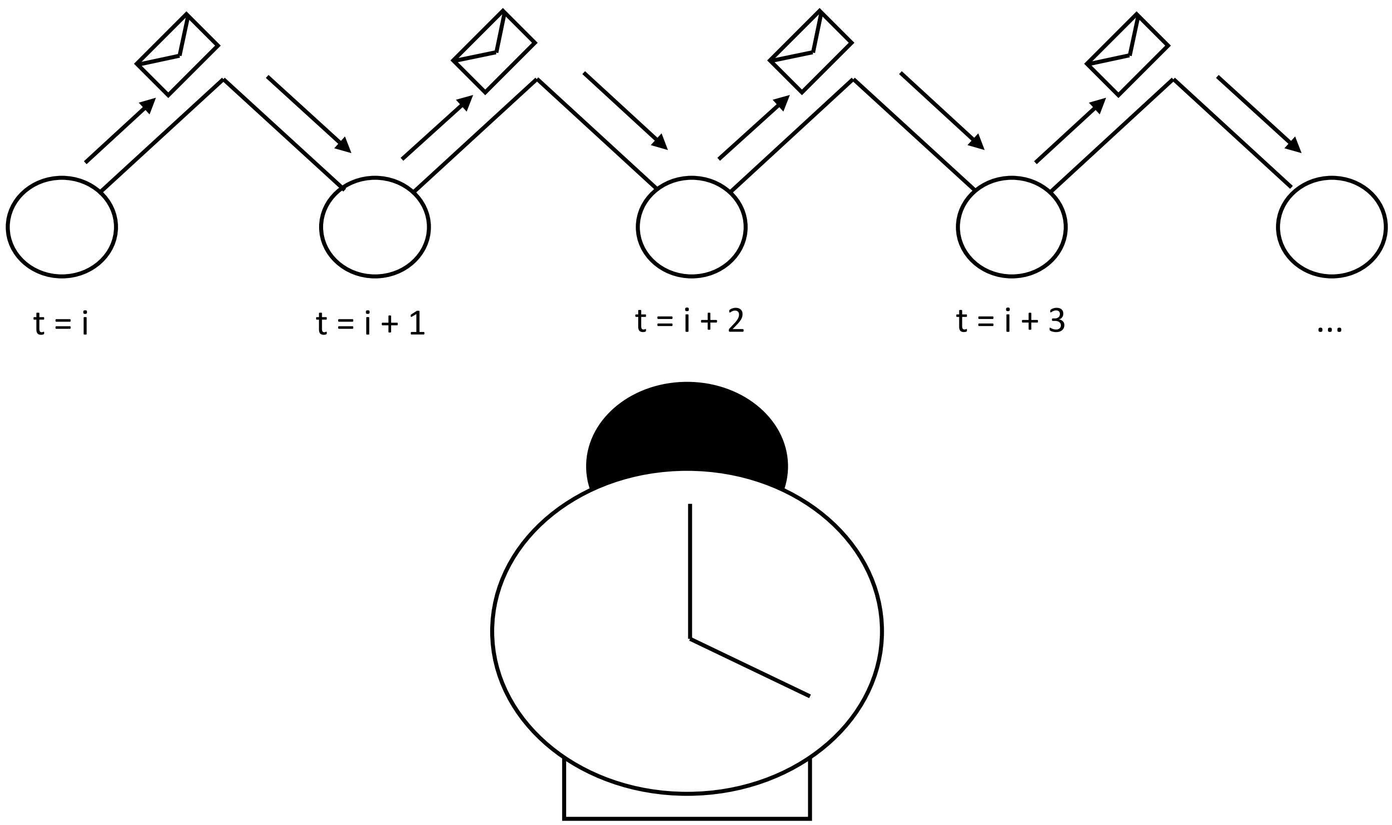
Synchronous Systems

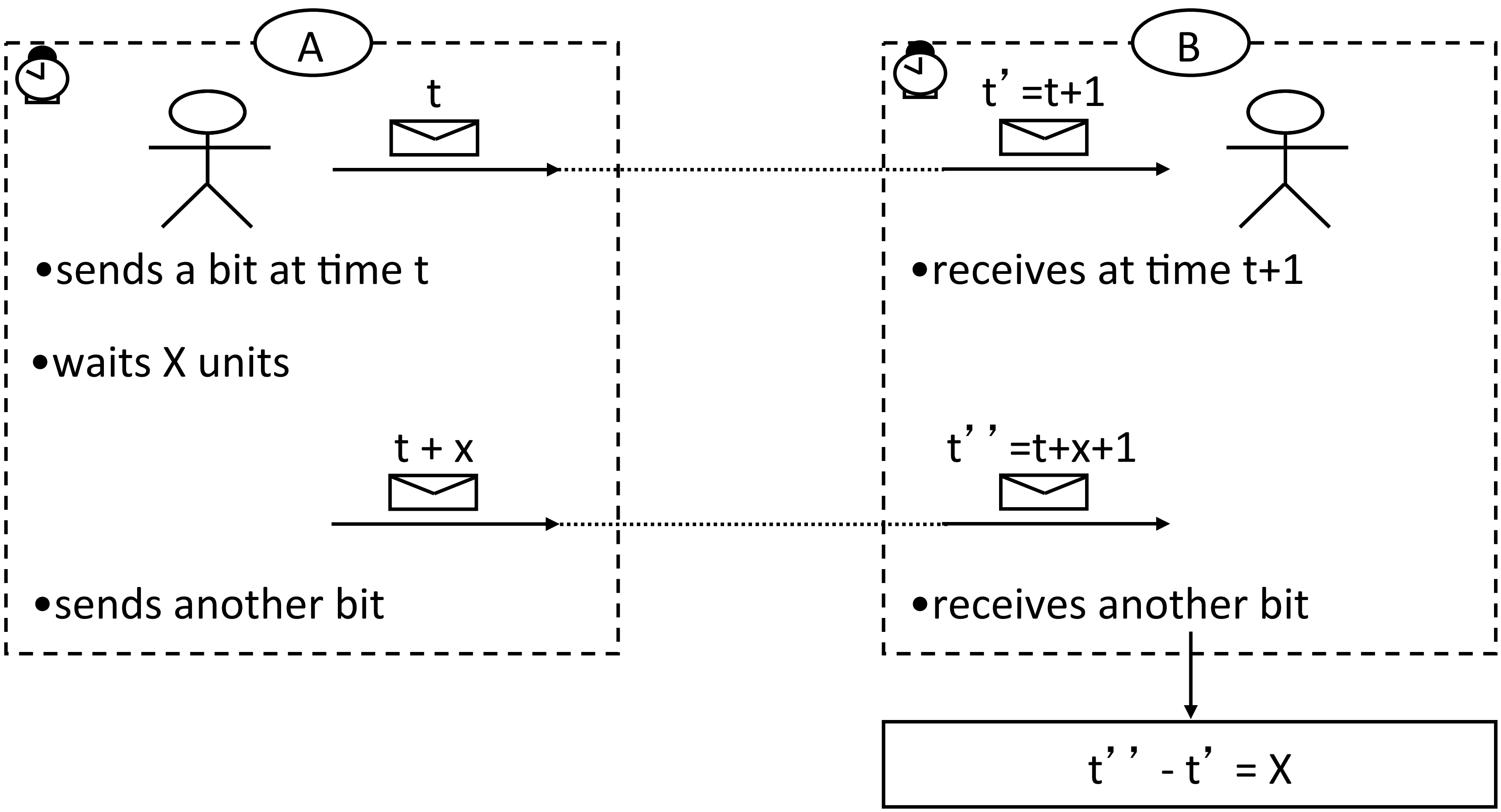


Overcoming Transmission Costs: 2-bit Communication

Any information can be transmitted using 2 BITS

Try to transmit the value 1.384.752.600 with 2 bits

A wants to send value X to B.

