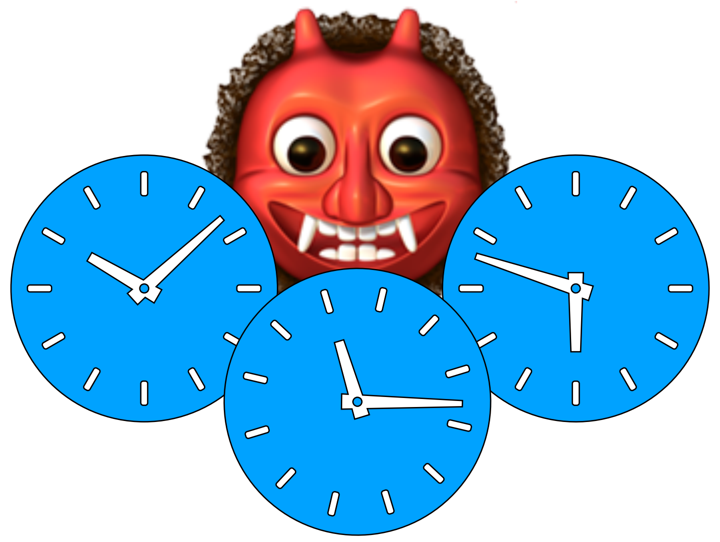
Self-stabilising Byzantine clock synchronisation is almost as easy as consensus

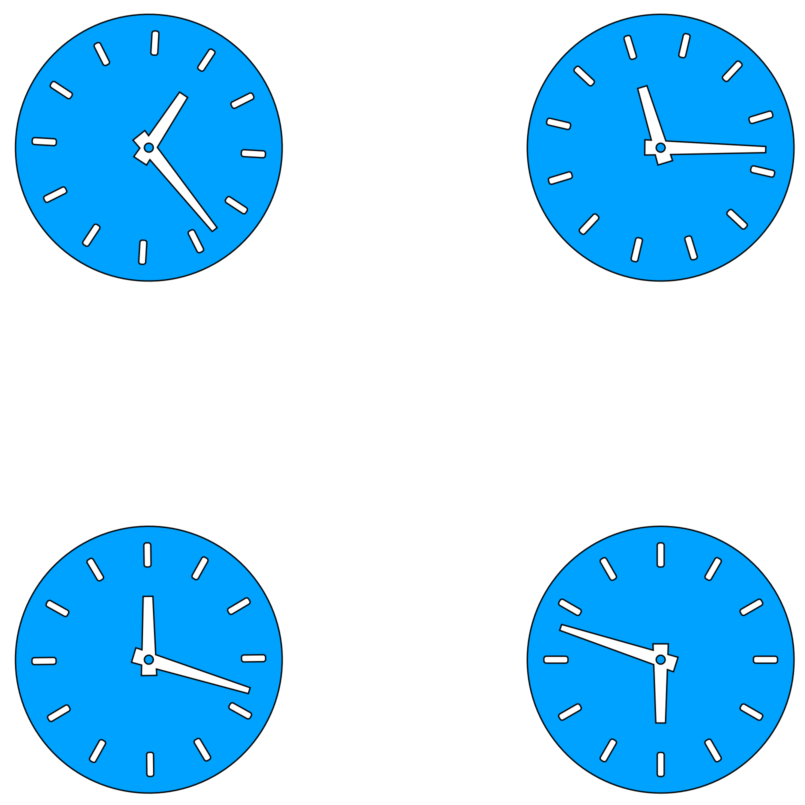
Christoph Lenzen
Max Planck Institute for Informatics

Joel Rybicki
University of Helsinki

DISC 2017
October 17, 2017
Vienna, Austria

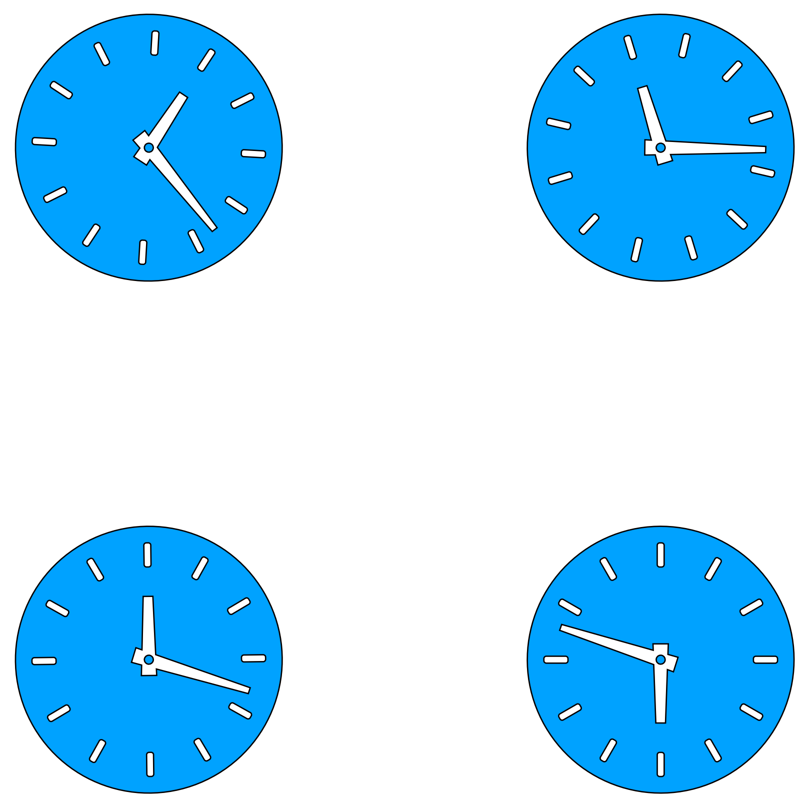


Clock synchronisation



Question:
How to efficiently synchronise a network of **imprecise clocks** in a **fault-tolerant** manner?

Clock synchronisation

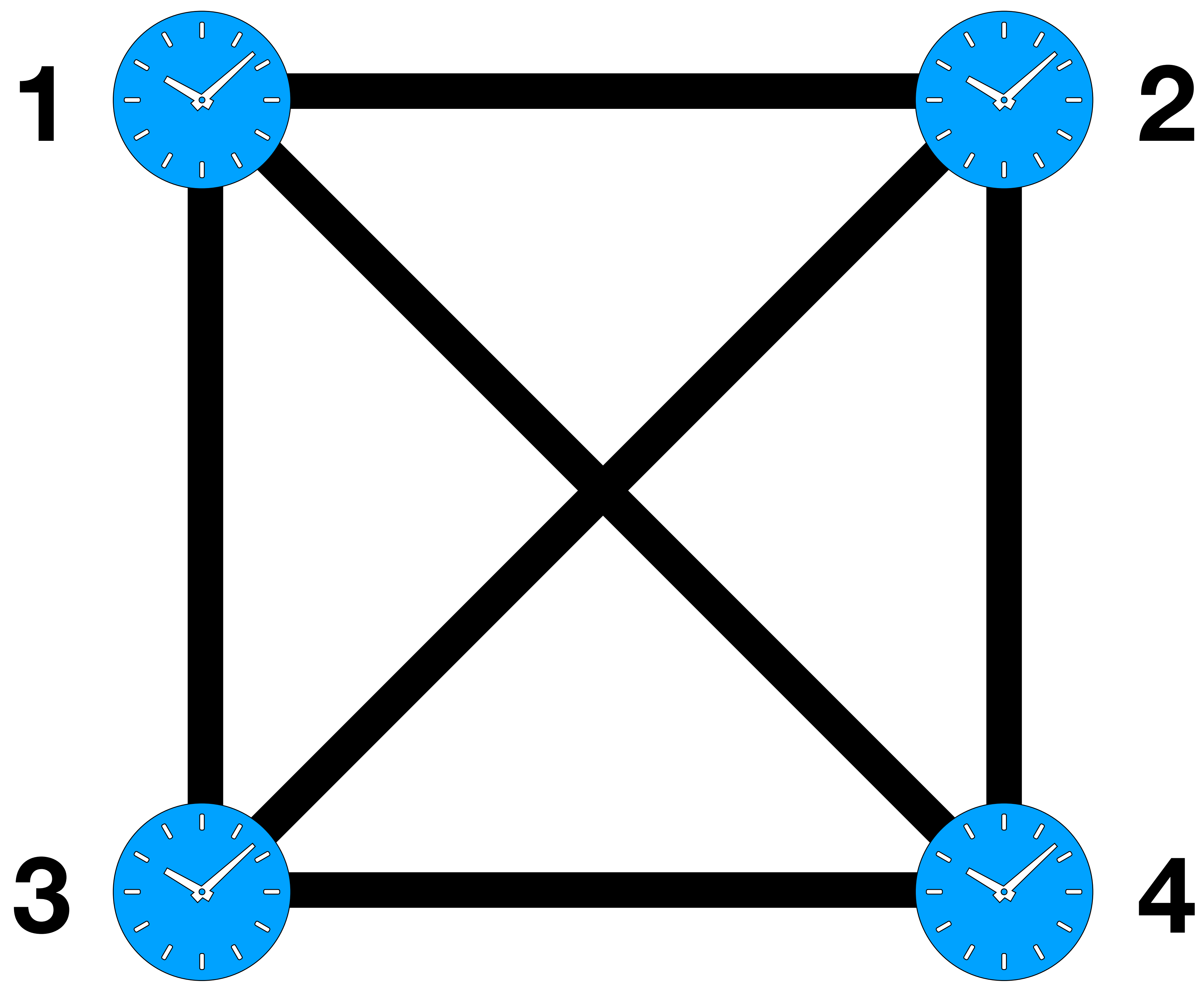


Question:
How to efficiently synchronise a network of **imprecise clocks** in a **fault-tolerant** manner?

Real clocks exhibit **drift:** even if started at the same time, values eventually differ

Distributed systems are prone to both **transient faults** and **permanent faults**

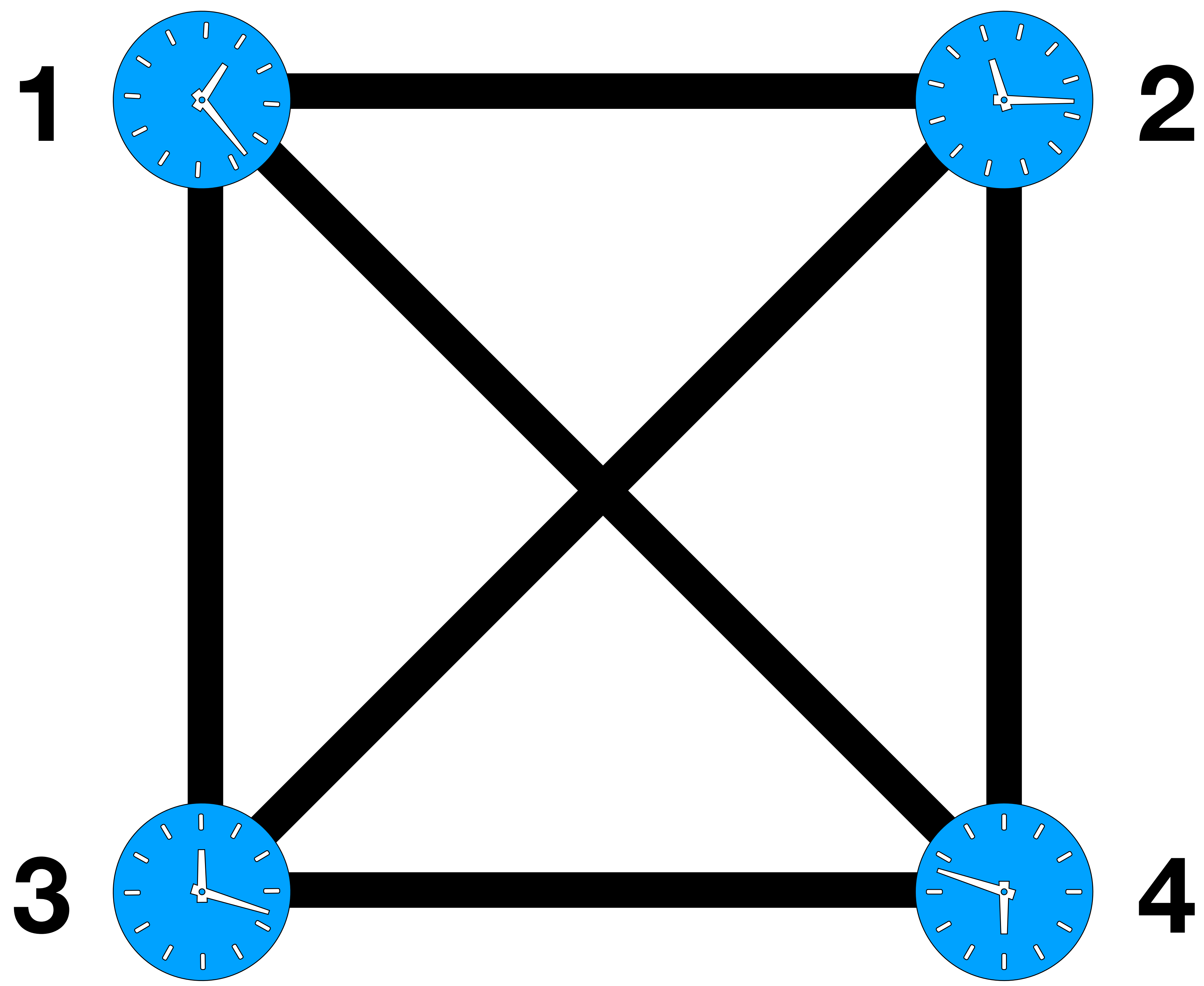
Model of computation



- n nodes with **local clocks**

Bounded clock drift:
fastest clock progresses
at most factor $\vartheta > 1$ faster
than the slowest clock

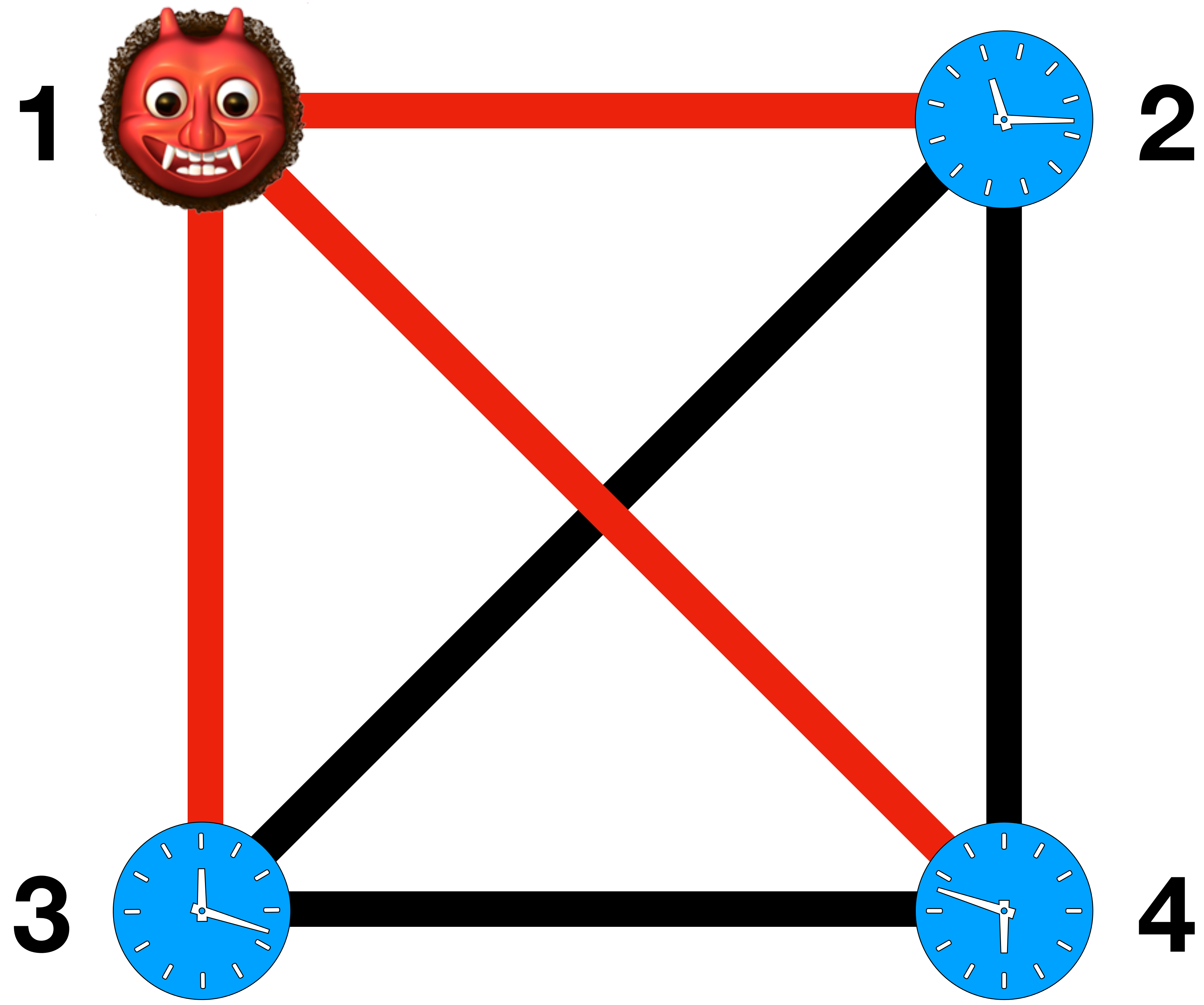
Model of computation



- *n* nodes with **local clocks**
- arbitrary initial state

Self-stabilisation:
Initial clock values and local state are arbitrary

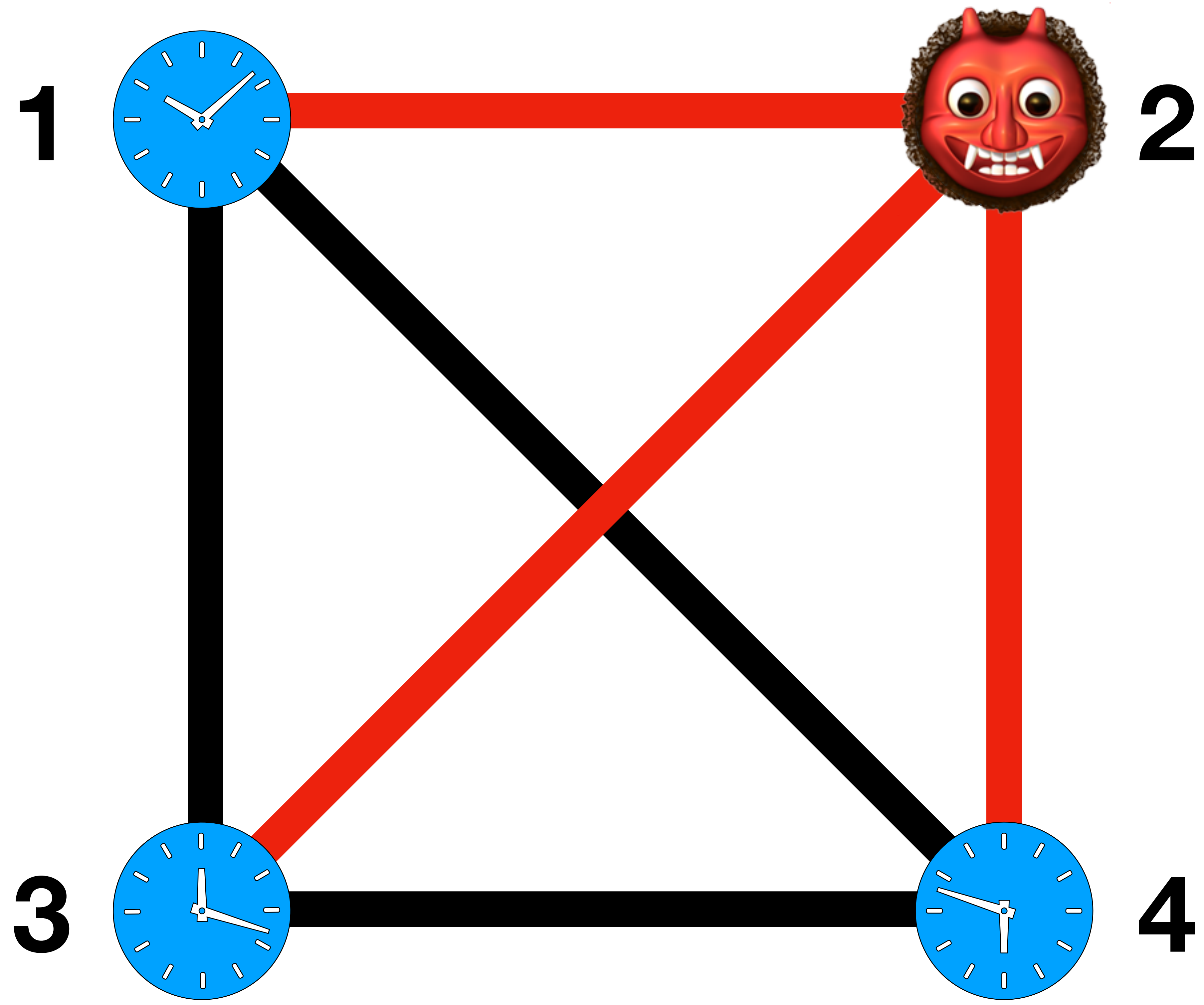
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes

Arbitrary misbehaviour:
can crash, lose messages,
send different misinformation
to correct nodes, etc.

