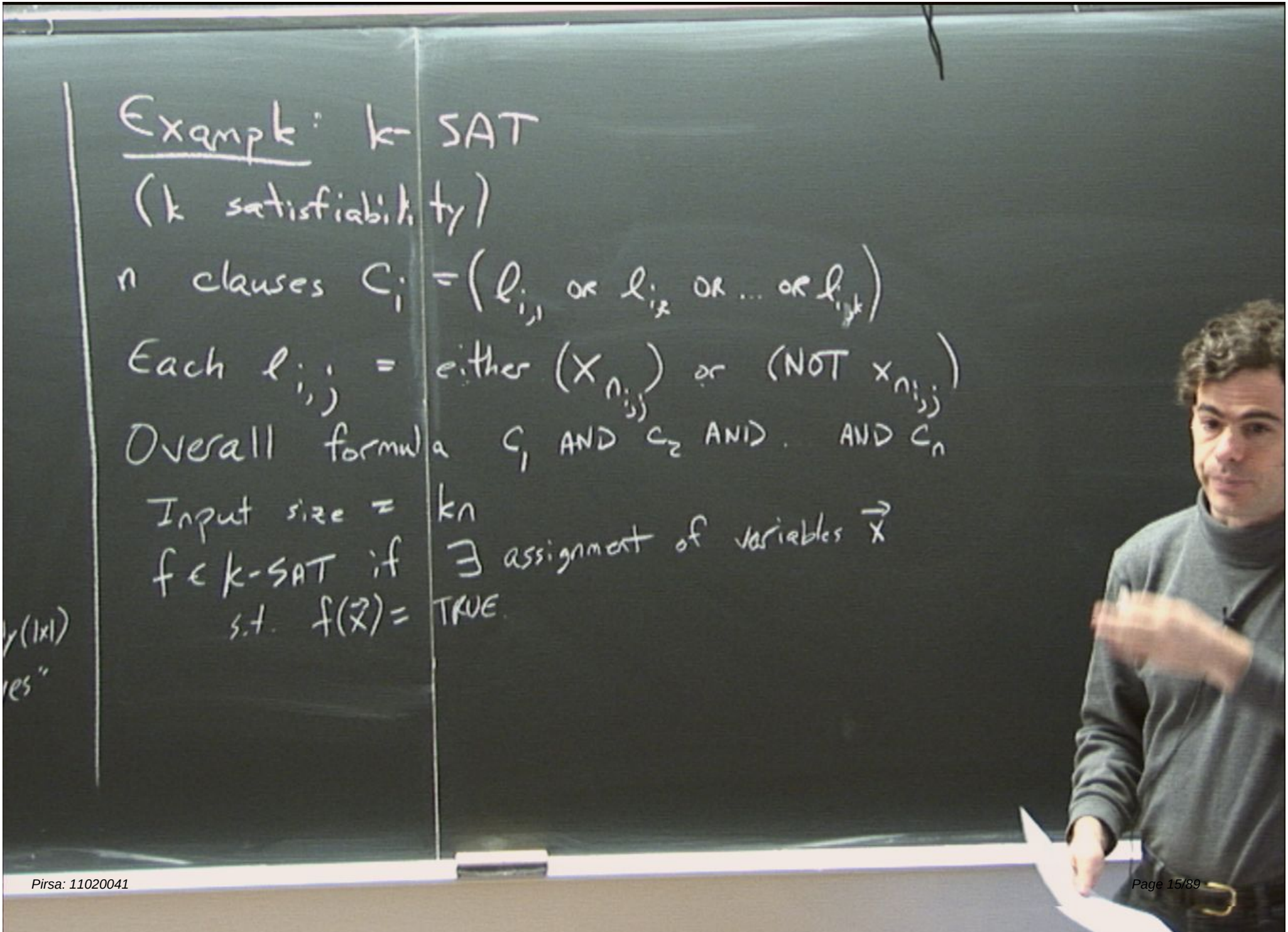
(k satisfiability)

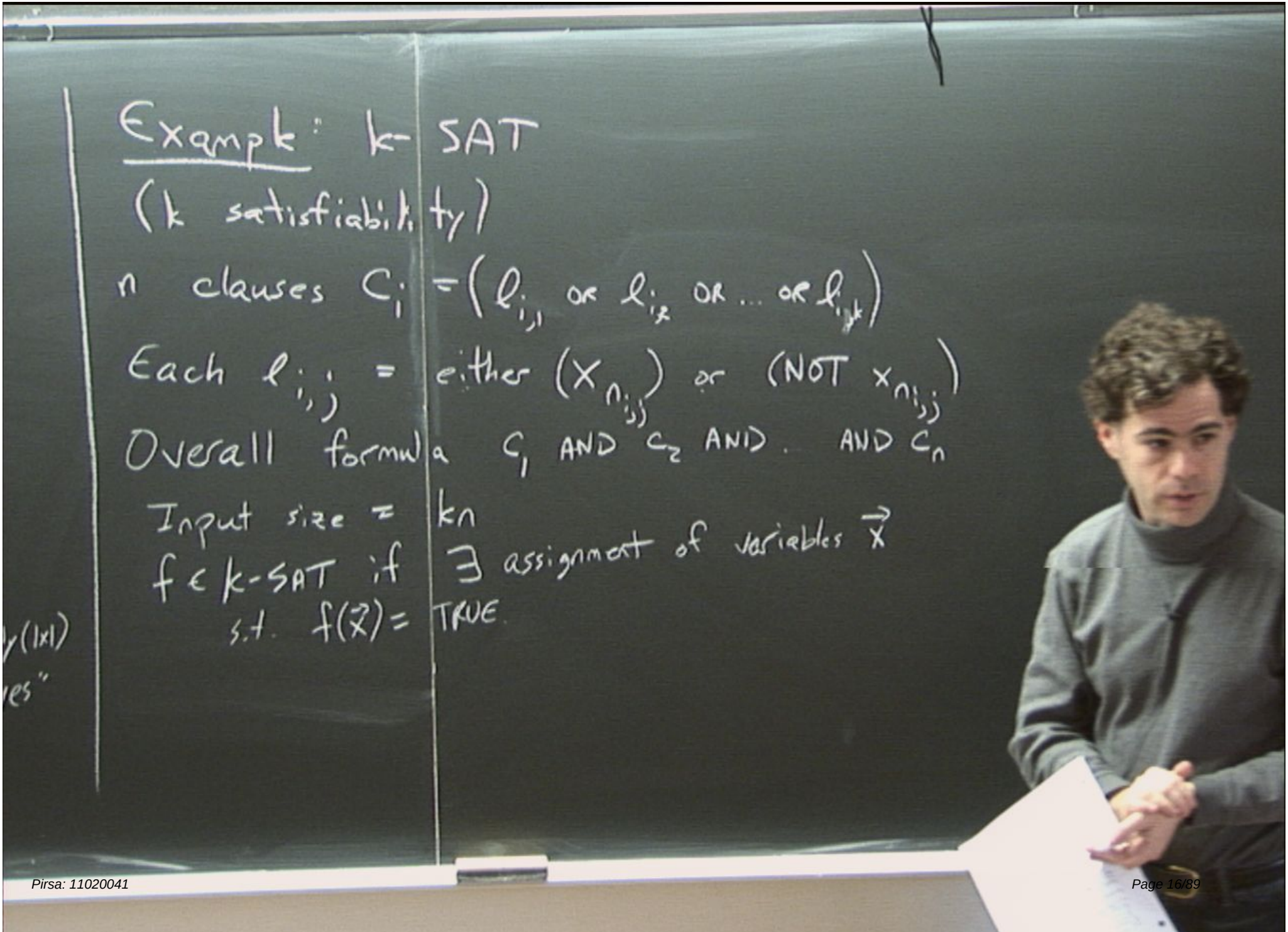
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$

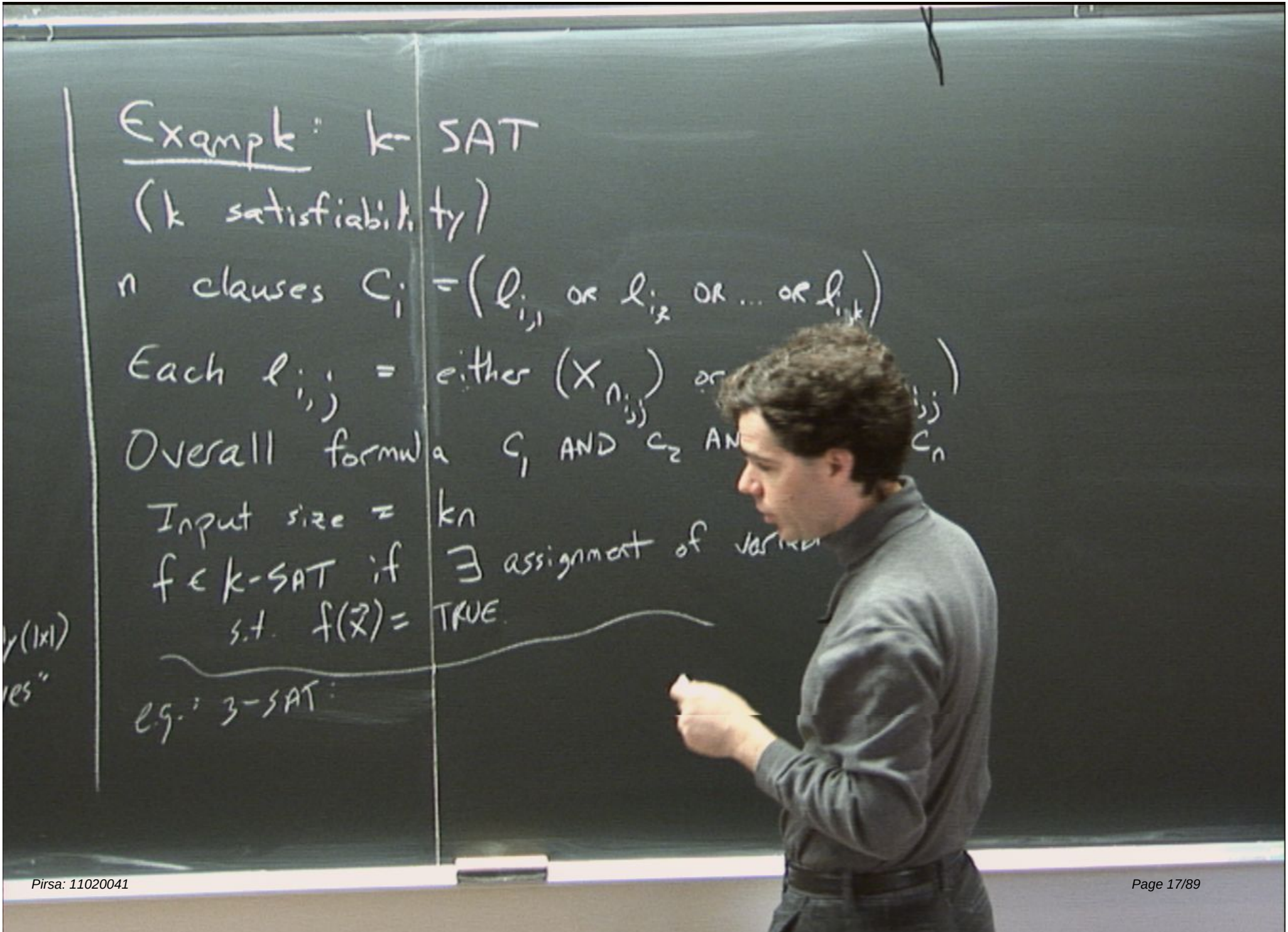
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

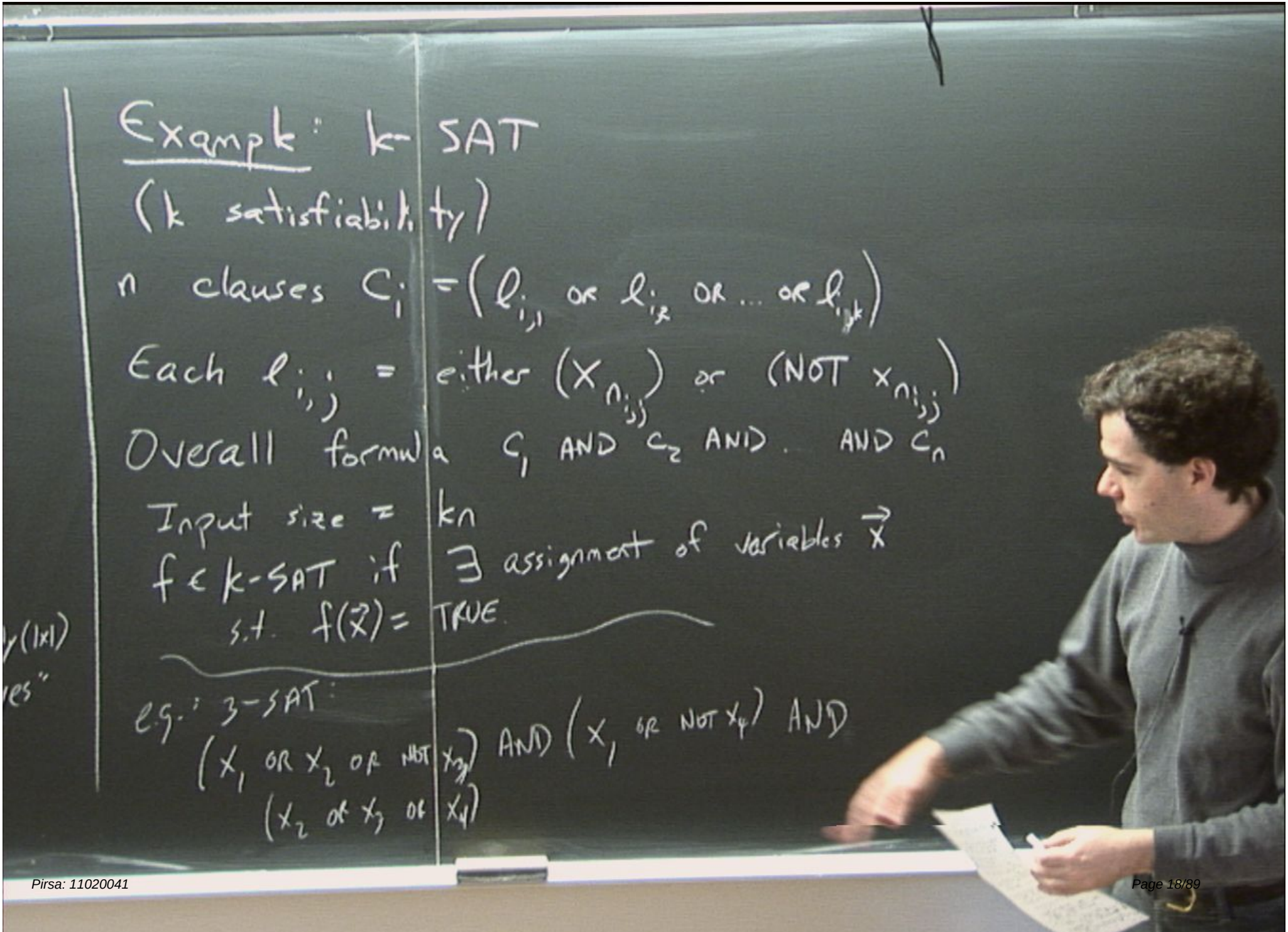
Overall formula C_1 AND C_2 AND . AND C_n

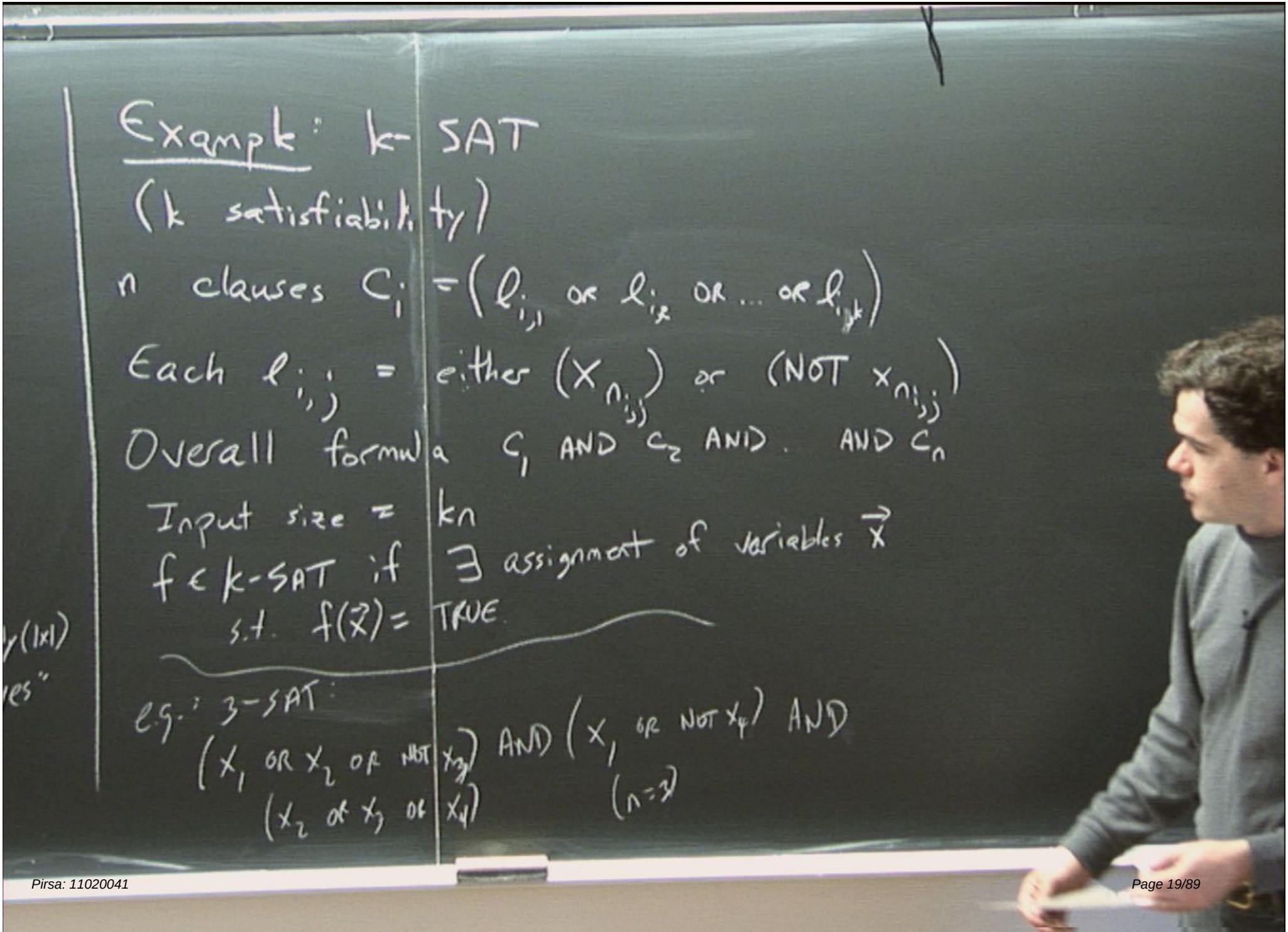

Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn

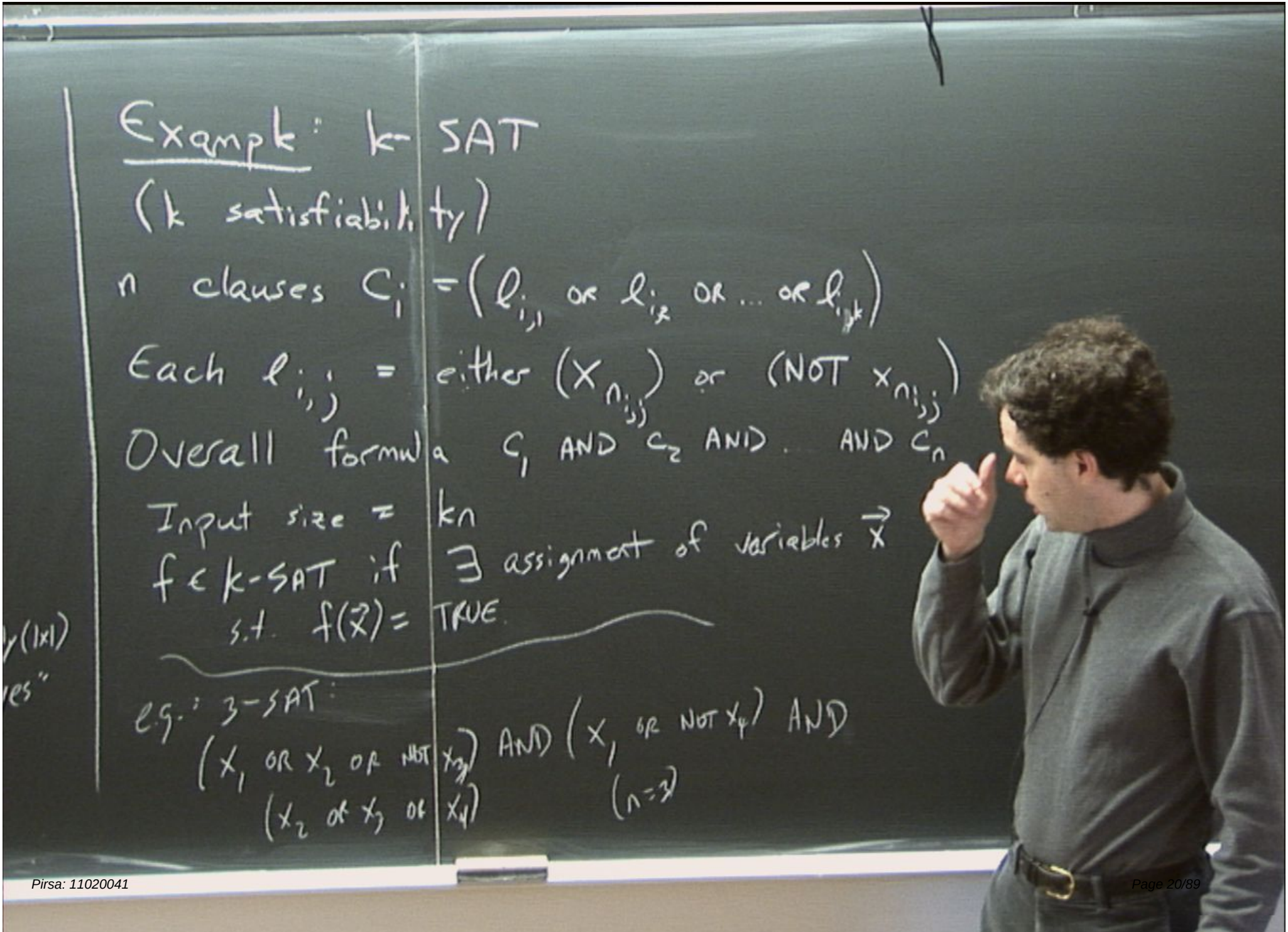

Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE

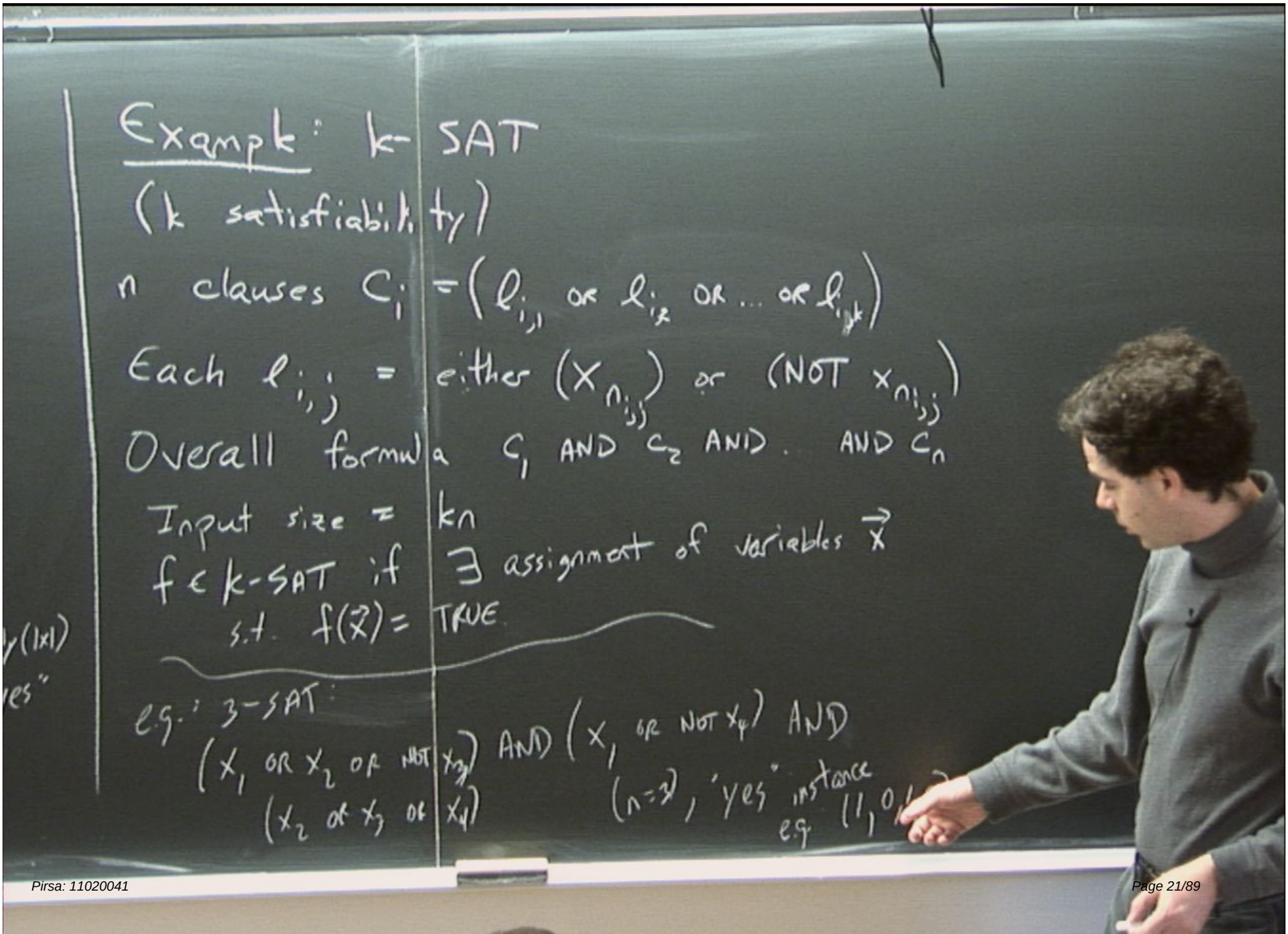

Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE.


