Towards optimal synchronous counting

Christoph Lenzen
MPI for Informatics

Joel Rybicki
MPI for Informatics
Aalto University

Jukka Suomela
Aalto University

PODC 2015
July 23

Focus on fault-tolerance

Fault-tolerant ***co-ordination*** primitives:

- **permanent** failures (*Byzantine faults*)
- **transient** failures (*self-stabilisation*)

Focus on fault-tolerance

Fault-tolerant ***co-ordination*** primitives:

- **permanent** failures (*Byzantine faults*)
- **transient** failures (*self-stabilisation*)

Find solutions that are

- **fast** to recover
- **space** and **communication**-efficient

Our contribution

A **deterministic** ***round counter*** with

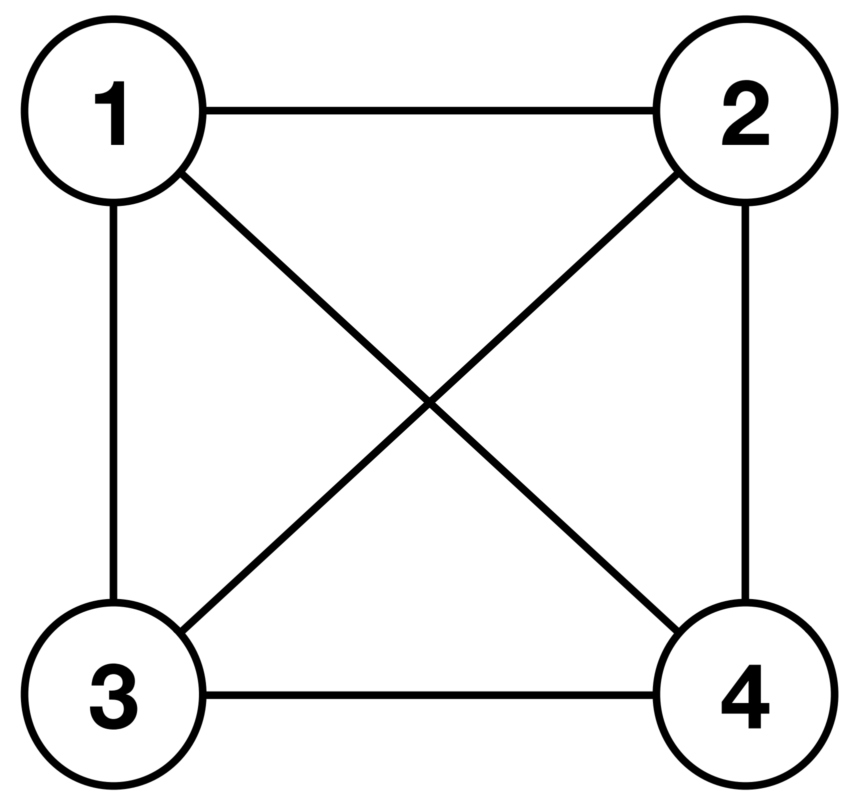
- **high** resilience
- **optimal** recovery time
- **low** space/message complexity

Model of computing

n state machines

Synchronous rounds:

1. broadcast
2. receive
3. update state

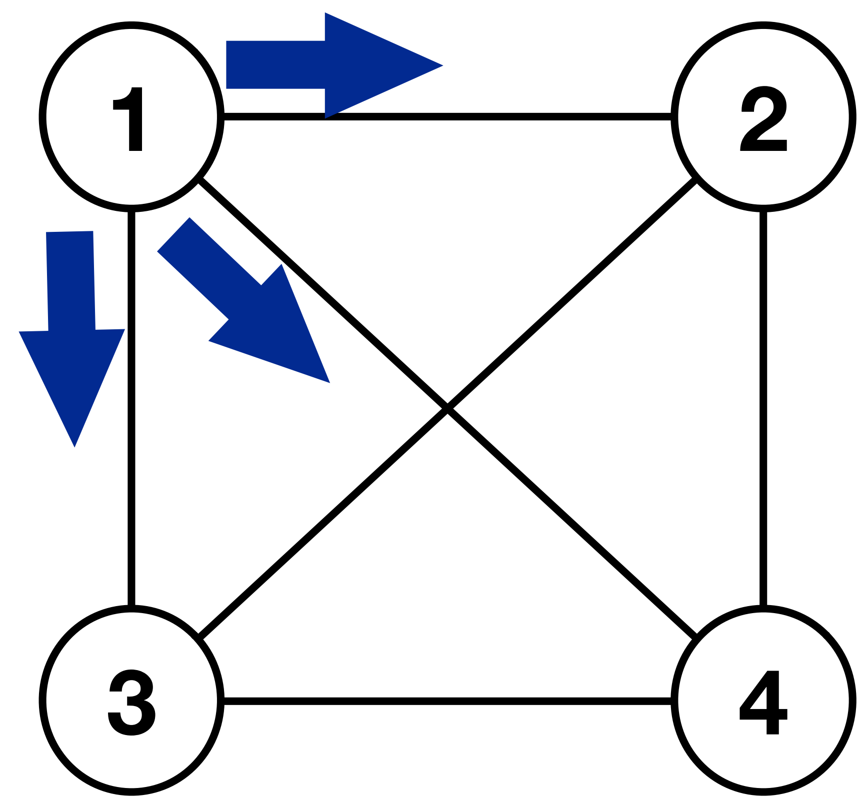


Model of computing

n state machines

Synchronous rounds:

1. **broadcast**
2. receive
3. update state

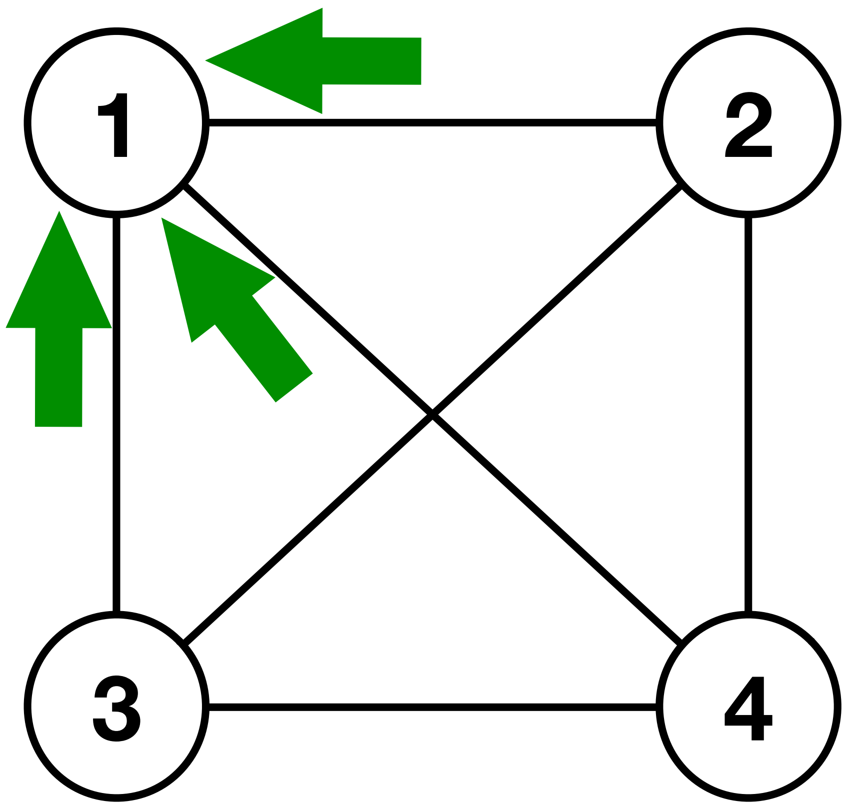


Model of computing

n state machines

Synchronous rounds:

1. broadcast
2. **receive**
3. update state

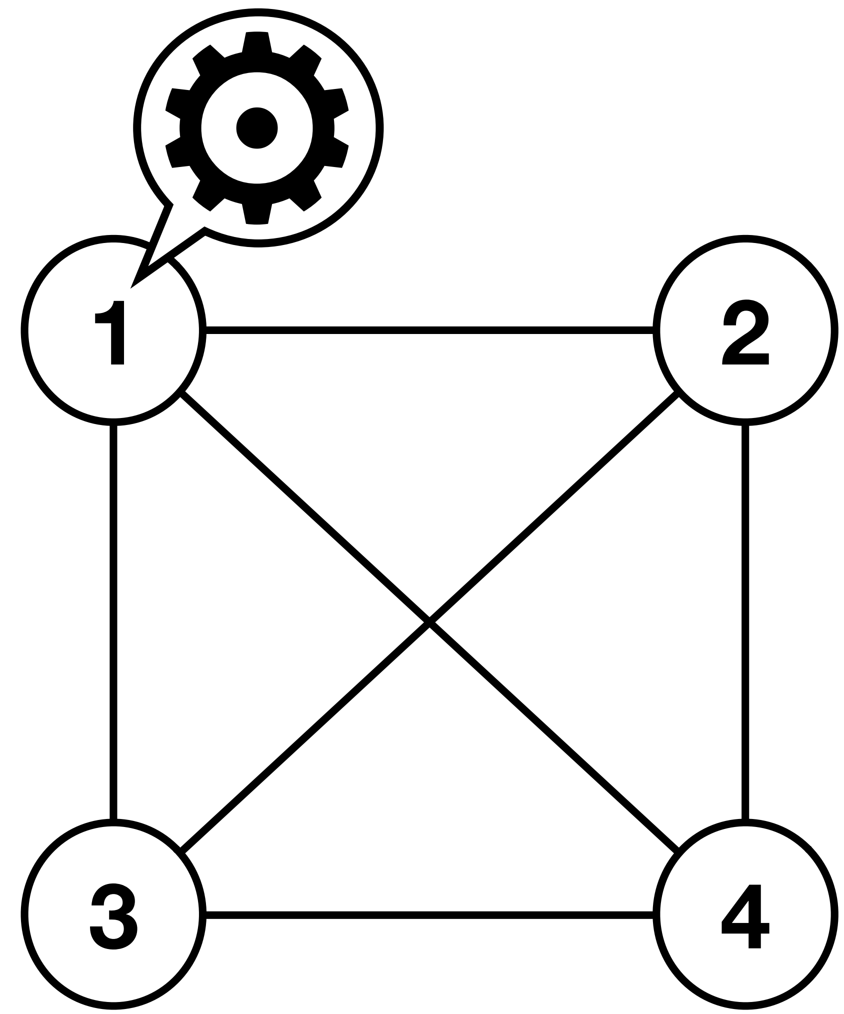


Model of computing

n state machines

Synchronous rounds:

1. broadcast
2. receive
3. **update state**



Algorithm **A** maps a *vector* of states to a new state!

