Quantum Oracle Engineering

Lesson 3: Ship it

“It’s just an implementation detail.”

The rollout in ten lines

```
board = start
for h in 1..H:
    black places on a random empty cell
    white places on a random empty cell
    for every stone:
        roll a d20
        count its friendly neighbors
        if die < 4 - friends: mark it
    flip every marked stone
return black > white
```

What does one call compute?

input

empty 3 × 3 board

two rounds

ranks + dice in registers

→

rollout A

place → event

place → event

count the final colors

→

output

one payoff bit

1 when Black > White

scratch 0; records retained

Today: both players choose uniformly among legal cells.

To evaluate action i: supply that first move, then randomize the continuation.

The benchmark win rate is an average over first moves.

Which line is the hard one?

A `roll a d20`

B `pick a random empty cell`

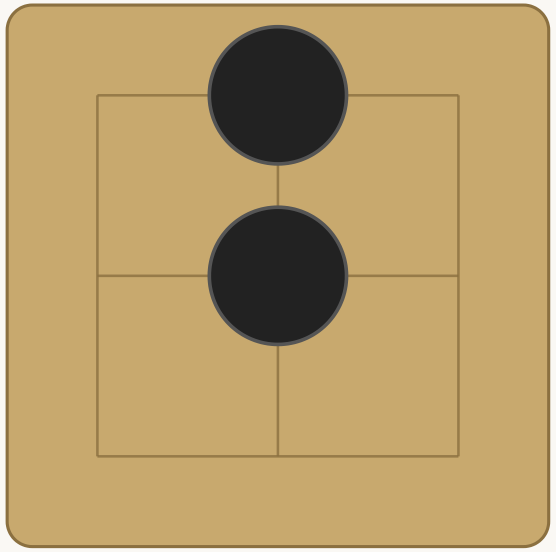
C `count the friendly neighbors`

D `black > white`

hands up

Two qubits per cell

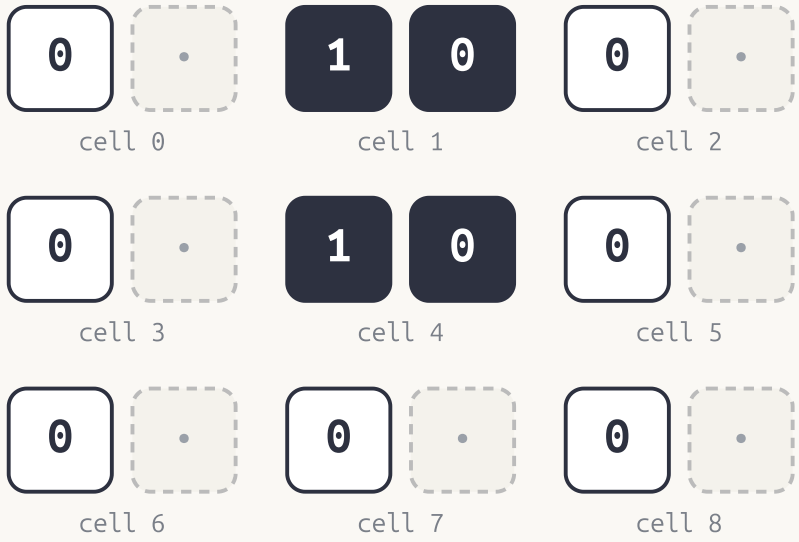
our trace, after round 1



18 qubits per state

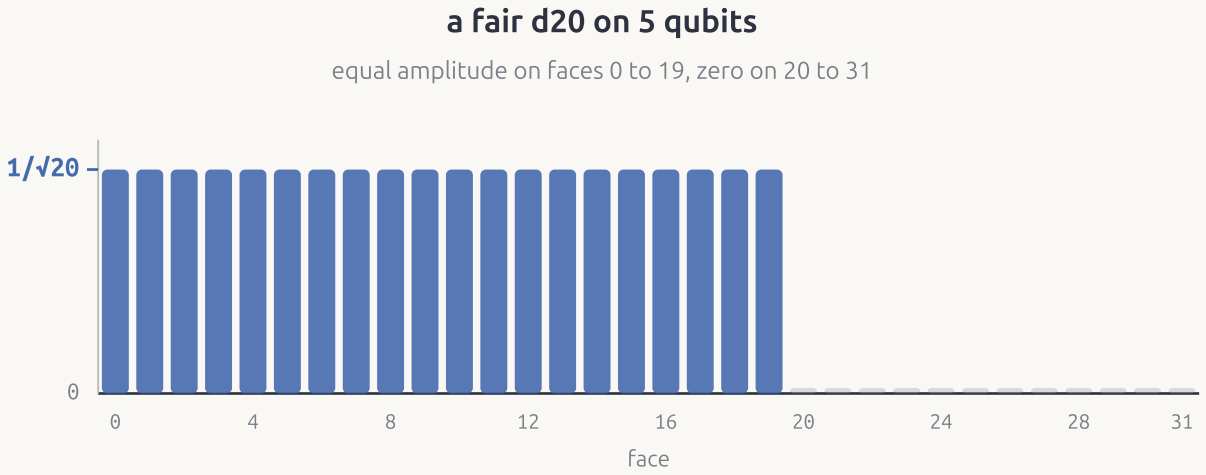
One occupancy register; a new color register for each round.

occupiedcolour



0 black, 1 white; color is ignored while empty

Twenty faces on five qubits



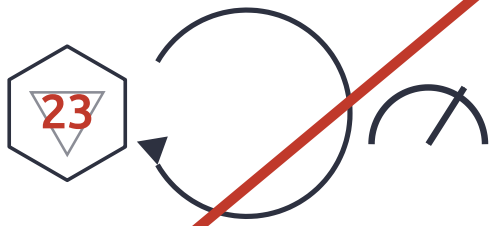

```
from math import sqrt
from qiskit import QuantumCircuit, QuantumRegister
from qiskit.circuit.library import StatePreparation

die = QuantumRegister(5, "die")
qc = QuantumCircuit(die)
amplitudes = [1 / sqrt(20)] * 20 + [0] * 12
qc.append(StatePreparation(amplitudes), die)
```