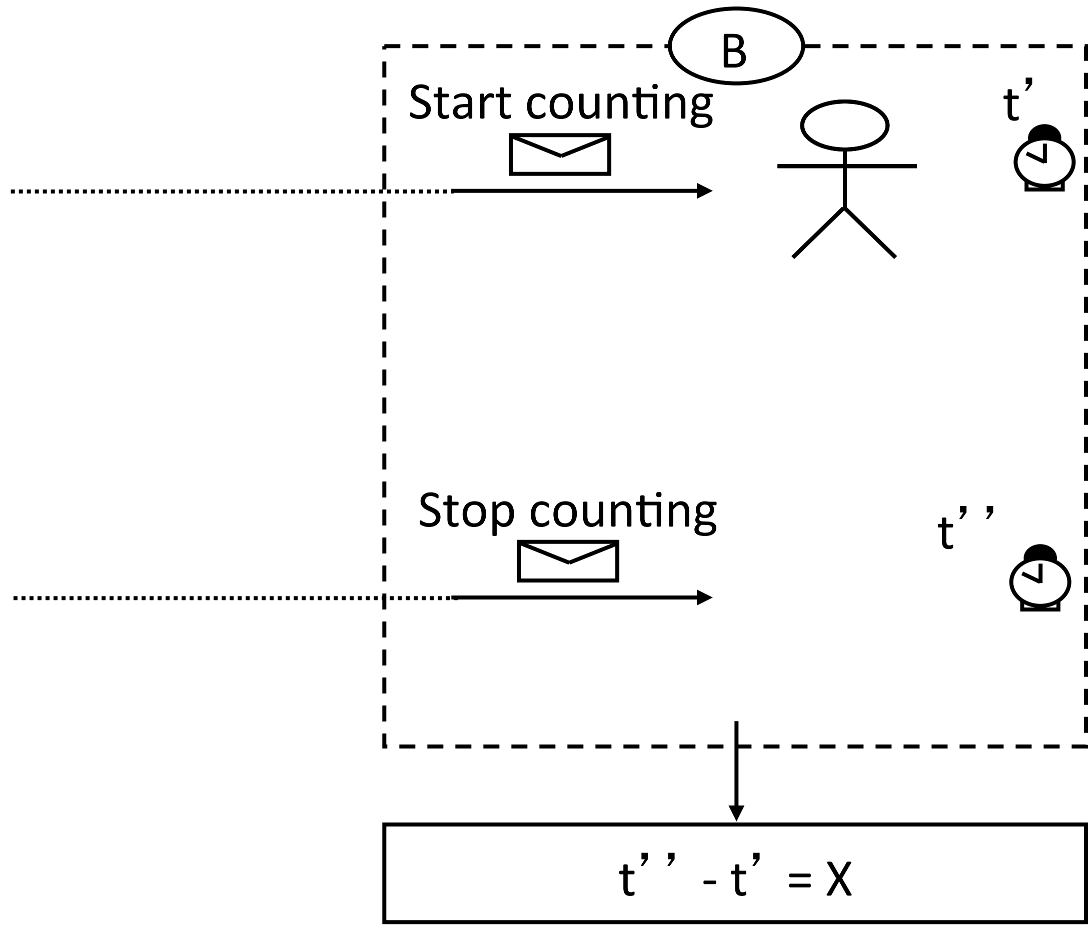


BITS: 2

TIME: X

Silence is expressive

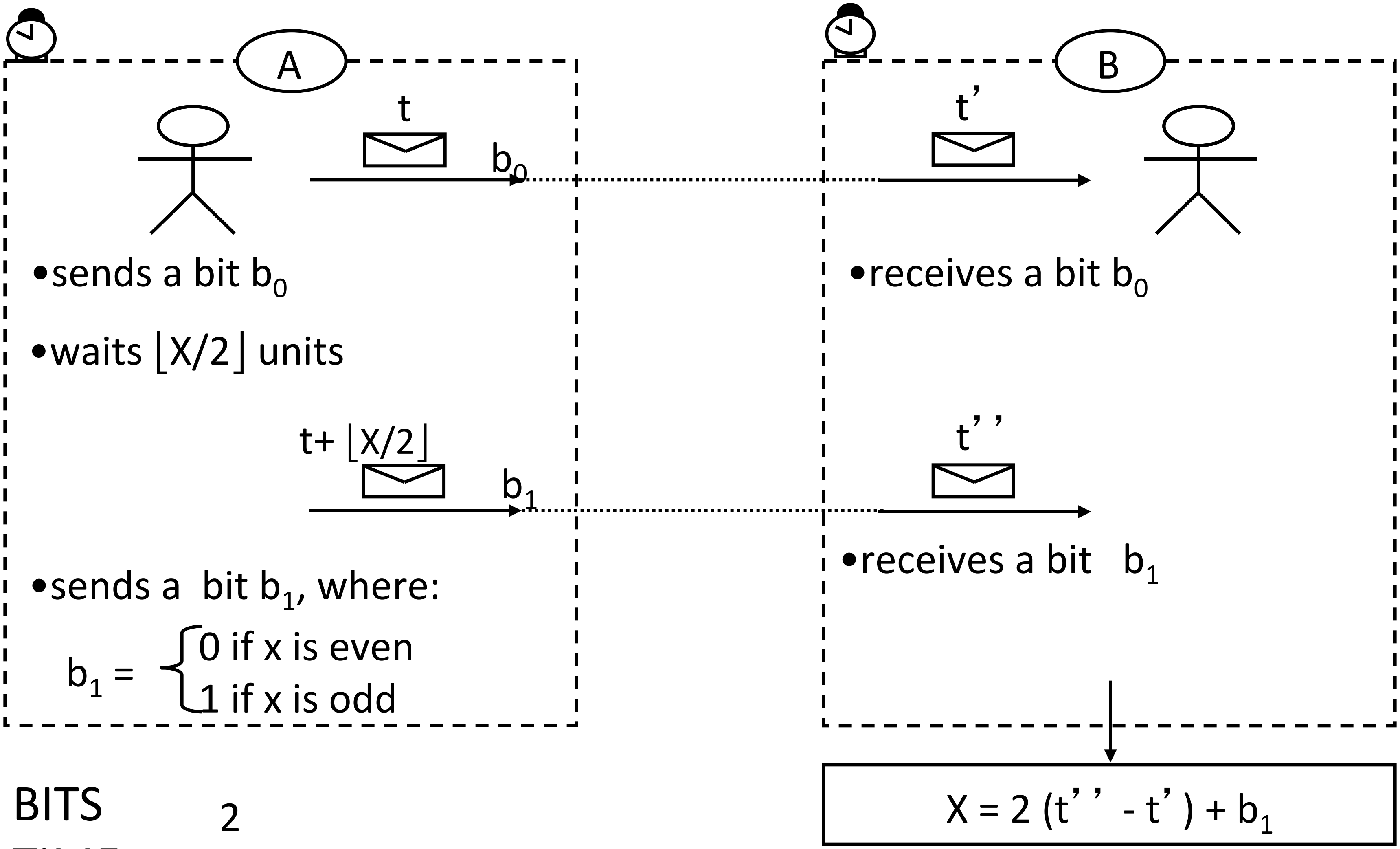
The protocol



2-Party Communication

2-bits Communicators

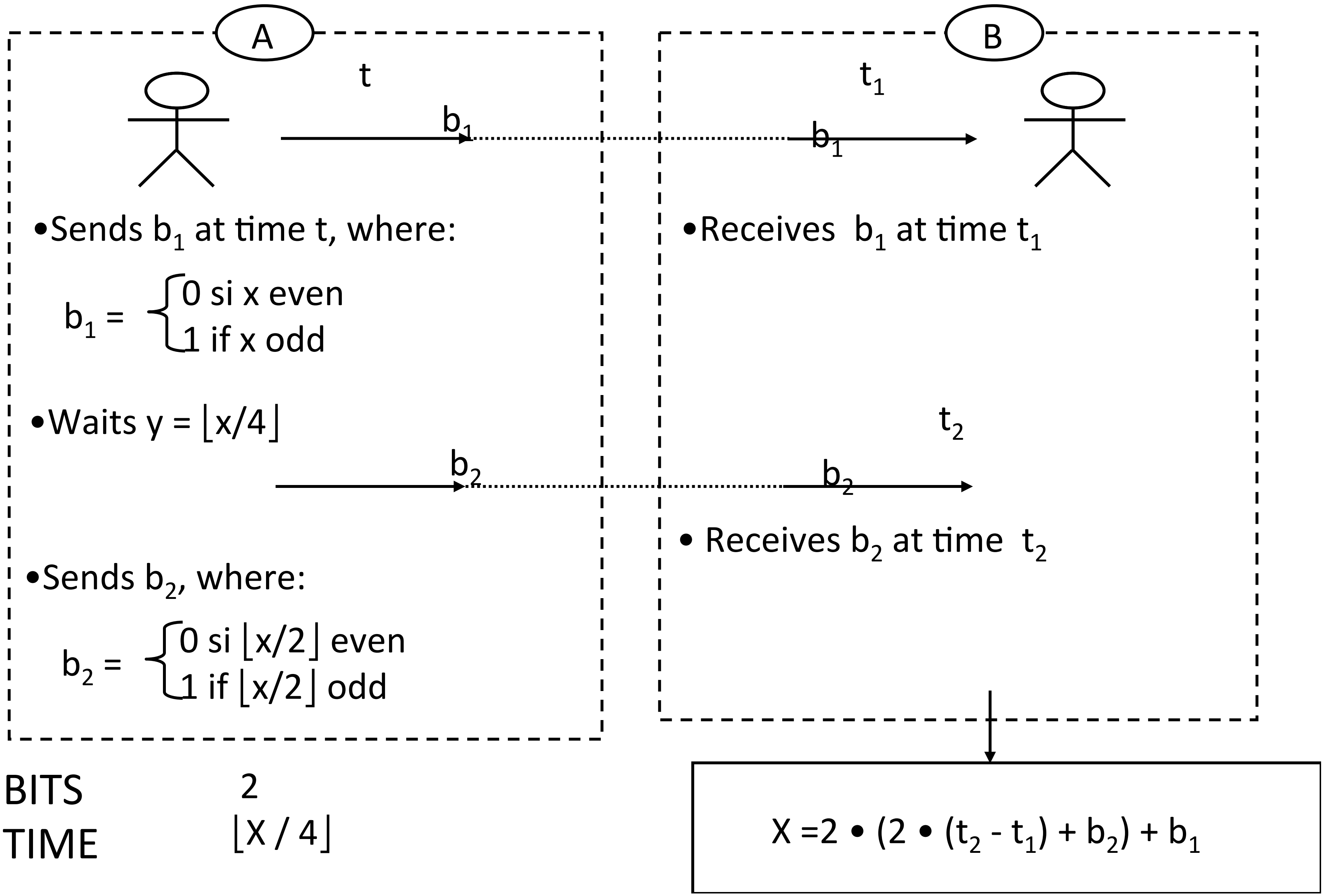
A wants to send value X to B.



BITS 2

TIME $\lfloor X/2 \rfloor$

A wants to send value X to B.



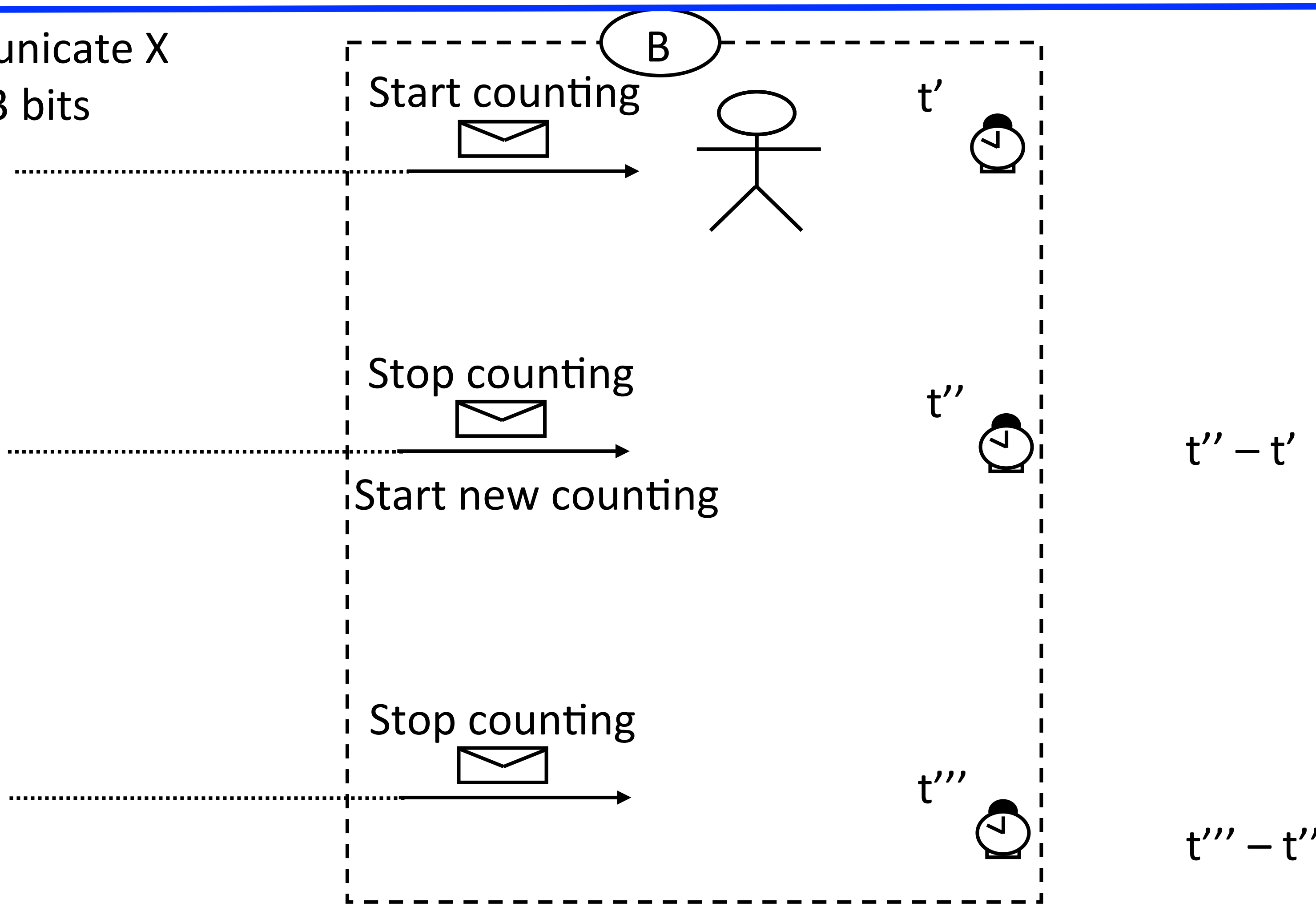
BITS 2

TIME $\lfloor X / 4 \rfloor$

BIT	TIME
2	X
2	$\lfloor X/2 \rfloor$
2	$\lfloor X/4 \rfloor$
3	?
4	?
...	...

3-Bit Communicators

To communicate X
with 3 bits

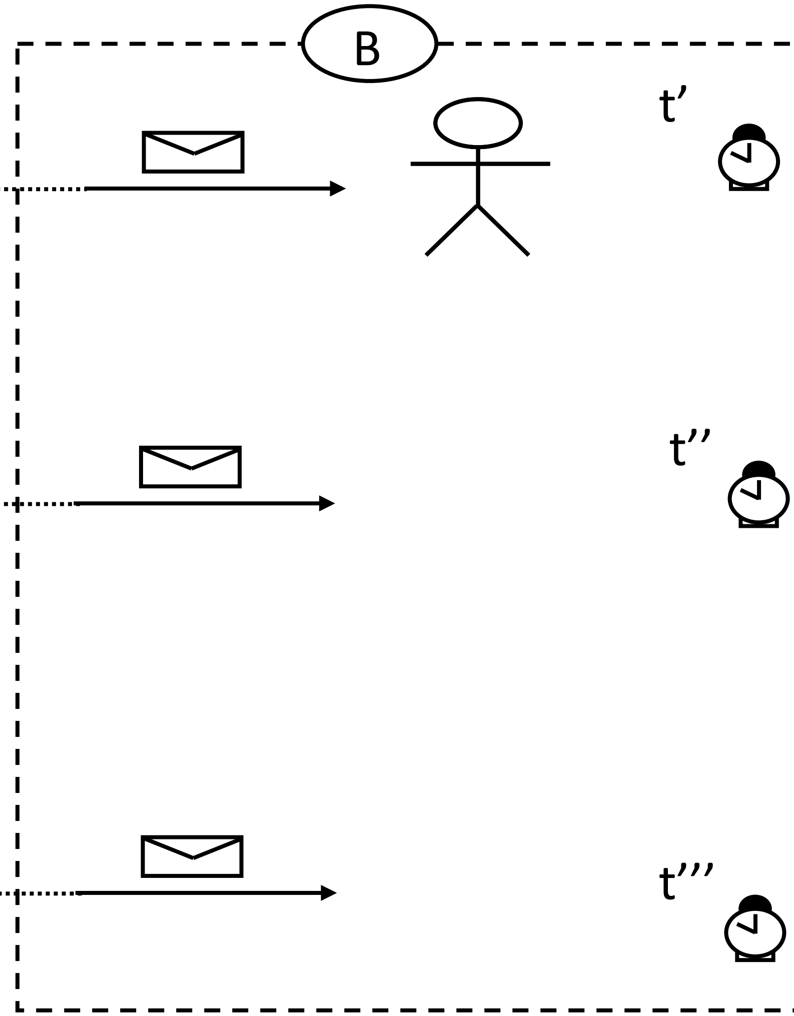


How much to wait so that X can be efficiently
reconstructed from (t’’-t’) and (t’’’-t’’) ?

To communicate X with 3 bits

3-Bit Communicators

(Ex: X=40)



Send a bit
at time 0

Receive at time t'
(at time 1)

Wait($\lceil\sqrt{X}\,\rceil$)

Wait(7)

Receive at time t''
(at time 8)

Send another bit

Compute t'' – t'
(=7)

Wait($\lceil\sqrt{X}\,\rceil^2$ -X)

Wait(49 - 40) = 9

Send another bit
at time 7+ 9 = 16

Receive at time t'''
(at time 17)

Compute t''' – t''
(= 9)

The receiver, to decode the information must compute:

$$(t'' - t')^2 - (t''' - t'')$$

In the example:

$$(\mathbf{7})^2 - (\mathbf{9}) = \mathbf{49-9} = \mathbf{40}$$

BITS	TIME
3	$O(\lceil \sqrt{X} \rceil)$

Another Example:

$X = 23$ $\lceil \sqrt{X} \rceil$ $\lceil \sqrt{X} \rceil^2 - X$

$q_0 = \lceil \sqrt{23} \rceil = \mathbf{5}$

$q_1 = 25\text{-}23 = \mathbf{2}$

$t = 0$ send b_0 receive b_0 $t_0 = 1$

wait **5**

send b_1 receive b_1 $t_1 = 6$

wait **2**

send b_2 receive b_2 $t_2 = 8$

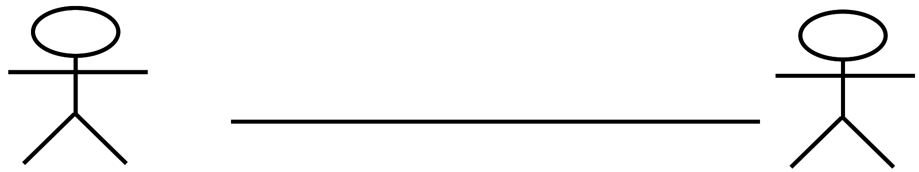
$$(t_1 - t_0)^2 - (t_2 - t_0)$$

$$25 - 2 = 23$$

k-bits Communicators

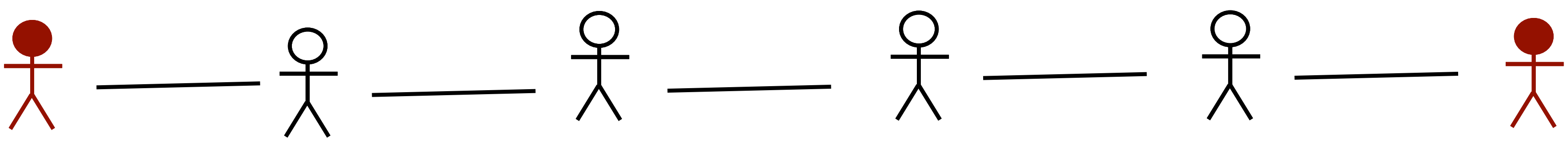
BITS	TIME
k	$O(\lceil k X^{1/k} \rceil)$

With communicators we achieve communication between neighbours



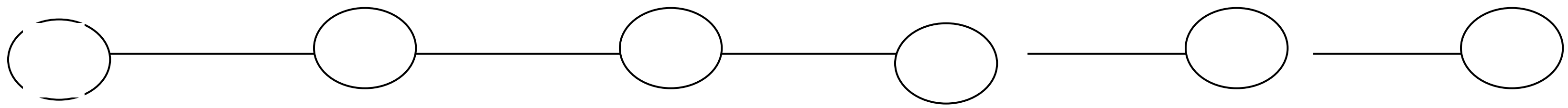
PIPELINE technique

To communicate at a distance



Example: With communicators

communicate: "30"



Start bit

0 → 1

wait 30

End bit 30 → 31 → 31

wait 30

61 → 62 → 63

wait 30

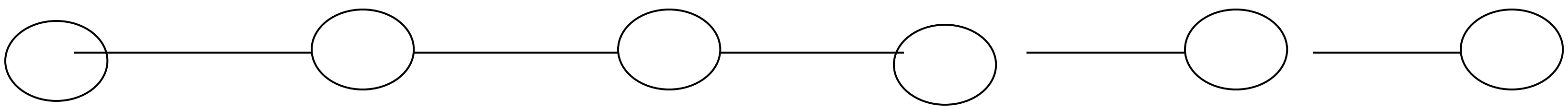
92 → 93 → 94

wait 30 …

123 → 124

With PIPELINE

communicate: “30”