Complexity measures

Time complexity: #rounds
≈ “recovery time”

Space complexity: log #states
≈ complexity of the circuit
≈ number of bits broadcast per node

On failures

our adversary

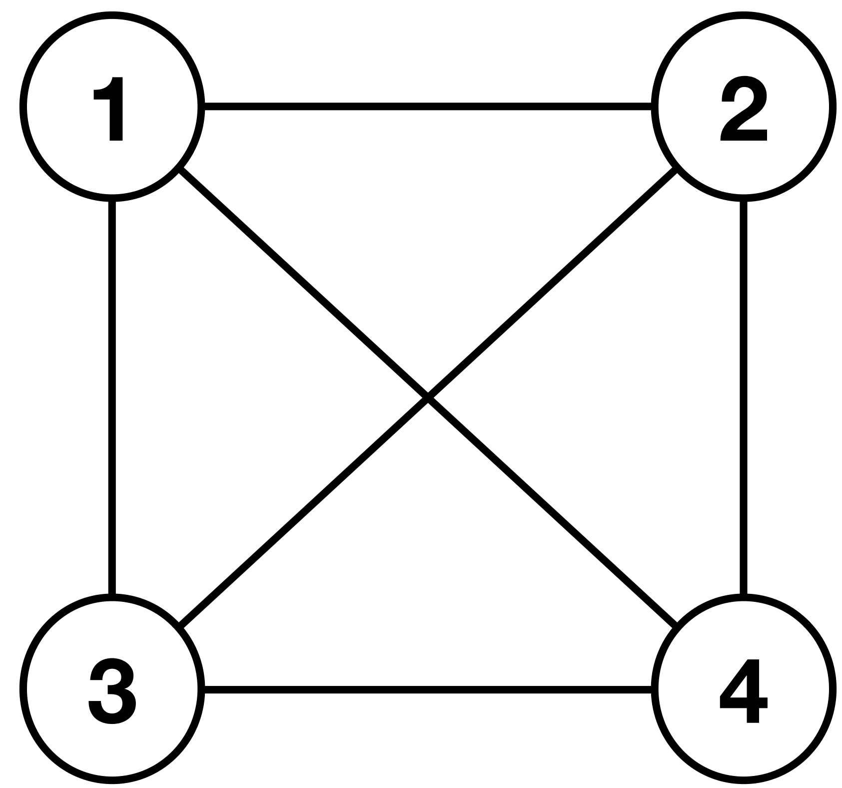
Transient failures

n state machines

arbitrary initial states

chosen by adversary!