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(X_{n_{i,j}})$ or
Overall formula C_1 AND C_2 AN
C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of
s.t. $f(\vec{x})$ = TRUE
e.g.: 3-SAT:


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE
e.g.: 3-SAT:
(x_1 OR x_2 OR NOT x_3) AND (x_1 OR NOT x_4) AND
(x_2 OR x_3 OR x_4)


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE
e.g.: 3-SAT:
(x_1 OR x_2 OR NOT x_3) AND (x_1 OR NOT x_4) AND
(x_2 OR x_3 OR x_4)
(n=3)


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE
e.g.: 3-SAT:
(x_1 OR x_2 OR NOT x_3) AND (x_1 OR NOT x_4) AND
(x_2 OR x_3 OR x_4)
(n=3)


Example: k-SAT
(k satisfiability)
n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$
Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
Overall formula C_1 AND C_2 AND . AND C_n
Input size = kn
$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$
s.t. $f(\vec{x})$ = TRUE
e.g.: 3-SAT:
$(x_1$ OR x_2 OR NOT $x_3)$ AND $(x_1$ OR NOT $x_4)$ AND
$(x_2$ OR x_3 OR $x_4)$
(n=3, 'yes' instance
e.g. (1, 0,

Example: k-SAT

(k satisfiability)

n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$

Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or $(\text{NOT } x_{n_{i,j}})$

Overall formula C_1 AND C_2 AND . AND C_n

Input size = kn

$f \in k\text{-SAT}$ if $\exists$ assignment of variables $\vec{x}$ s.t. $f(\vec{x}) =$ TRUE

e.g.: 3-SAT

$(x_1$ OR x_2 OR NOT $x_3)$ AND $(x_1$ OR NOT $x_4)$ AND

$(x_2$ OR x_3 OR $x_4)$

$(n=3)$, "yes" instance

e.g. $(1,0,1,0)$

Example: k-SAT

(k satisfiability)

n clauses $C_i = (\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k})$

Each $\ell_{i,j}$ = either $(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

Overall formula C_1 AND C_2 AND . AND C_n

Input size = kn

$f \in$ k-SAT if $\exists$ assignment of variables $\vec{x}$ s.t. $f(\vec{x}) =$ TRUE

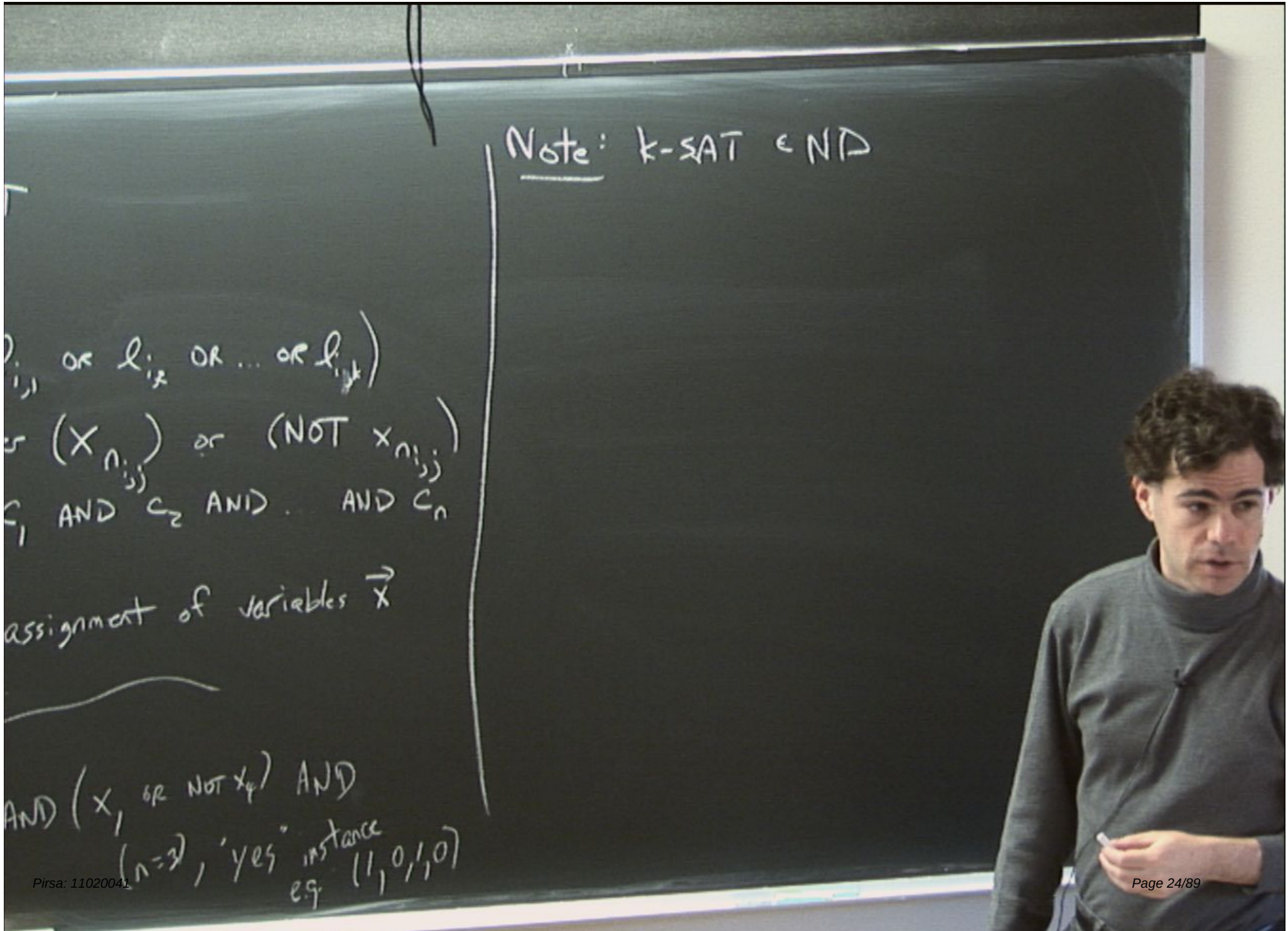
e.g.: 3-SAT:

$(x_1$ OR x_2 OR NOT $x_3)$ AND $(x_1$ OR NOT $x_4)$ AND

$(x_2$ OR x_3 OR $x_4)$

$(n=3)$, "yes" instance

e.g. $(1,0,1,0)$


Note: k-SAT ∈ NP
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND (x_1 OR NOT x_4) AND
(n=3, 'yes' instance
e.g. (1,0,1,0)

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x}) = 1$

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

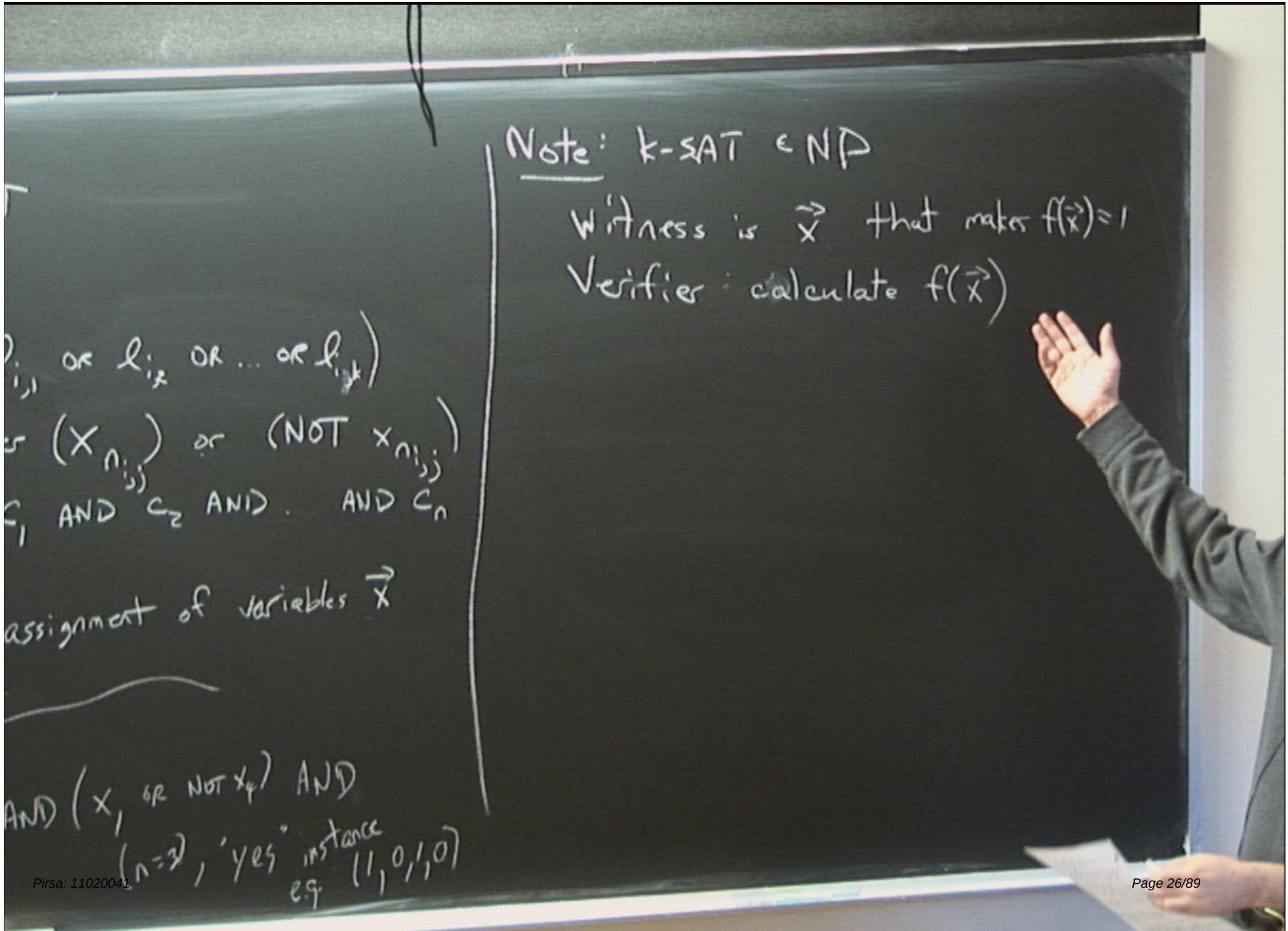
C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND (x_1 OR NOT x_4) AND

($n=3$), "yes" instance

e.g. (1, 0, 1, 0)


Note: k-SAT ∈ NP
Witness is $\vec{x}$ that makes $f(\vec{x})=1$
Verifier: calculate $f(\vec{x})$
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND $(x_1$ OR NOT $x_4)$ AND
$(n=3$, 'yes' instance
e.g. $(1,0,1,0)$

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND (x_1 OR NOT x_4) AND

($n=3$), 'yes' instance

e.g. (1,0,1,0)

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$

(runs in time $O(n)$)

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND (x_1 OR NOT x_4) AND

($n=3$, 'yes' instance
e.g. (1,0,1,0)

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND (x_1 OR NOT x_4) AND

($n=3$), "yes" instance

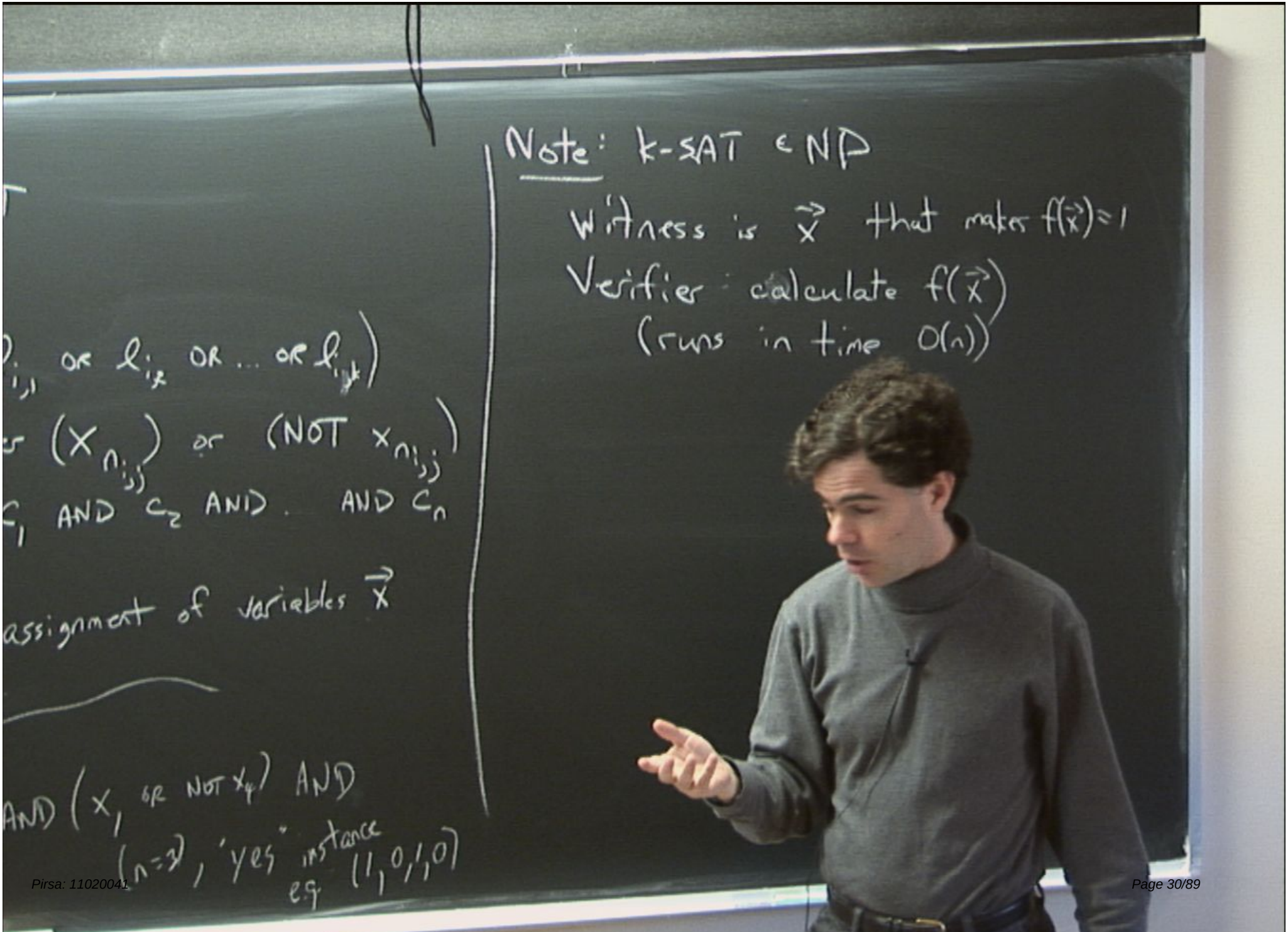
e.g. (1,0,1,0)

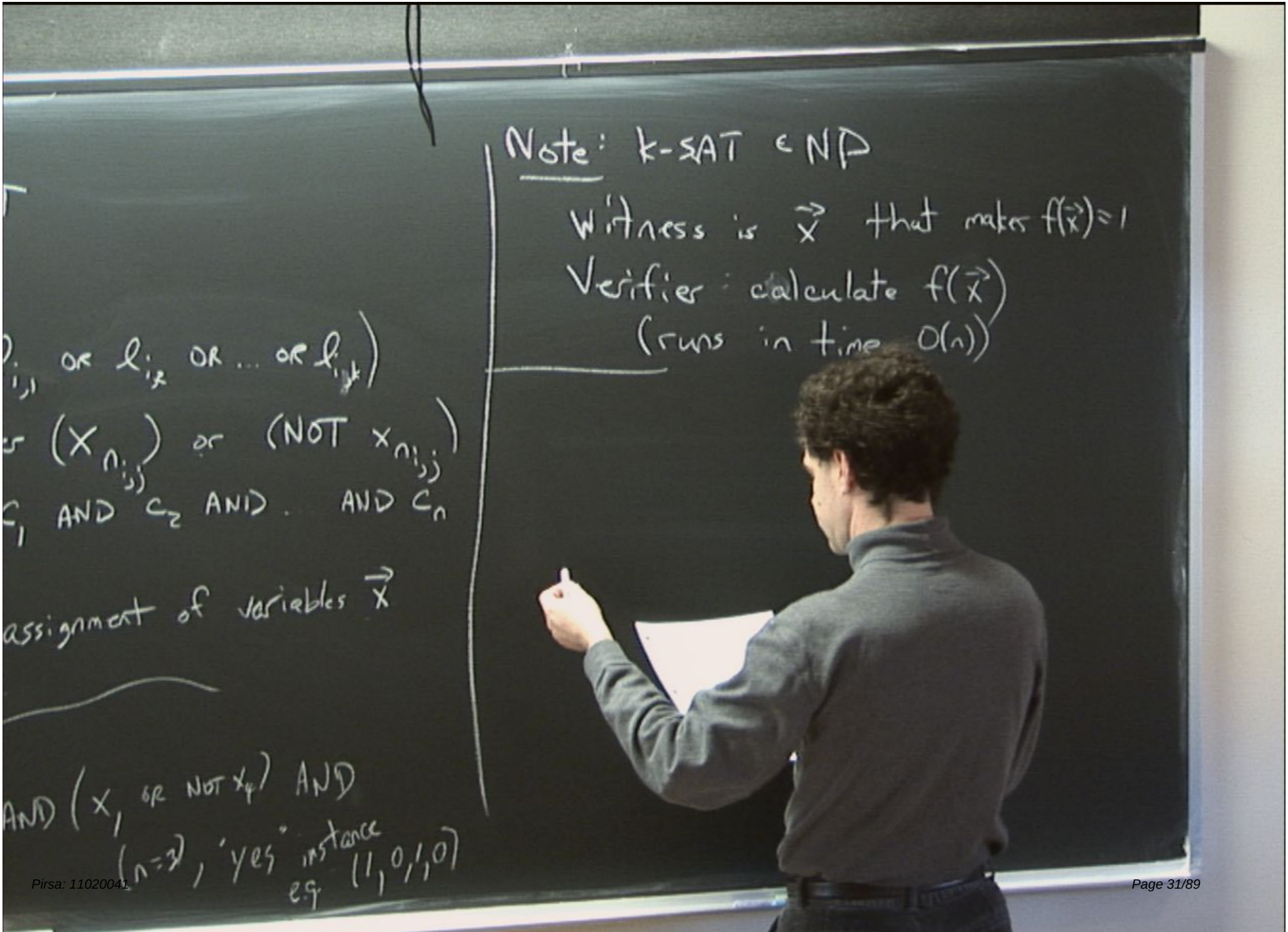
Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$

(runs in time $O(n)$)


Note: k-SAT ∈ NP
Witness is $\vec{x}$ that makes $f(\vec{x})=1$
Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND $(x_1$ OR NOT $x_4)$ AND
$(n=3$, 'yes' instance
e.g. $(1,0,1,0)$


Note: k-SAT ∈ NP
Witness is $\vec{x}$ that make $f(\vec{x})=1$
Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND (x_1 OR NOT x_4) AND
(n=3, 'yes' instance
e.g. (1,0,1,0)

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND $(x_1$ OR NOT $x_4)$ AND

$(n=3$, "yes" instance

e.g. $(1,0,1,0)$

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$

(runs in time $O(n)$)

Def: Language L **reduces** to language M if $\exists$ poly-time $f: \{0,1\}^* \to \{0,1\}^*$

s.t. $x \in L \Leftrightarrow f(x) \in M$

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND $(x_1$ OR NOT $x_4)$ AND

$(n=3$, "yes" instance
e.g. $(1,0,1,0)$

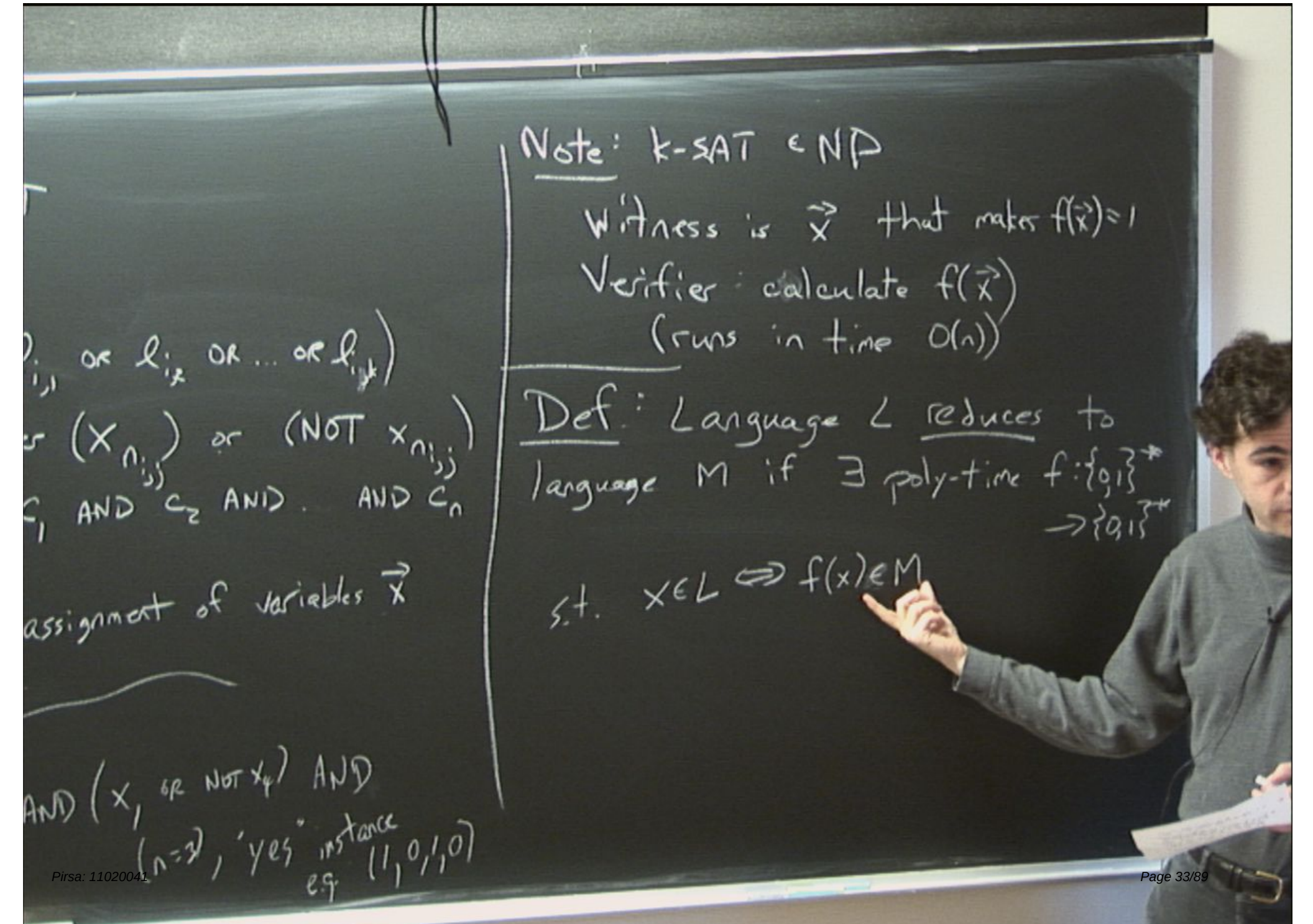
Note: k-SAT $\in$ NP

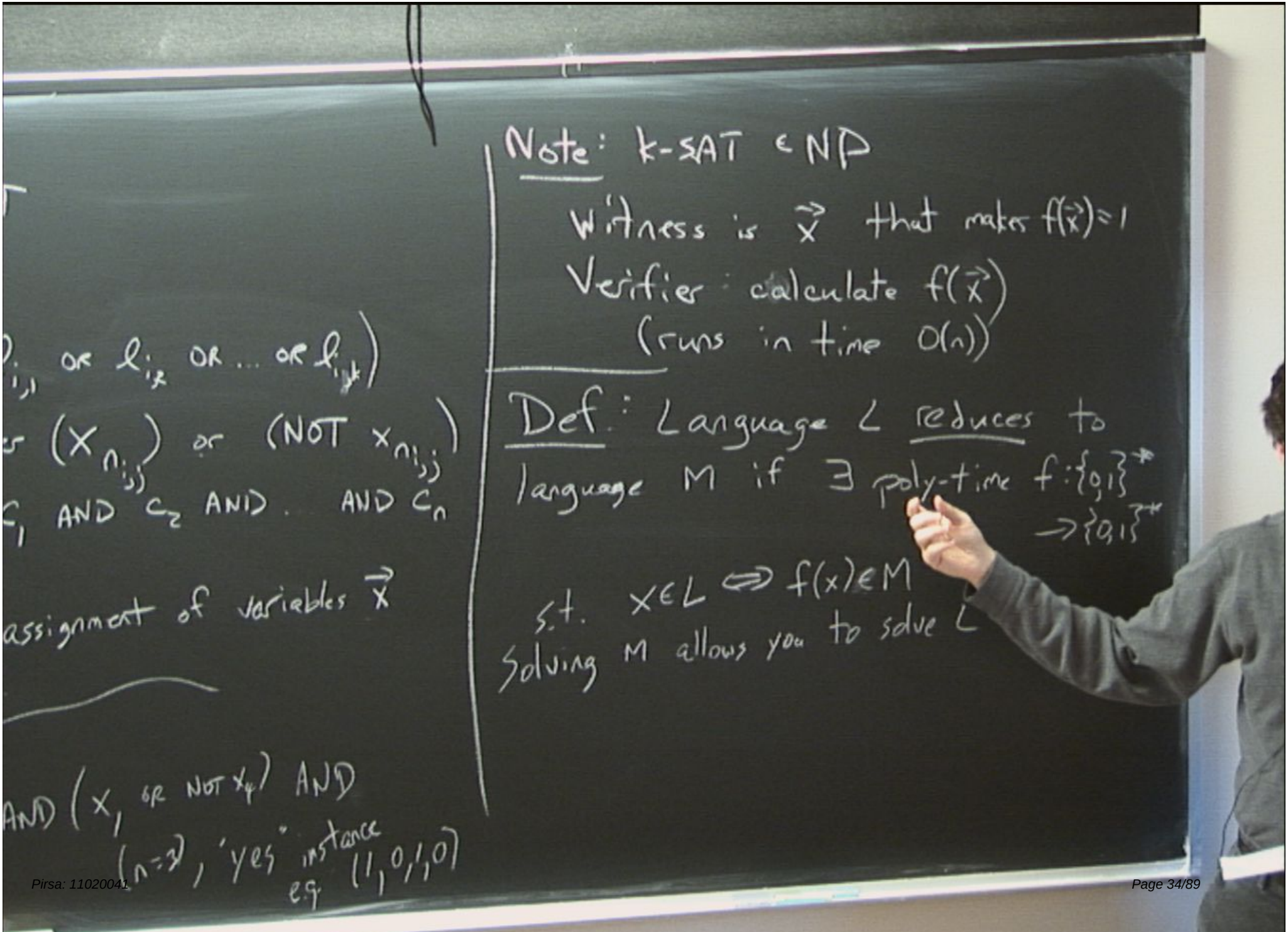
Witness is $\vec{x}$ that makes $f(\vec{x})=1$