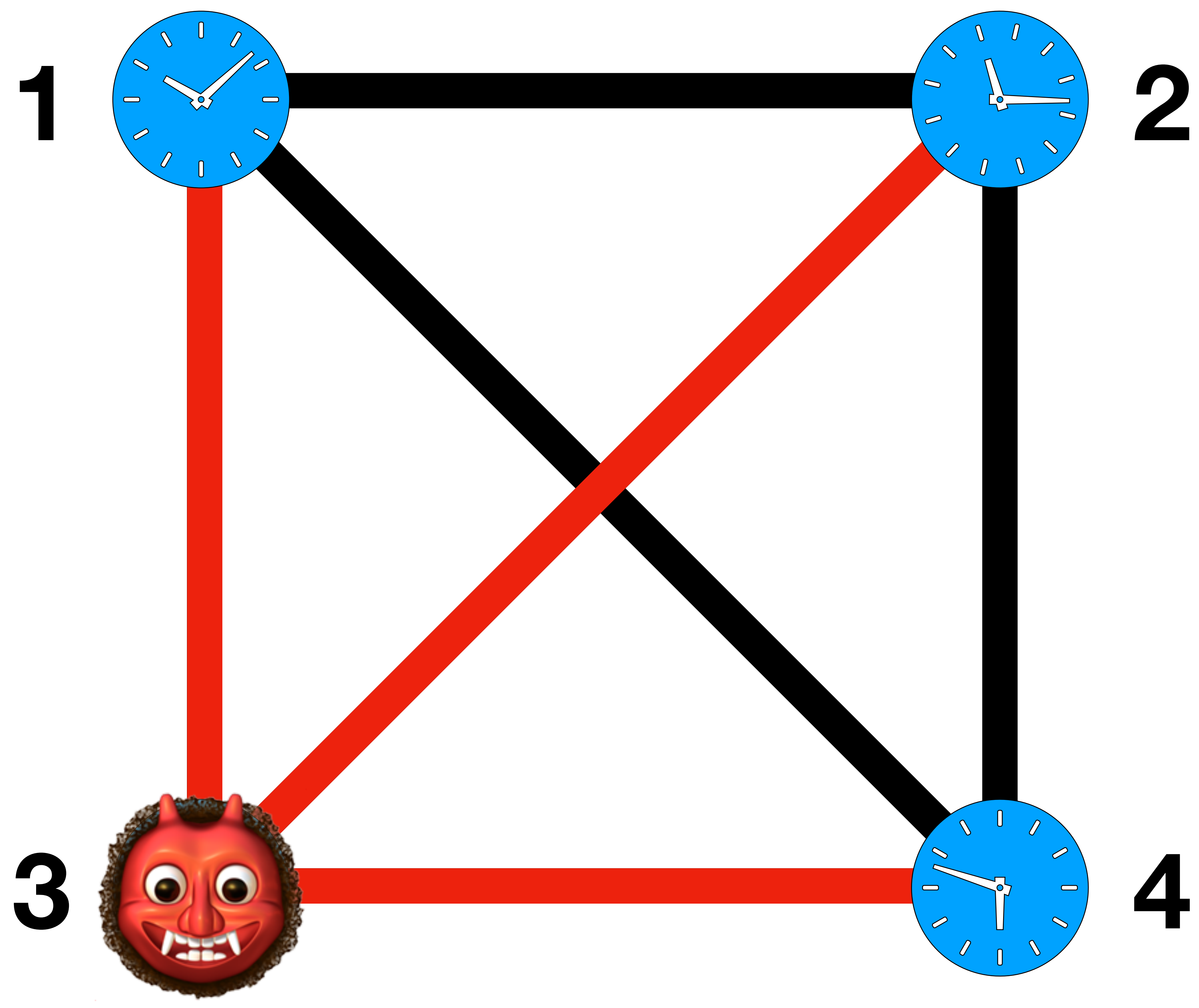
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes

Arbitrary misbehaviour:
can crash, lose messages, send different misinformation to correct nodes, etc.

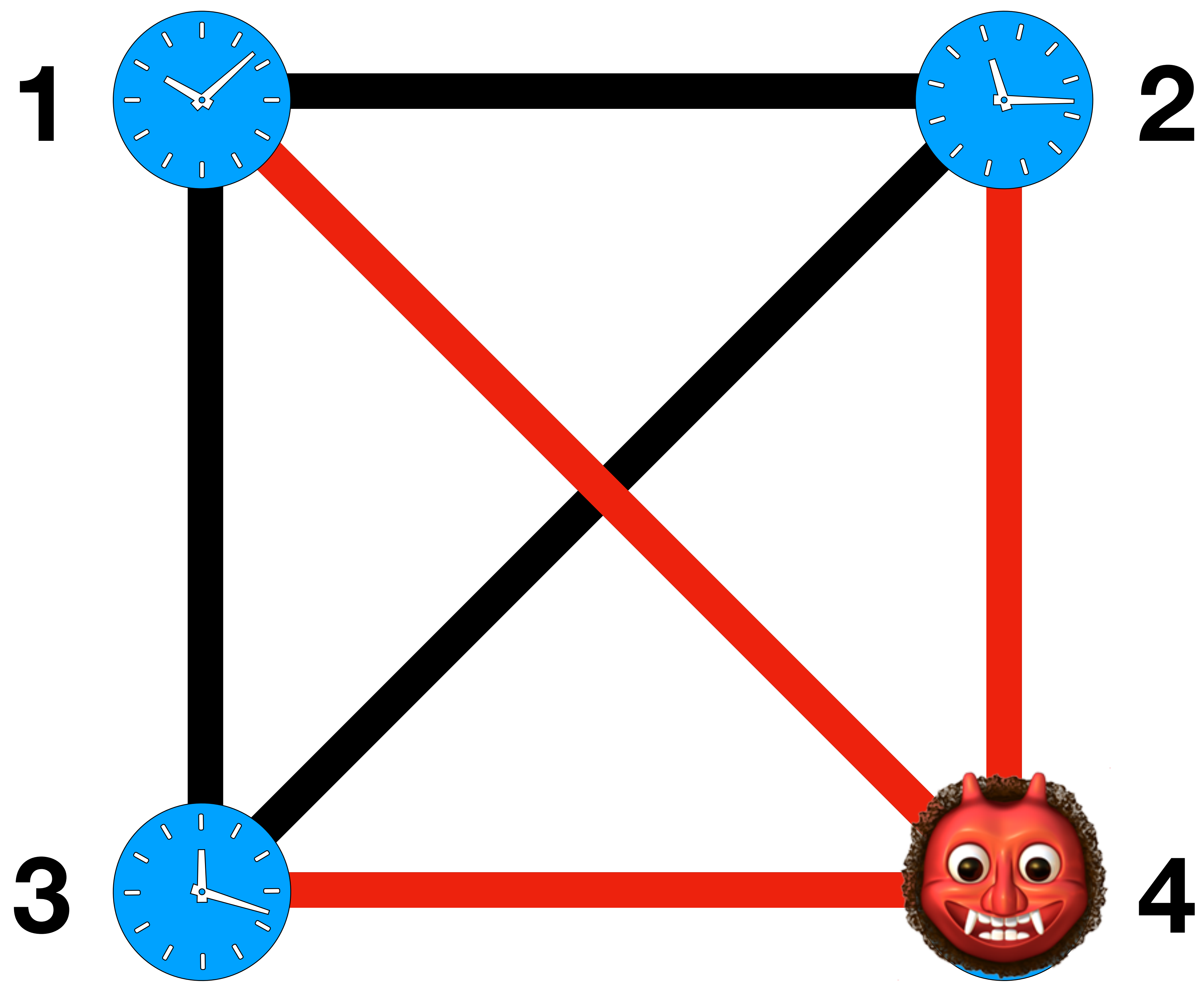
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes

Arbitrary misbehaviour:
can crash, lose messages, send different misinformation to correct nodes, etc.

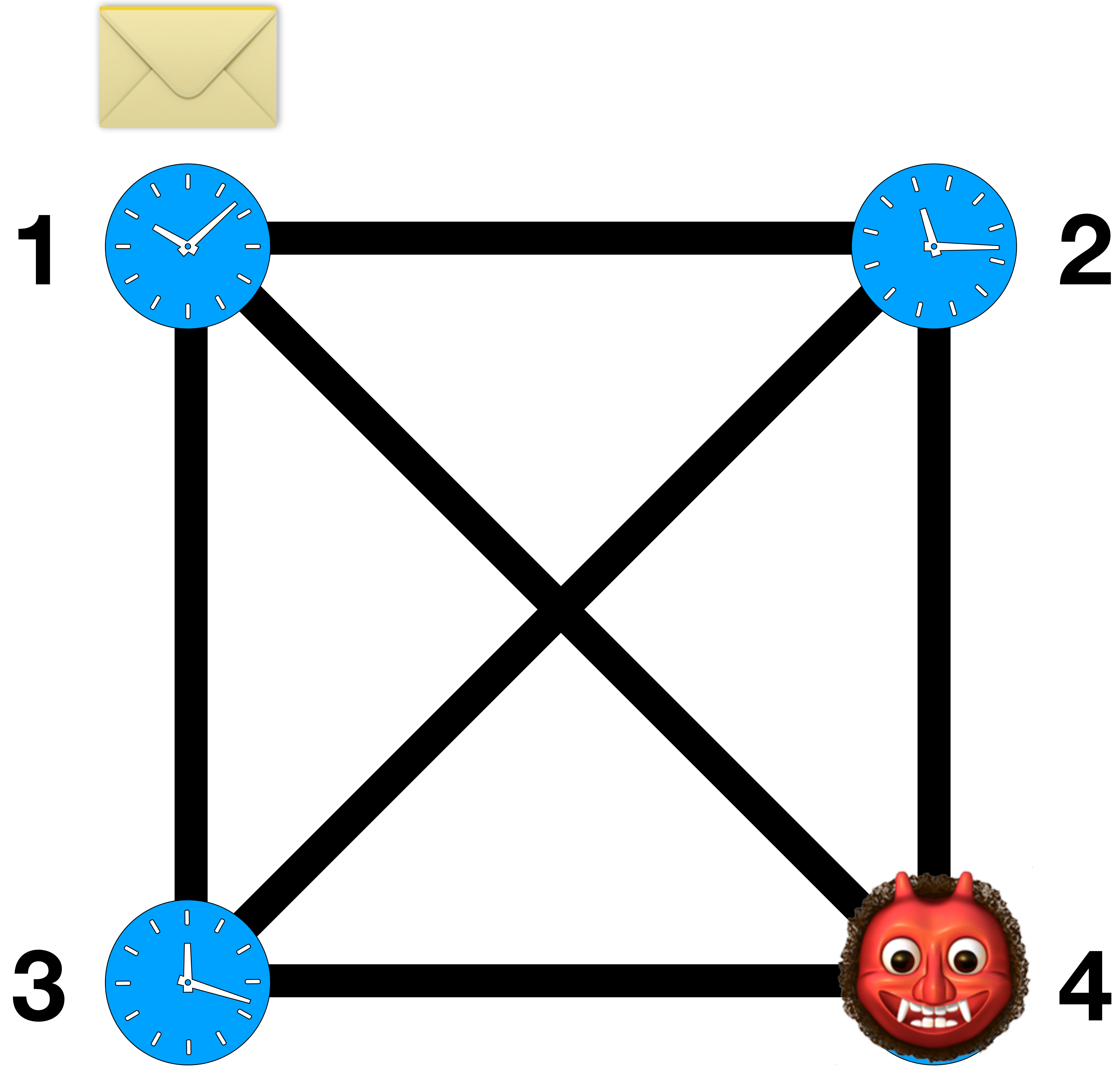
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes

Arbitrary misbehaviour:
can crash, lose messages, send different misinformation to correct nodes, etc.

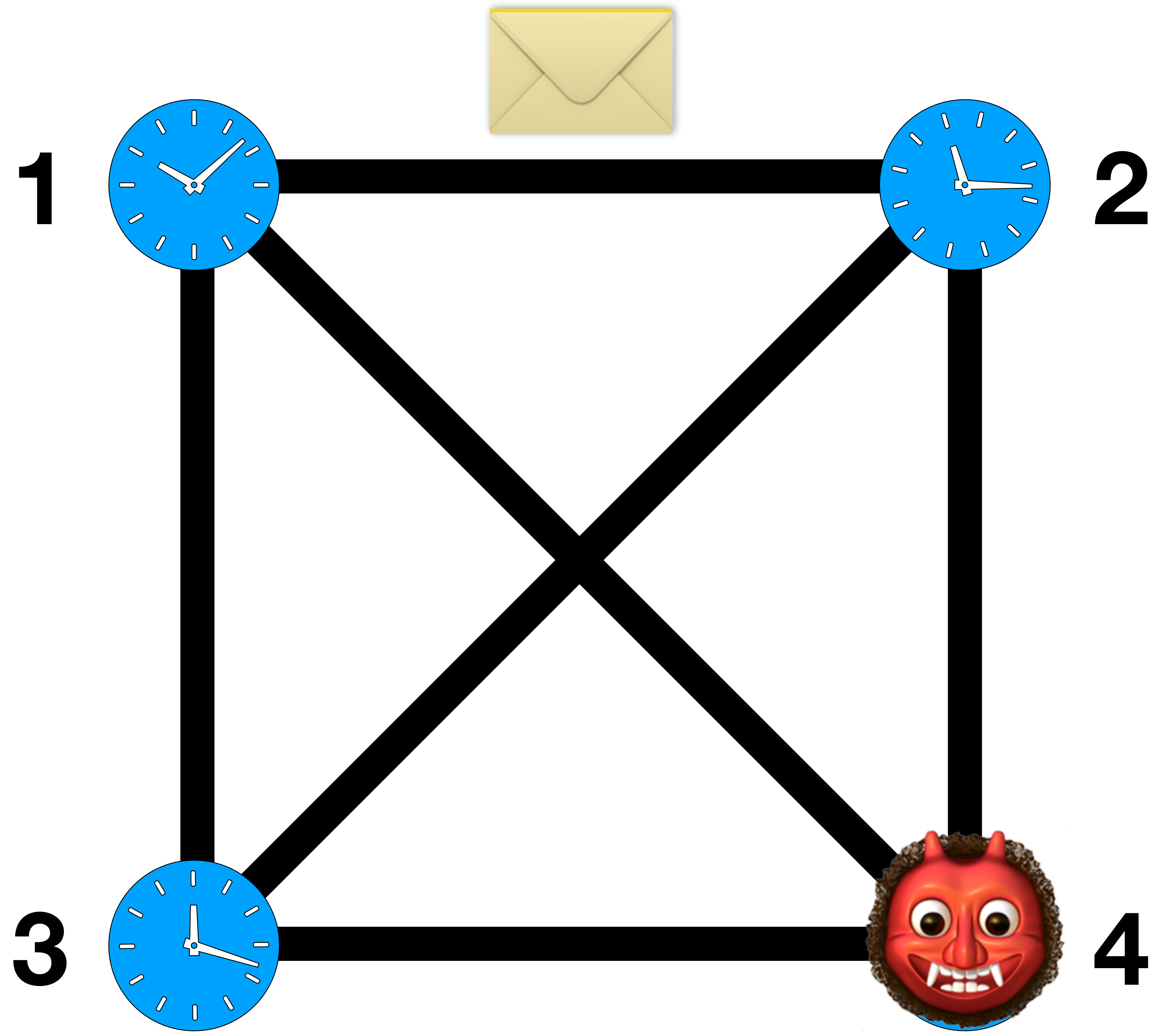
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes
- bounded delay communication