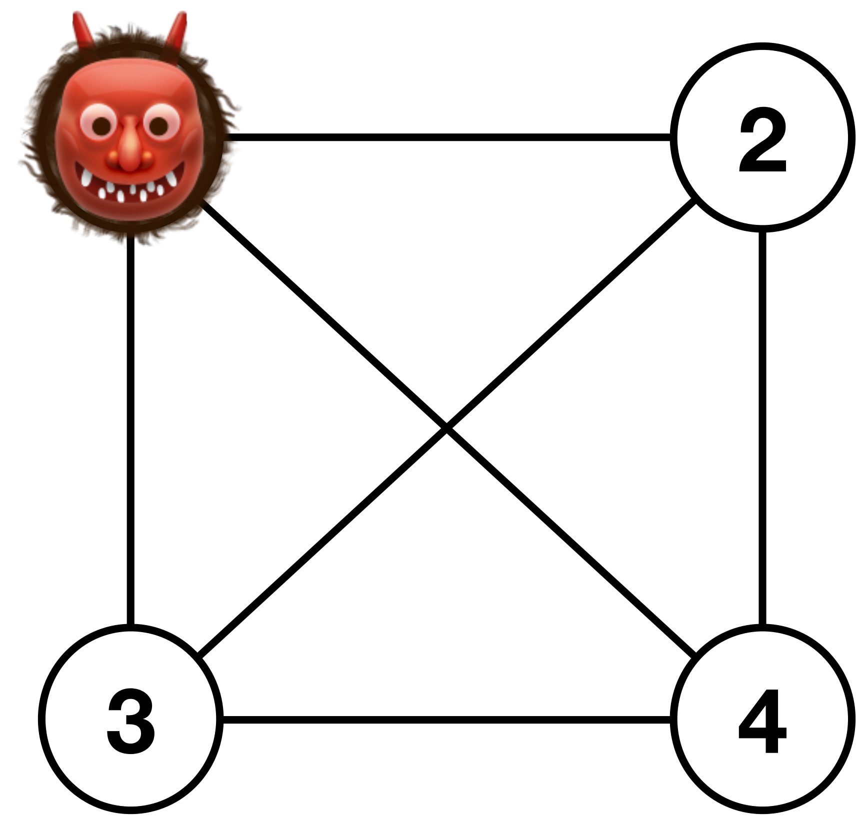
= self-stabilisation



Byzantine failures

n state machines

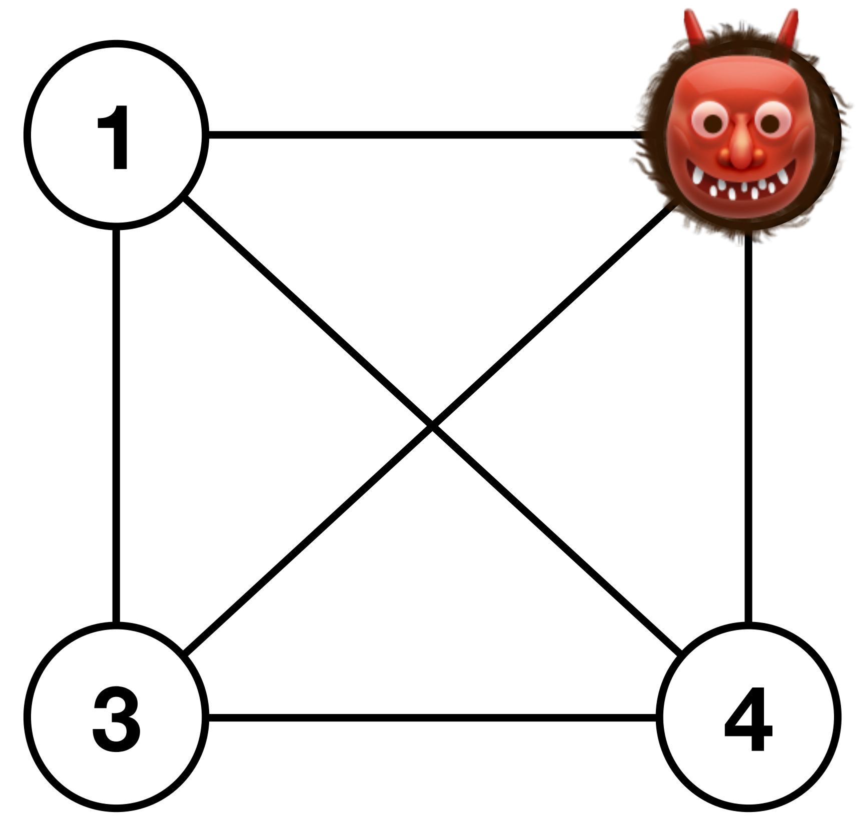
f Byzantine failures



Byzantine failures

n state machines

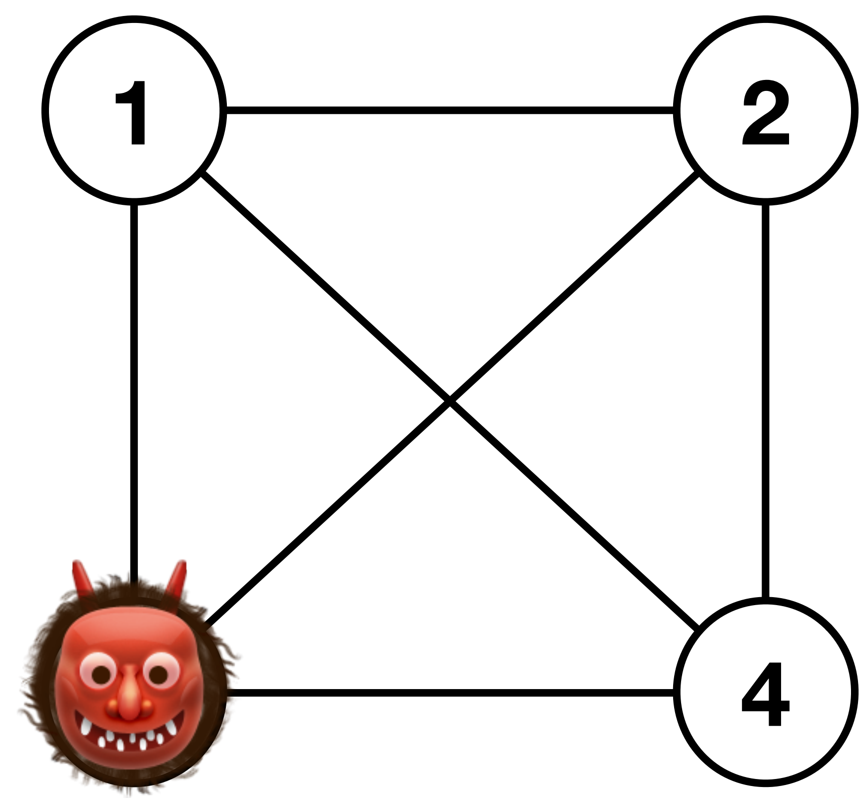
f Byzantine failures



Byzantine failures

n state machines

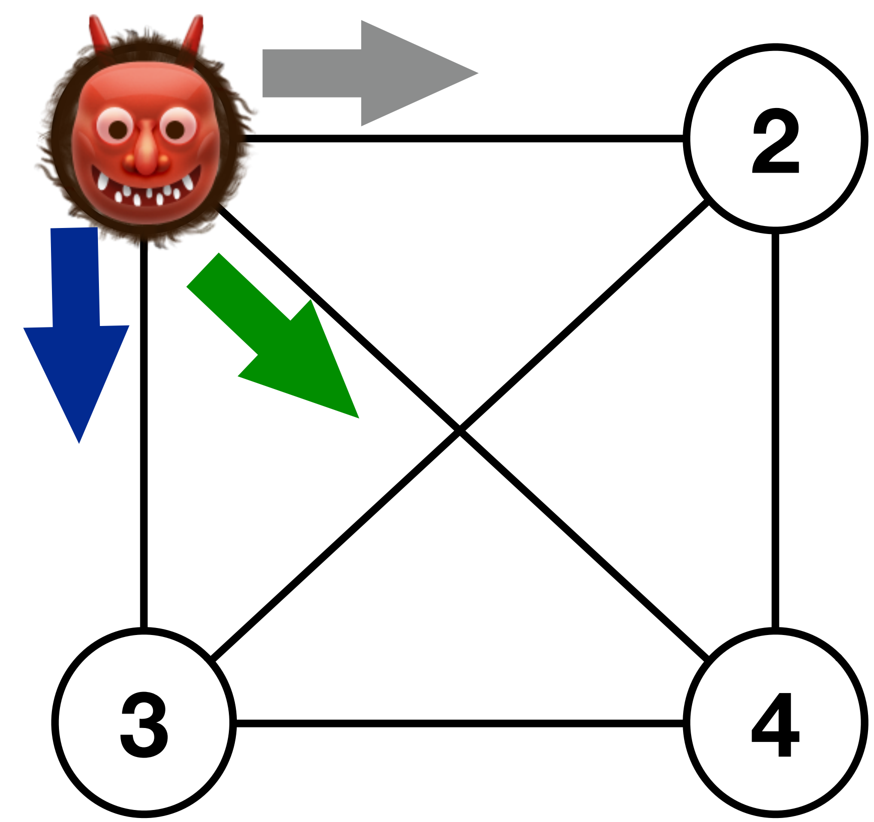
f Byzantine failures



Byzantine failures

n state machines

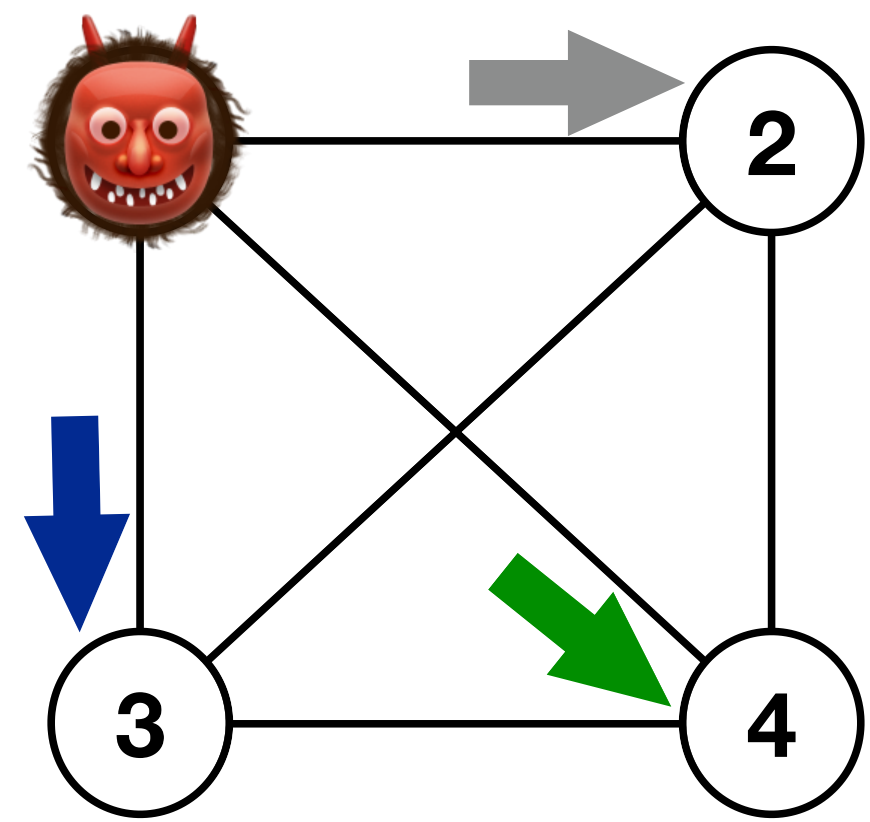
f Byzantine failures



Byzantine failures

n state machines

f Byzantine failures



Non-faulty nodes can observe *different* states for the system!