Prepare a d20 before decoding a move

Two shortcuts change the computation

reroll the bad faces



face 23? measure, roll again

a measurement inside the circuit

use all thirty-two faces

`4 / 20 = 1 in 5`

`4 / 32 = 1 in 8`

every flip probability drops by 20/32

a different game

Prepare a d20 before decoding a move

Ninety qubits of dice

90 qubits

9 cells × 2 rounds × 5 qubits; encoded faces 0–19

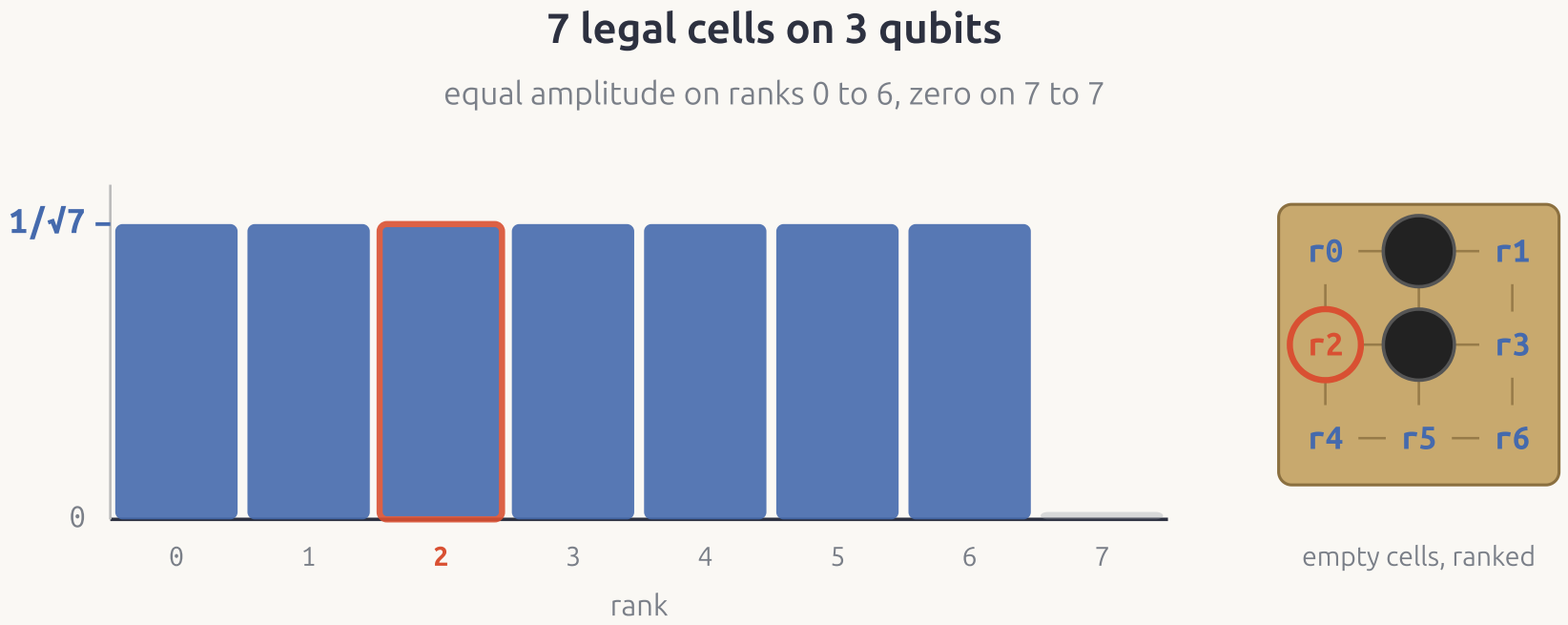
“Pick a random empty cell”

```
1  board = start
2  for h in 1..H:
3      black places on a random empty cell
4      white places on a random empty cell
5      for every stone:
6          roll a d20
7          count its friendly neighbors
8          if die < 4 - friends: mark it
9      flip every marked stone
10 return black > white
```

now

Lines 3 and 4: decode a legal move

Random means a register

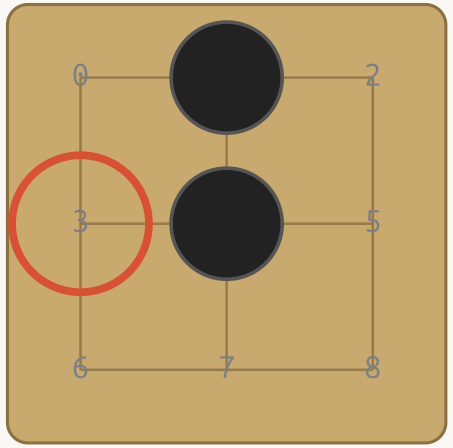


Our trace: round 2, Black's rank is 2.

Which cell does the rank select?

Both branches have seven empty cells. Rank r = 2 is zero-based.

our trace, round 2

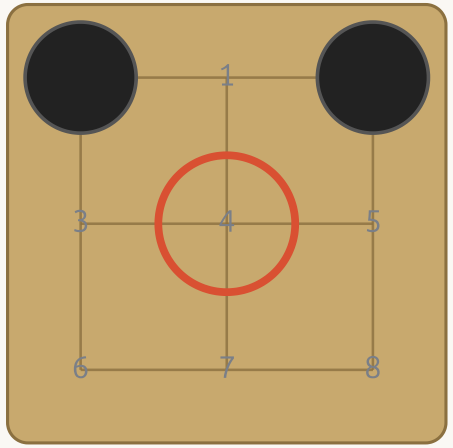


rank 2 → cell 3

Which cell in each branch?

numbers label cell indices

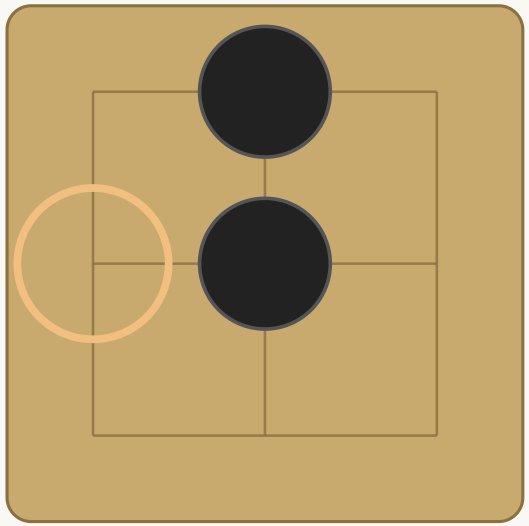
another branch



rank 2 → cell 4

Decode coherently: the board controls the answer; we do not measure it.

Compare first, then increment



prefix counter: 4 qubits (0–9)