Message delay:
transmitting a messages takes
between 0 and d time units

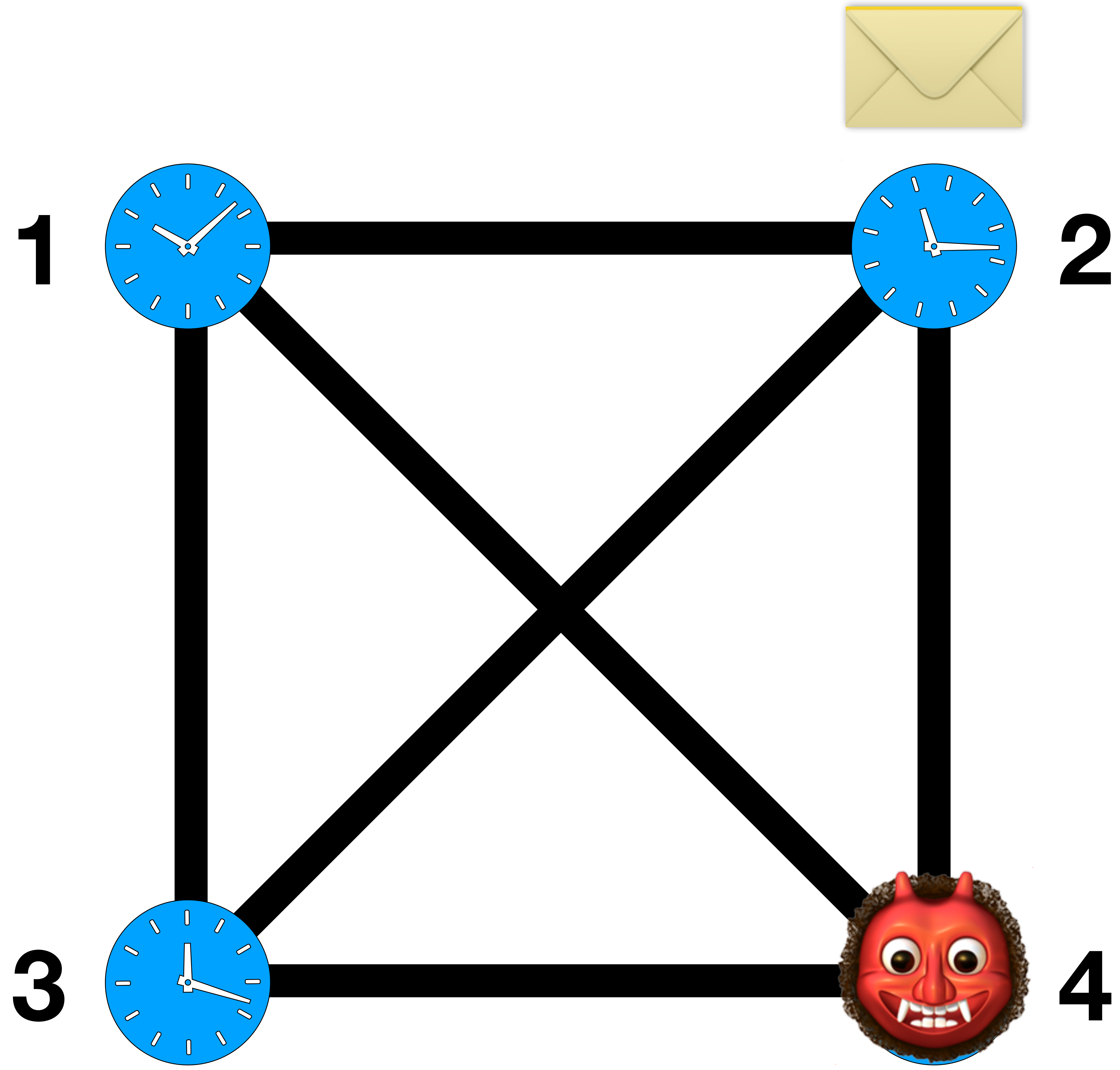
Model of computation



- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes
- bounded delay communication

Message delay:
transmitting a messages takes **between** 0 and d time units

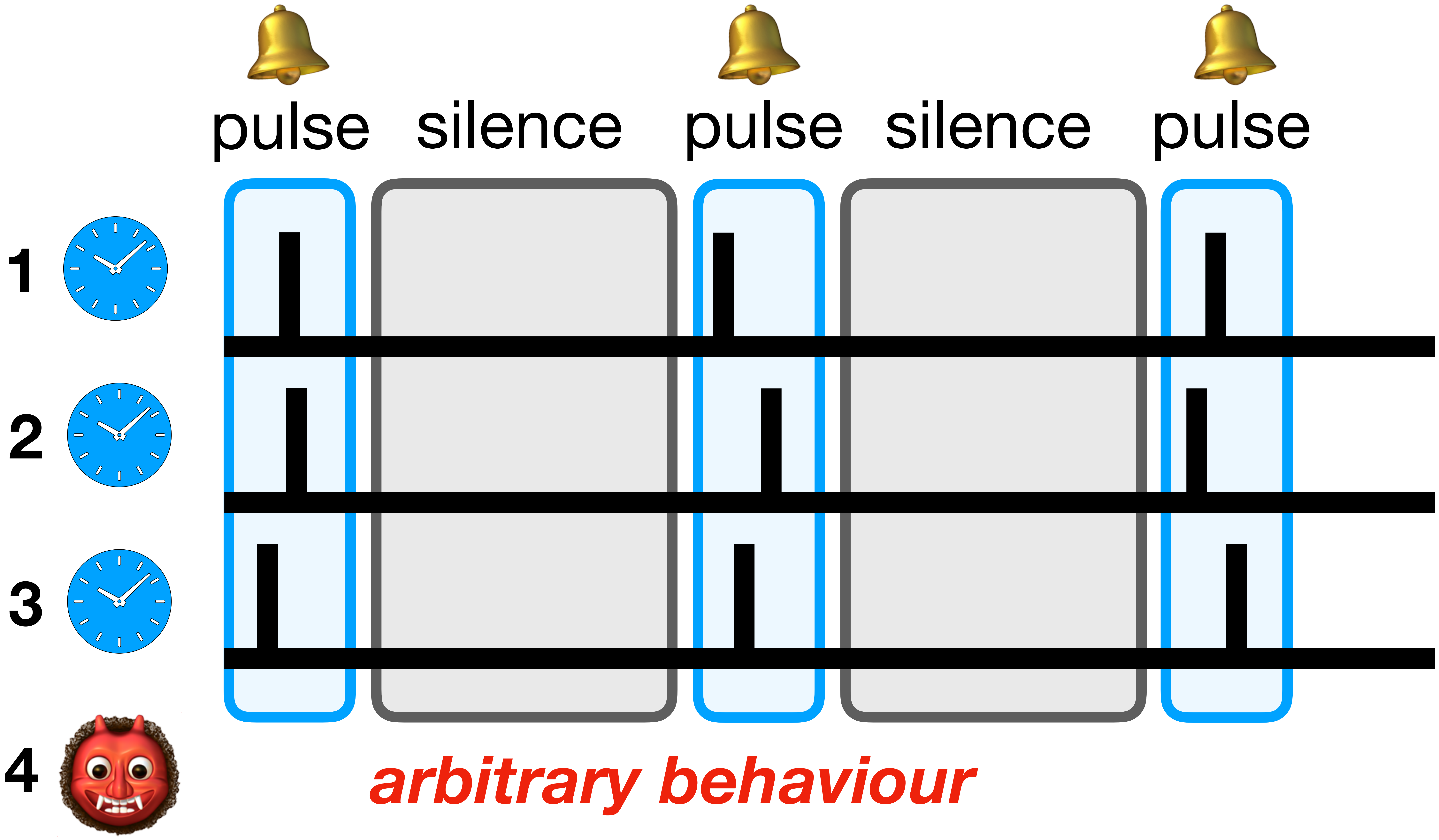
Model of computation



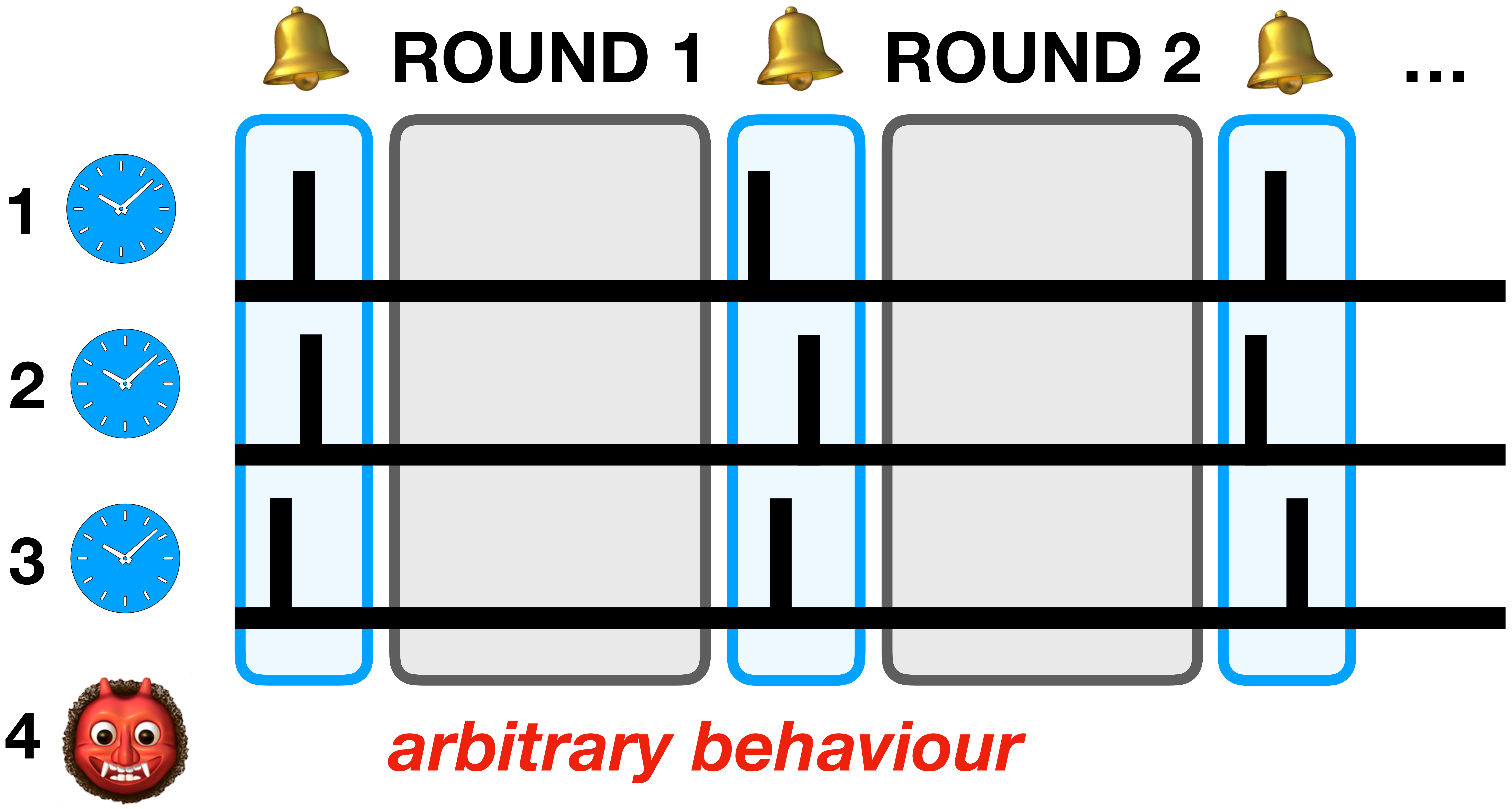
- n nodes with **local clocks**
- arbitrary initial state
- $f < n/3$ **Byzantine faulty** nodes
- bounded delay communication