Start bit

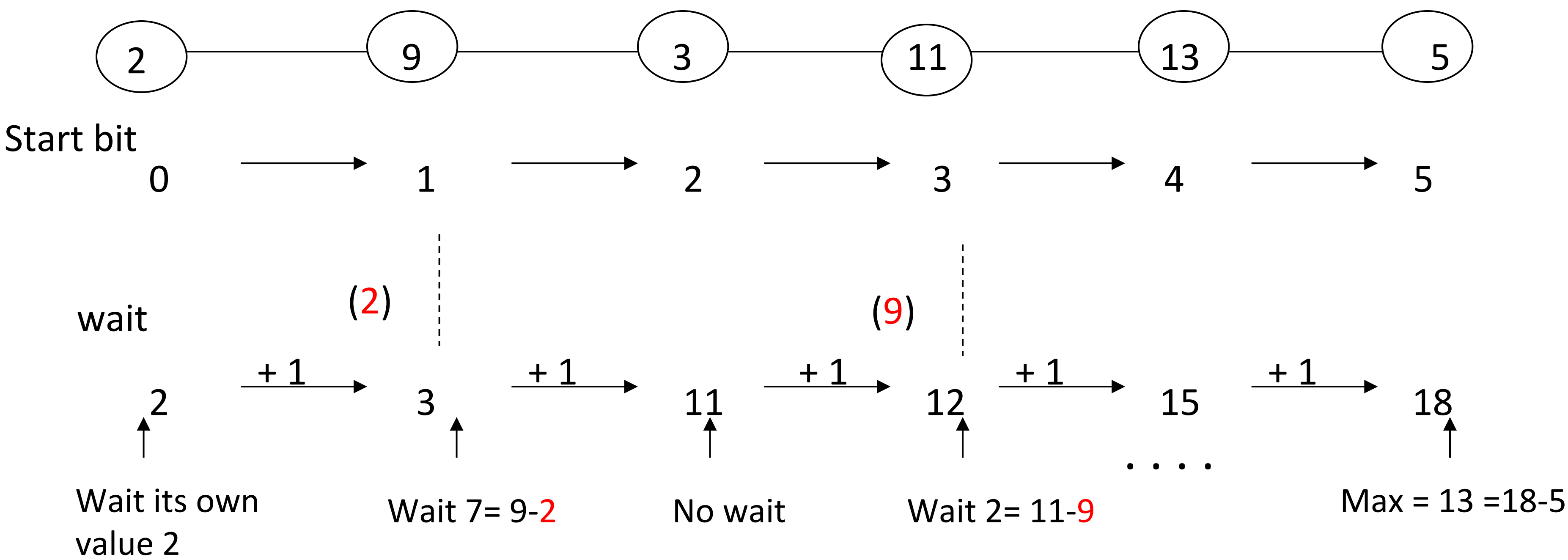
0 → 1 → 2 → 3 → 4 → 5

wait 30

End bit 30 → 31 → 32 → 33 → 34 → 35

PIPELINE

Example: communicate the maximum

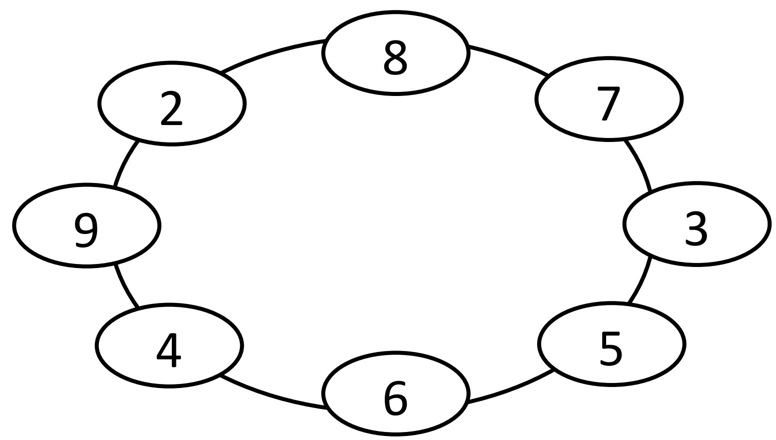


Min-finding and Election

overcoming the lower bound by “Speeding”

General Idea: Messages travel at different speed

- Knowledge of n is not necessary
- unidirectional version
- synchronous



We assume simultaneous start, but it is not necessary

Two ways of eliminating Ids:

- Like in AS FAR, large Ids are stopped by smaller Ids

-Small Ids travel **faster** so to catch up with larger Ids and eliminate them.

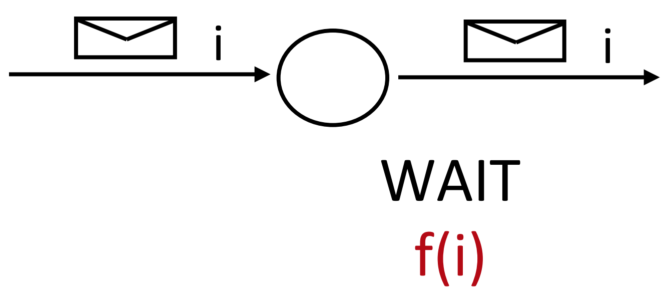
Each message travels at a speed which depends on the identity it contains.

Identity i travel at some speed f(i)

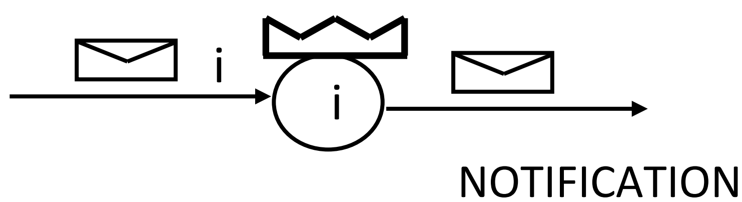
Speed is assumed to be unitary, the same for every message. How can we change it ?

By introducing appropriate **DELAYS**

When a node receives a message containing *i*, it waits f(i) ticks.

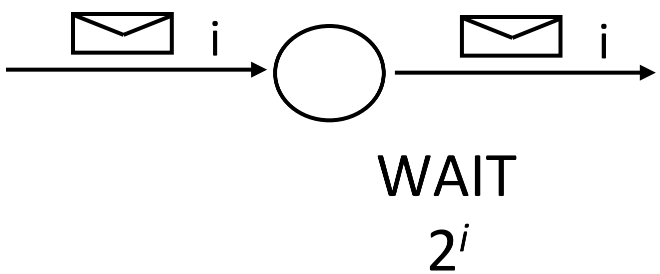


When a node receives its own id, it becomes the leader and send a notification message around the ring. This message will not be delayed.

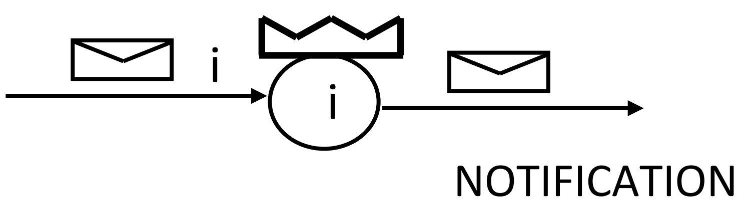


E.g., $f(i) = 2^i$

When a node receives a message containing *i*, it waits 2^i ticks.



When a node receives its own id, it becomes the leader and send a notification message around the ring. This message will not be delayed.

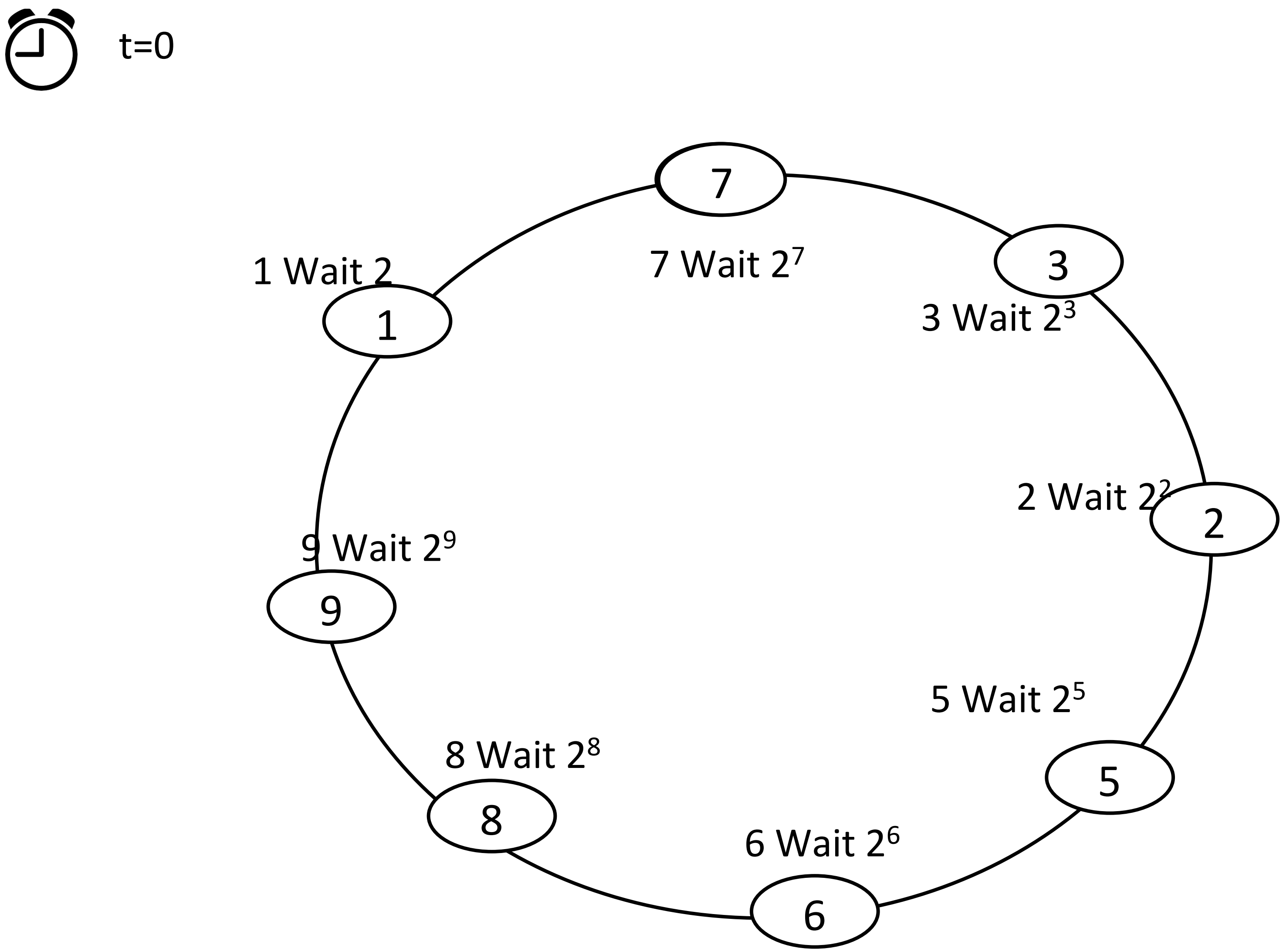


In time $2^i n + n$ the smallest Id i traverses the ring

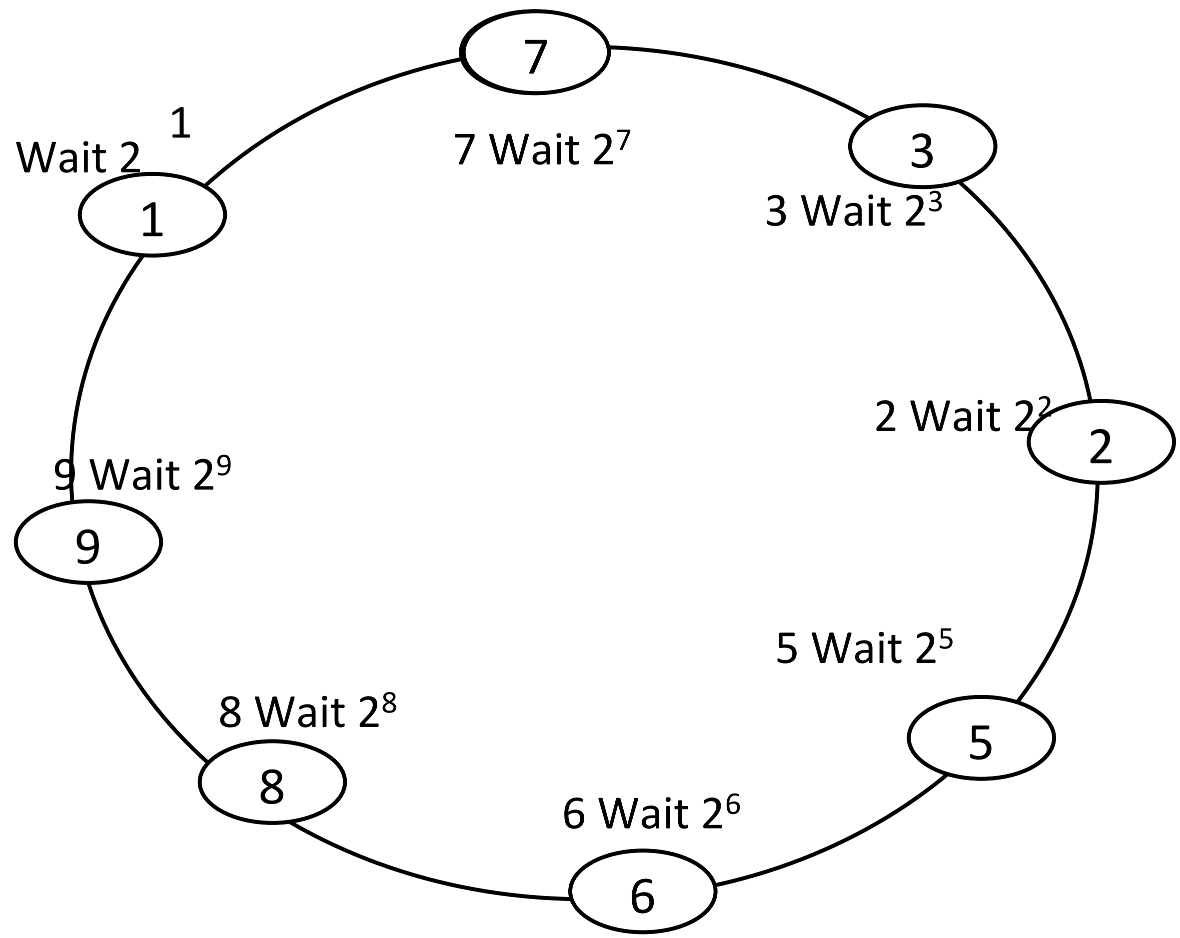
Let the second smallest be i+1 (with waiting time 2^{i+1}).
How many links does it have the time to traverse (at most) while the smallest Id goes around ?