0	1	1	1	= 7

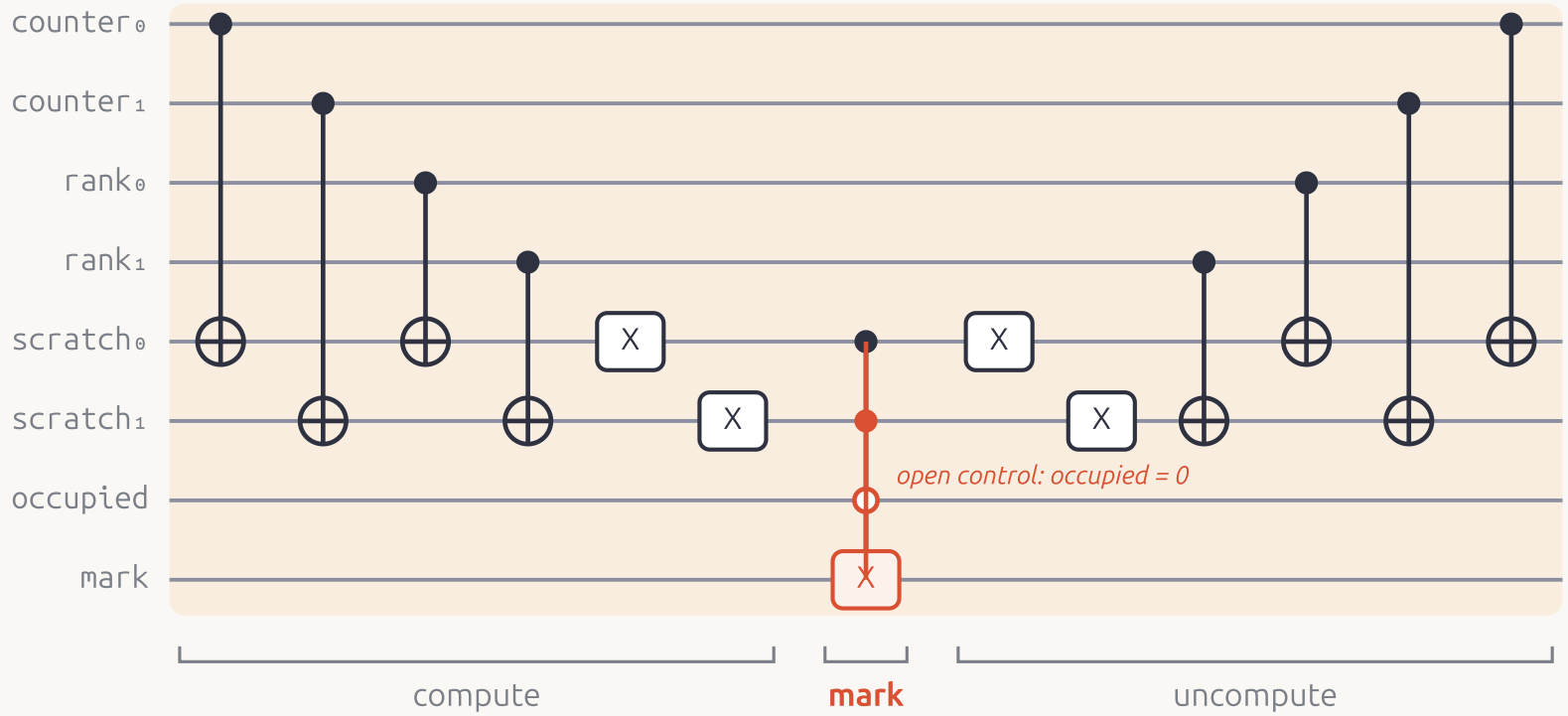
`rank r = 2`

move index = 3

unwind next

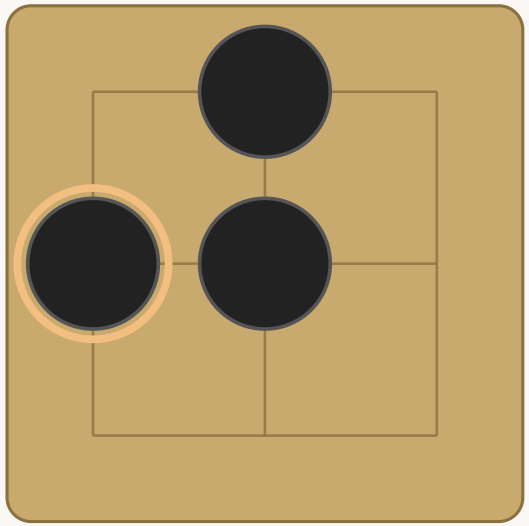
round 2: Black selects cell 3

The equality test as gates



Lines 3 and 4: decode a legal move

Unwind before placing



prefix counter: 4 qubits (0–9)

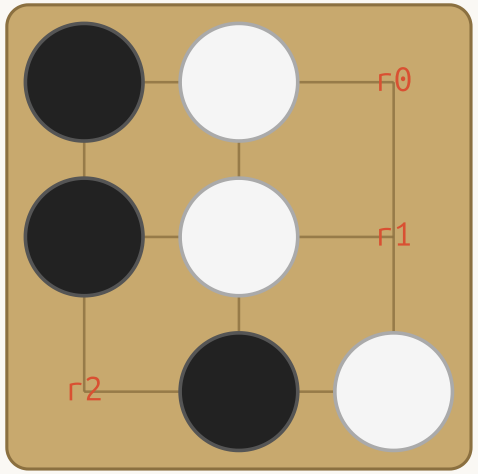
0	0	0	0	= 0

`rank r = 2`

place Black

round 2: Black selects cell 3

Every rank has a defined result



rank	index
0	2
1	5
2	6
3–15	9: sentinel → no-op

The decoder was the hard part

A	`roll a d20`	**state preparation** reuse it for dice and ranks
B	**`pick a random empty cell`**	**the decoder we built** board-dependent index; clean scratch
C	`count the friendly neighbors`	**routine** flags and a counter
D	`black > white`	**terminal predicate** next: count colors and compare

Sequential scan: $O(Nw)$ gates and $O(w)$ reusable scratch, $w = \lceil \log_2(N + 1) \rceil$.

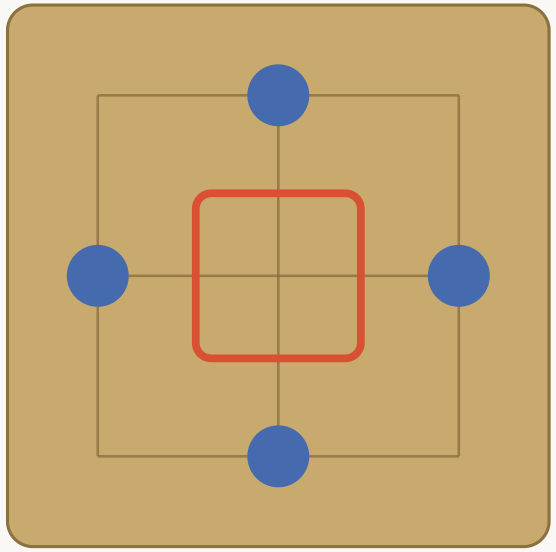
“Decide, then flip”

```
1   board = start
2   for h in 1..H:
3       black places on a random empty cell
4       white places on a random empty cell
5       for every stone:
6           roll a d20
7           count its friendly neighbors
8           if die < 4 - friends: mark it
9       flip every marked stone
10  return black > white
```