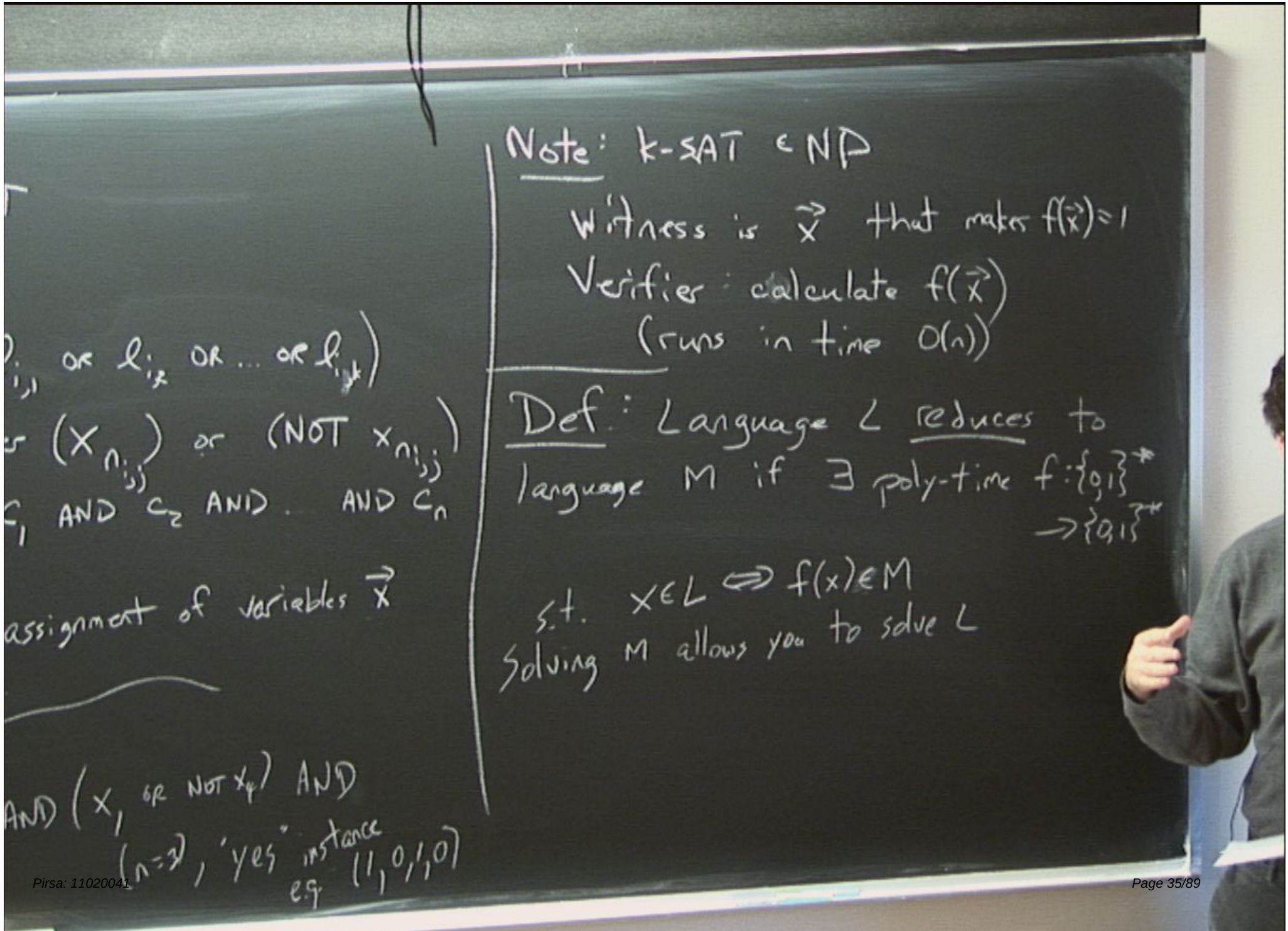
Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)

Def: Language L reduces to language M if $\exists$ poly-time $f: \{0,1\}^* \to \{0,1\}^*$

s.t. $x \in L \Leftrightarrow f(x) \in M$




Note: k-SAT ∈ NP
Witness is $\vec{x}$ that makes $f(\vec{x})=1$
Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)
Def: Language L reduces to language M if $\exists$ poly-time $f:\{0,1\}^* \to \{0,1\}^*$
s.t. $x \in L \Leftrightarrow f(x) \in M$
Solving M allows you to solve L
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND $(x_1$ OR NOT $x_4)$ AND
$(n=3$, 'yes' instance
e.g. $(1,0,1,0)$


Note: k-SAT ∈ NP
Witness is $\vec{x}$ that makes $f(\vec{x})=1$
Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)
Def: Language L reduces to language M if ∃ poly-time $f:\{0,1\}^* \to \{0,1\}^*$
s.t. $x \in L \Leftrightarrow f(x) \in M$
Solving M allows you to solve L
$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)
$(X_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)
C_1 AND C_2 AND . AND C_n
assignment of variables $\vec{x}$
AND $(x_1$ OR NOT $x_4)$ AND
(n=3, 'yes' instance
e.g. (1,0,1,0)

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

or $(x_{n_{i,j}})$ or $(\text{NOT } x_{n_{i,j}})$

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND $(x_1$ OR NOT $x_4)$ AND

$(n=3$, "yes" instance
e.g. $(1,0,1,0)$

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)

Def: Language L **reduces** to language M if $\exists$ poly-time $f: \{0,1\}^* \to \{0,1\}^*$

s.t. $x \in L \iff f(x) \in M$

Solving M allows you to solve L

Def: A language M is **NP-complete** if $\forall L \in NP$, L reduces to M.

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND $(x_1$ OR NOT $x_4)$ AND

$(n=3$, "yes" instance
e.g. $(1,0,1,0)$

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)

Def: Language L **reduces** to language M if $\exists$ poly-time $f: \{0,1\}^* \to \{0,1\}^*$

s.t. $x \in L \iff f(x) \in M$

Solving M allows you to solve L

Def: A language M is **NP-complete**
if $\forall L \in NP$, L reduces to M.

Thm. (Cook-Levin): 3-SAT is NP-complete

$\ell_{i,1}$ OR $\ell_{i,2}$ OR ... OR $\ell_{i,k}$)

$(x_{n_{i,j}})$ or (NOT $x_{n_{i,j}}$)

C_1 AND C_2 AND . AND C_n

assignment of variables $\vec{x}$

AND $(x_1$ OR NOT $x_4)$ AND

$(n=3$, "yes" instance
e.g. $(1,0,1,0)$

Note: k-SAT $\in$ NP

Witness is $\vec{x}$ that makes $f(\vec{x})=1$

Verifier: calculate $f(\vec{x})$
(runs in time $O(n)$)

Def: Language L **reduces** to language M if $\exists$ poly-time $f: \{0,1\}^* \to \{0,1\}^*$

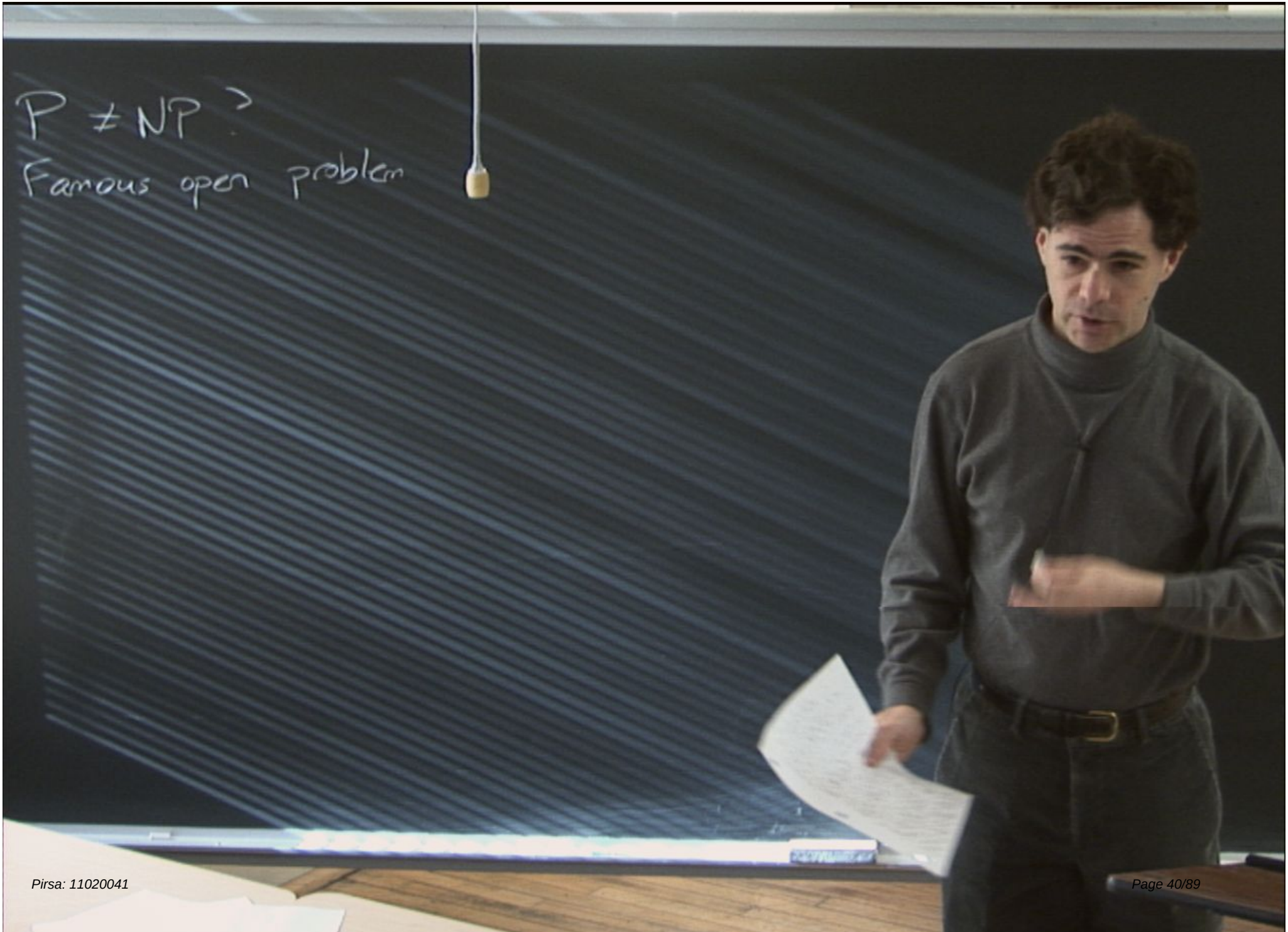
s.t. $x \in L \Leftrightarrow f(x) \in M$

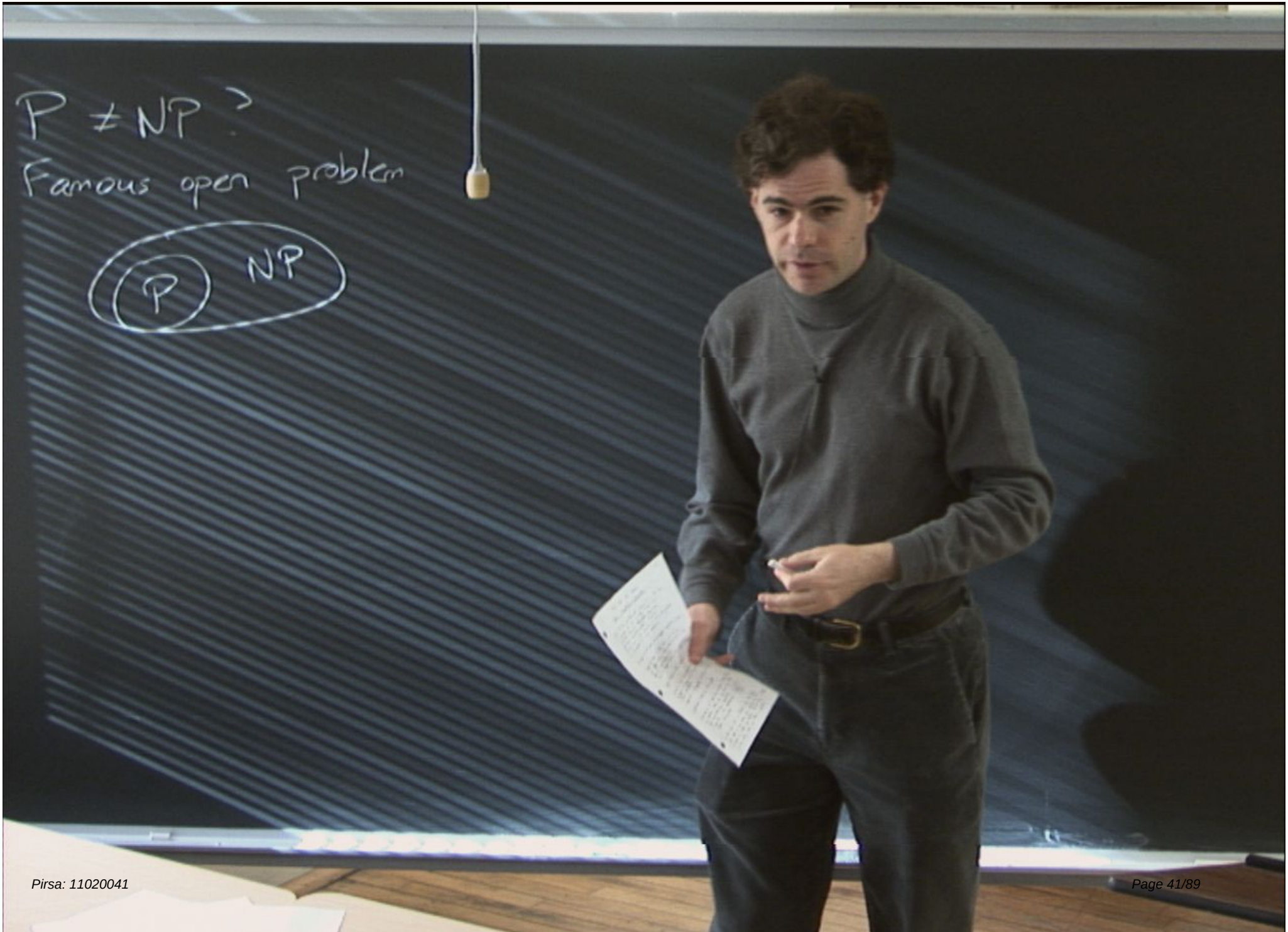
Solving M allows you to solve L

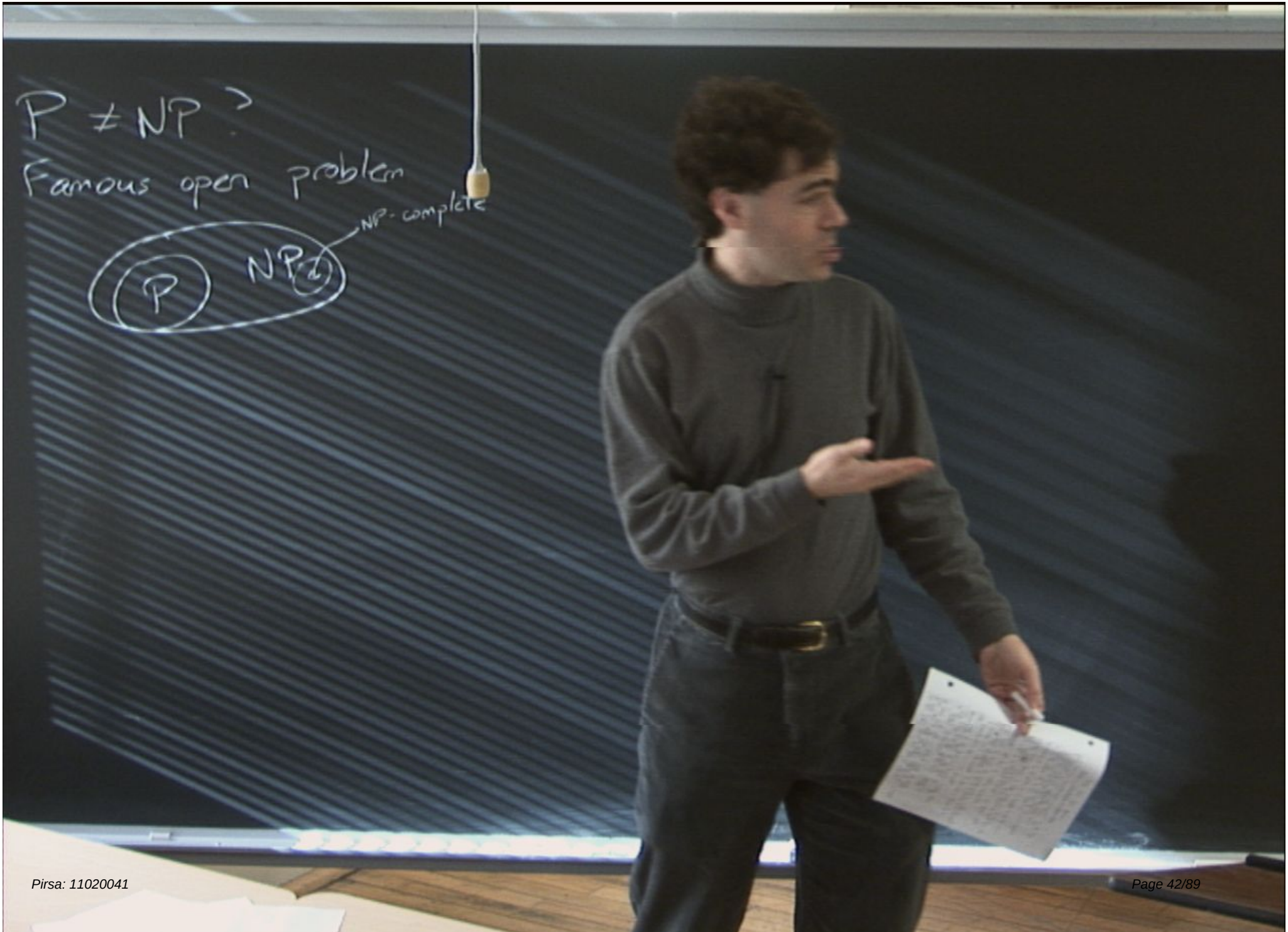
Def: A language M is **NP-complete**
if $\forall L \in NP$, L reduces to M.

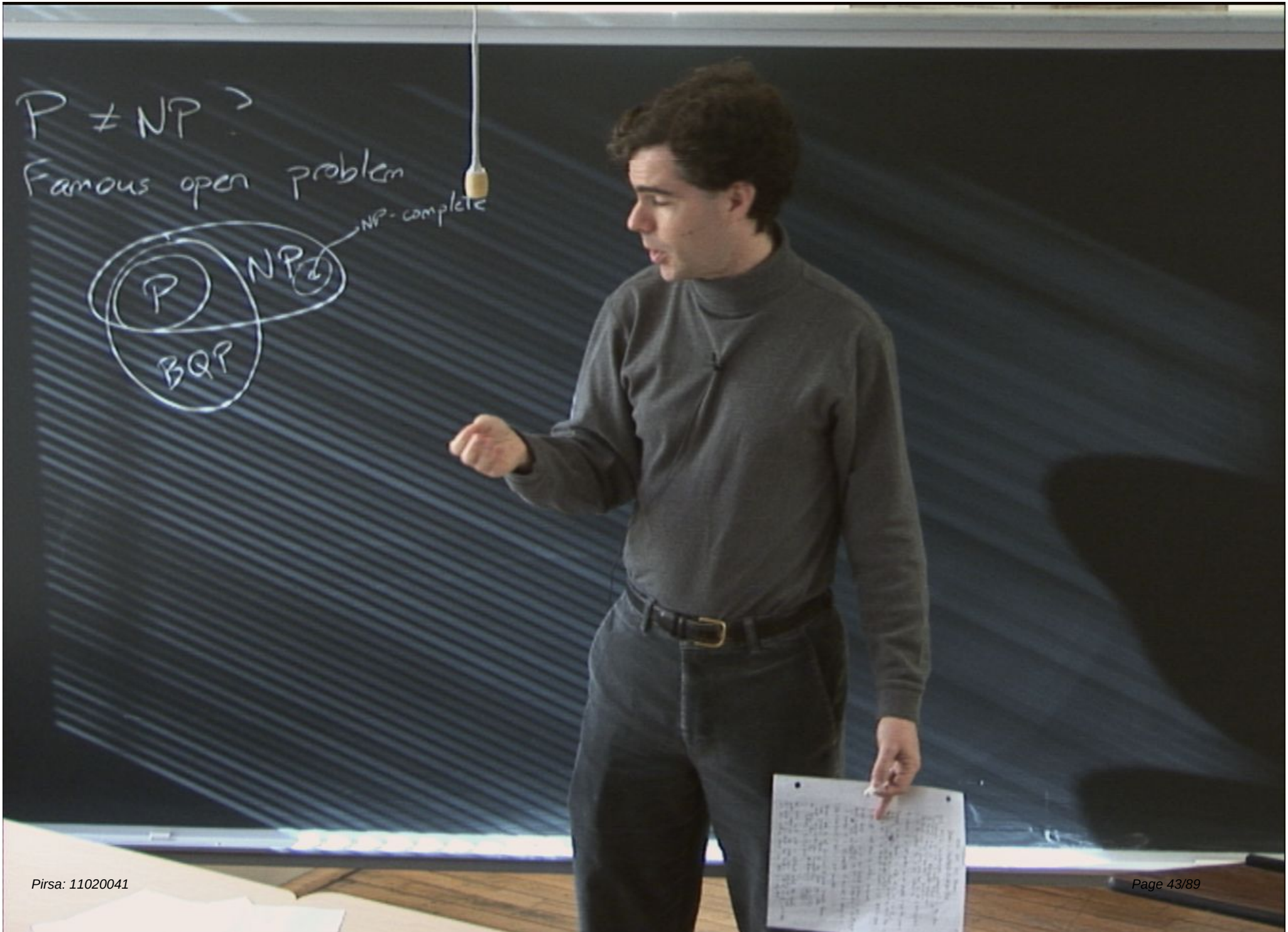
Thm. (Cook-Levin): 3-SAT is NP-complete

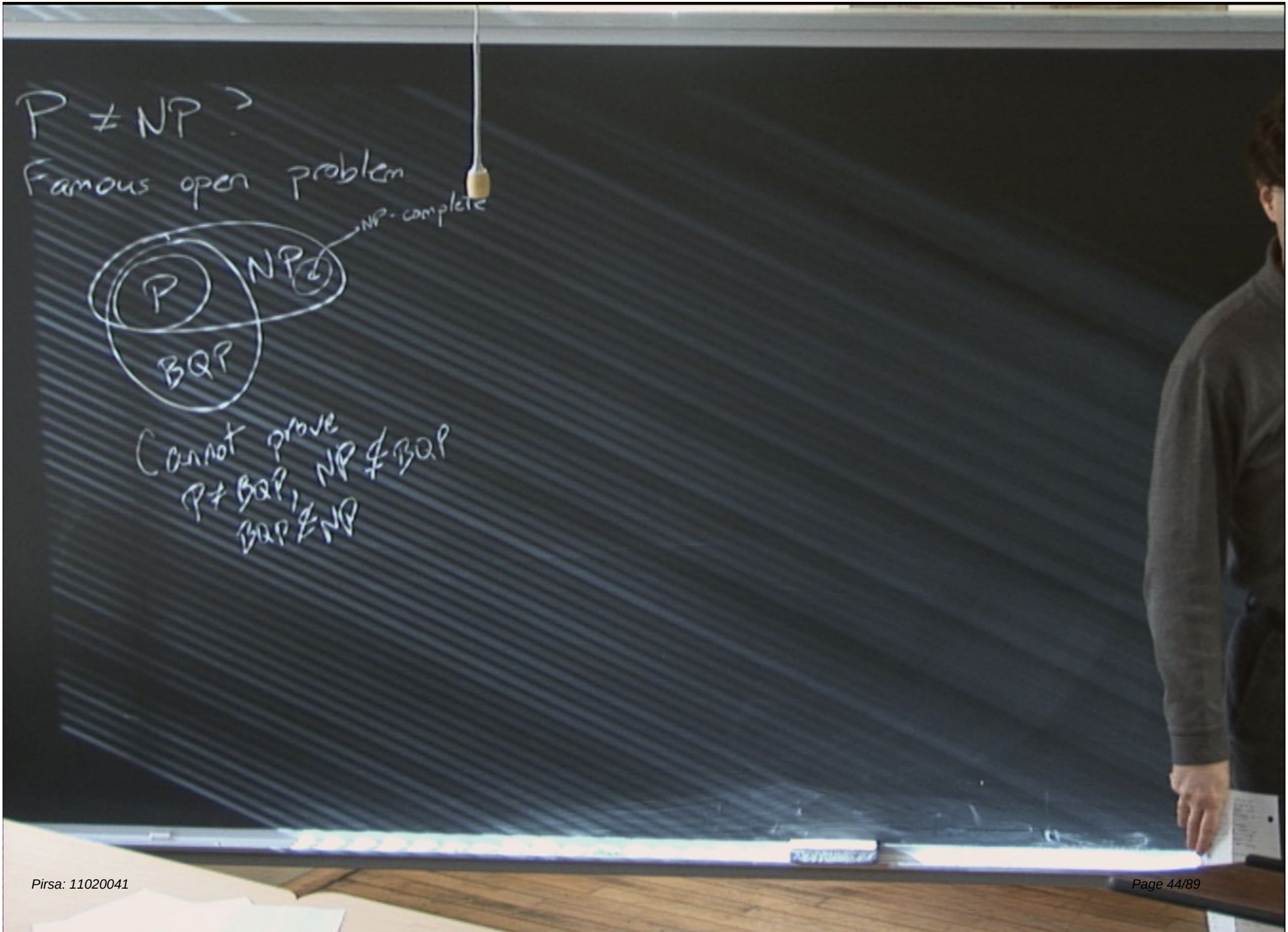

P ≠ NP ?