$(2^i n+n) / 2^{i+1}$

roughly n/2 links

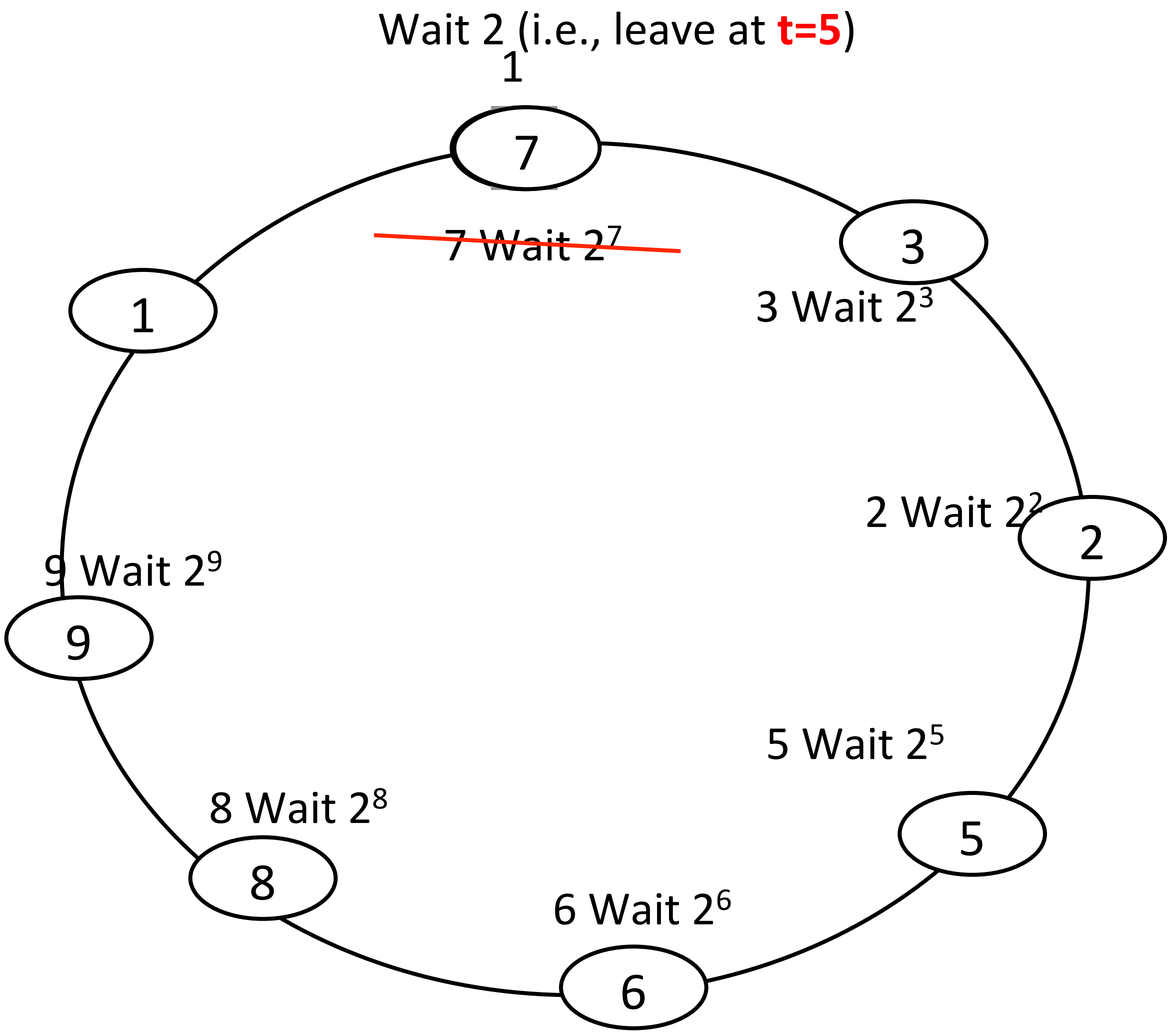

t=0
7
7 Wait 2^7
1 Wait 2
1
3
3 Wait 2^3
2 Wait 2^2
2
9 Wait 2^9
9
5 Wait 2^5
5
8 Wait 2^8
8
6 Wait 2^6
6

t=2

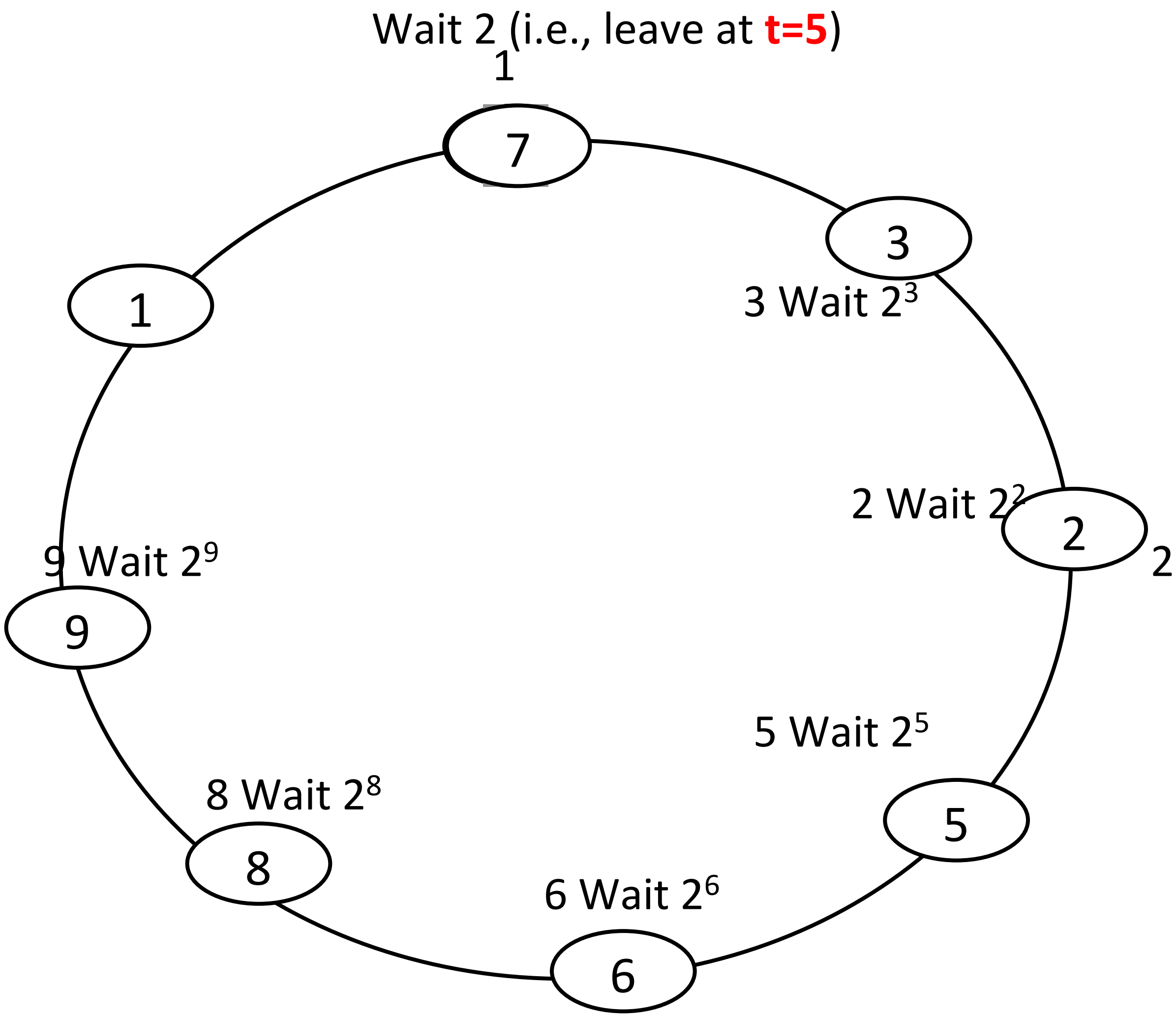


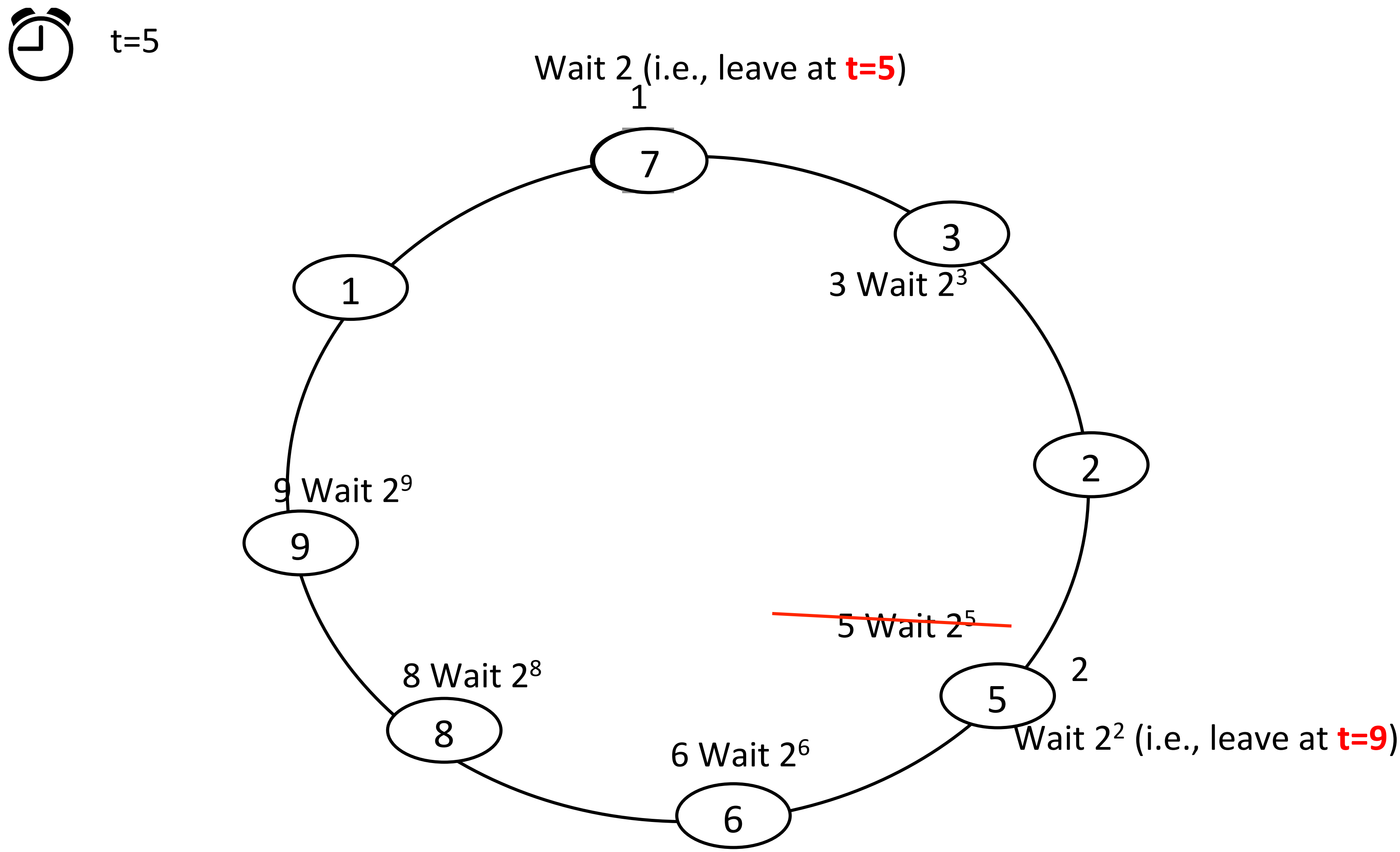
7
7 Wait 2^7
3
3 Wait 2^3
1
Wait 2
1
2 Wait 2^2
2
9 Wait 2^9
9
5 Wait 2^5
5
8 Wait 2^8
8
6 Wait 2^6
6

t=3


Wait 2 (i.e., leave at t=5)
1
7
7 Wait 2^7
3
3 Wait 2^3
1
2 Wait 2^2
2
9 Wait 2^9
9
5 Wait 2^5
5
8 Wait 2^8
8
6 Wait 2^6
6

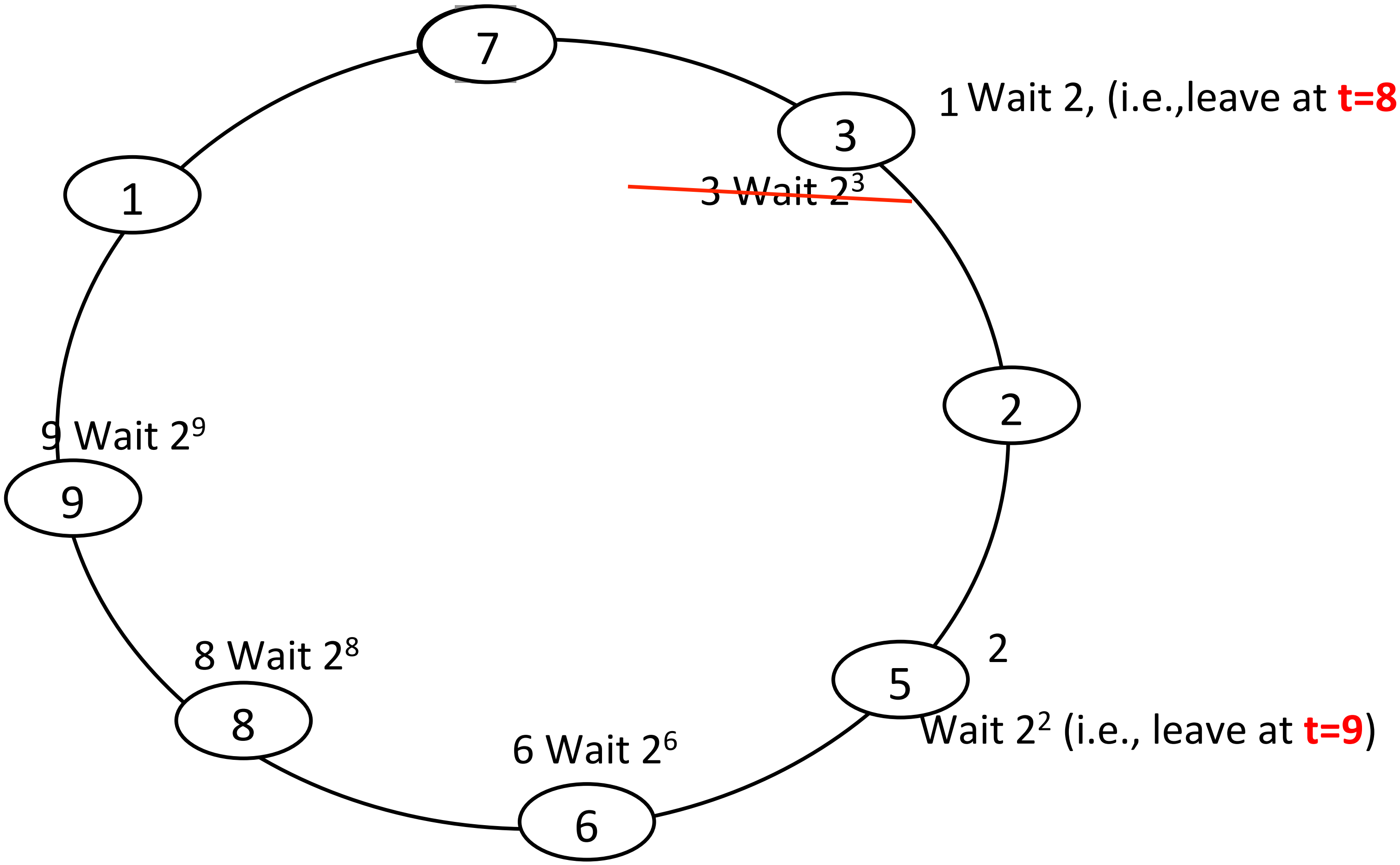
t=4


Wait 2 (i.e., leave at **t=5**)
1
7
3
3 Wait 2^3
1
2 Wait 2^2
2
2
9 Wait 2^9
9
5 Wait 2^5
5
8 Wait 2^8
8
6 Wait 2^6
6