now

Lines 7 to 9: decide from one board, write the next

A cell knows its neighbors



maximum possible friend count

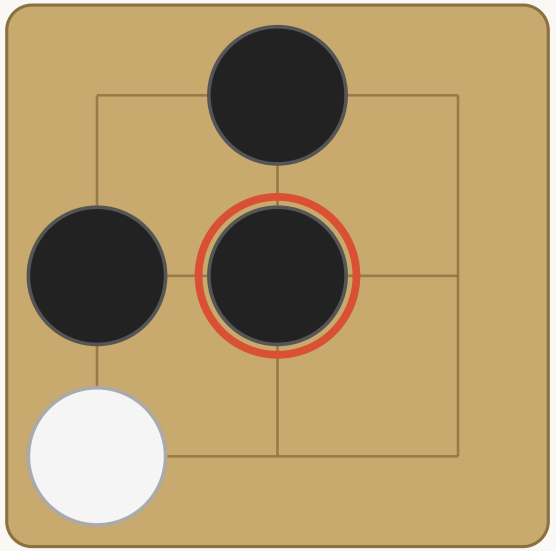
2^2	2^1	2^0
1	0	0

centre: 4 neighbours

count = 4

three qubits cover 0 to 4

Count the friends



same-colour flags

1	0	1	0
up	down	left	right

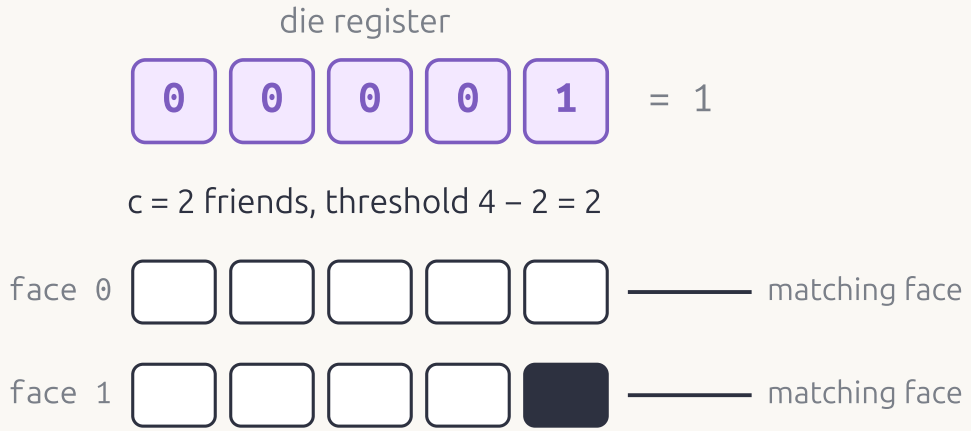
count

0	1	0

c = 2 friends

round 2, cell 4: both occupied, colors equal

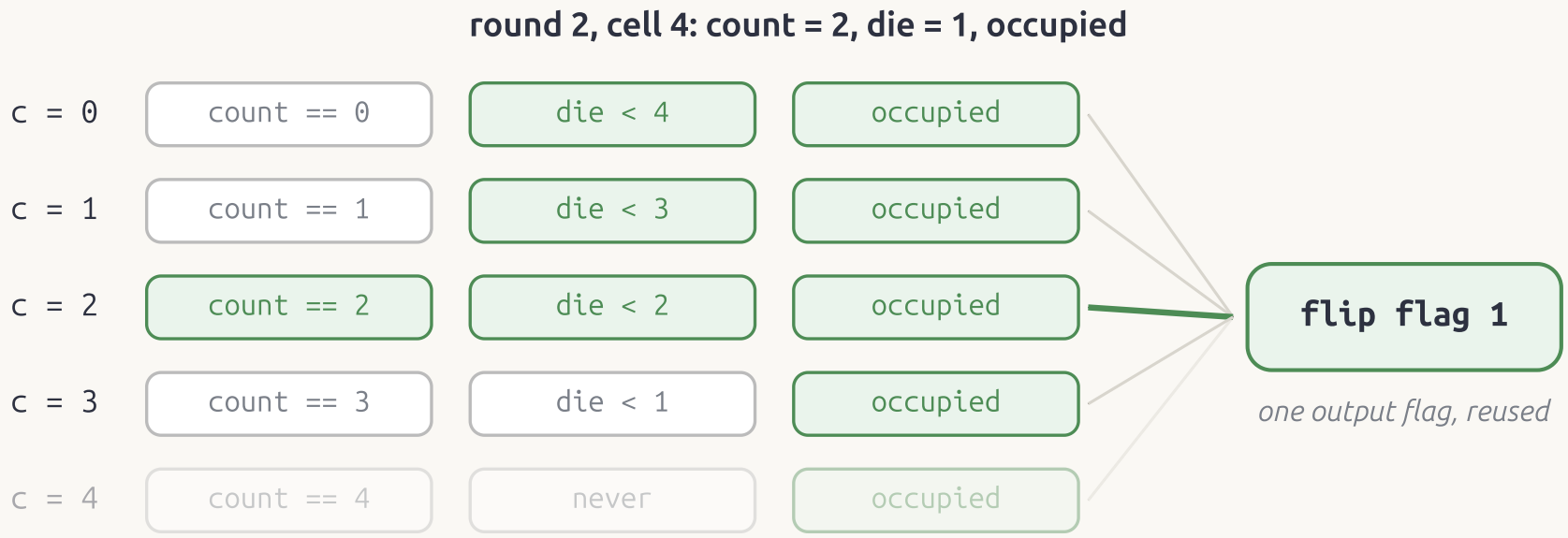
Die below threshold, as control patterns



friends	threshold	patterns
0	4	4
1	3	3
2	**2**	**2**
3	1	1
4	0	none

`flag 1`

Five cases, one flip flag



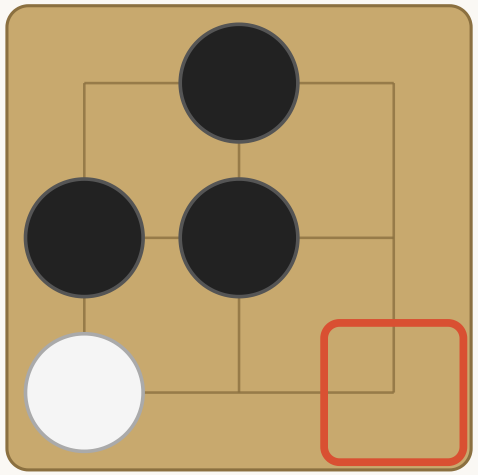
Each row expands into joint count + die control patterns.

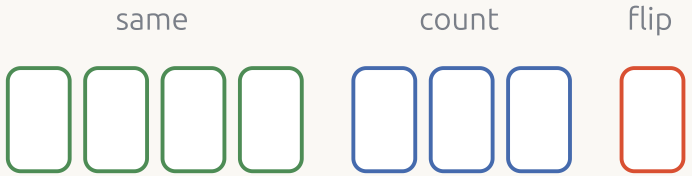
Read old colors, write new colors

toggled on the new board; the old board is untouched

```
9 occupancy + 3 × 9 color qubits = 36 state qubits
```

Eight qubits borrowed from the shared pool





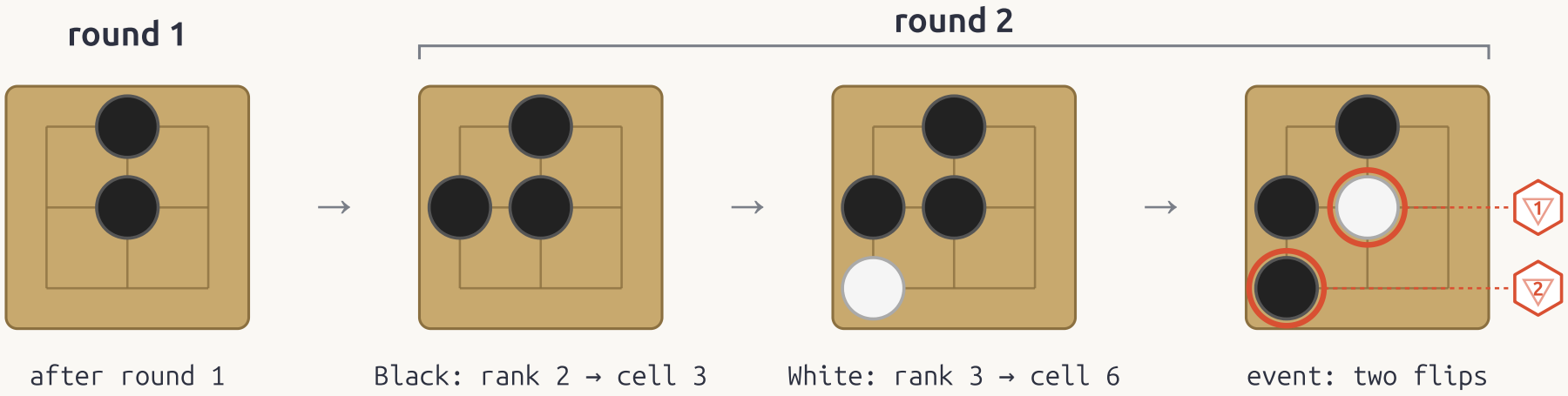
8 event scratch qubits

all clean

`reused 9 times`

borrowed from the 13-qubit shared pool

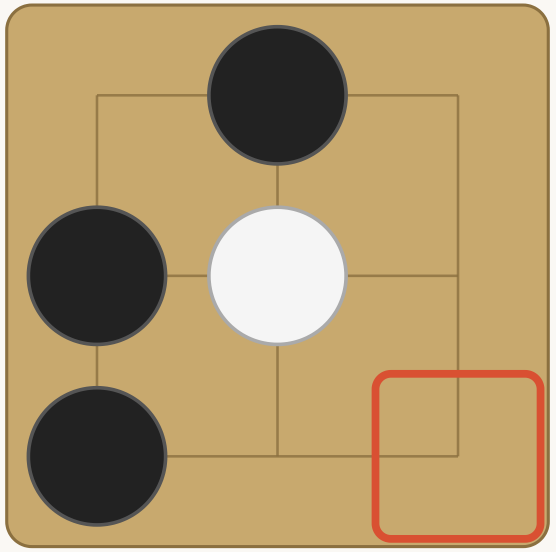
Assemble the two rounds




```
oracle.compose(round1, qubits=wires1, inplace=True)
oracle.compose(round2, qubits=wires2, inplace=True)
```

Fresh ranks, dice, move records, and destination colors for each round.

Count the final colors



black

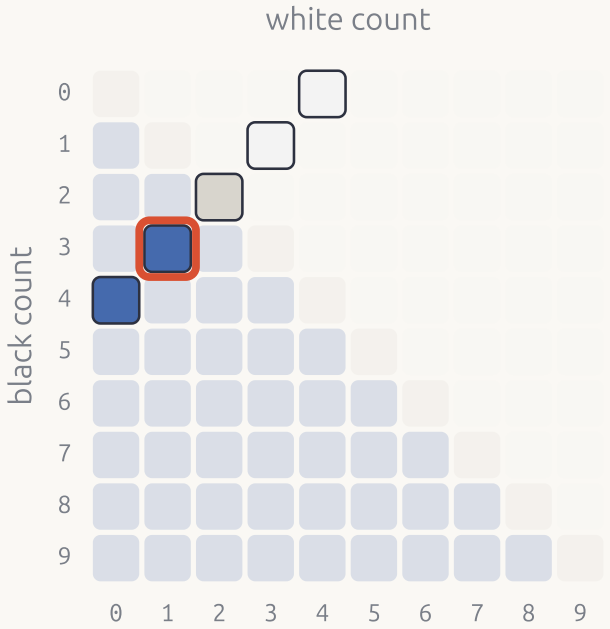
0	0	1	1	= 3

white

0	0	0	1	= 1

Our trace: four occupied cells after two rounds

A comparator includes unreachable inputs



generic comparator: 45 winning patterns

outlined pairs have black + white = 4

`payoff = 1`

black 3 > white 1

A truth table includes inputs absent from this rollout.