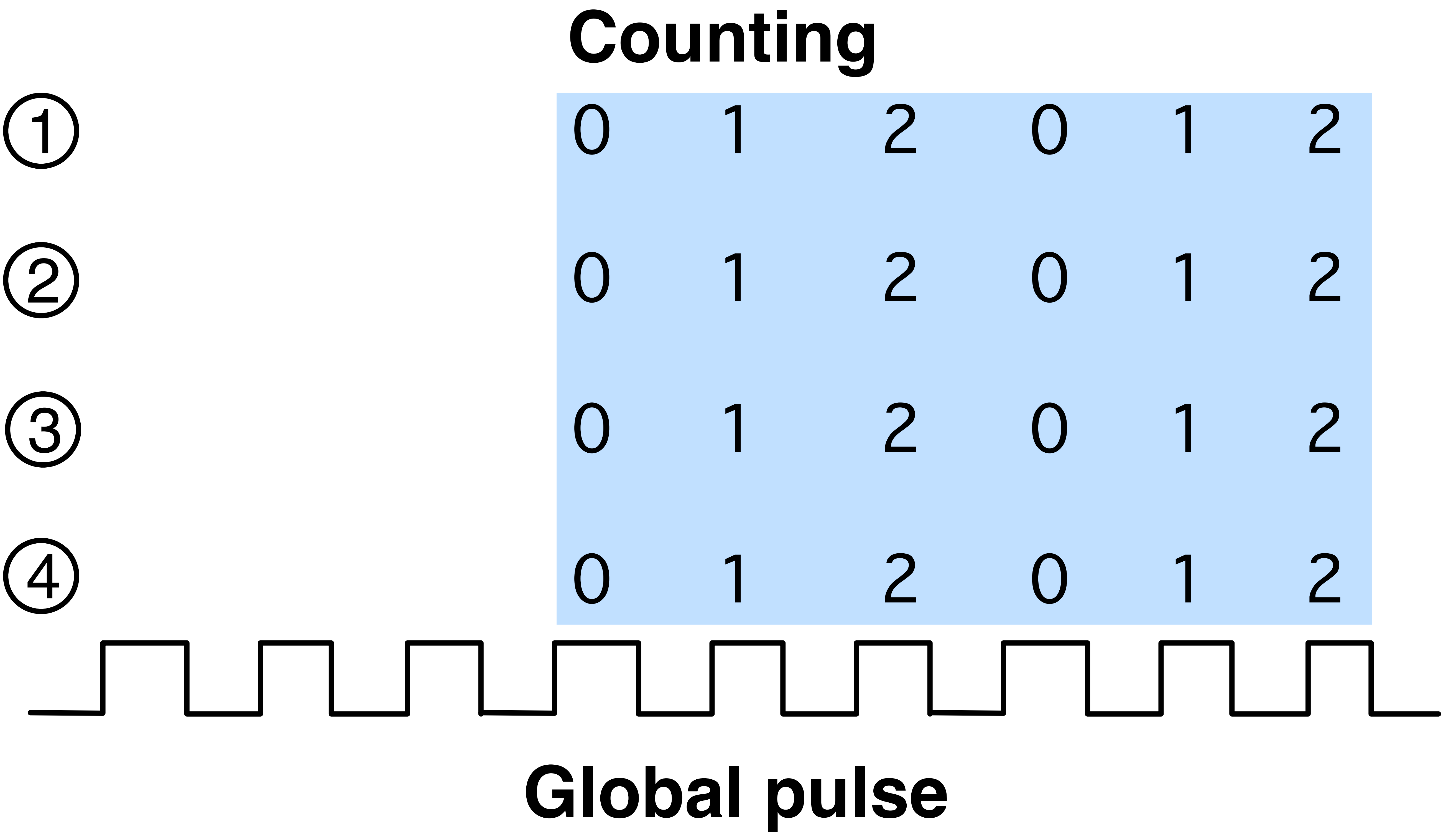
Counting mod c

3-counting

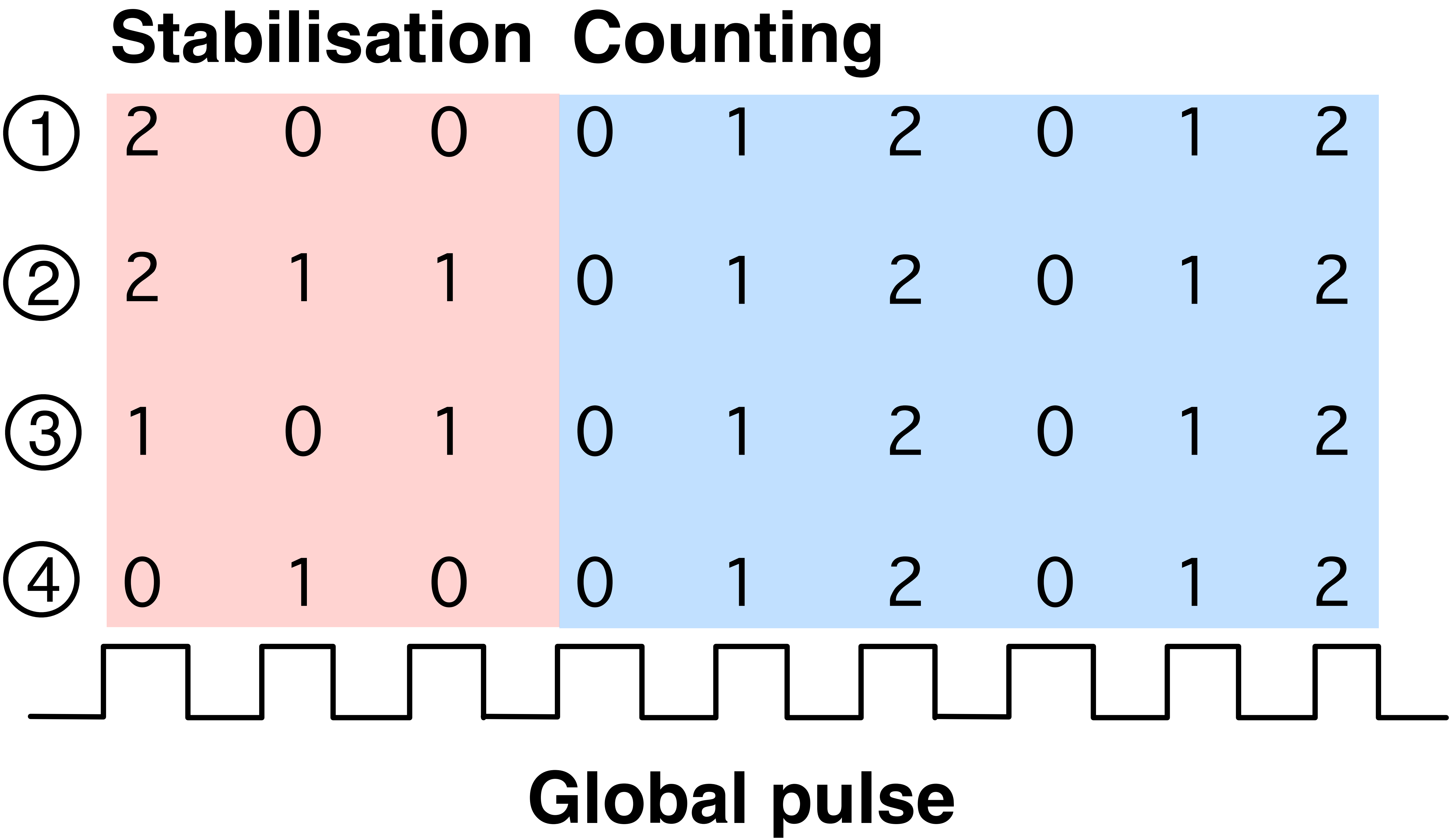
0	1	2	0	1	2

increment counter +1 mod *c*

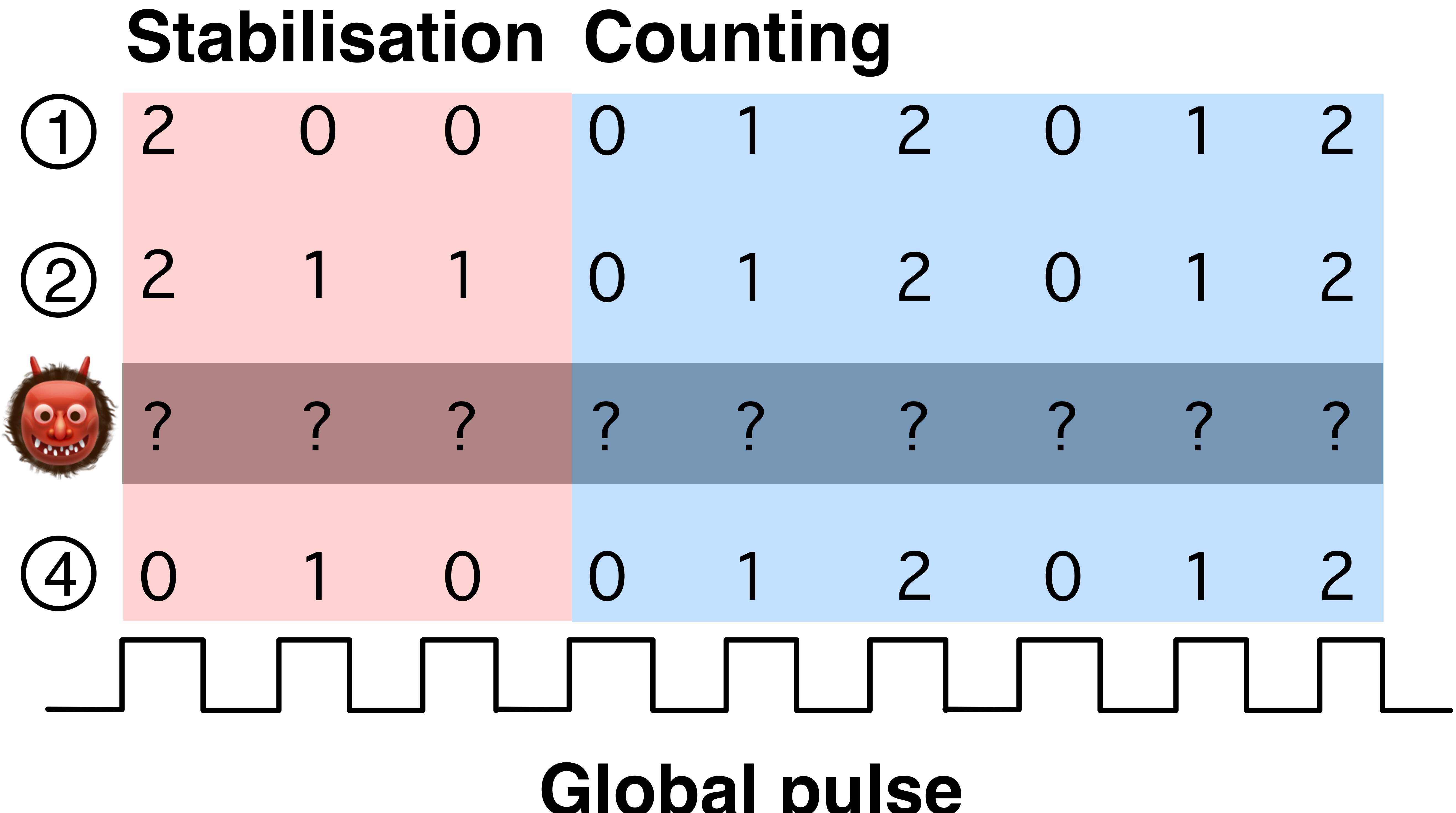
Synchronous counting



Synchronous counting



Synchronous counting



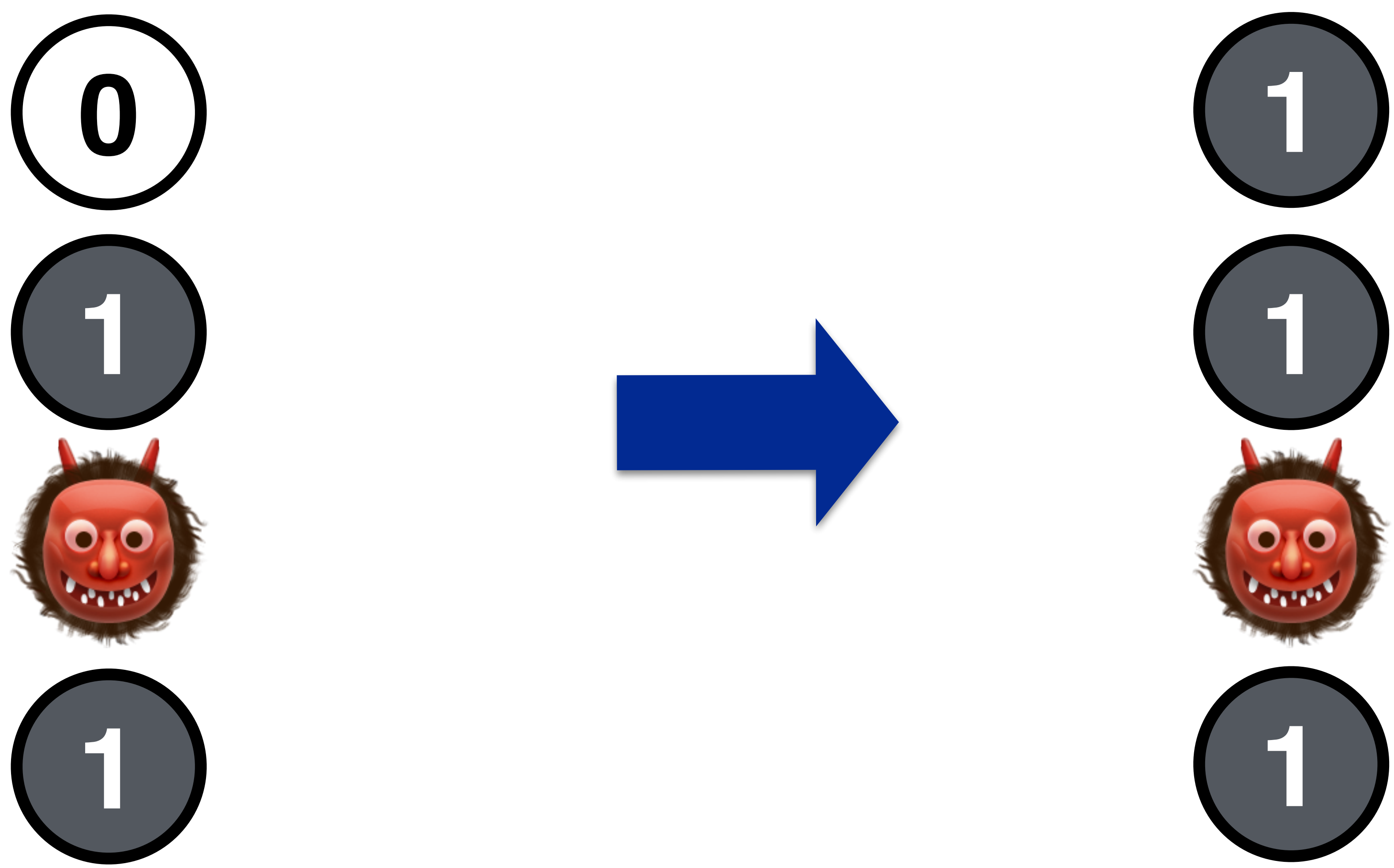
A related problem: Consensus

Input: private bit **Output:** agreement

A related problem: Consensus

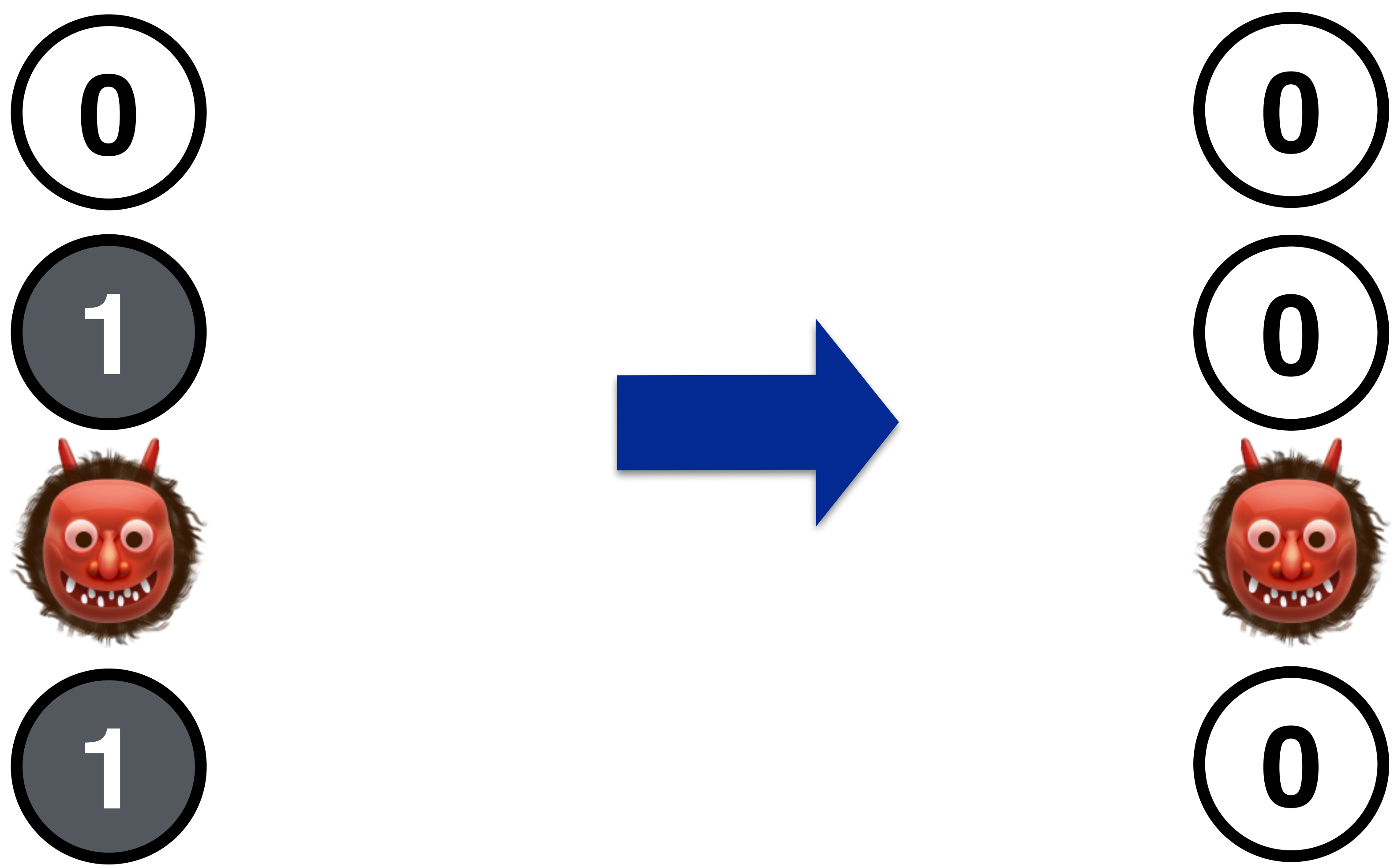
Input: private bit

Output: agreement



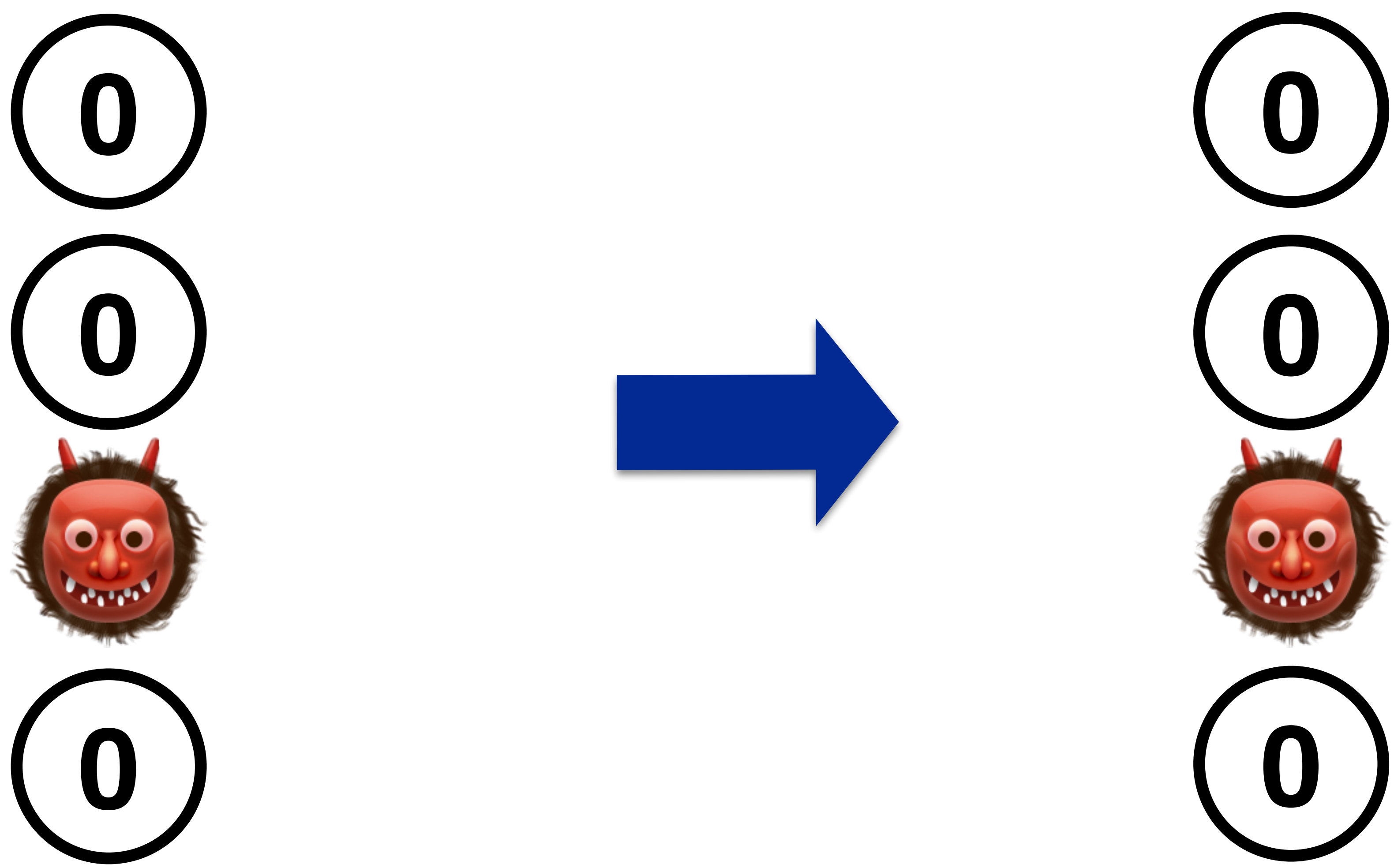
A related problem: Consensus

Input: private bit **Output:** agreement



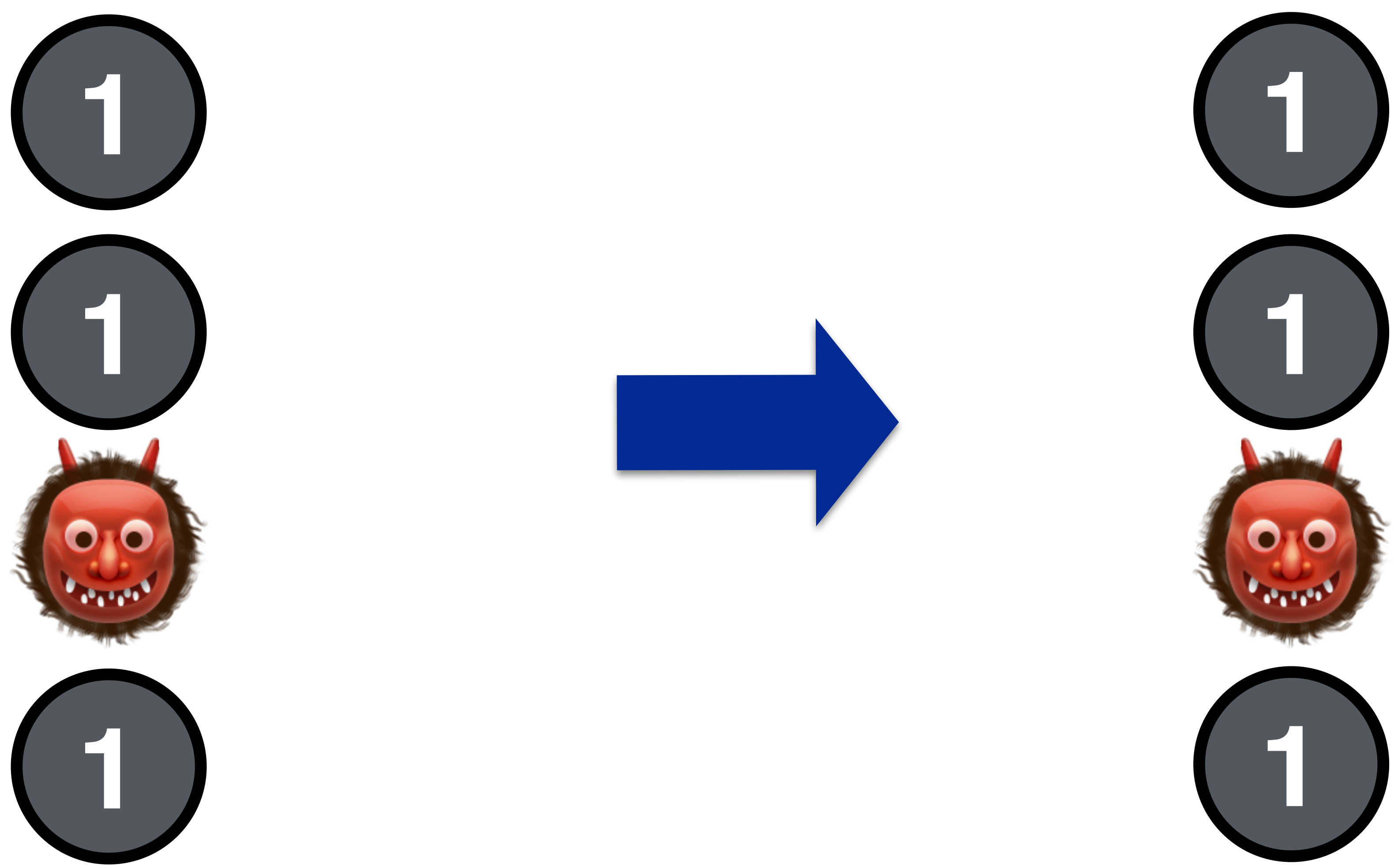
A related problem: Consensus

Input: private bit **Output:** agreement



A related problem: Consensus

Input: private bit **Output:** agreement



Reduction from consensus

Given a 2-counting algorithm
A that stabilises in ***t*** rounds

$\Rrightarrow$

consensus solvable in ***t*** rounds

Dolev et al. (2013)

Consensus lower bounds*

Resilience

$$f < n/3$$

Pease et al. (1980)

Time

At least f rounds to reach agreement

Fischer & Lynch (1982)