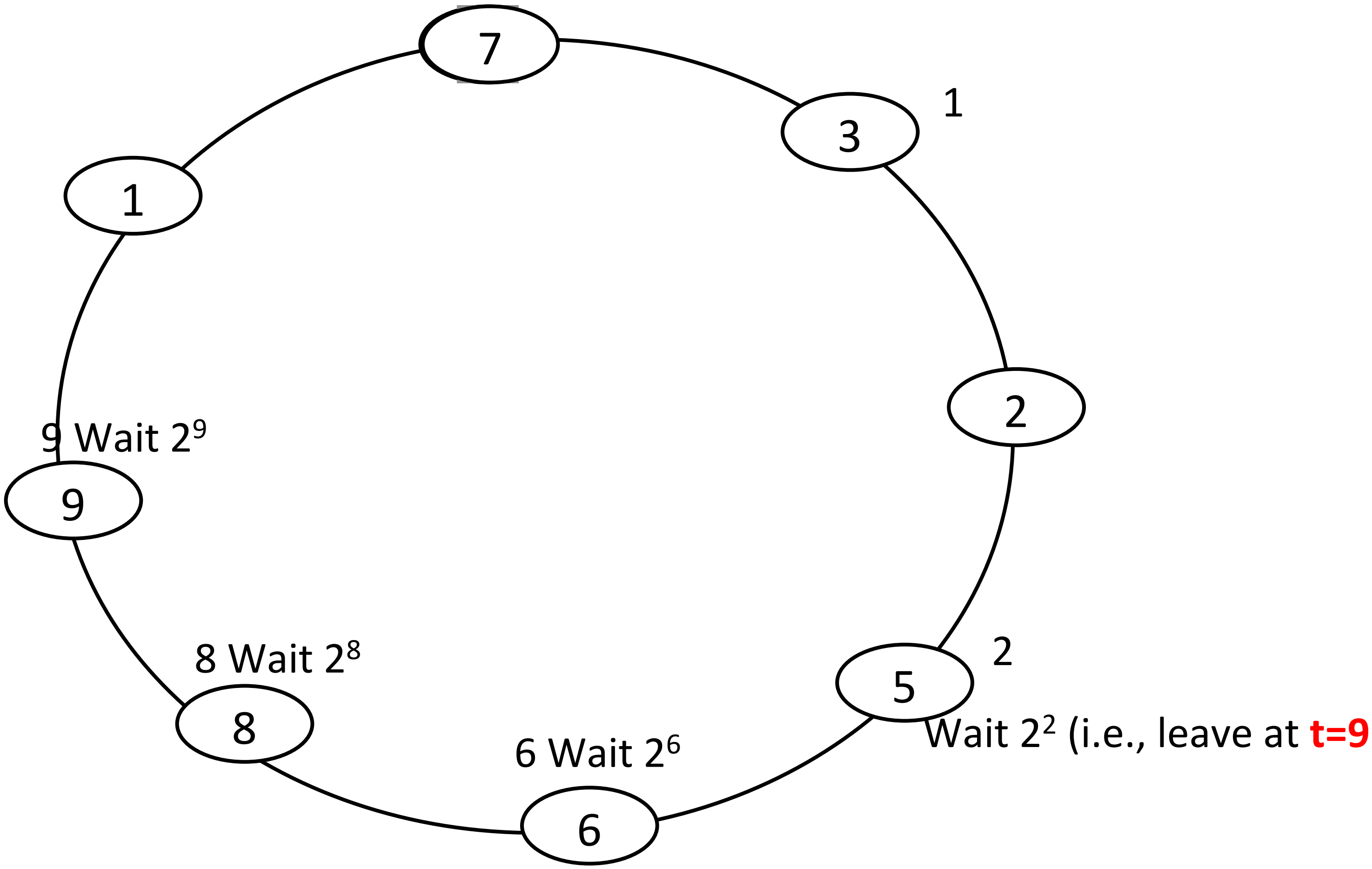
t=5
Wait 2 (i.e., leave at t=5)
1
7
3
3 Wait 2^3
1
2
9 Wait 2^9
9
5 Wait 2^5
2
5
Wait 2^2 (i.e., leave at t=9)
8 Wait 2^8
8
6 Wait 2^6
6

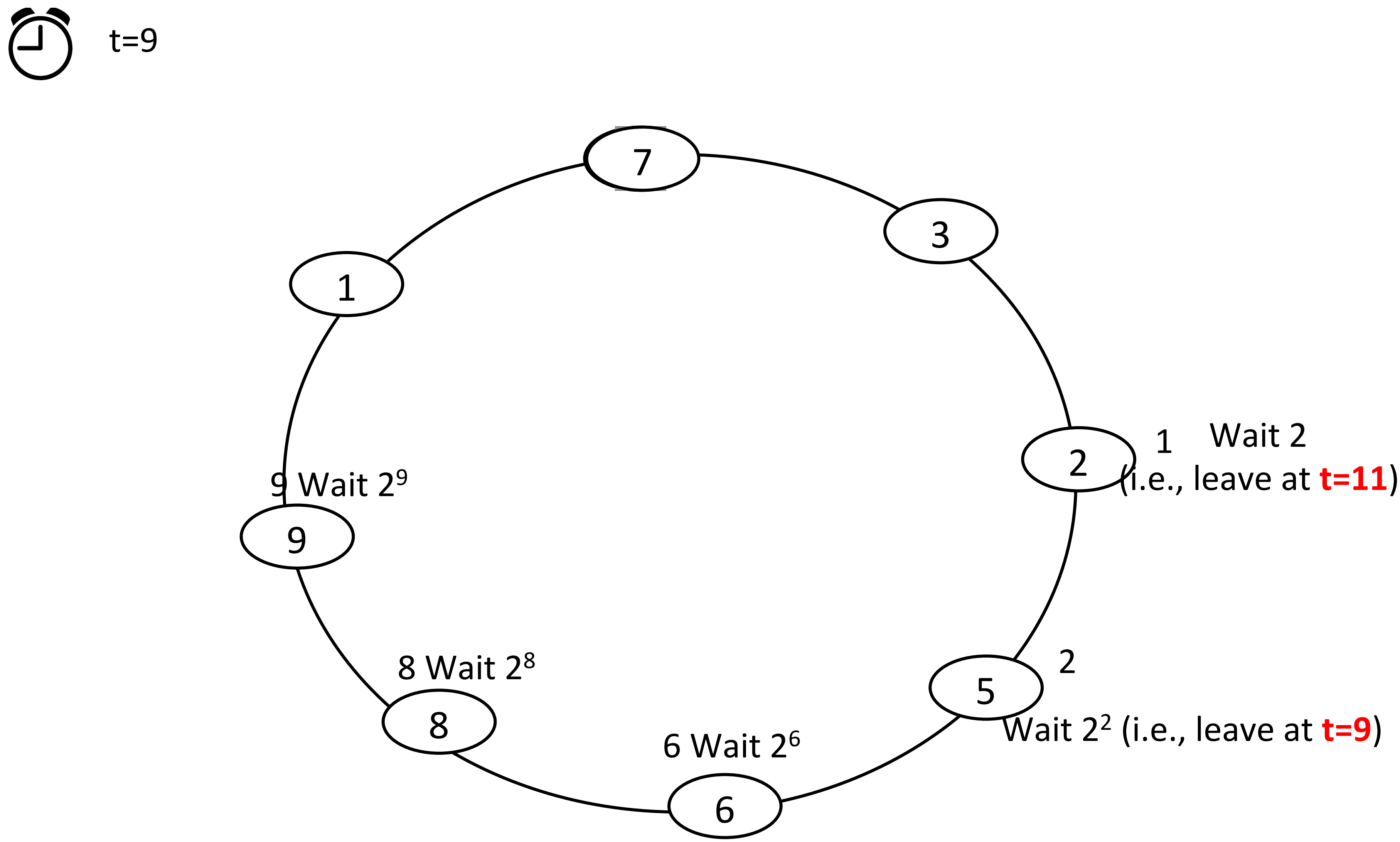
t=6



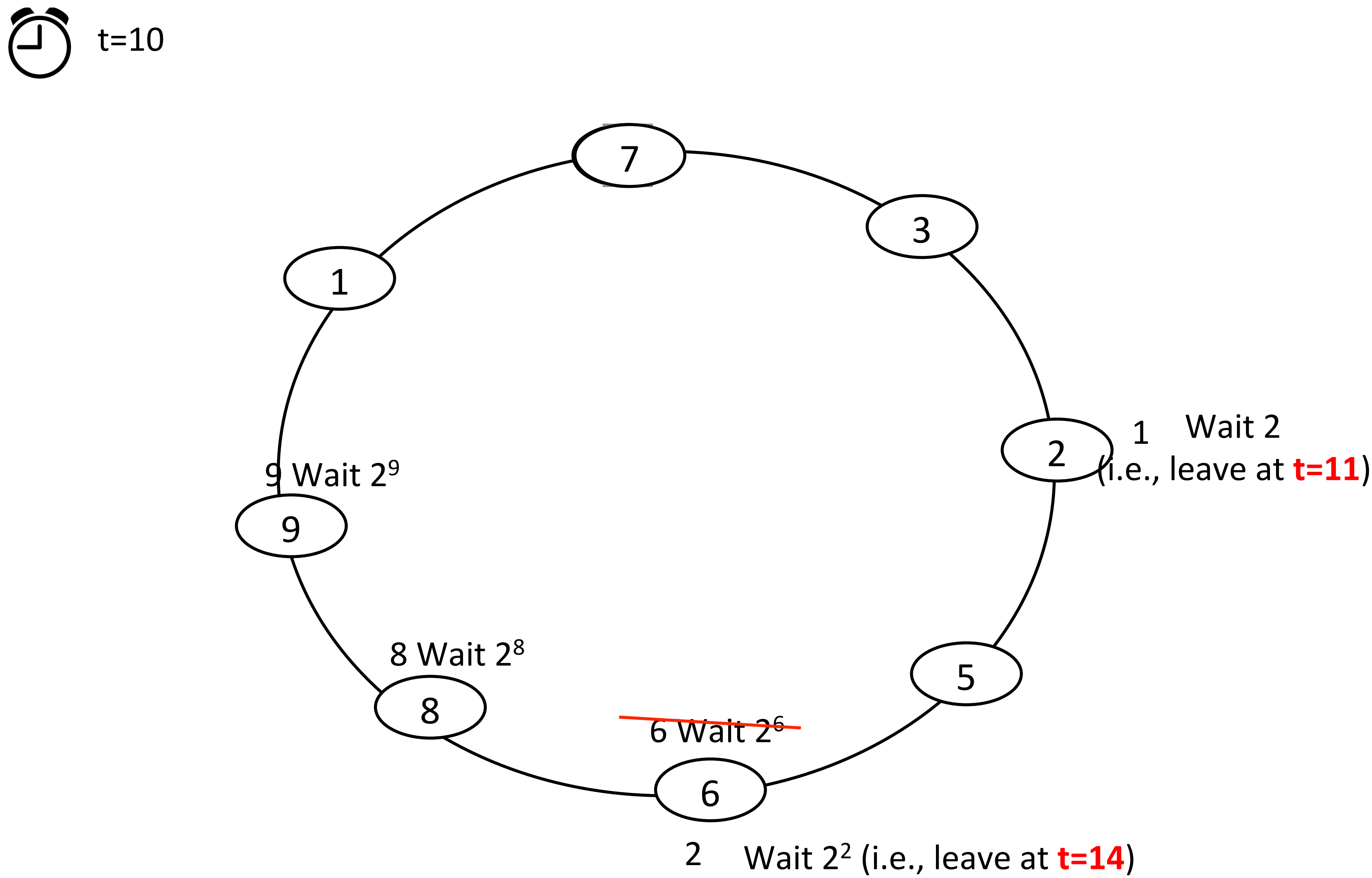
7
3
1 Wait 2, (i.e.,leave at **t=8**)
3 Wait 2^3
1
2
9 Wait 2^9
9
2
5
Wait 2^2 (i.e., leave at **t=9**)
8 Wait 2^8
8
6 Wait 2^6
6