How many qubits?

A about 30

B about 100

C about 170

D about 500

hands up

Where the 169 go

dice 90 · boards 36 · move records · ranks 13 · scratch 13 · payoff 1

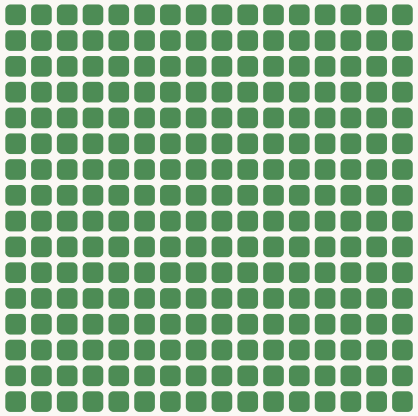
169 qubits

9,768 gates · depth 3,079

paper benchmark; before native-gate decomposition

Check the trace, then the distribution

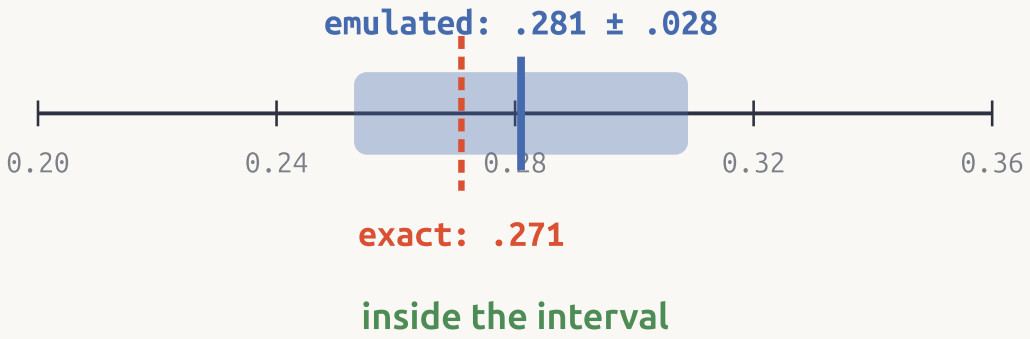
256 seeds, bit for bit



256 / 256 agree

Our trace: Black 3, White 1, payoff 1

1,000 seeded basis runs; 95% interval



inside the interval

Separate checks: preparation amplitudes and scratch restoration

It reverses. Does it simulate the right game?

A followed by A† returns the input—even for the wrong game.

- ✓ round semantics defined
- ✓ old colors preserved; flips written to the next register
- ✓ selection scratch cleared before placement
- ✓ randomness kept in explicit registers

Lesson 4: change the ordering and diagnose what breaks.

Lesson 4: Reversible by design

Continue to Lesson 4

“It runs backward.
Does it simulate the right game?”

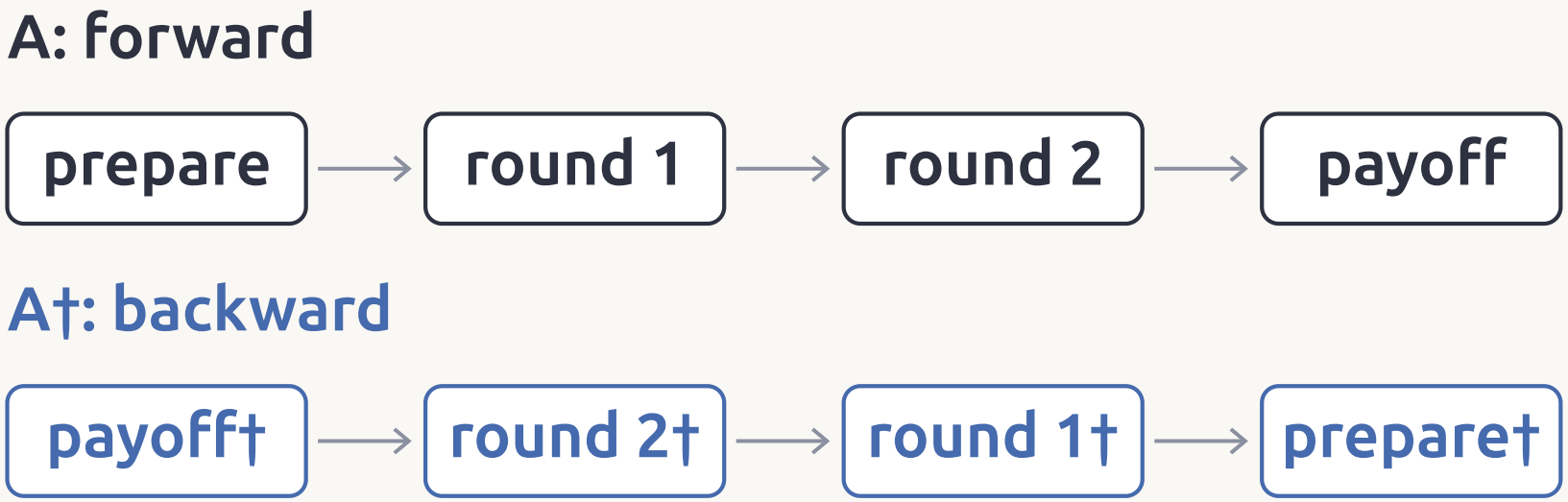
shukla.io/quantum-oracle-engineering

Quantum Oracle Engineering

Lesson 4: Reversible by design

“Reversibility is just bookkeeping.”

One line of code

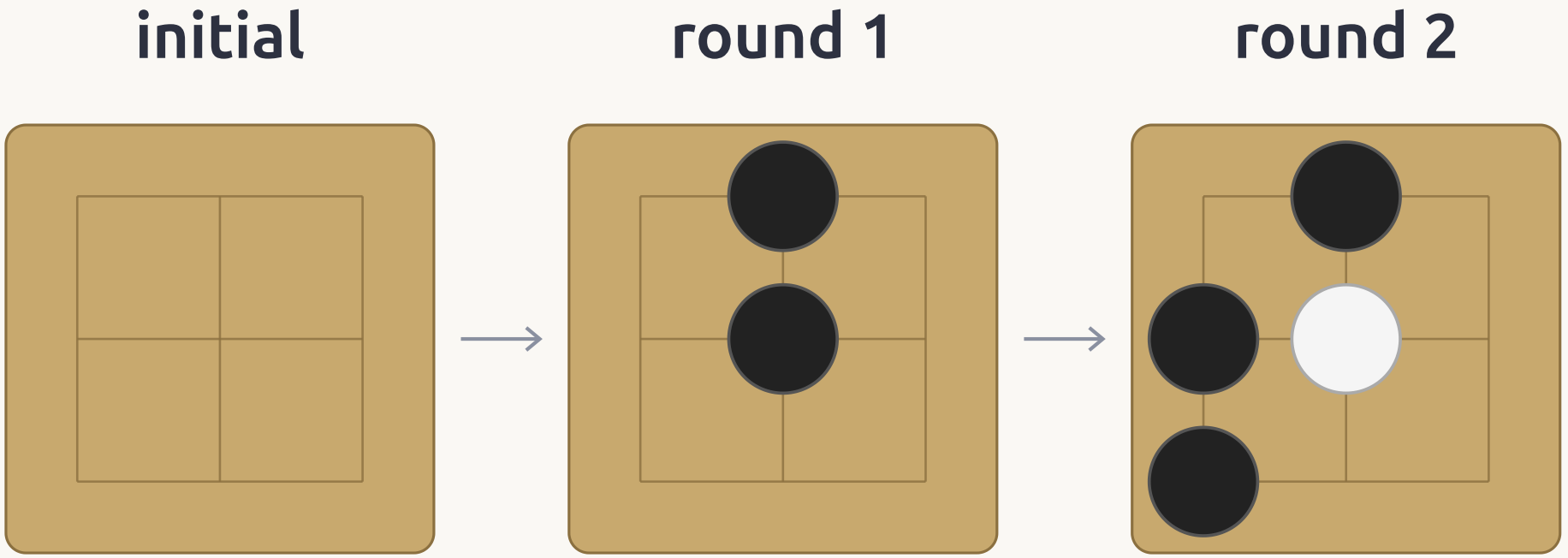



```
A_dagger = A.inverse()
```

Three questions to answer

1. **Undo: what must the inverse still read?**
2. **Scratch: when can it go back to zero?**
3. **Checks: does the circuit play the right game?**