Message delay:
transmitting a messages takes **between** 0 and d time units

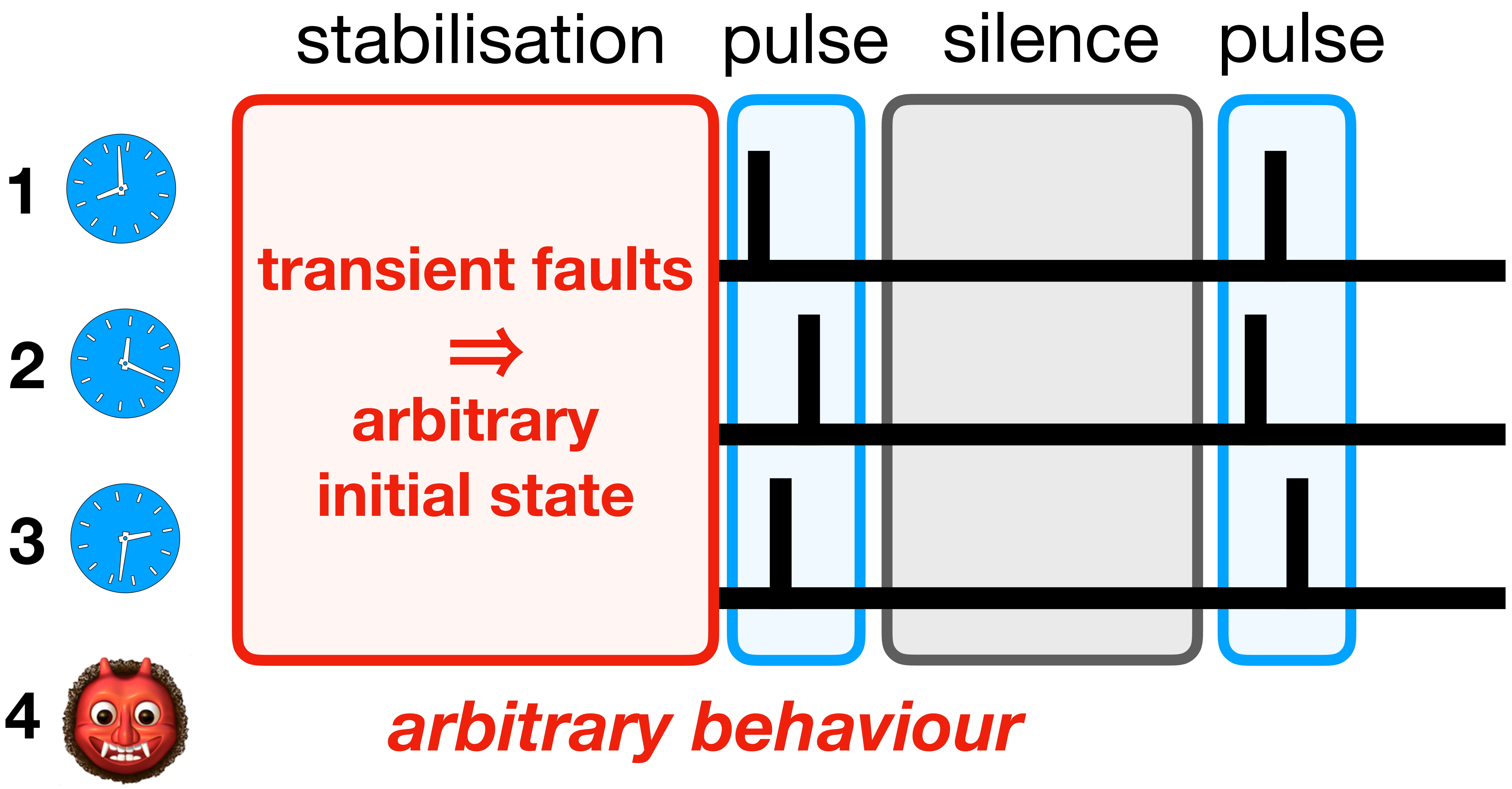
Pulse synchronisation



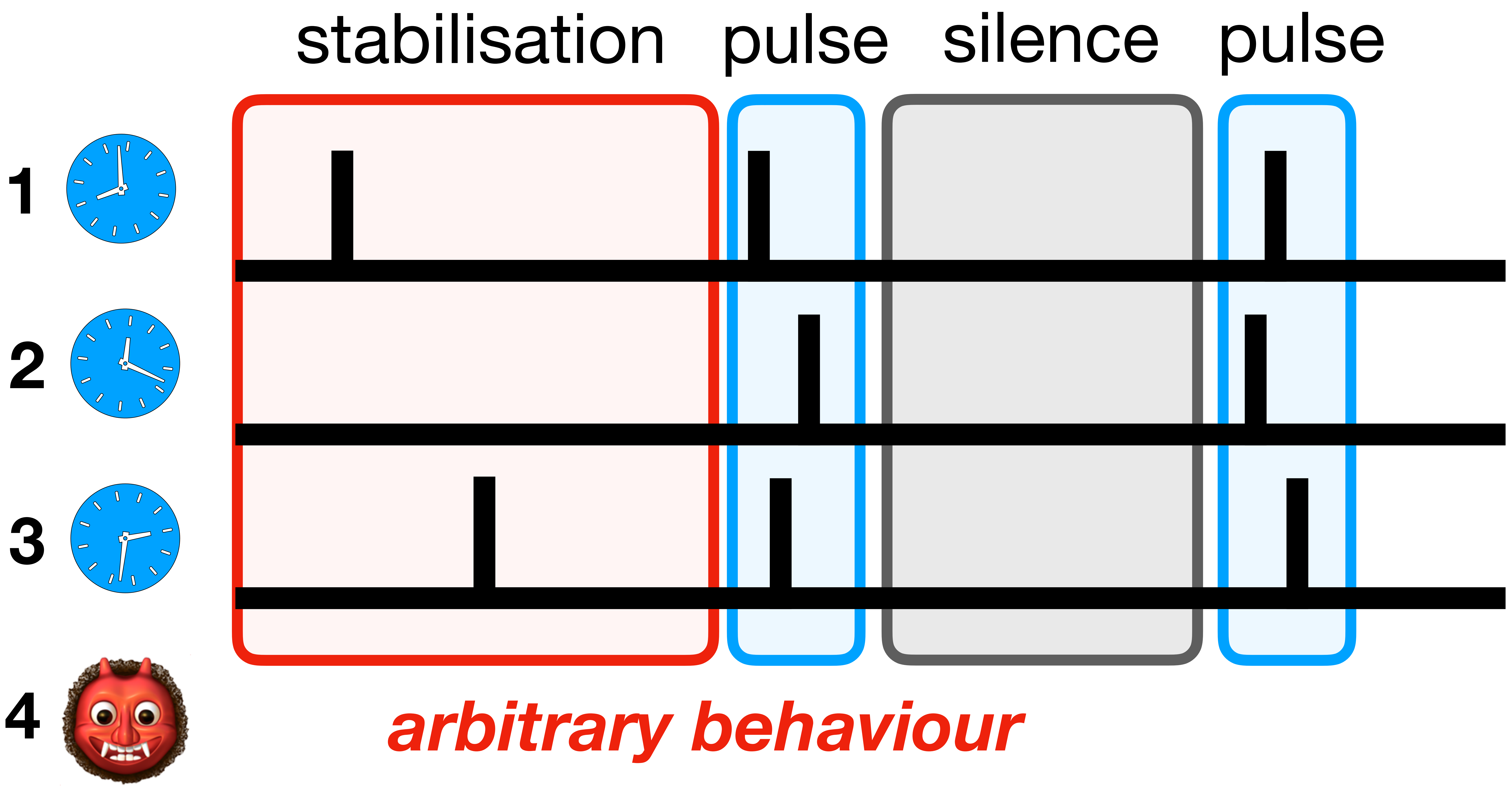
Gives the synchronous model



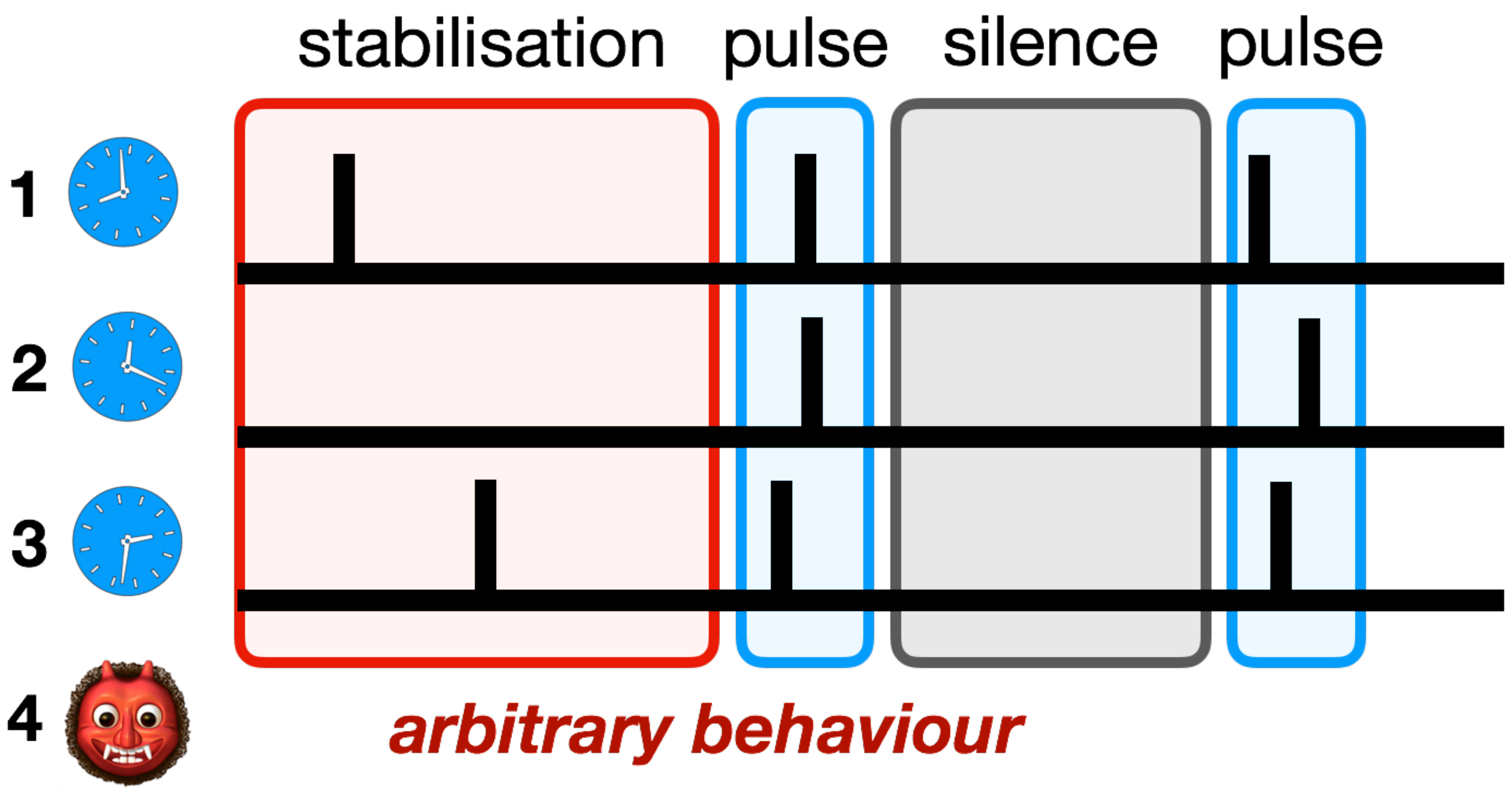
Self-stabilising pulse synchronisation



Self-stabilising pulse synchronisation



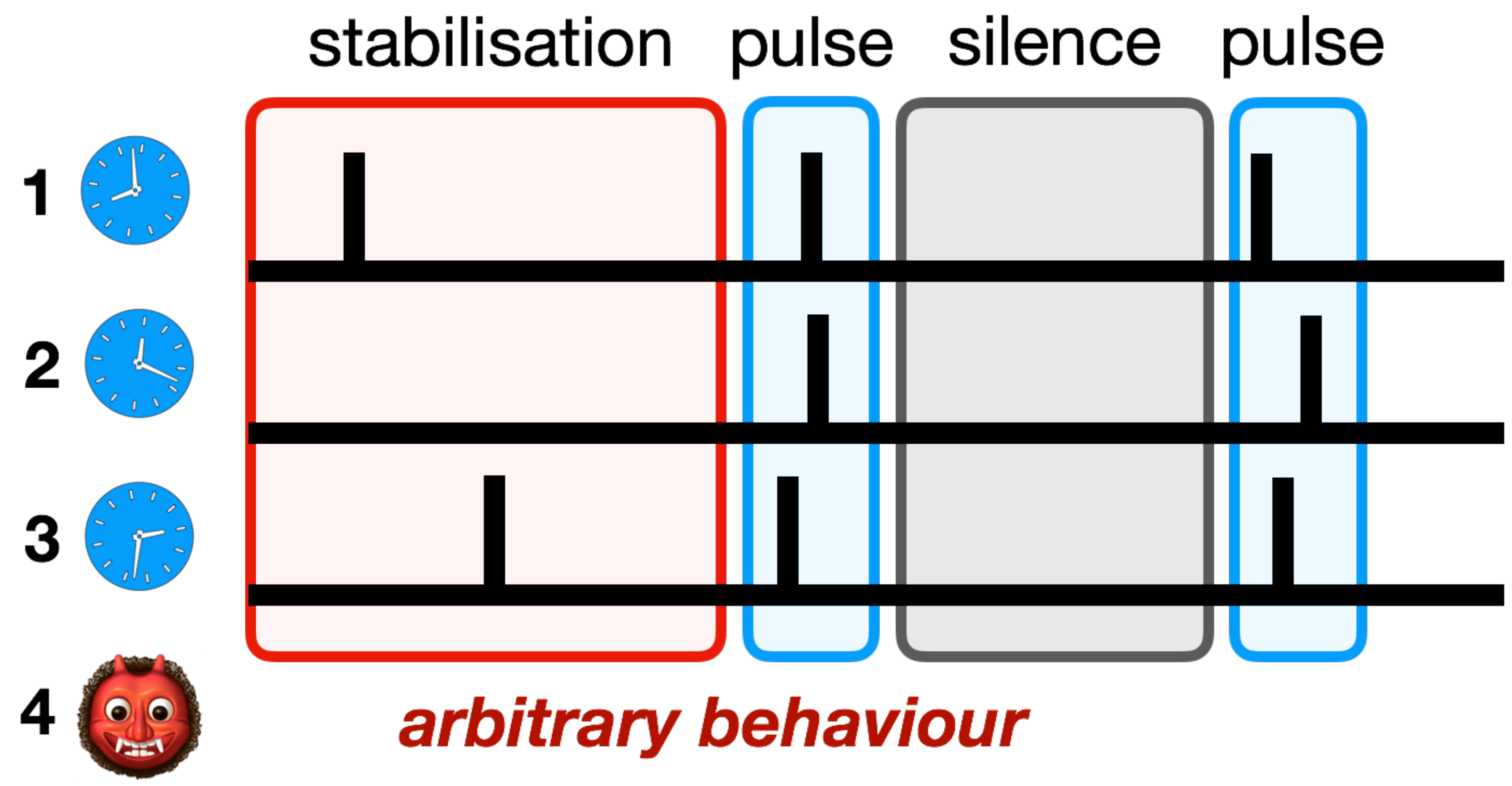
Complexity measures



Stabilisation time:
how many time units it takes before the correct nodes start pulsing synchronously

- n nodes
- $f < n/3$ faults
- drift parameter $\vartheta = O(1)$
- message delay $d = O(1)$

Complexity measures



- n nodes
- $f < n/3$ faults
- drift parameter $\vartheta = O(1)$
- message delay $d = O(1)$

Stabilisation time:
how many time units it takes before the correct nodes start pulsing synchronously

Bandwidth (per node):
how many bits per time unit a node broadcasts

Pulse synchronisation: status

authors	appears in		stabilisation time	bandwidth
S. Dolev & Welch	JACM 2004		exp(O(f))	**O(1)**
Daliot, D. Dolev & Parnas	SSS 2003		O(f^3)	**O(log f)**
D. Dolev & Hoch	SSS 2007		**O(f)**	O(f log f)
D. Dolev, Függer, Lenzen & Schmid	JACM 2014		O(f)	**O(1)**
this work		 	**polylog f** **O(log f)** **O(f)**	**polylog f** poly f **O(log f)**

Our result

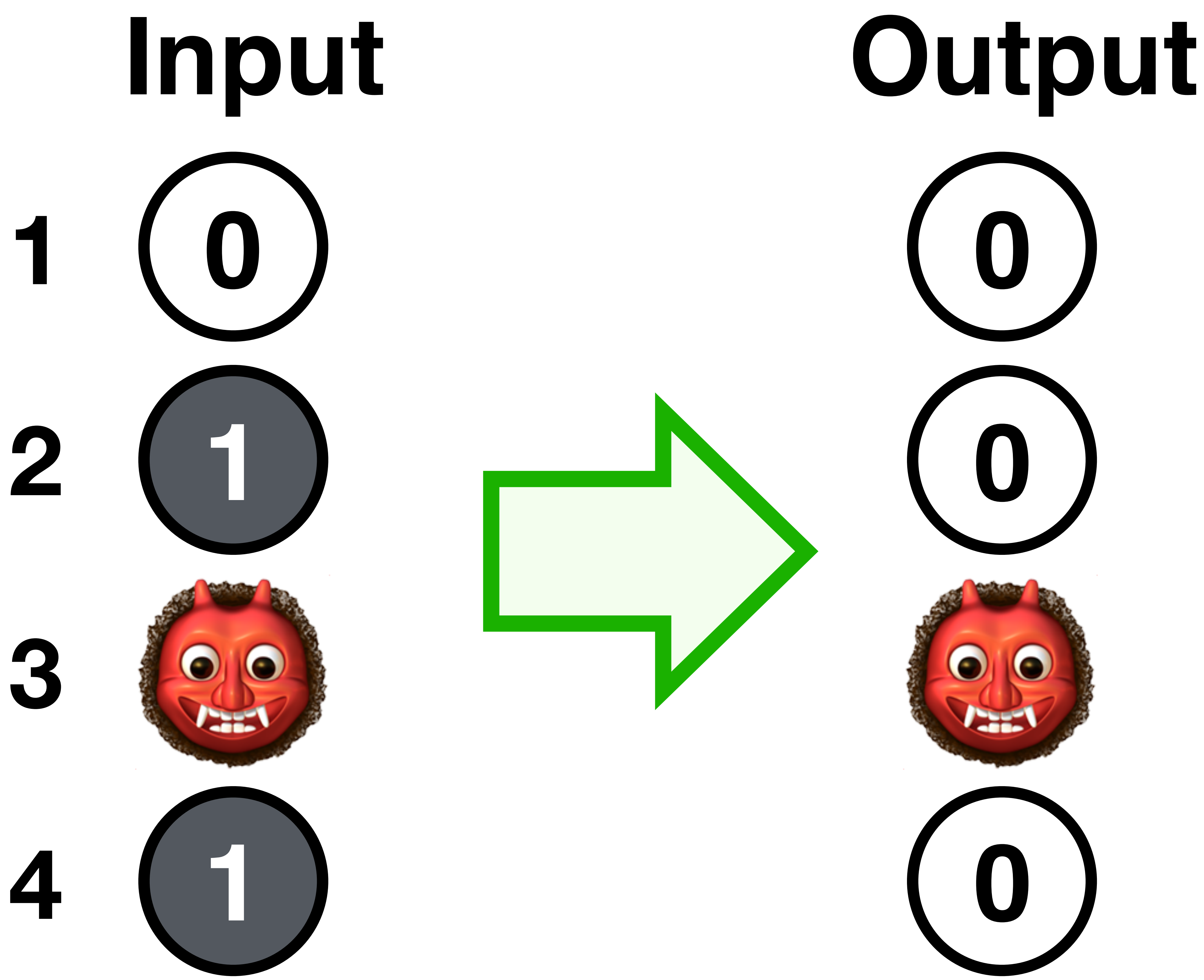
Synchronous BF consensus

$\Downarrow^*$

SS + BF pulse synchronisation
in the bounded delay model

(*) At most **logarithmic** overheads.

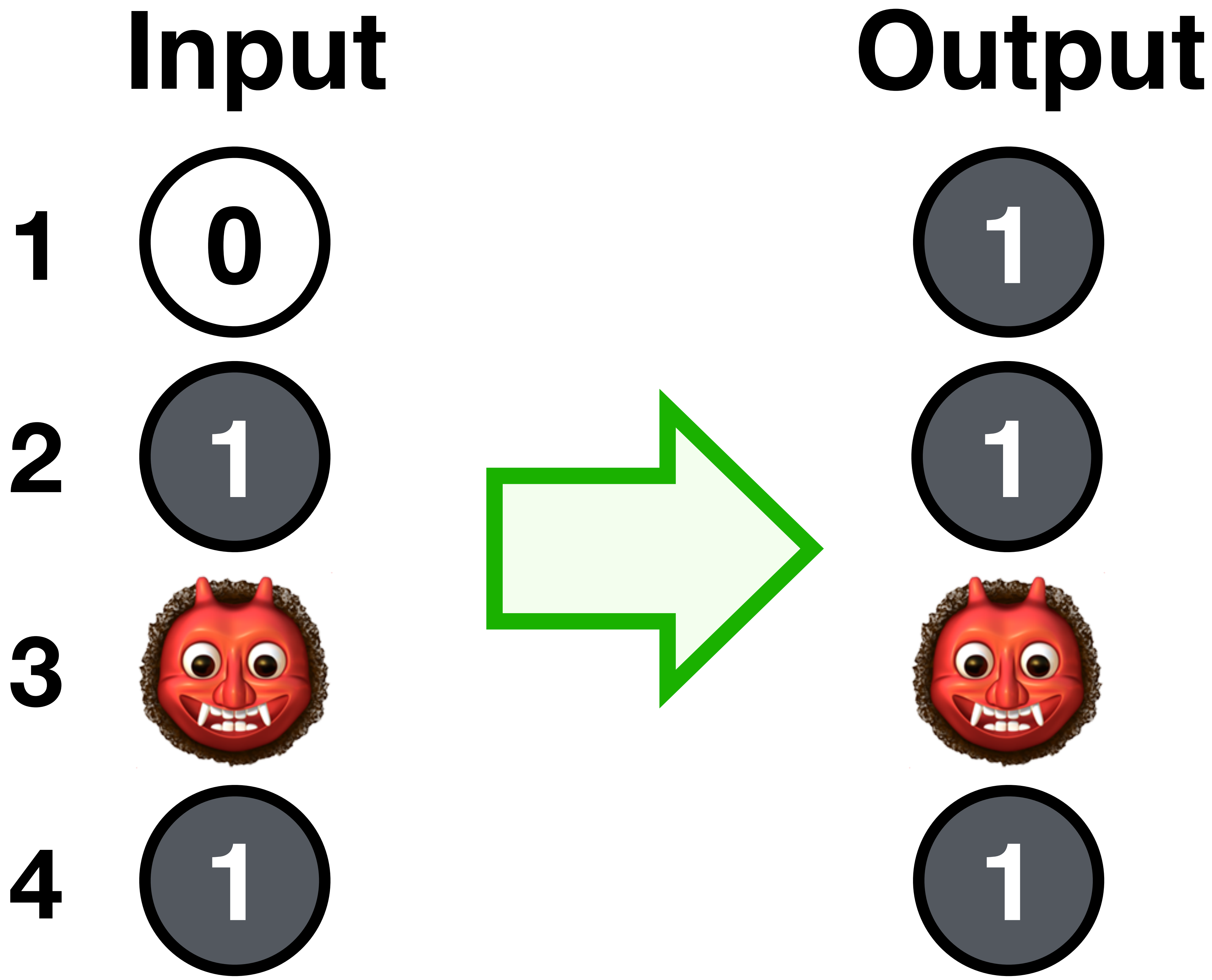
Synchronous binary consensus



Properties:

1. **Agreement**
2. Validity
3. Termination

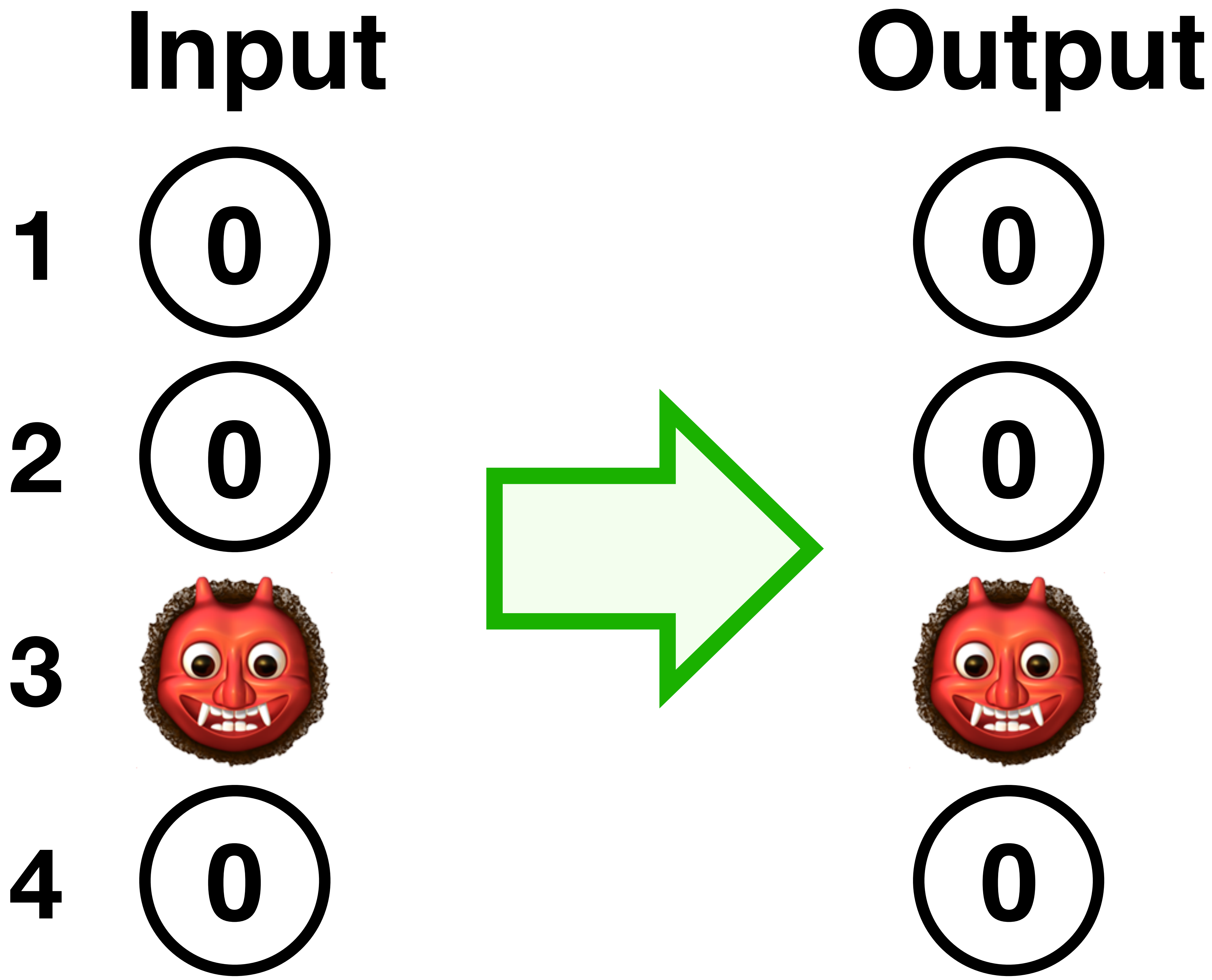
Synchronous binary consensus



Properties:

1. **Agreement**
2. Validity
3. Termination

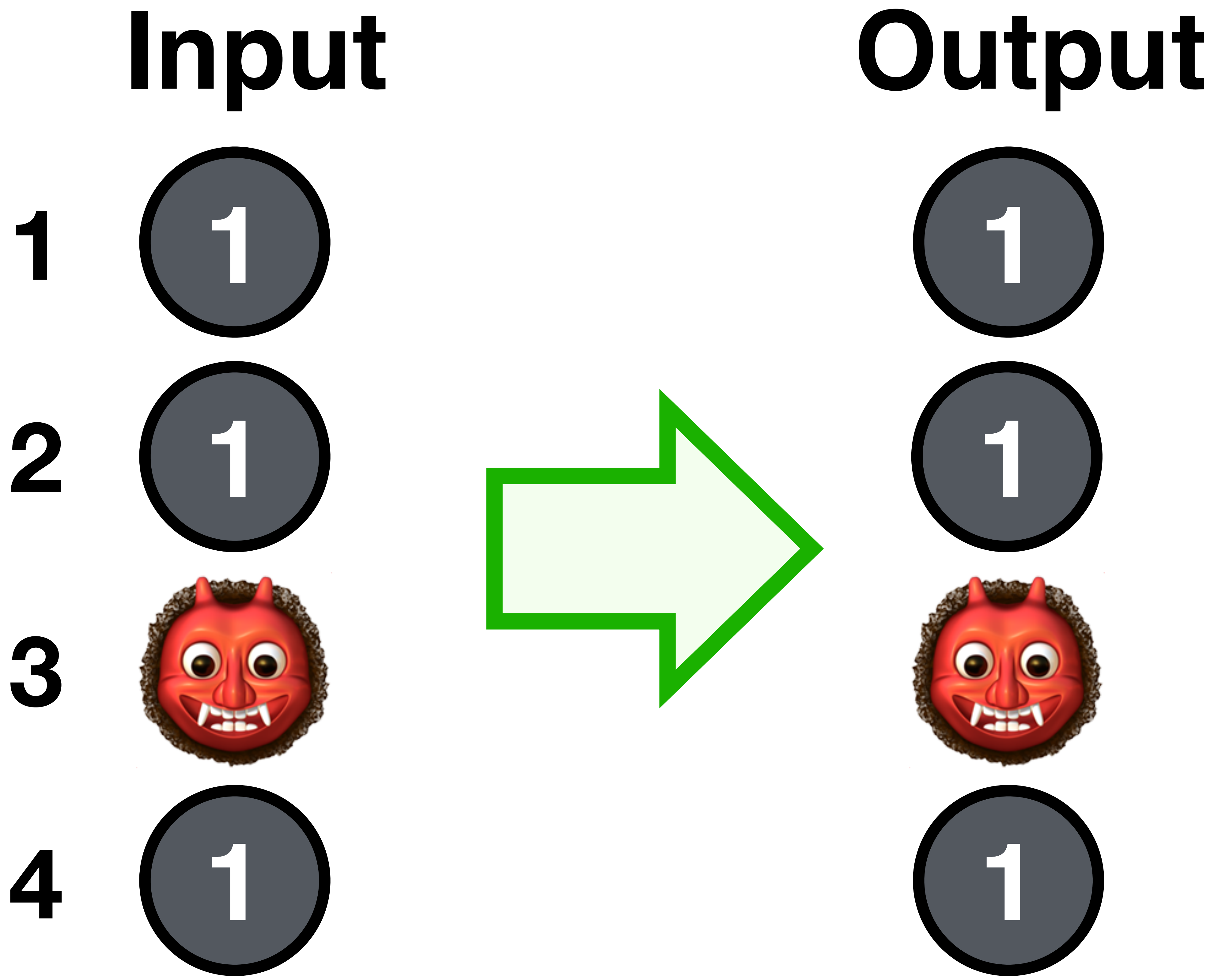
Synchronous binary consensus



Properties:

1. Agreement
2. **Validity**
3. Termination

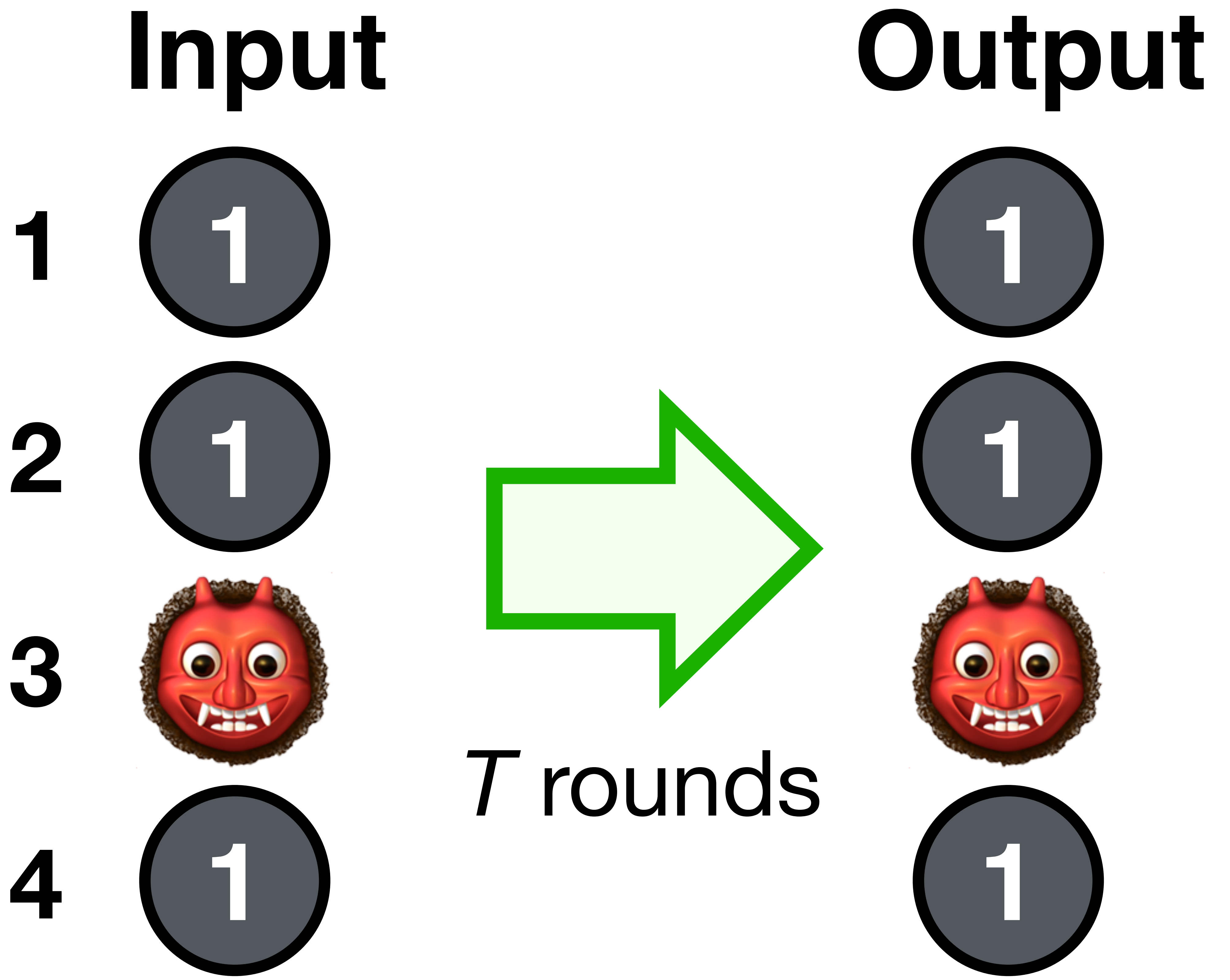
Synchronous binary consensus



Properties:

1. Agreement
2. **Validity**
3. Termination

Synchronous binary consensus



Properties:

1. Agreement
2. Validity
3. **Termination**

Synchronous consensus:

- **no** self-stabilisation
- **synchronous** communication
- terminates in *T* rounds

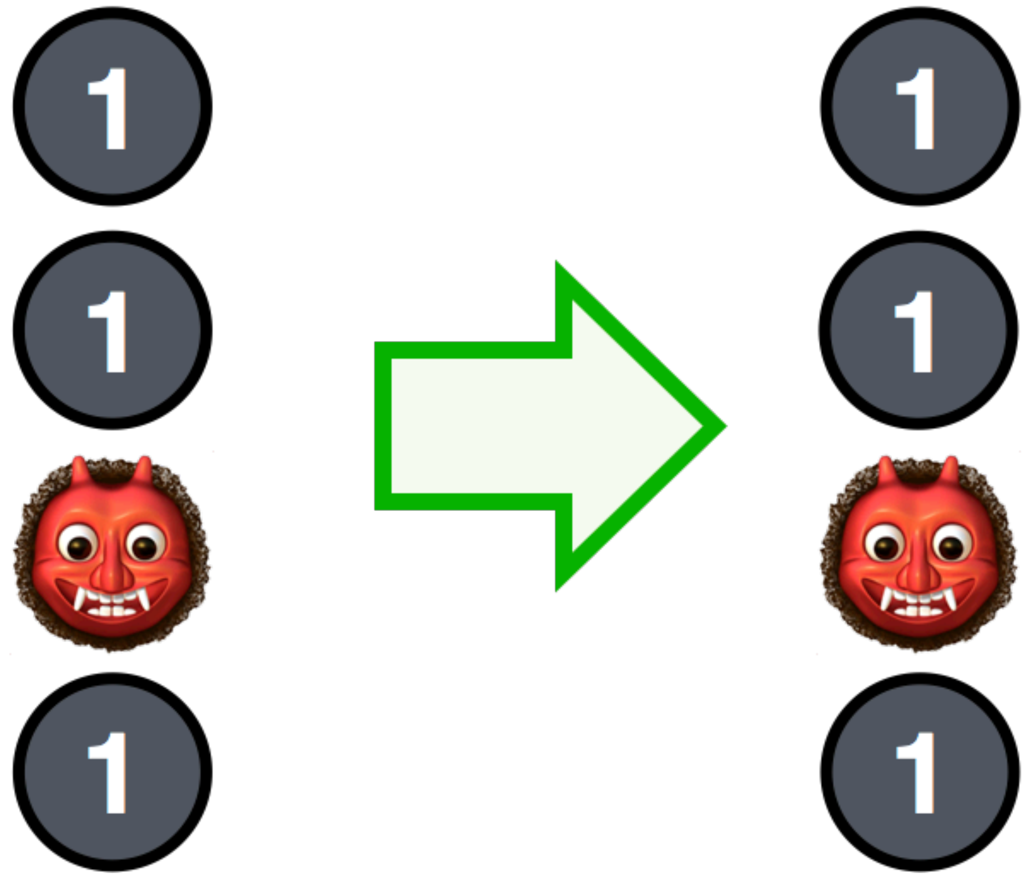
Our result

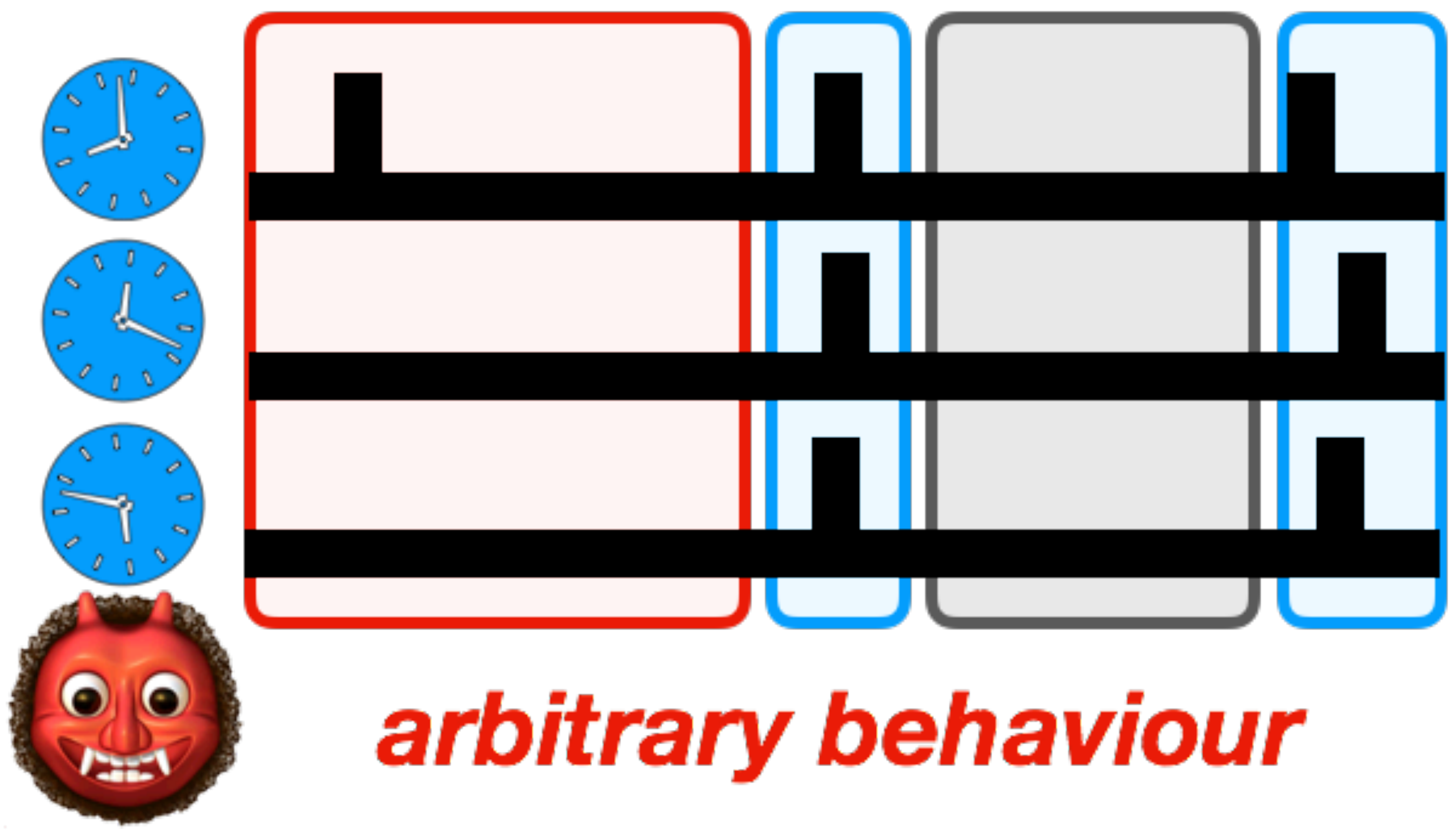
Synchronous BF consensus

$\Downarrow^*$

SS + BF pulse synchronisation in the bounded delay model

(*) At most **logarithmic** overheads.





Consensus ➡ pulse sync.

		consensus		pulse sync	
source		rounds	bits/round	stabilisation	bandwidth
Berman, Garay & Perry '1992		O(*f*)	O(1)	O(*f*)	O(log *f*)
King & Saia '2011		polylog(*f*)	polylog(*f*) *	polylog(*f*)	polylog(*f*)
Feldman & Micali '1988		O(1)	poly(*f*)	O(log *f*)	poly(*f*)

Approach: resilience boosting

Adapt ideas and techniques from prior work on *synchronous counting* (digital clock synchronisation)*:*

— Lenzen & R. (SSS 2016)
— Lenzen, R. & Suomela (SICOMP 2017)

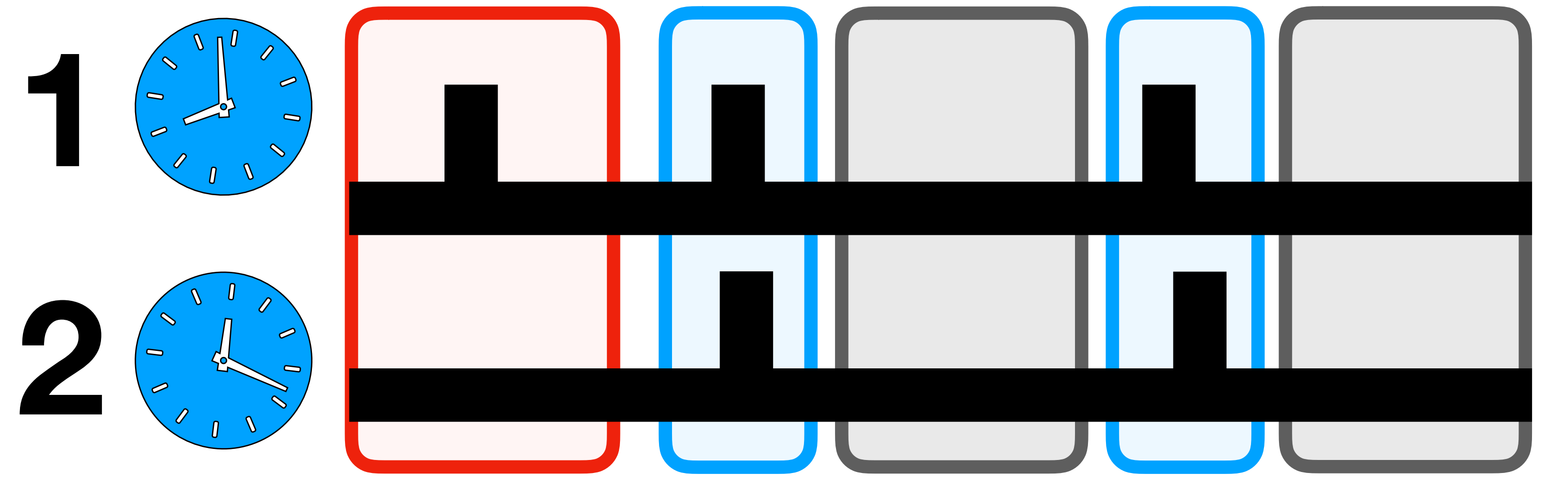
Difficulty: translating techniques from the synchronous model to bounded-delay model with clock drift

Resilience boosting: idea

Given:

Pulser **A**$_0$

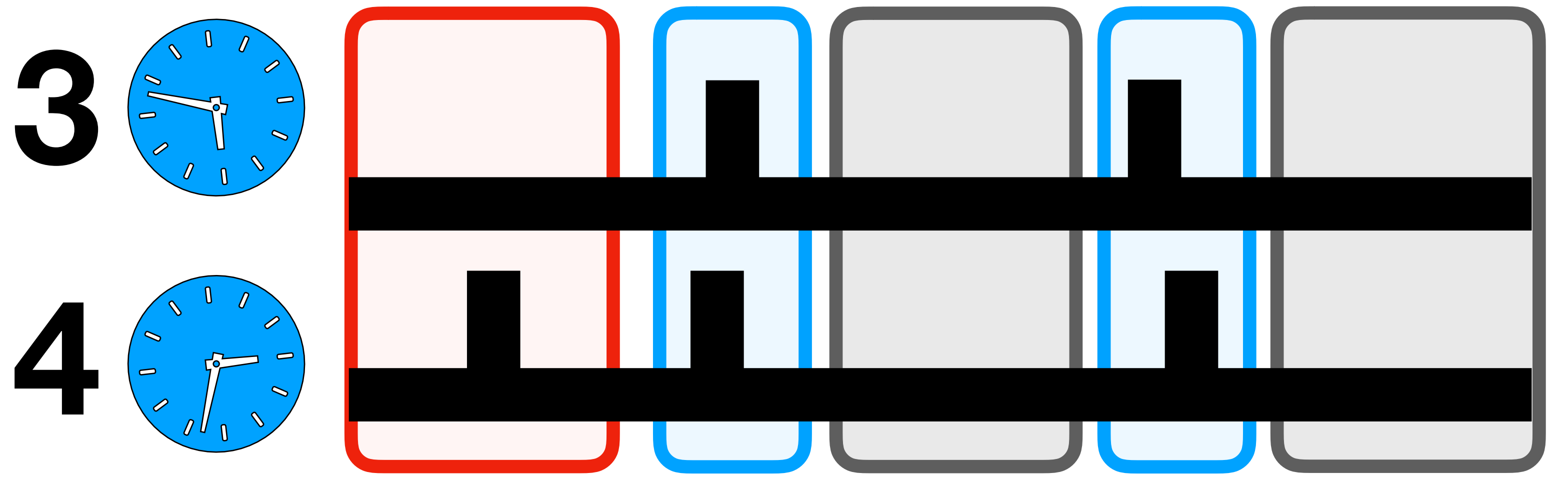
- n_0 = 2 nodes
- f_0 = **0** resilience