t=8

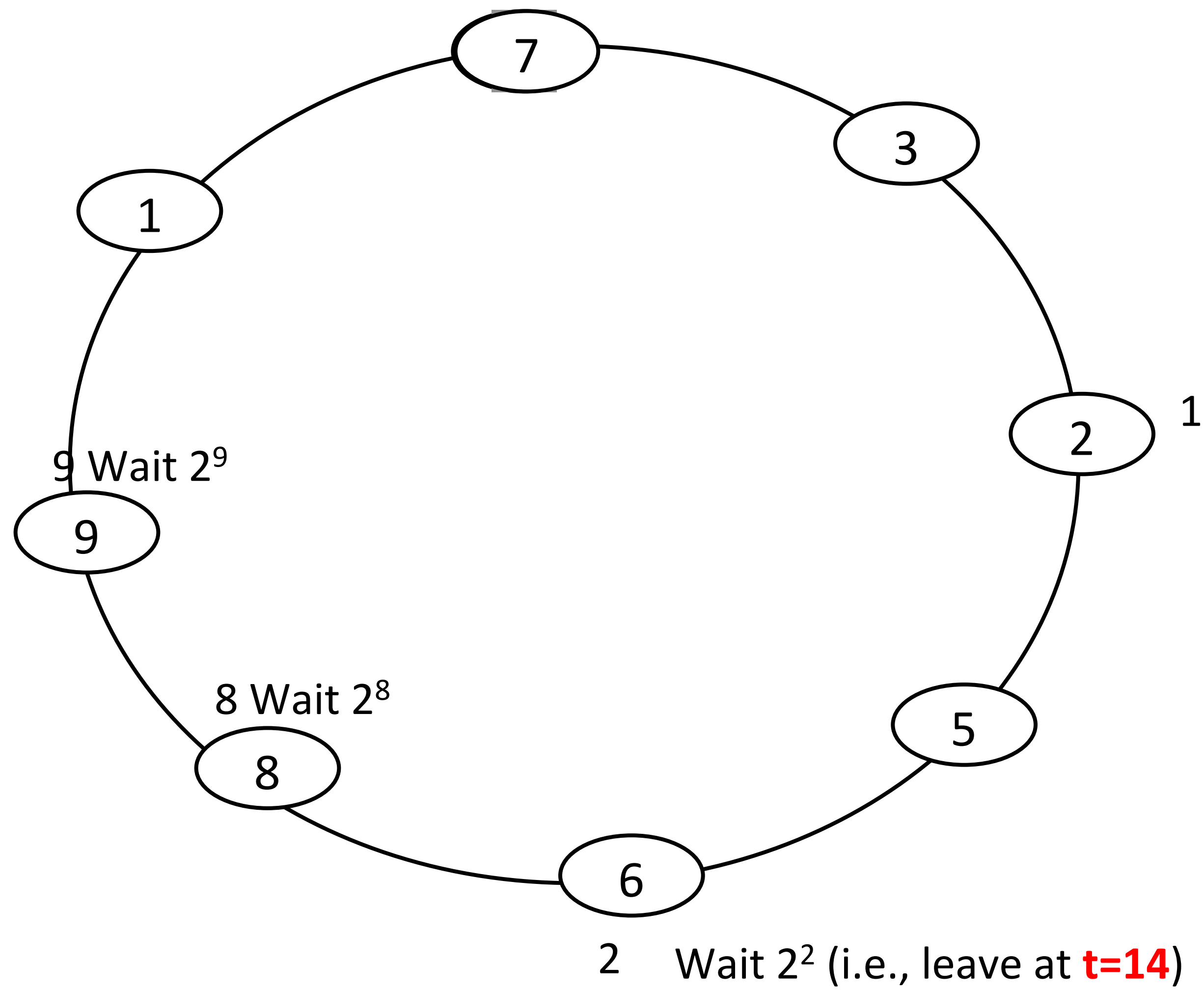


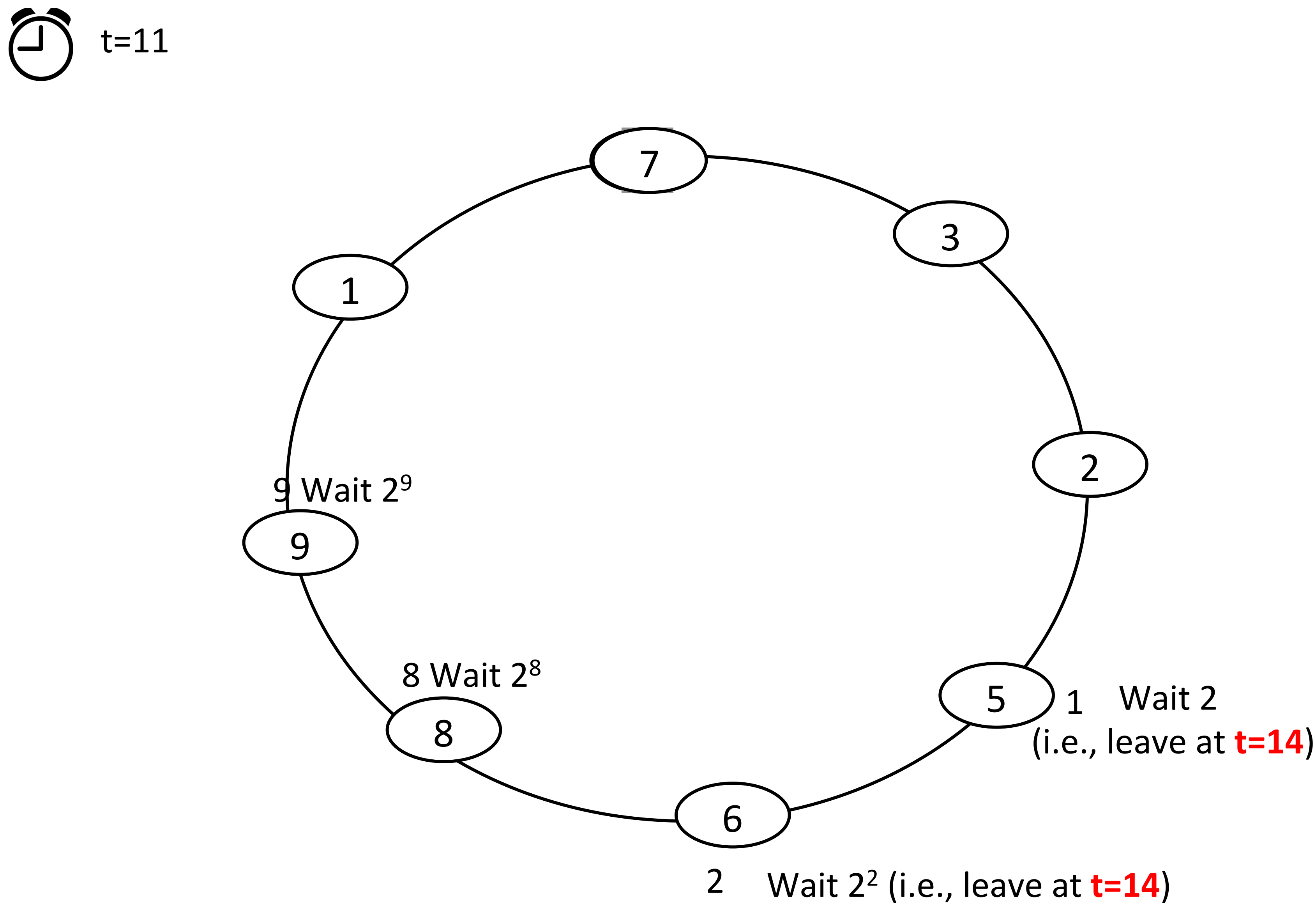


t=9
7
3
1
2
1
Wait 2
(i.e., leave at t=11)
9 Wait 2^9
9
2
5
Wait 2^2 (i.e., leave at t=9)
8 Wait 2^8
8
6 Wait 2^6
6


t=10
7
3
1
2
1 Wait 2
(i.e., leave at t=11)
9 Wait 2^9
9
8 Wait 2^8
8
5
6 Wait 2^6
6
2 Wait 2^2 (i.e., leave at t=14)

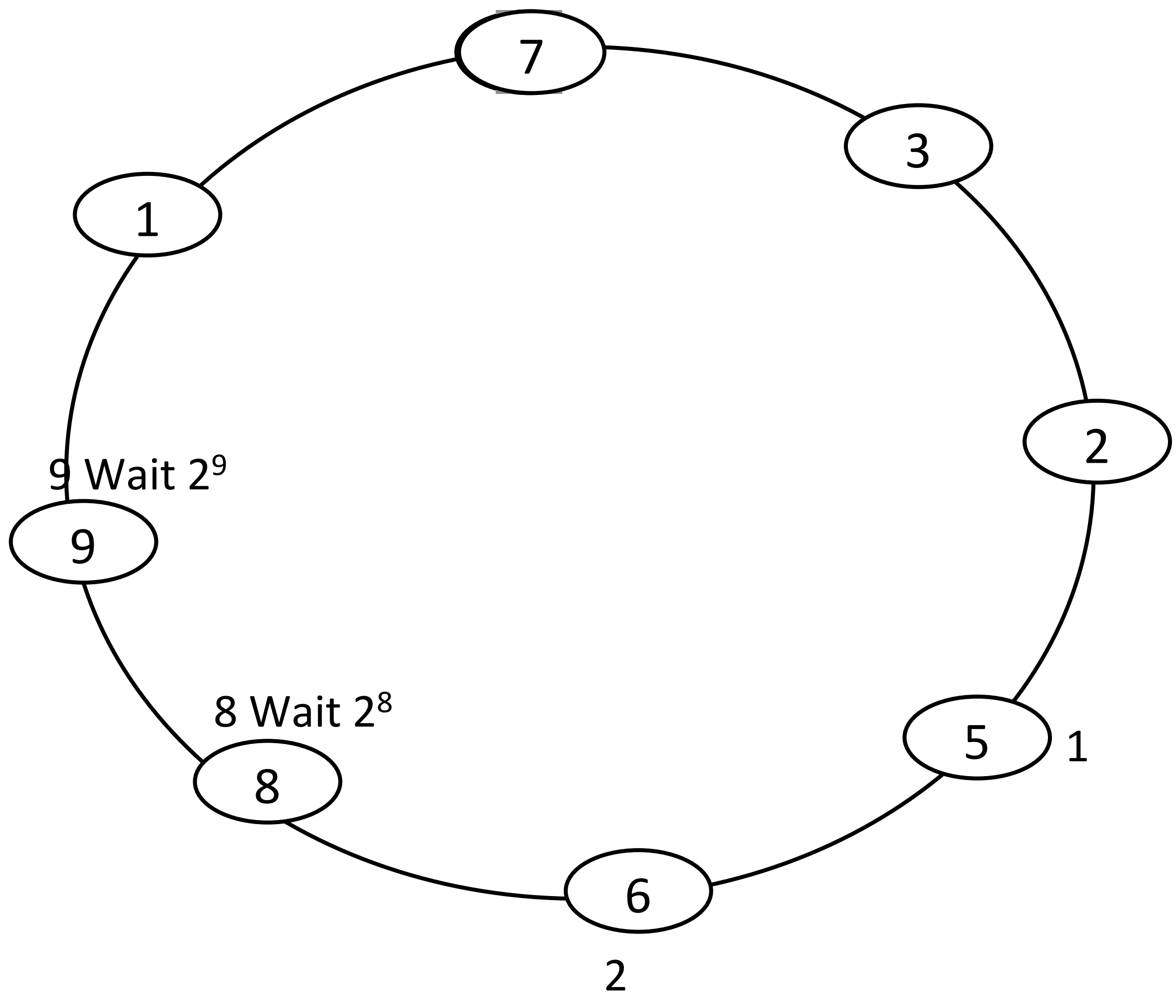
t=11

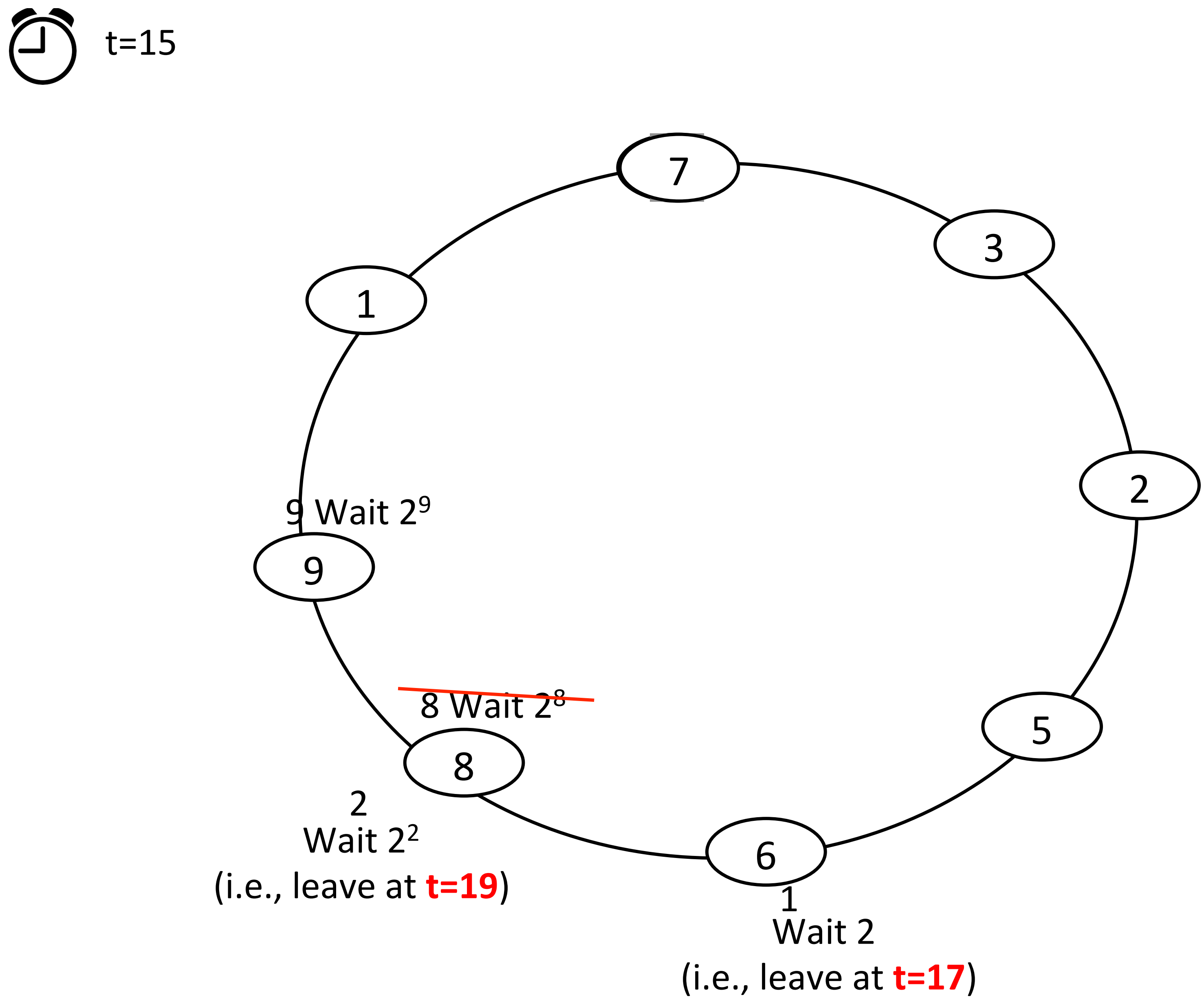


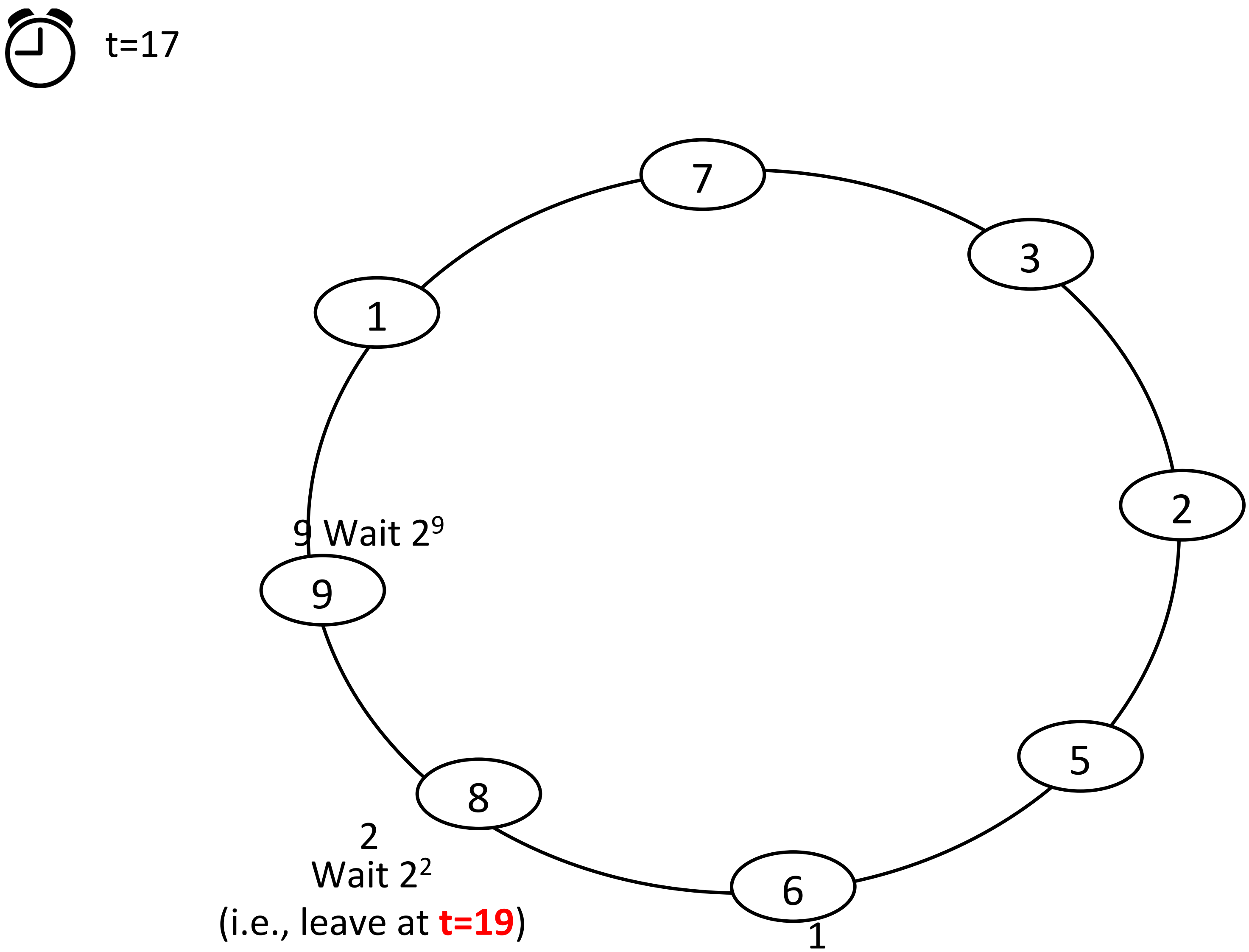


t=11
7
3
1
2
9 Wait 2^9
9
8 Wait 2^8
5
1 Wait 2
(i.e., leave at **t=14**)
8
6
2 Wait 2^2 (i.e., leave at **t=14**)