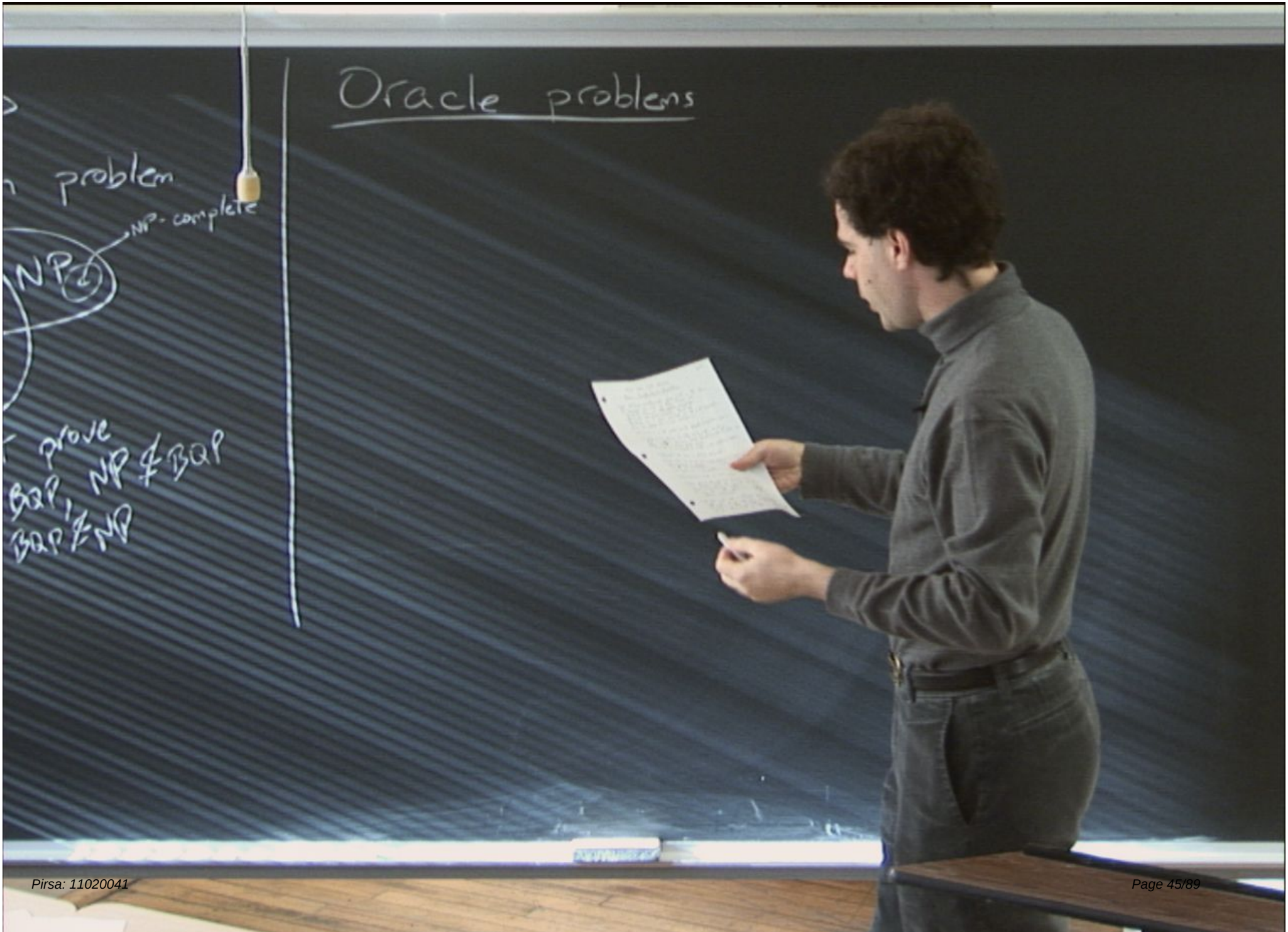

P ≠ NP ?
Famous open problem

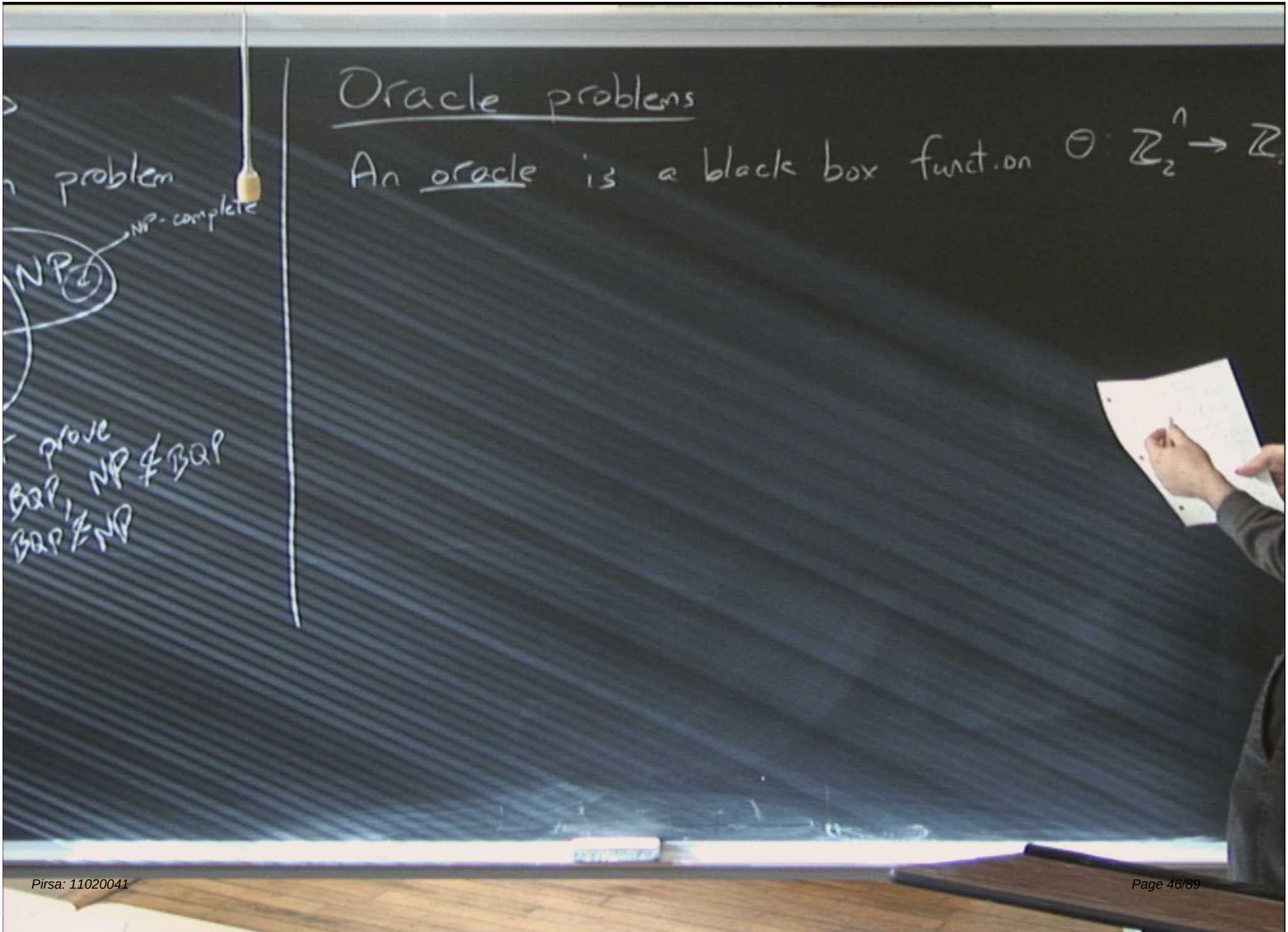

P ≠ NP ?
Famous open problem
P
NP

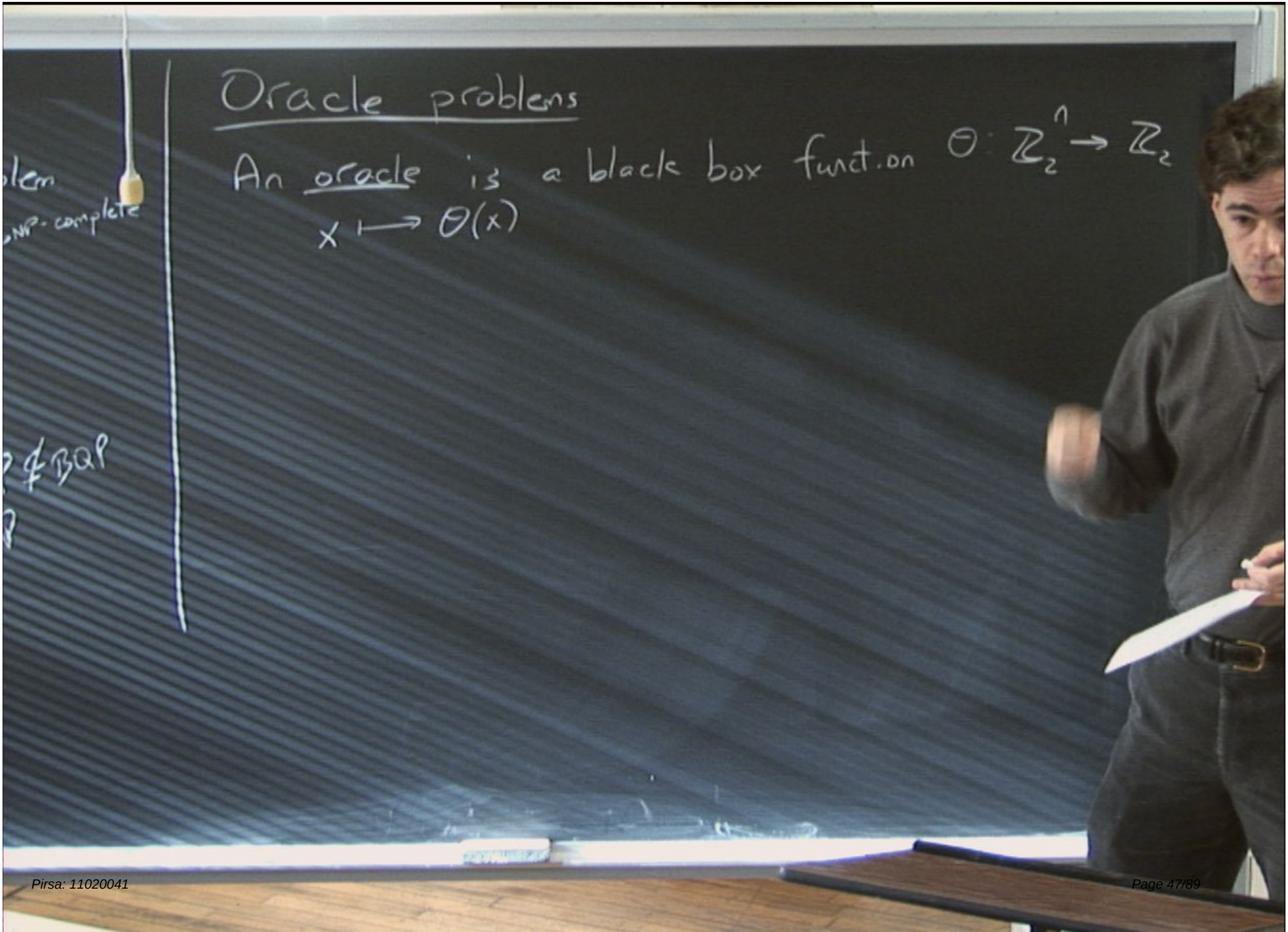

P ≠ NP ?
Famous open problem
NP - complete
P
NP

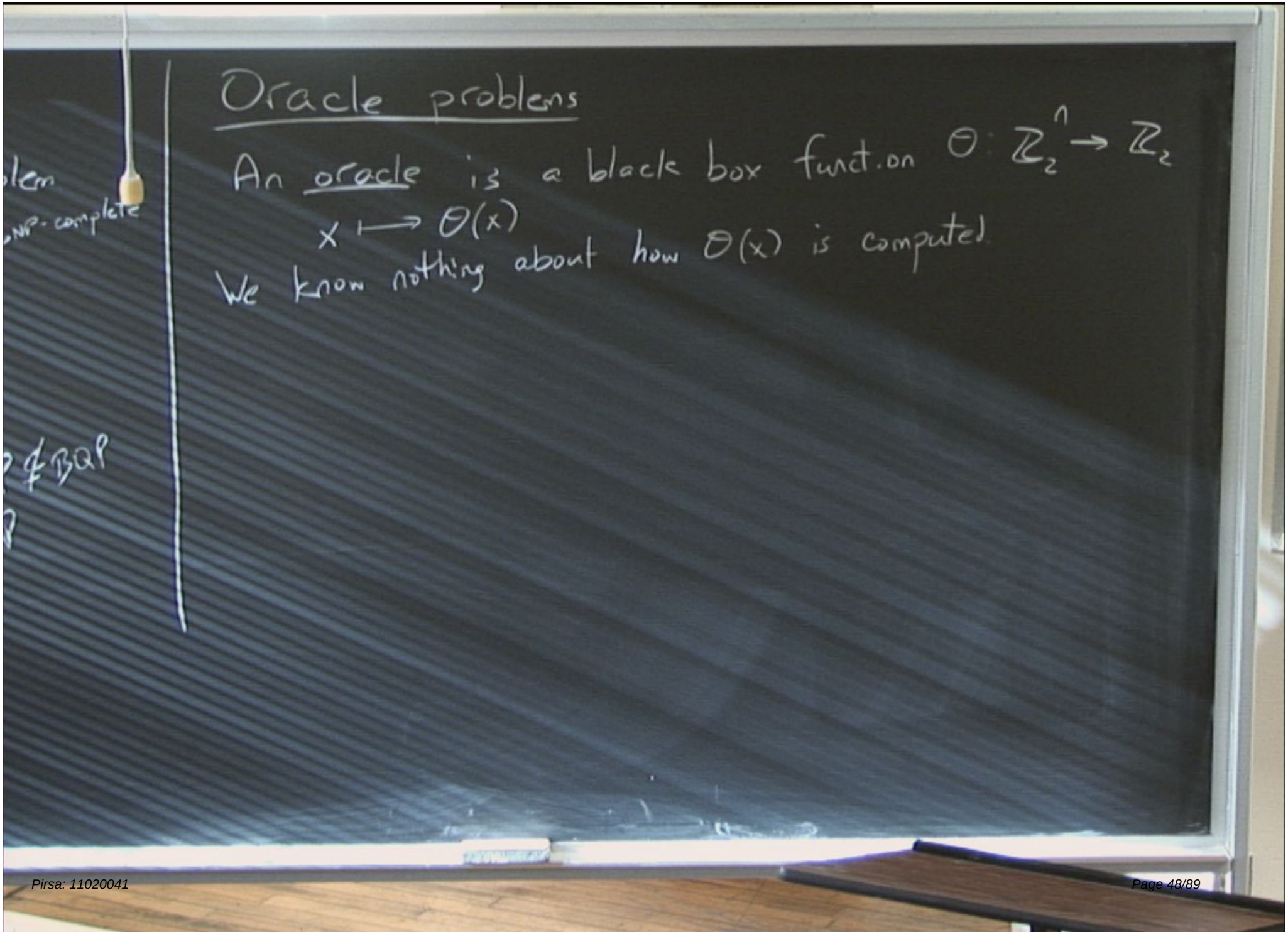

P ≠ NP ?
Famous open problem
NP - complete
NP
P
BQP

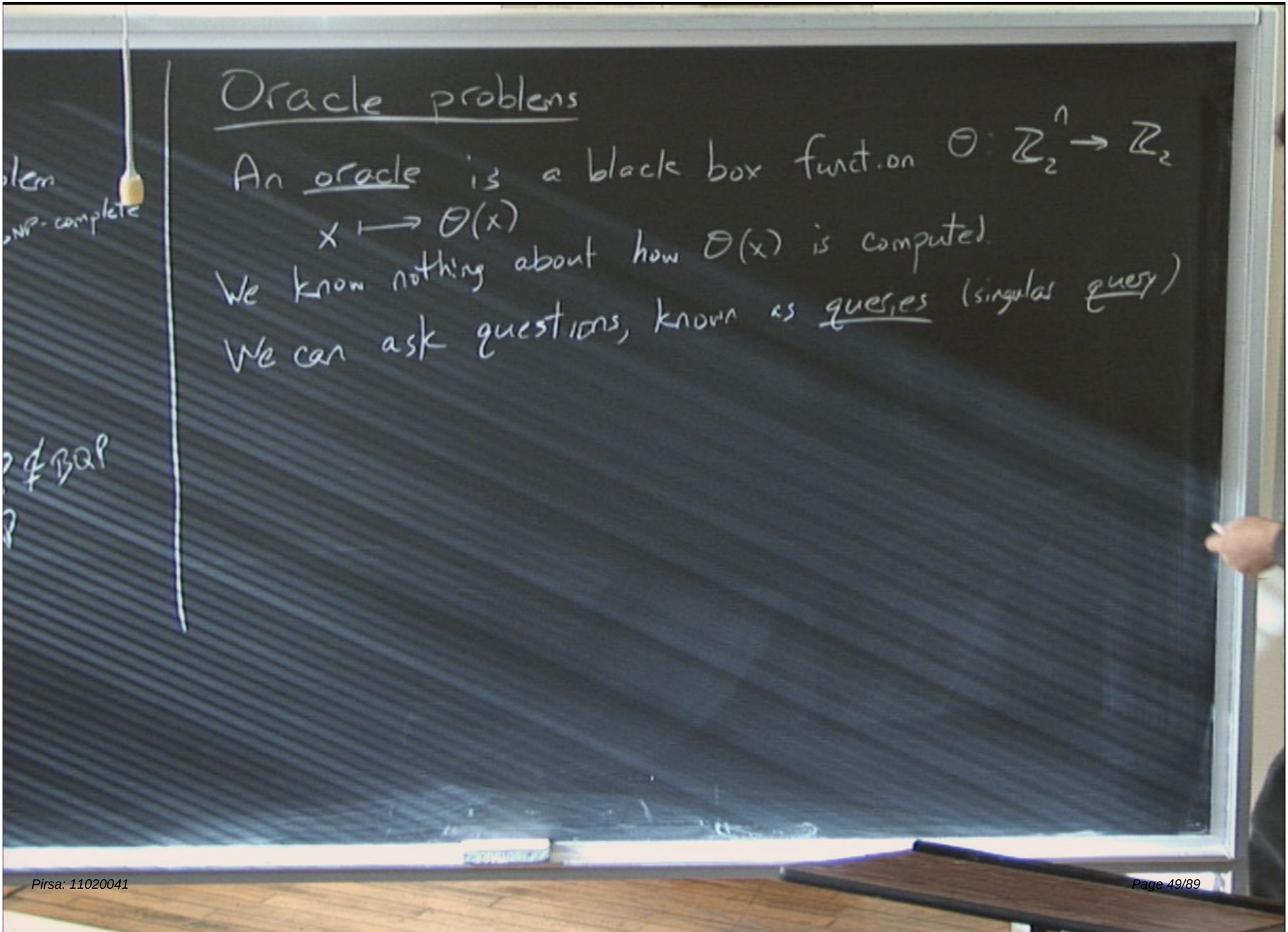

P ≠ NP ?
Famous open problem
NP-complete
NP
P
BQP
Cannot prove
NP ⊄ BQP
P ⊄ BQP, BQP ⊄ NP

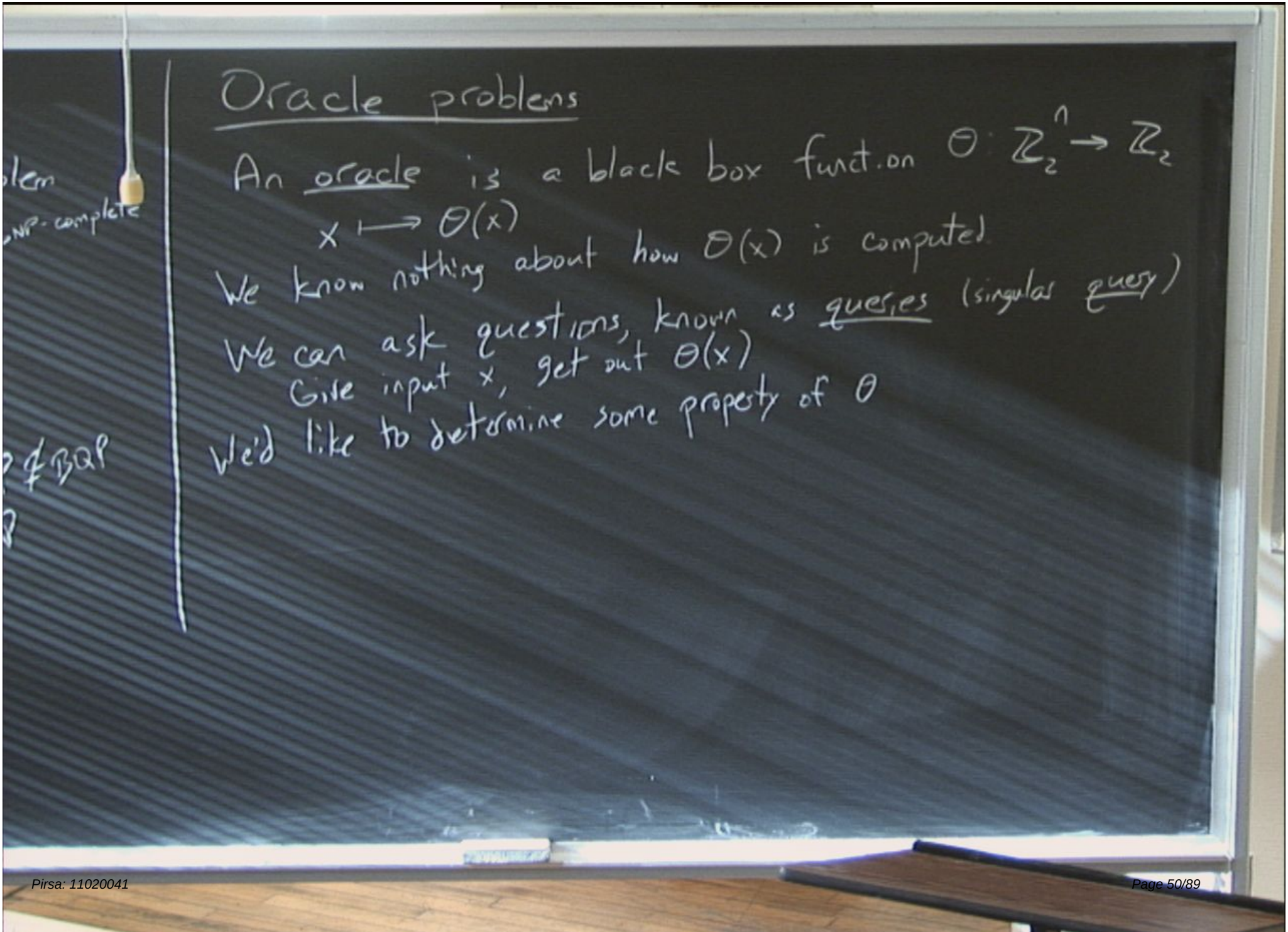

Oracle problems
problem
NP - complete
NP
prove
NP ≠ BQP
BQP ≠ NP

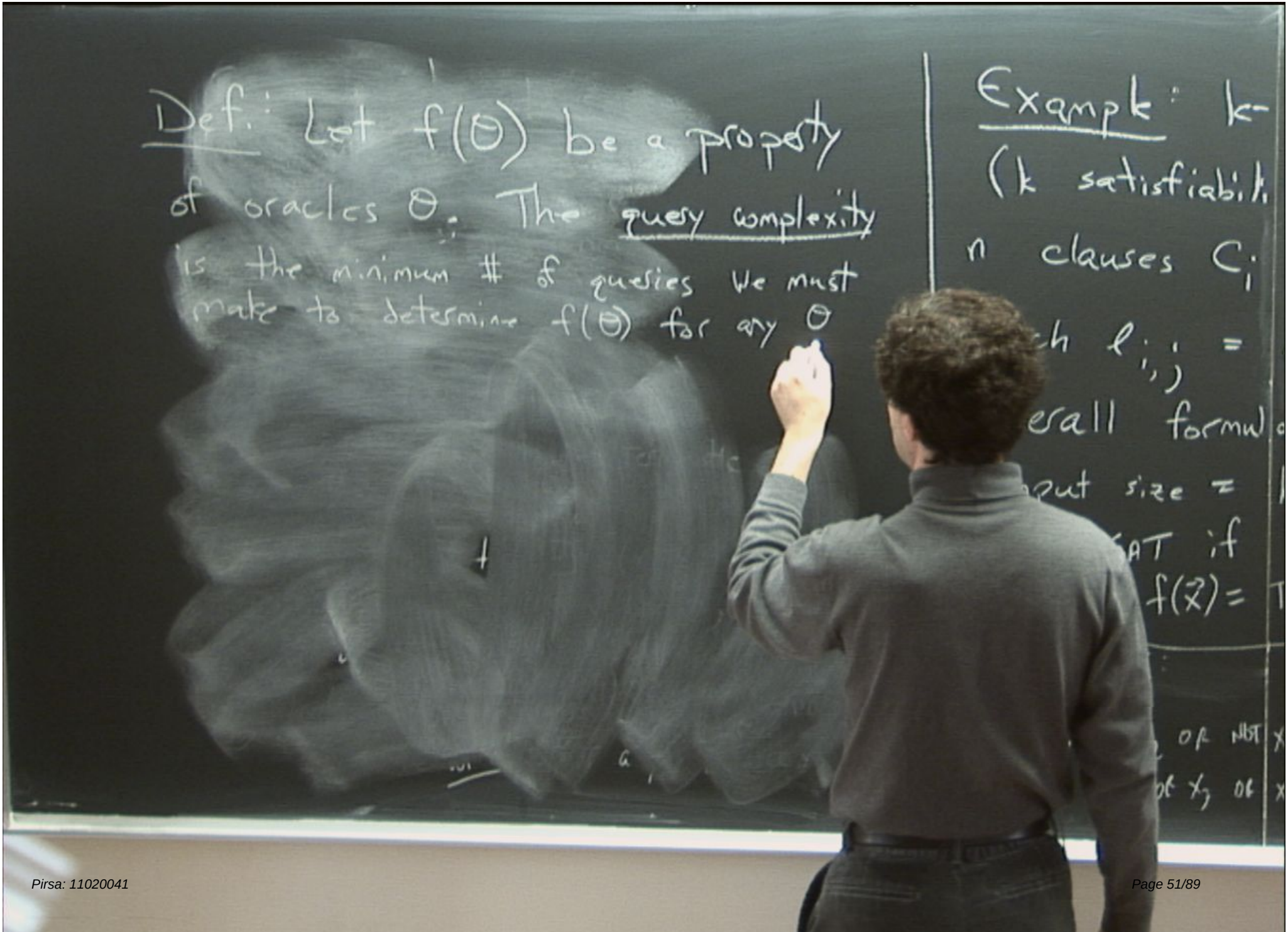

problem
NP-complete
NP
prove
NP ≠ BQP
BQP ≠ NP
Oracle problems
An oracle is a black box function

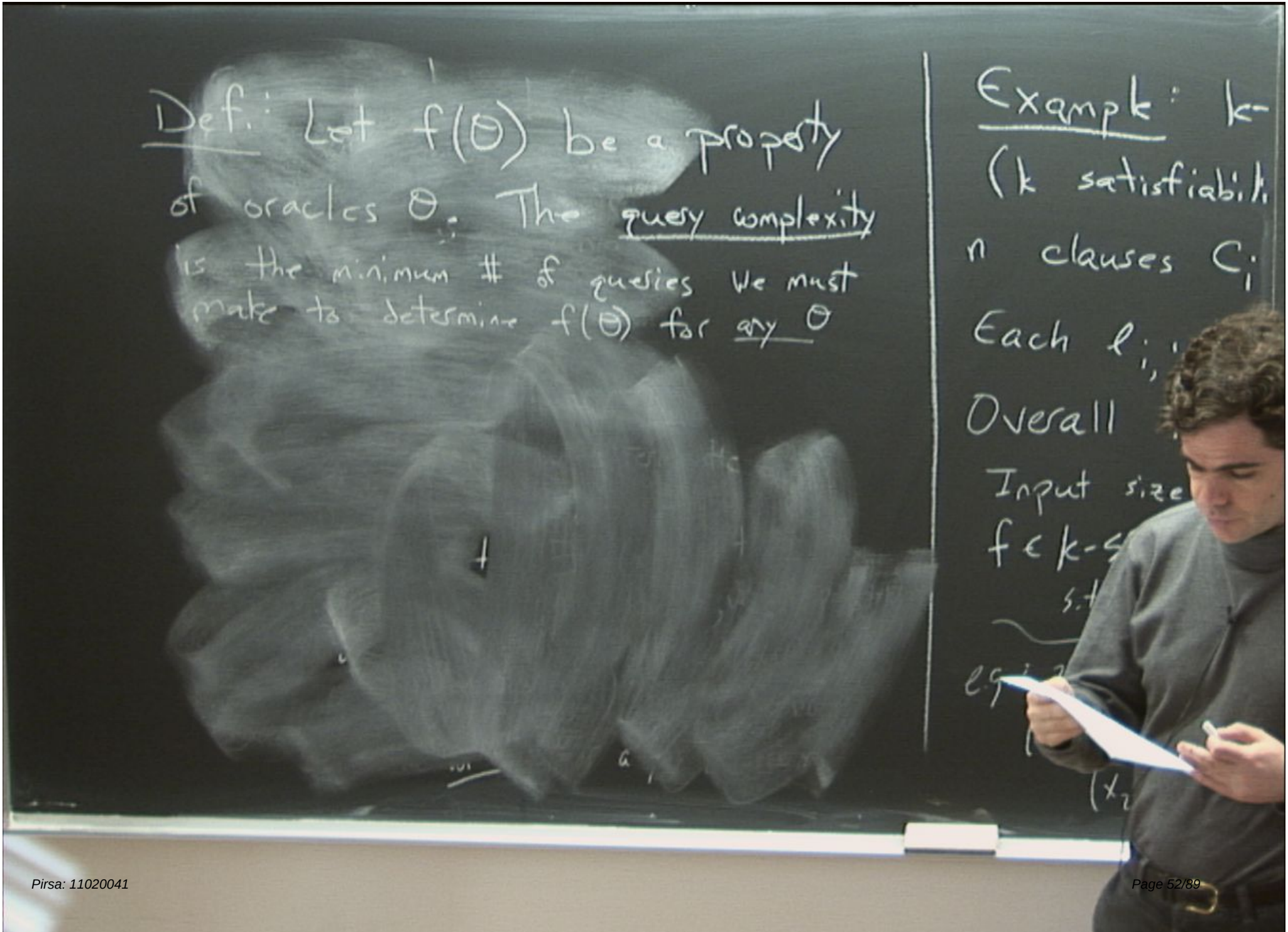

Oracle problems
An oracle is a black box function $\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$
$x \longmapsto \Theta(x)$
NP-complete
$\not\subset BQP$