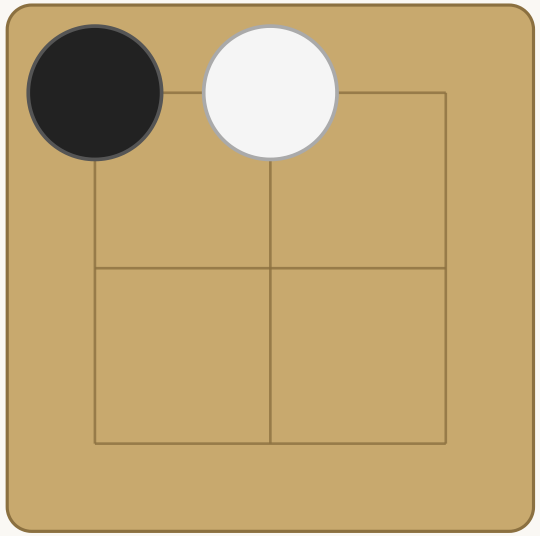
Why three color registers?



Keep the inputs that undo needs

Predict the second stone

dice: 0, 3



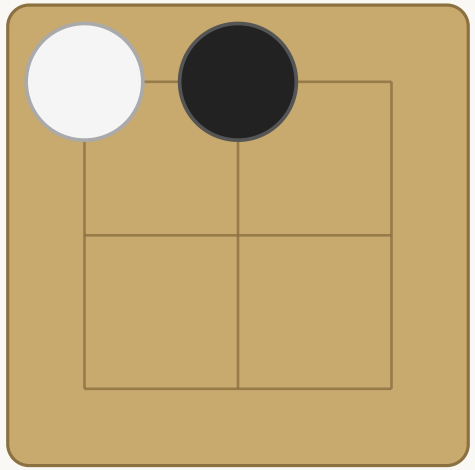
die < 4 – friends

cell 0 → White

cell 1 → ?