***deterministic**

Prior work on 2-counting

	Resilience	Time	State bits
D. Dolev & Hoch 2007 *	$f < n/3$	$O(f)$	$O(f \log f)$

***deterministic**

Prior work on 2-counting

	Resilience	Time	State bits
D. Dolev & Hoch 2007 *	$f < n/3$	$O(f)$	$O(f \log f)$
S. Dolev & Welch 2004	$f < n/3$	$2^{2(n-f)}$	$O(1)$

***deterministic**

Prior work on 2-counting

	Resilience	**Time**	**State bits**
D. Dolev & Hoch 2007 *	$f < n/3$	$O(f)$	$O(f \log f)$
S. Dolev & Welch 2004	$f < n/3$	$2^{2(n-f)}$	$O(1)$
Ben-Or *et al.* 2008	$f < n/3$	$O(1)$	$n^{O(1)}$

***deterministic**

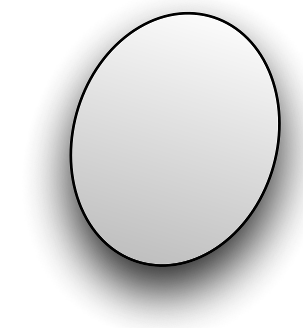
Current state

	Resilience	Time	State bits
D. Dolev & Hoch 2007 *	$f < n/3$	$O(f)$	$O(f \log f)$
S. Dolev & Welch 2004	$f < n/3$	$2^{2(n-f)}$	$O(1)$
Ben-Or *et al.* 2008	$f < n/3$	$O(1)$	$n^{O(1)}$
Our work*	$f = n^{1-o(1)}$	$O(f)$	$o(\log^2 f)$

***deterministic**

Counting vs consensus

?



Counting

Consensus

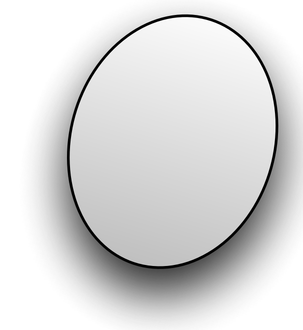
Counting vs consensus

🐔 🍳

Counting $\Leftarrow$ **Consensus**

Solve consensus to agree on counters

Counting vs consensus