+

Pulser **A**$_1$

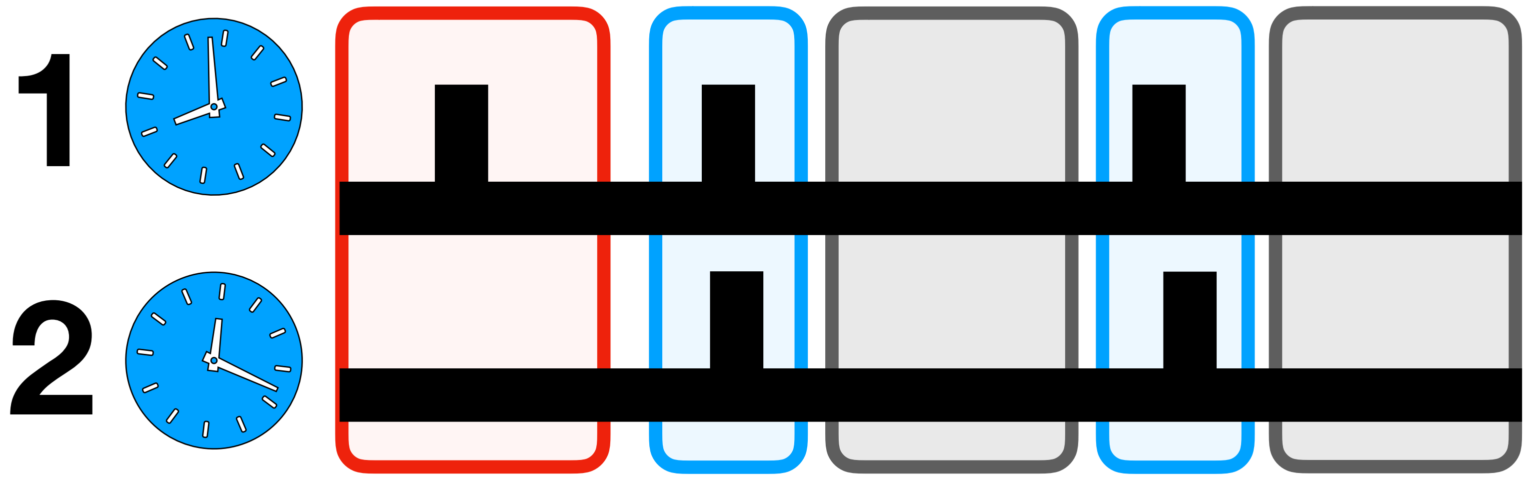
- n_1 = 2 nodes
- f_1 = **0** resilience



Resilience boosting: idea

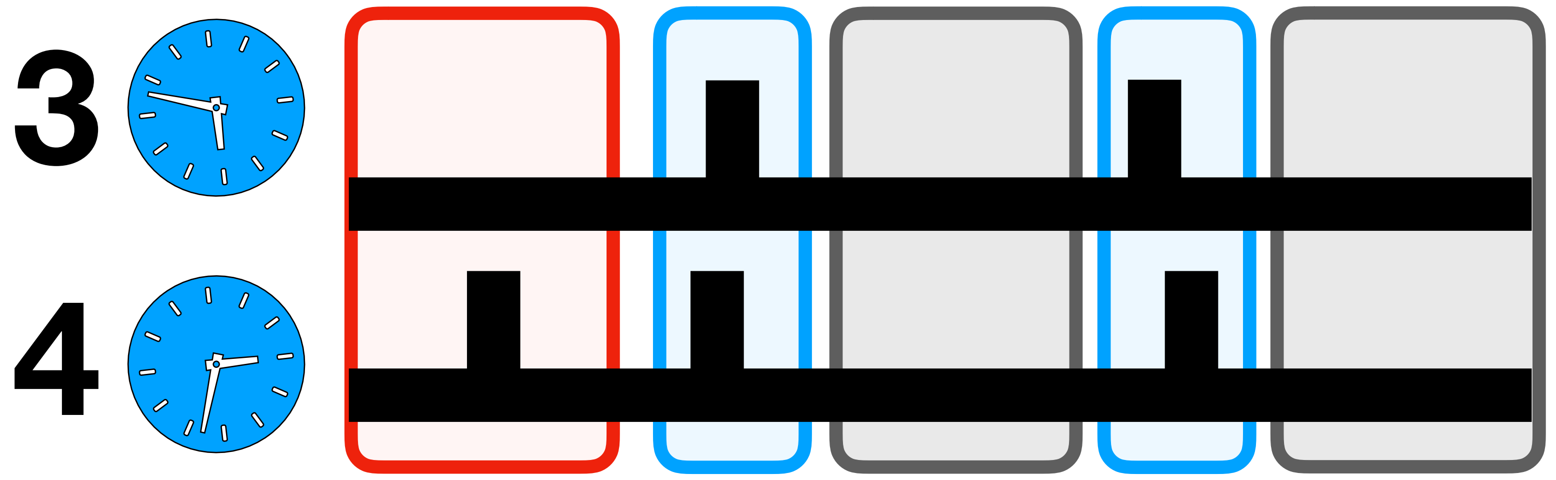
Given:

Pulser $\mathbf{A_0}$
- $n_0 = 2$ nodes
- $f_0 = \mathbf{0}$ resilience



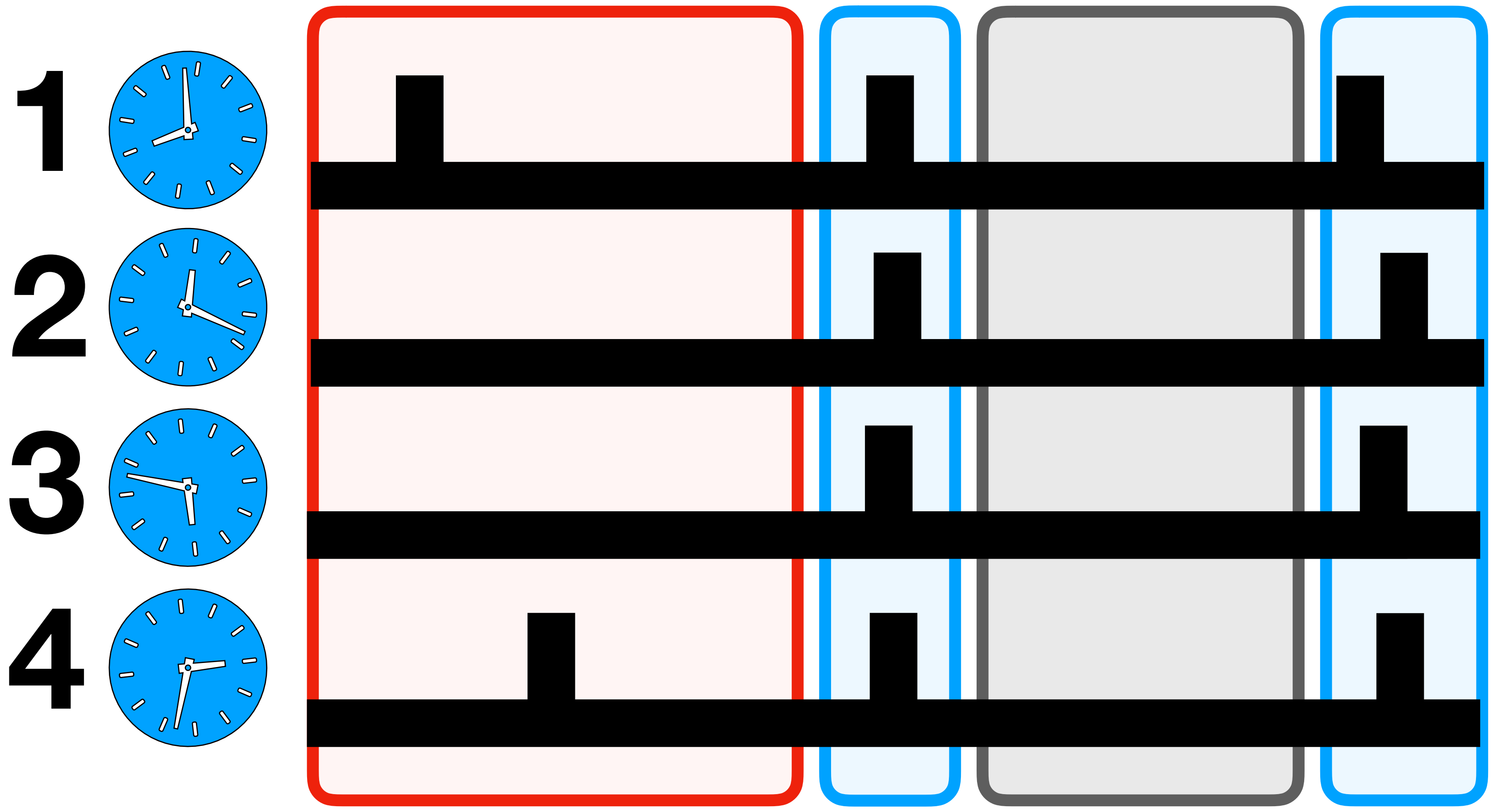
+

Pulser $\mathbf{A_1}$
- $n_1 = 2$ nodes
- $f_1 = \mathbf{0}$ resilience



Result:

Pulser $\mathbf{A}$
- $n = n_0 + n_1$ nodes
- $f = f_0 + f_1 + \mathbf{1}$ resilience

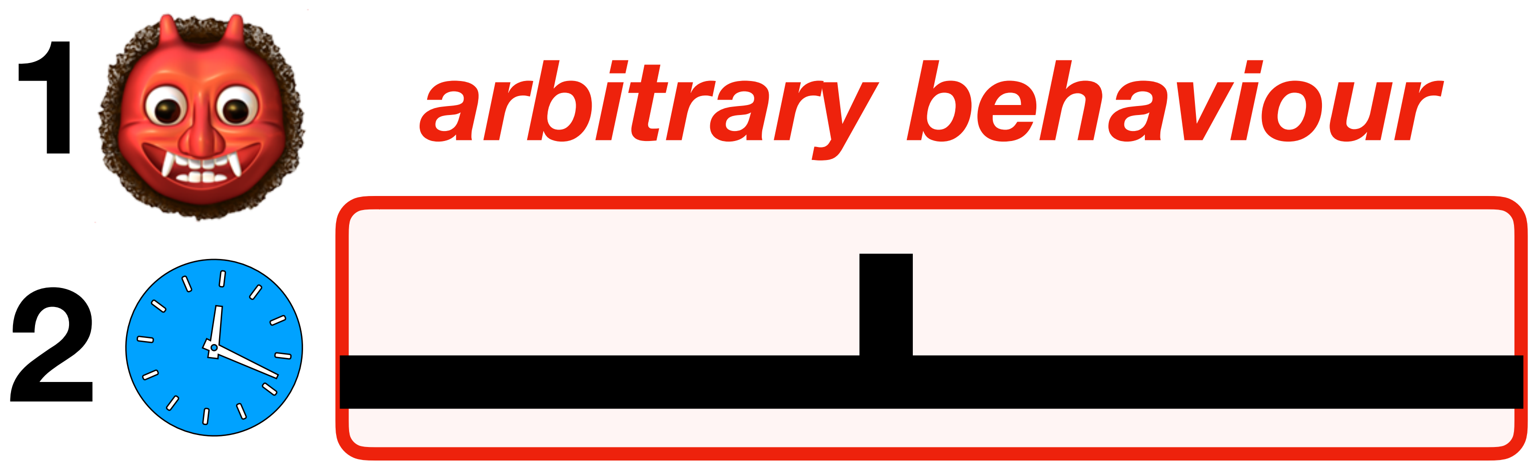


Resilience boosting: idea

Given:

Pulser $\mathbf{A_0}$

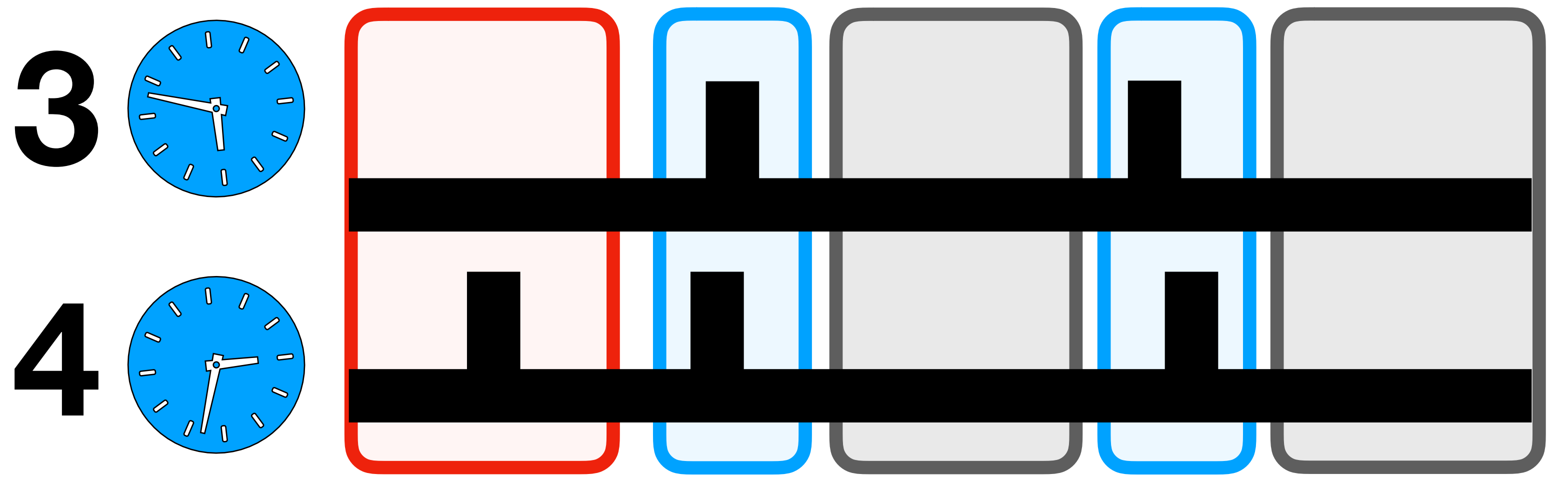
- $n_0 = 2$ nodes
- $f_0 = \mathbf{0}$ resilience



+

Pulser $\mathbf{A_1}$

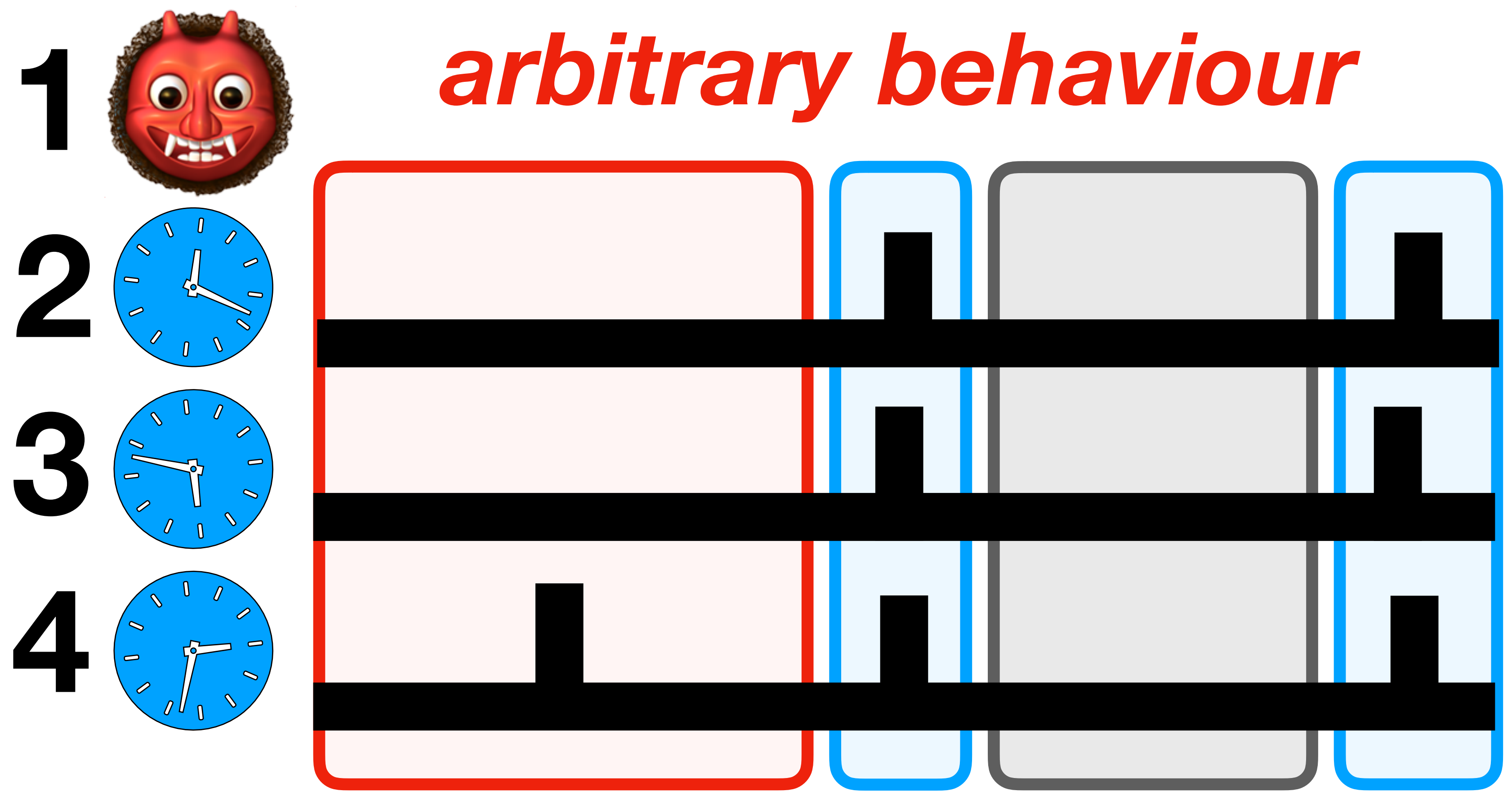
- $n_1 = 2$ nodes
- $f_1 = \mathbf{0}$ resilience