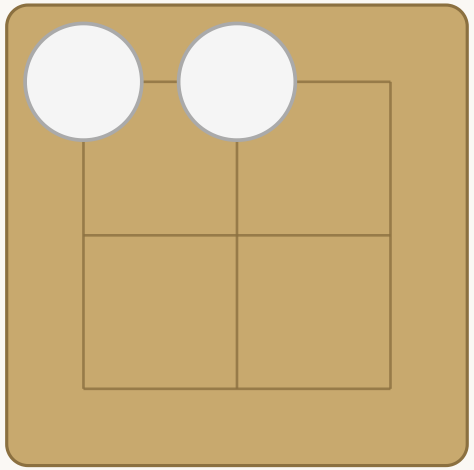
Cell 1 read a flipped neighbor

Old colors



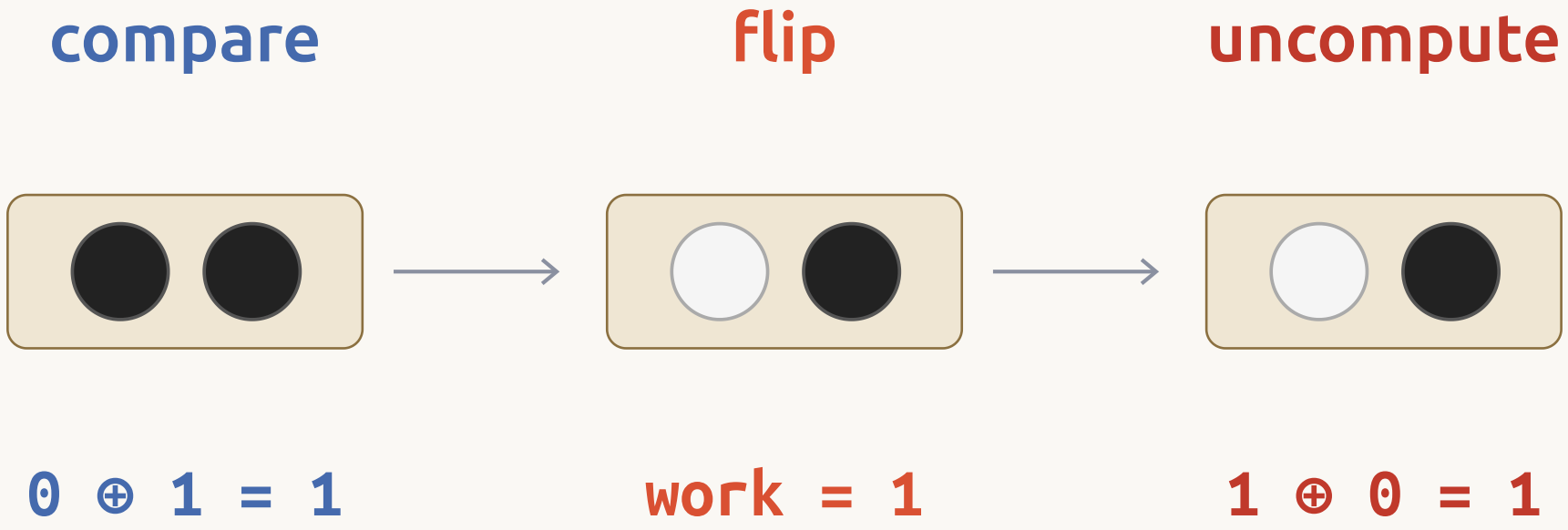
3 < 4: flip

Updated colors

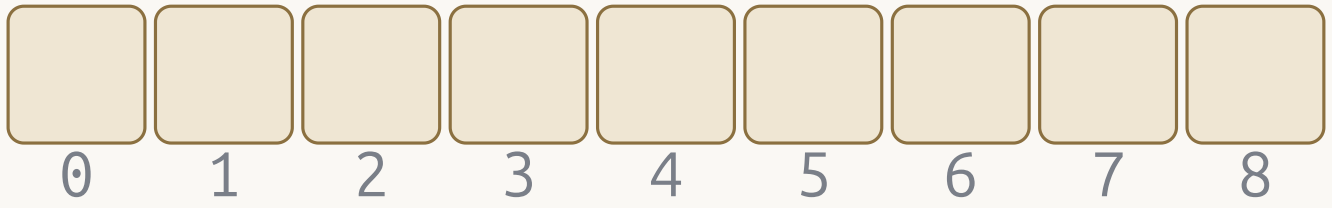


3 ≥ 3: stay

Cleanup can read a changed input too



Occupancy can update in place

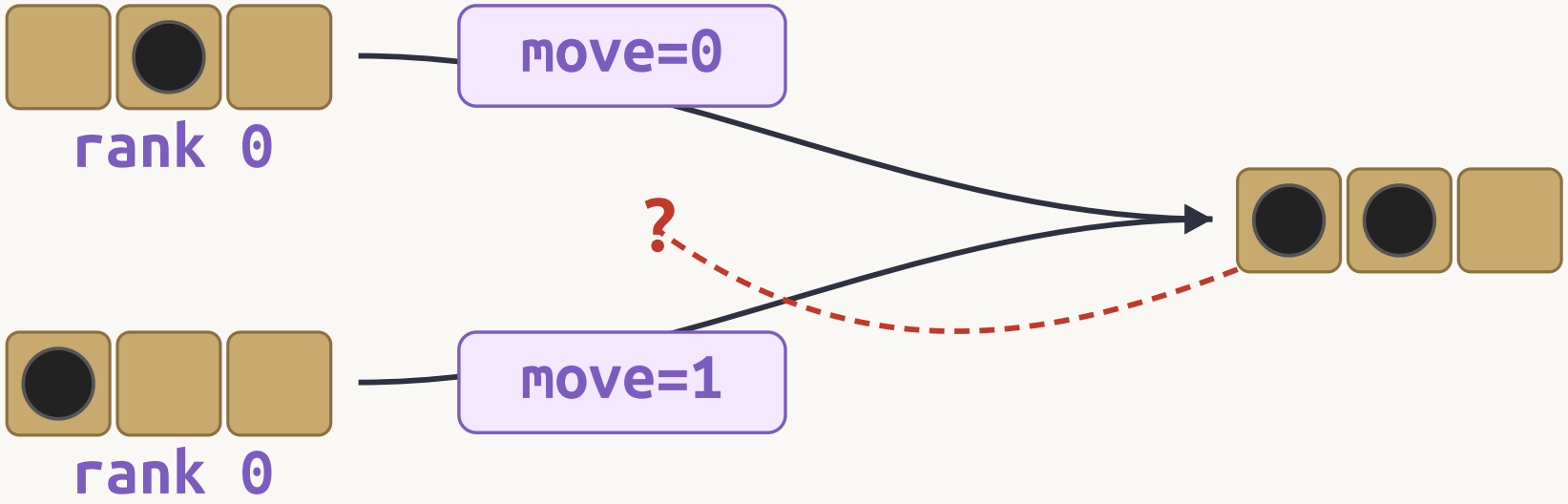


`move 1=4` `move 2=1` `move 3=3` `move 4=6`

back to empty

Keep the inputs that undo needs

Two inputs, one visible result

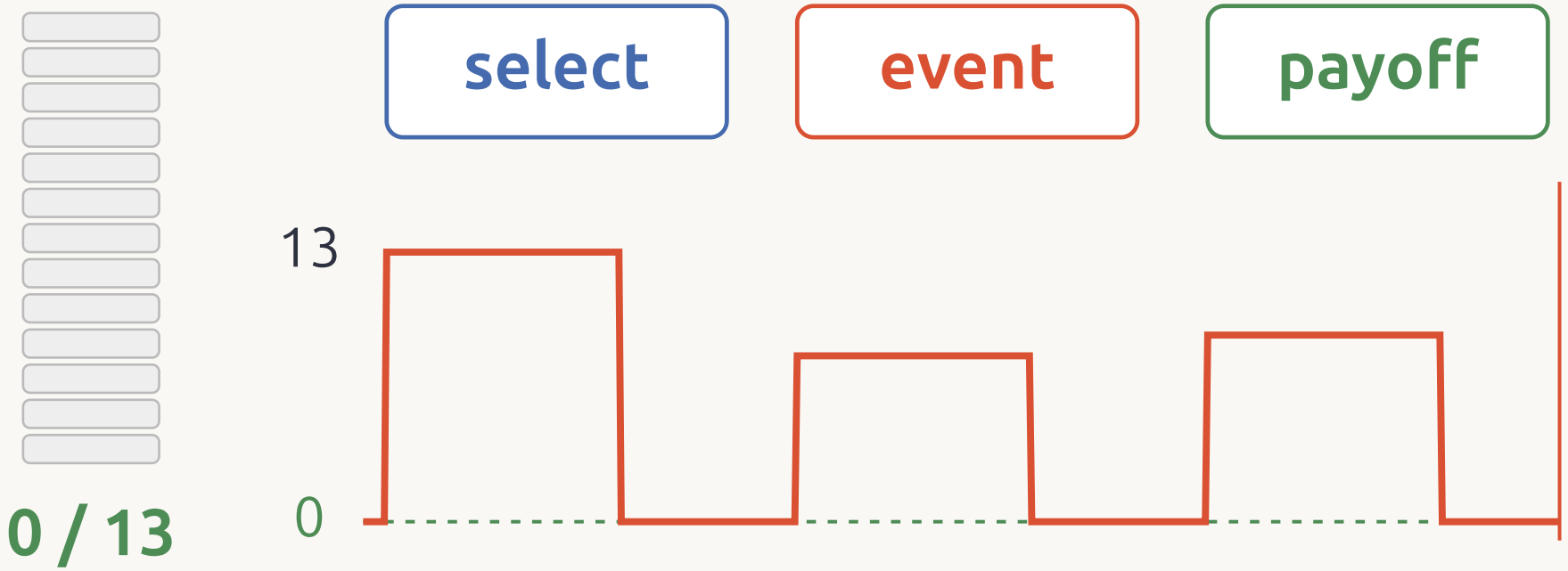


Records persist; scratch is borrowed

records	
dice	90
boards	36
move indices	16
ranks	13
155 qubits	

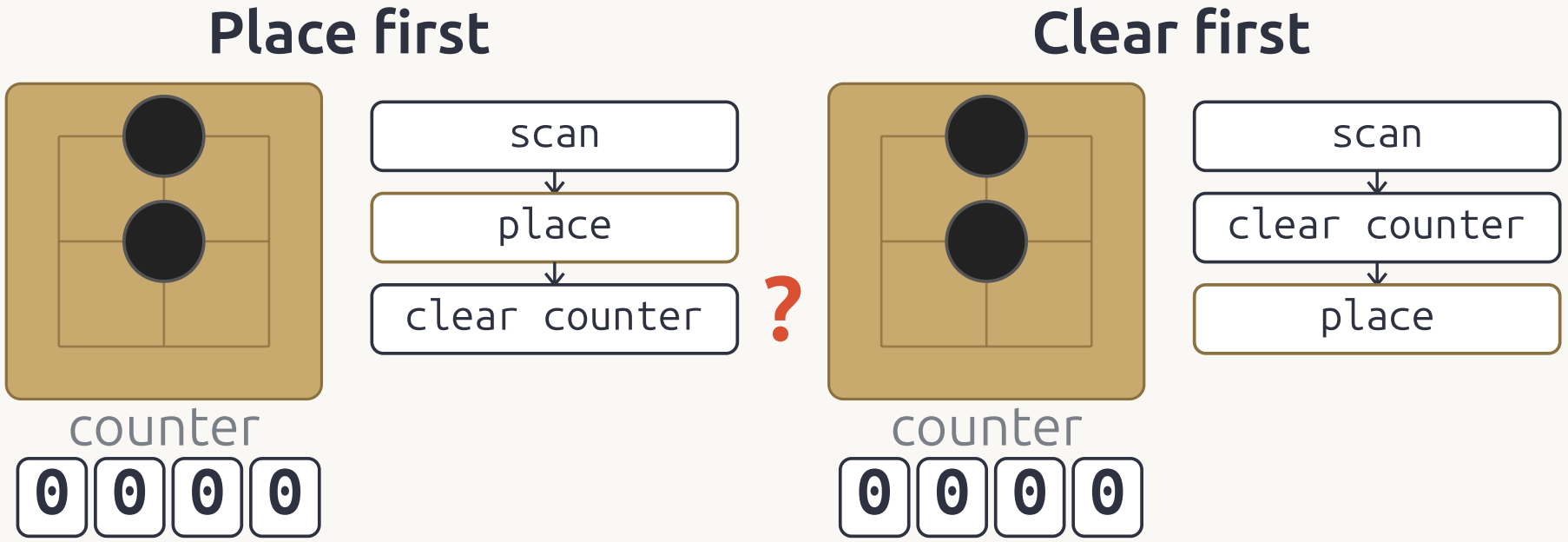
rank-select scratch	
prefix counter	4
equality bits	4
temporary index	4
flag	1
13 qubits	

Thirteen qubits, borrowed repeatedly



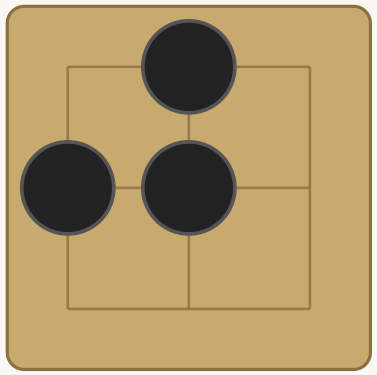
Return scratch before the next borrower

Place first, or clear the counter first?