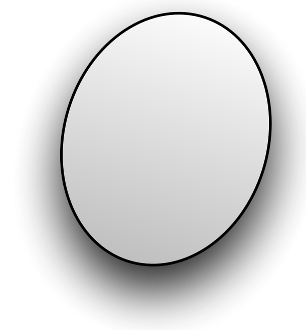


Counting $\Rightarrow$ **Consensus**

Use a *c*-counter as a round counter;
execute a consensus algorithm

Counting vs consensus

?

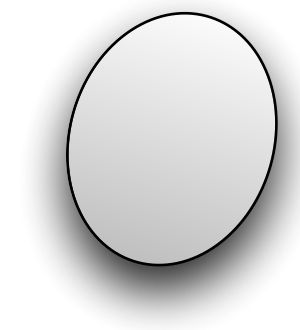


Counting

Consensus

Counting vs consensus

!



Counting **Consensus**

Solution: counters that work *once in a while*

Counting once in a while

①	2	0	0	1	2	2	0	0	0	1	2
②	2	1	0	1	2	0	1	2	0	1	2
👹	?	?	?	?	?	?	?	?	?	?	?
④	0	1	0	1	2	0	0	1	0	1	2
	Arbitrary		**Counting**			**Arbitrary**			**Counting**		

Counting once in a while

Use *proper* counters with **low resilience**, to count *once in a while* with **high resilience!**

Counting

low resilience

Counting
once in a while
high resilience

Clock for consensus

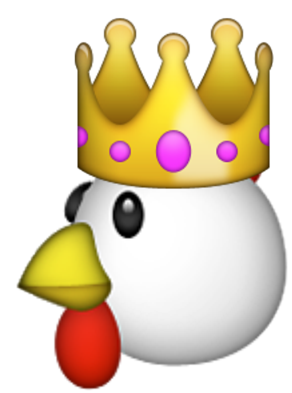
If we can count *once in a while* with **high resilience..**

then we can execute a **highly-resilient consensus** algorithm (*phase king*)

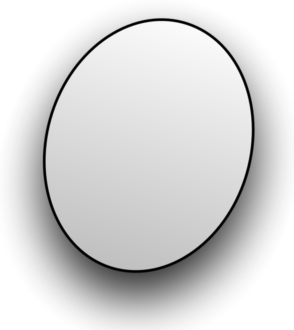
Berman et al. (1989)

Agree on a new counter

Use consensus protocol
with **high resilience**, to agree on a
***new proper* counter!**