Result:

Pulser A

- $n = n_0 + n_1$ nodes
- $f = f_0 + f_1 + \mathbf{1}$ resilience

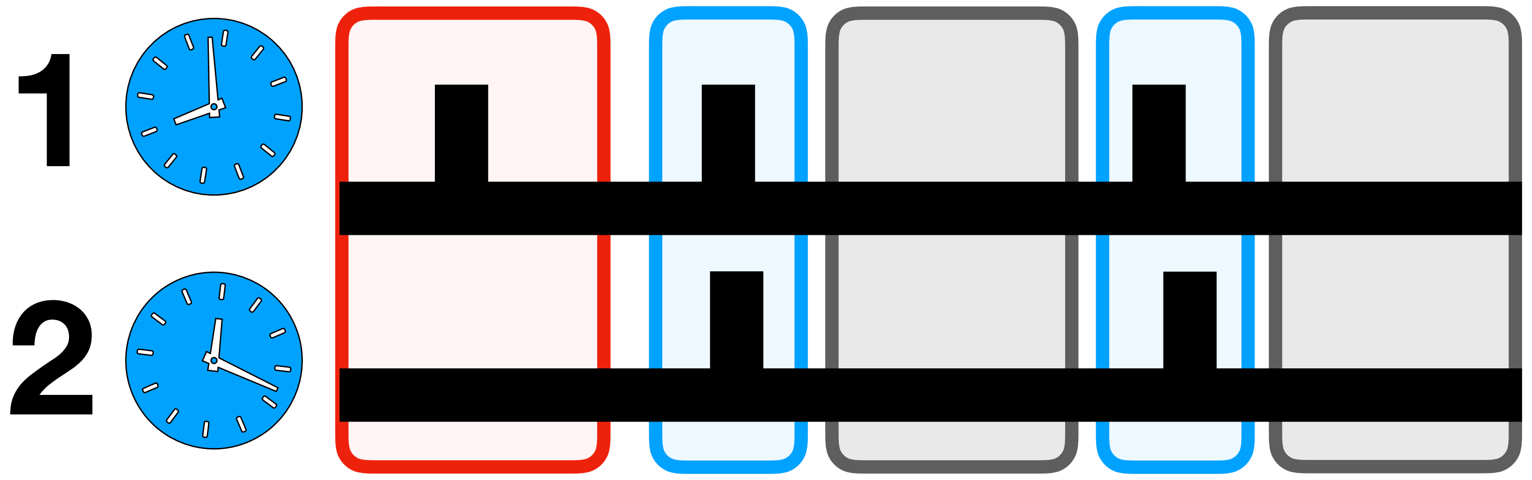


Resilience boosting: idea

Given:

Pulser $\mathbf{A_0}$

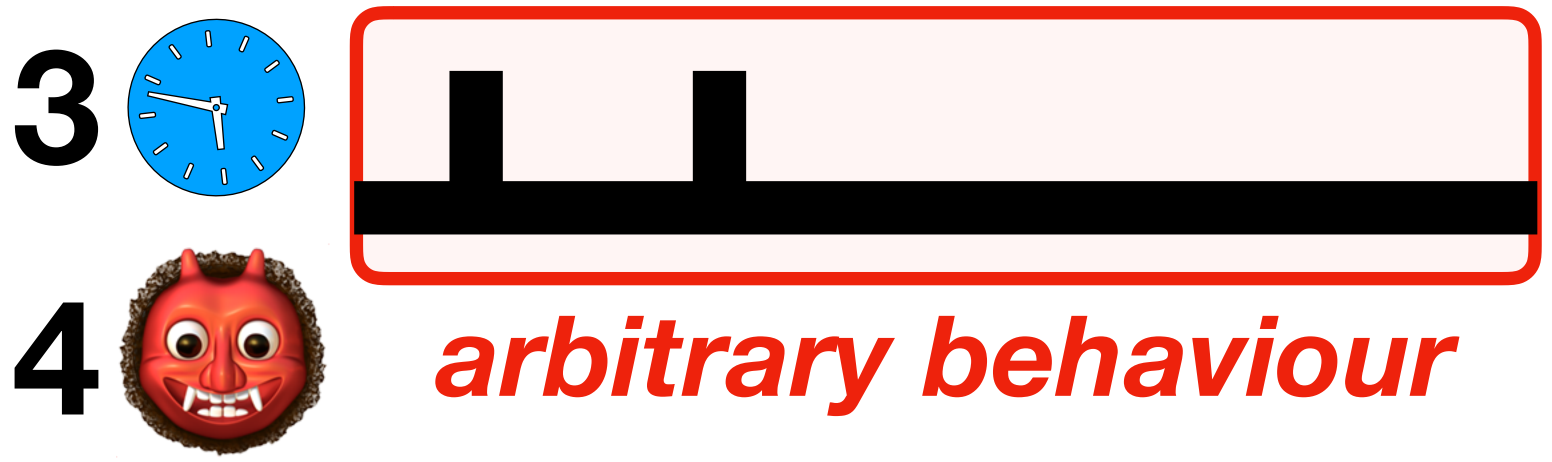
- $n_0 = 2$ nodes
- $f_0 = \mathbf{0}$ resilience



+

Pulser $\mathbf{A_1}$

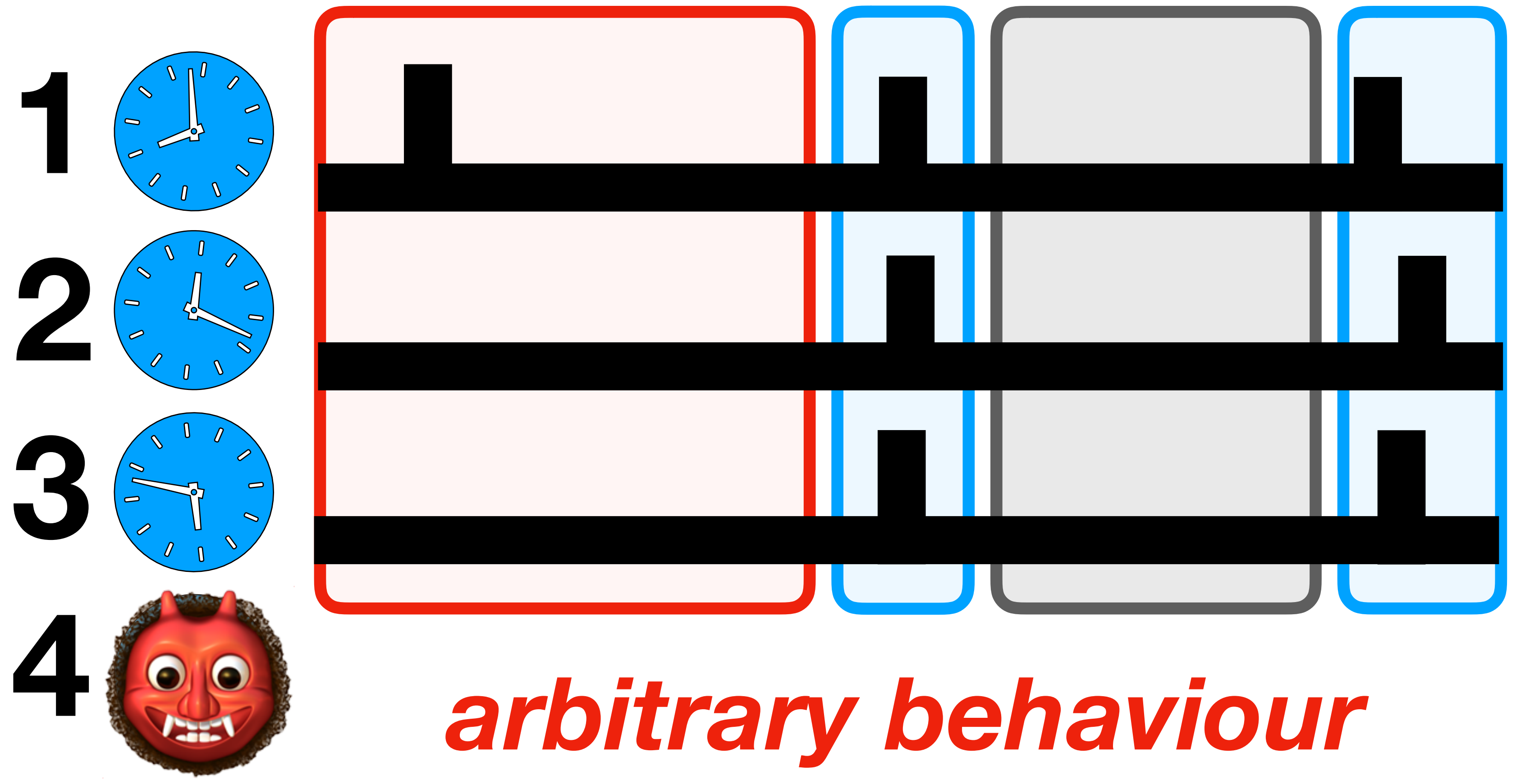
- $n_1 = 2$ nodes
- $f_1 = \mathbf{0}$ resilience



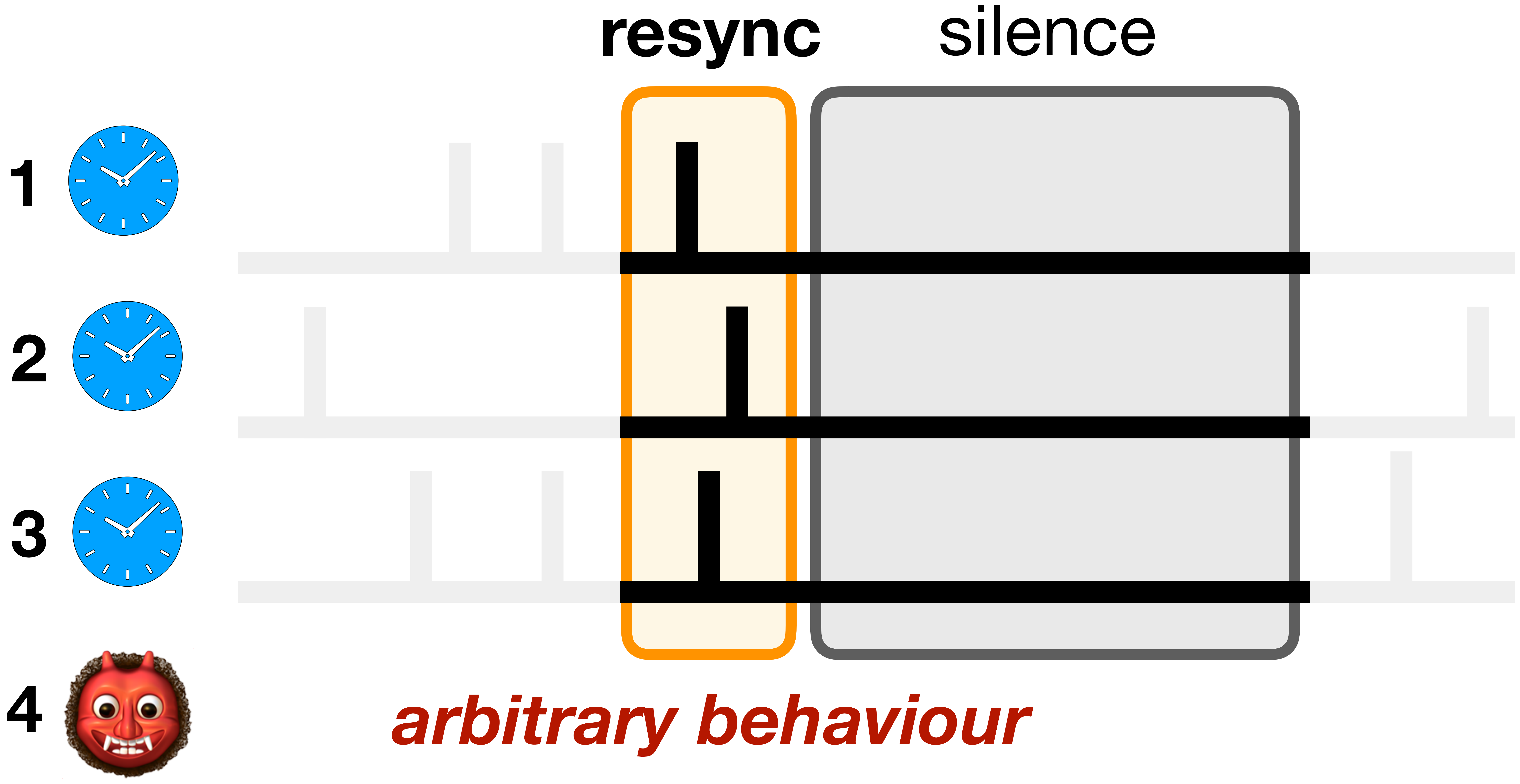
Result:

Pulser $\mathbf{A}$

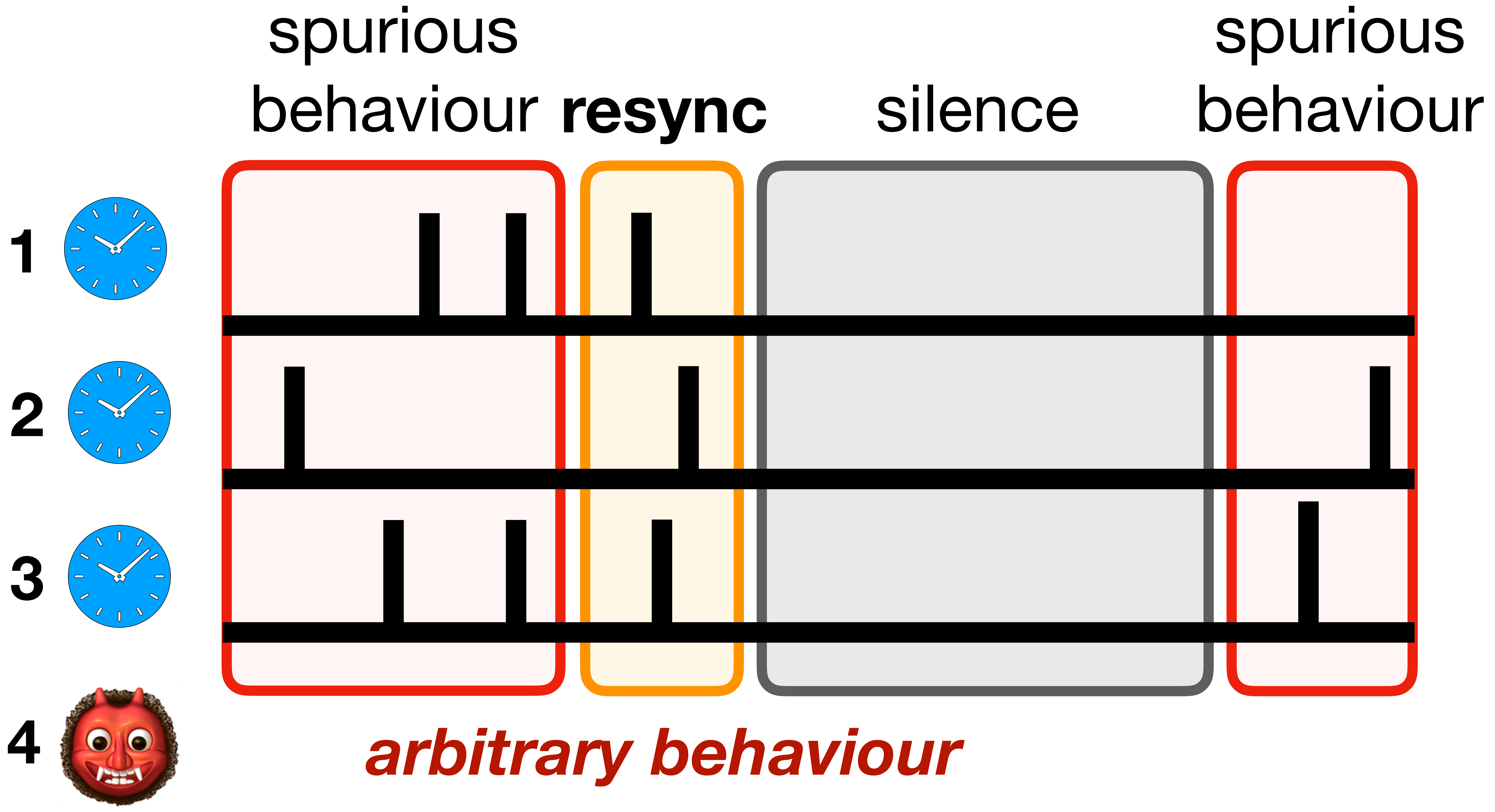
- $n = n_0 + n_1$ nodes
- $f = f_0 + f_1 \mathbf{+ 1}$ resilience