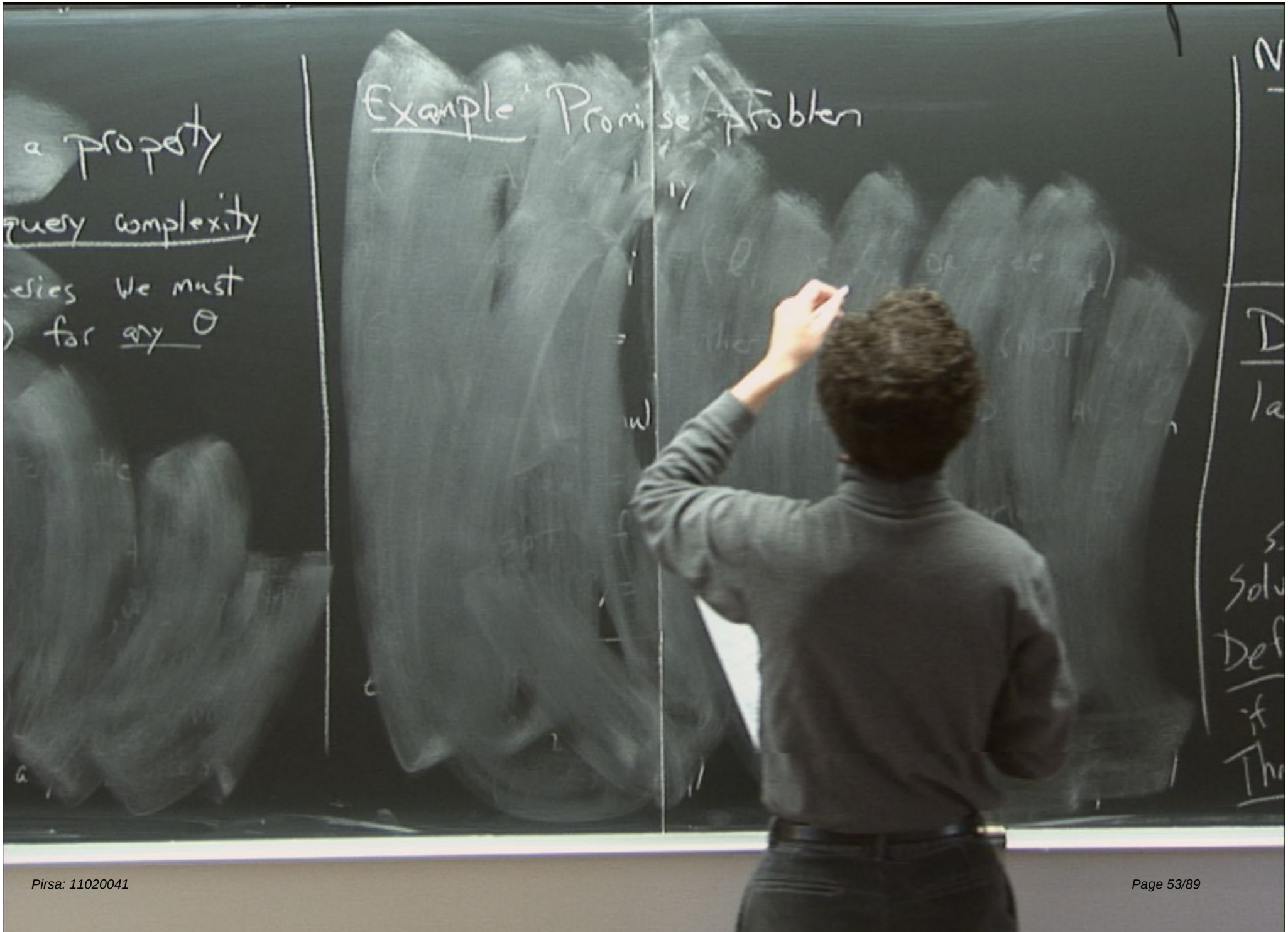

Oracle problems
An oracle is a black box function $\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$
$x \longmapsto \Theta(x)$
We know nothing about how $\Theta(x)$ is computed

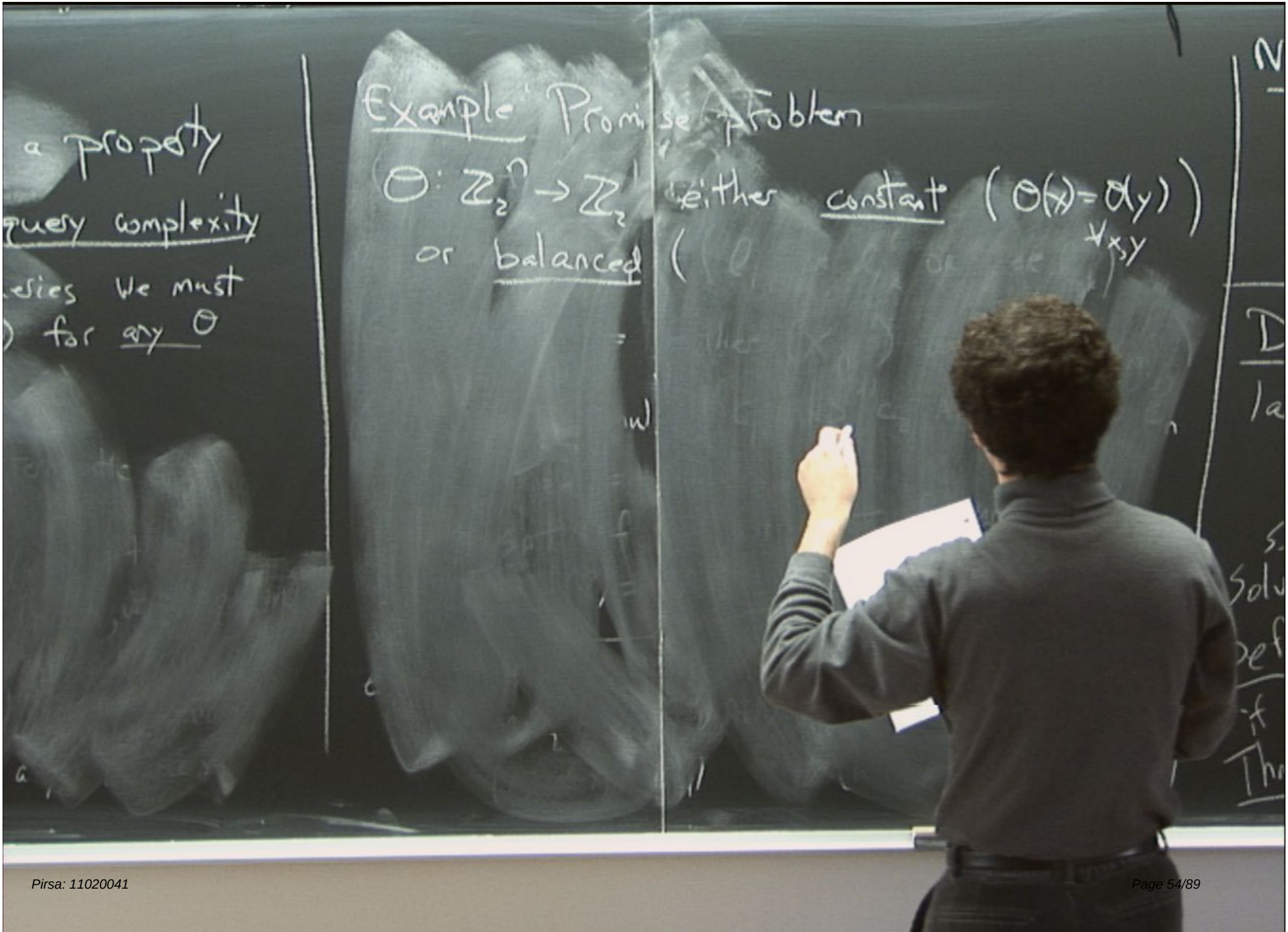

Oracle problems
An oracle is a black box function $\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$
$x \longmapsto \Theta(x)$
We know nothing about how $\Theta(x)$ is computed
We can ask questions, known as queries (singular query)

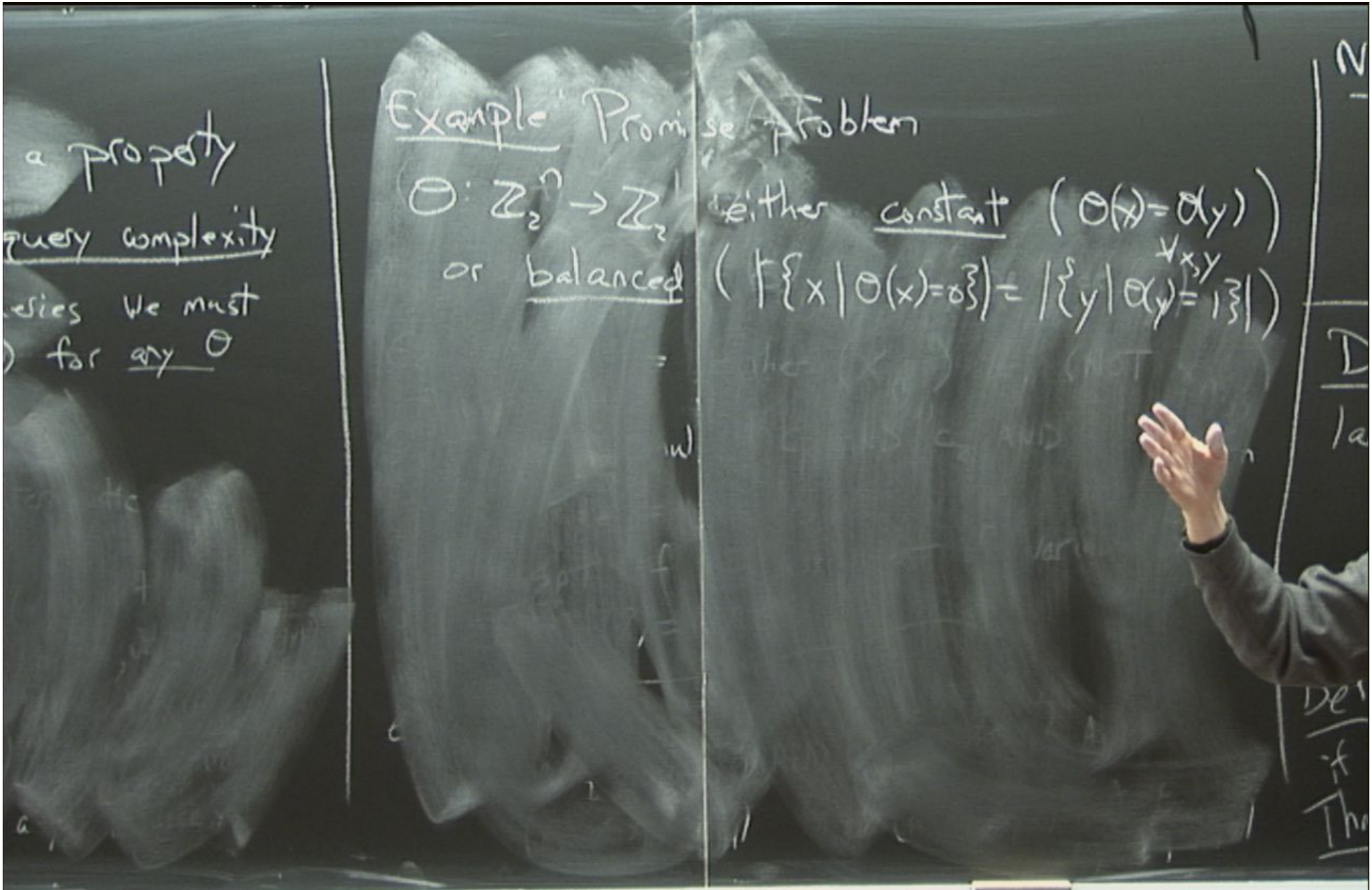

Oracle problems
An oracle is a black box function $\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$
$x \longmapsto \Theta(x)$
We know nothing about how $\Theta(x)$ is computed
We can ask questions, known as queries (singular query)
Give input x, get out $\Theta(x)$
We'd like to determine some property of Θ
NP-complete
$\not\subset$ BQP

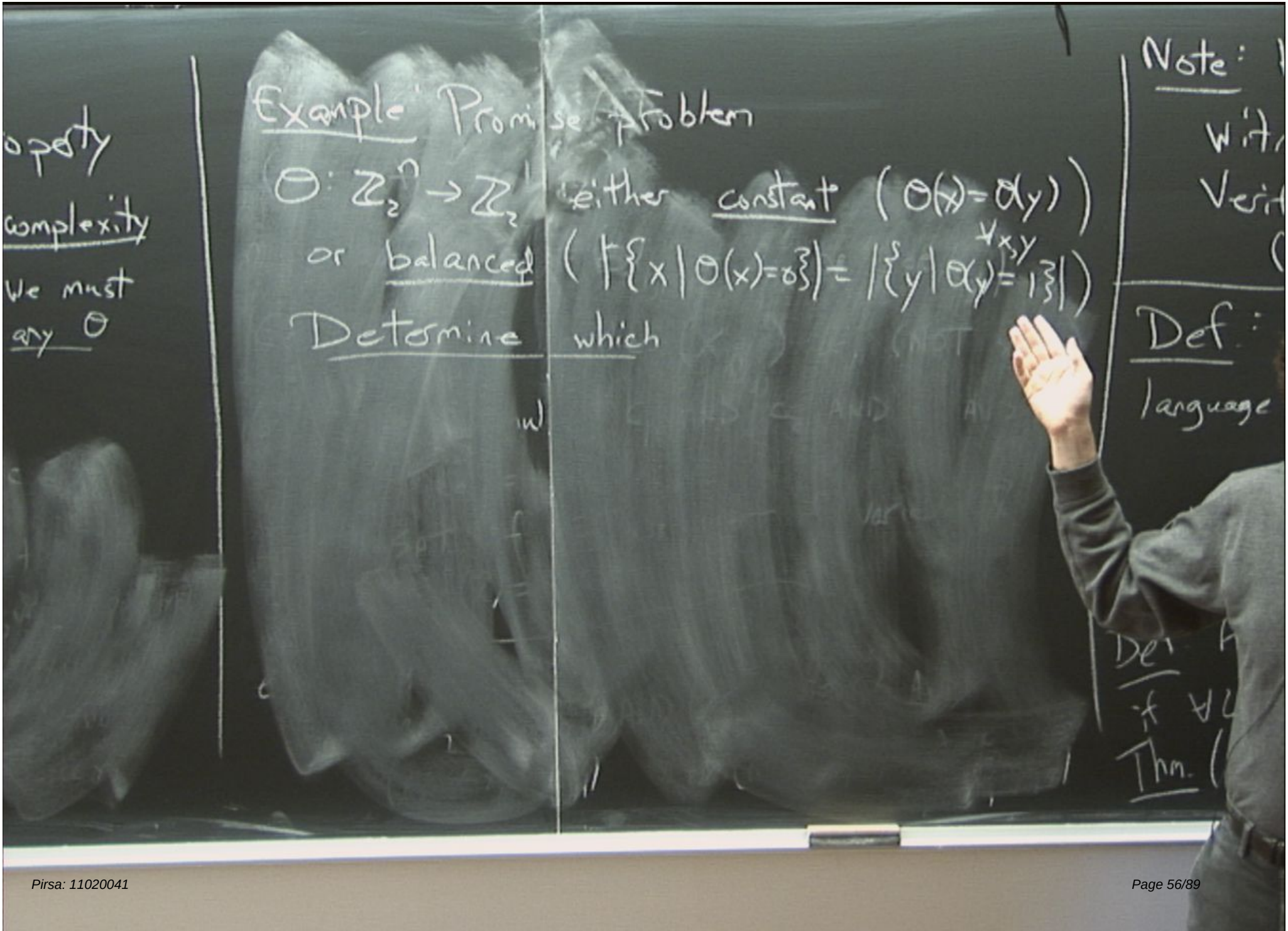

Def: Let f(O) be a property
of oracles O. The query complexity
is the minimum # of queries we must
make to determine f(O) for any O
Example: k-
(k satisfiabi
n clauses C
erall formu
put size =
AT ; if
f(x) =

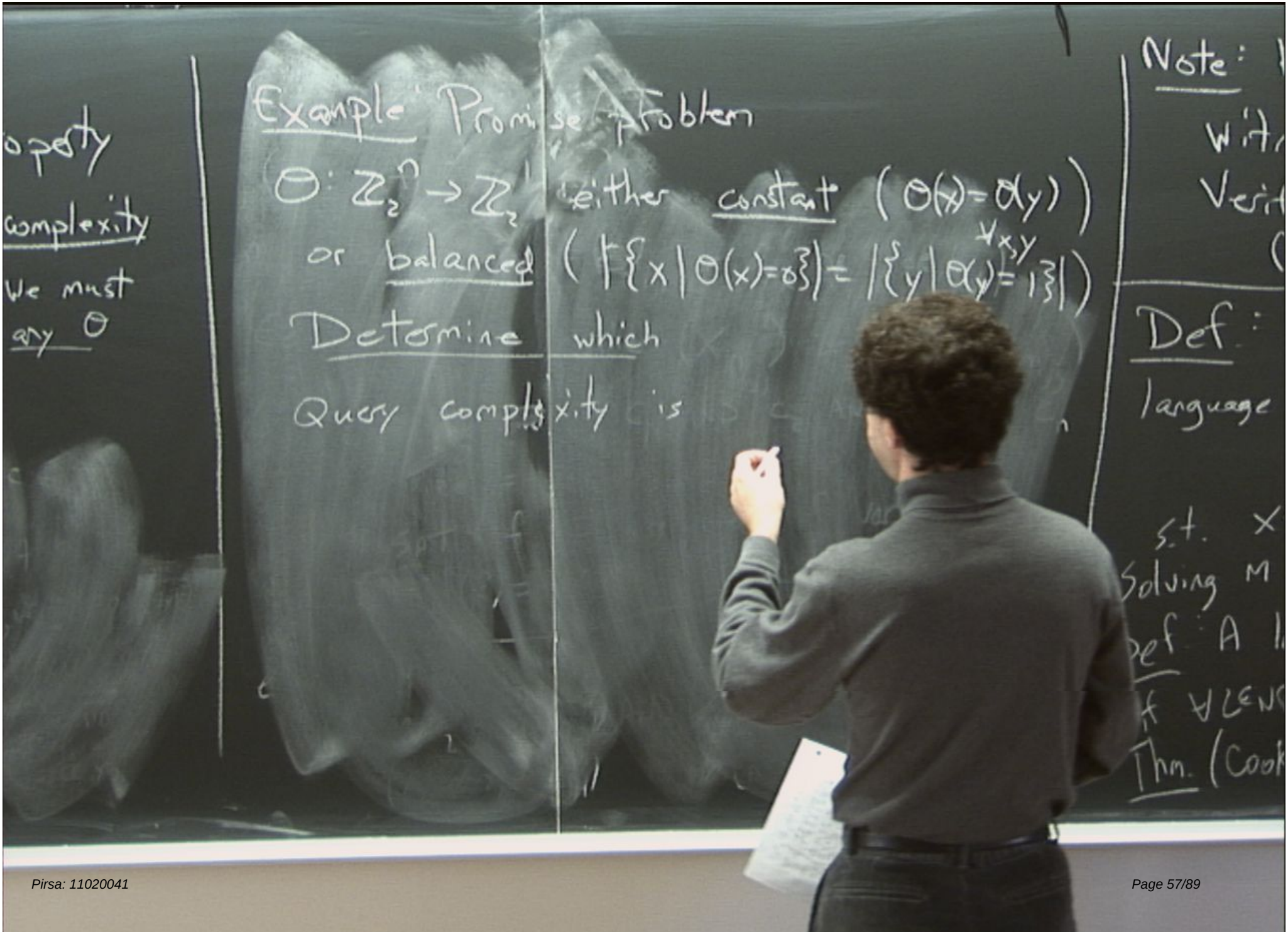

Def: Let f(O) be a property of oracles O. The query complexity is the minimum # of queries we must make to determine f(O) for any O
Example: k-
(k satisfiabi
n clauses C
Each
Overall
Input size
f ∈ k-S
s.t

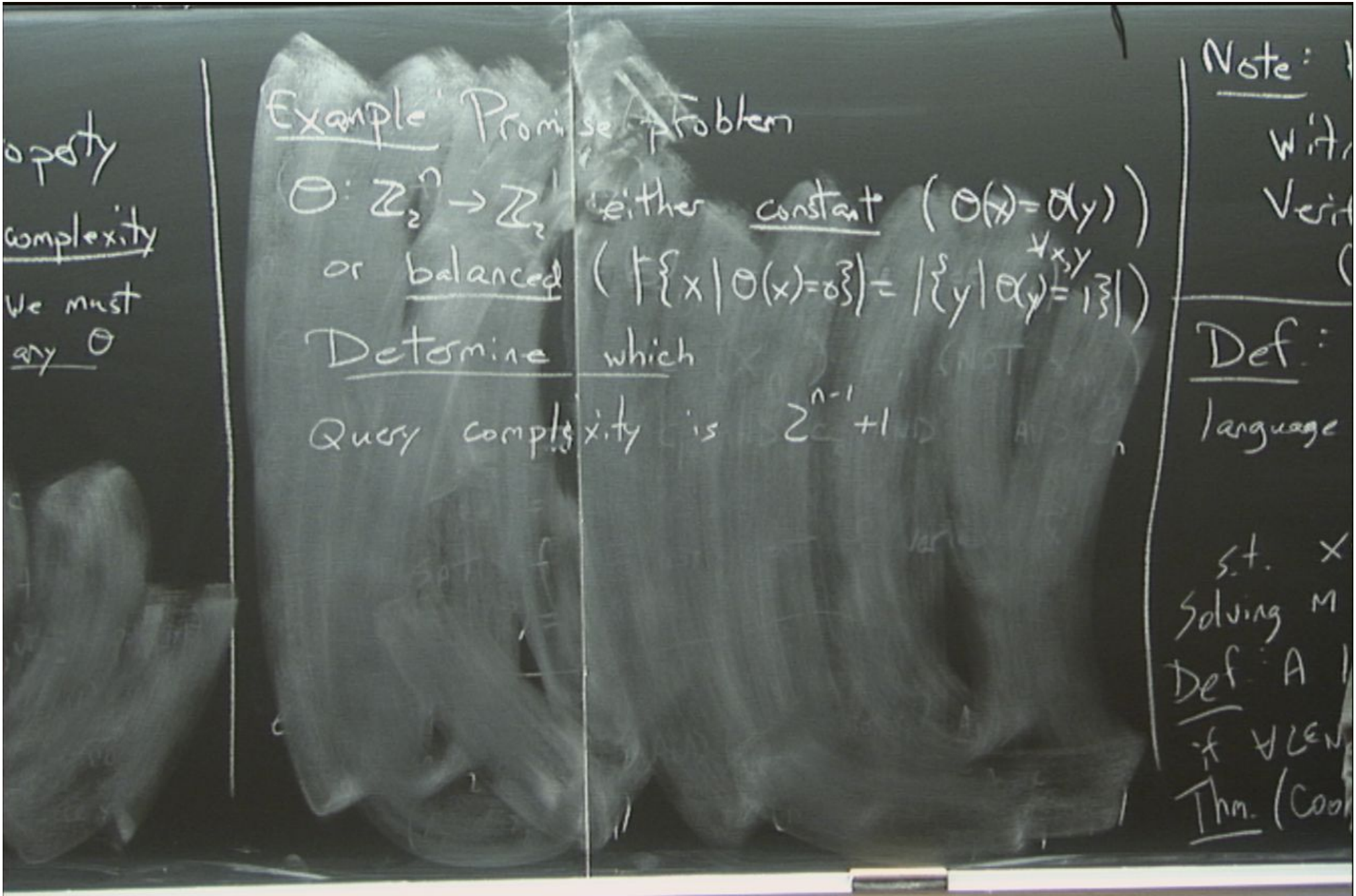

a property
query complexity
for any
Example Promise Problem

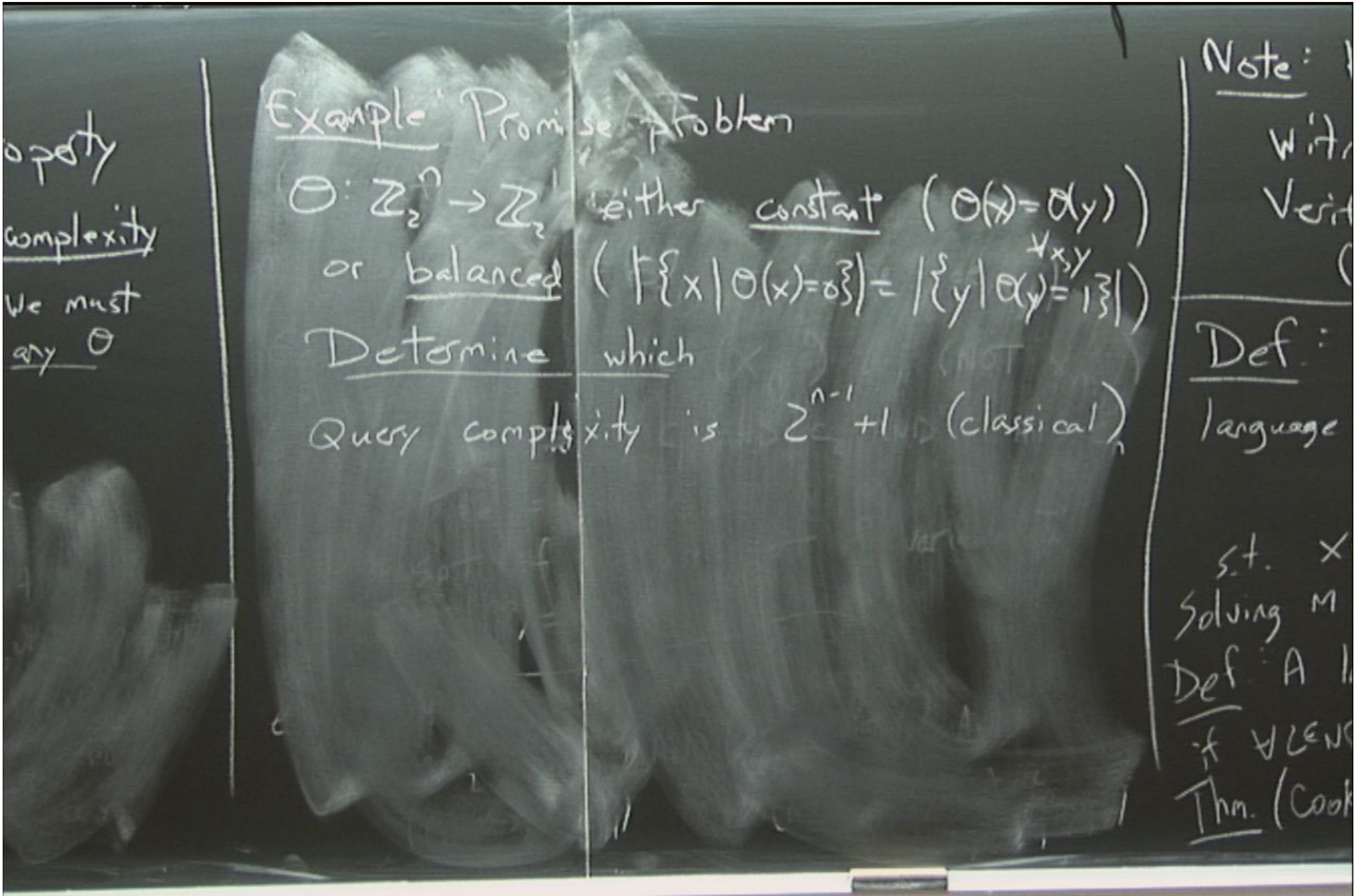

a property
query complexity
for any
Example Promise problem
either constant
or balanced