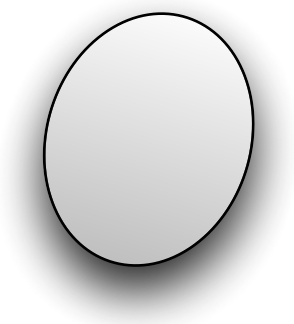


High-level idea



Counting
low resilience

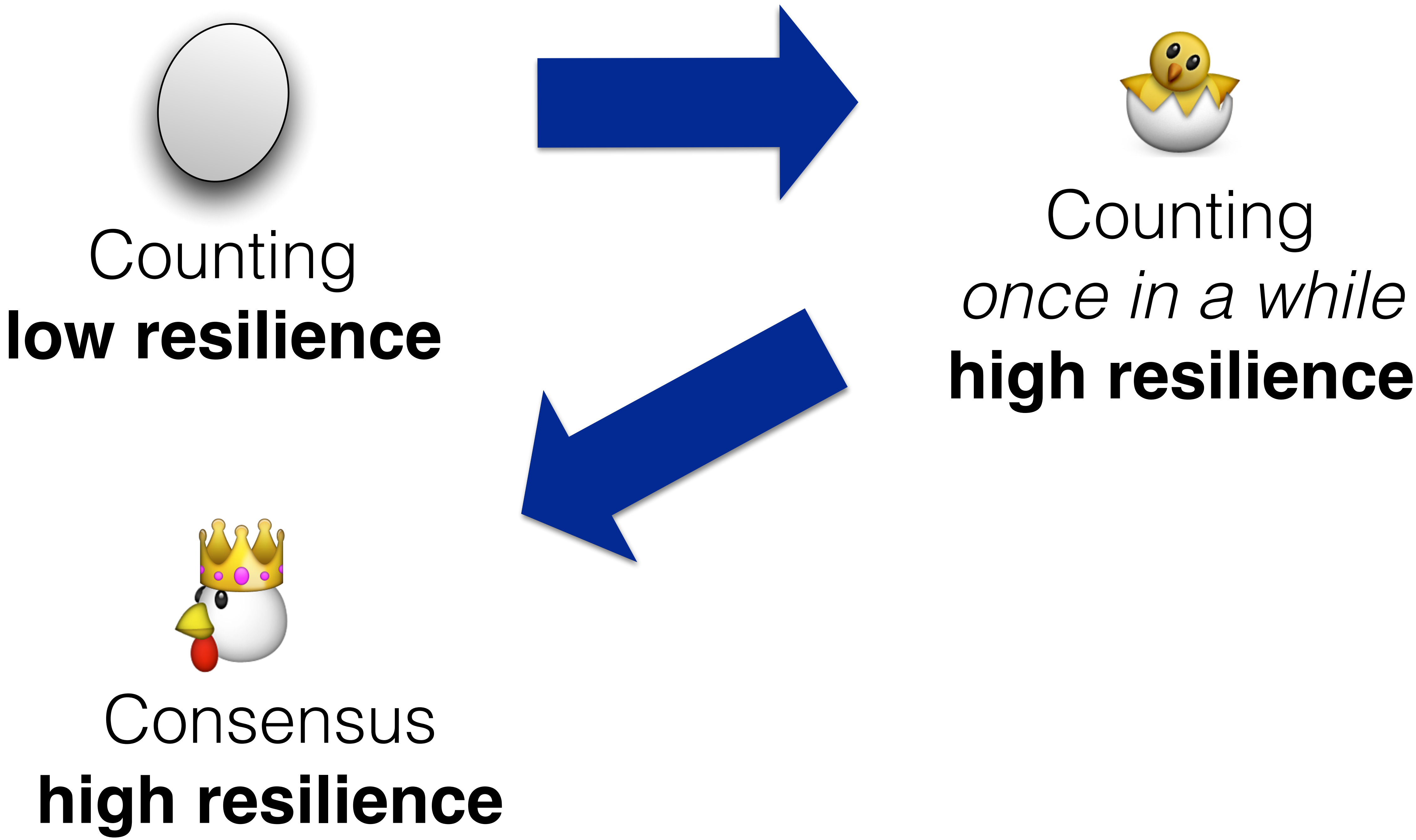
High-level idea



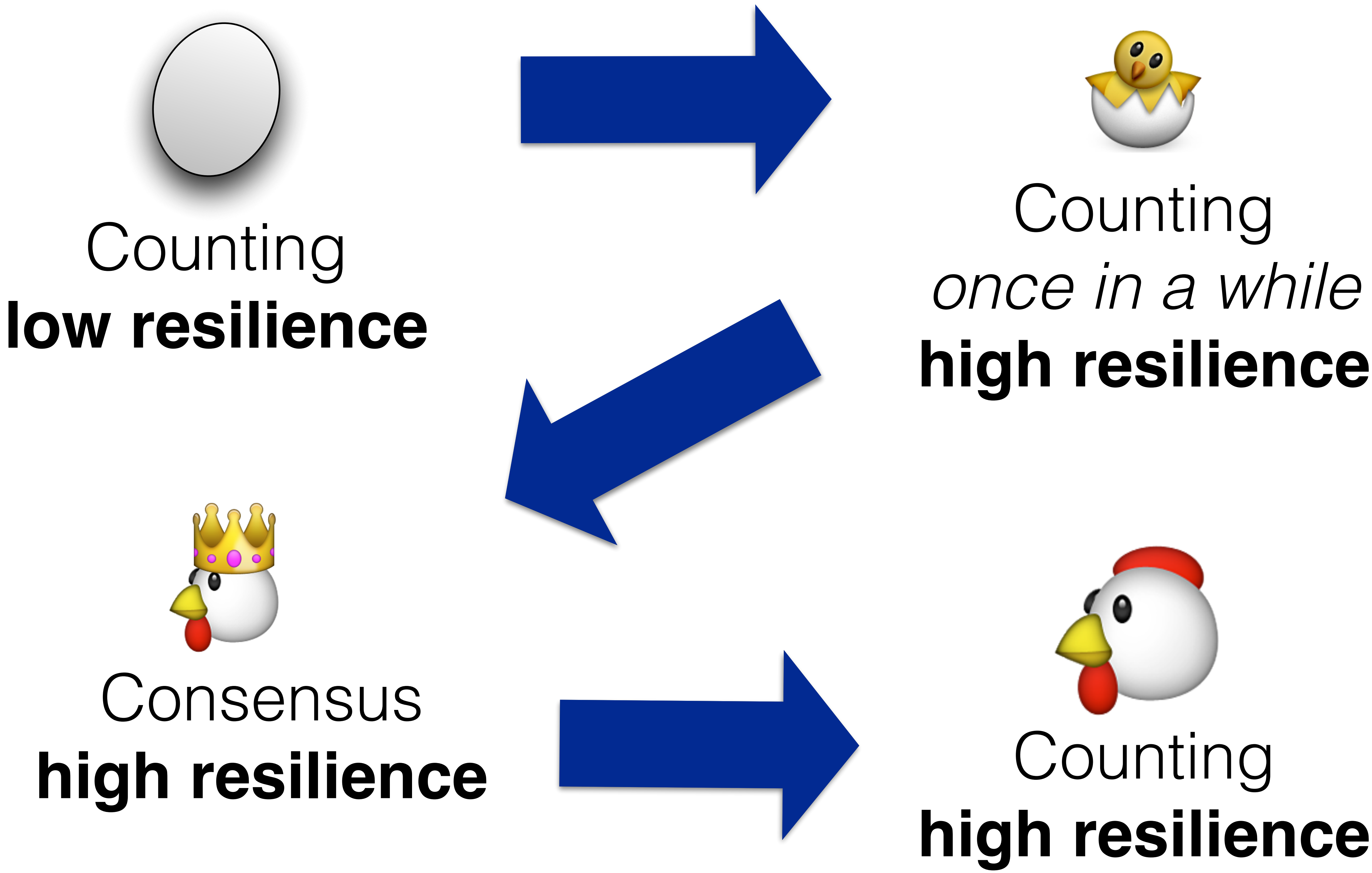
Counting
low resilience

Counting
once in a while
high resilience

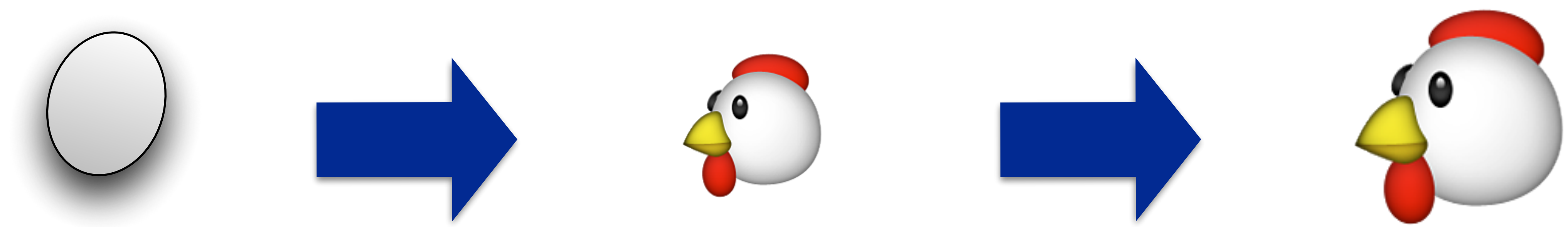
High-level idea



High-level idea



Rinse and repeat



low resilience

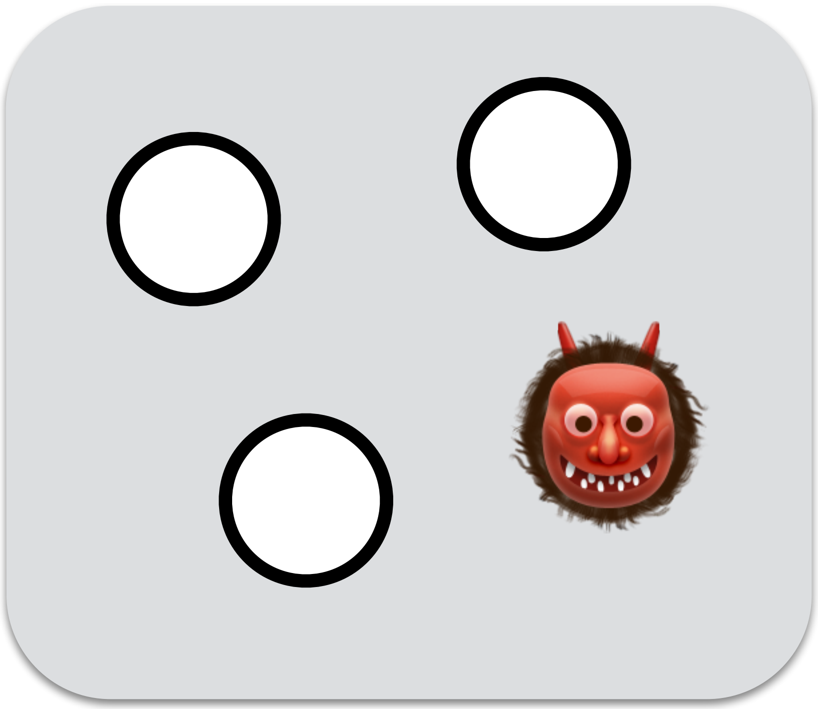
high resilience

***higher* resilience**

The boosting theorem

More formally…

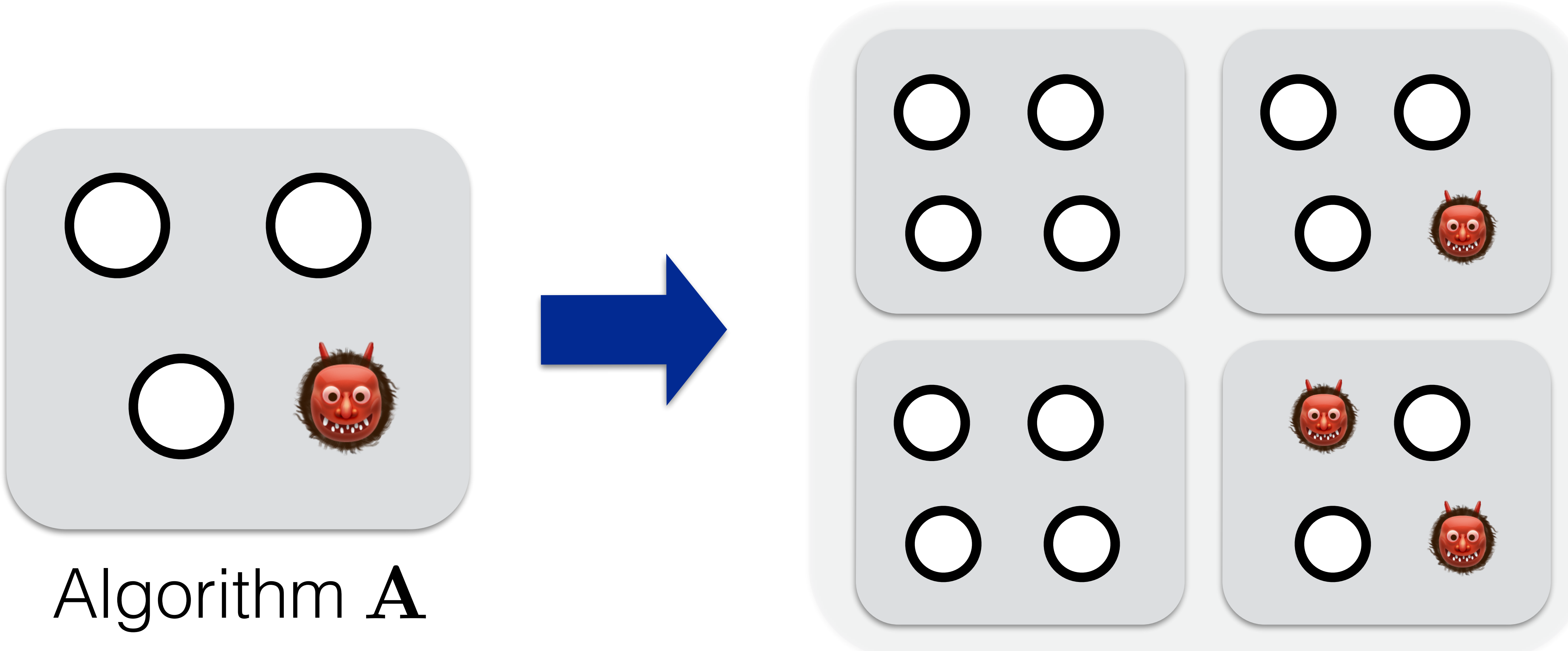
The boosting theorem



Algorithm $\mathbf{A}$

n nodes
f faults

The boosting theorem



Algorithm $\mathbf{A}$

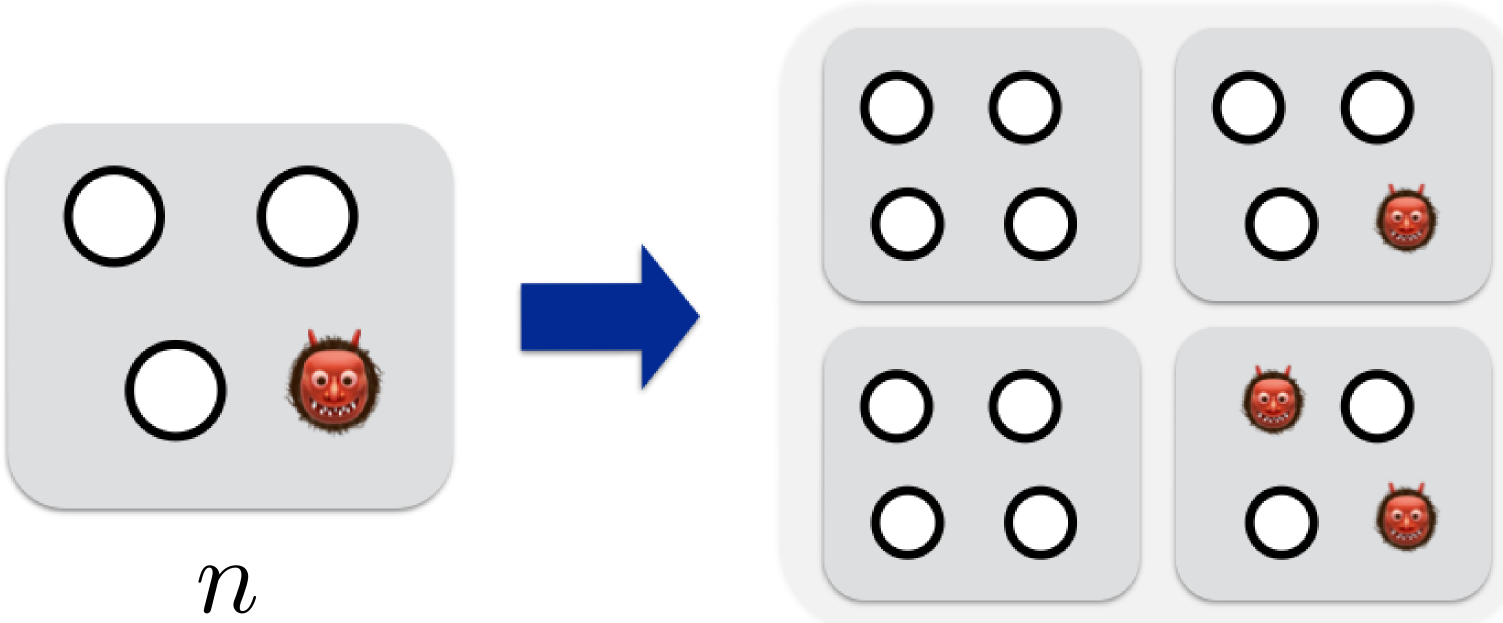