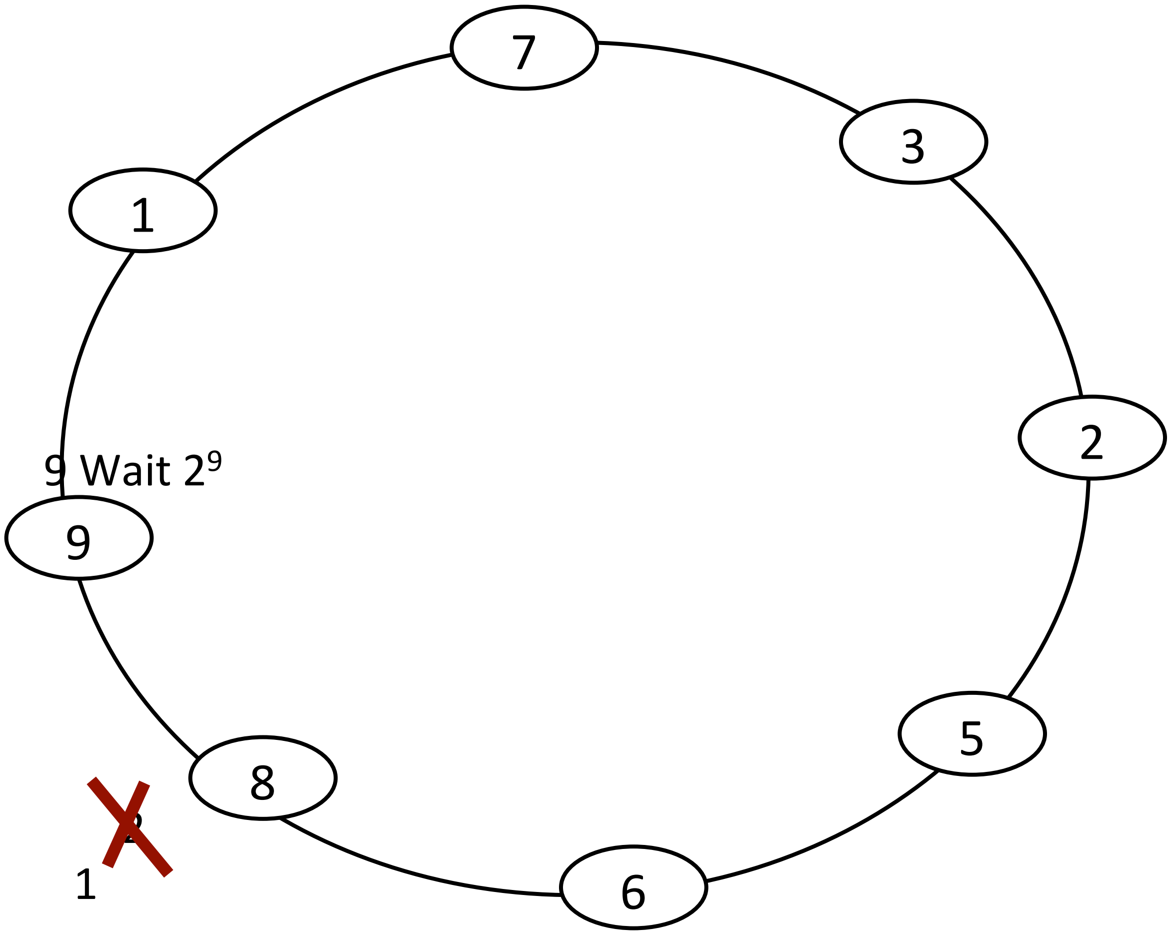
t=14




t=15
7
3
1
2
9 Wait 2^9
9
8 Wait 2^8
5
8
2
Wait 2^2
(i.e., leave at t=19)
6
1
Wait 2
(i.e., leave at t=17)


t=17
7
3
1
2
9 Wait 2^9
9
5
8
2
Wait 2^2
(i.e., leave at **t=19**)
6
1

t=18



Complexity

The smallest identity i is the quickest to go around the ring

Messages: n $O(n)$

Time: $2^i n + n$ units $O(2^i n)$

The second smallest id: $i + 1$

in time: $2^i n + n$ has traversed $(2^i n + n) / 2^{i+1}$ --- $(n / 2)$ links

The third smallest id: $i + 2$

in time: $2^i n + n$ has traversed $(2^i n + n) / 2^{i+2}$ --- $(n / 4)$ links

...

The j^th^ id

in time: $2^i n + n$ has traversed $2^i n + n) / 2^{i+j}$ --- $(n / 2^j)$ links

Total number of messages

$$\sum_{j=1}^{n-1} n/2^j = n \sum 1/2^j = O(n)$$

Total Time

$O(2^i n)$

BITS	TIME
$O(n \log Id)$	$O(2^i n)$

i : smallest id

Id : biggest id

Min-finding and Election

The cost of election can be reduced by using different techniques:
waiting and guessing

Waiting

Idea: The entities wait for some condition to happen before doing something

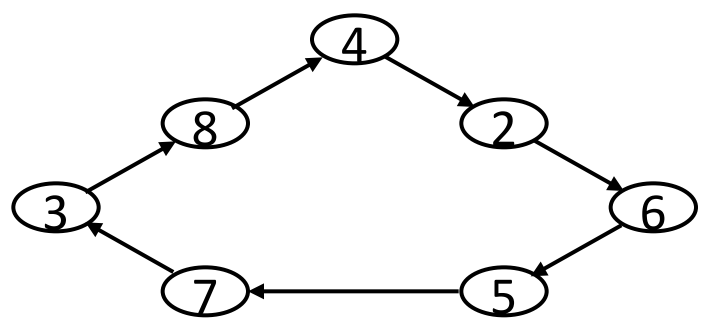
synchronous ring

- Knowledge of n
- Identities are integers
- In this example the ring is unidirectional but it could be done in the bidirectional

Waiting

With Simultaneous Initiation

1. Each awake entity waits a certain amount of time
2. If nothing happens it becomes the leader and
   notifies the others.



Entity with identity *i* must wait an appropriate amount of time *f*(*i,n*).

(let us assume that they start all simultaneously for the moment)

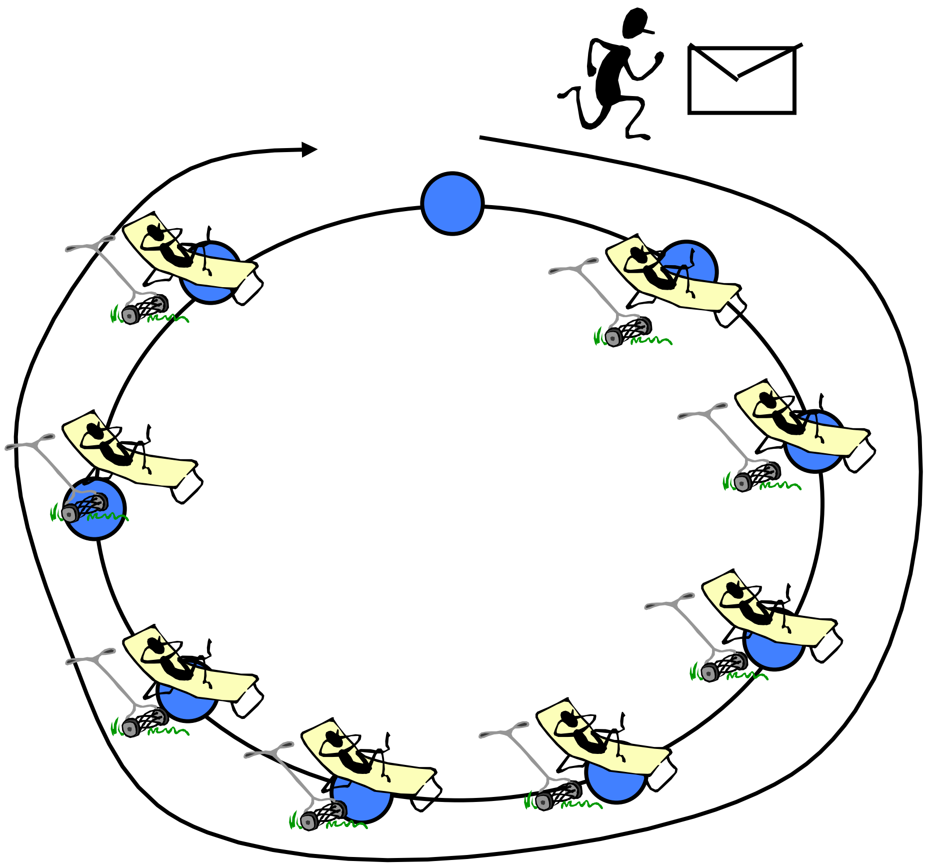
Let x be the minimum

Let d(x, y) be the distance between x and y (x < y).
Function *f*(.,.) must be such that $\forall y$:

$$f(x,n) + d(x, y) < f(y,n)$$

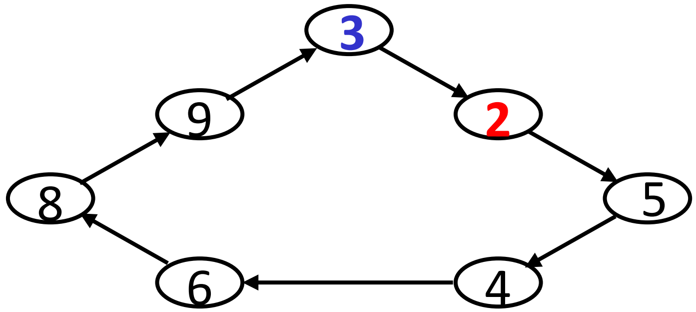
Function *f*(.,.) must be such that, if x is the minimum, ∀*y*:

$$f(x,n) + d(x, y) < f(y,n)$$



This must be true in the worst case, i.e., when

$y = x + 1$ and $d(x, x + 1) = n - 1$.



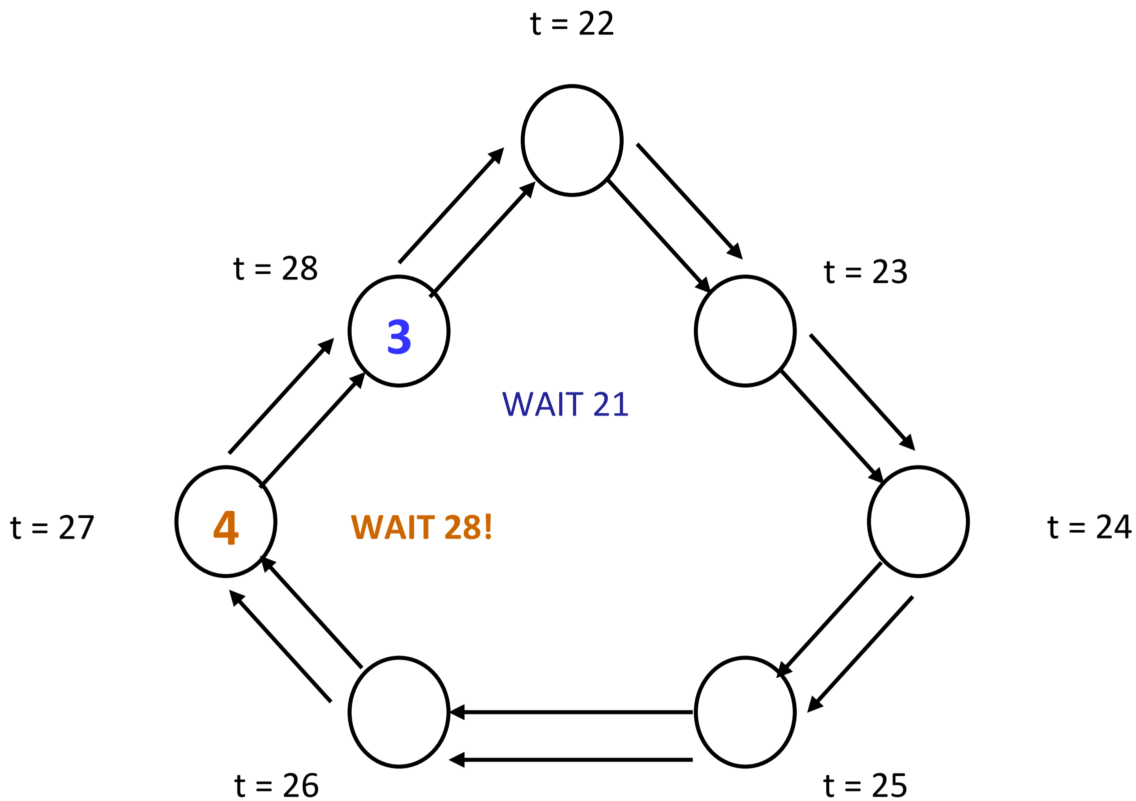
$$\begin{cases} f(0,1) = 0 \\ f(x+1,n) - f(x,n) > n-1 \end{cases}$$

A solution is **$f(x,n) = xn$**

$(x+1)n - xn = xn\text{-}n\text{-}xm = n > n-1$

Example:

$f(x,n) = xn$



n = 7
x = 3
nx = 21

x + 1 = 4
n(x+1) = 28

Complexity

Bits: Only the smallest entity send messages n bits

Time: $Id_{min} \bullet n + n$

BITS	TIME
$O(n)$	$O(Id_{min} \bullet n)$

Compare with "Speed"

BITS	**TIME**
$O(n \log I_{max})$	$O(2^{Idmin}\, n)$

Without Simultaneous Initiation

1) WAKE UP

when I spontaneously start, I wake up my neighbour before starting the waiting process.
An inactive entity: receiving the wake-up msg, forwards it and start the waiting process.

$t(x)$: time when x becomes awake

It is easy to see that

for all y
$$t(x) - t(y) < n$$

We have id(x) < id(y)

x must finish waiting before any **y** and its message should reach **y** while still waiting, so we want:

$$t(x) + f(x,n) + d(x, y) < t(y) + f(y, n)$$

Easy to see that: $f(x,n) = 2\,n\,x$ guarantees the inequality

Universal Waiting

1) Wake-up (Start messages) are sent around
2) As soon as an entity becomes ACTIVE, it starts waiting f(x) time units
3) If, while waiting, nothing happens, x decides it is the minimum and send a Stop message around
4) If an entity receives Stop while waiting, determines it is not the minimum and forwards the Stop messgae

$f(x) = 2 \times n$
would still be ok

Guessing

Used to compute a function of the input values without transmitting the actual values.

Search Process

1. Try to guess the result
2. Verify your guess
3. If it's correct, ok
4. Otherwise go-to 1

Example: Find the minimum value in a ring of known size.