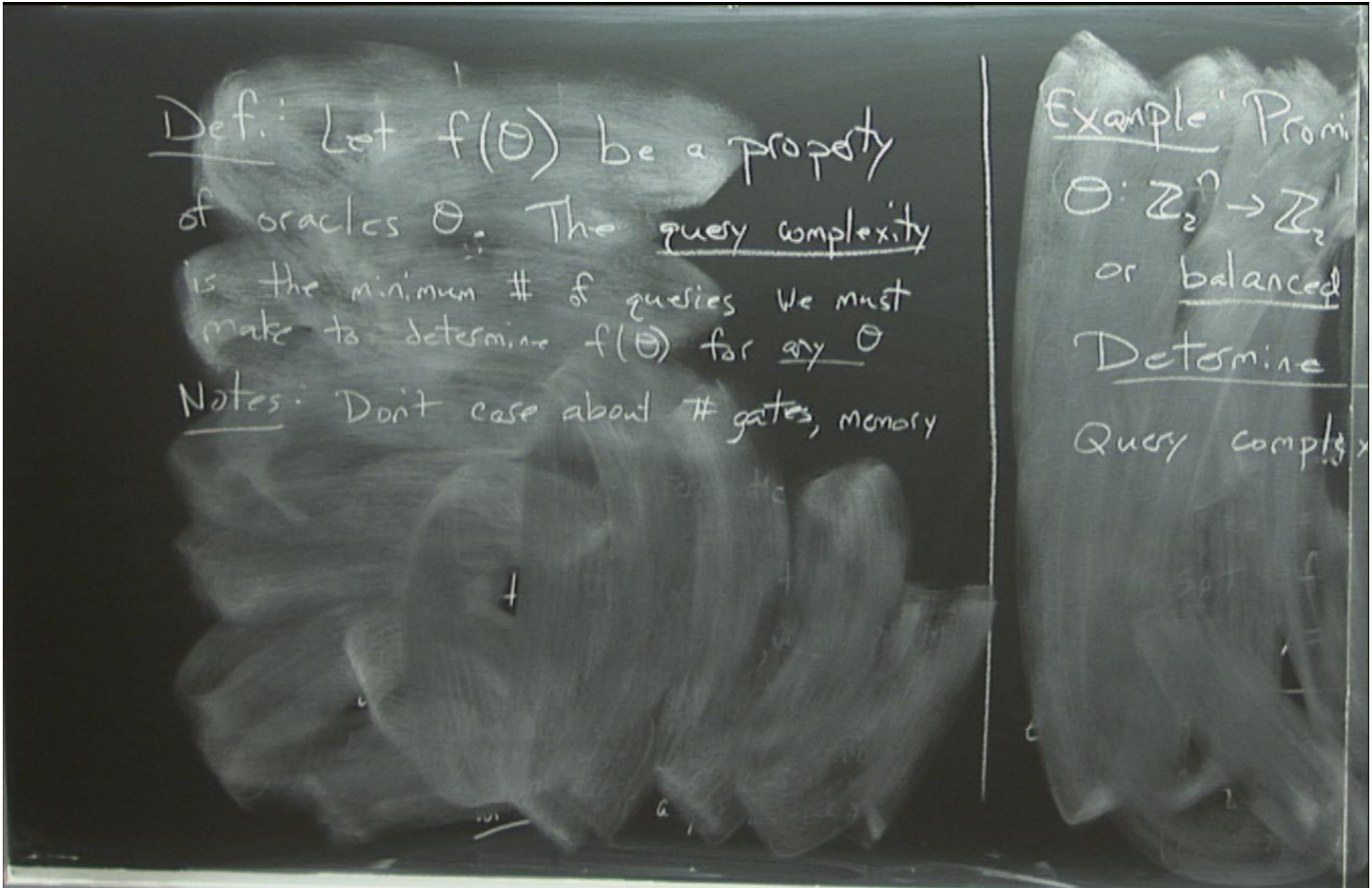

a property
query complexity
sies We must
for any Θ
Example Promise Problem
Θ: $\mathbb{Z}_2^n \to \mathbb{Z}_2$
either constant $(\Theta(x)=\Theta(y)\ \forall x,y)$
or balanced $(|\{x|\Theta(x)=0\}|=|\{y|\Theta(y)=1\}|)$

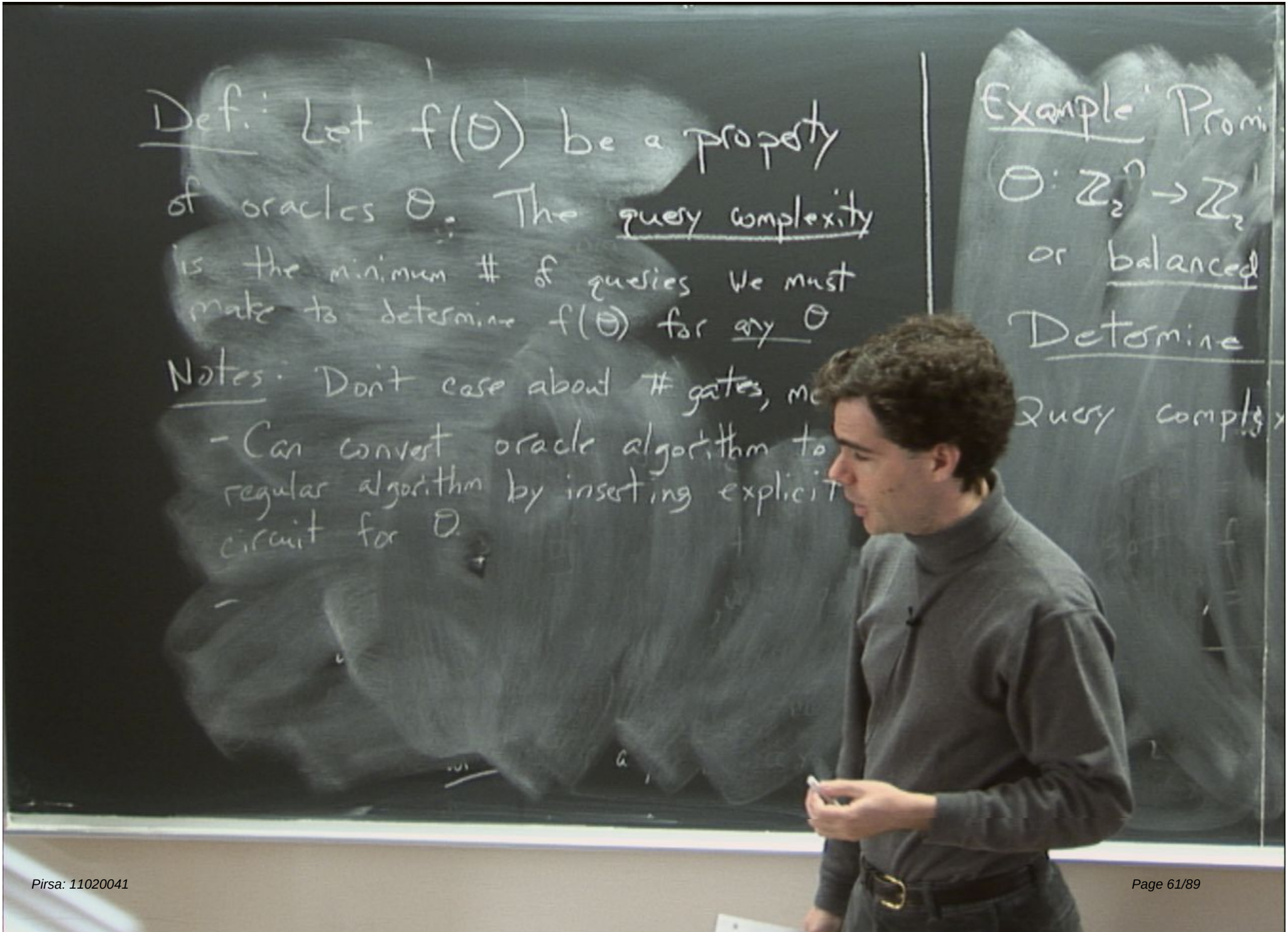

Example: Promise Problem
either constant ($\forall x,y\ O(x)=O(y)$)
or balanced ($|\{x \mid O(x)=0\}| = |\{y \mid O(y)=1\}|$)
Determine which
Note:
Def:
language
Thm.

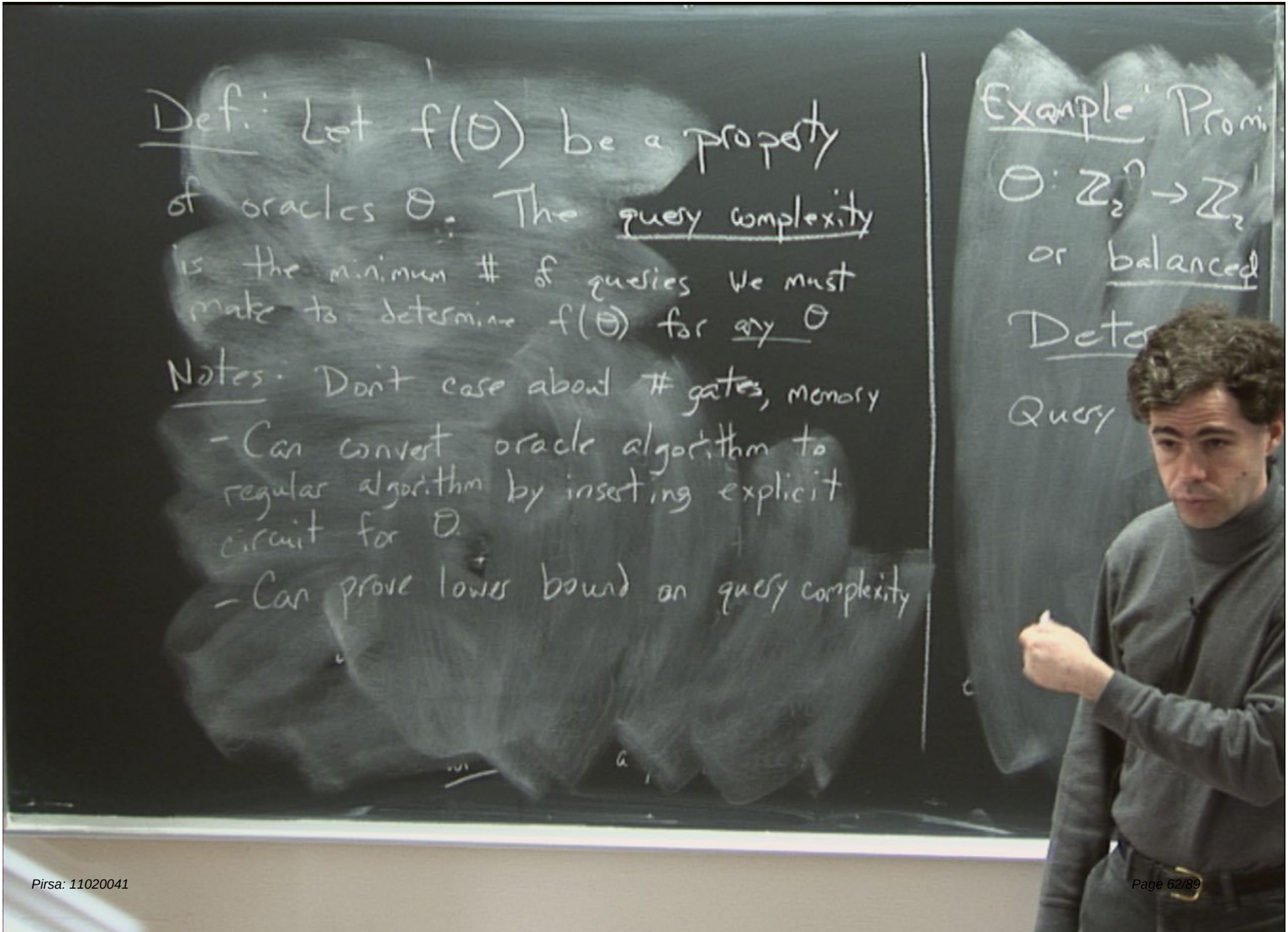

Example Promise Problem
either constant
or balanced
Determine which
Query complexity is
Note:
Def:
language
Thm.


Example Promise Problem
$\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$
either constant $(\Theta(x)=\Theta(y))$
or balanced $(|\{x \mid \Theta(x)=0\}| = |\{y \mid \Theta(y)=1\}|)$
$\forall_{x,y}$
Determine which
Query complexity is $2^{n-1}+1$
Note:
Def:
language


Example Promise problem
$\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant $(\Theta(x)=\Theta(y))$
or balanced $(|\{x|\Theta(x)=0\}| = |\{y|\Theta(y)=1\}|)$
$\forall_{x,y}$
Determine which
Query complexity is $2^{n-1}+1$ (classical)
Note:
Def:
language


Def: Let f(Θ) be a property of oracles Θ. The query complexity is the minimum # of queries we must make to determine f(Θ) for any Θ
Notes: Don't care about # gates, memory
Example: Promi
Θ: Z2^n → Z2
or balanced
Determine
Query complex


Def: Let f(Θ) be a property
of oracles Θ. The query complexity
is the minimum # of queries we must
make to determine f(Θ) for any Θ
Notes: Don't care about # gates, m
- Can convert oracle algorithm to
regular algorithm by inserting explicit
circuit for Θ.
Example: Promi
Θ: Z2^n → Z2
or balanced
Determine
Query complex


Def: Let f(Θ) be a property of oracles Θ. The query complexity is the minimum # of queries we must make to determine f(Θ) for any Θ
Notes: Don't care about # gates, memory
- Can convert oracle algorithm to regular algorithm by inserting explicit circuit for Θ.
- Can prove lower bound on query complexity
Example: Prom
Θ: Z2^n → Z2
or balanced
Dete
Query

Def: Let $f(\Theta)$ be a property of oracles Θ. The query complexity is the minimum # of queries we must make to determine $f(\Theta)$ for any Θ

Notes: Don't care about # gates, memory

- Can convert oracle algorithm to regular algorithm by inserting explicit circuit for Θ.
- Can prove lower bound on query complexity $\Rightarrow$ Does not imply lower bound on regular complexity – it only says any algorithm that beats lower bound must use internal structure of $\Theta(x)$.

Example: Prom

$\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$

or balanced

Determ:

Query c

Def: Let $f(\Theta)$ be a property of oracles Θ. The query complexity is the minimum # of queries we must make to determine $f(\Theta)$ for any Θ

Notes: Don't care about # gates, memory

- Can convert oracle algorithm to regular algorithm by inserting explicit circuit for Θ.
- Can prove lower bound on query complexity $\Rightarrow$ Does not imply lower bound on regular complexity – it only says any algorithm that beats lower bound must use internal structure of $\Theta(x)$.

Example: Prom

$\Theta: \mathbb{Z}_2^n \to \mathbb{Z}_2$

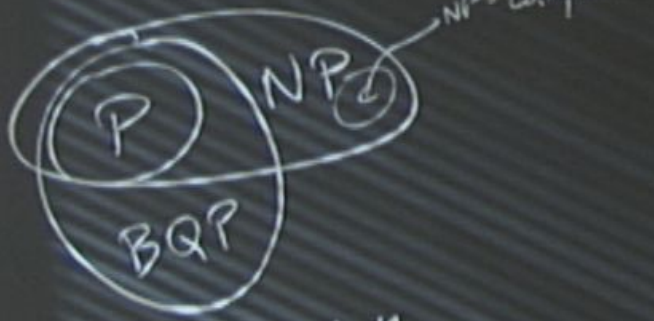
or balanced

Determine

Query complex

$P \neq NP$?

Famous open problem



Cannot prove
$P \neq BQP$, $NP \not\subseteq BQP$
$BQP \not\subseteq NP$

Oracle problems

An oracle is a black box funct

$x \longmapsto O(x)$

We know nothing about how $O(x)$ is co

We can ask questions, known as
Give input x, get out $O(x)$

We'd like to determine some propert

Example: Promise problem

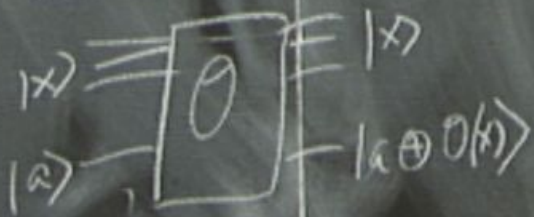
$O: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant ($O(x)=O(y)$ $\forall x,y$)

or balanced ($|\{x \mid O(x)=0\}| = |\{y \mid O(y)=1\}|$)

Determine which

Query complexity is $2^{n-1}+1$ (classical)

For quantum oracle, make oracle reversible:



Note: k-SA

Witness

Verifier

(runs

Def

language

Example: Promise problem

$O: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant $(O(x)=O(y)\ \forall x,y)$

or balanced $(|\{x \mid O(x)=0\}| = |\{y \mid O(y)=1\}|)$

Determine which

Query complexity is $2^{n-1}+1$ (classical)

Note: k-SA

Witness

Verifier

(runs

f: Lan

age M

For quantum oracle, make oracle reversible:

Superpositions wo

Example: Promise problem

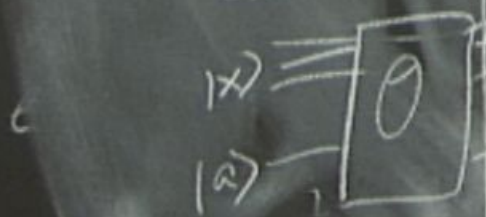
$f: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant ($f(x)=f(y)$ $\forall x,y$)

or balanced ($|\{x \mid f(x)=0\}| = |\{y \mid f(y)=1\}|$)

Determine which

Query complexity is $2^{n-1}+1$ (classical)

For quantum oracle, make oracle reversible:



Superpositions work by linearity

$(|x\rangle + |x'\rangle)|0\rangle \to |x\rangle|O(x)\rangle + |x'\rangle|O(x')\rangle$

Note: k-SA

Witness

Verifier

(runs

Def: Lan

language M

s.t. $x \in L$

Solving M allow

Def: A langua

if $\forall L \in NP$, L

Thm. (Cook-Levi

Example: Promise problem

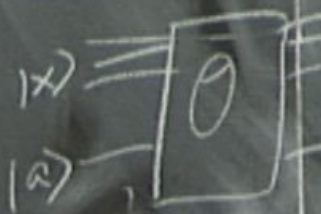
$O: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant ($O(x) = O(y)$ $\forall x,y$)

or balanced ($|\{x \mid O(x)=0\}| = |\{y \mid O(y)=1\}|$)

Determine which

Query complexity is $2^{n-1}+1$ (classical)

For quantum oracle, make oracle reversible:



Superpositions work by linearity

$(|x\rangle + |x'\rangle)|0\rangle \to |x\rangle|O(x)\rangle + |x'\rangle|O(x')\rangle$

Example: Promise problem

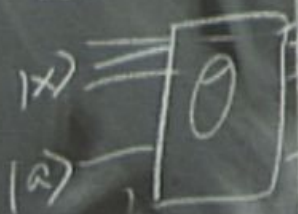
$f: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant ($f(x)=f(y)$ $\forall x,y$)

or balanced ($|\{x \mid f(x)=0\}| = |\{y \mid f(y)=1\}|$)

Determine which

Query complexity is $2^{n-1}+1$ (classical)

For quantum oracle, make oracle reversible:



Superpositions work by linearity.

$(|x\rangle + |x'\rangle)|0\rangle \to |x\rangle|f(x)\rangle + |x'\rangle|f(x')\rangle$

Example: Promise problem

$f: \mathbb{Z}_2^n \to \mathbb{Z}_2$ either constant ($f(x) = f(y)$ $\forall x,y$)

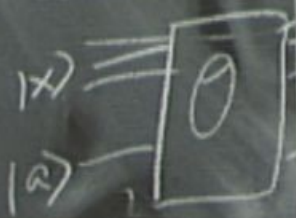
or balanced ($|\{x \mid f(x)=0\}| = |\{y \mid f(y)=1\}|$)

Determine which

Query complexity is $2^{n-1}+1$ (classical)

Quantum query complexity is ≤ 2

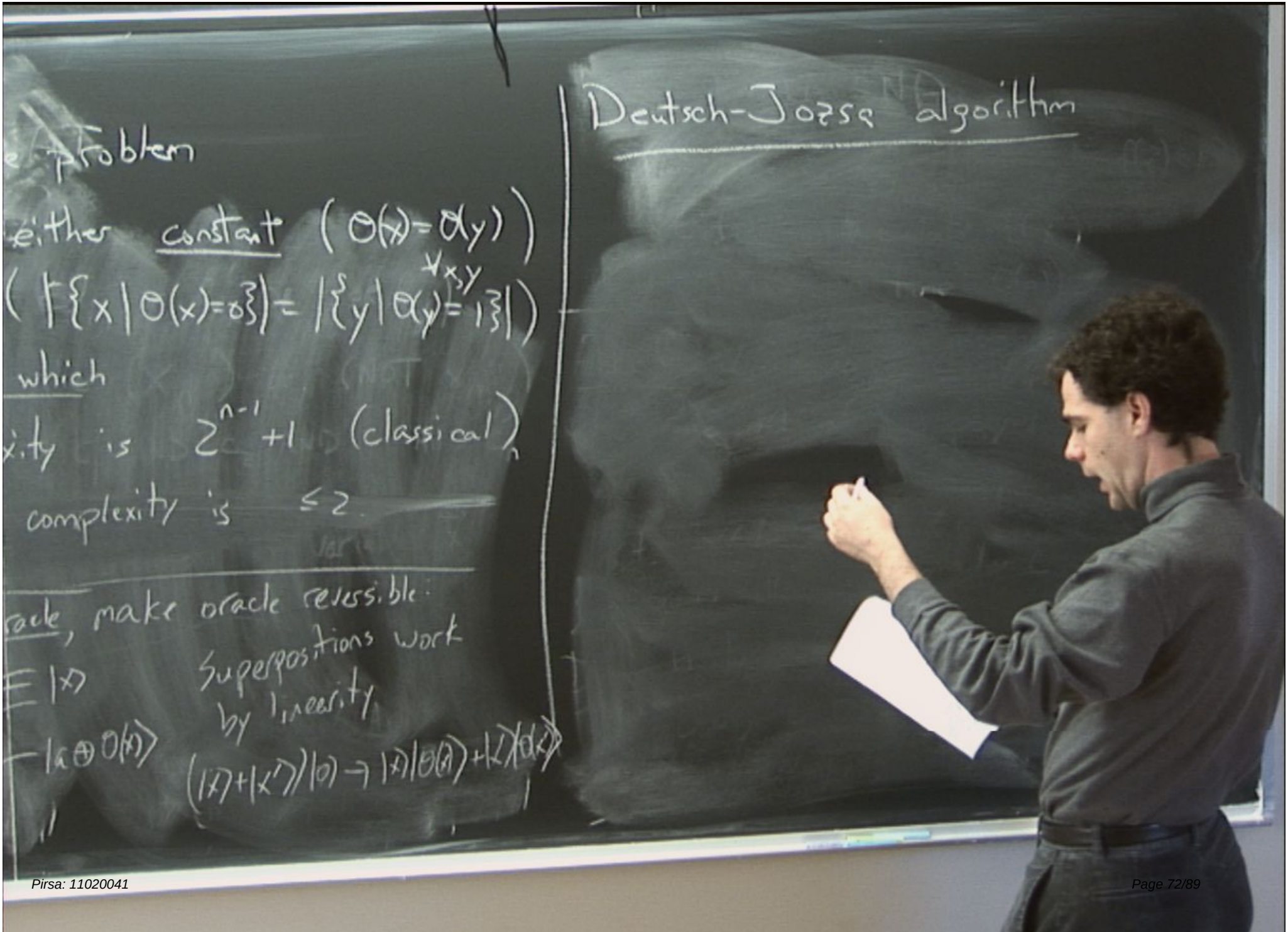
For quantum oracle, make oracle reversible:

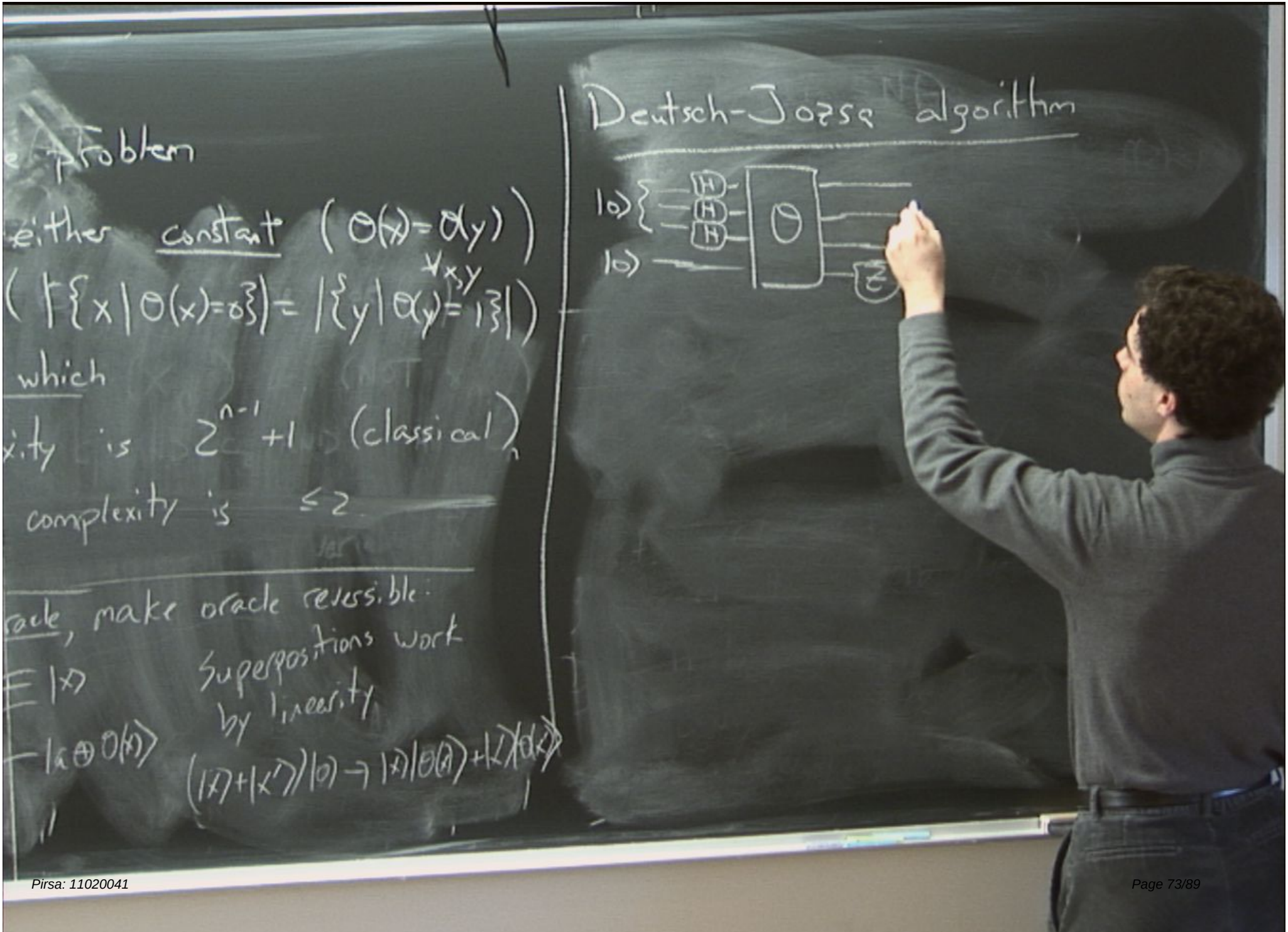


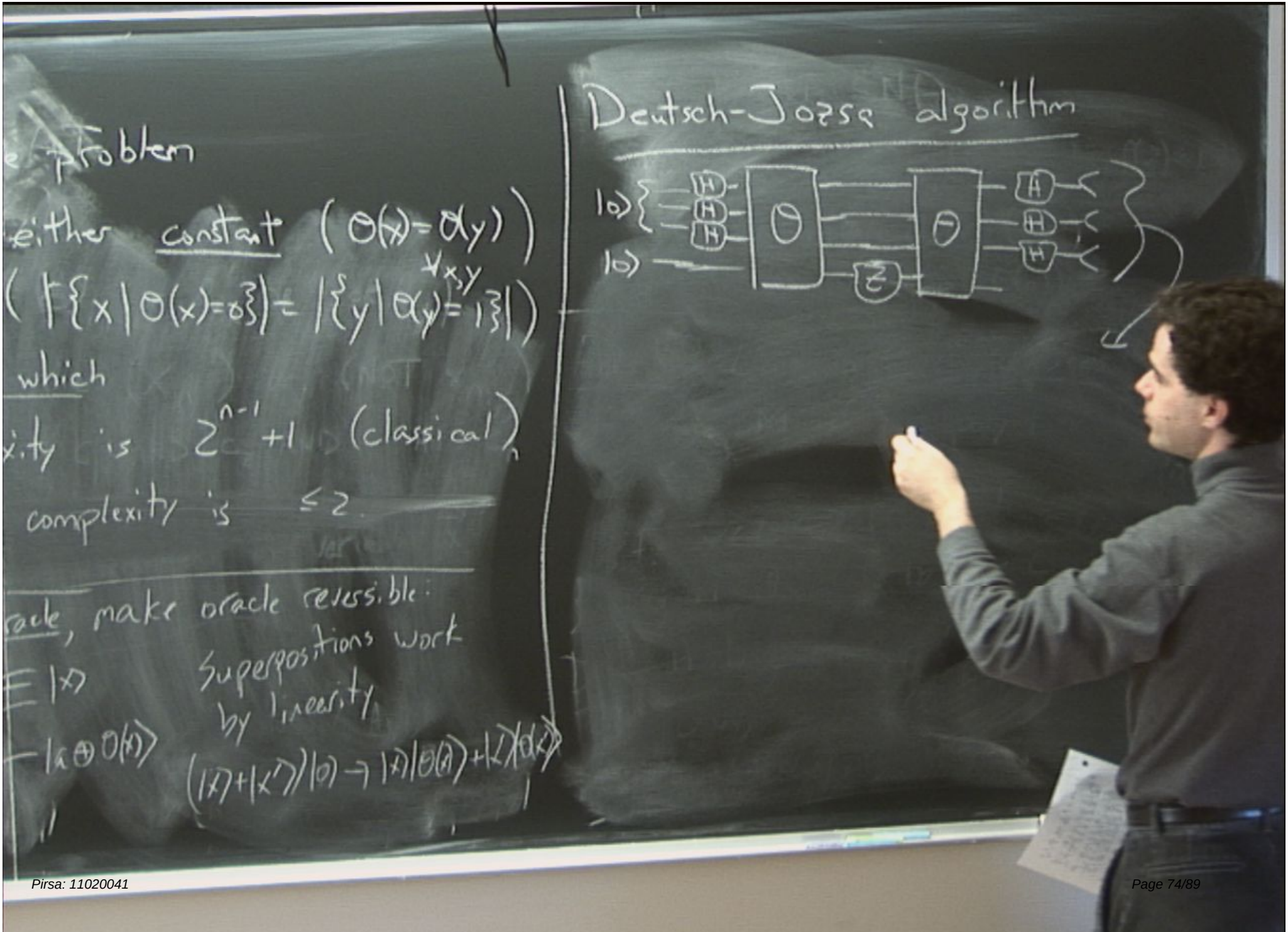
Superpositions work by linearity

$(|x\rangle + |x'\rangle)|0\rangle \to |x\rangle|f(x)\rangle + |x'\rangle|f(x')\rangle$

Deutsch


Deutsch-Josza algorithm
problem
either constant $(f(x)=f(y)\ \forall x,y)$
$(|\{x|f(x)=0\}| = |\{y|f(y)=1\}|)$
which
$2^{n-1}+1$ (classical)
complexity is
≤ 2
oracle, make oracle reversible:
Superpositions work
by linearity
$|x\rangle$
$|a \oplus f(x)\rangle$
$(|x\rangle+|x'\rangle)|0\rangle \rightarrow |x\rangle|f(x)\rangle+|x'\rangle|f(x')\rangle$


Problem
either constant (O(x)=O(y))
∀x,y
(|{x | O(x)=0}| = |{y | O(y)=1}|)
which
ity is 2^{n-1}+1 (classical),
complexity is ≤2
acle, make oracle reversible.
Superpositions work
by linearity
|x⟩
|a⊕O(x)⟩
(|x⟩+|x'⟩)|0⟩ → |x⟩|O(x)⟩+|x'⟩|O(x')⟩
Deutsch-Josza algorithm
|0⟩
|0⟩
H
H
H
O
H


Deutsch-Jozsa algorithm
problem
either constant
which
complexity is
complexity is
oracle, make oracle reversible:
Superpositions work
by linearity

e problem

either constant $(f(x)=f(y)\ \forall x,y)$

$(|\{x \mid f(x)=0\}| = |\{y \mid f(y)=1\}|)$

which

xity is $2^{n-1}+1$ (classical),

complexity is ≤ 2

racle, make oracle reversible:

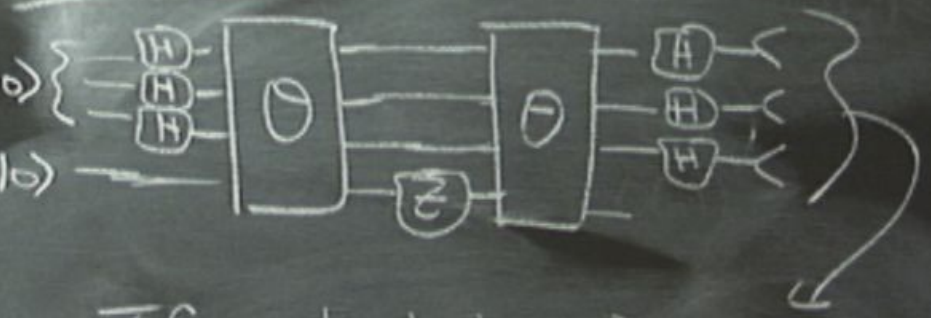
$|x\rangle$

$|a \oplus f(x)\rangle$

Superpositions work by linearity

$(|x\rangle+|x'\rangle)|0\rangle \to |x\rangle|f(x)\rangle+|x'\rangle|f(x')\rangle$

Deutsch-Jozsa algorithm



If output = |00⟩ ⟶

oblem

constant $(\forall_{x,y}\ f(x)=f(y))$

$|\{x \mid f(x)=0\}| = |\{y \mid f(y)=1\}|)$

is $2^{n-1}+1$ (classical)

lexity is ≤ 2

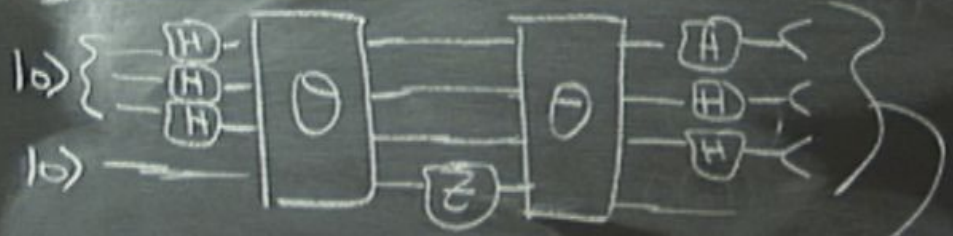
make oracle reversible:

Superpositions work by linearity

$|x\rangle$ $f(x)$

$(|x\rangle+|x'\rangle)|0\rangle \to |x\rangle|f(x)\rangle+|x'\rangle|f(x')\rangle$

Deutsch-Jozsa algorithm



If output = |00...0⟩ ⇒ constant

output ≠ |00...0⟩ ⇒ balanced.

oblem

constant $(\Theta(x)=\Theta(y))$

$\forall x,y$

$|\{x|\Theta(x)=0\}| = |\{y|\Theta(y)=1\}|)$

is $2^{n-1}+1$ (classical)

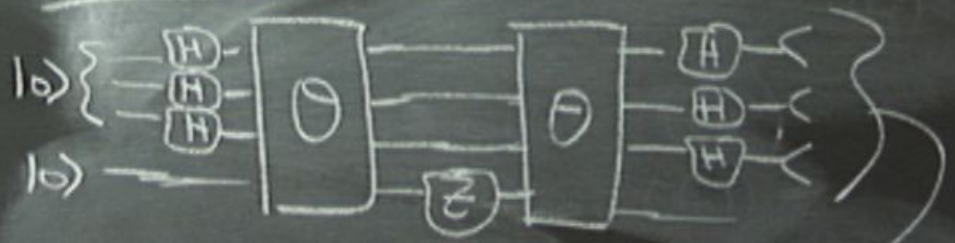
lexity is ≤ 2

make oracle reversible:

Superpositions work by linearity

$O(|x\rangle)$ $(|x\rangle+|x'\rangle)|0\rangle \rightarrow |x\rangle|\Theta(x)\rangle+|x'\rangle|\Theta(x')\rangle$

Deutsch-Jozsa algorithm



If output = $|00\ldots0\rangle$ → constant

output ≠ $|00\ldots0\rangle$ → balanced.

e problem

either constant $(O(x)=O(y)\ \forall x,y)$

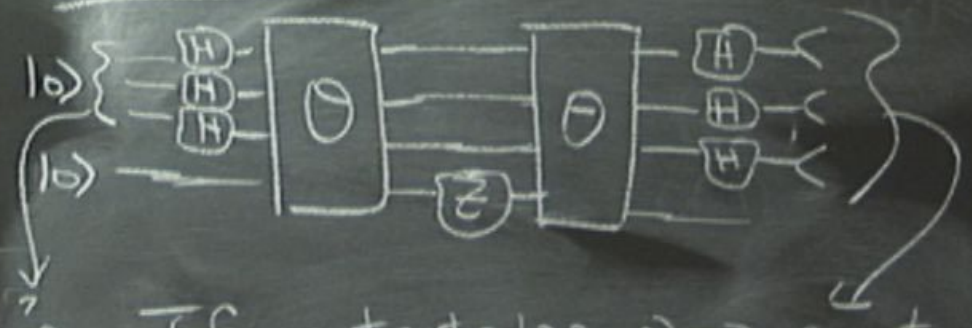
$\left(|\{x \mid O(x)=0\}| = |\{y \mid O(y)=1\}|\right)$

which

$\ldots$ity is $2^{n-1}+1$ (classical),

complexity is ≤ 2

Deutsch-Jozsa algorithm



If output = |00⟩ → constant
output ≠ |00⟩ → balanced.

scale, make oracle reversible:

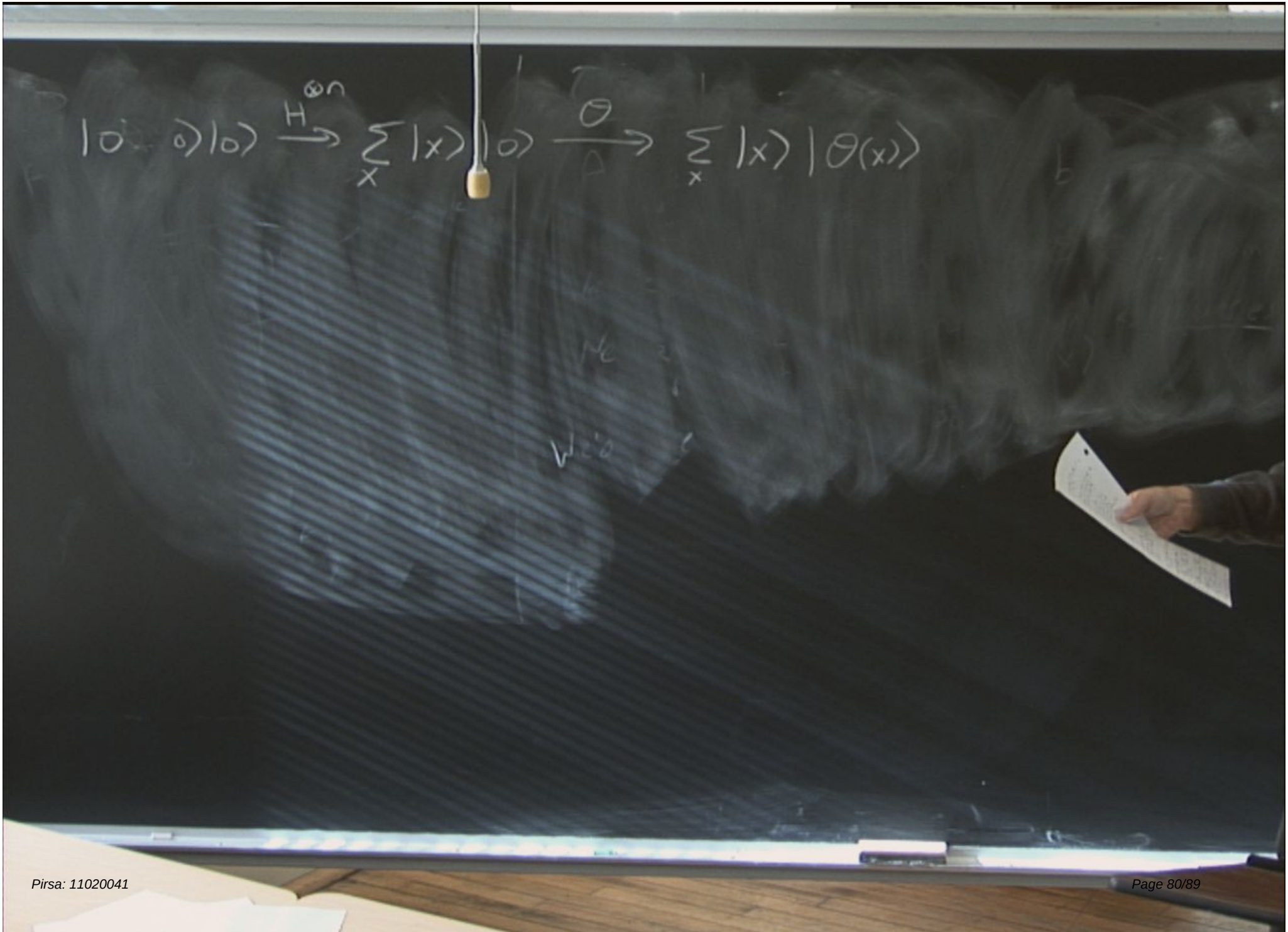
$|x\rangle$

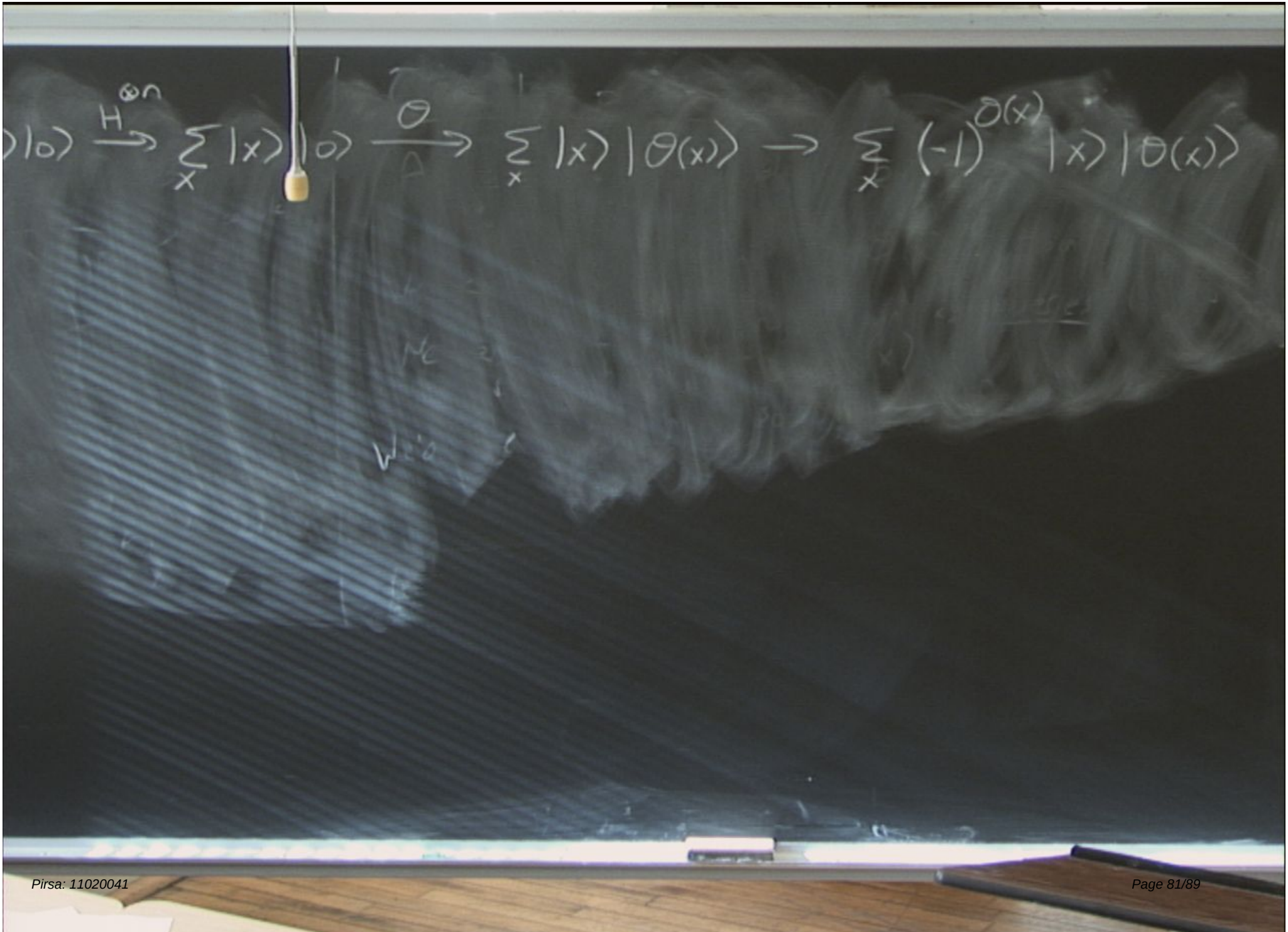
$|a \oplus O(x)\rangle$

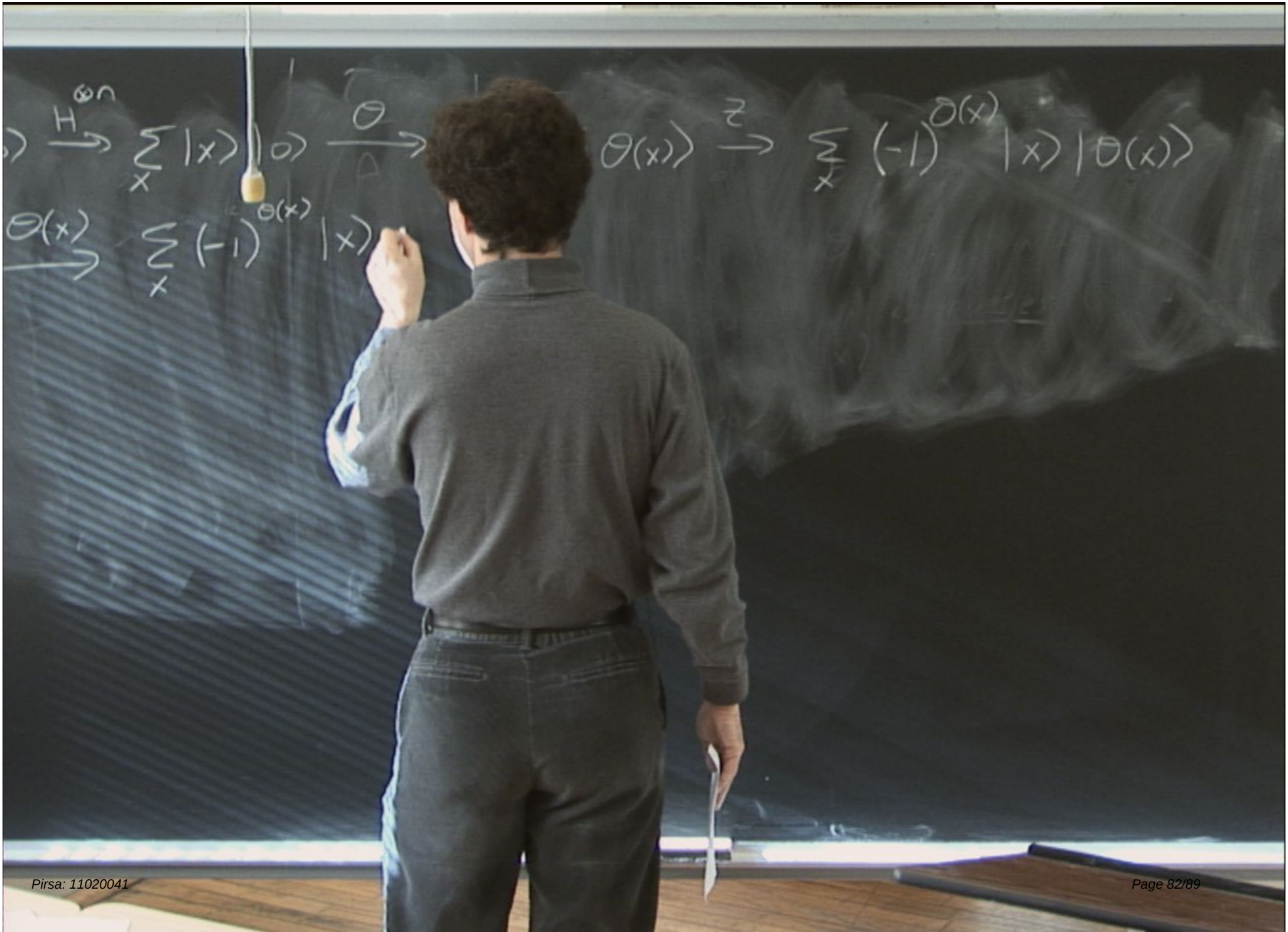
Superpositions work by linearity

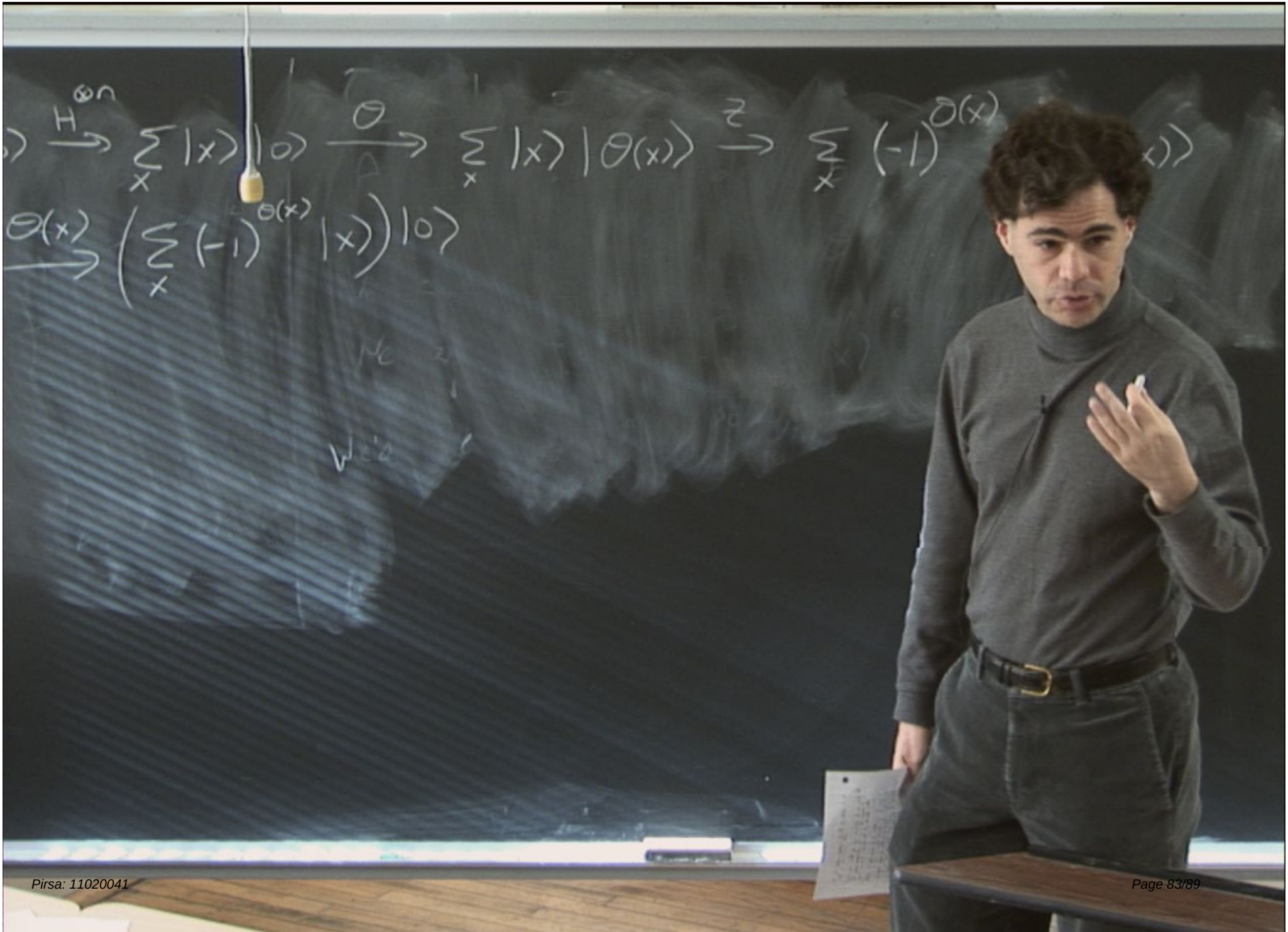
$(|x\rangle+|x'\rangle)|0\rangle \rightarrow |x\rangle|O(x)\rangle+|x'\rangle|O(x')\rangle$