n nodes
f faults

Algorithm $\mathbf{B}$

$N = kn$

$F \approx (f+1)k/2$

The boosting theorem



n
f

$N = kn$

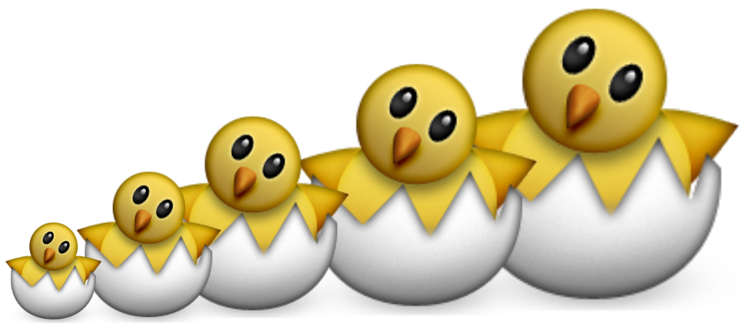
$F \approx (f+1)k/2$

Stabilisation time: $T(\mathbf{B}) = T(\mathbf{A}) + O(Fk^k)$

Space complexity: $S(\mathbf{B}) = S(\mathbf{A}) + O(\log C)$

C = new counter range

Boosting resilience



Boost resilience recursively
while keeping **time** and **space** complexity *small enough*

Main result

Resilience

$$f = n^{1-o(1)}$$

Stabilisation time

$$O(f)$$

Space complexity

$$O(\log^2 f / \log\log f)$$

Main result

Resilience

$$f = n^{1-o(1)}$$

Stabilisation time

$$O(f)$$

Space complexity

$$O(\log^2 f / \log\log f)$$

Thanks!