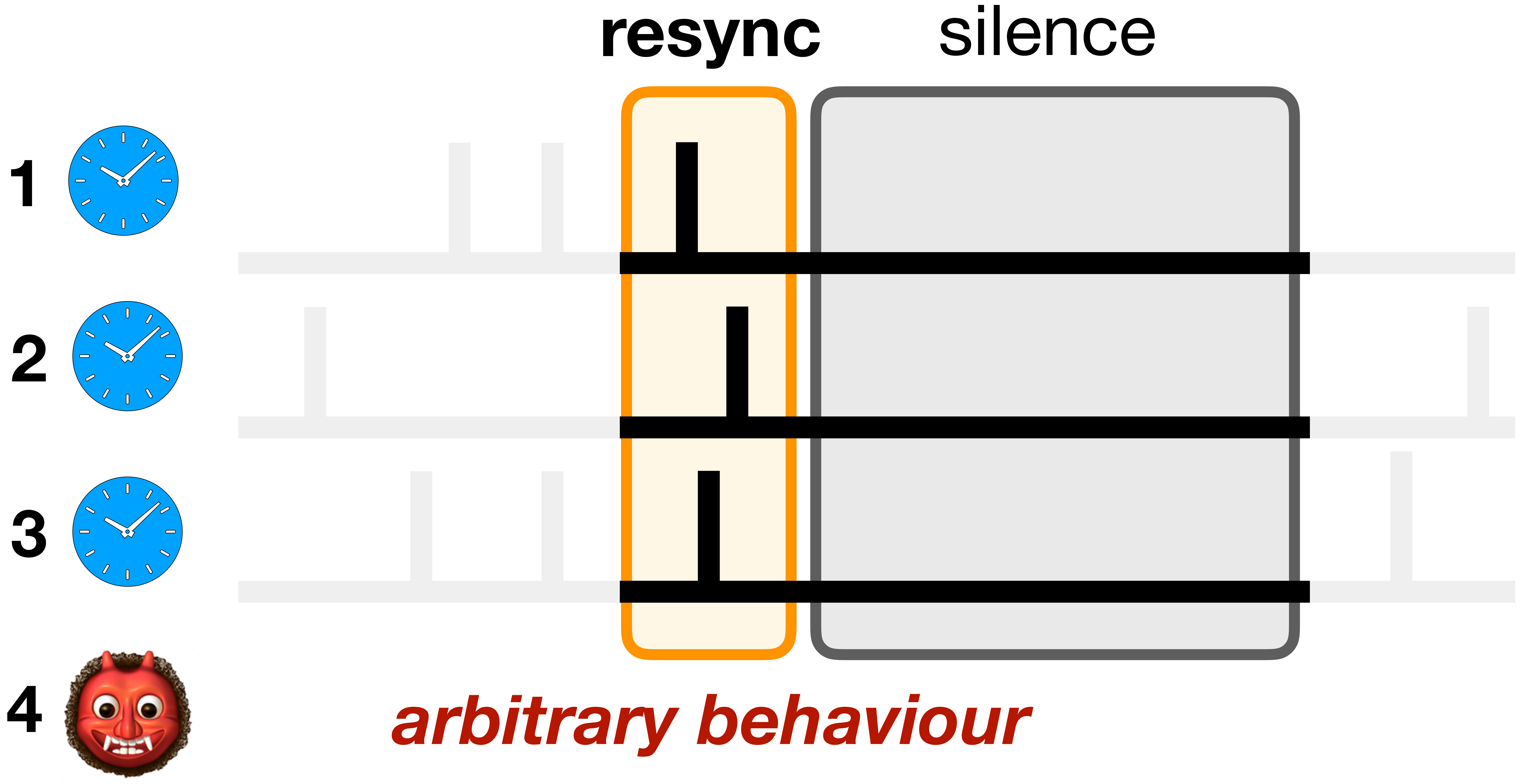
Ingredient: resynchronisation pulse



Ingredient: resynchronisation pulse



Ingredient: resynchronisation pulse

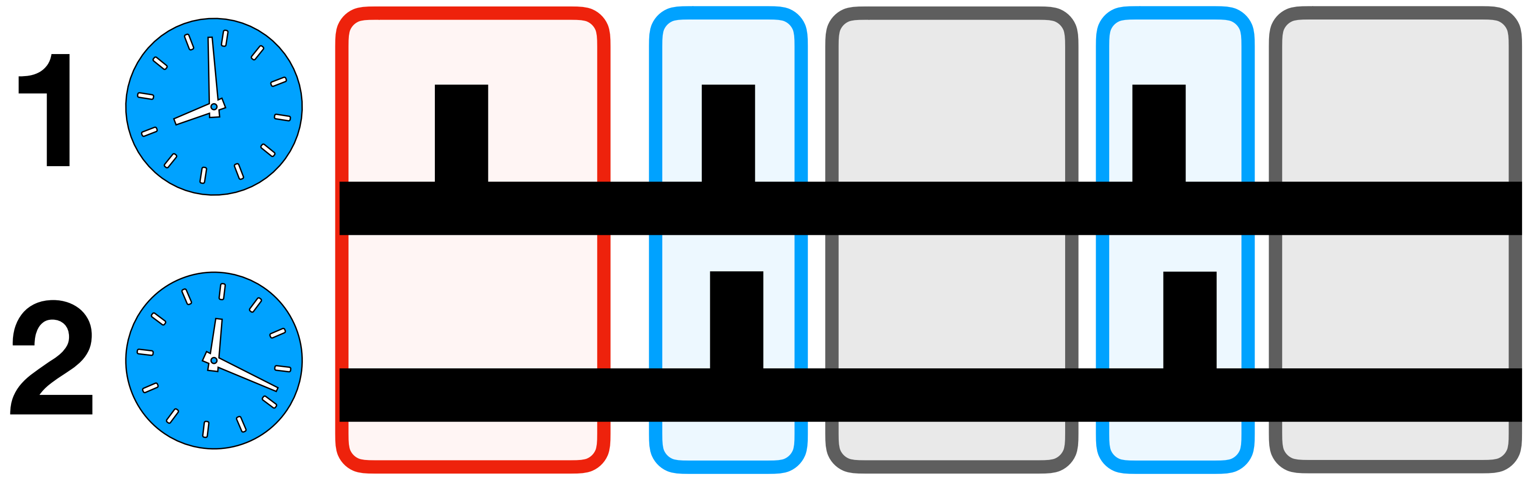


Constructing resync. algorithms

Given:

Pulser A$_0$

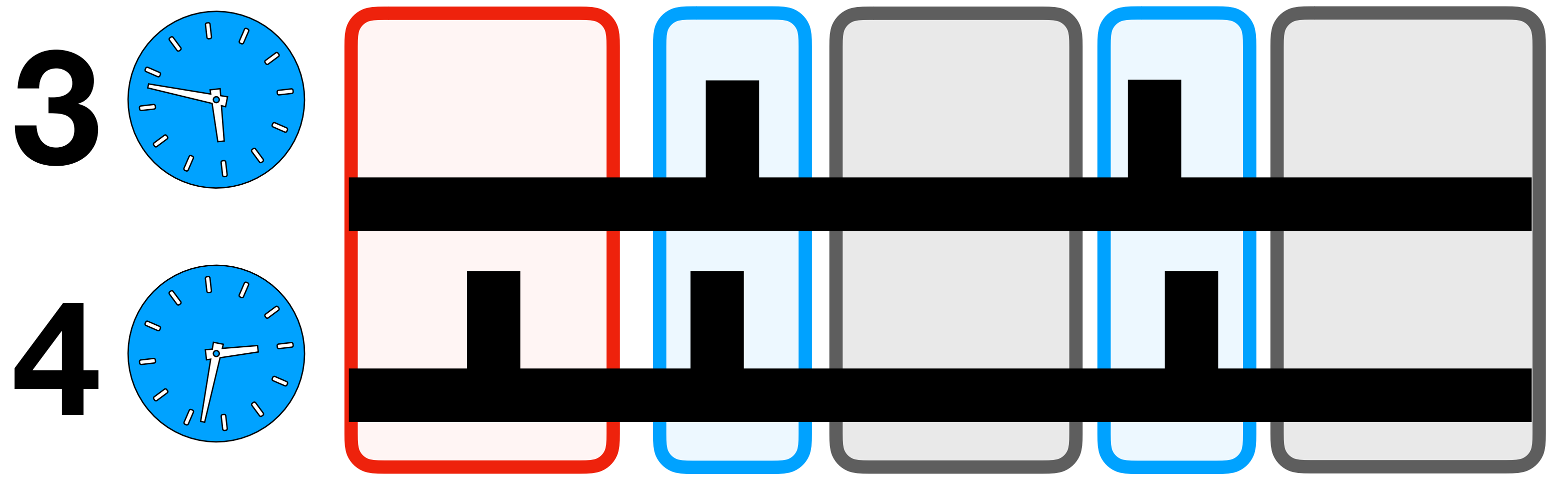
- n_0 = 2 nodes
- f_0 = 0 resilience



+

Pulser A$_1$

- n_1 = 2 nodes
- f_1 = 0 resilience

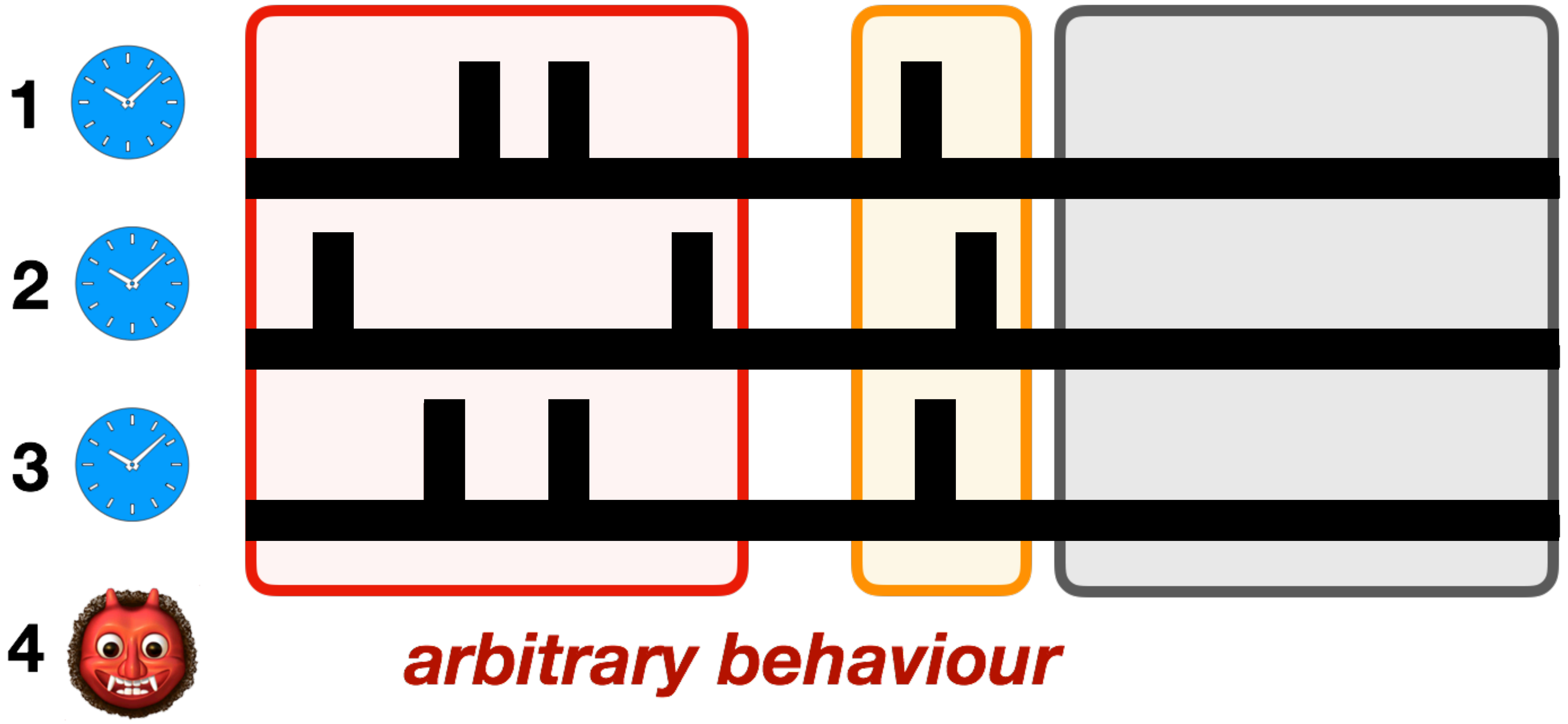


Result:

Resynchroniser B

- $n = n_0 + n_1$
- $f = f_0 + f_1 + 1$

Pulser A$_0$ Pulser A$_1$



Resilience boosting: details

Given:

Resynchroniser B

- $n = n_0 + n_1$
- $f = f_0 + f_1 + 1$

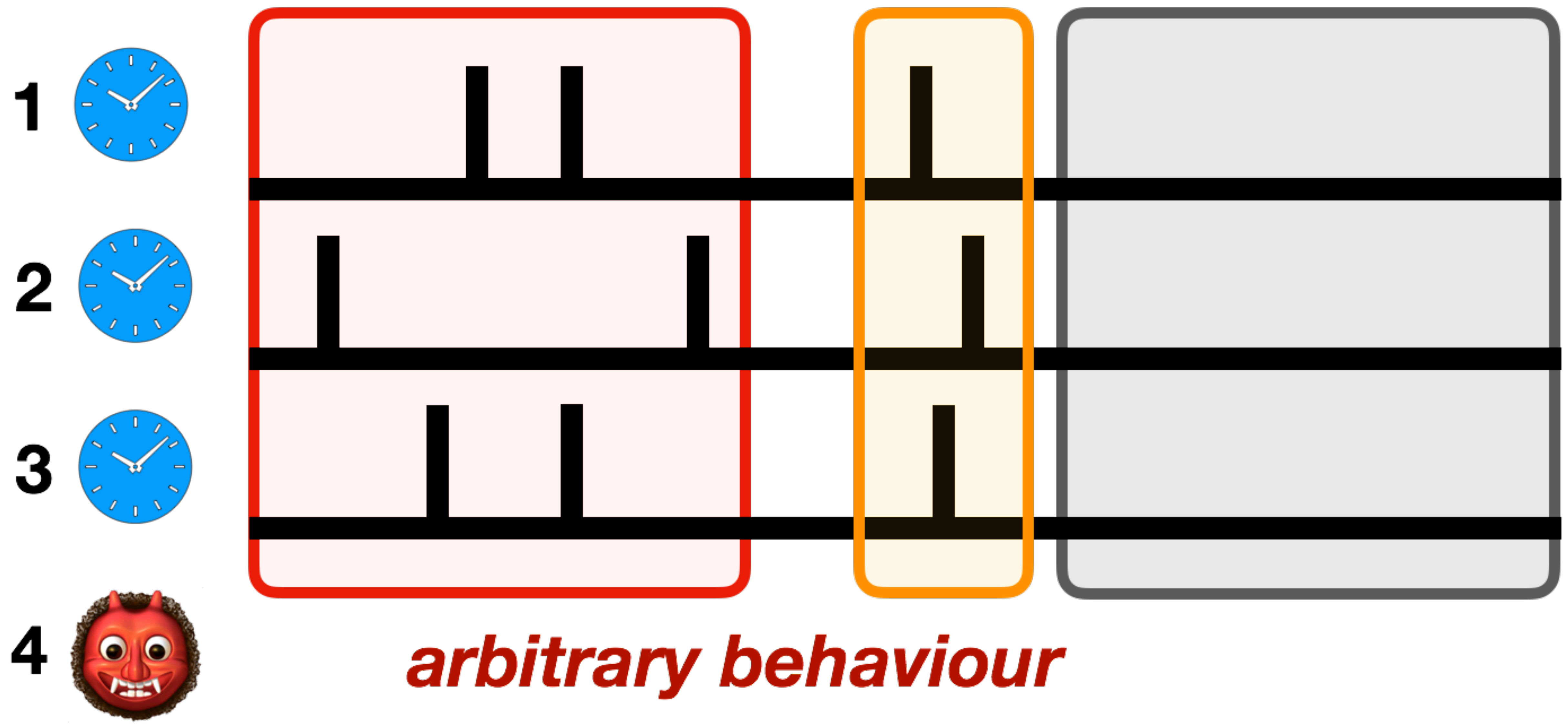
Pulser $\mathbf{A_0}$ | Pulser $\mathbf{A_1}$

\+

Wrapper W

Consensus C

- $n = n_0 + n_1$
- $f = f_0 + f_1$
- synchronous



Resilience boosting: details

Given:

Resynchroniser B

- $n = n_0 + n_1$
- $f = f_0 + f_1 + 1$

Pulser $\mathbf{A_0}$ Pulser $\mathbf{A_1}$

+

Wrapper W

Consensus C

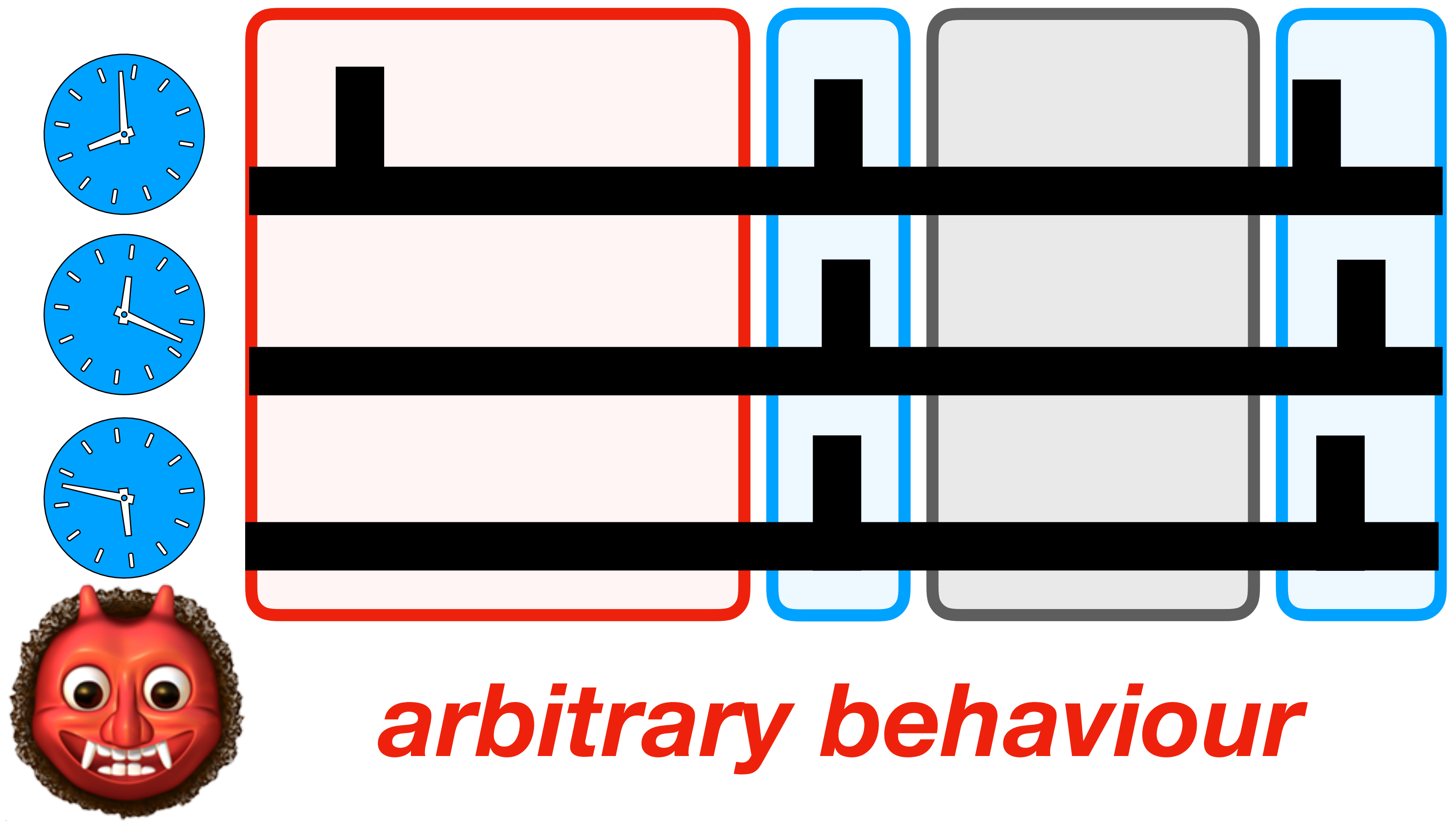
- $n = n_0 + n_1$
- $f = f_0 + f_1$
- synchronous

Result:

Pulser A

- $n = n_0 + n_1$ nodes
- $f = f_0 + f_1$ **+ 1** resilience

B: $\mathbf{A_0}$ $\mathbf{A_1}$ — **C**



Hardness of pulse synchronisation

Synchronous BF consensus

$\Downarrow^*$

SS + BF pulse synchronisation in the bounded delay model

(almost)
as easy as
consensus

Hardness of pulse synchronisation

Synchronous BF consensus

$\Uparrow$ **???**

SS + BF pulse synchronisation in the bounded delay model

but is it **as hard as** consensus?

Conclusions

Upper bounds:

	stabilisation time	bandwidth
	**polylog *f***	**polylog *f***
	**O(log *f*)**	poly *f*
	O(*f*)	**O(log *f*)**

Lower bounds:

No lower bounds known!

Conclusions

Upper bounds:

	stabilisation time	bandwidth
	polylog f	polylog f
	O(log f)	poly f
	O(f)	O(log f)

Lower bounds:

No lower bounds known!

Thanks!

arXiv:1705.06173