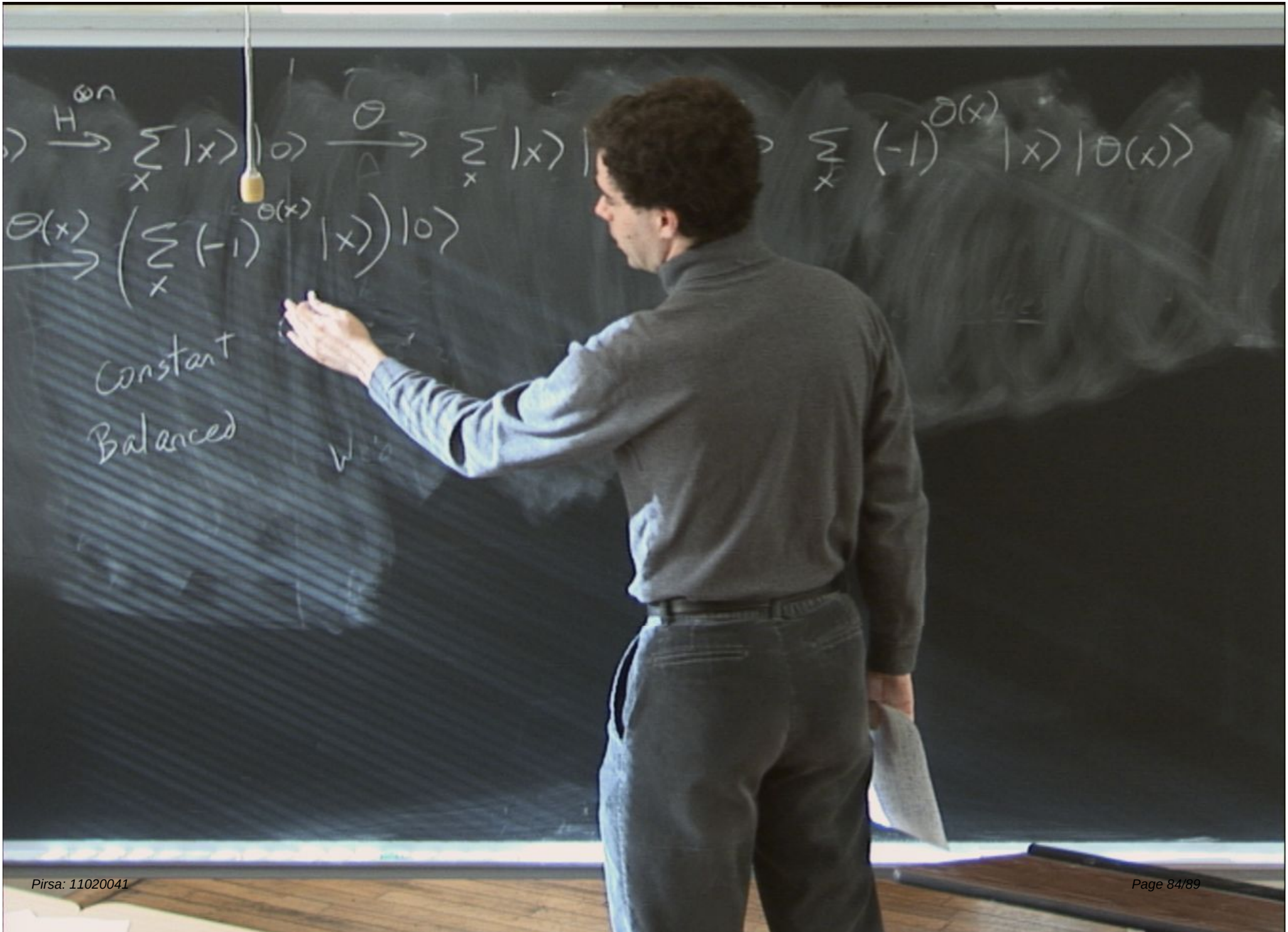

$|0\ldots 0\rangle|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{O}$

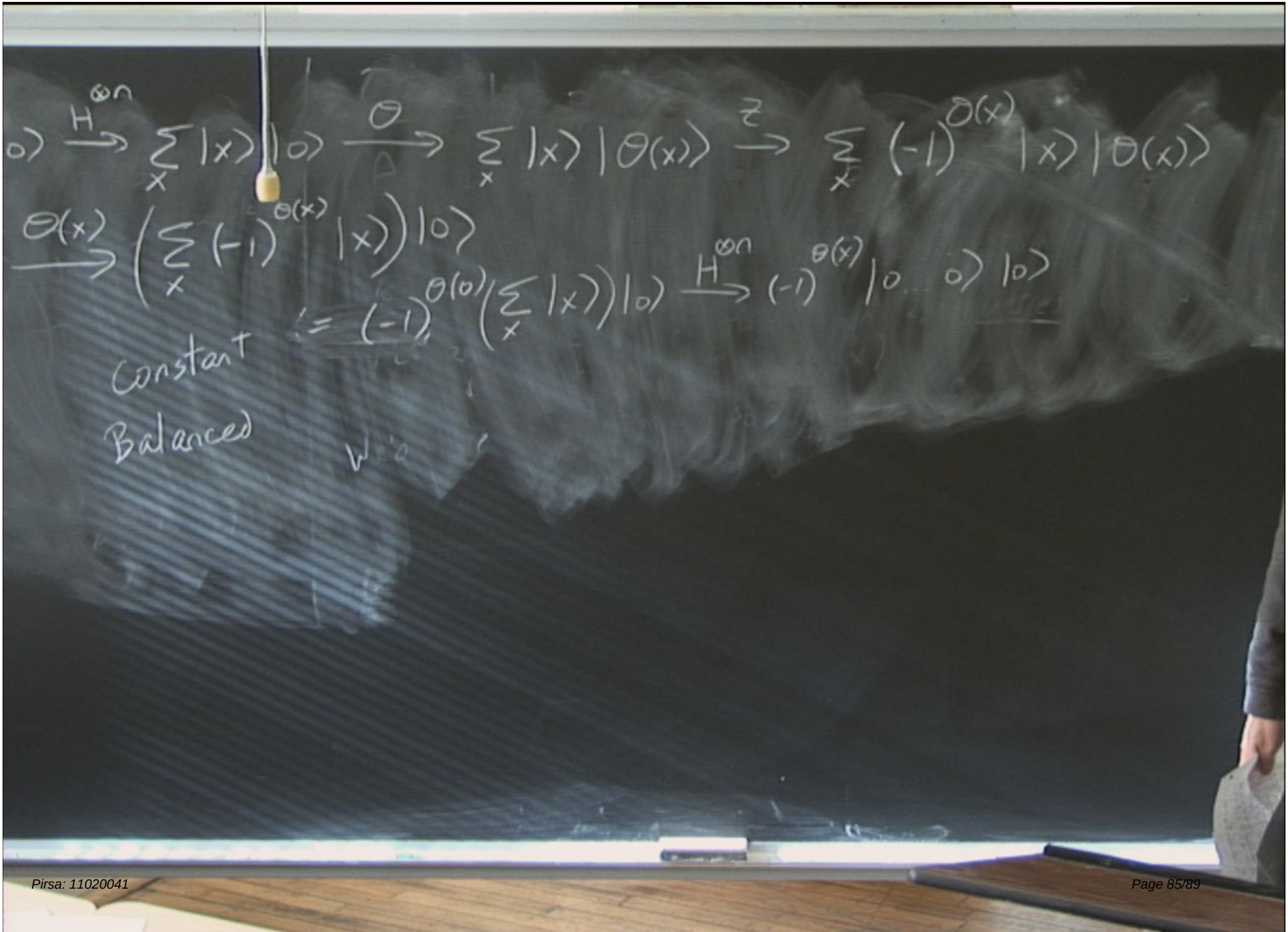

$|0\ldots 0\rangle|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{\theta} \sum_x |x\rangle|\theta(x)\rangle$

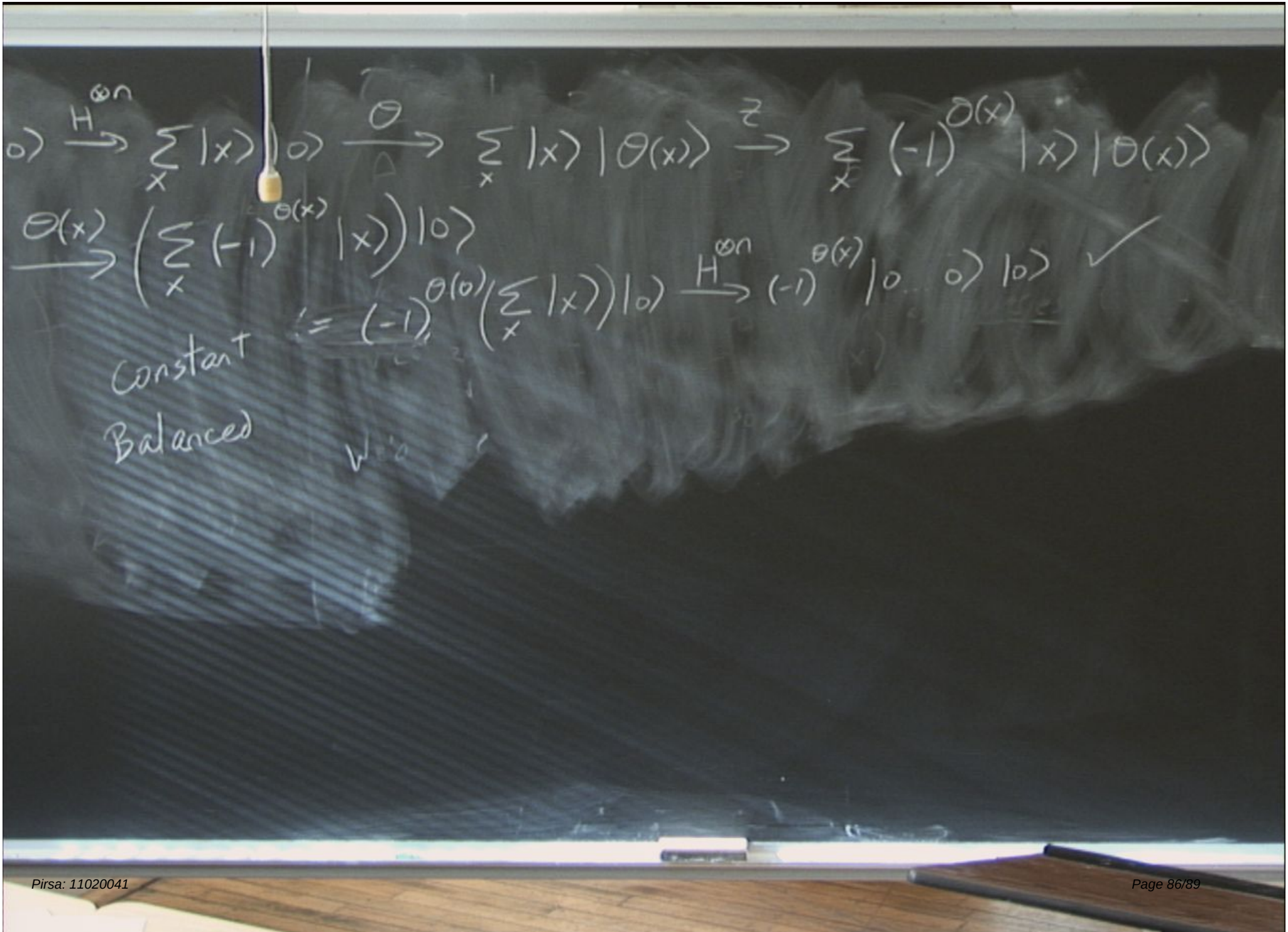

$|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{\Theta} \sum_x |x\rangle|\theta(x)\rangle \rightarrow \sum_x (-1)^{\theta(x)} |x\rangle|\theta(x)\rangle$

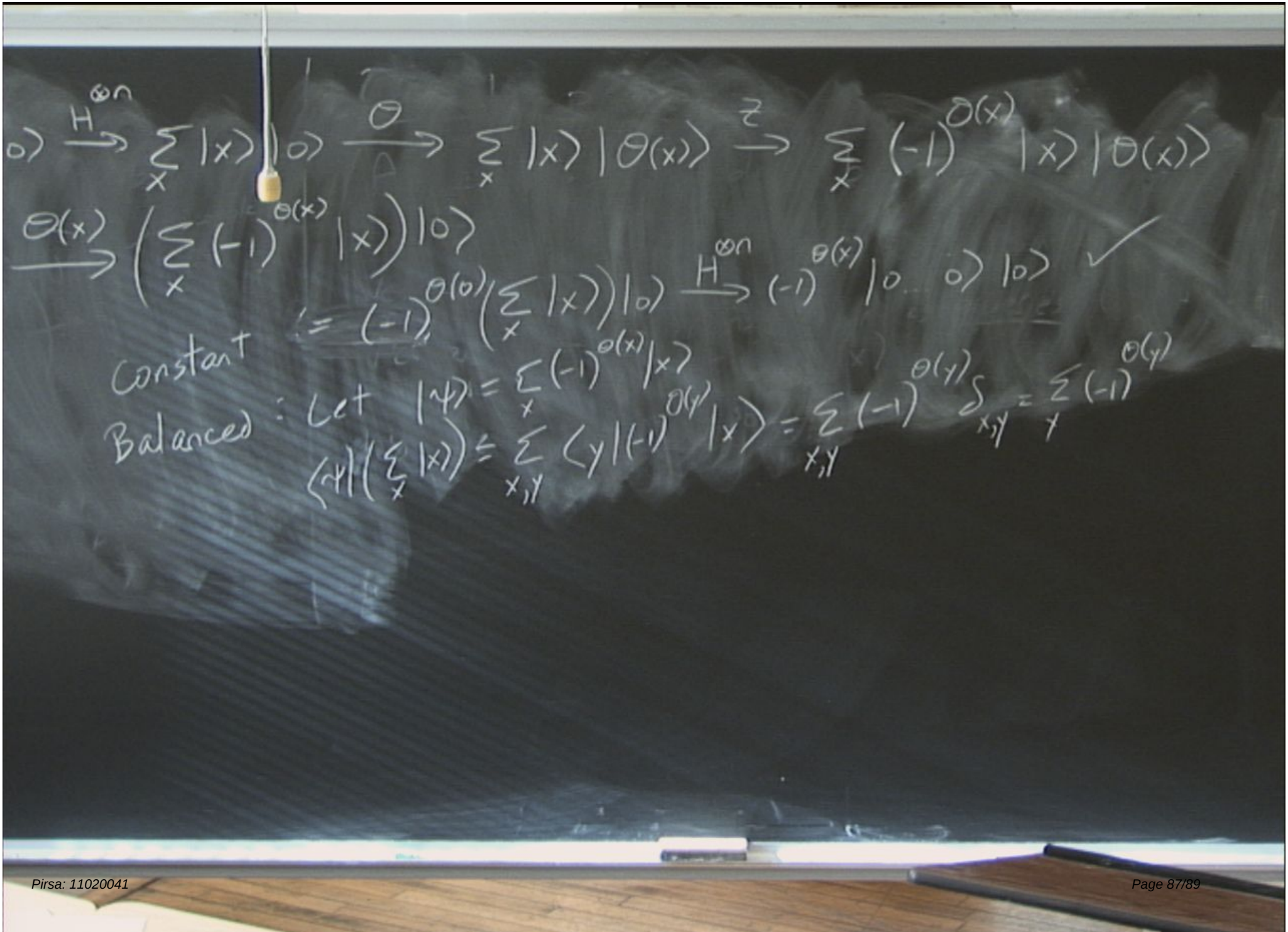

H^⊗n
Σ_x |x⟩|0⟩
O
|θ(x)⟩
Z
Σ_x (-1)^θ(x) |x⟩|θ(x)⟩
θ(x)
Σ_x (-1)^θ(x) |x⟩

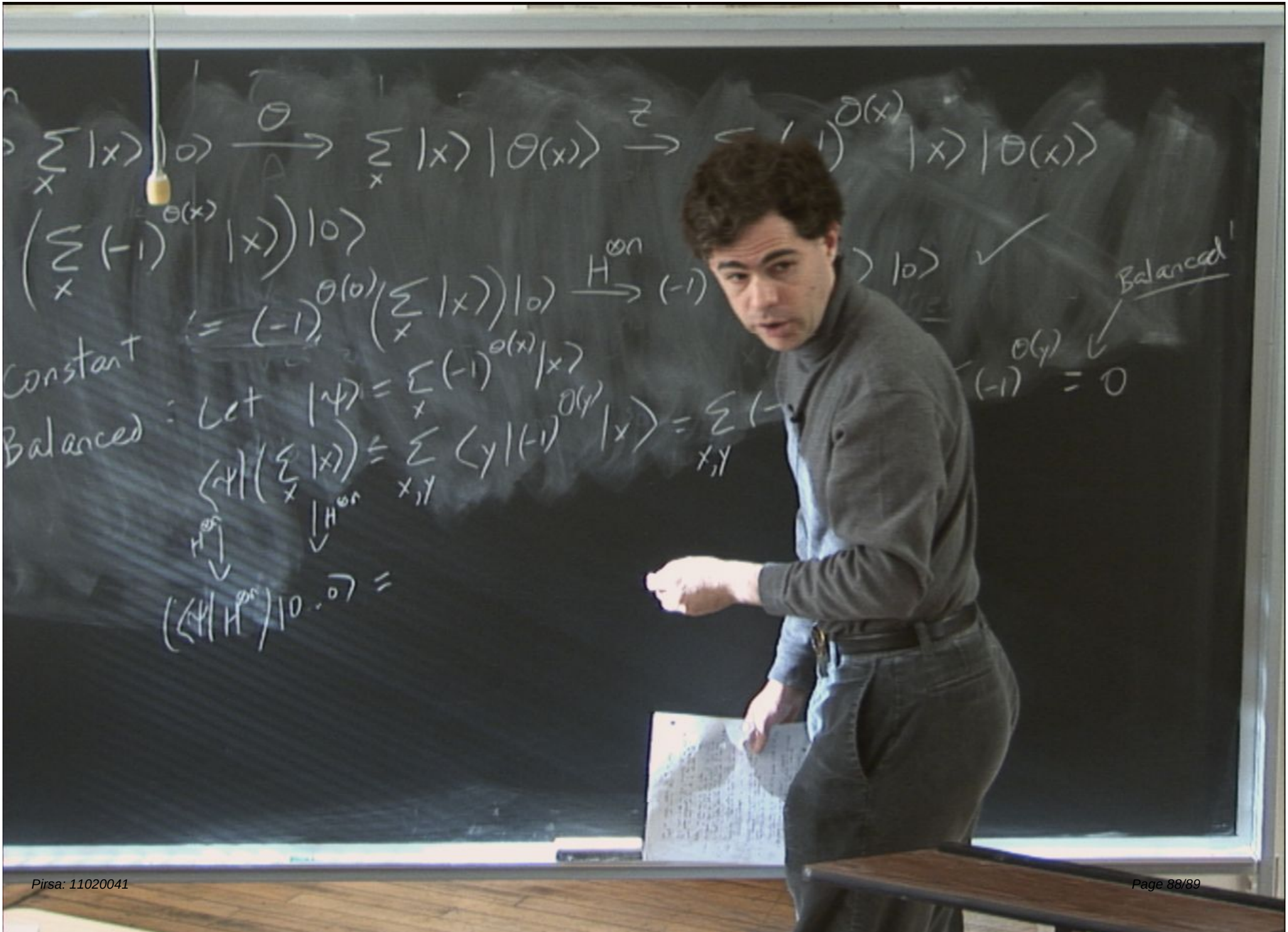

$\xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{\theta} \sum_x |x\rangle|\theta(x)\rangle \xrightarrow{z} \sum_x (-1)^{\theta(x)}$
$\xrightarrow{\theta(x)} \left(\sum_x (-1)^{\theta(x)}|x\rangle\right)|0\rangle$

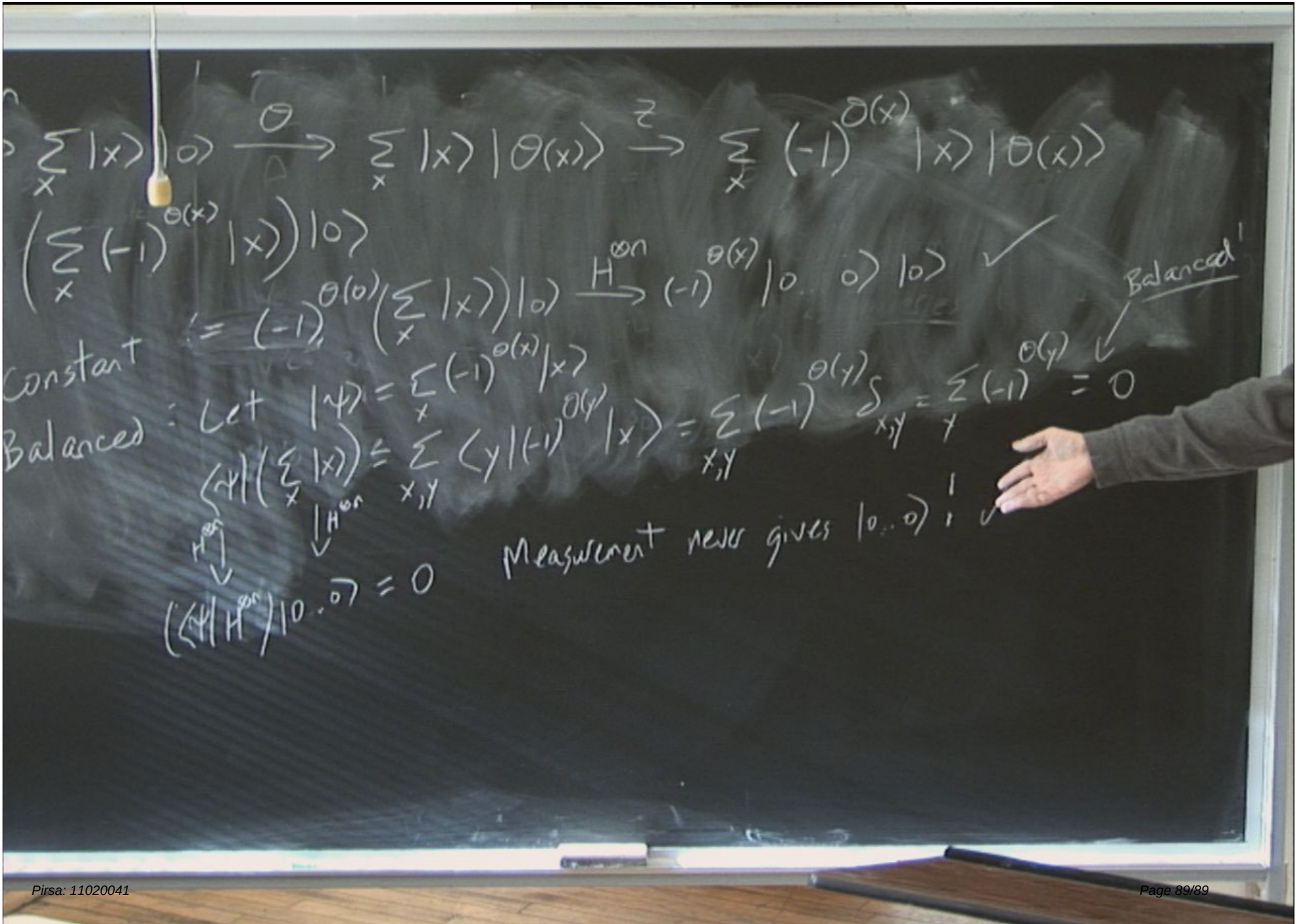

Constant
Balanced


$|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{\theta} \sum_x |x\rangle|\theta(x)\rangle \xrightarrow{Z} \sum_x (-1)^{\theta(x)}|x\rangle|\theta(x)\rangle$
$\xrightarrow{\theta(x)} \left(\sum_x (-1)^{\theta(x)}|x\rangle\right)|0\rangle$
$= (-1)^{\theta(0)}\left(\sum_x |x\rangle\right)|0\rangle \xrightarrow{H^{\otimes n}} (-1)^{\theta(x)}|0\ldots0\rangle|0\rangle$
Constant
Balanced


$|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle |0\rangle \xrightarrow{\theta} \sum_x |x\rangle |\theta(x)\rangle \xrightarrow{Z} \sum_x (-1)^{\theta(x)} |x\rangle |\theta(x)\rangle$
$\xrightarrow{\theta(x)} \left(\sum_x (-1)^{\theta(x)} |x\rangle\right) |0\rangle$
$= (-1)^{\theta(0)} \left(\sum_x |x\rangle\right) |0\rangle \xrightarrow{H^{\otimes n}} (-1)^{\theta(x)} |0 \ldots 0\rangle |0\rangle$ ✓
Constant
Balanced


$|0\rangle \xrightarrow{H^{\otimes n}} \sum_x |x\rangle|0\rangle \xrightarrow{\theta} \sum_x |x\rangle|\theta(x)\rangle \xrightarrow{Z} \sum_x (-1)^{\theta(x)}|x\rangle|\theta(x)\rangle$
$\xrightarrow{\theta(x)} \left(\sum_x (-1)^{\theta(x)}|x\rangle\right)|0\rangle$
$= (-1)^{\theta(0)}\left(\sum_x |x\rangle\right)|0\rangle \xrightarrow{H^{\otimes n}} (-1)^{\theta(x)}|0\ldots 0\rangle|0\rangle$ ✓
Constant
Balanced: Let $|\psi\rangle = \sum_x (-1)^{\theta(x)}|x\rangle$
$\langle\psi|\left(\sum_x |x\rangle\right) = \sum_{x,y}\langle y|(-1)^{\theta(y)}|x\rangle = \sum_{x,y}(-1)^{\theta(y)}\delta_{x,y} = \sum_y (-1)^{\theta(y)}$


Constant
Balanced
Balanced!


$\sum_x |x\rangle|0\rangle \xrightarrow{\Theta} \sum_x |x\rangle|\Theta(x)\rangle \xrightarrow{Z} \sum_x (-1)^{\Theta(x)}|x\rangle|\Theta(x)\rangle$
$\left(\sum_x (-1)^{\Theta(x)}|x\rangle\right)|0\rangle$
$= (-1)^{\Theta(0)}\left(\sum_x |x\rangle\right)|0\rangle \xrightarrow{H^{\otimes n}} (-1)^{\Theta(x)}|0\ldots0\rangle|0\rangle$ ✓
Balanced!
Constant
Balanced: Let $|\psi\rangle = \sum_x (-1)^{\Theta(x)}|x\rangle$
$\langle\psi|\left(\sum_x |x\rangle\right) = \sum_{x,y}\langle y|(-1)^{\Theta(y)}|x\rangle = \sum_{x,y}(-1)^{\Theta(y)}\delta_{x,y} = \sum_y (-1)^{\Theta(y)} = 0$
$H^{\otimes n}$ $H^{\otimes n}$
$(\langle\psi|H^{\otimes n})|0\ldots0\rangle = 0$
Measurement never gives $|0\ldots0\rangle$; !