Undo must see what do saw

Place first



scan → place → clear counter

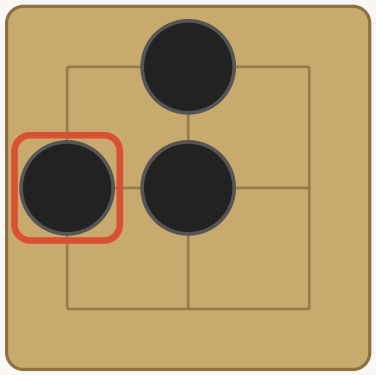
counter

0 0 0 1

6 empties left: off by one

= 1, dirty

Clear first



scan → clear counter → place

counter

0 0 0 0

the 7 empties the scan saw

= 0, clean

clear the counter first, then place

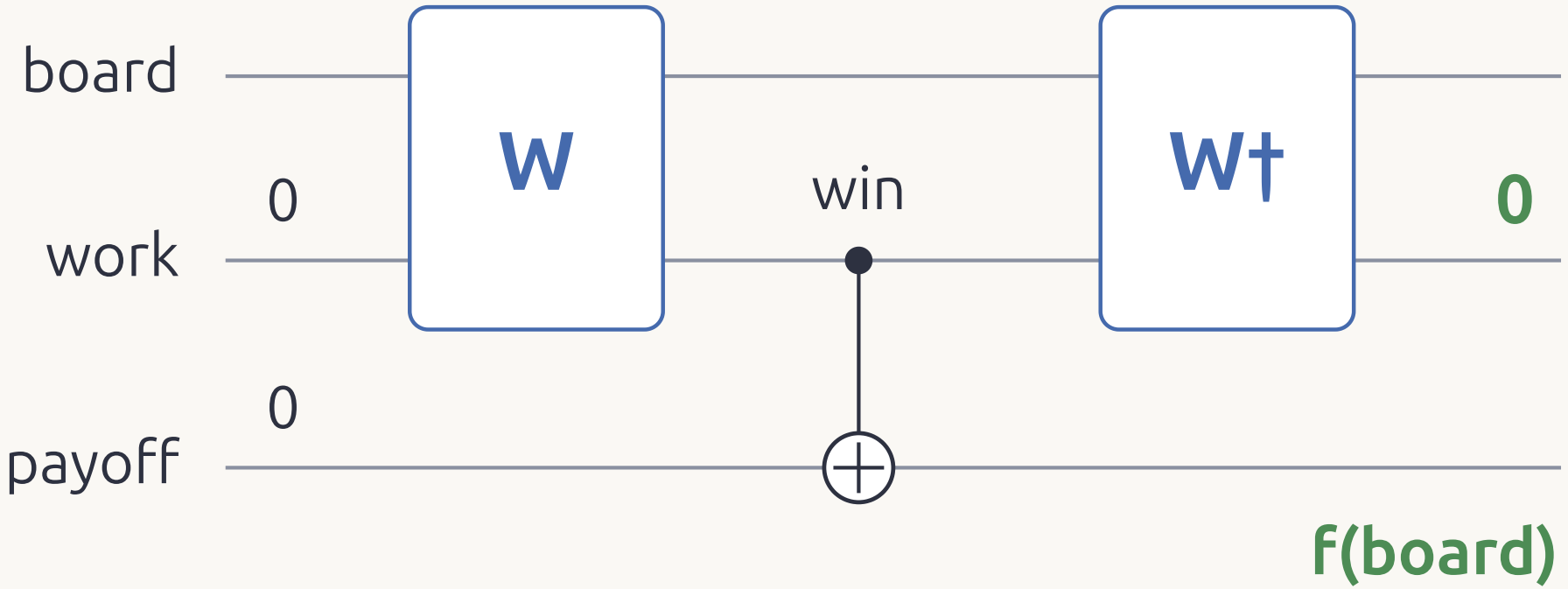
The next borrower assumes zero

rank = 2

cell	0	2	3	5	6	7	8
clean	0	1	**2**	3	4	5	6
dirty	1	**2**	3	4	5	6	7

Return scratch before the next borrower

Compute, copy the bit, uncompute



Three checks, three different claims

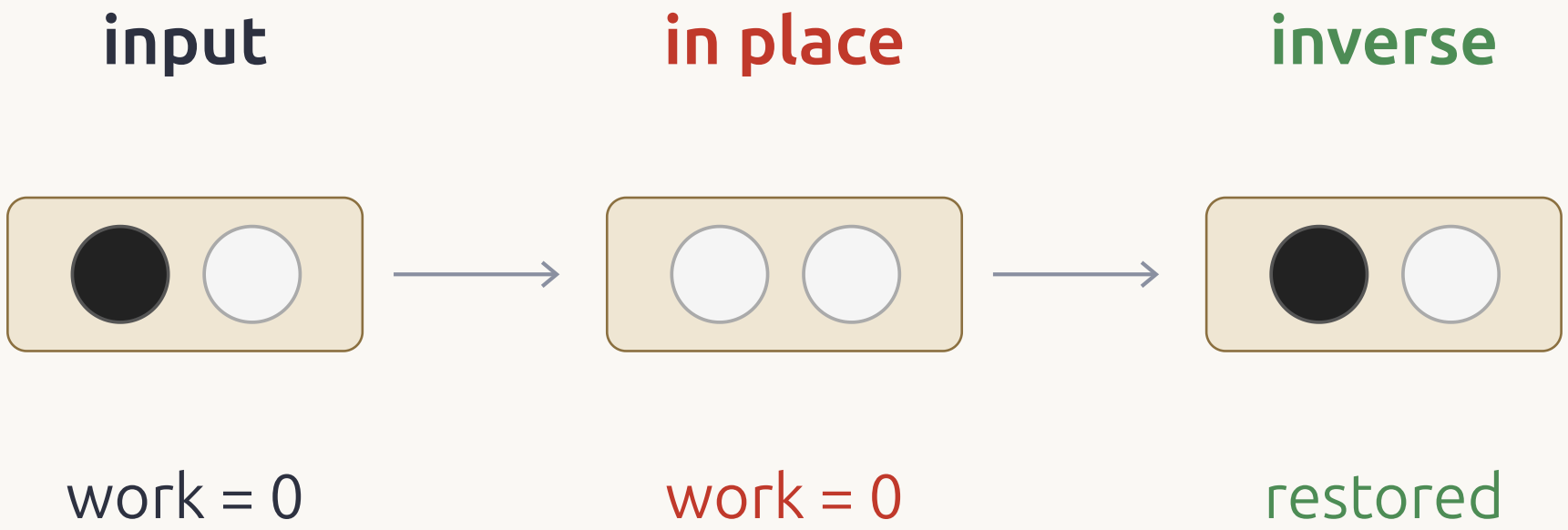
Rules — 4/4 inputs

Phases — matched

Inverse — passed

Test the meaning as well as the inverse

A round trip can pass the wrong game



One preparation, then repeated turns

Put the complete oracle into Q

After the phase mark, zero is not the goal

A → A†

A → Z → A†

P(all zero)?

1

¼

Records and scratch: 169 qubits

155

records

13

scratch

1

payoff

Larger boards and longer rollouts

Board · rounds	Qubits	Gates
3×3 · 2	169	9,768
5×5 · 5	916	76,720
10×10 · 5	3,363	481,201
10×10 · 10	6,503	793,901
20×20 · 10	25,189	6,072,641

Review the order, not just the gate list

1. Right game?
2. Original inputs kept?
3. Scratch back to zero?
4. Complete A?
5. Rules, phases, inverse tested?

Run the examples

```
uv run demo.py
```

Download demo.py

Lesson 3 · Course