- The Ids are not necessarily distinct
- *n* is known
- The entities start at the same time

If the entities predefine a sequence of guesses:

$g_1, g_2, \ldots g_k$

For each guess g_i, they collectively verify

Verification function

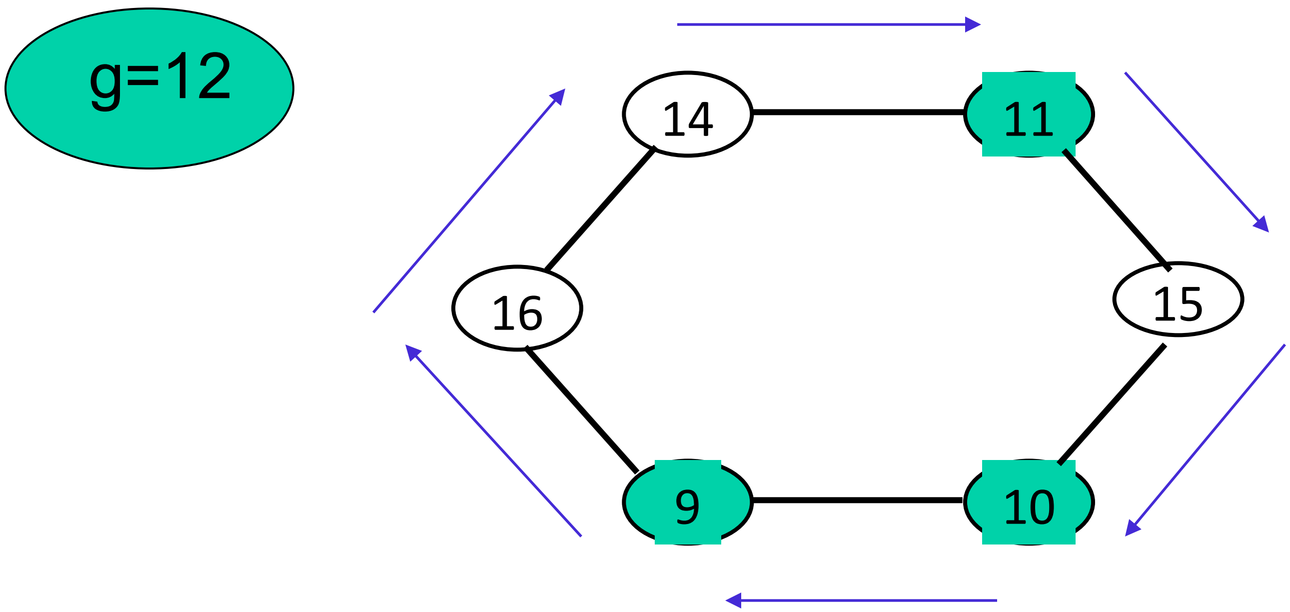
DECIDE(*g*)

Every entity compares its value ID with g

If ID ≤ *g*, send a message.

Otherwise (ID > *g*) only forward arriving messages

Example



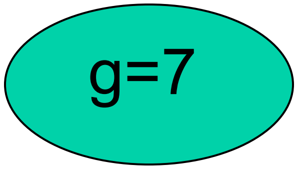
DECIDE(*g*)

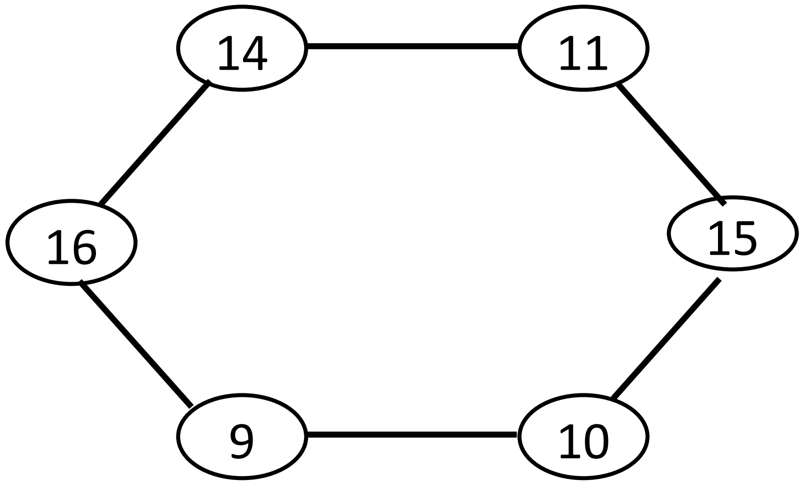
Every entity compares its value with ***g***

if value ≤ ***g*** send a message.

else forward any received message

Example





DECIDE(*g*)

Every entity compares its value with *g*

if value ≤ *g* send a message.

else forward any received message

all values > g ⟶ SILENCE

at least one value ≤ g: ⟶ MESSAGES

Everybody finds out within n time units

```
              DECIDE(g)
Every entity compares its value with g
   if value ≤ g send a message.
   else forward any received message
```

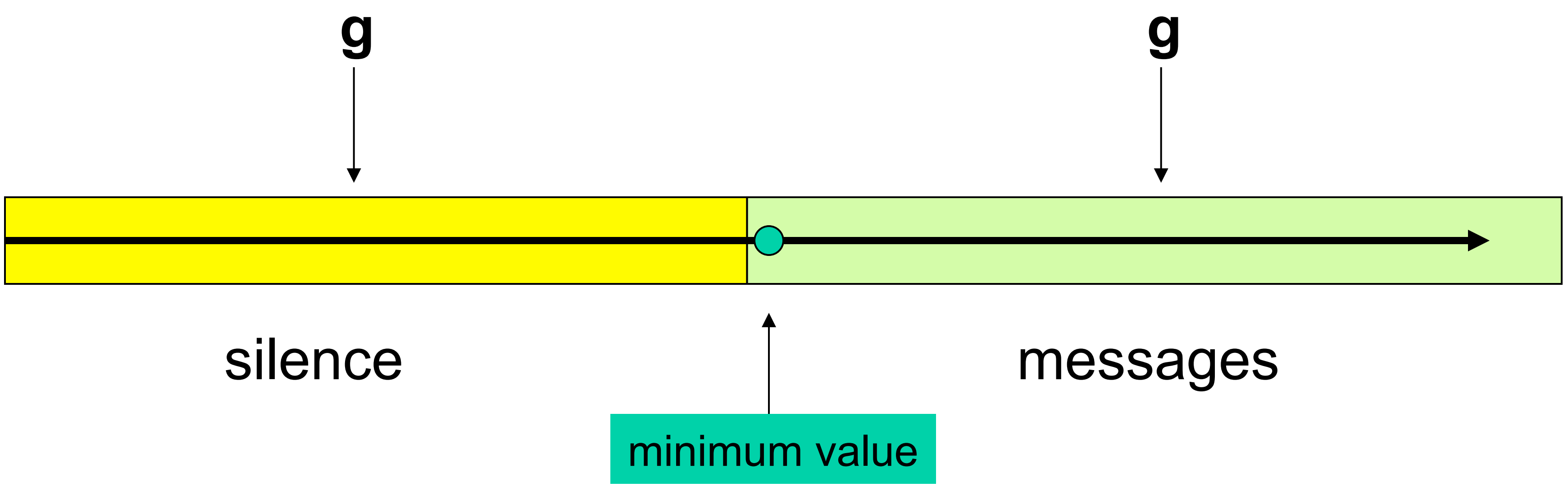
After n time units

i) Nothing happens

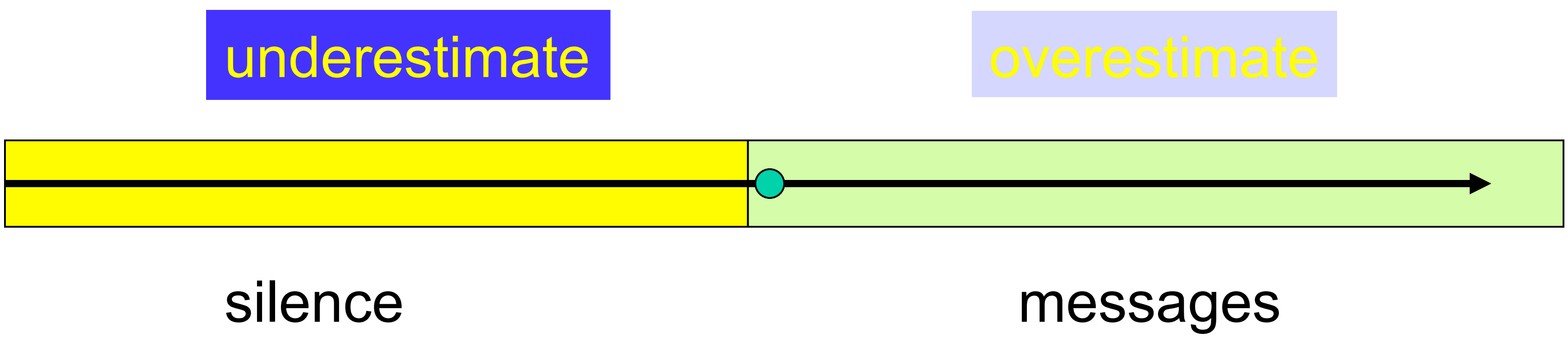
All the Ids are bigger than g

ii) A message is received

There is at least one Id smaller then or equal to *g*

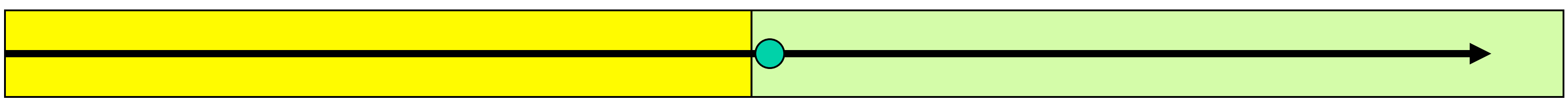

g
g
silence
messages
minimum value

GUESSING GAME: Our guess is **g**



GUESSING GAME: Our guess is **g**

underestimate overestimate



0 bits n bits

n time units n time units

START
Spontaneously
set alarm = c(x)+n
If id(x) ≤ g then
decision = high
send(high) to right
become(DECIDED)
Else
become(UNDECIDED)

We call this an overestimate (but could be correct)

I don't know

UNDECIDED
receiving(high)
decision=high
send(high)
become(DECIDED)
When c(x)=alarm
decision=low
become(DECIDED)

At the end everybody is DECIDED
The decision could be low or high

Sequence of guesses:

$g_1, g_2, \ldots g_k$

DECIDE(g_i)

choose(g_{i+1})

DECIDE(g_{i+1})

....

DECIDE(g)

TIME	BITS	
n	0 (under-estimate)	n (over-estimate)

GUESSING GAME: how about **g** ?

Every question costs | n time units

0 bits — underestimate

n bits — overestimate

GOAL: $\Theta(n)$ BITS → O(1) overestimates

We would like a strategy that

MINIMIZES the number of overestimates

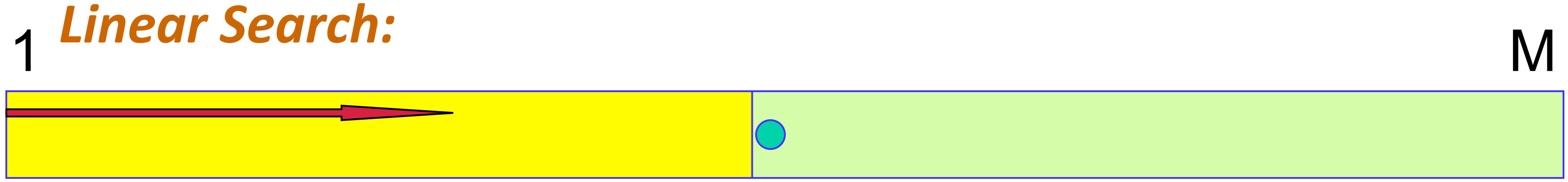
Question: What can I do with
ONE over-estimate only ????

What can I do with
TWO over-estimate only ????

ONE over-estimate allowed

Assumption: the number to guess is **between 1 and M**

Linear Search:



Try: 1,2,3,4

Until you get an overestimate

You found the value to guess

TIME

$O(n\ id_{min})$

$O(n)$

W.C. $O(M)$

BITS

ONE over-estimate allowed

Q # of guesses (worst case)
K # overestimates

Linear Search:

sequential search

K=1 → Q= M-1

TWO over-estimates allowed

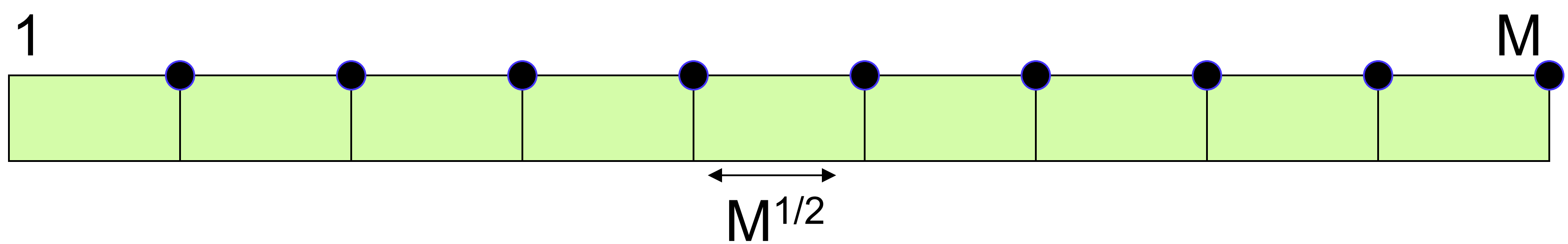
Q	# of guesses (worst case)
K	# overestimates

1 M

K=2

TWO over-estimates allowed

Q # of guesses (worst case)
K # overestimates

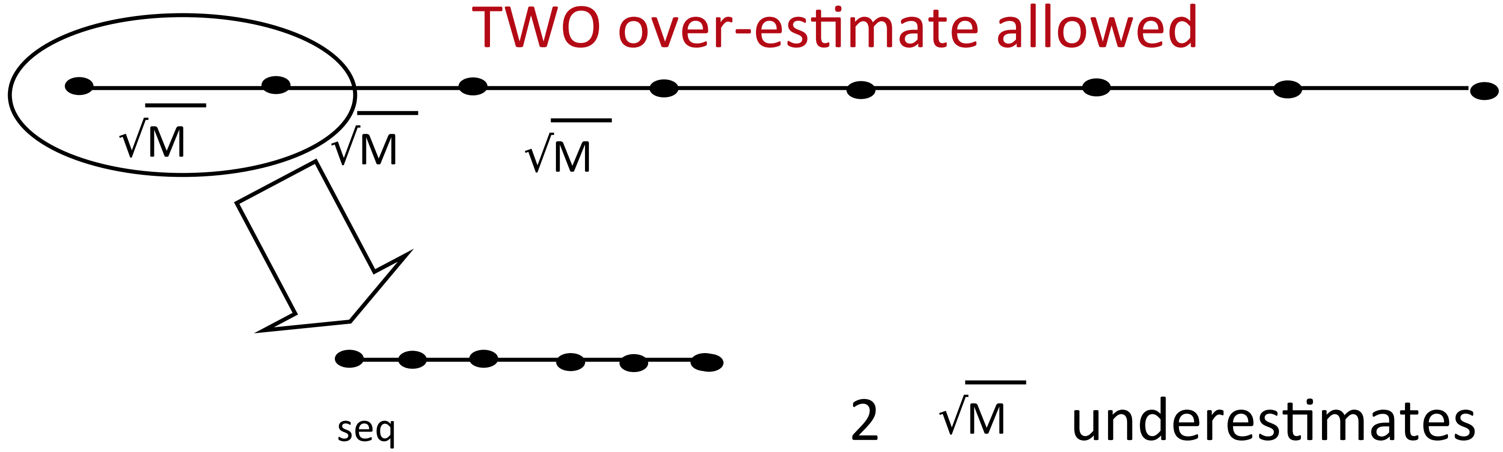


K=2 → $Q = 2\,M^{1/2} - 2$

sequential search on 

K=1 sequential search on 

TWO over-estimate allowed



2 $\sqrt{M}$ underestimates

2 overestimates

W.C.
In the
last interval

TIME
$O(n M^{1/2})$

BITS
O(2 n)

In general, complexity:

TIME
$O(n M id^{1/k})$

BITS
O(k n)

K constant

Is this the optimal strategy ?